INTRODUCTION TO JAVA PROGRAMMING FOR THE HIGH SCHOOL STUDENT

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE,
FACULTY OF SCIENCE
AT THE UNIVERSITY OF CAPE TOWN
IN PARTIAL FULFILMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF PHILOSOPHY IN INFORMATION TECHNOLOGY

By
Sinclair Tweedie
August 2004

Supervised by
Donald Cook

Table of Contents

Table of Figures

Tables

Acknowledgements

Thanks to Donald Cook, my supervisor, for guiding me through the process. Thanks for the invaluable assistance in directing reading, loaning books and bouncing off ideas.

Thanks to Rudy Neeser, my tutor, for technical advice on programming.

Thanks to Prof Tim Dunne, Prof Joan Juritz and Dr Rene Navarro of the Statistical Sciences Department at UCT for their help

Thanks to Tom West of Holt Software for invaluable advice and help on technical aspects of programming.

Thanks to my school for continued support in this endeavour.

Thanks to the students who took part in the study, used the program and provided many innovative and interesting examples of code.

Thanks to my wife and family for their support.

Abstract

The objective of this project was to evaluate the effectiveness of teaching high school students the Java language utilising assistant Java classes. These classes were designed to simplify the syntax of the language and to introduce the concept of inheritance. Two Java classes were created. The main class used an artefact called a Tortoise, based on the Logo idea of a Turtle, and provided a number of graphical methods for the user. The second class was called "Please" and simplified the Java syntax using a number of class methods which required a very straightforward English-like syntax. The Tortoise class was used to teach four classes of Grade Nine students in Reddam College over a period of two months. The Please class was used in teaching Grade Ten Computer Studies students. The Grade Nine students were then tested using two questionnaires. One of these was issued after two introductory lessons and the second was issued at the end of the teaching period. The results were then evaluated using McNemar's test of change. There was a positive response to the use of the Tortoise class in the short term by the students.

Introduction

We ought to continually question what we teach and how we teach it. It is a healthy and positive activity which tends to produce a vibrant and constructive learning environment. The goal of this project is to produce Java classes to assist the novice programmer, particularly at the High School level and to assist teachers of these students. The "Tortoise" class was created with a view to producing a graphical output using a tortoise artifact, similar to Seymour Papert's turtle which he used in the LOGO language (Papert, 1980). The "Please" class was created in an attempt to simplify input. This is a recognized problem for novice Java programmers (Grissom, 2000, Proulx and Rasala, 2004, Roberts, 2001, Roberts, 2004, Wolz and Koffman, 1999).

Learning the Java language is not straightforward for many students and teachers. The precise reasons are unclear. Novices have to deal immediately with a lot of new concepts and Clarke et al argue that when students have to use code that they do not understand, it leads to "puzzlement, discouragement, and often apathy" (Clark et al, pp 174, 1998). These researchers raised the question of whether the syntax of Java was inherently difficult, or whether it might be that a change from the procedural paradigm to the object-oriented paradigm caused the difficulty. They also highlighted student problems arising from input and exception handling. It was suggested that students did not understand what they were doing initially and had to accept certain things which might become clear later. This left students feeling a lack of confidence (Clark et al, 1998). Teachers may also play a role in adding to students' confusion. Eric Roberts stated that "The computer science education community must find a way to separate the essential characteristics of object-oriented programming from those accidental features that are merely artifacts of the particular ways in which technology is implemented today" (Roberts, pp 116, 2004). He drew a distinction between essential and accidental complexity. Examples of essential complexity were object-oriented concepts such as encapsulation, inheritance, re-usability and interfaces. An example of accidental complexity was the "fact that there are on the order of 50,000 public methods in the Java 2 class libraries" (Roberts, pp 116, 2004). Roberts further went on to

discuss the move in 1995 by the College Board[1] of New York who administered the Advanced Placement program for Computer Science (APCS). They decided to replace Pascal as the language of instruction with C++ from 1999. In 2001, two years after implementation, they announced that this would change to Java in 2004. Many schools dropped the APCS program as a result. "Even in programs that continue, teachers feel a high level of anxiety about the new material" (Roberts, pp 116, 2004). In discussion with the subject advisors for Computer Studies in the Western Cape Department of Education it is clear that this is reflected by many teachers in the Western Cape who come from a background of programming exclusively in Pascal for many years and for whom the move to Java has been difficult.

The graphical approach to teaching programming does have an effect on learning (Becker, 2001, Calloni and Bagert, 1994, Gilnert and Tanimoto, 1984, Green, 1997, Jones et al, 2002). Thomas Green posed the question "Why do people like graphical widgets?" (Green, 1997, pp 25). In a survey of programmers of various levels the majority expressed a preference for graphical representation. Some of the reasons given were that it was less formal, more memorable, more comprehensible, made structure more visible and provided more information with less clutter (Green, 1997). Seymour Papert suggested that a child's use of a programming language (LOGO) could affect learning in other areas (Papert, 1998, Papert, 1999). In developing LOGO he was particularly interested in assisting children to learn mathematical concepts (Battista and Clements, 1988, Clements and Meredith, 1992). Some studies have focused on the effects that programming in Logo has on problem solving and creativity across other learning areas in Kindergarten and Second Grade students (Degelman et al, 1986). These researchers reported a significant reduction in problem solving errors and observed a statistically significant difference in creativity. Creativity was measured via the Torrance Test of Creative Thinking (Torrance, 1972, Kim, 2002). However, there is neither general agreement on the effects of using computers in schools nor even that the effects are beneficial (Clements and Meredith, 1992). Jane Healy suggested that the many negative aspects of exposure to computers at an early age outweigh any benefits (Healy, 1998). Research results on Mathematical achievement are

[1] College Board – Non Profit educational organization in New York city

inconclusive overall. Clements and Meredith suggest that Logo affects a limited number of mathematical topics and tests only assess limited areas of mathematical knowledge (Clements and Meredith, 1992). These researchers point out that students learning Logo have to explore relationships and consider their own actions and in so doing are building up experience and establishing concepts which will serve as a "framework" (Clements and Meredith, pp 2, 1992) for formal learning later on.

The importance of practical experience has been emphasized by a number of researchers. "*Experimental* pedagogy is a *necessary* compliment of cognitive developmental psychology" (Smock, pp 53, 1981). This hypothesis draws heavily on the Constructivist theories of learning proposed by Jean Piaget (1896-1980), which argue that children progress through stages of learning in the acquisition of concepts (Jones and Brader-Araje, 2002, Kafai, 1996, Papert, 1996, Sigel et al, 1981, Smock 1981). Pierre van Hiele proposed that students' concepts develop through a series of levels. He argued that if children are not ready and if teachers use concepts and language from higher levels then they will not attain higher levels (van Hiele, 1986). Clements and Meredith suggest that the use of Logo helps students to progress to those higher levels (Clements and Meredith, 1992). Throughout their commentary these researchers point out that results of testing show no difference between control groups and those using Logo. They do emphasize that much of the usefulness appears to lie in practical aspects of pedagogical habits such as learning from mistakes and they are "guardedly optimistic" that Logo does help in developing problem solving ability(Clement and Meredith, pp 7, 1992). Much of the research has been directed toward Pre-school and Primary School children. There is also a significant body of research on the effectiveness of graphical programming with novice programmers at the tertiary education level (Buck and Stucki, 2001, Calloni and Bagert, 1994, Corbett and Myers, 1996, Gilnert and Tanimoto, 1984, Green, 2000, Modugno et al, 1996).

Errors have been always been a source of frustration for novice programmers and as such are something that can be targeted when creating an environment for novices. When Seymour Papert designed LOGO he incorporated a shift in the paradigm for dealing with errors. Previously, errors were always indicated using a message which identified it as

being from a set of standard errors occurring at a given line in a program. The responsibility for the fault was then handed over to the programmer. Logo reports errors with a message such as "Logo doesn't know how to .....". This seeks to place the burden of error on the computer rather than the user. Andrew Ko, as part of "Project Marmalade" at Carnegie Mellon University, identified six types of learning barrier to successful programming (Ko et al, 2004). As part of this study the researchers developed an interesting process for dealing with programming errors. "A significant proportion of the errors are ultimately caused by a vicious cycle of assumptions" (Ko, 2004, http://www-2.cs.cmu.edu/~NatProg/marm-whyline.html). When a program failed, a programmer could ask direct questions about the error and receive an answer presented as a visualisation of the error, accompanied by an explanation. The researchers' aim was to stimulate programmers to actively examine their error and relate that error to its cause. The objective was to encourage the programmer to take an active role in understanding program failure instead of passively receiving an error notification and then trying random solutions which might well end up confusing the programmer even further (Ko, 2004).

The Focus of this Study

The objective of this project was to produce pre-defined classes which would be used to introduce and facilitate the underlying concepts of the Java language. This approach has been used successfully at undergraduate level using the Karel Robot artifact (Becker, 2001, Buck and Stucki, 2001). The approach does require an understanding of the object-oriented paradigm on the part of the teacher. It is also preferable to introduce important object-oriented concepts such as instantiation of objects, invocation of methods and extension of classes from the beginning of teaching (Clark et al, 1998, Duke et al, 2000, Becker, 2001, Nevison and Wells, 2004). Becker suggests that the procedural paradigm approach results in a need for a paradigm shift after the introduction of the traditional concepts of selection, iteration and procedural decomposition. Becker further suggests that the traditional procedural approach results in missed opportunities. "Why should students be summing numbers when they could be doing more interesting things which are pedagogically just as valid?" (Becker, pp 53, 2001). Becker emphasizes that the use of the Karel Robot artifact is not perfect but "The result has been a course that is fun for both

students and instructors, and where students understand the fundamental concepts early, allowing them to approach advanced topics with confidence" (Becker, pp 54, 2001). Other researchers suggest that, having accepted the object-oriented paradigm, it is important to introduce fundamentals in an enjoyable and exciting way (Duke et al, 2000). Duke et al stress the importance of providing a novice with the opportunity to experiment in an interactive programming environment. Their course is designed such that lectures, tutorials, programming and assessment are based around a set of "over 160 carefully crafted programming problems" (Duke et al, pp 80, 2000) Other researchers have looked at student difficulties and concluded that students preferred practical programming through assignments and tutorial assistance. This was more effective than lectures or laboratory sessions (Madden and Chambers, 2002)

Many teachers are reluctant to drop the procedural paradigm. This problem is well documented (Ross and Zhang, 1997, Van Roy et al, 2003, Halland and Malan, 2003). The aim of this project was to reduce the difficulties faced by students dealing with syntax and programming concepts in the early stages of learning to program and also to stimulate teachers to come to grips with Java programming concepts. The classes used draw on familiar ideas but with some innovation and effort users can produce interesting, valuable and enjoyable projects which extend and improve their facility in Java programming. The possibilities of these strategies have been looked at by numerous researchers (Becker, 2004, Buck and Stucki 2001, Calloni and Bagert, 1994, Gilnert and Tanimoto, 1984, Lister, 2004, Nevison and Wells, 2004, Zhu and Zhou, 2003).

Strategy 1 uses the idea of a Logo-like turtle (implemented as the Tortoise class. The primary objective was to introduce the concept of inheritance and explaining it to students. This inductive approach has been suggested by other researchers (Kluit et al, 1998, Zhu and Zhou, 2003). A teacher from the Pascal tradition might also benefit from having to evolve explanations to show children how to use the Tortoise class.

There are numerous implementations of the Logo turtle available now in Java (Braught, 2003, Devarakonda, 2003, Eckert, 2001, Gibbons, 2001, Grieser, 2002, Khalil, 2002, Kono, 2000, Lambert and Osborne, 2003, Oelschlegel, 2003, Proulx and Rasala, 2002, Ventura,

2000, Weissinger et al, 2001, Wolz, 2002). Turtle Graphics goes back to the seventies when Seymour Papert of MIT's Artificial Intelligence Laboratory invented it as an aid for beginner programmers in Logo (Papert, 1980). His objectives were somewhat different then but the style of turtle graphics is very useful in dealing with young programmers. The focus in using an artifact moves the user's attention to problem solving using common sense and personal experience in a structured way. Papert described it as a "constructed computational object-to-think-with" (Papert, pp 11, 1980).

In considering the design of the Tortoise class and its methods several goals were kept in mind:

1. The need for assimilation of syntax was kept to a minimum. The essence of the original Java syntax was retained but selected functions were performed automatically with no input from the programmer.

   For example, a student has to use standard Java syntax to instantiate a tortoise but the tortoise constructor creates frames, adds components and draws backgrounds with no intervention from the user.

2. Students were to be sensitized to the idea that increased complexity provided increased functionality. A number of constructors were provided which allowed the student to combine single operations

   For example, to begin with the user might instantiate a turtle and invoke separate methods to turn it, hide the trail and move it a given number of positions to start a pattern drawing from a different place. The alternative constructor with parameters allows this to be accomplished in one line.

3. A repetitive pattern first requires the student to identify pattern. Methods which incorporated the use of loops were deliberately not provided.

For example, activities which were deliberately repetitive and quite arduous were chosen so that students might ask if there was not an easier way to perform the task. It was possible that syntax such as that for loops would cause no concern for novices its. Convenient methods could have been provided to do such things as drawing specific shapes but this was deliberately omitted so that students might construct their own.

4. Willingness to experiment may be related to goal directedness (Papert, 1980). Information on available methods was initially withheld. The "hidden" functionality of Java should be associated with curiosity, usefulness and excitement rather than frustration, stonewalling and irritation. By the time the students need more adaptability they should be prepared to try a little harder to make a program work.

   For example, the grid size could be altered (meaning the size of the blocks making up the grid). The default grid blocks were deliberately made to be slightly too small. The number of rows and columns in the grid could also be increased. The default constructor deliberately provided quite a small grid. A larger grid provided the opportunity for bigger, more complex patterns to be drawn

5. Students were introduced to the Java technique of assigning variables a value from a method's returned value. This was done in as natural a way as possible. This portion of the program's functionality has great potential for directing students in their move away from "easy" syntax to raw Java. This section is more suitable for more experienced students and was not tested in the study. An "age" attribute was introduced to start this process.

It became clear as development of the program progressed that a whole teaching methodology could be developed to guide students and teachers through a productive and interesting introduction to Java programming. This has been explored by other researchers (Duke et al, 2000). Papert suggested that there was a measure of dishonesty in teaching that was masquerading as a game. Teaching is always concerned with finding ways of developing basic concepts and preparing the ground for easy assimilation of difficult

concepts at a later stage. If games are productive and useful in this process then the argument surrounding games and serious teaching, which Papert emphasized, is irrelevant (Papert 1998).

Strategy 2 uses a simplified standard text class to reduce the difficulty of input and output. One of the strengths of Pascal in the early stages of learning to program was its capacity to lead the student to write simple programs where a user could interact with the program. It is beneficial to a novice programmer to get quick positive feedback (Barnes, 2002, Becker, 2001, Bruce, 2001, Gilnert and Tanimoto, 1984, Papert, 1998, Sangwan et al, 1998). In considering what presents the problems for new programmers, especially those students who do not have any sort of natural affinity for the language, there is an attraction to the idea that novices in the High School could start with a set of standard methods which had high usability but which still retained the essence of the Java methodology. From my experience in teaching I have seen that the main difficulty for beginners in the earliest stages lies in what they view as obscure syntax. Many novices are initially obstructed by an overload of information (Proulx, 2000). While they are struggling with the abstract concepts of the language they become discouraged by the necessity to learn a language syntax which uses unfamiliar words in a complex environment. A number of researchers have considered the causes of novice programmers' failure (Jenkins, 2002, Jones, Boyle and Rickard, 2002, Matthíasdóttir, 2004). Bruce Eckel in the introduction to Entsminger's book, "The Tao of Objects", suggests that novice programmers experience difficulty in establishing a perception of what it is they are learning (Entsminger, 1995). Although many authors promote the idea of the relationship of objects to everyday life the "naturalness of object-oriented programming" (Davies, pp 61, 2000) is not accepted without argument (Davies, 2000, Ross and Zhang, 1997). Teachers who are familiar with Pascal tend to be constrained by the procedural paradigm and find the Java syntax awkward (Halland and Malan, 2003, Ross and Zhang, 1997, Wolz and Koffman, 1999). This tends to stifle their ability to simplify the language syntax for their students. It would seem reasonable to assume that any class that promotes the easy and intuitive use of object-oriented syntax must be of benefit to these teachers.

One does not want to remove the syntax entirely because the introductory class should be designed to become redundant and not remain as a permanent crutch. Each utility should be explained at a later stage of teaching and used to facilitate the crossover to the original syntax as soon as is feasible. There is also a danger of moving away from the flexibility of Java into an entirely proprietary class that does the same thing as existing Java classes, but not as well and with inevitable bugs. The "Please" class was designed to be a temporary buffer for novices and the redundancy of the class should be made clear to those novices as they gain confidence in using the standard Java classes.

By the very nature of the learning process, especially with regard to the successful assimilation of concepts, observations and testing would have to be done over a period involving much of a school year. The most obvious test would be to look at results of students sitting the same exam at the end of the year and compare results of the trial group with those of the traditionally taught group. This was not possible within the timeframe of this project. There was also the possibility that using an unsuccessful approach (with a control group) would lead to criticism from parents that their children were not being taught properly. The tests documented in this report thus provide only preliminary data.

"Before" and "after" tests were administered to establish if any change in understanding occurred. Almost all of the students involved in this study had no exposure to programming of any sort. Teaching was deliberately controlled and kept to a minimum. The Ready IDE from Holt Software was used throughout the study by the students when creating programs[2]. Certain information was not volunteered in order to see if the students picked up conceptual facts by applying the methods in a purely practical way. Students in this study have access to computers at home, are both male and female and have a wide range of ability in the school subjects that they study.

Two questionnaires were administered, one at the start of the course of study and the other at the conclusion of the course of study. "In experiments designed to study rates of learning as a function of treatment effects, repeated measures on the same subject are a

[2] Please see Appendix G for full details.

necessary part of the design" (Winer, pp 300, 1962). After discussion with the University of Cape Town Statistics Department it was decided that contingency tables should be used to summarize the results of the questionnaires. McNemar's test for correlated proportions was used to calculate significance on the pair of dependent observations. This test provides data on whether or not a change in performance occurred as a result of the teaching. The amount of the change has to be judged by looking at data as a whole. "Assessment of the magnitude of the difference must allude to the remainder of the data" (Armitage and Berry, pp 126, 1994). In McNemar's test a student acts as their own control (Lee, 2004, Siegel and Castellan, 1988). The null hypothesis (H_0) was that no change in performance occurred. This was tested with reference to the Chi Squared distribution at the 95% level with one degree of freedom (Conover, 1971, Siegel and Castellan, 1988). For very small frequencies , that is less than 5, the binomial test can be used (Siegel and Castellan, pp 79, 1988).

Thesis Statement

Observations

Programming behaviour and habits are heavily affected by the first contact that a student has with any programming language, especially if students have little or no prior involvement with problem solving in a computer environment (Ross and Zhang, 1997, Van Roy et al, 2003, Halland and Malan, 2003).

Some students are confused by programming syntax, to the extent that they lose attention and focus. The foundational phase of learning to program is when the student should gradually become familiar with the object oriented paradigm. For some students the important abstract concepts become obfuscated by superficial syntactical problems. They become immersed in the terminology and fail to move on to engage with the more important aspects of learning a computer language. This prevents some students from becoming competent programmers and considerably slows down the progress of others (Halland and Malan, 2003, Ross and Zhang, 1997, Wolz and Koffman, 1999).

Students become frustrated by activities which they feel ought to be simple but which, in their view, require an inordinate amount of effort to produce a program (Barnes, 2002, Becker, 2001, Bruce, 2001, Gilnert and Tanimoto, 1984, Papert, 1998, Sangwan et al, 1998).

My previous experience of teaching students to program has shown that the following types of behaviour may occur, resulting in students reaching a point where they no longer show improvement:

(i) Some students may be using the Tortoise class as they would a game and may remain at the level that they regard as fun and view the complex syntax as undesirable.

(ii) Some students may use the Tortoise class with the rules they have been given and will not seek to extend capability or refine their use of language constructs. (For example, the introduction of a loop in the early stages). This may mean that the

activities are of little use to some students. They may not be capable of overcoming the difficulties inherent in programming in Java.

(iii) The assisting classes may well help in the early stages but it is important that any class which has hidden, underlying Java standard classes are dropped at a suitable stage. Some of these classes may be counter-productive in that some students may continue to include them in programs long after their usefulness has ended.

(iv) Students are demotivated by the number and frequency of errors and by the technical jargon that they observe when an error occurs. Many High School students who are failing in school tend to have a negative self-image (Alpay, 2001, Huitt, 1998, Sánchez and Roda, 2002). Students placed in a stressful situation where they are not succeeding often resort to non-productive and negative behaviour. This situation may be improved or the erosion of confidence may be delayed by providing less threatening error messages.

Objectives

The overall objective of the assistant classes is to gradually introduce the Java syntax. This is done by using two different classes.

The Tortoise class provides an enjoyable introductory experience and uses Java syntax in a concealed fashion. The student becomes familiar with syntax in an informal setting. Concentration is directed towards producing a graphical output using skills that they already possess and this relies heavily on their common sense to use the restricted set of methods innovatively. The Tortoise class may be useful in introducing concepts such as inheritance and threads.

The Please class has been set up to make input seem easy by using static methods that are named to resemble English phrases. As students gain confidence they should be encouraged to drop the use of the proprietary classes and explore the standard Java syntax. By the stage when they are more familiar with programming in general they may appreciate and accept the adaptability involved in the "raw" Java syntax.

Hypotheses

The hypotheses to be tested in this study are:

1. By simplifying the interaction with the complexities of the Java programming language in the early stages of the learning experience we can facilitate concept learning in novice programmers.
2. By providing simple problems using an additional artifact (the tortoise) we can improve students' problem solving skills when using a programming language.

Chapter 1 - The Tortoise program

1.1 The Tortoise Class

The overall aim of the Tortoise class was to make the program simple to use for the Grade 9 students[3].

The Tortoise class in this study offers the following functionality:

1. A tortoise artifact is represented by an image in JPEG format (see Section 1.2).
2. The default constructor produces a tortoise at the top left corner of a rectangular grid made up of twelve horizontal and twelve vertical lines (see Section 1.4.1).
3. A slider bar is displayed above the grid and can be used to increase or decrease the speed at which the tortoise moves.
4. An object can also be instantiated using a choice of three parameterized constructors which change the number of rows and columns in the grid, the size of the grid as it appears on the screen or both.
5. Several tortoise objects can be instantiated to appear on the same grid. This is implemented in the program using the Grid class (see Section 1.4.2).
6. The tortoise can turn in eight directions (Compass points N, E, W, S, NE, SE, SW and NW).
7. A tortoise can leave a trail of its path as it moves or move to a new position with no trail left behind. The grid is made up of distinct blocks which are either in the background colour or the colour of the trail.
8. A tortoise can jump to a given position on the grid using the parameterized "setPosition()" method.
9. A tortoise has an attached screen label which the student can display using the "says()" method (see Section 1.4.5).
10. A tortoise has an "age" attribute which the student can access using the "ages" method or change using the "setAge()" method (see Section 1.4.5).

[3] For full program listings please see Appendix A. For Javadoc HTML documentation see Appendix B.

11. When a tortoise reaches the edge of the grid in any direction the student is informed of this using a pop-up message (see Section 1.4.3).
12. Several objects can move at the same time on the same grid but this requires explicit implementation of threads in the program created by the user (see Section 1.4.2).

1.2 Program Design

The program, which eventually developed in three phases, was peculiar in that it was a utility that was used as a resource by students to construct their own programs. The Tortoise class was made up of a number of constructors and mainly void class methods which provided the functionality of the Tortoise. The general look of the screen and what the user could access as a result of his/her programming was considered initially. Planning involved deciding on what was going to be made available to the user. Before settling on an approach a number of options with regard to the Tortoise itself were looked at. A number of existing Turtle code sources were looked at (Braught, 2003, Devarakonda, 2003, Eckert, 2001, Gibbons, 2001, Grieser, 2002, Khalil, 2002, Kono, 2000, Lambert and Osborne, 2003, Oelschlegel, 2003, Proulx and Rasala, 2002, Ventura, 2000, Weissinger et al, 2001, Wolz, 2002). Exploratory code was written to look at drawing the Tortoise as a polygon and then rotating it to move realistically in different directions (sample code trial runs shown below).

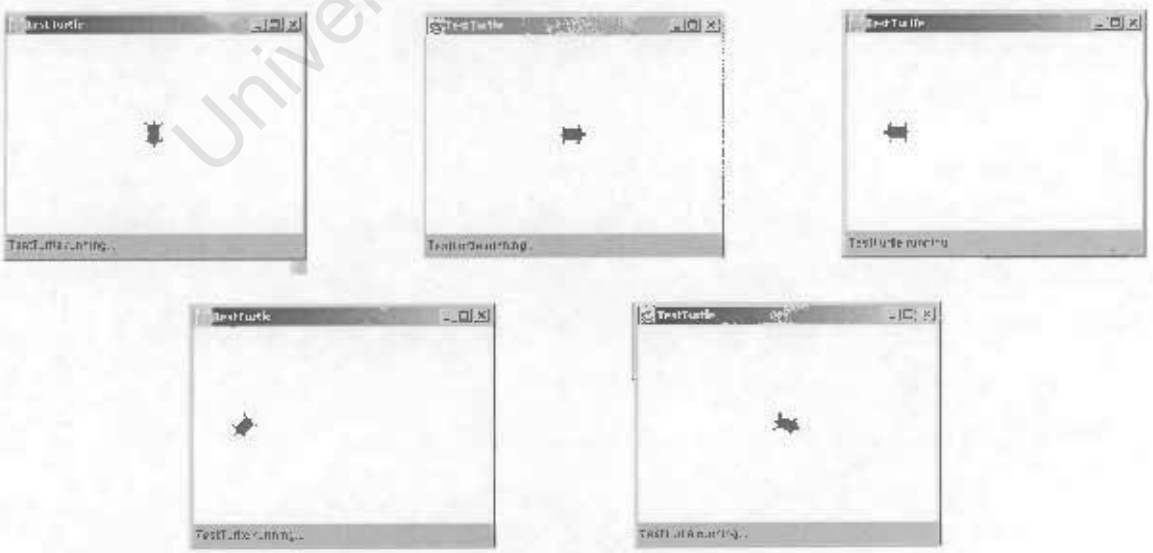


Figure 1 Sample polygonal tortoises

Most turtle programs work with a grid drawn on the screen. Some applications require the user to design and implement this background as a separate entity (Eckert, 2001, Khalil, 2002, Rinkens and Windhorst, 2003). The Tortoise grid in this project appears automatically when an object is instantiated. Students focus only on the Tortoise and its movement. In many programs the turtle trail is drawn on the lines of the grid and a turtle can move in any direction, defined usually as a bearing from North (Eckert, 2001, Gibbons, 2001, Grieser, 2002, Khalil, 2002, Rinkens and Windhorst, 2003). Papert's original focus in utilising the turtle was to improve geometry skills. Papert described the environment as "a microworld, a 'place' where certain kinds of mathematical thinking could hatch and grow with particular ease" (Papert, pp125, 1980). Logo syntax was deliberately designed to be as simple as possible and the Logo turtle moved to produce in lines with given length and direction. The Tortoise in this study was designed to move into squares one at a time using a background of gridlines to emphasize that movement. An attempt was made to change the focus from drawing whole lines to making individual movements associated to programming statements. The language syntax of the Java "object to think with" (Papert, pp 11, 1980) is less user friendly than the Logo equivalent. The Tortoise is used to motivate students to use that syntax. Tortoise movement is restricted to up, down, left, right and diagonal directions. The use of North, East, West, South, Northeast, Northwest, Southeast and Southwest is a straightforward approach for this age group. Feedback from students may suggest a change to complete control of direction and a decision to define direction as a bearing would also mean a completely different approach to the programming. The Tortoise trail is implemented as a series of grid blocks filled with colour. The display has been simplified to focus attention on the individual movements which occur as a result of invoking a method in the program. An image is used for the tortoise (or whatever creature one wanted) instead of filling a polygon for the reasons shown below.

The polygonal shape is shown as an aerial view:

Figure 2 Tortoise polygon

whereas clipart tends to be a side view :

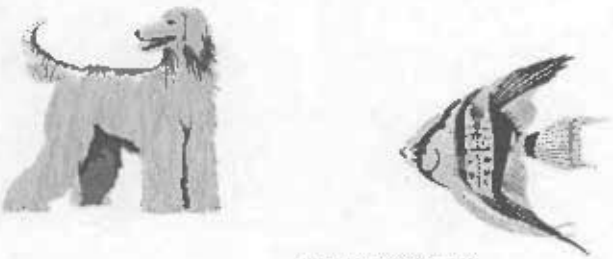


Figure 3 Clipart

Tortoise images, one for each direction:

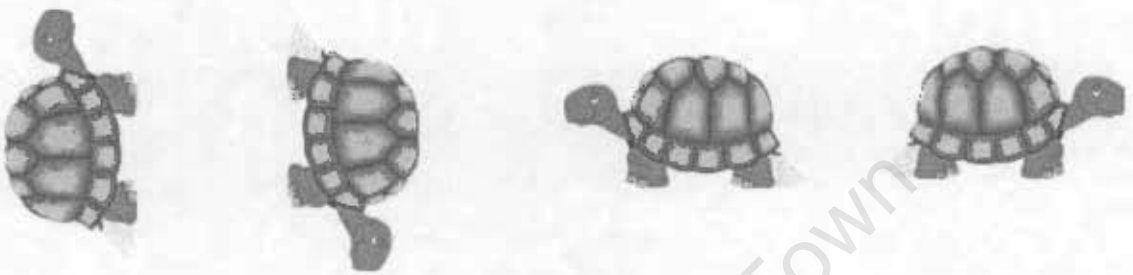


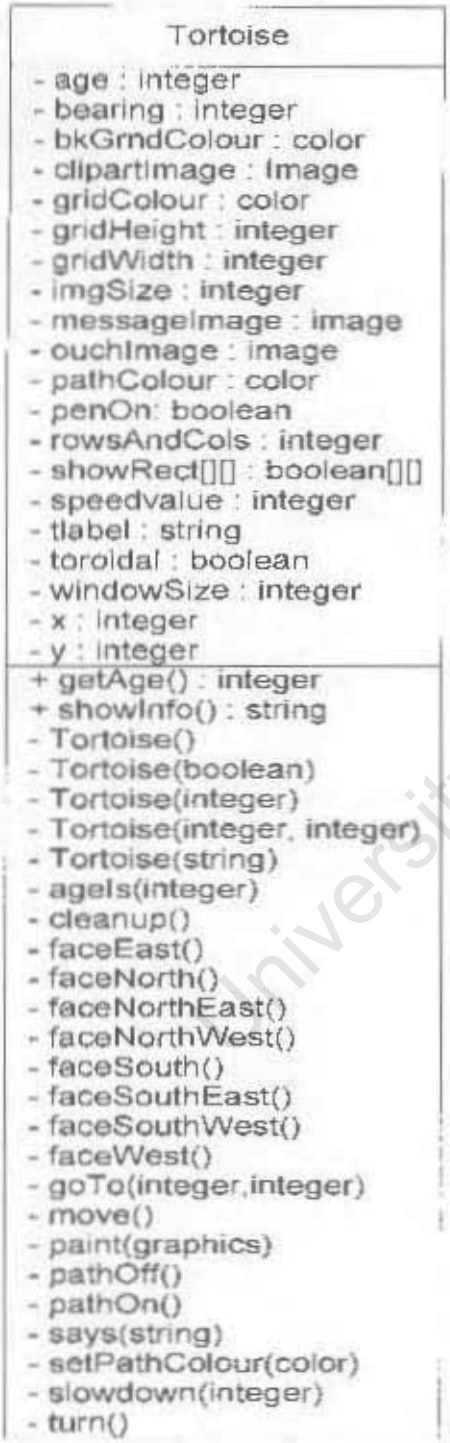
Figure 4 Tortoise directions

This approach also allows simple scaling of the image to match the screen display when grid sizes and block sizes are changed (See section 1.4.1 pp.19).

The original design of the Tortoise involved one class which provided a number of methods that the user could invoke to perform various actions on the screen. The first instantiation of the Tortoise automatically provided the frame for the user. It was decided at an early stage that the user would not be involved with anything other than the creation of a Tortoise. The user would not have to specifically provide code to create a background, unlike other programs (Jones, 2003). As the program developed it became clear that instantiation of more than one tortoise would be a distinct advantage. All new Tortoise instances would share the same screen frame. The Grid class was then developed and some of the attributes and methods in the original Tortoise class were transferred into the Grid class.

1.3 Class Diagrams

First Tortoise version



Tortoise second version, incorporating the Grid class

The original single Tortoise class was replaced by the Tortoise and Grid classes. Functionality was enhanced to allow for several objects to share the same grid.

1.4 Methods

1.4.1 The Paint() Method

The paint() method (in the Grid class) constructed the screen as a series of squares. A 2D array of Booleans was used to store the status of each square and alter the colour if the tortoise's trail was required. The following are some examples using the various constructors and showing simple trails.

Example using the default constructor

```
public class Tortoisetry
{
   public static void main (String [] args)
   {
      Tortoise fred = new Tortoise ();          ◄ First constructor, no parameters
      fred.pathON ();
      fred.setPosition (6, 8);                  ◄ The default position is (0,0)
      fred.faceNorth();
      for (int i = 0 ; i < 5 ; i++)
      {
         fred.move ();
      }
   } // main method
} // Tortoisetry class
```

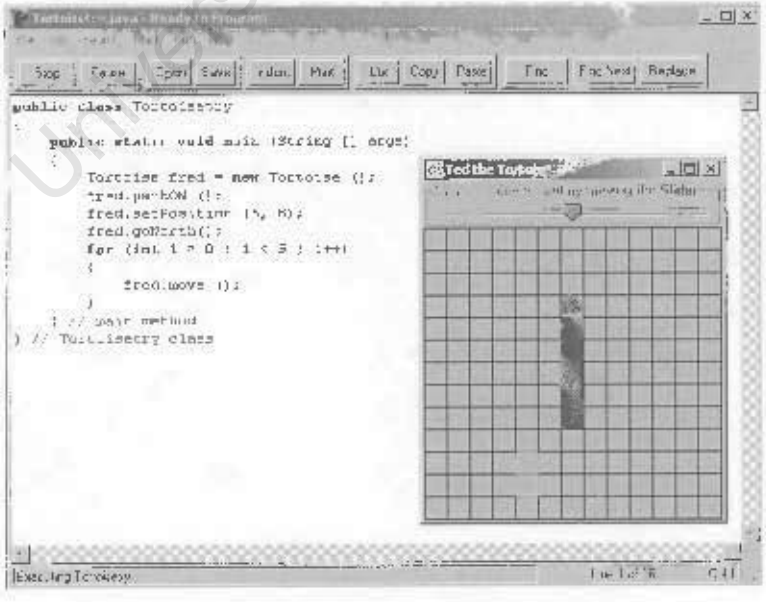


Figure 5 Basic example of Tortoise

Example using a constructor with size parameter

This constructor provides a bigger grid but with the same number of rows and columns as the default constructor.

```
public class Tortoisetry
{
  public static void main (String [] args)
  {
    Tortoise Ted – new Tortoise (45);          ◄Constructor with scale parameter
    Ted.pathON ();
    fred.setPosition (6, 8);
    fred.faceNorth();
    for (int i = 0 ; i < 5 ; i++)
    {
       fred.move ();
    }
  } // main method
} // Tortoisetry class
```

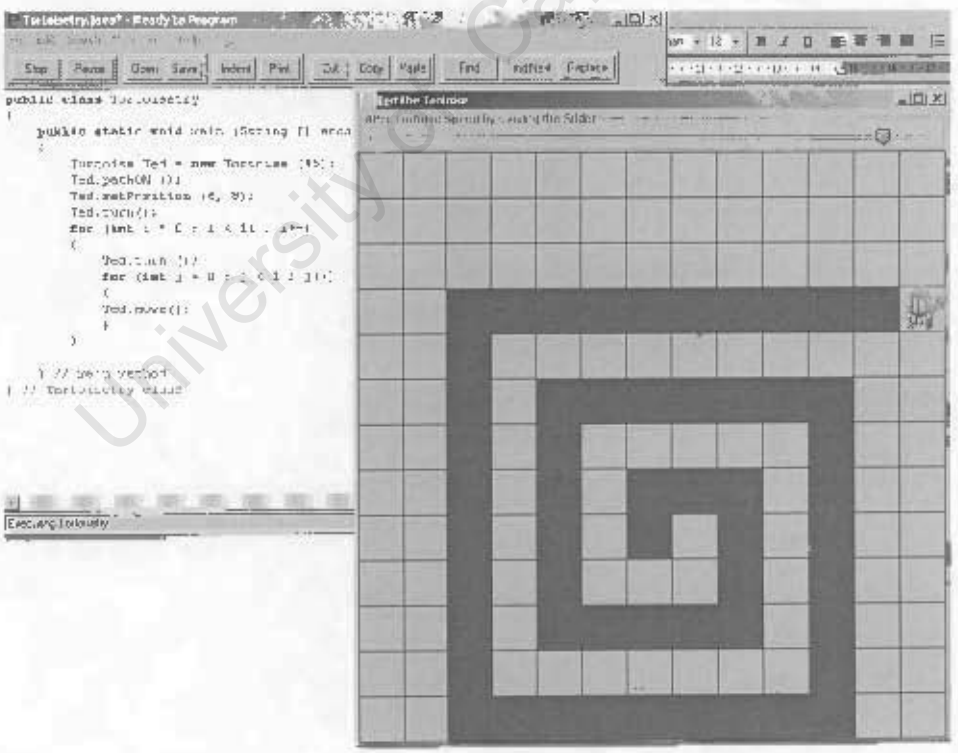


Figure 6 Scaled up tortoise display

Example using a constructor with scale and grid size parameters

This allowed a bigger grid with a stipulated number of rows and columns.

```
import java.awt.*;
public class Tortoisetry
{
   public static void main (String [] args)
   {
      Tortoise Ted = new Tortoise (50, 10); ◄Scale and rows and columns parameter
      Ted.pathOn ();
      Ted.setPosition (1, 1);
      Ted.faceEast ();
      for (int j = 0 ; j < 2 ; j++)
      {
       for (int i = 0 ; i < 3 ; i++)
       {
          Ted.move ();
       }
       Ted.turn ();
       for (int i = 0 ; i < 6; i++)
       {
          Ted.move ();
       }
       Ted.turn();
      }
   } // main method
} // Tortoisetry class
```

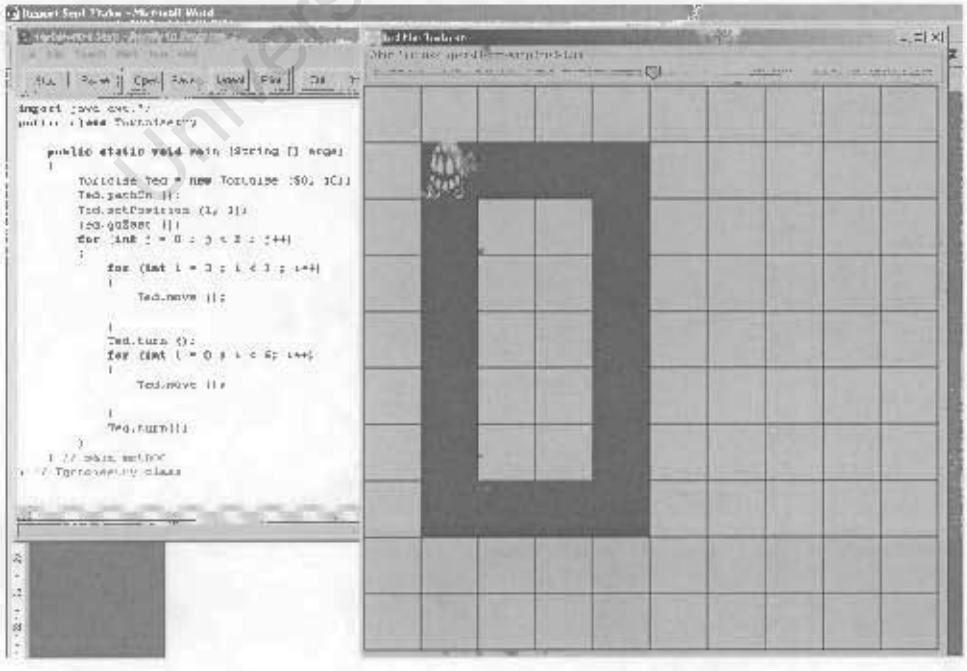


Figure 7 Scaled up version with different number of rows

1.4.2 The Grid class

This was used to enable two or more tortoises to appear on the same frame, which enhances the functionality of the program considerably. The default frame is created by the first Tortoise constructor. There are a number of constructors to match the Tortoise call for a scaled up display or an increase in the number of rows and columns in the grid. Each time a constructor is called it checks to see if an instance of the Grid class exists. If no memory address has been assigned then a new Grid object is instantiated. Subsequent tortoises then use this instance of Grid and a count of current tortoises is kept. The appropriate draw() method belonging to that particular instance of Tortoise can then be invoked by the Grid object's paint() method (See Appendix A).

Example using 3 Tortoises

```
public class Tortoisetry
{
   public static void main (String [] args)
   {
      Tortoise Tom = new Tortoise (35, 20);
      Tortoise Dick = new Tortoise ();
      Tortoise Harry = new Tortoise ();
      Tom.pathOn ();
      Dick.pathOn ();
      Harry.pathOff ();
      Tom.goTo (1, 1);
      Dick.goTo (6, 1);
      Harry.goTo (4, 4);
      Tom.faceEast ();
      Dick.faceEast ();
      for (int j = 0 ; j < 2 ; j++)
      {
         for (int i = 0 ; i < 7 ; i++)
         {
            Tom.move ();
            Dick.move ();
            Harry.move ();
            Harry.says ("Counting " + i);
         }
         Tom.turn ();
         Dick.turn ();
         Harry.faceWest ();
         for (int i = 0 ; i < 6 ; i++)
         {
            Tom.move ();
            Dick.move ();
         }
         Tom.turn ();
         Dick.turn ();
      }
   } // main method
} // Tortoisetry class
```

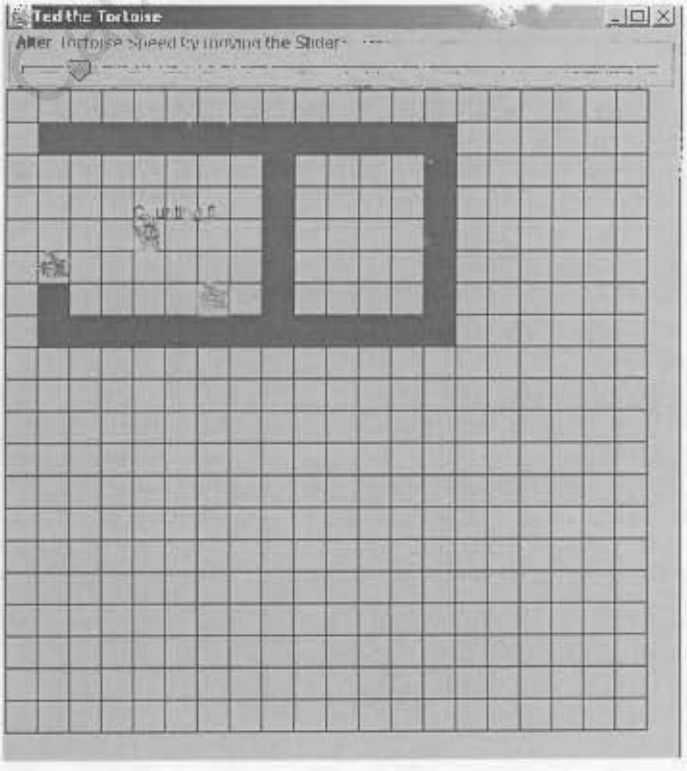


Figure 8 More than one tortoise

This is a very important aspect of the program, especially for Computer Studies students. It allows the development of programs which can involve methods created by students. They can then extend the existing classes and incorporate their own methods. Tortoises are not automatically identified as different. It is left to the student to differentiate them by using a label. The concept of Threads can then be developed in a meaningful way. Having more than one Tortoise on the screen allows students to observe if the tortoises were moving at the same time or one after the other. The question "Can I get the tortoises to move at the same time?" arose from one Computer Studies student's experimentation. This provided an opportunity to introduce the idea of threads in an immediate meaningful way. An example is shown below.

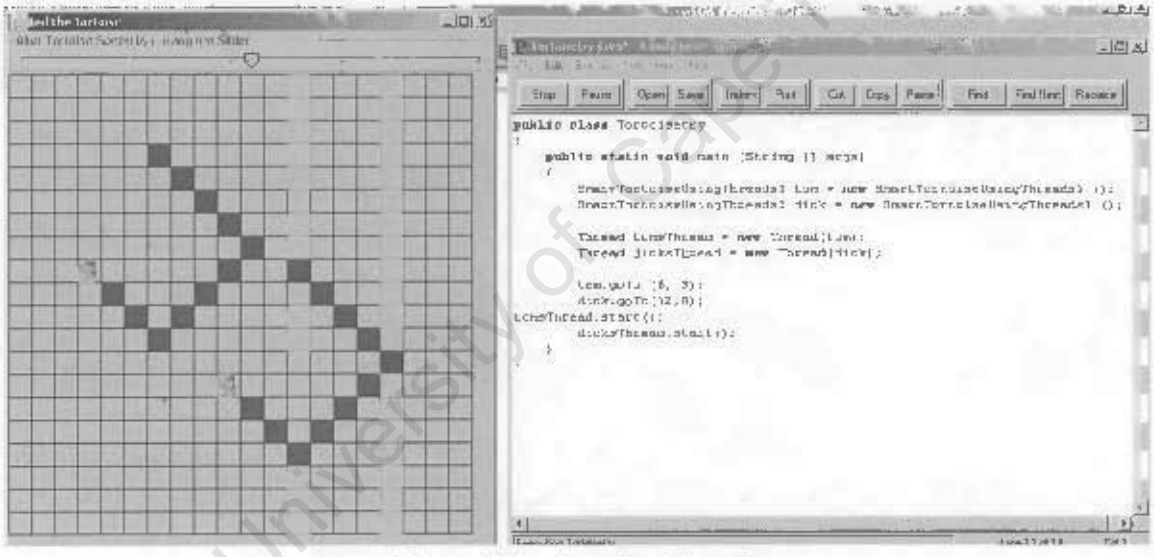


Figure 9 Tortoise using Threads

```
public class Tortoisetry
{
   public static void main (String [] args)
   {
      SmartTortoiseUsingThreads1 tom = new SmartTortoiseUsingThreads1 ();
      SmartTortoiseUsingThreads1 dick = new SmartTortoiseUsingThreads1 ();

      Thread tomsThread = new Thread (tom);
      Thread dicksThread = new Thread (dick);

      tom.goTo (6, 3);
      dick.goTo (12, 8);

      tomsThread.start ();
      dicksThread.start ();
```

```
    }
}

// The "SmartTortoise" class.
import Tortoise.*;
public class SmartTortoise extends Tortoise
{
   public SmartTortoise ()
   {
      super (35,20);
   }

  // Additional method to make a diamond
   public void makeDiamond ()
   {
      this.pathOn ();
      this.faceSouthEast ();
      for (int i = 0 ; i < 4 ; i++)
      {
         for (int j = 0 ; j < 4 ; j++)
         {
            this.move ();
         }
         this.turn ();
      }
   }
} // SmartTortoise class

// Enabling Threads
import Tortoise.*;
public class SmartTortoiseUsingThreads1 extends SmartTortoise1 implements Runnable
{
   public SmartTortoiseUsingThreads1 ()
   {
      super ();
   }

   public void run ()
   {
      this.makeDiamond ();
   }
}
```

1.4.3 Movement Methods

Movement methods were in two categories. The first methods provided simple movement and turning. The other methods required more precise instructions from the user.

Simple:
move() - moves one square in the current direction
turn() - turns clockwise 90° from current direction

A method for moving ahead for a given number of squares was not provided. This method would simply have repeated the move() method for a given number of times. This was seen as an opportunity to force the student to use a loop. In their terms this would probably amount to avoidance of repetitious behaviour but discussion of this facility (or lack thereof) could be used to modify students' thinking in analyzing problems to highlight that feature of problem solving. It could be used to point out clearly that one continuously comes across problems where identification of repeated activities is crucial.

A set of methods is provided to set the direction:
faceNorth(), faceSouth(), faceEast(), faceWest()
faceNorthEast() , faceNorthWest(), faceSouthWest(), faceSouthEast()

Direction is controlled as an attribute: for example: **char** bearing = 'E';

Cardinal		**Half Cardinal**	
E	East (right of the screen)	e	NorthEast (45° up and to the right)
S	South	s	SouthEast
W	West	w	SouthWest
N	North	n	NorthWest

(None of these parameters are explicitly accessed by the user. They are altered internally when the user calls a method such as faceNorthEast())

Tortoise movement could have been designed to be toroidal or to indicate that the grid had been exited when a user exceeded the grid size. At the level at which classes worked for this trial it was not of any consequence. The choice of both was built into the program and defaulted to not being able to wrap around the grid. In addition it was decided not to halt the program when this occurred. When the Tortoise hits an edge, it displays the message "Ouch" (in a bubble), and a friendly message to the effect that it has banged its head on the edge, and then carries on drawing at that point. So, if movement were in a loop then the remainder of the loop would result in a continual banging of heads. This was a light-hearted way to deal with this issue for young students. This was handled in the draw() method in the Tortoise class.

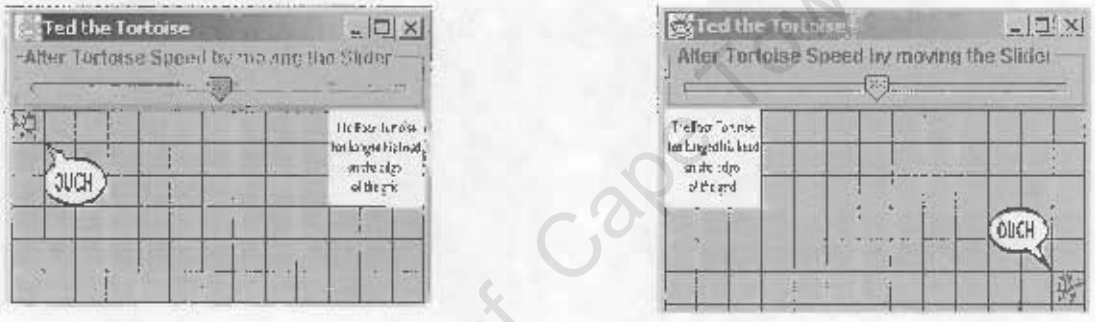


Figure 10 Overstepping the grid boundary

1.4.4 Miscellaneous Methods

The cleanup() method was provided mainly to stimulate sequencing of activities. For example, the tortoise could draw a pattern, then remove it and go on to draw something else. The method also served to initialize the grid cleanly

The says() method allowed a label to be attached to the tortoise.

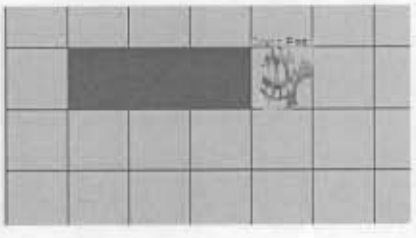


Figure 11 The "says" method

The pathOn() and pathOff() methods allowed the student to display a tortoise trail, identical to the logo idea of penUp and penDown,

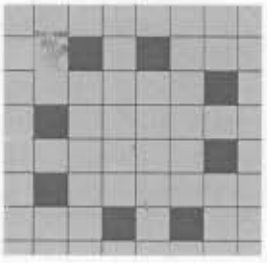


Figure 12 Path On/Off

The goTo() method was used to jump to a given position on the grid.

The slowdown() method was used to cause a delay. This was linked to the JSlider to allow the tortoise display to be speeded up or slowed down. Sometimes the display was flashy. The delaying method also improved the visual effect when placed at suitable points in the program (For example Tortoise.java Line 136). This is an issue involving the processing of images and update of display.

The getAge() and ageIs() methods associated to the age attribute were the beginnings of a set of methods which can be used to introduce the concept of attributes and how to access them.

1.4.5 Students' Work

Some examples of students' work are shown on the following pages. These were collected as lessons progressed and the order of the examples shown is roughly chronological. Due to normal school timetable commitments (such as holidays and cultural events in school), some classes attended lessons over a much longer period than others, sometimes with significant gaps between the lessons. For this reason the level of skill displayed does not follow a pattern in these examples. In the short period available many students only managed to complete one or two projects. With hindsight I would have tried to allocate a much longer period of time to this study and chosen a quiet time of the school year. I would then have taken samples of programs and the output produced by certain students over the entire period of the study in order to compare and contrast them.

The initial lessons involved students first completing a simple square shape. They were then asked to try to create their own work and were given a minimum of instruction and no

suggestions. The more creative students experimented and produced some very interesting shapes immediately. Many of these were then copied by the less able students. A small number of students started to produce much more complicated shapes and wanted to know if there was a quicker way of producing the same result. These students were receptive to learning to use more complicated syntax. Most students started out with relatively simple shapes but did use larger grids right from the start of their experimentation, for example, Figures 16, 19, 22, and 25. Some students used the uncoloured squares to create the pattern, for example, Figure 14. Most students were focused on producing patterns but as the lessons progressed some students started to use labels, for example, Figures 26 – 29. Many of the shapes did not lend themselves to the use of timesaving structures (loops), for example, Figures 15, 20, 23, 24, 25-31, 33, 34 38, 40. A small number of students did incorporate loops, for example, Figures 16, 17, 19, 41 (Figures 16 and 19 were revised by the students to incorporate loops. Figure 19 was done by a Computer Studies student). Towards the end of the period of instruction some students began to experiment with more than one Tortoise instantiation, for example, Figures 36, 39, 40, 43 and 45.

There were few, if any, completely random patterns. One might have expected this if students focused on programming statements to produce output. Students seemed to have a pattern or familiar shape in mind and then concentrated on achieving that end. Papert gave a number of examples which had been created by students but these quite often involved shapes and patterns which were suitable for creation by repeating turtle movements (Papert, 1980). Perhaps the children in his study were encouraged to look for repeatable activities or perhaps the children in this study had not reached a stage where they were comfortable with placing code in loops.

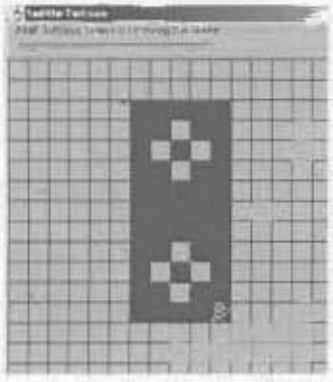


Figure 13 Student 21 G9D

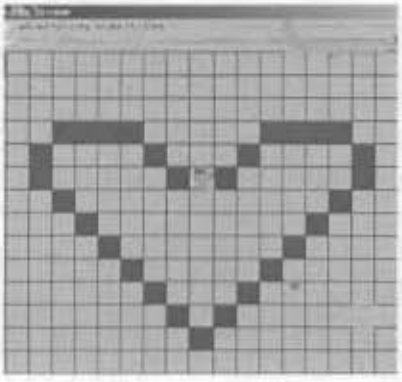


Figure 14 Student 12 G9D

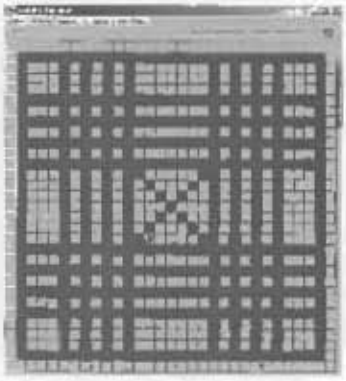


Figure 15 Student 1 G10R

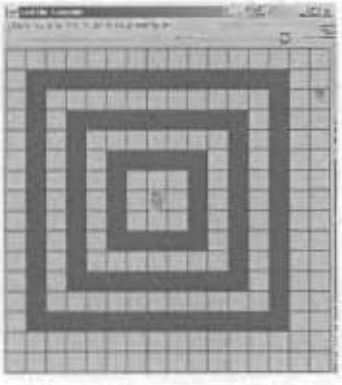


Figure 16 Student 18 G9R

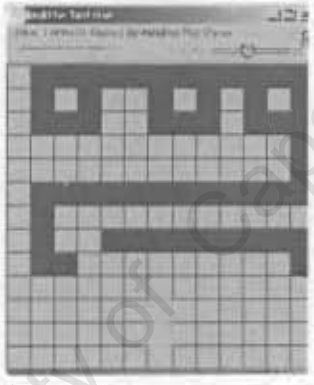


Figure 17 Student 3 G9D

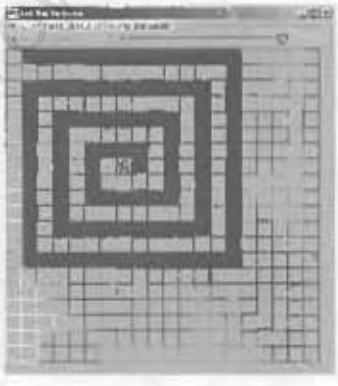


Figure 18 Student 29 G9R

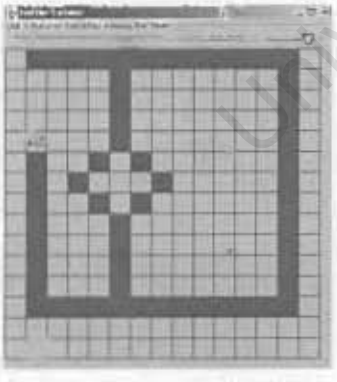


Figure 19 Student 27 G9M

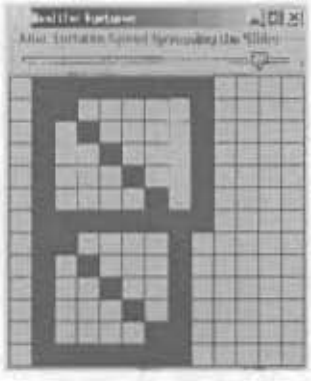


Figure 20 Student 2 G9M

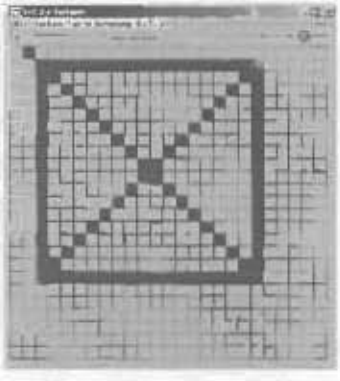


Figure 21 Student 2 G9M

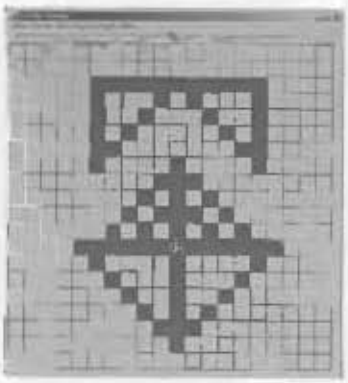


Figure 22 Student 22 G9D

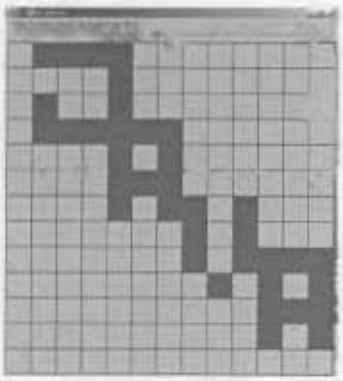


Figure 23 Student 4 G9D

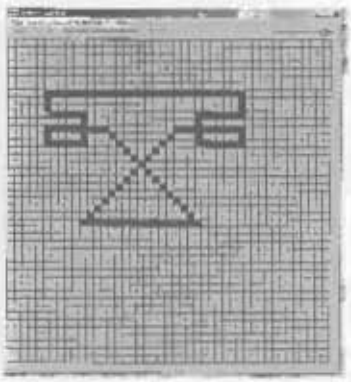


Figure 24 Student 17 G9D

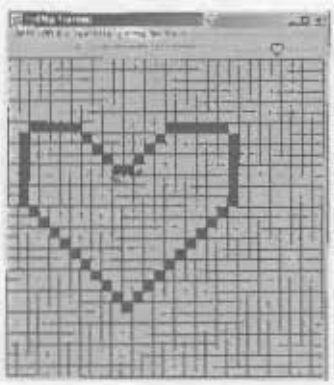


Figure 25 Student 7 G9E

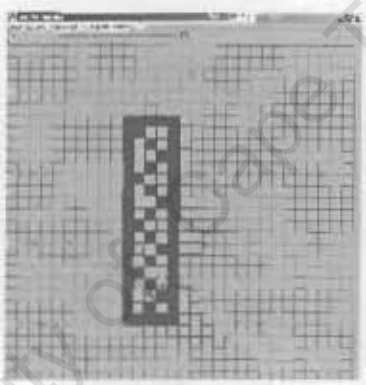


Figure 26 Student 8 G9E

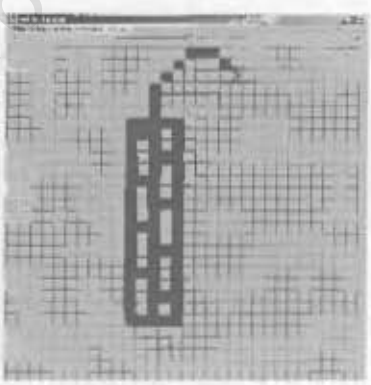


Figure 27 Student 8 G9E

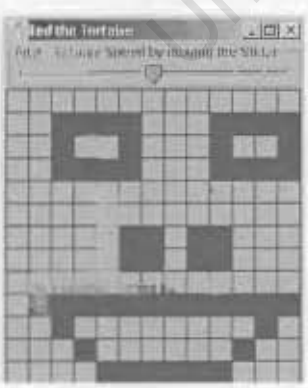


Figure 28 Student 23 G9E

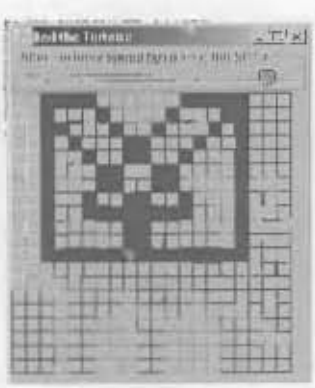


Figure 29 Student 19 G9E

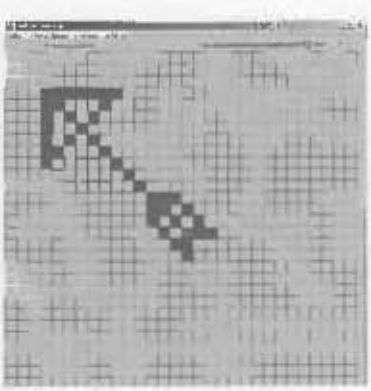


Figure 30 Student 4 G9E

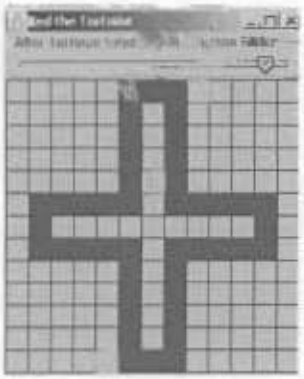


Figure 31 Student 25 G9E

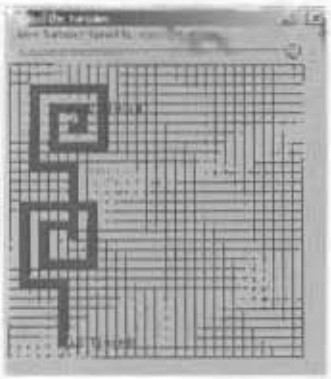


Figure 32 Student 17 G9E

Figure 33 Student 21 G9M

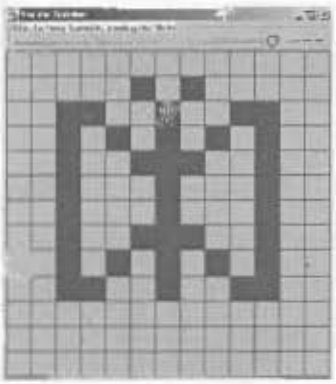


Figure 34 Student 8 G9E

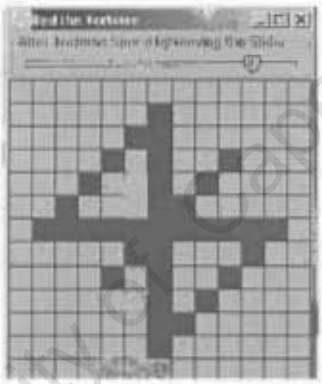


Figure 35 Student 10 G9M

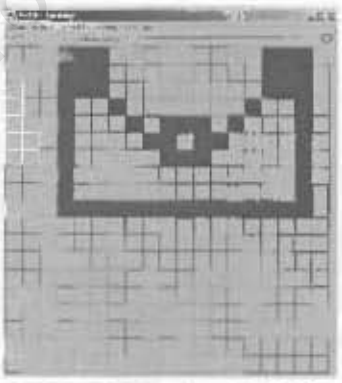


Figure 36 Student 24 G9M

Figure 37 Student 18 G9M

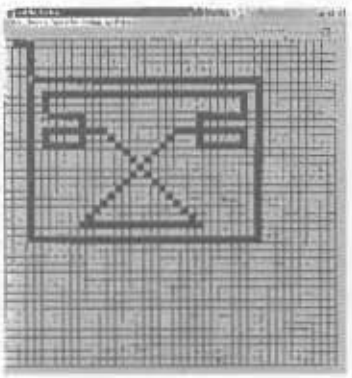


Figure 38 Student 17 G9D

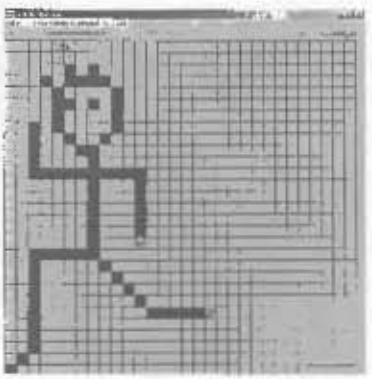


Figure 39 Student 24 G9D

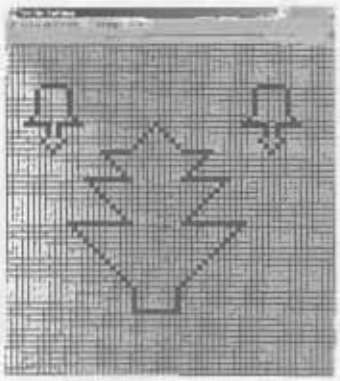


Figure 40 Student 18 G9D

Figure 41 Student 5 G9M

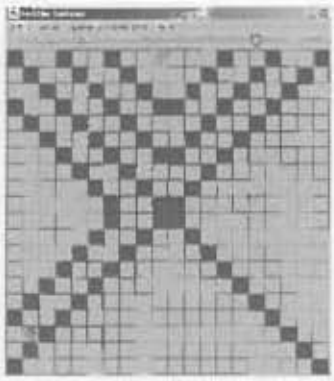


Figure 42 Student 11 G9R

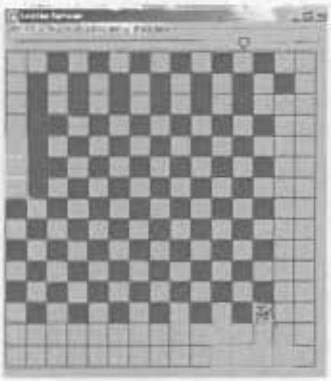


Figure 43 Student 9 G9R

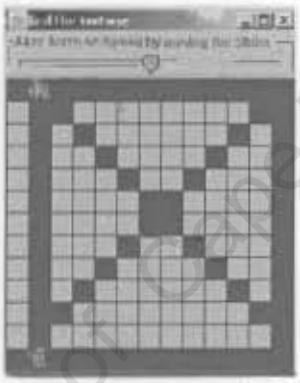


Figure 44 Student 24 G9R

Chapter 2 The Please Class

The Please class is designed to simplify input for the novice programmer. The question of how to introduce programming to novices is one that comes up again and again. Seymour Papert's view is that learning is much more to do with the individual's experience than the structure of teaching or what the teacher says. He refers to it as "learning without being taught" (Papert, pp7, 1980). However, it is crucial that a novice can progress in programming without the continual frustration caused by inherent problems of the language. To this end a number of authors have considered the suitability of Java as a first programming language (Biddle and Tempero, 1998, Fleury, 2001, Grissom, 2000, Hosch, 1996, Jenkins, 2002, King, 1997, Lea, 1996, Madden et al, 2002, Matthíasdóttir, 2004, Miller et al, 1994, Moström and Carr, 1997, Proulx and Rasala, 2004, Ramalingham and Wiedenbeck, 1997, Roberts, 2001, Wolz and Koffman, 1999). "While Java has much to recommend it as a language for advanced students, the standard implementation—which includes the various libraries in the Java Development Kit provided by Sun— forces introductory students to assimilate too much information too fast, making it difficult for many students to understand the critical conceptual issues involved in programming and algorithmic design" (Roberts, pp 1, 2001). Roberts then goes on to discuss the response of SIGCSE[4] members in 2000 to a question regarding novice programmers (also discussed in Grissom, 2000, Proulx and Rasala, 2004). Many of the responses advocated the creation of a set of classes to circumvent the difficulties that novices met early on in their learning of the language. Interactivity is attractive to novices (Wolz and Koffman, 1999) but one of the first things that causes problems for novice Java programmers is the difficulty in creating a program which contains input from the keyboard. For example, the conceptual background needed to input an integer is significant. Typical code, taken from an introductory high school course text book, is shown below (Wilcock et al, pp 45 ,2002).

```
BufferedReader stdin =  new BufferedReader (new InputStreamReader (System.in));
int value;
while (true)
{
   try
   {
      String valueString = stdin.readLine ();
      try
      {
```

[4] Special Interest Group on Computer Science Education

```
            value = Integer.parseInt (valueString);
            break;
        }
        catch (NumberFormatException e)
        {
            System.out.println ("Not a valid integer: " + valueString);
        }
    }
    catch (IOException e)
    {
        System.out.println ("I/O error while reading from standard input!");
        System.exit (0);
    }
} // while
```

Some of the concepts in this example that novices find difficult include the loop, the boolean value, the wrapper class and exception handling. Grissom suggests that there are four approaches to introducing Java input and output – text only, text followed by GUI, GUI from the outset or almost no input/output code generated by students (Grissom, 2000). Within the context of text only input and output Grissom argues that although it might be challenging, the use of standard Java classes should be within reach of novices. This may be true of undergraduates but is not necessarily the case for many high school students. A number of educators and authors believe that input in Java is a serious impediment to a novice's progress and have created their own classes in an attempt to deal with this issue (Baldwin, 1999, Becker, 2003, Bishop, 2001, Garside and Mariana, 1987, van der Linden, 1997, Wilcock et al, 2002, Winder and Roberts, 1998, Wolz and Koffman, 1999). All of these researchers agree that specialist input classes should be used only as a temporary measure. Grissom suggests that the transition from the proprietary class to standard Java classes can be made into a positive teaching experience. "Revealing the design and internal implementation details of a package can be an effective case study" (Grissom, pp 58, 2000). The Please class was created to facilitate the use of a number of useful activities which might cause problems associated to syntax.

2.1 Program Structure

The class was designed to be used by invoking class methods, thereby avoiding the instantiation of an object. The name "Please" was chosen for the class[5]. The class was

[5] For a full program listing please see Appendix C. For Javadoc HTML documentation see Appendix D.

made up largely of static methods to provide a familiar natural feel to the syntax. For example:

Please.helpme() - indicates that the user wants assistance

int n = Please.getNumber() - allowed the user to input an integer

All primitive types were retained because it is not desirable to provide only one method for many types. One runs the combined risk of having to re-teach students at a later date and of over-simplifying the subject matter to the extent that it loses its significance. Some types (**int**) were referred to using simple user-friendly method names (**getNumber()**) and were taught first. More specialist types (**long**) were left until the programming need arose and were given more Java-like names (**getLongInteger()**) in order to assist the transition from the easy introductory class to traditional Java syntax. The use of **getNumber()** allows the student to try a real number but then reports the error with the message "*Please try again* – **getNumber()** *only gets integers*". This aimed to sensitize students to the existence of different numerical types and the need for exception handling at a later stage.

The class methods were grouped into simple general input of real numbers and integers, with additional methods for specific Java types. For example:

Please.getDecimal() allows the user to input a double and calls on the Grade 9 high school students' current mathematical knowledge of integers and real numbers (commonly referred to inaccurately as "whole" numbers and "decimal" numbers)

Please getShortInteger() specifies the type to be inputted.

Please.getLetter() allows the user to input a character. Many students expect only alphabetical characters but this should then be consolidated with a discussion of Unicode characters.

Please.getString() allows the user to input a string. Once again a full discussion consolidating the use of the "char" primitive type and the "String" class is the aim.

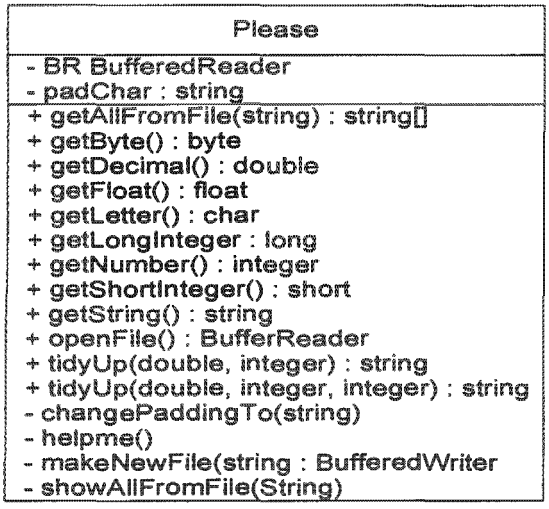
Simplified access to files was provided. For example:

Please.showAllFromFile(<Filename>)	opens the file and displays the contents.
Please getAllFromFile(<Filename>)	retrieves a file's contents and places them into an array of strings.
Please.makeNewFile(<Filename>)	initializes a BufferedWriter to allow access to a file named as a parameter (String).

Simplified formatting operations were implemented. For example:

Please.tidyUp(<decimal number>, <decimal places>)	specifies number of decimal places. Parameters are the original number(Double) and the number of decimal places (Int)
Please.tidyUp(<decimal number>, <decimal places>, <field width>)	specifies width of the number and number of decimal places. Parameters are the number (Double), number of decimal places (Int) and the width of the final number made up of the number digits before the decimal point plus one for the point plus the number of digits after the decimal point (Int).
Please.changepaddingTo(<character>)	allows the user to specify something other than zeroes to use as padding when formatting a decimal number. The parameter is the Unicode character that has been chosen (Char).

2.2 Please Class Diagram



2.3 Program Design

The String Tokenizer class was used to facilitate input. This allows one to create an instance of the class using the input stream data as a parameter. The inputted data is treated as a line of tokens separated by white space. The hasMoreTokens() method checks for any more data in the input stream and is accessed by invoking the nextToken() method. The white space is one or more spaces or a separator. An additional constructor allows one to specify the separator.

Numerical Input

The general format of the numerical input methods is shown below.

```
/**
* Class Method to input an Integer
 */
public static int getNumber () throws IOException
{
    try
    {
        StringTokenizer ST = new StringTokenizer (BR.readLine ().trim ());
        return Integer.parseInt (ST.nextToken ());
    }
    catch (NumberFormatException nfe)
    {
        System.err.println ("Please try again - getNumber only accepts integers ");
    }
    return 0;
}
```

Alternative form:

```
public static int getNumber () throws IOException
{
    try
    {
        StringTokenizer ST = new StringTokenizer (BR.readLine ().trim ());
        return new Integer (ST.nextToken ()).intValue ();
    }
    catch (NumberFormatException nfe)
    {
        System.err.println ("Please try again - getNumber only accepts integers");
    }
    return 0;
}
```

return Integer.parseInt (ST.nextToken ()) was used to convert the value (on advice from my tutor). The **return new Integer (ST.nextToken()).intValue ())** and associated methods are used by Judith Bishop in the Stream class (Bishop, 2000)

2.4 Method Summary

Numerical input methods

getNumber ()	returns an integer
getLongInteger ()	returns a long integer
getShortInteger ()	returns a short integer
getByte()	returns a byte
getDecimal()	returns a double
getFloat	returns a float

Non-numerical input methods

String and Character input was also catered for:

getLetter()	returns a letter
getString()	returns a string

File handling methods

There are four file handling methods:

openFile (String filename)	returns a BufferedReader
makeNewFile(String filename)	returns a BufferedWriter
showAllFromFile (String filename)	displays the contents of a file
getAllFromFile (String filename)	returns an array of strings (the file contents)

Format methods

There are three formatting methods:

tidyUp(double n, int d)	takes a double with a specified number of decimal places and returns a formatted String, if the formatting is sensible.

and the overloaded method:

tidyup (double n, int l, int d)	takes a double with a specified number of decimal places and a field width and returns a formatted String, if the formatting is sensible.
ChangePaddingTo(String ch)	alters the character to pad the field width.

For example:

```
double n = 6.12345;
Please.ChangePaddingTo(" ");
System.out.println(Please.tidyup(n,3)+"XXXX");
System.out.println(Please.tidyup(n,8)+"XXXX");      More than the available number of decimal places
System.out.println(Please.tidyup(n,7,3)+"XXXXX");
System.out.println(Please.tidyup(n,2,3)+"XXXXX"); Field width less than the number of decimal places
```

which gives the following output:

```
6.123XXXX
6.12345XXXX          inappropriate format ignored
6.123  XXXXX
6.12345XXXXX         inappropriate format ignored
```

2.5 Students' Reaction

Three groups of students were questioned individually during their course of instruction and questioned in classes after the twelve week period of introductory instruction had finished. The evidence presented here is anecdotal and is based on this group's reaction to the introduction of Java using the Please class compared to last year's group who were exposed to the standard classes only. One other class started programming in Pascal in the previous year (This was the last group to use the Pascal language).

The students who used the standard Java classes were difficult to motivate as a group. The capable students within the class were able to successfully defer explanations until a later time in the interests of completing programs but expressed frustration by the pace at which the lessons progressed. This was slow because of the continuous help and reassurance that the less able students required. The more capable students also indicated that having to defer explanations (especially with regard to input) left them feeling unsure of what they were doing even though they accepted the situation. This group of students was then shown the Please class some time after they had been working with Java and indicated that they felt that the simplification would have helped. Most suggested that they would not have felt so helpless. There was an almost unanimous feedback from the class which suggested that they felt a great lack of confidence and a few said they had wanted to give up. There was a concern that things were only going to get more difficult and that they were making no progress. They indicated that anything which would have allowed them to get some positive results would have helped. It is difficult to assess if this was merely wisdom in hindsight but their comments did indicate that their introduction to Java was an uncomfortable experience for many and a stressful experience in some cases. It seems that there was a possibility that the Please class could have helped them to feel less overwhelmed. These remarks were made once the initial exposure to Java had been completed and students were feeling more at ease with many of the concepts but the fact remains that unsolicited comments repeatedly indicated the feelings of inadequacy in dealing with the syntax initially. Many of these students might have had a more pleasant introductory experience by using the assistant class. This observation is mirrored by a number of similar studies which looked at the introduction to programming at the

undergraduate level (Halland and Malan, 2003, Ross and Zhang, 1997, Wolz and Koffman, 1999). There is no claim made here that this contention has been proven but the reaction of the students does indicate that some further investigation into high school novice programmers' experiences would be in order.

Many students at this stage of their education career are focusing for the first time on an important academic goal. This is the year where they commence on a three year course of study culminating in matriculation examinations. From the feedback received from teachers and counselors responsible for pastoral care it is clear that many high school students feel that they are under significant pressure to succeed academically for the first time. Comments from students indicate that many of them feel that programming is a unique experience and represents a novel way of thinking. Many feel unsure of themselves and feel threatened by the number of mistakes that they make. Many of their comments followed a similar theme. There was "so much to remember". "There is a huge amount of new stuff". "You get tired of all the error messages". "I get confused by all the things you have to put into the program". "It takes a while to get used to it". Anything which might reduce this pressure would seem to be academically justifiable.

The Pascal group by comparison was focusing much more on the problems presented. The syntax was much more acceptable in terms of not ending up as the main focus. They commented on the strangeness of the words but conversation moved on to problems associated with variables and the methodology of solving problems. The only parallel to Java was the necessity to use the "uses crt" clause to facilitate clearing of the screen. This elicited some comments such as "I would have liked to know how that worked". Others accepted it as something which should appear in every program and did not question it. The Pascal syntax seemed to be accepted much more quickly than the Java syntax.

The reaction of the Grade 10 Computer Studies students (15 students, all male) to the Please class was mixed. When reviewing their learning experience the majority commented on the amount of new information they had to assimilate. None of them felt that it would have been better to have used the standard java syntax from the beginning. It was better to be gradually exposed to the syntax. They felt more comfortable in

successfully getting a program to compile and execute quickly. The number of errors was once again a concern. Some students commented that the language "looked so strange" and there seemed to be no time to assimilate the ideas because you "had to spend so much time fixing up all those little things". Some of this group also said that there was "too much information". The majority also said that it was important to get on to the "real" Java syntax as soon as possible. Once they had become familiar with the various variable types and how and when they were used, they felt that the class should be dropped entirely. Many suggested that the class should be used in tandem with the introduction of native syntax. They also suggested that the Tortoise should be the first thing that should be used. They commented to the effect that it was enjoyable to experiment with the Tortoise class but noted that although some Java syntax was built into it they felt comfortable working with it. They felt that they were forced to deal with syntax, but in a limited way that was easy to cope with. Some suggested that the Please class could follow on from work with the Tortoise class or it could be done at the same time. They unanimously agreed that the timeframe for its use should be limited. Once again this may well have been a case of wisdom in hindsight and some of the comments appear to be contradictory. However, their comments on the introductory phase were more focused on the positive aspects of the assistant class than on the negative aspects of the raw Java syntax. This group seemed much more willing to accept the syntactical problems.

It would seem that the use of assistant classes is beneficial in the short term. Based on the reactions of these groups of students it appears to have something to do with the amount of new information that the students are exposed to. Errors also seem to be a source of concern. The students' reaction was similar to that reported from studies done with undergraduates (Halland and Malan, 2003, Ross and Zhang, 1997, Wolz and Koffman, 1999).

Chapter 3 Testing

3.1 Questionnaires – Notes on questions

Two questionnaires were formulated and given to four Grade 9 classes, one after a short introduction to Java and the other after a period of instruction. Grade 9R is streamed with academically successful students. Grades 9E, D and M are mixed ability. All classes are mixed male and female in the 13-14 years age group. Numbers vary from 24 to 28 per class. Classes have three one hour lessons based on a two week cycle, usually two lessons in week one and one lesson in week two. The classes were given an introduction and then one lesson of work before the first questionnaire was given out. The classes then worked for three further lesson periods, after which the second questionnaire was given out. The time period was not ideal but, unfortunately, with standard school internal examination and project commitments this was all that could be accommodated in the time available. Learning could occur through personal experience, instruction from the teacher or through cooperation with peers. No attempt was made to discourage or encourage cooperation among peers. The response to questions during instruction, whether from individuals or from pairs or groups, followed the pattern that students were familiar with. The objective of the questionnaires was to try to show that some change in behaviour occurred as the lessons progressed. The questionnaires attempted to get some information on concept development. The following is a breakdown of the questions used and the rationale behind them. The final questions in the questionnaire were repeated in questionnaire one and questionnaire two. Although no formal discussion of the questionnaires took place through the teacher, some students may have re-thought their solutions or discussed them with their peers out of class. This was viewed as a normal acceptable form of learning.

Question 1.

Questionnaire 1 *How would you get a tortoise called "**Thabo**" to draw this rectangle on the screen? Just give the movement instructions.*

public.....
.....)
{.....
}

► *Put the movement instructions just below HERE*

This question was accompanied by a graphic of the smallest and simplest closed shape that the tortoise might create. Students could repeat the movement instructions, 11 in all, to achieve the shape. The students were asked to provide only the methods to obtain the shape. This was intended to be the simplest of the questions to encourage students to participate meaningfully in the questionnaire.

Questionnaire 2 *How would you get the tortoise to draw this rectangle on the screen in the most efficient way?*

This question provided a large rectangle. Students could once again repeat instructions but the size was deliberately chosen so that it would be arduous to do so to achieve the shape. The question indicated that students should attempt to find a quicker/easier way to complete the task.

Expected outcome More students should be successful the second time around and use a loop.

Question 2.

Questionnaire 1 *The following program might not work. It might give errors when we try to run it. If you think it will run correctly then say that you think it is OK. If **NOT** then correct it by adding in lines and/or removing lines and/or changing lines.*

```
public class Tortoisetry
import Tortoise.*;
import java.awt.*;
{
 public static void main (String [] args)
  {
     pathOn ();
     setPosition (1, 1);
     faceEast ();
     move ();
     move ();
     turn();
  }
```

This question was aimed at testing the students' recall of Java syntax. Each program they completed in class required standard syntax to create a tortoise object and associated methods so it was expected that this should be a straightforward recall of a familiar activity. Several methods were provided but the object references were omitted. Import statements were also provided. It first required the instantiation of a Tortoise and then the addition of the appropriate Object references in front of the given methods.

Questionnaire 2 *Every Java Tortoise program needs certain lines of code otherwise it doesn't work. It will give errors when we try to run it. What are these vital instructions?*

```
public class Tortoisetry
►
►
{
 public static void main (String [] args)
 {
►
 }
```

The Second questionnaire question had the same objective but was worded in a more formal manner. Instruction in class began by emphasizing the necessity of items such as "import" statements and students should by this stage have been familiar with the standard syntax necessary in each of the programs they completed. The omissions were further highlighted using "►" at the appropriate position. Standard instantiation statements and import statements were required.

Expected outcome Students using a tortoise should have sufficient motivation to successfully remember this, at least partially. More students should be successful the second time around.

Question 3.

Questionnaire 1 *The following is a* ***section*** *of a program. It allows me to get a tortoise called "Betty" to move to the top left of the screen before she starts drawing a pattern but it is a rather long way to do this. What is the most efficient method to get Betty to get to the starting position?*

```
public class Tortoisetry
import Tortoise.*;
import java.awt.*;
{
 public static void main (String [] args)
```

```
{
    .....
    .....
    betty.pathOff ();
    betty.turn ();
    betty.move ();
    betty.move ();
    betty.move ();
    betty.move ();
    betty.move ();
    betty.move ();
    betty.move ();
    betty.move ();
    betty.turn();
    betty.pathOn()
}
```

The question was designed with a large number of statements and no accompanying graphic was provided. It was hoped that the students would see this as an inefficient approach and try to recall an alternative constructor method.

Questionnaire 2 *When you create a new tortoise, you have a choice of display, size, starting position etc. What options do you have and how do you access them?*

```
public class Tortoisetry
import Tortoise.*;
import java.awt.*;
{
  public static void main (String [] args)
  {
► Write the different options below here
```

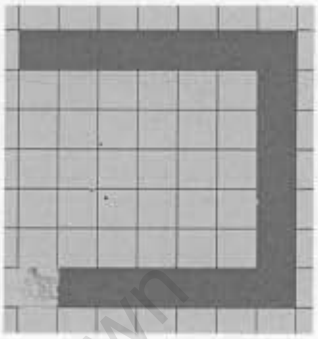
The second questionnaire question continued the theme of the first but asked the question in a more formal manner. It required the recall of a number of alternative constructors and students would need to have assimilated the concept that a variety of constructors could exist to provide enhanced or different functionality. The question in both questionnaires required the students to have assimilated the concept of a constructor method.

Expected outcome More students should be successful the second time around.

Question 4.

Questionnaire 1 *The following is a **section** of a program which does run properly. It seems like a lot of lines of code to have to type. What is the <u>best</u> way of doing this? Suggest a method or change the lines of code*

```
.....
ted.setPosition(1,1);
ted.faceEast();
ted.move();
ted.move();
ted.move();
ted.move();
ted.move();
ted.move();
ted.turn();
ted.move();
ted.move();
ted.move();
ted.move();
ted.move();
ted.move();
ted.turn();
ted.move();
ted.move();
ted.move();
ted.move();
ted.move();
ted.move();
ted.turn();
.....
```



This question explicitly used a large number of invocations of the same method. Students were asked to alter this in such a way as to make it more efficient. The graphic did not indicate a closed shape in order to focus attention away from the completion of a geometric pattern. It was hoped that students would concentrate on the programming statements provided.

Questionnaire 2 *What is the best way of writing a program to get a tortoise to repeat some actions over and over again? Comment below and give an example if you think it's suitable. (You don't need to write a whole program)*

The second questionnaire asked a formal question which was aimed specifically at eliciting a comment about syntax. The question asked for the "best way" to try to focus the students' attention on improving on the first attempts they made.

Expected outcome More students should successfully employ a loop the second time around.

Question 5.

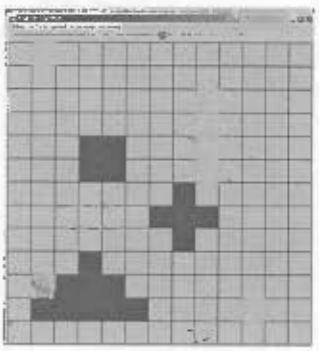
Questionnaire 1

A Tortoise has three new methods called makeSquare, makeCross and makeTriangle.

makeSquare draws a solid Square,

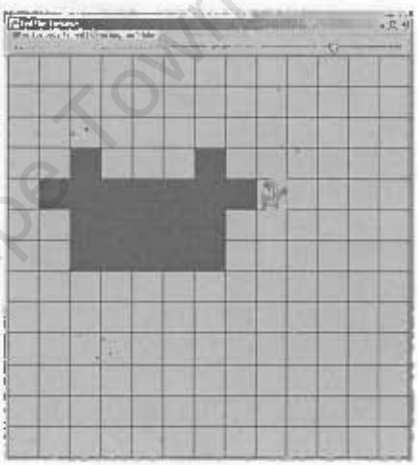
makeCross draws a solid cross and

makeTriangle draws a solid triangle.



Write the lines of code which would draw the shape shown below.

```
public class Tortoisetry
import Tortoise.*;
import java.awt.*;
{
public static void main (String [] args)
{
    ..........
    ..........
```



This requires the use of the stated methods (which they have not met). They are used in sequence and need to be placed in the correct position beside each other to obtain the given composite shape. Students were gradually introduced to new methods during the course of instruction where the point was made that each new method provided advantages over what they had already done. Teaching was structured to try to create the expectation that new methods tended to combine previous repetitive tasks into one more efficient method. No mention is made of start position in this question but the full graphic grid was provided to indicate where the composite shape was placed. A student would have to supply the setPosition() method as well in order to have a completely correct solution.

Questionnaire 2

The same question was used in the second questionnaire. No solutions were provided after administering the first questionnaire and no examples of a similar type were provided. The

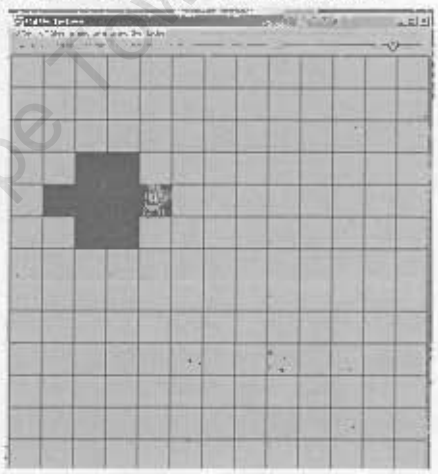
questionnaires were not discussed with the students at any stage. Having already attempted the question once a student might have considered a better solution as they met the same difficulties in class and saw how more complex methods might achieve results more efficiently. Students may also have discussed the question among themselves. This was not discouraged but no input was provided from the teacher.

Expected outcome More students should be successful the second time around. A student who ignored the given methods on the first questionnaire would invoke them on the second questionnaire

Question 6.

Questionnaire 1 *Write the lines of code which would draw the shape shown below.*

```
public class Tortoisetry
import Tortoise.*;
import java.awt.*;
{
 public static void main (String [] args)
 {
     ..........
     ..........
```



This question required the repeated use of the same shape. The supplied display is deliberately made to look like a broad cross. One of the more efficient methods of drawing it requires two crosses from the makeCross() method) to be partially superimposed. (The same result can be attained by freehand drawing or using a shape and freehand but this was regarded as only a partial success). A student will have to first solve the problem and then decide on the program statements to apply the solution.

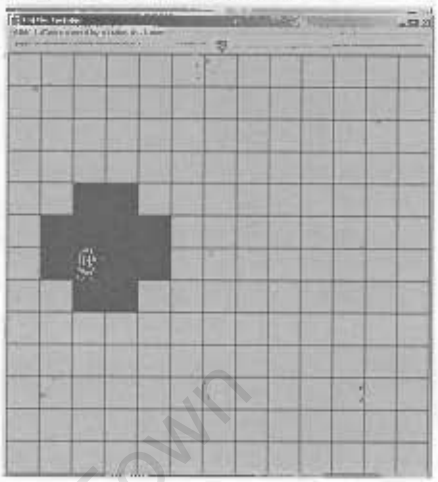
Questionnaire 2 The same question was used again with no solutions or discussion. Students once again might apply their experience to come up with a better solution than they applied the first time around.

Expected Outcome More students should use only shapes the second time around.

Question 7.

Questionnaire 1 *Write the lines of code which would draw the shape shown below.*

```
public class Tortoisetry
import Tortoise.*;
import java.awt.*;
{
  public static void main (String [] args)
  {
     ..... ..... ... ...
     ..... .....
```



This requires the repeated use of the shapes. This time the given result is deliberately made to look like an enlarged cross. Superficially it appears to be the same as the previous question and a student has to look carefully to see how to produce the shape. It can be completed using four squares which are partially superimposed. (The same result can be attained by freehand drawing or using a shape and freehand). This is a more difficult composite shape to produce and might well have been discussed by the students among themselves.

Questionnaire 2 The same question was once again used.

Expected Outcome More students should use only shapes the second time around. Even if the same solution appeared for several students it is an example of the type of cooperative instruction which may take place in a class of students.

Question 8. This question is quite difficult and requires a high degree of conceptualization on the part of the student.

Questionnaire 1 *Ted the tortoise has a friend called Betty. Betty isn't a* ***Tortoise****, she is a* ***ScreenBug****. She can't draw trails or move or turn but she can do lots of good stuff like putting messages on the screen and she can do some mathematics like adding and subtracting.*

Any ScreenBug has several methods at their disposal two of them are: "add(a,b)" and "show()".

add(a,b) - adds any two numbers. (The first number is "a" and the second number is "b")
show( ) - displays anything that is inside the brackets on to the screen.

Write a short program which creates a ***ScreenBug*** *called* ***Betty*** *, adds two numbers and shows the answer on the screen.*

```
public class ScreenBugtry

import java.awt.*;
{
  public static void main (String [] args)
  {
  }
```

The final question is much more formal and requires the application of more difficult concepts. The question suggests a class and methods associated with it. A return type has been deliberately introduced to see if students might suggest the correct syntax **s = add(a,b);**

Questionnaire 2 The same question was once again used. Students would have had some experience to assist them to complete this question after the period of instruction. They may also have discussed this because it was the last most difficult question in the first questionnaire.

Expected Outcome More students should make some attempt at this the second time around.

Questions 9 –13
When you started programming what was the easiest thing to do?
When you were programming what was the most difficult thing to do?
What was the thing that you found to be the most strange? (You can give more than one example)
What did the tortoise do the best?
What else should tortoises be able to do?

Information questions.

Questionnaire 1. These questions were included to seek some feedback on the students' experience and also to try to gain some insight into what students' views were on programming in general.

Questionnaire 2.
When you were programming what did you enjoy doing?
What did you learn from programming?

Same motivation.

3.2 General Observations

In some of the classes involved in the study a number of students repeatedly come late to class, forget work and have to be continually reminded to commence and continue working at almost every activity. They tend to be disruptive occasionally and require major effort on any teacher's part to get them to focus and to stop them distracting the other students who need to maintain their concentration. The experience of introducing the tortoise to them was quite surprising. Almost all of the students enjoyed this experience, so much so that the disruptive students' usual captive audience ignored them on this occasion. For this reason the disruptive group felt a little lost at first and struggled to commence work. However, many from this difficult group recovered quickly and then joined in and experimented happily along with the majority. They seemed to have forgotten their usual disruptive behaviour. It is difficult to pinpoint the critical factor which inhibited the expected counterproductive behaviour but the majority of students were eager to experiment with the tortoise and this had a significant effect. This was unusual because previous attempts to stimulate these students and had been unsuccessful on many occasions. The attractiveness of the activity was immediately observable as soon as practical involvement started. Some disruptive students settled down to work happily but others appeared to undergo a short period of isolation. It is often difficult to decide if certain students are just not interested or if their behaviour is disguising a problem. With the removal of their audience these students settled down quickly and tried to do the work. They were forced by the circumstances of the task to act as individuals but many of them were very uncomfortable with this and struggled to sustain their concentration and efforts.

As part of the experiment the task was structured to introduce alternative constructors once the majority of students had become involved in the work. These introduced shortcuts and variety to Tortoise instantiations. The students began to offer questions and comments such as:

"Is there a quicker way to place the tortoise at a different starting place?"
"Can't you say something like **fred.move(10)** to make him move ten places?"
"There's not enough room to draw nice stuff"

The objective was to try to build up activities as quickly but as comfortably as possible. The difficult students, who usually do not cooperate, were unable to perceive certain things even when

they were concentrating on the tasks. Their usual disguising behaviour was gone and they seemed to be trying genuinely to originate their own material but some of these students did not seem to be able to assimilate patterns. The problems observed were as follows:

- Additional "import" statements were placed anywhere and everywhere.
- In starting a new program some students left out the "import" statements. They failed to recognize the errors associated with this, even though they had just met and been helped to overcome the same problems in the previous program.
- When additional methods were introduced these students omitted brackets, semi-colons and made similar minor errors even though the methods in question were situated among a large number of similar methods.
- Many of these students did not appreciate that adding arbitrary spaces caused errors throughout the program. Even when encouraged to look around at other lines of code and previously correct programs for similar situations they struggled to correct programs themselves.
- Some students could not assimilate the new constructors as alternatives. For example, they started by instantiating a tortoise and then changed the colour by invoking a tortoise method. With the additional constructors revealed, these actions could be combined into one operation by invoking a constructor with suitable parameters. Some students seemed to believe that the first method now no longer existed. When asked why they had done what they did, they responded by saying something like "I don't know". This could indicate a number of things: they really did not know, they had just "hacked" the code in, or perhaps they were embarrassed because they had copied it from someone else.

The lessons worked better when not too much information was given to the students. Sufficient information was given to allow students to commence the activity but they were required to experiment. When students asked questions solutions were offered but additional syntax was introduced as part of that solution where possible. When individual students were stimulated enough to ask they seemed stimulated enough to accept the increased syntax. They seemed to ignore any inherent complexity of syntax because of their eagerness to use it. However, when it came to the questionnaire many students appeared to have forgotten the syntax. The immediacy of the feedback on screen and the facility to experiment seemed to be the most important factors. A questionnaire is

possibly not the best way to test their ability to manage a successful program execution. It is likely that some students may eventually succeed in performing the practical task but still fail to get an answer on the questionnaire.

Frustration with the continual compilation errors tended to increase as time went on and the more difficult students tended increasingly to abandon their efforts. The willingness to persist with a problem is something which is missing in many of the difficult students and even in a few of the more able students.

3.3 Test Results

The objective of this study was to extract evidence as to whether or not a change occurred in understanding as a result of the teaching method. The results of the questionnaires for each class were recorded on a spreadsheet with students' answers being rated 1 or 2 as follows (Appendix G, Table 1):

One full class of student questionnaires was marked and the variety of answers considered. A decision was made in each case to classify the student's answer as success or failure. The criteria were reviewed after each subsequent class was marked and the allocation of success or failure was checked.

1 The student appeared to understand the concept and was correct in the method chosen. The solution was a success if it would execute as it stood or with minor corrections. Trivial errors, such as omission of semi-colons, were ignored. If a solution would work but was overly long or inefficient then it was regarded as a success. In a few cases where it was difficult to allocate success or failure the student was recalled and asked to explain his answer.

2 The student did not appear to understand and was incorrect or only partially correct in the method used. The solution was a failure if the code would only execute after extensive correction or if the answer was left blank. If the student partially remembered syntax and omitted critical parameters or provided only a portion of the solution then it was regarded as a failure. In some cases the solutions were discussed with the students. Answers in this

category were generally quite clearly inadequate, for example, keywords were misspelt or wrong or incorrectly used.

The results of the questionnaires were recorded in a spreadsheet comprising of five worksheets (Appendix G, Table 1). One worksheet was allocated for the results of each class and a further worksheet allocated to record the totals of the four classes. The results for each class were summarised in contingency tables (Appendix G, Tables 2 – 6).

The value of χ^2 was calculated using the formula $\chi^2 = (|c - b| - 1)^2 / (c + b)$ (See Figure 50 below). This is known as McNemar's Test of Change (Armitage and Berry, 1994, Conover, 1980, Levin and Serlin, 2000, Siegel and Castellan, 1988). The continuity correction (-1) is used where the sample numbers less than 200 (Armitage and Berry, 1994).

Result	After		
Before	Able	Not Able	Total
Able	a	b	a+b
Not Able	c	d	c+d
Total	a+c	b+d	a+b+c+d

$$\chi^2 = (|c - b| - 1)^2 / (c + b)$$

$$\chi^2 < 3.84 \Rightarrow \text{No Change}$$

Figure 45 Contingency table and Formulae

The expected value of χ^2 at the 95% level with one degree of freedom was 3.84. The null hypothesis, that no change occurred, was rejected for results below this value.

3.4 Results Summary

Grade 9R was a streamed class of students who have a record of academic success in English and Mathematics. Grade 9E is a class of mixed ability but the majority are of average ability. The class also has a few difficult students. Grade 9D is a class of mixed ability but with a significant number of below average students. Grade 9M is an average ability class with a significant number of difficult students.

Original data results are shown in Appendix G, Tables 1 – 6. The summary of results obtained by applying McNemar's formula are shown in Figure 51 below, in the column entitled "κ^2". Significant change is indicated by a "✓" in the columns entitled "Sig". The total number of students who were able to perform the task is also shown as before and after values in the column entitled "ABLE". The arrows: "↗", "↘" and "→" represent an increase, decrease and no change in numbers respectively.

	Q.1			Q.2			Q.3			Q.4		
	κ^2	ABLE	Sig	κ^2	ABLE	Sig	κ^2	ABLE	Sig	κ^2	ABLE	Sig
9R	0.2	24 ↗ 26	×	8.6	11 ↗ 23	✓	3.2	1 ↗ 6	×	10	1 ↗ 13	✓
9E	2.3	16 ↗ 21	×	2.8	8 ↗ 15	×	3.1	7 ↘ 1	×	5.1	1 ↗ 8	✓
9D	0.5	3 ↗ 5	×	2.3	2 ↗ 6	×	1	0 ↗ 3	×	0	1 ↗ 2	×
9M	0	6 ↘ 5	×	4	2 ↗ 9	✓	1.3	0 ↗ 3	×	3	0 ↗ 5	×
Total	2	49 ↗ 57	×	21	23 ↗ 53	✓	0.8	8 ↗ 13	×	21	3 ↗ 28	✓

	Q.5			Q.6			Q.7			Q.8		
	κ^2	ABLE	Sig	κ^2	ABLE	Sig	κ^2	ABLE	Sig	κ^2	ABLE	Sig
9R	8.1	5 ↗15	✓	4.2	5 ↗11	✓	20	3 ↗ 25	✓	0	0 ↗ 1	×
9E	8.1	4 ↗14	✓	1.8	6 ↗11	×	4.9	3 ↗ 11	✓	0	0 ↗ 1	×
9D	2.3	0 ↗ 4	×	2.3	2 ↗ 6	×	4	0 ↗ 6	✓	0	0 ↗ 1	×
9M	0.8	1 ↗ 4	×	1.3	1 ↗ 4	×	3.2	0 ↗ 5	×	0	0 → 0	×
Total	23	10 ↗ 37	✓	13	14 ↗ 32	✓	37	6 ↗ 47	✓	1.3	0 ↗ 3	×

Figure 46 Summary of Chi Squared Rsults

Significance Test

Question 1. exhibited no significant change for any of the individual classes or as a total of all classes.

Question 2. showed significant change for two of the classes (including the academically streamed class) and the combined classes result.

Question 3. exhibited no significant change for any of the classes or as a total.

Question 4. exhibited significant change for two of the classes (including the academically streamed class) and the combined classes result.

Question 5. exhibited significant change for the academically streamed class, Grade 9E and the combined classes result.

Question 6. exhibited significant change for the academically streamed class and the combined classes result.

Question 7. exhibited significant change for three of the classes (excluding the academically below average class) and the combined classes result.

Question 8. exhibited no change for any of the classes or as a total.

Percentages of successful answers

Class Nos	R 28		E 27		D 19		M 27		ALL 101	
Qne	1	2	1	2	1	2	1	2	1	2
Q.1.	85.7%	92.9%	59.3%	77.8%	15.8%	26.3%	22.2%	18.5%	48.5%	56.4%
Q.2.	39.3%	82.1%	29.6%	55.6%	10.5%	31.6%	7.4%	33.3%	22.8%	52.5%
Q.3.	3.6%	21.4%	25.9%	3.7%	0.0%	15.8%	0.0%	11.1%	7.9%	12.9%
Q.4.	3.6%	46.4%	3.7%	29.6%	5.3%	10.5%	0.0%	18.5%	3.0%	27.7%
Q.5.	17.9%	53.6%	14.8%	51.9%	0.0%	21.1%	3.7%	14.8%	9.9%	36.6%
Q.6.	17.9%	39.3%	22.2%	40.7%	10.5%	31.6%	3.7%	14.8%	13.9%	31.7%
Q.7.	10.7%	89.3%	11.1%	40.7%	0.0%	31.6%	0.0%	18.5%	5.9%	46.5%
Q.8.	0.0%	3.6%	0.0%	3.7%	0.0%	5.3%	0.0%	0.0%	0.0%	3.0%

Figure 47 Summary of Success Percentages

Question 1. In the academically streamed class (9R) successful answers increased from 24 out of 28 to 26 out of 28, that is an increase to 92.9% of the students. The other classes recorded percentages of 77.8%, 26.3% and 18.5% (9E, 9D and 9M). Successful answers, in the four classes combined, increased from 49 out of 101 to 57 out of 101. The question was designed as an easy question but only 56.4% of the students completed it successfully

on the second occasion. Similar work to that tested in the question had been done repeatedly in class and it was expected that students would complete the task without difficulty.

Question 2. A similar trend to Question 1 with regard to final percentages was exhibited. The academically streamed class recorded 82.1% success but the percentage overall was only 52.5%. There was a significant change to reach this result.

Question 3. The results were very low in this question with no significant change as a result of teaching. The final success rate was only 12.9%. The academically streamed class had the highest result, 21.9%. This would seem to indicate the possibility that the question was flawed.

Question 4. Results were also low for this question, 27.7%. The academically streamed class recorded 46.4% with a significant change between before and after results.

Question 5. The final result was a low 36.6%. The academically streamed class recorded 53.6%. Grade 9E recorded 51.9% and both of these classes exhibited significant change after teaching.

Question 6. The final result was a low 31.7%. The class with a significant number of less able students (9M) recorded 14.8% while the other classes were similar to the combined classes result.

Question 7. The final result was 46.5%. The academically streamed class recorded 89.3%. The other classes were much lower, 40.7%, 31.6% and 18.5%. but all classes with the exception of the class with a significant number of less able students exhibited a significant change in before and after results.

Question 8. All results were extremely low, less than 6%. There was no significant change in before and after results. This seemed to indicate the possibility that the question was flawed.

The academically streamed class (9R) exhibited significant change in all but three of the questions and gained a higher percentage than any other class in every question with only one exception (Q.6., 9E). The class with a significant number of less able students (9M) had consistently low successful results, 11.1% - 33.3%, with all but one result less than 20%. The class results exhibited significant change in only one question (Q.2.).

Some students did not fill in the questionnaire with any real intention of displaying their knowledge. Because it was "not for marks" they filled it in as quickly as possible and often left blanks or did not bother to make an effort. The effectiveness of the questionnaire is debatable as a method of evaluating what was a practical exercise. The motivated students did better as they became familiar with the work but it was noticeable that only a few of the difficult students offered examples of their work. They began well but were affected significantly by the frequency and number of their errors. As a result they displayed an unwillingness to experiment with anything that they viewed as difficult. By the time it came to complete the questionnaire a number of students were unmotivated to recall what they had done. It would have been interesting to carefully compare the performance of students who produced anomalous results with their performance in school in general.

Conclusion

Shortcomings

The study took place at a difficult time of year in school. Students were undergoing testing for many of their subjects for national exams and completing a variety of projects to fulfill the requirements for their portfolio work. Many students were a little resentful about any form of extra testing.

A questionnaire does not pinpoint the learning areas and associated concepts that were being considered. A thorough testing would require one academic school year with some form of practical test administered to two distinct groups at the beginning of the year when school life is still quiet and then again at the beginning of the fourth semester before the school exam period commences. It would also have been useful to have chosen several individuals of differing academic abilities and followed them closely through the course of the testing period, trying to isolate certain features of learning to program.

Discussion of Results

The results from testing did indicate some degree of learning. There was a significant change in some questions and successful results increased in before and after readings. There was a noticeable difference in the successful readings for the academically streamed class compared to the other classes and there was a noticeable difference between the class with the largest number of less able students and the other classes.

From a teacher's perspective during the classroom activities it was clear that many of the capable students had their curiosity aroused about how programs could be extended to provide more functionality and many of the less motivated students made an uncharacteristic effort to experiment. Moreover, the majority of students in the study enjoyed the project. Feedback from Computer Studies students regarding the assistant classes, which incorporated a simplified syntax, showed evidence of a short term usefulness, which is what was hoped for. There was a danger that the simplification of the syntax might have been counterproductive in that students might come to depend on the

syntax.' However, many students questioned the simplicity of the Please class syntax and suspected that something was being hidden. This does not necessarily invalidate the use of this class. Students still have the opportunity to gradually assimilate the new material and some of the pressure associated with meeting more new concepts was removed. Previous observations have indicated that many students felt that they were being left behind at this stage of learning the language. This period is crucial in learning and anything that can be done to alleviate the stress experienced by some students and give them time to absorb the new information would seem to be beneficial. The nature of the assistant class also served to encourage many students to question how the methods operated. This allowed them time to consider why it was better to use the standard Java syntax in the long term and to look at how they might set up their own programs using an accepted style.

Extensions and Future Work

Having gone through the experience of teaching using the tortoise I am convinced that it has huge potential for introducing the fundamental concepts necessary for success in programming using an object-oriented language.

A questionnaire may well not be the best way to test a practical course of instruction. A practical test administered on the computer may be more pertinent but will require much more time to design and administer.

The Tortoise class itself is due for a major metamorphosis. It may become a "Creature" class which can instantiate a variety of animals on one screen. It will be useful to build in some more resources to the class such as mutator and accessor methods. These can be used to "get" and "set" a variety of attributes such as age, colour, height, weight, achievement points or anything that holds the interest of young students. This will promote the concepts surrounding the scope of variables in Java.

Figure 48 Multiple creatures

The method of display was correspondingly changed so that only one image file is used. (instead of one each for "NORTH.gif", "SOUTH.gif", "EAST.gif" and "WEST.gif"). Movement in different directions involves rotating the image using the AffineTransform class.

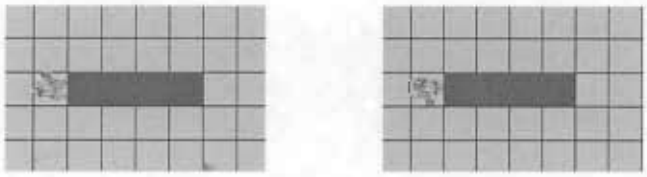


Figure 49 Rotations

Final Word

Teaching is easier using assistant classes. Using a warm, familiar phraseology puts the user more at ease and focuses him on producing working programs. A few users may build up a set of over-simplified building block concepts which are insufficient for anything other than a superficial understanding of the Java syntax but they may never become proficient or even comfortable with the Java language. Students were asked to try to pinpoint when and how they felt that they had become confident in using the language. Most students in this study in a general discussion of this question agreed that the introduction of objects and classes and continual reinforcement by discussion and examples were the most important factors contributing to success. Having time to let the ideas solidify and to struggle with new concepts was important to those who eventually succeeded. Many of the less successful students claimed that they became bored during these phases. They demonstrated unwillingness to engage with the language. The Tortoise was fun at first for them but they did not progress beyond that superficial experience. Some students stated that they felt that making their own class with methods that they created was one crucial milestone. The use of turtle-like classes like the Karel Robot (Becker, 2001, Buck and Stucki, 2001), the Turtle (Braught, 2003, Devarakonda, 2003, Eckert, 2001, Gibbons, 2001, Grieser, 2002, Khalil, 2002, Kono, 2000, Lambert and Osborne, 2003, Oelschlegel, 2003, Proulx and Rasala, 2002, Ventura, 2000, Weissinger et al, 2001, Wolz, 2002), the Gogga (Wilcock et al, 2002) and the Tortoise were useful to get a young student's attention. More serious students (and usually these were older students), who were interested programmers, quickly wanted to move on and create programs which had some originality. They quickly

saw the possibilities of extending the Tortoise class to produce interesting and innovative programs.

In considering both the statistical and anecdotal evidence which emerged from this project my conclusions are that it is difficult to pin down the factors which affect success or failure in the High School. There are an enormous number of factors to consider. Many young students have had, and continue to have, a bad academic experience. Many of them are also affected negatively by their social situation at home and at school. This may always be true but young students have little experience in dealing with these pressures. A few individuals exhibited some surprising changes in behaviour but trying to quantify the results overall was much more difficult. There are some students who succeed at programming and who have a drive to explore and create whereas others stop at various lower levels. One can speed the process of conceptual understanding with the brighter students and alleviate some of the problems experienced by the less able but in the end the individual has got to be prepared to take charge of his own learning. If curiosity and the willingness to be persistent are not present then a student has little chance of becoming a competent programmer.

Appendix A

Program Listing for Tortoise

```
/**
 * @version    1.0, 10/01/2003
 * @author Sinclair Tweedie
 * @modified 0815bc : started
 * @modified 0815bc : s
 */
import java.awt.*;
import java.awt.event.*;
import java.awt.Image;
import javax.swing.event.*;
import java.awt.geom.*;

public class Tortoise

{
   static Grid grid;                    // The SINGLE grid that is instantiated with the first
tortoise
   private Image clipartImage;          // The creature that is being used in the display
   private Image ouchImage;             // Bubble message flashed up when edge of grid is
reached
   private Image messageImage;          // Notice message flashed up when edge of grid is
reached
   String imgName = "EAST.GIF";         // First image
   boolean toroidal = false;            // Wrapping the tortoise around when it hits an edge,
default to no wrap
   char bearing = 'E';                  // Current direction
   int x;                               // X direction
   int y;                               // Y direction
   int age;                             // Age of Tortoise
   boolean penON;                       // Default Show Trail value
   String tlabel = "NIL";               // Label attached to the tortoise

   /**
    * Constructs a tortoise
    * Default image is an East-facing tortoise
    * Direction is East
    */
   public Tortoise ()
   {
        clipartImage = Toolkit.getDefaultToolkit ().getImage (imgName);
        if (grid == null)               // To make new tortoises use the existing grid,
          // instantiated by the first tortoise
```

```
        {
            grid = new Grid ();
        }
    grid.addTortoise (this);
}


/**
 * Constructs a tortoise
 * Default image is an East-facing tortoise
 * Direction is East
 * @param t Boolean value (true) for movement to be toroidal
 */
public Tortoise (boolean t)
{
    clipartImage = Toolkit.getDefaultToolkit ().getImage (imgName);
    toroidal = t;
    if (grid == null)              // To make new tortoises use the existing grid,
      // instantiated by the first tortoise
      {
          grid = new Grid ();
      }
    grid.addTortoise (this);
}


/**
 * Constructs a tortoise
 * Default image is an East-facing tortoise
 * Direction is East
 * @param imgName String representing the Image name and Extension For example
"EAST.GIF"
 */
public Tortoise (String imgName)
{
    clipartImage = Toolkit.getDefaultToolkit ().getImage (imgName);
    if (grid == null)              // To make new tortoises use the existing grid,
      // instantiated by the first tortoise
      {
          grid = new Grid ();
      }
    grid.addTortoise (this);
}


/**
 * Constructs a tortoise
```

```
    * Default image is an East-facing tortoise
    * Direction is East
    * @param scale Integer giving a size validated to the range 10 to 50
    */
   public Tortoise (int scale)
   {

        clipartImage = Toolkit.getDefaultToolkit ().getImage (imgName);
        if (grid == null)                    // To make new tortoises use the existing grid,
          // instantiated by the first tortoise
          {
               grid = new Grid (scale);
          }
        grid.addTortoise (this);
   }


   /**
    * Constructs a tortoise
    * Default image is an East-facing tortoise
    * Direction is East
    * @param scale Integer giving a grid size validated to the range 10 to 50
    * @param n Integer giving the number of rows (and columns) in the grid
    */
   public Tortoise (int scale, int n)
   {
        clipartImage = Toolkit.getDefaultToolkit ().getImage (imgName);
        if (grid == null)                    // To make new tortoises use the existing grid,
          // instantiated by the first tortoise
          {
               grid = new Grid (scale, n);
          }
        grid.addTortoise (this);
   }



   /**
    * Displays the creature being used, default is a Tortoise
    * Also handles "bumping" into the grid edges with two images
    */
public void draw (Graphics g)
   {
     g.drawImage (clipartImage, x, y, grid.gridWidth, grid.gridHeight, grid);
     // Draw a label
     if (tlabel != "NIL")
     {
```

```
        g.setColor (Color.black);
        g.drawString (tlabel, x, y);
        //g.setColor(
      }
      //Check edges
      slowdown (100);
      if (y < 0)
      {
        if ((bearing == 'N') || (bearing == 'n') || (bearing == 'e'))  //North, NorthWest or
NorthEast
        {
          if (x < grid.windowsize - 5 * grid.gridWidth)
          {
          ouchImage = Toolkit.getDefaultToolkit ().getImage ("OUCHNW.GIF");
          g.drawImage (ouchImage, x + grid.gridWidth, 0, 2 * grid.gridWidth, 2 *
grid.gridHeight, grid);
          }
          else
          {
          ouchImage = Toolkit.getDefaultToolkit ().getImage ("OUCHNE.GIF");
          g.drawImage (ouchImage, x - 2 * grid.gridWidth, 0, 2 * grid.gridWidth, 2 *
grid.gridHeight, grid);
          }
          messageImage = Toolkit.getDefaultToolkit ().getImage ("message.GIF");

// Tried this, but message does not fit on small grids or when tortoise is close to edges
// Too many exceptions to cater for and the images worked more effectively in any case
//
//gp.setColor (Color.black);
//gp.drawString("Ted has just banged his head on the edge of the grid", x + 2*gridWidth,
gridWidth);
//gp.setColor (PathColour);

          if (x > 3 * grid.gridWidth)
          {
            g.drawImage (messageImage, 0, 0, 3 * grid.gridWidth, 3 * grid.gridHeight,
grid);
          }
          else
          {
            g.drawImage (messageImage, grid.windowsize - 5 * grid.gridWidth, 0, 3 *
grid.gridWidth, 3 * grid.gridHeight, grid);
          }

          slowdown (200);
        }
        y = 0;
```

```
        // At present the tortoise carries on when it hits an edge - see report for rationale
      }

      if (x < grid.gridWidth)
      {

        if ((bearing == 'W') || (bearing == 'w') || (bearing == 'n')) //West, SouthWest or
NorthWest
        {
          if ( y < grid.windowsize - 5* grid.gridWidth)
          {
          ouchImage = Toolkit.getDefaultToolkit ().getImage ("OUCHNW.GIF");
          g.drawImage (ouchImage,grid.gridHeight , y + grid.gridHeight, 2 *
grid.gridWidth, 2 * grid.gridHeight, grid);
          }
          else
          {
          ouchImage = Toolkit.getDefaultToolkit ().getImage ("OUCHSW.GIF");
          g.drawImage (ouchImage, grid.gridHeight, y - grid.gridHeight, 2 *
grid.gridWidth, 2 * grid.gridHeight, grid);
          }
          messageImage = Toolkit.getDefaultToolkit ().getImage ("message.GIF");
          if (y > 3 * grid.gridWidth)
          {
            g.drawImage (messageImage, 0, 0, 3 * grid.gridWidth, 3 * grid.gridHeight,
grid);
          }
          else
          {
            g.drawImage (messageImage, grid.windowsize - 5 * grid.gridWidth, 0, 3 *
grid.gridWidth, 3 * grid.gridHeight, grid);
          }

          slowdown (200);
        }
        x = grid.gridWidth;
      }

      if (x > grid.windowsize - 4 * grid.imgSize)
      {
        if ((bearing == 'E') || (bearing == 'e')|| (bearing == 's'))  // East, NorthEast or
SouthEast
          {
            if ( y < grid.windowsize - 5 * grid.gridHeight)
            {
            ouchImage = Toolkit.getDefaultToolkit ().getImage ("OUCHNE.GIF");
```

```
            g.drawImage (ouchImage, x - 2 * grid.gridWidth, y + grid.gridHeight, 2 *
grid.gridWidth, 2 * grid.gridHeight, grid);
            }
            else
            {
            ouchImage = Toolkit.getDefaultToolkit ().getImage ("OUCHSE.GIF");
            g.drawImage (ouchImage, x - 2 * grid.gridWidth, y - grid.gridHeight, 2 *
grid.gridWidth, 2 * grid.gridHeight, grid);
            }
            messageImage = Toolkit.getDefaultToolkit ().getImage ("message.GIF");
            if (x > 3 * grid.gridWidth)
            {
              g.drawImage (messageImage, 0, 0, 3 * grid.gridWidth, 3 * grid.gridHeight,
grid);
            }
            else
            {
              g.drawImage (messageImage, grid.windowsize - 5 * grid.gridWidth, 0, 3 *
grid.gridWidth, 3 * grid.gridHeight, grid);
            }

            slowdown (200);
          }
        x = grid.windowsize - 3 * grid.imgSize;
      }

      if (y > grid.windowsize - 3 * grid.imgSize)
      {
        if ((bearing == 'S') || (bearing == 's') || (bearing == 'w')) // South, SouthEast or
SouthWest
        {
          if (x < grid.windowsize - 5 * grid.gridWidth)
          {
           ouchImage = Toolkit.getDefaultToolkit ().getImage ("OUCHSW.GIF");
           g.drawImage (ouchImage, x + grid.gridWidth, y - 2 * grid.gridWidth, 2 *
grid.gridWidth, 2 * grid.gridHeight, grid);
          }
          else
          {ouchImage = Toolkit.getDefaultToolkit ().getImage ("OUCHSE.GIF");
           g.drawImage (ouchImage, x - 2 * grid.gridWidth, y - 2 * grid.gridWidth, 2 *
grid.gridWidth, 2 * grid.gridHeight, grid);
          }
          messageImage = Toolkit.getDefaultToolkit ().getImage ("message.GIF");

          if (x > 3 * grid.gridWidth)
          {
```

```
                g.drawImage (messageImage, 0, 0, 3 * grid.gridWidth, 3 * grid.gridHeight,
grid);
            }
            else
            {
                g.drawImage (messageImage, grid.windowsize - 5 * grid.gridWidth, 0, 3 *
grid.gridWidth, 3 * grid.gridHeight, grid);
            }
            slowdown (200);
        }
        y = grid.windowsize - 3 * grid.imgSize;
    }
  } // draw method

  /**
  * Sets the age of the Tortoise
  */
  public void AgeIs (int data)
  {
      age = data;
  }


  /**
  * Gets the age of the Tortoise
  */
  public int getAge ()
  {
      return age;
  }


  /**
  * Displays Tortoise's age
  */
  public void showInfo ()
  {
      tlabel = "I am a " + age + " year old " + this.getClass ().getName ();
  }


  /**
  * Sets the existing tortoise track colour
  * @param c The colour of the path
  */
  public void setPathColour (Color c)
  {
```

```
        grid.setPathColour (c);
    }


    /**
    * Clears the screen but leaves the turtle at present position
    */
    public void cleanup ()
    {
        grid.cleanup ();
        penON = false;
        //bearing = 'E';
    }


    /**
    * Sets the x and y coordinate positions on the grid
    */
    public void goTo (int xcoord, int ycoord)
    {
        x = xcoord * grid.gridWidth;
        y = ycoord * grid.gridHeight;
    }


    /**
    *  turns 90 degrees clockwise
    */
    public void turn ()
    {
        // Future development : tx = new AffineTransform();
        // Future development : double radians;

        switch (bearing)
        {
          case 'N':     // was North, now clockwise to East
                {
                  bearing = 'E';
                  clipartImage = Toolkit.getDefaultToolkit ().getImage ("EAST.GIF");
                  // Future development : radians = -Math.PI/4;
                  // Future development : tx.rotate(radians);

                  grid.repaint ();
                  break;
                }
          case 'n':     // was NorthEast, now clockwise to SouthEast
                {
```

```
            bearing = 'e';
            clipartImage = Toolkit.getDefaultToolkit ().getImage ("EAST.GIF");
            grid.repaint ();
            break;
        }
case 'w':     // was NorthWest, now clockwise to SouthWest
        {
            bearing = 's';
            clipartImage = Toolkit.getDefaultToolkit ().getImage ("WEST.GIF");
            grid.repaint ();
            break;
        }
case 'S':     // was South, now clockwise to West
        {
            bearing = 'W';
            clipartImage = Toolkit.getDefaultToolkit ().getImage ("WEST.GIF");
            grid.repaint ();
            break;
        }
case 'e':     // was SouthEast, now clockwise to SouthWest
        {
            bearing = 's';
            clipartImage = Toolkit.getDefaultToolkit ().getImage ("WEST.GIF");
            grid.repaint ();
            break;
        }
case 's':     // was SouthWest, now clockwise to NorthWest
        {
            bearing = 'w';
            clipartImage = Toolkit.getDefaultToolkit ().getImage ("WEST.GIF");
            grid.repaint ();
            break;
        }
case 'E':     // was East, now clockwise to South
        {
            bearing = 'S';
            clipartImage = Toolkit.getDefaultToolkit ().getImage ("SOUTH.GIF");
            grid.repaint ();
            break;
        }
case 'W':   // was West, now clockwise to North
        {
            bearing = 'N';
            clipartImage = Toolkit.getDefaultToolkit ().getImage ("NORTH.GIF");
            grid.repaint ();
            break;
        }
```

```
        }
        Toolkit.getDefaultToolkit ().sync ();
    }


    /**
    * Turns the image to face upwards
    */
    public void faceNorth ()
    {
        bearing = 'N';
        clipartImage = Toolkit.getDefaultToolkit ().getImage ("NORTH.GIF");
        grid.repaint ();
    }


    /**
     * Turns the image to face to the right
     */
    public void faceNorthEast ()
    {
        bearing = 'n';
        clipartImage = Toolkit.getDefaultToolkit ().getImage ("EAST.GIF");
        grid.repaint ();
    }


    /**
     * Turns the image to face to the left
     */
    public void faceNorthWest ()
    {
        bearing = 'w';
        clipartImage = Toolkit.getDefaultToolkit ().getImage ("WEST.GIF");
        grid.repaint ();
    }


    /**
    * Turns the image to face downwards
    */
    public void faceSouth ()
    {
        bearing = 'S';
        clipartImage = Toolkit.getDefaultToolkit ().getImage ("SOUTH.GIF");
        grid.repaint ();
    }
```

```
/**
 * Turns the image to face to the right
 */
public void faceSouthEast ()
{
      bearing = 'e';
      clipartImage = Toolkit.getDefaultToolkit ().getImage ("EAST.GIF");
      grid.repaint ();
}


/**
* Turns the image to face downwards and left
*/
public void faceSouthWest ()
{
      bearing = 's';
      clipartImage = Toolkit.getDefaultToolkit ().getImage ("WEST.GIF");
      grid.repaint ();
}


/**
* Turns the image to face to the right
*/
public void faceEast ()
{
      bearing = 'E';
      clipartImage = Toolkit.getDefaultToolkit ().getImage ("EAST.GIF");
      grid.repaint ();
}


/**
* Turns the image to face to the left
*/
public void faceWest ()
{
      bearing = 'W';
      clipartImage = Toolkit.getDefaultToolkit ().getImage ("WEST.GIF");
      grid.repaint ();
}


/**
```

```
   * Moves the Image in the current direction
   */
   public void move ()
   {
        slowdown (grid.speedvalue);
        if (penON)
        {
          grid.showRect [x] [y] = true;
        }
        switch (bearing)
        {
                //North
          case 'N':
                {
                  y = (y - grid.imgSize) % (grid.windowsize - 10);
                  if (toroidal)          // This is the condition for a toroidal grid - the
tortoise wraps around
                  {
                        if (y < 0)
                        {
                          y = grid.windowsize - 3 * grid.imgSize;
                        }
                  }
                  grid.repaint ();
                  break;
                }

                //Northeast
          case 'n':
                {
                  y = (y - grid.imgSize) % (grid.windowsize - 10);
                  x = (x + grid.imgSize) % (grid.windowsize - 10);
                  if (toroidal)
                  {
                        if (y < 0)
                        {
                          y = grid.windowsize - 3 * grid.imgSize;
                        }
                  }

                  grid.repaint ();
                  break;
                }

                //Southeast
          case 'e':
                {
```

```
            y = (y + grid.imgSize) % (grid.windowsize - 10);
            x = (x + grid.imgSize) % (grid.windowsize - 10);
            if (y < 0)
            {
                if (toroidal)
                {
                    y = grid.windowsize - 3 * grid.imgSize;
                }
            }
            grid.repaint ();
            break;
        }

        // South
    case 'S':
        {
            y = (y + grid.imgSize) % (grid.windowsize - 10);
            if (y > grid.windowsize - 3 * grid.imgSize)
            {
                if (toroidal)
                {
                    y = 0;
                }
            }
            grid.repaint ();
            break;
        }

        // Southwest
    case 's':
        {
            y = (y + grid.imgSize) % (grid.windowsize - 10);
            x = (x - grid.imgSize) % (grid.windowsize - 10);

            if (y > grid.windowsize - 3 * grid.imgSize)
            {
                if (toroidal)
                {
                    y = 0;
                }
            }
            grid.repaint ();
            break;
        }

        //Northwest
    case 'w':
```

```
                {
                    y = (y - grid.imgSize) % (grid.windowsize - 10);
                    x = (x - grid.imgSize) % (grid.windowsize - 10);
                    if (toroidal)
                    {
                        if (y < 0)
                        {
                            y = grid.windowsize - 3 * grid.imgSize;
                        }
                    }
                    grid.repaint ();
                    break;
                }
                //East
            case 'E':
                {
                    x = (x + grid.imgSize) % (grid.windowsize - 10);

                    if (x > grid.windowsize - 3 * grid.imgSize)
                    {
                        if (toroidal)
                        {
                            x = 0;
                        }
                    }
                    grid.repaint ();
                    break;
                }
                //West
            case 'W':
                {
                    x = (x - grid.imgSize) % (grid.windowsize - 10);
                    if (x < 0)
                    {
                        if (toroidal)
                        {
                            x = x + grid.windowsize - 2 * grid.imgSize;
                        }
                    }
                    grid.repaint ();
                    break;
                }
        }
        Toolkit.getDefaultToolkit ().sync ();
    }
```

```
    /**
    * Attachs a label which moves with the image
    * @param String which displays a message attached to the tortoise
    */
    public void says (String s)
    {
        tlabel = s;
    }


    /**
    * Flags the trail to ON
    */
    public void pathOn ()
    {
        penON = true;
    }


    /**
    * Flags the trail to OFF
    */
    public void pathOff ()
    {
        penON = false;
    }


    /**
     * Routine to cause a delay
     */
    private void slowdown (int msecs)
    {
        try
        {
          Thread.sleep (msecs);
        }
        catch (InterruptedException e)
        {
        }
    }
}
```

Program Listing for Grid

```
/**
 * @version    1.0, 10/01/2003
 * @author Sinclair Tweedie
 * @modified 0815bc : started
 * @modified 0815bc : s
 */

// "imports" itemized on advice of Tom West, keynote speaker at the "Programming
Symposium"
// Conference at St. John's College Johannesburg, September 16-18 2003
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Image;
import java.awt.Panel;
import java.awt.Toolkit;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.swing.BorderFactory;
import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.JSlider;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

public class Grid extends JComponent
{
   int windowsize = 300;                    // Default window size
   int imgSize = 20;                       // Display Area
   int speedvalue = 300;                    // Initial "slowdown" factor
   int gridWidth = 20;                      // Grid Column width within display area
   int gridHeight = 20;                     // Grid Row width within display area
   int rowsAndCols = 12;                    // Size of the square grid
   Color bkGrndColour = Color.pink;               // Background Coour
   Color pathColour = Color.red;                 // Tortoise trail colour
   Color gridColour = Color.black;               // Gridlines colour
   boolean [] [] showRect = new boolean [750] [750];      // Number of rectangles for
filling
   Tortoise [] tortoises;                  // For several instances of Tortoise on one grid
   int numTortoises;                       // Number of active instances of Tortoise

   /**
    * Constructs a grid
    * Default is size 20 with 12 rows and columns
    */
```

```
    public Grid ()
    {
        JFrame frame = new JFrame ("Ted the Tortoise");

        Panel content = new Panel (new BorderLayout ());

        content.add (this, BorderLayout.CENTER);

        JSlider slider = new JSlider ();              //Alternative :
JSlider(JSlider.VERTICAL, 0, 30, 15);
        slider.setBorder (BorderFactory.createTitledBorder ("Alter Tortoise Speed by
moving the Slider"));
        slider.addChangeListener (new SliderListener ());
        content.add (slider, BorderLayout.NORTH);

        frame.setSize ((windowsize - (imgSize * 2 - 10)), (windowsize - (imgSize * 2 -
70)));

        frame.setLocation (10, 10);                   // Location on Screen

        frame.addWindowListener (new WindowAdapter ()
        {
          public void windowClosing (WindowEvent e)
          {
              System.exit (0);
          }
        }
        );

        cleanup ();
        frame.setContentPane (content);
        frame.setVisible (true);

        // Maybe change this to a Vector in future versions
        tortoises = new Tortoise [100];               // 100 should be more than enough

    } // Grid constructor


    /**
     * Constructs a grid with 12 rows but scaled up or down
     * @param scale ranges from 10 to 50 and is validated within that range
     */
    public Grid (int scale)
    {
        JFrame frame = new JFrame ("Ted the Tortoise");
```

```
        Panel content = new Panel (new BorderLayout ());

        content.add (this, BorderLayout.CENTER);

        JSlider slider = new JSlider ();       //Alternative : JSlider(JSlider.VERTICAL, 0,
30, 15);
        slider.setBorder (BorderFactory.createTitledBorder ("Alter Tortoise Speed by
moving the Slider"));
        slider.addChangeListener (new SliderListener ());
        content.add (slider, BorderLayout.NORTH);
        if (scale < 10)                    // Force to 10-50 range, otherwise too big or too small
        {
          scale = 10;
        }
        if (scale > 50)
        {
          scale = 50;
        }
        windowsize = scale * 15;              // Display Area
        gridWidth = scale;                  // Grid Column width within display area
        gridHeight = scale;                 // Grid Column height within display area
        imgSize = scale;                    // Matching image size to fit within on grid square

        frame.setSize ((windowsize - (imgSize * 2 - 10)), (windowsize - (imgSize * 2 -
70)));

        frame.setLocation (10, 10);          // Location on Screen

        frame.addWindowListener (new WindowAdapter ()
        {
          public void windowClosing (WindowEvent e)
          {
              System.exit (0);
          }
        }
        );

        cleanup ();
        frame.setContentPane (content);
        frame.setVisible (true);

        tortoises = new Tortoise [100];
    } // Grid constructor with scale


    /**
     * Constructs a grid with specified number of rows
```

```
    * and the display size is also scaled up or down
    * @param n the number of rows
    * @param scale ranges from 10 to 50 and is validated within that range
    */
   public Grid (int scale, int n)
   {
        JFrame frame = new JFrame ("Ted the Tortoise");

        Panel content = new Panel (new BorderLayout ());

        content.add (this, BorderLayout.CENTER);

        JSlider slider = new JSlider ();       //Alternative : JSlider(JSlider.VERTICAL, 0,
30, 15);
        slider.setBorder (BorderFactory.createTitledBorder ("Alter Tortoise Speed by
moving the Slider"));
        slider.addChangeListener (new SliderListener ());
        content.add (slider, BorderLayout.NORTH);
        if (scale < 10)
        {
          imgSize = 10;
        }
        if (scale > 50)
        {
          imgSize = 50;
        }

        // n is left unvalidated to see what students will try
        windowsize = scale * 15;              // Display Area
        gridWidth = (int) (windowsize / (n + 2)); // Grid Column width within display area
        gridHeight = (int) (windowsize / (n + 2)); // Grid Column height within display area
        imgSize = (int) (windowsize / (n + 2));   // Matching image size
        rowsAndCols = n - 1;

        frame.setSize ((windowsize - (imgSize * 2 - 10)), (windowsize - (imgSize * 2 -
70)));

        frame.setLocation (10, 10);           // Location on Screen

        frame.addWindowListener (new WindowAdapter ()
        {
          public void windowClosing (WindowEvent e)
          {
               System.exit (0);
          }
        }
        );
```

```
        cleanup ();
        frame.setContentPane (content);
        frame.setVisible (true);

        tortoises = new Tortoise [100];
    } // Grid constructor with scale and row/column setting


    /**
     * Paints the grid lines and fills the squares
     */

    public void paint (Graphics g)
    {
        // Graphics2D has improved functionality overall so may as well use it
        // Will have to in any case in later versions to do rotations of images
        Graphics2D gp = (Graphics2D) g;

        // Fill window
        gp.setColor (bkGrndColour);
        gp.fillRect (0, 0, windowsize, windowsize);

        //Draw the gridlines
        gp.setColor (gridColour);
        for (int lineX = 0 ; lineX <= (rowsAndCols + 1) * gridWidth ; lineX += gridWidth)
        {
            gp.drawLine (0, lineX, (rowsAndCols + 1) * gridWidth, lineX);
        }

        for (int lineY = 0 ; lineY <= (rowsAndCols + 1) * gridHeight ; lineY +=
gridHeight)
        {
            gp.drawLine (lineY, 0, lineY, (rowsAndCols + 1) * gridHeight);

        }
        Toolkit.getDefaultToolkit ().sync ();

        // Draw the Tortoise trail
        gp.setColor (pathColour);
        for (int paintx = 0 ; paintx < windowsize ; paintx += gridWidth)
        {
            for (int painty = 0 ; painty < windowsize ; painty += gridHeight)
            {
                if (showRect [painty] [paintx])
                {
                    gp.fillRect (painty, paintx, gridWidth, gridHeight);
```

```
                }
            }
        }
        Toolkit.getDefaultToolkit ().sync ();

        // And for more than one instance of Tortoise
        for (int count = 0 ; count < numTortoises ; count++)
        {
            tortoises [count].draw (g);
        }
    } // paint method


    /**
    * Adds an instance of Tortoise to the array
    * Increments the number of active instances
    * @param Tortoise instance
    */
    public void addTortoise (Tortoise t)
    {
        tortoises [numTortoises] = t;
        numTortoises++;
    } // addTortoise method


    /**
    * Changes the path colour
    */
    public void setPathColour (Color c)
    {
        pathColour = c;
    } // setPathColour method


    /**
    * Sets the grid squares to be blanked on repainting
    */
    public void cleanup ()
    {
        for (int x = 0 ; x < windowsize ; x += gridWidth)
        {
            for (int y = 0 ; y < windowsize ; y += gridHeight)
            {
                showRect [y] [x] = false;
            }
        }
        repaint ();
```

```
    } // cleanup method


    /**
    * Slider to change the speed
    */
    class SliderListener implements ChangeListener
    {
        public void stateChanged (ChangeEvent e)
        {
          JSlider source = (JSlider) e.getSource ();
          if (!source.getValueIsAdjusting ())
          {
              speedvalue = 1000 - (int) source.getValue () * 10;
          }
        }
    }
} // Grid class
```

Appendix B

Javadoc file for Tortoise

Class Tree Deprecated Index Help

PREV CLASS NEXT CLASS
SUMMARY: INNER | FIELD | CONSTR METHOD

FRAMES NO FRAMES
DETAIL: FIELD | CONSTR | METHOD

Class Tortoise

java.lang.Object
|
+-Tortoise

public class Tortoise
extends java.lang.Object

Constructor Summary
Tortoise() Constructs a tortoise Default image is an East-facing tortoise Direction is East
Tortoise(boolean t) Constructs a tortoise Default image is an East-facing tortoise Direction is East
Tortoise(int scale) Constructs a tortoise Default image is an East-facing tortoise Direction is East
Tortoise(int scale, int n) Constructs a tortoise Default image is an East-facing tortoise Direction is East
Tortoise(java.lang.String imgName) Constructs a tortoise Default image is an East-facing tortoise Direction is East

Method Summary	
void	Ages(int data) Sets the age of the Tortoise
void	cleanup() Clears the screen but leaves the turtle at present position
void	draw(java.awt.Graphics g) Displays the creature being used, default is a Tortoise Also handles "bumping" into the grid edges with two images

void	**faceEast**() Turns the image to face to the right
void	**faceNorth**() Turns the image to face upwards
void	**faceNorthEast**() Turns the image to face to the right
void	**faceNorthWest**() Turns the image to face to the left
void	**faceSouth**() Turns the image to face downwards
void	**faceSouthEast**() Turns the image to face to the right
void	**faceSouthWest**() Turns the image to face downwards and left
void	**faceWest**() Turns the image to face to the left
int	**getAge**() Gets the age of the Tortoise
void	**goTo**(int xcoord, int ycoord) Sets the x and y coordinate positions on the grid
void	**move**() Moves the Image in the current direction
void	**pathOff**() Flags the trail to OFF
void	**pathOn**() Flags the trail to ON
void	**says**(java.lang.String s) Attachs a label which moves with the image
void	**setPathColour**(java.awt.Color c) Sets the existing tortoise track colour
void	**showInfo**() Displays Tortoise's age
void	**turn**() turns 90 degrees clockwise

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Tortoise

public **Tortoise**()

Constructs a tortoise Default image is an East-facing tortoise Direction is East

---

Tortoise

public **Tortoise**(boolean t)

Constructs a tortoise Default image is an East-facing tortoise Direction is East

Parameters:

t - Boolean value (true) for movement to be toroidal

Tortoise

public **Tortoise**(java.lang.String imgName)

Constructs a tortoise Default image is an East-facing tortoise Direction is East

Parameters:

imgName - String representing the Image name and Extension For example "EAST.GIF"

Tortoise

public **Tortoise**(int scale)

Constructs a tortoise Default image is an East-facing tortoise Direction is East

Parameters:

scale - Integer giving a size validated to the range 10 to 50

Tortoise

public **Tortoise**(int scale,
int n)

Constructs a tortoise Default image is an East-facing tortoise Direction is East

Parameters:

scale - Integer giving a grid size validated to the range 10 to 50

n - Integer giving the number of rows (and columns) in the grid

Method Detail

draw

public void **draw**(java.awt.Graphics g)

Displays the creature being used, default is a Tortoise Also handles "bumping" into the grid edges with two images

---

AgeIs

public void **AgeIs**(int data)

Sets the age of the Tortoise

---

getAge

public int **getAge**()

Gets the age of the Tortoise

---

showInfo

public void **showInfo**()

Displays Tortoise's age

---

setPathColour

public void **setPathColour**(java.awt.Color c)

Sets the existing tortoise track colour

Parameters:

c - The colour of the path

---

cleanup

public void **cleanup**()

Clears the screen but leaves the turtle at present position

---

goTo

public void **goTo**(int xcoord,

int ycoord)

Sets the x and y coordinate positions on the grid

---

turn

public void **turn**()

turns 90 degrees clockwise

---

faceNorth

public void **faceNorth()**

Turns the image to face upwards

---

faceNorthEast

public void **faceNorthEast()**

Turns the image to face to the right

---

faceNorthWest

public void **faceNorthWest()**

Turns the image to face to the left

---

faceSouth

public void **faceSouth()**

Turns the image to face downwards

---

faceSouthEast

public void **faceSouthEast()**

Turns the image to face to the right

---

faceSouthWest

public void **faceSouthWest()**

Turns the image to face downwards and left

---

faceEast

public void **faceEast()**

Turns the image to face to the right

---

faceWest

public void **faceWest()**

Turns the image to face to the left

---

move

public void **move**()

Moves the Image in the current direction

---

says

public void **says**(java.lang.String s)

Attachs a label which moves with the image

Parameters:

String - which displays a message attached to the tortoise

---

pathOn

public void **pathOn**()

Flags the trail to ON

---

pathOff

public void **pathOff**()

Flags the trail to OFF

Javadoc file for Grid

Class Tree Deprecated Index Help

PREV CLASS NEXT CLASS
SUMMARY: INNER | FIELD | CONSTR | METHOD

FRAMES NO FRAMES
DETAIL: FIELD | CONSTR | METHOD

Class Grid

java.lang.Object
|
+-java.awt.Component
|
+-java.awt.Container
|
+-javax.swing.JComponent
|
+-Grid

All Implemented Interfaces:
java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable

public class **Grid**
extends javax.swing.JComponent
See Also:
Serialized Form

Inner classes inherited from class javax.swing.JComponent
javax.swing.JComponent.AccessibleJComponent

Inner classes inherited from class java.awt.Container
java.awt.Container.AccessibleAWTContainer

Inner classes inherited from class java.awt.Component
java.awt.Component.AccessibleAWTComponent

Fields inherited from class javax.swing.JComponent
accessibleContext, listenerList, TOOL_TIP_TEXT_KEY, ui, UNDEFINED_CONDITION, WHEN_ANCESTOR_OF_FOCUSED_COMPONENT, WHEN_FOCUSED, WHEN_IN_FOCUSED_WINDOW

Fields inherited from class java.awt.Component
BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT

Fields inherited from interface java.awt.image.ImageObserver
ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH

Constructor Summary
Grid() Constructs a grid Default is size 20 with 12 rows and columns
Grid(int scale) Constructs a grid with 12 rows but scaled up or down
Grid(int scale, int n) Constructs a grid with specified number of rows and the display size is also scaled up or down

Method Summary	
void	**addTortoise**(Tortoise t) Adds an instance of Tortoise to the array Increments the number of active instances
void	**cleanup**() Sets the grid squares to be blanked on repainting
void	**paint**(java.awt.Graphics g) Paints the grid lines and fills the squares
void	**setPathColour**(java.awt.Color c) Changes the path colour

Methods inherited from class javax.swing.JComponent
addAncestorListener, addNotify, addPropertyChangeListener, addPropertyChangeListener, addVetoableChangeListener, computeVisibleRect, contains, createToolTip, disable, enable, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, firePropertyChange, fireVetoableChange, getAccessibleContext, getActionForKeyStroke, getActionMap, getAlignmentX, getAlignmentY, getAutoscrolls, getBorder, getBounds, getClientProperty, getComponentGraphics, getConditionForKeyStroke, getDebugGraphicsOptions,

getGraphics, getHeight, getInputMap, getInputMap, getInputVerifier, getInsets, getInsets, getListeners, getLocation, getMaximumSize, getMinimumSize, getNextFocusableComponent, getPreferredSize, getRegisteredKeyStrokes, getRootPane, getSize, getToolTipLocation, getToolTipText, getToolTipText, getTopLevelAncestor, getUIClassID, getVerifyInputWhenFocusTarget, getVisibleRect, getWidth, getX, getY, grabFocus, hasFocus, hide, isDoubleBuffered, isFocusCycleRoot, isFocusTraversable, isLightweightComponent, isManagingFocus, isMaximumSizeSet, isMinimumSizeSet, isOpaque, isOptimizedDrawingEnabled, isPaintingTile, isPreferredSizeSet, isRequestFocusEnabled, isValidateRoot, paintBorder, paintChildren, paintComponent, paintImmediately, paintImmediately, paramString, print, printAll, printBorder, printChildren, printComponent, processComponentKeyEvent, processFocusEvent, processKeyBinding, processKeyEvent, processMouseMotionEvent, putClientProperty, registerKeyboardAction, registerKeyboardAction, removeAncestorListener, removeNotify, removePropertyChangeListener, removePropertyChangeListener, removeVetoableChangeListener, repaint, repaint, requestDefaultFocus, requestFocus, resetKeyboardActions, reshape, revalidate, scrollRectToVisible, setActionMap, setAlignmentX, setAlignmentY, setAutoscrolls, setBackground, setBorder, setDebugGraphicsOptions, setDoubleBuffered, setEnabled, setFont, setForeground, setInputMap, setInputVerifier, setMaximumSize, setMinimumSize, setNextFocusableComponent, setOpaque, setPreferredSize, setRequestFocusEnabled, setToolTipText, setUI, setVerifyInputWhenFocusTarget, setVisible, unregisterKeyboardAction, update, updateUI

Methods inherited from class java.awt.Container

add, add, add, add, add, addContainerListener, addImpl, countComponents, deliverEvent, doLayout, findComponentAt, findComponentAt, getComponent, getComponentAt, getComponentAt, getComponentCount, getComponents, getLayout, insets, invalidate, isAncestorOf, layout, list, list, locate, minimumSize, paintComponents, preferredSize, printComponents, processContainerEvent, processEvent, remove, remove, removeAll, removeContainerListener, setLayout, validate, validateTree

Methods inherited from class java.awt.Component

action, add, addComponentListener, addFocusListener, addHierarchyBoundsListener, addHierarchyListener, addInputMethodListener, addKeyListener, addMouseListener, addMouseMotionListener, bounds, checkImage, checkImage, coalesceEvents, contains, createImage, createImage, disableEvents, dispatchEvent, enable, enableEvents, enableInputMethods, getBackground, getBounds, getColorModel, getComponentOrientation, getCursor, getDropTarget, getFont, getFontMetrics, getForeground, getGraphicsConfiguration, getInputContext, getInputMethodRequests, getLocale, getLocation, getLocationOnScreen,

getName, getParent, getPeer, getSize, getToolkit, getTreeLock, gotFocus, handleEvent, imageUpdate, inside, isDisplayable, isEnabled, isLightweight, isShowing, isValid, isVisible, keyDown, keyUp, list, list, list, location, lostFocus, mouseDown, mouseDrag, mouseEnter, mouseExit, mouseMove, mouseUp, move, nextFocus, paintAll, postEvent, prepareImage, prepareImage, processComponentEvent, processHierarchyBoundsEvent, processHierarchyEvent, processInputMethodEvent, processMouseEvent, remove, removeComponentListener, removeFocusListener, removeHierarchyBoundsListener, removeHierarchyListener, removeInputMethodListener, removeKeyListener, removeMouseListener, removeMouseMotionListener, repaint, repaint, repaint, resize, resize, setBounds, setBounds, setComponentOrientation, setCursor, setDropTarget, setLocale, setLocation, setLocation, setName, setSize, setSize, show, show, size, toString, transferFocus

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

Grid

public Grid()

Constructs a grid Default is size 20 with 12 rows and columns

---

Grid

public Grid(int scale)

Constructs a grid with 12 rows but scaled up or down

Parameters:

scale - ranges from 10 to 50 and is validated within that range

---

Grid

public Grid(int scale,

int n)

Constructs a grid with specified number of rows and the display size is also scaled up or down

Parameters:

n - the number of rows

scale - ranges from 10 to 50 and is validated within that range

Method Detail

paint

public void **paint**(java.awt.Graphics g)

Paints the grid lines and fills the squares

Overrides:

paint in class javax.swing.JComponent

---

addTortoise

public void **addTortoise**(Tortoise t)

Adds an instance of Tortoise to the array Increments the number of active instances

Parameters:

Tortoise - instance

---

setPathColour

public void **setPathColour**(java.awt.Color c)

Changes the path colour

---

cleanup

public void **cleanup**()

Sets the grid squares to be blanked on repainting

Appendix C

Program Listing for the Please Class

```
import java.io.*;
import java.util.*;

public class Please
{
  //private static StringTokenizer ST;
  private static BufferedReader BR =
       new BufferedReader (new InputStreamReader (System.in));
  private static String PadChar = "00000000000000000000";
  /**
   * Displays information on the variety of available types covered by methods in this
class
   */
  public static String helpme ()
  {
       System.out.println ("int    - An integer, range is from " + Integer.MAX_VALUE + "
to " + Integer.MIN_VALUE);
       System.out.println ("short  - An integer, range is from " + Short.MAX_VALUE + "
to " + Short.MIN_VALUE);
       System.out.println ("byte   - An integer, range is from " + Byte.MAX_VALUE + "
to " + Byte.MIN_VALUE);
       System.out.println ("long   - An integer, range is from " + Long.MAX_VALUE + "
to " + Long.MIN_VALUE);
       System.out.println ();
       System.out.println ("double - A decimal, in the range from " +
Double.MAX_VALUE + " to " + Double.MIN_VALUE);
       System.out.println ("float  - A decimal, in the range from " + Float.MAX_VALUE
+ " to " + Float.MIN_VALUE);
       System.out.println ();
       System.out.println ("char   - A Unicode character");
       System.out.println ("String - A bunch of characters such as a word, name, address
or  sentence\n");
       System.out.println ("You can format the output to screen for a decimal (declared as
a double)\n");
       System.out.println ("e.g.  If n = 7.123456789");
       System.out.println ("      then using Please.tidyup(n,2)    n displays as 7.12");
       System.out.println ("      and       Please.tidyup(n,7,2)  n displays as 7.12000\n");
       System.out.println ("You can also use a different padding character\n");
       char q = '"';
       System.out.println ("e.g. Please.ChangePaddingTo(" + q +"#" + q + ") - will use
spaces\n");
```

```
        System.out.println (" and then using Please.tidyup(n,7,2)  n will now display as
7.12###\n");
        return " ";
    }


    /**
     * Class Method to input a String
     */
    public static String getString () throws IOException
    {
        try
        {
            return BR.readLine ().trim ();
        }
        catch (IOException e)
        {
            System.err.println ("Please try again - there was something I didn't understand in
your string");
            return "";

        }
    }


    /**
     * Class Method to input a Character
     */
    public static char getLetter () throws IOException
    {
        try
        {
            String s = BR.readLine ().trim ();
            return s.charAt (0);
        }
        catch (IOException e)
        {
            System.err.println ("Please try again - getLetter only accepts a single character");
            return ' ';

        }
    }


    /**
     * Class Method to input an Integer
     */
```

```
    public static int getNumber () throws IOException
    {
        try
        {
          StringTokenizer ST = new StringTokenizer (BR.readLine ().trim ());
          return Integer.parseInt (ST.nextToken ());
        }
        catch (NumberFormatException nfe)
        {
          System.err.println ("Please try again - getNumber only accepts integers ");
        }
        return 0;
    }


    //public static int getNumber () throws IOException
    //{
    //  try
    //  {
    //    StringTokenizer ST = new StringTokenizer (BR.readLine ().trim ());
    //    return new Integer (ST.nextToken ()).intValue ();
    //  }
    //  catch (NumberFormatException nfe)
    //  {
    //    System.err.println ("Please try again - getNumber only accepts integers");
    //  }
    //  return 0;
    //}

    /**
     * Class Method to input a Long Integer
     */
    public static long getLongInteger () throws IOException
    {
        try
        {
          StringTokenizer ST = new StringTokenizer (BR.readLine ().trim ());
          return Long.parseLong (ST.nextToken ());
        }
        catch (NumberFormatException nfe)
        {
          System.err.println ("Please try again - getNumber only accepts integers of type:
long");
        }
        return 0;
    }
```

```
  //public static long getLongInteger () throws IOException
  //{
  //   try
  //   {
  //      StringTokenizer ST = new StringTokenizer (BR.readLine ().trim ());
  //      return new Long (ST.nextToken ()).longValue ();
  //   }
  //   catch (NumberFormatException nfe)
  //   {
  //      System.err.println ("Please try again - getDecimal only gets numbers of type:
double");
  //   }
  //   return 0;
  //}

  /**
   * Class Method to input a Short integer
   */
  public static short getShortInteger () throws IOException
  {
        try
        {
          StringTokenizer ST = new StringTokenizer (BR.readLine ().trim ());
          return Short.parseShort (ST.nextToken ());
        }
        catch (NumberFormatException nfe)
        {
          System.err.println ("Please try again - getNumber only accepts integers of type:
short");
        }
        return 0;
  }


  //public static long getShortInteger () throws IOException
  //{
  //   try
  //   {
  //      StringTokenizer ST = new StringTokenizer (BR.readLine ().trim ());
  //      return new Short (ST.nextToken ()).shortValue ();
  //   }
  //   catch (NumberFormatException nfe)
  //   {
  //      System.err.println ("Please try again - getDecimal only gets numbers of type:
double");
  //   }
```

```
    //    return 0;
    //}


    /**
     * Class Method to input a Byte
     */
    public static byte getByte () throws IOException
    {
        try
        {
            StringTokenizer ST = new StringTokenizer (BR.readLine ().trim ());
            return Byte.parseByte (ST.nextToken ());
        }
        catch (NumberFormatException nfe)
        {
            System.err.println ("Please try again - getNumber only accepts integers of type:
byte");
        }
        return 0;
    }


    //public static long getByte () throws IOException
    //{
    //    try
    //    {
    //        StringTokenizer ST = new StringTokenizer (BR.readLine ().trim ());
    //        return new Byte (ST.nextToken ()).byteValue ();
    //    }
    //    catch (NumberFormatException nfe)
    //    {
    //        System.err.println ("Please try again - getDecimal only gets numbers of type:
double");
    //    }
    //    return 0;
    //}

    /**
     * Class Method to input a double
     */
    public static double getDecimal () throws IOException
    {
        try
        {
            StringTokenizer ST = new StringTokenizer (BR.readLine ().trim ());
            return Double.parseDouble (ST.nextToken ());
```

```
        }
        catch (NumberFormatException nfe)
        {
          System.err.println ("Please try again - getDecimal only gets numbers of type:
double");
        }
        return 0;
  }


  //public static double getDecimal () throws IOException
  //{
  //   try
  //   {
  //      StringTokenizer ST = new StringTokenizer (BR.readLine ().trim ());
  //      return new Double (ST.nextToken ()).doubleValue ();
  //   }
  //   catch (NumberFormatException nfe)
  //   {
  //      System.err.println ("Please try again - getDecimal only gets numbers of type:
double");
  //   }
  //   return 0;
  //}

  /**
   * Class Method to input a Float
   */
  public static double getFloat () throws IOException
  {
        try
        {
          StringTokenizer ST = new StringTokenizer (BR.readLine ().trim ());
          return Float.parseFloat (ST.nextToken ());
        }
        catch (NumberFormatException nfe)
        {
          System.err.println ("Please try again - getDecimal only gets numbers of type:
double");
        }
        return 0;
  }


  //public static double getFloat () throws IOException
  //{
  //   try
```

```
//  {
//      StringTokenizer ST = new StringTokenizer (BR.readLine ().trim ());
//      return new Float (ST.nextToken ()).floatValue ();
//  }
//  catch (NumberFormatException nfe)
//  {
//      System.err.println ("Please try again - getDecimal only gets numbers of type:
double");
//  }
//  return 0;
//}

/**
 * Class method to open a file for reading
 * @param filename String giving the file specification
 * @return BufferedReader for the specified file
 */
public static BufferedReader openFile (String filename) throws FileNotFoundException
// Public - to allow for external and internal use
{
    return new BufferedReader (new FileReader (filename));
}


/**
 * Class Method to read all of the data from a file
 * @return BufferedReader for the specified file
 */
public static void showAllFromFile (String filename) throws FileNotFoundException,
IOException
{
    BufferedReader infile = openFile (filename);

    while (infile.ready ())
    {
      String data = infile.readLine ();
      System.out.println (data);
    }
    infile.close ();
}


/**
 * Class Method to read all of the data from a file and place it into an array of Strings
 * @param String giving the file specification
 * @return Array of Strings containing the content of the file
 */
```

```
public static String [] getAllFromFile (String filename) throws FileNotFoundException,
IOException
{
    BufferedReader infile = openFile (filename);
    String [] data = new String [100];
    int i = 0;
    while (infile.ready ())
    {
      data [i++] = infile.readLine ();
    }
    return data;
}


/**
 * Class method to create a new file for writing
 */
public static BufferedWriter makeNewFile (String filename) throws IOException
{
    return new BufferedWriter (new FileWriter (filename));
}


/**
 * Class method to format a double to a given number of decimal places
 * @param n The number (as a double) to format
 * @param d The number of decimal places
 * @return The number as a string
 */

public static String tidyup (double n, int d)
{
    int l = Double.toString (n).length ();
    int pt = Double.toString (n).indexOf (".");
    int de = l - pt;
    if (de > d)
    {
      return Double.toString (n).substring (0, Double.toString (n).indexOf (".") + d +
1);
    }
    else
    {
      return Double.toString (n);
    }
}
```

```
    /**
     * Class method to format a double to a given number of decimal places
     * @param n The number (as a double) to format
     * @param l The overall length of the final displayed number
     * @param d The number of decimal places
     * @return The number as a string
     */
    public static String tidyup (double n, int l, int d)
    {
        if (l <= d)
        {
            return Double.toString (n);
        }
        else
        {
            String str = Double.toString (n).substring (0, Double.toString (n).indexOf (".") +
d + 1);

            if (l > str.length ())
            {
                return Double.toString (n).substring (0, Double.toString (n).indexOf (".") +
d + 1) + PadChar.substring (0, l - (Double.toString (n).indexOf (".") + d + 1));
            }
            else
            {
                return str;
            }
        }
    }

    /**
     * Class method to change the padding for a double from the default zeroes to a given
character
     * @param ch The character to use for padding
     */
    public static void ChangePaddingTo (String ch)
    {
        PadChar = ch.substring(0,0);
        for (int i = 0 ; i < 20 ; i++)
        {
            PadChar = PadChar + ch;
        }
    }
}
```

Appendix D

Javadoc files for Please Class

Class Tree Deprecated Index Help

PREV CLASS NEXT CLASS
SUMMARY: INNER | FIELD | CONSTR | METHOD

FRAMES NO FRAMES
DETAIL: FIELD CONSTR METHOD

Class Please

java.lang.Object
|
+-Please

public class **Please**
extends java.lang.Object

Constructor Summary
Please()

Method Summary	
static void	**ChangePaddingTo**(java.lang.String ch) Class method to change the padding for a double from the default zeroes to a given character
static java.lang.String[]	**getAllFromFile**(java.lang.String filename) Class Method to read all of the data from a file and place it into an array of Strings
static byte	**getByte**() Class Method to input a Byte
static double	**getDecimal**() Class Method to input a double
static double	**getFloat**() Class Method to input a Float
static char	**getLetter**() Class Method to input a Character
static long	**getLongInteger**() Class Method to input a Long Integer

static int	**getNumber**() Class Method to input an Integer
static short	**getShortInteger**() Class Method to input a Short integer
static java.lang.String	**getString**() Class Method to input a String
static java.lang.String	**helpme**() Displays information on the variety of available types covered by methods in this class
static java.io.BufferedWriter	**makeNewFile**(java.lang.String filename) Class method to create a new file for writing
static java.io.BufferedReader	**openFile**(java.lang.String filename) Class method to open a file for reading
static void	**showAllFromFile**(java.lang.String filename) Class Method to read all of the data from a file
static java.lang.String	**tidyup**(double n, int d) Class method to format a double to a given number of decimal places
static java.lang.String	**tidyup**(double n, int l, int d) Class method to format a double to a given number of decimal places

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Please

public Please()

Method Detail

helpme

public static java.lang.String **helpme**()

Displays information on the variety of available types covered by methods in this class

getString

```
public static java.lang.String getString()
                    throws java.io.IOException
```

Class Method to input a String

---

getLetter

```
public static char getLetter()
                throws java.io.IOException
```

Class Method to input a Character

---

getNumber

```
public static int getNumber()
                throws java.io.IOException
```

Class Method to input an Integer

---

getLongInteger

```
public static long getLongInteger()
                throws java.io.IOException
```

Class Method to input a Long Integer

---

getShortInteger

```
public static short getShortInteger()
                throws java.io.IOException
```

Class Method to input a Short integer

---

getByte

```
public static byte getByte()
                throws java.io.IOException
```

Class Method to input a Byte

---

getDecimal

```
public static double getDecimal()
                throws java.io.IOException
```

Class Method to input a double

---

getFloat

public static double **getFloat**()
throws java.io.IOException
Class Method to input a Float

---

openFile

public static java.io.BufferedReader **openFile**(java.lang.String filename)
throws java.io.FileNotFoundException
Class method to open a file for reading
Parameters:
filename - String giving the file specification
Returns:
BufferedReader for the specified file

---

showAllFromFile

public static void **showAllFrom**File(java.lang.String filename)
throws java.io.FileNotFoundException,
java.io.IOException
Class Method to read all of the data from a file
Returns:
BufferedReader for the specified file

---

getAllFromFile

public static java.lang.String[] **getAllFromFile**(java.lang.String filename)
throws java.io.FileNotFoundException,
java.io.IOException
Class Method to read all of the data from a file and place it into an array of Strings
Parameters:
String - giving the file specification
Returns:
Array of Strings containing the content of the file

---

makeNewFile

public static java.io.BufferedWriter **makeNewFile**(java.lang.String filename)
throws java.io.IOException
Class method to create a new file for writing

---

tidyup

public static java.lang.String **tidyup**(double n,
int d)

Class method to format a double to a given number of decimal places

Parameters:

n - The number (as a double) to format

d - The number of decimal places

Returns:

The number as a string

---

tidyup

public static java.lang.String **tidyup**(double n,
int l,
int d)

Class method to format a double to a given number of decimal places

Parameters:

n - The number (as a double) to format

l - The overall length of the final displayed number

d - The number of decimal places

Returns:

The number as a string

ChangePaddingTo

public static void **ChangePaddingTo**(java.lang.String ch)

Class method to change the padding for a double from the default zeroes to a given character

Parameters:

ch - The character to use for padding

Class Tree Deprecated Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: INNER | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Appendix E

Questionnaire 1

Questionnaire 1 Grade 9 Class

Q.1. How would you get a tortoise called "*Thabo*" to draw this rectangle on the screen? Just give the movement instructions.

```
public.....
.....)
{.....
{
  ► Put the movement instructions just below HERE
```

Q.2. The following program might not work. It might give errors when we try to run it. If you think it will run correctly then say that you think it is OK. ***If NOT*** then correct it by adding in lines and/or removing lines and/or changing lines.

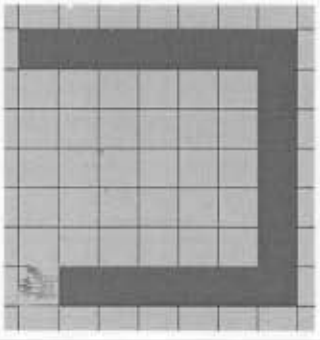
```
public class Tortoisetry
import Tortoise.*;
import java.awt.*;
{
 public static void main (String [] args)
 {

     pathOn ();
     setPosition (1, 1);
     faceEast ();
     move ();
     move ();
     turn();
 }
```

Q.3. The following is **a section** of a program. It allows me to get a tortoise called "Betty" to move to the top left of the screen before she starts drawing a pattern but it is a rather long way to do this. What is the most efficient method to get Betty to get to the starting position?

```
public class Tortoisetry
import Tortoise.*;
import java.awt.*;
{
 public static void main (String [] args)
 {
      .....
      .....
      betty.pathOff ();
      betty.turn ();
      betty.move ();
      betty.move ();
      betty.move ();
      betty.move ();
      betty.move ();
      betty.move ();
      betty.move ();
      betty.move ();
      betty.turn();
      betty.pathOn()
 }
```

Q.4. The following is a **section** of a program which does run properly. It seems like a lot of lines of code to have to type. What is the ***best*** way of doing this? Suggest a method or change the lines of code.

```
.....
ted.setPosition(1,1);
ted.faceEast();
ted.move();ted.move();ted.move();ted.move();
ted.move();ted.move();ted.turn();ted.move();
ted.move();ted.move();ted.move();ted.move();
ted.move();ted.turn();
ted.move();ted.move();
ted.move();ted.move();
ted.move();ted.move();
ted.turn();
.....
```

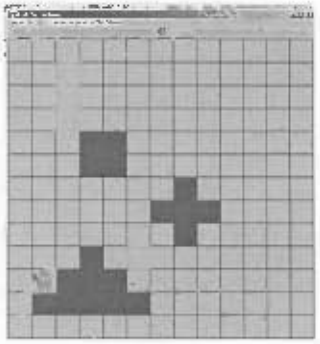


Q.5. A Tortoise has three new methods called makeSquare, makeCross and makeTriangle.

makeSquare draws a solid Square,

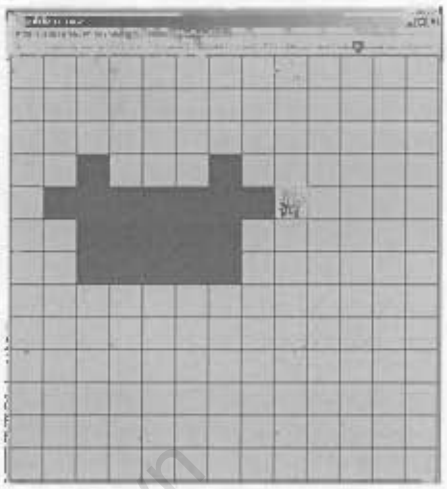
makeCross draws a solid cross and

makeTriangle draws a solid triangle.



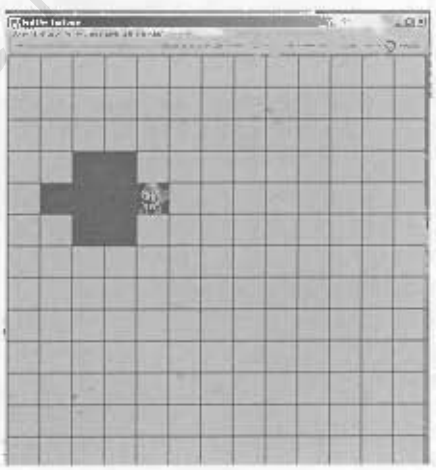
Write the lines of code which would draw the shape shown below.

```
public class Tortoisetry
import Tortoise.*;
import java.awt.*;
{
 public static void main (String [] args)
 {
     ..........
     ..........
```



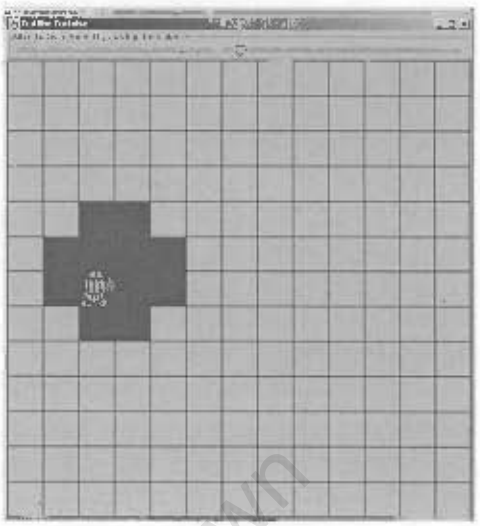
Q.6. Write the lines of code which would draw the shape shown below.

```
public class Tortoisetry
import Tortoise.*;
import java.awt.*;
{
 public static void main (String [] args)
 {
     ..........
     ..........
```



Q.7. Write the lines of code which would draw the shape shown below.

```
public class Tortoisetry
import Tortoise.*;
import java.awt.*;
{
 public static void main (String [] args)
 {
     ..........
     ..........
```



Q.8. Ted the tortoise has a friend called Betty. Betty isn't a **Tortoise**, she is a **ScreenBug**. She can't draw trails or move or turn but she can do lots of good stuff like putting messages on the screen and she can do some mathematics like adding and subtracting.

Any ScreenBug has several methods at their disposal two of them are: "add(a,b)" and "show()".

add(a,b) - adds any two numbers. (The first number is "a" and the second number is "b")
show( ) - displays anything that is inside the brackets on to the screen.

Write a short program which creates a **ScreenBug** called **Betty** , adds two numbers and shows the answer on the screen.

```
public class ScreenBugtry

import java.awt.*;
{
 public static void main (String [] args)
 {
 }
```

Q.9. When you started programming what was the easiest thing to do?

Q.10. When you were programming what was the most difficult thing to do?

Q.11. What was the thing that you found to be the most strange? (You can give more than one example)

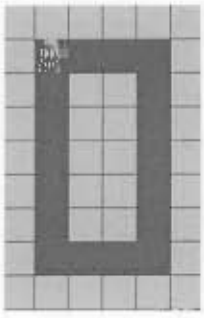
Q.12. What did the tortoise do the best?

Q.13. What else should tortoises be able to do?

Questionnaire 2

Questionnaire 2 Grade 9 Class

Q.1. How would you get the tortoise to draw this rectangle on the screen in the most efficient way?



Q.2. Every Java Tortoise program needs certain lines of code otherwise it doesn't work. It will give errors when we try to run it. What are these vital instructions?

```
public class Tortoisetry
►
►
{
 public static void main (String [] args)
 {
►
 }
```

Q.3. When you create a new tortoise, what options do you have and how do you access them?

```
public class Tortoisetry
import Tortoise.*;
import java.awt.*;
{
 public static void main (String [] args)
 {
► Write the different options below here
```

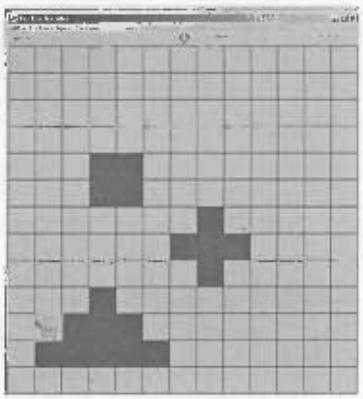
Q.4. What is the best way of writing a program to get a tortoise to repeat some actions over and over again? Comment below and give an example if you think it's suitable. (You don't need to write a whole program)

Q.5. A Tortoise has three new methods called makeSquare, makeCross and makeTriangle.

makeSquare draws a solid Square,

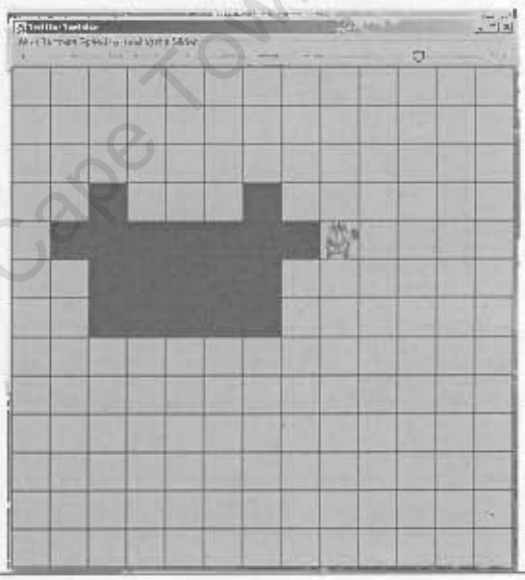
makeCross draws a solid cross and

makeTriangle draws a solid triangle.



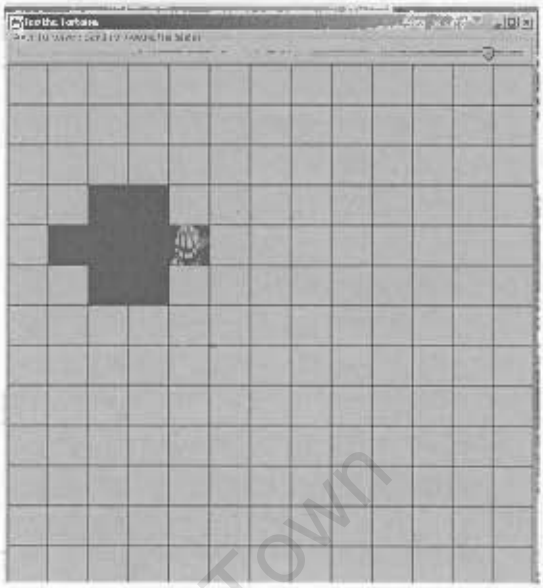
Write the lines of code which would draw the shape shown below.

```
public class Tortoisetry
import Tortoise.*;
import java.awt.*;
{
 public static void main (String [] args)
 {
      ..........
      ..........
```



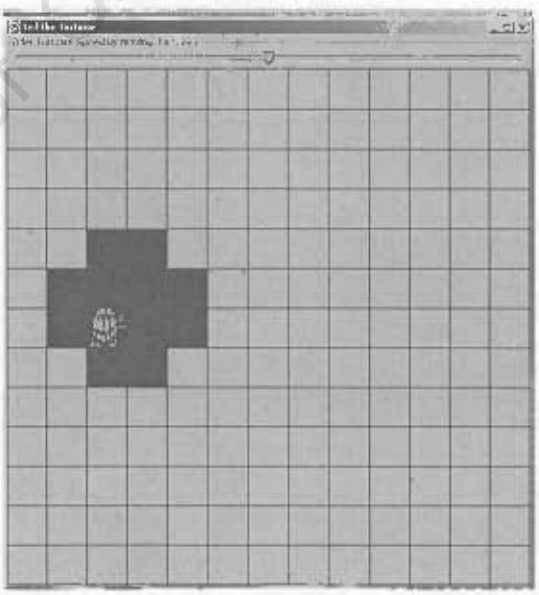
Q.6. Write the lines of code which would draw the shape shown below.

```
public class Tortoisetry
import Tortoise.*;
import java.awt.*;
{
 public static void main (String [] args)
 {
     ..........
     ..........
```



Q.7. Write the lines of code which would draw the shape shown below.

```
public class Tortoisetry
import Tortoise.*;
import java.awt.*;
{
 public static void main (String [] args)
 {
     ..........
     ..........
```



Q.8. Ted the tortoise has a friend called Betty. Betty isn't a **Tortoise**, she is a **ScreenBug**. She can't draw trails or move or turn but she can do lots of good stuff like putting messages on the screen and she can do some mathematics like adding and subtracting.

Any ScreenBug has several methods at their disposal – two of them are: "add(a.b)" and "show()".

add(a,b) - adds any two numbers. (The first number is "a" and the second number is "b")

show( ) - displays anything that is inside the brackets on to the screen.

Write a short program which creates a **ScreenBug** called **Betty** , adds two numbers and shows the answer on the screen.

```
public class ScreenBugtry

import java.awt.*;
{
 public static void main (String [] args)
 {
 }
```

Q.9. When you were programming what did you enjoy doing?

Q.10. When did you learn from programming?

Appendix F

The Ready IDE from Holt Software

"Ready" is designed as a stand alone IDE and is aimed at running independently of JDK. It also provides basic templates which may be used to provide a framework of basic setup commands to get going on a program. The package is free to schools.

The primary constituent of the HSA package (Holt Software Associates) is the Console class which provides a stand-alone environment for simple input and output in a standard window. The Ready IDE provides standard templates for standard applications and applets and applications utilizing the Console class. Output may be saved or printed.

Appendix G Tables

Table 1 – Results Spreadsheet

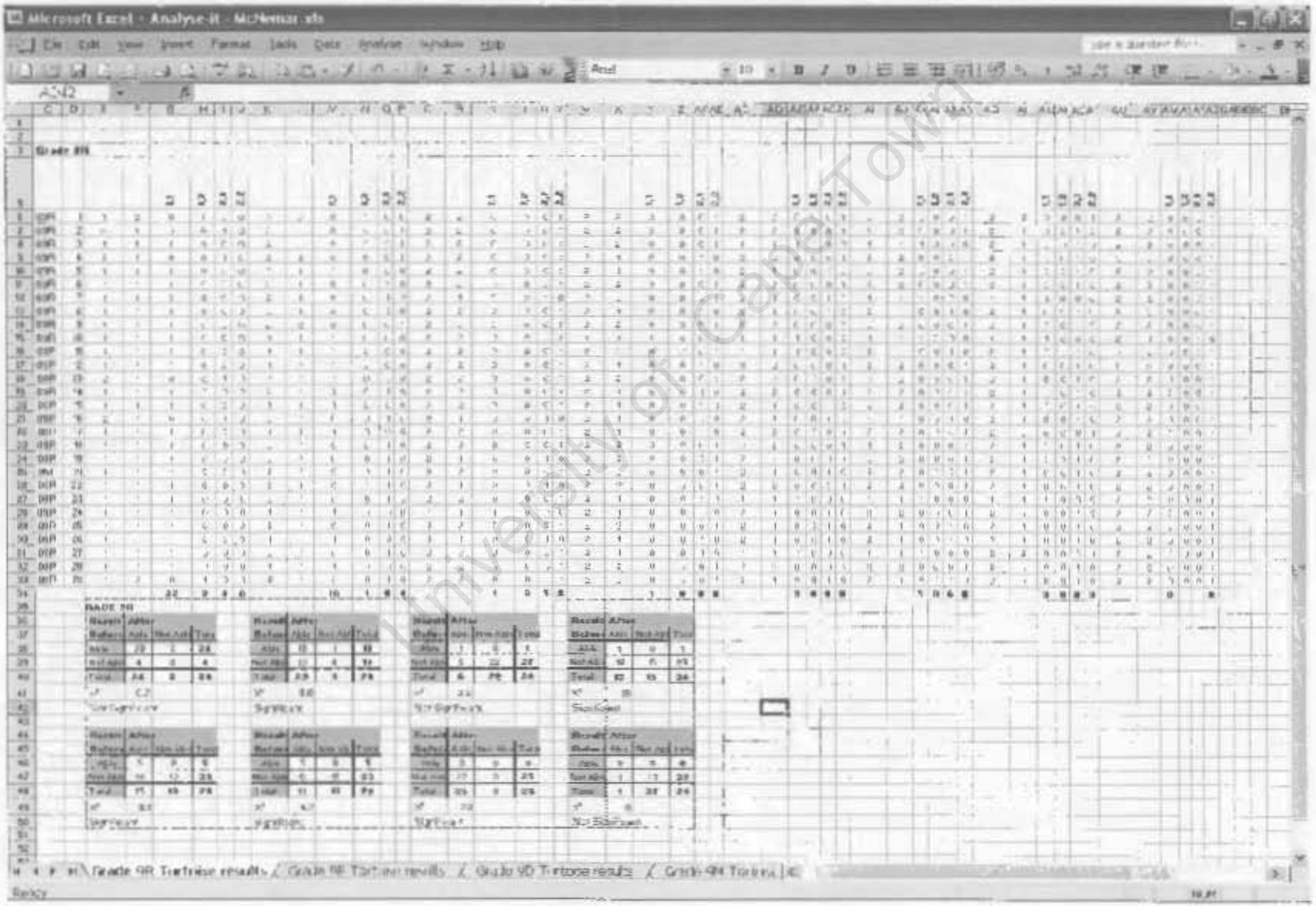


Table 2 – 9R Contingency Tables

GRADE 9R

Q.1

Result	After		
Before	Able	Not Able	Total
Able	22	2	24
Not Able	4	0	4
Total	26	2	28

X^2 0.2
Not Significant

Q.2.

Result	After		
Before	Able	Not Able	Total
Able	10	1	11
Not Able	13	4	17
Total	23	5	28

X^2 8.6
Significant

Q.3.

Result	After		
Before	Able	Not Able	Total
Able	1	0	1
Not Able	5	22	27
Total	6	22	28

X^2 3.2
Not Significant

Q.4.

Result	After		
Before	Able	Not Able	Total
Able	1	0	1
Not Able	12	15	27
Total	13	15	28

X^2 10
Significant

Q.5.

Result	After		
Before	Able	Not Able	Total
Able	5	0	5
Not Able	10	13	23
Total	15	13	28

X^2 8.1
Significant

Q.6.

Result	After		
Before	Able	Not Able	Total
Able	5	0	5
Not Able	6	17	23
Total	11	17	28

X^2 4.2
Significant

Q.7.

Result	After		
Before	Able	Not Able	Total
Able	3	0	3
Not Able	22	3	25
Total	25	3	28

X^2 20
Significant

Q.8.

Result	After		
Before	Able	Not Able	Total
Able	0	0	0
Not Able	1	27	28
Total	1	27	28

X^2 0
Not Significant

Table 3 – 9E Contingency Tables

GRADE 9E

Q.1.

Result Before	After Able	After Not Able	Total
Able	15	1	16
Not Able	6	5	11
Total	21	6	27

X^2 2.3
Not Significant

Q.2.

Result Before	After Able	After Not Able	Total
Able	5	3	8
Not Able	10	9	19
Total	15	12	27

X^2 2.8
Not Significant

Q.3.

Result Before	After Able	After Not Able	Total
Able	0	7	7
Not Able	1	19	20
Total	1	26	27

X^2 3.1
Not Significant

Q.4.

Result Before	After Able	After Not Able	Total
Able	1	0	1
Not Able	7	19	26
Total	8	19	27

X^2 5.1
Significant

Q.5.

Result Before	After Able	After Not Able	Total
Able	4	[illegible]	4
Not Able	10	13	23
Total	14	13	27

X^2 8.1
Significant

Q.6.

Result Before	After Able	After Not Able	Total
Able	4	2	6
Not Able	7	14	21
Total	11	16	27

X^2 1.8
Not Significant

Q.7.

Result Before	After Able	After Not Able	Total
Able	2	1	3
Not Able	9	15	24
Total	11	16	27

X^2 4.9
Significant

Q.8.

Result Before	After Able	After Not Able	Total
Able	0	0	0
Not Able	1	26	27
Total	1	26	27

X^2 0
Not Significant

Table 4 – 9D Contingency Tables

GRADE 9D

Q.1

Result	After		
Before	Able	Not Able	Total
Able	3	0	3
Not Able	2	14	16
Total	5	14	19

X^2 0.5

Not Significant

Q.2

Result	After		
Before	Able	Not Able	Total
Able	2	0	2
Not Able	4	13	17
Total	6	13	19

X^2 2.3

Not Significant

Q.3.

Result	After		
Before	Able	Not Able	Total
Able	0	0	0
Not Able	3	16	19
Total	3	16	19

X^2 1

Not Significant

Q.4.

Result	After		
Before	Able	Not Able	Total
Able	0	1	1
Not Able	2	16	18
Total	2	17	19

X^2 0

Not Significant

Q.5.

Result	After		
Before	Able	Not Able	Total
Able	0	0	0
Not Able	4	15	19
Total	4	15	19

X^2 2.3

Not Significant

Q.6.

Result	After		
Before	Able	Not Able	Total
Able	2	0	2
Not Able	4	13	17
Total	6	13	19

X^2 2.3

Not Significant

Q.7.

Result	After		
Before	Able	Not Able	Total
Able	0	0	0
Not Able	6	13	19
Total	6	13	19

X^2 4

Significant

Q.8.

Result	After		
Before	Able	Not Able	Total
Able	0	0	0
Not Able	1	18	19
Total	1	18	19

X^2 0

Not Significant

Table 5 – 9M Contingency Tables

GRADE 9M

Q.1.

Result / Before	After: Able	After: Not Able	Total
Able	1	5	6
Not Able	4	17	21
Total	5	22	27

X^2 0

Not Significant

Q.2.

Result / Before	After: Able	After: Not Able	Total
Able	1	1	2
Not Able	8	17	25
Total	9	18	27

X^2 4

Significant

Q.3.

Result / Before	After: Able	After: Not Able	Total
Able	0	0	0
Not Able	3	24	27
Total	3	24	27

X^2 1.3

Not Significant

Q.4.

Result / Before	After: Able	After: Not Able	Total
Able	0	0	0
Not Able	5	22	27
Total	5	22	27

X^2 3

Not Significant

Q.5.

Result / Before	After: Able	After: Not Able	Total
Able	0	1	1
Not Able	4	22	26
Total	4	23	27

X^2 0.8

Not Significant

Q.6.

Result / Before	After: Able	After: Not Able	Total
Able	1	0	1
Not Able	3	23	26
Total	4	23	27

X^2 1.3

Not Significant

Q.7.

Result / Before	After: Able	After: Not Able	Total
Able	0	0	0
Not Able	5	22	27
Total	5	22	27

X^2 3.2

Not Significant

Q.8.

Result / Before	After: Able	After: Not Able	Total
Able	0	0	0
Not Able	0	27	27
Total	0	27	27

X^2 0

Not Significant

Table 6 – Class Totals Contingency Tables

TOTALS

Q.1

Result	After		
Before	Able	Not Able	Total
Able	41	8	49
Not Able	16	36	52
Total	57	44	101

X^2 2.042
Not Significant

Q.2

Result	After		
Before	Able	Not Able	Total
Able	18	5	23
Not Able	35	43	78
Total	53	48	101

X^2 21.03
Significant

Q.3

Result	After		
Before	Able	Not Able	Total
Able	1	7	8
Not Able	12	81	93
Total	13	88	101

X^2 0.842
Not Significant

Q.4.

Result	After		
Before	Able	Not Able	Total
Able	2	1	3
Not Able	26	72	98
Total	28	73	101

X^2 21.33
Significant

Q.5.

Result	After		
Before	Able	Not Able	Total
Able	9	1	10
Not Able	28	63	91
Total	37	64	101

X^2 23.31
Significant

Q.6.

Result	After		
Before	Able	Not Able	Total
Able	12	2	14
Not Able	20	67	87
Total	32	69	101

X^2 13.14
Significant

Q.7.

Result	After		
Before	Able	Not Able	Total
Able	5	1	6
Not Able	42	53	95
Total	47	54	101

X^2 37.21
Significant

Q.8.

Result	After		
Before	Able	Not Able	Total
Able	0	0	0
Not Able	3	98	101
Total	3	98	101

X^2 1.333
Not Significant

Bibliography

Allen, J.R., Burke, M.E. and Johnson, J.F., *Thinking about Logo*, Holt Rinehart & Winston, New York, 1983

Alpay, E., *Self-Concept and Self-Esteem*, Department of Chemical Engineering and Chemical Technology, Imperial College of Science, Technology and Medicine, UK, November 2001, Available from: http://www.ce.ic.ac.uk/common-room/files/PsychEd_6.pdf, [Accessed June 19th 2004]

Anderson, R., Anderson, R. and Deibel, K., *Analyzing Concept Groupings of Introductory Computer Programming Students*, University of Washington, Department of Computer Science and Engineering, Seattle, WA, November, 2004, Available from: http://www.cs.washington.edu/homes/deibel/papers/iticse04-bootstrap/iticse04-bootstrap.pdf

Armitage, B. and Berry, G., *Statistical Methods in Medical Research*, Blackwell Science, Oxford, UK, 1994

Baldwin, D., *Standard Input and Output Streams*, Computer Studies Department Austin Community College, Austin, TX, USA, Java Programming, Lecture Notes No 34, March, 1999, Available from:http://www.dickbaldwin.com/java/Java034.htm, [Accessed July 5th 2004]

Baldwin, R.G., *The AWT Package, Placing Components in Containers, Absolute Coordinates*, Yerevan Physics Institute, Cosmic Ray Division, Armenia, Java Programming, Lecture Notes No 112, February, 1998, Available from: http:// crdlx5.yerphi.am/~nerses/doc/Java/Introduction/Java112.htm, [Accessed July 31st 2003]

Baldwin, R., *Java 2D Graphics, Simple Affine Transforms*, Developer.com, Java Programming, , Lecture Notes No 306, March, 2000, Available from: http://www.developer.com/net/cplus/article.php/626051 [Accessed 17th October 2003]

Barnes, D.J., *Teaching Introductory Java through LEGO MINDSTORMS Models* ACM SIGCSE Bulletin , Proceedings of the 33rd SIGCSE technical symposium on Computer science education, Vol 34 No 1, pp 147-151, February , 2002, ACM Press, New York, NY, USA, Available from: http://portal.acm.org/citation.cfm?id=563397&jmp =cit&coll=portal&dl=ACM&CFID=22777812&CFTOKEN=91008658#CIT, [Accessed June 16th 2004]

Baskerville, R.L., *Investigating Information Systems With Action Research*, Vol 2, No 19, 1999, Communications of the Association for Information Systems, Available from: http://www.cis.gsu.edu/~rbaskerv/CAIS_2_19/CAIS_2_19.html, [Accessed 26th January 2004]

Battista, M.T. and Clements, D.H., *A Case for a Logo-Based Elementary School Geometry Curriculum*, Arithmetic Teacher, No 36, pp 11-17, 1988, Available from: http://www.terc.edu/investigations/relevant/html/ACaseforLogo.html, [Accessed June 15th 2004]

Becker, B.W., *Teaching CS1 with Karel the Robot in Java*, Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education, Charlotte, North Carolina, USA, pp50-54, ACM Press, New York, NY, USA 2001, Available from: http://portal.acm.org/citation.cfm?id=364447.364536&dl=GUIDE&dl=ACM&type=series&idx=364447&part=Proceedings&WantType=Proceedings&title=Technical%20Symposium%20on%20Computer%20Science%20Education, [Accessed June 12th 2004]

Becker, K., *Simple Java I/O Source Code*, Department of Computer Science, Faculty of Science, University of Calgary, Canada, October, 2003, Available from: http://pages.cpsc.ucalgary.ca/~becker/235-Course-Directory/Stdin.java, [Accessed July 5th 2004]

Biddle, R. and Tempero, E., *Java Pitfalls for Beginners*, ACM SIGCSE Bulletin archive Vol 30 , No 2, pp 48-52, June, 1998, ACM Press, New York, NY, USA, 1998, Available from: http://portal.acm.org/citation.cfm?id=292441&coll=portal&dl=ACM&CFID=22777812&CFTOKEN=91008658, [Accessed June 15th 2004]

Bishop, J.M., *Java Gently: Programming Principles Explained*, Addison Wesley Publishing Company, Pretoria, 2001, Available from: http://www.cs.up.ac.za/javagently, [Accessed September 20th 2003]

Bowman, C.F. (Ed), *Wisdom of the Gurus – A Vision for Object Technology,* SIGS, New York, 1997

Braught, G., *Turtle Source Code*, Computer Science 132 Introduction To Computing II, Dickinson College Spring Semester 2003, Available from: http://www.dickinson.edu/~braught/courses/cs132s03/code/Turtle.INT.src.html, [Accessed August 24th 2003]

Buck, D. and Stucki, D.J., *JKarelRobot: A Case Study in Supporting Levels of Cognitive Development in the Computer Science Curriculum,* SIGSCE Bulletin, Vol 33, No 1, pp 16-20, March 2001

Bruce, K.B., Danyluk, A. and Murtagh. T., *A library to support a graphics-based object-first approach to CS 1*, Technical Symposium on Computer Science Education, Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education, Charlotte, NC, USA, pp 6-10, 2001, Available from: http://portal.acm.org/citation.cfm?id=364527&dl= ACM&coll=portal, [Accessed June 22nd 2004]

Burke, M.P., *Logo and Models of Computation,* Addison Wesley, Menlo Park, CA, USA, 1987

Calloni, B.A. and Bagert, D.J., *ICONIC Programming In BACCII Vs. Textual Programming: which is a better learning environment?,* Technical Symposium on Computer Science Education, proceedings of the twenty-fifth SIGCSE symposium on

Computer Science Education, Phoenix, Arkansas, United States, pp188-192, ACM Press, New York, NY, USA , 1994, Available from: http://portal.acm.org/citation.cfm?id=191029.191103&dl=GUIDE&dl=ACM&type=series&idx=191029&part=Proceedings&WantType=Proceedings&title=Technical%20Symposium%20on%20Computer%20Science%20Education, [Accessed June 10th 2004]

Carey, T.T. and Shepher, M.M., *Towards Empirical Studies of Programming in New Paradigms*, Proceedings of the 1988 ACM 16th annual conference on Computer Science, 1988, Atlanta, Georgia, United States, pp72 – 78, ACM Press, New York, NY, USA, Available from: http://portal.acm.org/citation.cfm?id=322618&jmp=cit&coll=portal&dl=ACM&CFID=22777812&CFTOKEN=91008658#CIT, [Accessed June 16th 2004]

Clark,D., MacNish, C. and Royle, G.F., *Java as a Teaching Language — Opportunities, Pitfalls and Solutions,* Proceedings of the Third Australasian Conference on Computer Science Education, ACM, Brisbane, Australia, July 1998, ACM Press, New York, NY, USA, 1998, Available from: http://portal.acm.org/citation.cfm?id=289418&coll=portal&dl=ACM&CFID=22777812&CFTOKEN=91008658, [Accessed June 14th 2004]

Clements, D.H., *Computers in Early Childhood Mathematics,* Contemporary Issues in Early Childhood, Vol 3, No 2, pp160-181, University at Buffalo, State University of New York, USA, 2002, Available from: http://www.gse.buffalo.edu/ RP/PDFs/ ECE_Comp _Math.pdf, [Accessed June 8th 2004]

Clements, D.H. and Meredith, J.S., *Research on Logo: Effects and Efficacy*, Logo Foundation, New York, NY, USA, 1992, Available from: http://el.media.mit.edu/logo-foundation/pubs/papers/research_logo.html, [Accessed June 15th 2004]

Clements, D.H. and Sarama, J., *The Role of technology in Early Childhood Learning,* Teaching Children Mathematics, February 2002, pp 340-343, National Council of Teachers Mathematics, USA, Available from: http://www.gse.buffalo.edu/RP/PDFs/Role_of_Technology.pdf, [Accessed June 8th 2004]

Cockburn, Alistair, *Surviving Object-Oriented Projects,* Addison-Wesley, Reading, MA, USA,1997

Cockburn, A. and Churcher, N., *Toward Literate Tools for Novice Programmers*, Proceedings of the 2nd Australasian conference on Computer Science Education The Univ. of Melbourne, Australia , July, 1997, pp 107-116, ACM Press, New York, NY, USA, 1997, Available from: http://portal.acm.org/citation.cfm?id=299376&jmp=cit&coll=portal&dl=ACM&CFID=23420851&CFTOKEN=6176502#CIT, [Accessed June 20th 2004]

Colella, V.S., Klopfer, E. and Resnick, M., *Adventures in Modeling: Exploring Complex Dynamic Systems with StarLogo,* Teachers College Press, 2001, Available from: http://education.mit.edu/starlogo/adventures/, [Accessed July 1st 2004]

Corritore, C. L. and Weidenbeck, S., *What do novices learn during program comprehension*, Int. J. Human-Computer Interaction , Vol 3 No 2, pp199-222, 1991, Available from: http://www2.umassd.edu/CISW3/coursepages/pages/CIS411/notes/ Corritore91.html, [Accessed June 5th 2004]

Czaja, R. & Blair, J., *Designing Surveys*, Pine Forge Press, Thousand Oaks, CA, USA, 1996

Davies, S.P., *Expertise and the Comprehension of Object-Oriented Programs*, 12th Workshop of the Psychology of Programming Interest Group, Cozenza Italy, pp 61-66, April 2000, Available from: http://www.ppig.org/papers/12th-davies.pdf, [Accessed June 2nd 2004]

Davies, S. P., Gilmore, D. J. and Green, T. R. G., *Are objects that important? The effects of expertise and familiarity on the classification of object-oriented code*. Human-Computer Interaction, Vol 10 Nos 2 and 3, pp 227-248, 1995, Available from: homepage.ntlworld. com/greenery/ workStuff/res-proglangs.html, [Accessed June 24th 2004]

Degelman, D., Brokaw, E.J. and Free, J.U., *Effects of Logo Experience and Grade on Concept learning and Creativity*, Baywood Publishing Company, Inc, Journal of Educational Computing Research, Vol 2, pp 199-205, 1986, Available from: http://www.vanguard.edu/faculty/ddegelman/logo.pdf, [Accessed June 3rd 2004]

Deppeler, D., *Turtle Javadocs*, Computer Sciences, University of Wisconsin, Madison, CS302 java Information, Galapagos API, July, 2003, Available from: www.cs.wisc.edu/~cs302/resources/galapagos/javadocs/galapagos/DrawingCanvas.html [Accessed 28th August 2003]

Devarakonda, S., *Turtle Source Code*, Duke University, Durham, NC 27708, USA, 2003, Available from: www.duke.edu/~ccg3/slogo/slogo/turtle.java [Accessed August 21st 2003]

Dick, B., *Action Research: action and research,* Paper for seminar "Doing Good Action Research", Southern Cross University, February, 2002, Available from: http://www.scu.edu.au/schools/gcm/ar/arp/aandr.html [Accessed 25th January 2004]

Dickens, P.M. and Thakur, R., *An Evaluation of Java's I/O Capabilities for High-Performance Computing*, Proceedings of the ACM 2000 conference on Java Grande, San Francisco, CA, USA, pp 26-35, June, 2000, ACM Press, New York, NY, USA, 2000, Available from: http://portal.acm.org/ft_gateway.cfm?id=331628&type=html&coll= ACM&dl=ACM&CFID=23724455&CFTOKEN=29386243, [Accessed July 5th 2004]

Duke, R., Salzman, E., Burmeister, J., Poon, J. and Murray, L., *Teaching Programming to Beginners - choosing the language is just the first step*, Proceedings of the Australasian Conference on Computing Education, Melbourne, Australia, 2000, pp79-86, ACM Press, New York, NY, USA, 2000, Available from: http://portal.acm.org/citation.cfm?id= 359381&dl =ACM&coll=portal, [Accessed June 14th 2004]

Eckert , M., *Turtle Source Code*, Technische Universität Darmstadt, Bibliothek fuer den Informatik-Vorkurs WS2001/TU-Darmstadt, Vorkurs im Programmieren,Wintersemester, February 2001, Available from: www.dvs1.informatik.tu-darmstadt.de/DVS1/lectures/ws01-02/vorkurs/beispielloesungen/Turtle.java
[Accessed 26th August 2003]

Entsminger, G., *The Tao of Objects,* M&T, Redwood City, CA, USA, 1995

Ezell, C. L., *Creating Pedagogical Programming Environments*, ACM SIGCSE Bulletin archive, Vol 22 No 2, pp 42-46, June, 1990, ACM Press, New York, NY, USA, 1990, Available from: http://portal.acm.org/citation.cfm?id=126454&coll=ACM&dl=ACM&CFID=23726493&CFTOKEN=59445810, [Accessed July 2nd 2004]

Farrell, J., *Java Programming*, Course Technology, Cambridge, MA, USA,1999

Fischler, M.A. and Firschein, O., *Intelligence – The Eye, the brain and the computer*, Addison-Wesley, Boston, MA, USA, 1987

Flanagan, D., *Java in a Nutshell*, O'Reilly, Sebastopol, CA, USA, 1999

Fleury, A.E., *Encapsulation and Reuse as Viewed by Java Students*, Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education, Charlotte, North Carolina, United States , Vol 33 No 1, pp189-193, 2001, ACM Press, New York, 2001, Available from: http://portal.acm.org/results.cfm?coll=portal&dl=ACM&CFID =22937284&CFTOKEN=24337902, [Accessed June 15th 2004]

Fox, D.J., *The Research Process in Education*, Holt, Reinhart and Winston, Inc, New York, 1969

Garside, R. and Mariani, J., *Java: First Contact*, Course Technology, Cambridge, MA, USA, 1987

Gibbons, A., *Turtle Source Code*, Introduction to Programming, CS1INP, Department of Computer Science, King's College London, Strand, London, UK, November, 2001, Available from:www.dcs.kcl.ac.uk/teaching/units/oldinp/Turtle.java, [Accessed 28th August 2003]

Gibbons, J., *Structured Programming in Java*, ACM SIGPLAN Notices archive, Vol 33 No 4, pp 40-43, April, 1998, ACM Press, New York, NY, USA, 1998, Available from: http://www.ulst.ac.uk/cticomp/gibbons.html, [Accessed July 5th 2004]

Gibson, J.P., *Java and education: A noughts and crosses Java applet to teach rogramming to primary school children*, Proceedings of the 2nd international conference on Principles and practice of programming in Java, Kilkenny City, Ireland, pp 85-88, June, 2003, Computer Science Press, Inc., New York, NY, USA, 2003, Available from:

http://portal.acm.org/citation.cfm?id=957315&jmp=cit&coll=portal&dl=ACM&CFID=23055039&CFTOKEN=48384762#CIT, [Accessed June 21st 2004]

Gilnert, E.P. and Tanimoto, S.L., *PICT: An Interactive Programming Graphical Environment*, Vol 17 , No 11, November 1984, pp 7-25, IEEE Computer Society Press, Los Alamitos, CA, USA, Available from: http://portal.acm.org/citation.cfm?id=2313&dl=ACM&coll=portal, [Accessed June 10th 2004]

Gipps, C.V., *Beyond Testing*, The Falmer Press, London, 1994

Green, T.R.G., *Cognitive Approaches to Software Comprehension: results, Gaps and Limitations*, Workshop on Experimental Psychology in Software Comprehension Studies 97, University of Limerick, Ireland, 1997, Available from: homepage.ntlworld.com/greenery/workStuff/ Papers/LimerickTalk1997/LimerickTalk.html, [Accessed June 7th 2004]

Green, T.R.G., *Instructions and Descriptions: some cognitive aspects of programming and similar activities*, Proceedings of Working Conference on Advanced Visual Interfaces (AVI 2000), pp 21-28, ACM Press, New York, 2000, Available from: http://homepage.ntlworld.com/greenery/workStuff/Papers/AVI2000.PDF, [Accessed June 3rd 2004]

Green, T.R.G. and Blackwell, A.F., *Ironies of Abstraction*, In Proceedings 3rd International Conference on Thinking, British Psychological Society, 1996, Available from: http://orion.ramapo.edu/~amruth/r/c/ictem02/paper.pdf, [Accessed June 8th 2004]

Grieser, G, Dr., *Turtle Source Code*, Technische Universität Darmstadt, Bibliothek fuer den Informatik-Vorkurs, WS2001/TU-Darmstadt, Vorkurs im Programmieren, March 2002, Available from:http://www.intellektik.informatik.tu-darmstadt.de/~gunter/VORKURS-WS02-03/FOLIEN/CODE1 [Accessed 29th August 2003]

Grissom, S., *A Pedagogical Framework for Introducing Java I/O in CS1*, ACM SIGCSE Bulletin,Vol 32, No 4, December, 2000, pp 57-59, ACM Press, New York, NY, USA, Available from: http://portal.acm.org/citation.cfm?id=369326&jmp=cit&coll=GUIDE&dl=ACM&CFID=23193899&CFTOKEN=71761720#CIT, [Accessed June 22nd 2004]

Hadjerrouit, S., *A Constructivist Approach to Object-Oriented Design and Programming*, Annual Joint Conference Integrating Technology into Computer Science Education, Cracow, Poland, Proceedings of the 4th annual SIGCSE/SIGCUE ITiCSE conference on Innovation and technology in computer science education, pp 171-174, 1999, ACM Press, New York, 1999, Available from: http://portal.acm.org/citation.cfm?id=305910&dl=ACM&coll=portal, [Accessed June 17th 2004]

Hagarwal, S.B., *JOption Pane*, Java API By Example, From Geeks To Geeks, KickJava site, Available from: http://www.kickjava.com/?http://www.kickjava.com/672.htm [Accessed July 27th 2003]

Hall, M., *JSlider*, Java Programming Resources, coreservlets.com, Reistertown, MD, USA, May, 2002, Available from:http://www.apl.jhu.edu/~hall/java/CWP-Sources/CWP-Examples/ Chapter13/Slider.html, [Accessed August 22nd 2003]

Halland, K. and Malan, K., *Reflections by Tteachers Learning to Program*, Proceedings of SAICSIT 2003, pp 165 –172, September, 2003, South African Institute for Computer Scientists and Information Technologists, Republic of South Africa, Available from: http://portal.acm.org/citation.cfm?id=954032&jmp=cit&coll=portal&dl= ACM&CFID =22777812&CFTOKEN=91008658#CIT, [Accessed June 16th 2004]

Harvey, B., *Symbolic Programming vs the A.P. Curriculum*, The Computing Teacher, pp 27-29, February 1991, Available from: http://www.cs.berkeley.edu/ ~bh/bridge.html

Healy, J.M., *Failure to Connect: How Computers Affect Our Children's Minds—for Better and Worse,* Simon & Schuster, New York, 1999

Hopkins, K.D. & Stanley, J.C., *Educational and Psychological Measurement and Evaluation*, Prentice-Hall, Englewood Cliffs, NJ, USA, 1981

Horton, M.J., *Input.java, Input.class Formerly known as Parse.java* , LMI.net, Berkeley, CA, USA, User Homepage, June, 2002, Available from: http://users.lmi.net/~mikeh/ portfolio/code/Input.htm, [Accessed September 22nd 2003]

Hosch, F., *Java as a first language: an evaluation*, SIGCSE Bulletin, Vol 28 No 3, September 1996, pp. 45-50., ACM Press, New York, NY, USA, 1996, Available from: http://doi.acm.org/10.1145/234867.234877, [Accessed June 23rd 2004]

Huitt, W., *Self-Concept and Self-Esteem*, Educational Psychology Interactive, Dept. of Psychology nd Counseling, Valdosta State University, Valdosta, GA, May 1998, http://chiron.valdosta.edu/whuitt/col/regsys/self.html, [Accessed June 19th 2004]

Hundhausen, C.D., *Toward Effective Algorithm Visualizaion Artifacts: Designing for Participation and Negotiation in an Undergraduate Algorithms Course*, Laboratory for Interactive Learning Technologies, Information and Computer Sciences Department, University of Hawaii at Manoa , June 1999, Available from: http://virtual.inesc. pt/rct/show.php3?id=93, [Accessed June 24th 2004]

Jenkins, T., *On the Difficulty of Learning to Program*, 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences, 27-29 August 2002, Loughborough University, Available from: http://www.ics.ltsn.ac.uk//pub/conf2002/jenkins.html, [Accessed June 22nd 2004]

Johnson, B., *Teacher-As-Researcher. ERIC Digest*, ERIC Clearinghouse on Teacher Education Washington DC, March, 1993, Available from: http://www.ericfacility. net/ericdigests/ed355205.html [Accessed 26th January 2004]

Jones, W.C., Jnr, Dr., *Java Au Naturel*, Central Connecticut State University, Free On-line book, May, 2003, Available from: http://www.cs.ccsu.edu/~jones/book.htm [Accessed August 24th 2003]

Jones, M.G. and Brader-Araje, L., *The Impact of Constructivism on Education: Language, Discourse, and Meaning*, Amercian Communication Journal, Vol 3, No 4, Spring 2002, Available from: http://acjournal.org/holdings/vol5/iss3/special/jones.pdf, [Accessed June 23rd 2004]

Jones, R., Boyle, T. and Rickard, P., *Objectworld: Helping Novice Programmers to Succeed through a Graphical Objects First Approach*, 4rd Annual Conference for LTSN-ICS, 28-28 August 2002, NUI, Galway, Available from: http://www.ics.ltsn.ac.uk/pub/conf2003/ray_jones.htm, [Accessed June 22nd 2004]

Kafai, Y., *Learning Design by making Games Children's Development of Design Strategies in the Creation of a Complex Computational Artefact*, in Kafai, Y. and Resnick, M., (Eds), *Constructivism in Practice Designing, Thinking and Learning in a Digital World*, Lawrence Erlbaum Asssociates, Mahweh, NJ, USA, 1996

Khalil, S., *Turtle Source Code*, University of North Texas, Department of Computer Science, Student Web Server, October, 2002, Available from: http://students.csci.unt.edu/~khalil/5250/assign_6/ [Accessed 28th August 2003]

Kim, K., *Critique on the Torrance Tests of Creative Thinking*, The University of Georgia, Enterprise Information Technology Services, Differeniated Curriculum Unit, 2002 Available from: http://www.arches.uga.edu/~kyunghee/portfolio/review%20of%20ttct.htm [Accessed June 8th 2004]

King, K. N., *The Case for Java as a First Language*, Proceedings of the 35th Annual ACM Southeast Conference, Murfreesboro, TN, April 2-4, 1997, Available from: http://www.gsu.edu/~matknk/java/reg97.htm, [Accessed June 23rd 2004]

Kluit, P.G., Sint, M. and Wester, F., *Visual programming with Java: evaluation of an introductory programming course*, Proceedings of the 6th Annual Conference on the Teaching of Computing and the 3rd Annual Conference on Integrating Technology into Computer Science Education: Changing the Delivery of Computer Science Education, Vol 30 No 3, pp 143-147, August 1998, Available from: http://portal.acm.org/citation.cfm?id=83100&jmpcit&coll=portal&dl=ACM&CFID=23055039&CFTOKEN=48384762#CIT, [Accessed June 21st 2004]

Ko, A.J., Myers, B.A. and Aung, H.H., *Six Learning Barriers in End-User Programming Systems*, Project Marmalade, Human Computer Interaction Institute, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, USA, 2004, Available from: http://www-2.cs.cmu.edu/~ajko/LearningBarriers.pdf, [Accessed June 3rd 2004] (to appear in the IEEE Symposium on Visual Languages and Human-Centric Computing, Rome, Italy, September 26-29, 2004)

Kock, N.F.,Jr., *Myths in Organisational Action Research: Reflections on a Study of Computer-Supported Process Redesign Groups*, Organizations & Society, Vol 4, No 9, pp 65-91, 1997, Available from: http://www.cis.temple.edu/~kock/public/ro&s/ro&s1.html [Accessed 26th January 2004]

Koike, Y., Maeda,Y and Koseki, Y, *Improving Readability of Iconic Programs with Multiple View Object Representation* , 11th International IEEE Symposium on Visual Languages, pp 37, September 1995, Available from: http://www.computer.org/conferences/vl95/html-papers/koike2/koike.htm, [Accessed June 10th 2004]

Kono, S., *Turtle Source Code*, University of the RyuKyus, Department of Information Engineering, "Shinji Kono's Page", Programming I, August, 2000, Available from: www.ie.u-ryukyu.ac.jp/~kono/lecture/2000/programming1/00-07-06/Turtle.java [Accessed August 24th 2003]

Korsh, J.F. and LaFollette, P.S., *A System for Program Visualization in the Classroom*, Proceedings of the twenty-ninth SIGCSE technical symposium on Computer Science Education, Atlanta, Georgia, USA, pp 272-276, March, 1998, ACM Press, New York, NY, USA, 1998, Available from: http://portal.acm.org/results.cfm?coll=portal&dl=ACM&CFID=23420851&CFTOKEN=6176502, [Accessed June 22nd 2004]

Koschmann, T., *Logo-as-Latin Redux*, Journal of Learning Sciences, Vol 6, No 4, pp 409-415, 1997, Available from: lcs.www.media.mit.edu/groups/el/events/files/Koschmann.doc, [Accessed June 11th 2004]

Koschmann, T., *Paradigm Shifts and Instructional Technology: An Introduction*, pp 1-23, in Koschmann, T., CSCL: Theory and Practice of an Emerging Paradigm, Lawrence Erlbaum, Mahwah, NJ, USA, 1996, Available from: http://dwebct.lanet.lv/doc/koschmanns_paradigm.pdf, [Accessed June 25th 2004]

Kozloff, M.A., *Constructivism in Education: Sophistry for a New Age*, Department of Speciality Studies, University of North Carolina at Wilmington, May,1998, Available from: http://www.uncwil.edu/people/kozloffm/ContraConstructivism.html, [Accessed June 2nd 2004]

Kushan, B., *Preparing Programming Teachers*, ACM SIGCSE Bulletin , Proceedings of the twenty-fifth SIGCSE symposium on Computer science education, Phoenix, Arkansas, United States, Vol 26 No1, pp 248-252, March 1994, ACM Press, New York, 1994 Available from: http://portal.acm.org/citation.cfm?id=191134&jmp =cit&coll=portal&dl=ACM&CFID=22937284&CFTOKEN=24337902#CIT, [Accessed June 16th 2004]

Lambert, K.A. and Osborne, M., *Turtle Graphics* (JAR Downloads), Western Washington University, Department of Computer Science, CS211 and CS241, Spring Quarter, 2003, Available from: http://faculty.cs.wwu.edu/ martin/Software%20Packages/BreezyGUI/egal_stuff.htm, [Accessed August 21st 2003]

Lea, D., *Some Questions and Answers about using Java in Computer Science Curricula-*

Computer Science Dept, State University of New York at Oswego, 1996, Available from:http://g.oswego.edu/dl/html/javaInCS.html [Accessed 10th August 2002]

Lee, M., *Introduction to Biostatistics*, Biostatistics Department, UCLA, Spring 2004, Available from: http://www.ph.ucla.edu/biostat/course/100a/paired.htm, [Accessed July 15th 2004]

Lemon, G., *Absoutely Positioning Components in Java*, Juicy Studio, Tutorials, 1999, Available from: www.juicystudio.com/tutorial/java/position.html, [Accessed July 31st 2003]

Lemos, R.S., *Teaching Programming Languages: A Survey of Approaches*, Proceedings of the tenth SIGCSE technical symposium on Computer science education, pp174-181,1979, ACM Press, New York, NY, USA , Available from: http://portal.acm.org/citation.cfm?id =809578&jmp=abstract&dl=GUIDE&dl=ACM, [Accessed June 16th 2004]

Lentczner, M., *Affine Transforms in Java2d* , Glyphic Technology, 1999,Available from: http://www.glyphic.com/transform/imageonly/8wiggly.html, [Accessed 17th October 2003]

Levin, J.R. and Serlin, R.C., *Changing Students' Perspectives of McNemar's Test of Change*, Journal of Statistics Education Vol 8, No 2, 2000, Available from: http://www.amstat.org/publications/jse/secure/v8n2/levin.cfm, [Accessed July 20 2004]

Lister, R., *Teaching Java First: Experiments with a Pigs-Early Pedagogy*, Faculty of Sixth Australasian Computing Education Conference (ACE2004), Dunedin, New Zealand. CRPIT, **30**. Lister, R. and Young, A. L., Eds., ACS. Pp 177-183, Available from: http://crpit.com/confpapers/CRPITV30Lister.pdf, [Accessed June 15th 2004]

Loy, M., Eckstein, R., Wodd, D., Elliott, J. and Cole, B., *Java Swing*, O'Reilly, Sebastapol, CA, USA, 2002

Lukas, G. and Beranek, B., *Uses of the Logo Programming Language in Undergraduate Instruction*, Communications of the ACM,Vol 29 , No7, pp 605-610, July, 1986, ACM Press, New York, NY, USA , 1986, Available from: http://portal.acm.org/citation.cfm ?id=6142&jmp=indexterms&dl=GUIDE&dl=ACM, [Accessed June 15th 2004]

Madden, M. and Chambers, D., *Evaluation of Student Attitudes to Learning the Java Language*, Proceedings of the inaugural conference on the Principles and Practice of programming, 2002 and Proceedings of the second workshop on Intermediate representation engineering for virtual machines, 2002, Dublin, Ireland, pp 125-130, ACM Press, New York, NY, USA , 2002, Available from: http://portal.acm.org/citation.cfm? id=638501&dl=ACM&coll=portal, [Accessed June 14th 2004]

Marshall, Dave *JOptionPane significant methods* Available from: http://www.cs.cf.ac.uk/Dave/HCI/HCI_Handout_CALLER/node96.html [Accessed July 30th 2003]

Matthíasdóttir, A.,*What Students Find Difficult in Learning Programming*, LTSN-ICS 5th Annual Conference 2004, 31st August – 3rd September, Submissions, Available from: www.ics.ltsn.ac.uk/events/ conf2004/Submissions/Matthiasdottir.pdf, [Accessed June 22nd 2004]

Mason, M., *The van Hiele Levels of Geometric Understanding*, Professional Handbook for Teachers, Geometry: Explorations And Applications, pp 4-8, 2004, Available from: http://www.mcdougallittell.com/state/tx/corr/levels.pdf, [Accessed June3rd 2004]

Mayer, R.E., Dyck, J.L. and Vilberg, W., *Learning To Program And Learning To Think: What's The Connection?*, Communications of the ACM, Vol 29 No 7, pp 605-610, July, 1986, ACM Press, New York, NY, USA , 1986, Available from: http://portal.acm.org/ citation.cfm?id=6142&jmp=indexterms&dl=GUIDE&dl=ACM, [Accessed June 17th 2004]

McKeown, J., *Why We Need To Develop Success In Introductory Programming Courses*, CCSC Central Plains Conference, Maryville, MO, April 8-10th, 1999, Available from: http://www.homepages.dsu.edu/mckeownj/CPCCCSCpaper.html, [Accessed June 14th 2004]

Meter, G. and Miller, P., *Engaging Students and Teaching Modern Concepts:Literate, Situated, Object-Oriented Programming*, Proceedings of the twenty-fifth SIGCSE symposium on Computer science education, ACM SIGCSE Bulletin , Vol 26 No 1, pp 329 -333 March 1994, ACM Press, New York, NY, USA , 1994, Available from: http://portal.acm.org/ citation.cfm?id =191161&jmp=cit&coll=portal&dl=ACM&CFID =22937284&CFTOKEN =24337902#CIT ,[Accessed June 16th 2004]

Miller, P., Pane, J. Meter, G. and Vorthmann, S., *Evolution of Novice Programming Environments: the Structure Editors of Carnegie Mellon University,* Interactive Learning Environments, Vol 4 No 2, pp 140-158, 1994, Available from: http://web.cs.cmu.edu/ ~pane/ftp/ILE.pdf, [Accessed June 5th 2004]

Modugno, F., Corbett, A.T. and Myers, A., *Evaluating Program Representation in a Demonstrational Visual Shell,* The ACM Transactions on Computer Human Interaction, Vol 4, No 3, pp 276-308., ACM Press, New York, NY, USA , 1996, Available from: http://www.acm.org/sigchi/chi95/Electronic/documnts/shortppr/mod_bdy.htm

Naps, T.L., Eagen, J.R. and Norton, L.L., *JHAVÉ – An Envirnoment to Actively Engage Students in Web-bases Algorithm Visualizations*, Proceedings of the thirty-first SIGCSE technical symposium on Computer Science Education, Austin, Texas, USA, pp 109-113, March, 2000, ACM Press, New York, NY, USA, Available from: http://portal.acm.org/ citation.cfm?id=331829&dl=ACM&coll=portal

Moström J.E. and Carr D.A., *Programming Paradigms and Program Comprehension by Novices* , Dept of Computer Science and Centre of Distance Spanning Technology, Lulea University of Technology, S-971 87, Lulea, Sweden, research Report, October, 1997, Available from: http://www.ida.liu.se/ ~davca/postscript/parvsserial.pdf, [Accessed June 8th 2004]

Nevison, C. and Wells, B., *Using a Maze Case Study to Teach Object-Oriented Programming and Design Patterns*, Sixth Australasian Computing Education Conference (ACE2004), Dunedin, New Zealand, in Conferences in Research and Practice in Information Technology, Lister, R. and Young, A. L., Eds., Australian Computer Society, Vol 30, pp 207-215, 2004, Available from:http: //crpit.com/confpapers/ CRPITV30Nevison.pdf, [Accessed 21st June 2004]

Oelschlegel, E.J., *Turtle Source Code,* University of Pittsburgh, PA, USA, January, 2003, Available from:http://hornygoat.org/source/java/proj3/, [Accessed 26th August 2003]

Pane, J.F. and Myers, B.A.,*Usability Issues in the Design of Novice Programming Systems*, School of Computer Science Technical Report CMU-CS-96-132, Carnegie Mellon University, Pittsburgh, PA, 1996, Available from: http://web.cs.cmu.edu/~pane/ftp/CMU-CS-96-132.pdf, [Accessed June 6th 2004]

Papert, S., *A Word for Learning*, in Kafai, Y. and Resnick, M., (Eds), *Constructivism in Practice Designing, Thinking and Learning in a Digital World*, Lawrence Erlbaum Asssociates, Mahweh, NJ, USA, 1996

Papert, S., *Does Easy Do It? Children, Games, and Learning*, Game Developer Magazine "Soapbox" section, pp 88, 1998, Available from: http://www.papert.org/articles/ Doeseasydoit.html, [Accessed June 9th 2004]

Papert, S., *Mindstorms Children, Computers, and Powerful Ideas*, The Harvester Press Ltd, Brighton, Sussex, UK, 1980

Papert, S., *Papert on Piaget*, Time Magazine Special Issue "The Century's Greatest Minds", March 29, 1999, Available from: http://www.papert.org/articles/ Papertonpiaget.html, [Accessed June 7th 2004]

Papert, S., *What is Logo? Who needs it?*, Essay from Logo Computer Systems Inc "Logo Philosophy and Implementation", Canada, 1999, Available from: http://www.microworlds. com/company/philosophy.pdf, [Accessed June 5th 2004]

Petre, M., Blackwell, A.F. and Green, T.R.G., *Cognitive Questions in Software Visualisation*, in Stasko, J., Domingue, J., Brown, M. and Price, B. (Eds), *Software Visualization: Programming as a Multi-Media Experience*, MIT Press, pp 453-480, January, 1998, Available from: http://www.cl.cam.ac.uk/users/afb21/publications/book-chapter.html, [Accessed June 14th 2004]

Pollack, M., *Code generation using Javadoc*, JavaWorld, August, 2000, Available from: http://www.javaworld.com/javaworld/jw-08-2000/jw-0818-javadoc.html, [Accessed September 22nd 2003]

Prendergast ,M., *Seven Stages in my First Action Research Project*, Queen's University , Faculty of Education, Kingston, Ontario, Canada, 2000, Available from: http://educ. queensu.ca/projects/action_research/michael.htm [Accessed 26th January 2004]

Proulx, V.K., *Programming Patterns and Design Patterns in the Introductory Computer Science Course*, Proceedings of the thirty-first SIGCSE technical symposium on Computer science education, Austin, Texas, United States, pp 80-84, March, 2000, ACM Press, New York, NY, USA, Available from: http://portal.acm.org/citation.cfm?id=331819&jmp= cit&coll=portal&dl=ACM&CFID=23055039&CFTOKEN=48384762#CIT, [Accessed June 20th 2004]

Proulx, V. and Rasala, R., *Turtle Source Code*, Northeastern University, Boston, MA, USA, College of Computer and Information Science, "Java Power Tools", October, 2002, Available from: www.ccs.neu.edu/jpt/src/pedagogy/Turtle.java [Accessed 26th August 2003]

Proulx, V. and Rasala, R., *Java IO and testing made simple*, Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education, Norfolk, Virginia, USA, pp 161-165, March, 2004, ACM Press, New York, NY, USA, 2004, Available from: http://portal.acm.org/citation.cfm?id=971358&jmp=cit&coll=portal&dl=ACM&CFID=230 55039&CFTOKEN=48384762#CIT, [Accessed June 23rd 2004]

Qiu, L., *Intelligent Educational Systems for Teaching Programming*, Department of Computer Science, Northwestern University, Evanston, IL , USA, March, 2004, Available from: http://www.cs.northwestern.edu/~qiu/critiquer/publications/acm2004.pdf , [Accessed June 12th 2004]

Ramalingham, V. and Wiedenbeck, S., *An empirical study of novice program comprehension in the imperative and object-oriented styles*, Papers presented at the 7th workshop on Empirical studies of programmers, pp124–139, Alexandria, Virginia, USA, October, 1997, ACM Press, 1997, Available from: http://portal.acm.org/citation.cfm?id= 266411&jmp=cit&coll=portal&dl=ACM&CFID=22777812&CFTOKEN=91008658#CIT, [Accessed June 16th 2004]

Resnick, M.,*Turtles, Termites and Traffic Jams*, MIT Press, Cambridge, MA, USA, 1994

Rinkens, T. and Windhorst, S., *Models: Turtle*, sourceFORGE.net, Open Source software site, 2003, Available from: http://rcxtools.sourceforge.net/model/sources/e_source04.html [Accessed 26th August 2003]

Roberts, E., *An Overview of MiniJava*, ACM SIGCSE Bulletin , Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education, Charlotte, NC, USA, Vol 33 No 1, pp 1-5, February, 2001, ACM Press, New York, NY, USA, 2001, Available from:http://portal.acm.org/citation.cfm?id=364525&jmp=cit&coll=ACM&dl= ACM&CFID=23724455&CFTOKEN=29386243#CIT United States, [Accessed July 5th 2004]

Roberts, E., *The Dream of a Common Language: The Search for Simplicity and Stability in Computer Science Education*, Proceedings of the 35th SIGCSE technical symposium on Computer science education, March 3–7, 2004, Norfolk, Virginia, USA., pp 115-119, ACM Press, New York, NY, USA, 2004, Available from: http://db.grinnell.edu/sigcse/sigcse2004/viewAcceptedSession.asp? sessionType=Paper&sessionNumber=283, [Accessed June 24th 2004]

Roberts, E. and Picard, A., *Designing a Java Graphic Library for CS1*, Proceedings of the 6th annual conference on the teaching of computing and the 3rd annual conference on Integrating technology into computer science education: Changing the delivery of computer science education, Dublin City University, Ireland, pp 213-217, August, 1998, Available at: http://portal.acm.org/citation.cfm?id=282991.283129&dl=portal&dl=ACM&type= series&idx=282991&part=Proceedings&WantType=Proceedings&title=Annual%20Joint%20Conference%20Integrating%20Technology%20into%20Computer%20Science%20Education, [Accessed une 3rd 2004]

Robinson, M. and Vorobiev, P., *Swing*, Manning, Greenwich, CT, USA, 2003

Ross, J.M. and Zhang, H., *Structured Programmers Learning Object-Oriented Programming*, SIGCHI, Vol 29 No 4, October, 1997, Available from: http://www.acm.org/sigchi/bulletin/1997.4/ross.html, [Accessed June 16th 2004]

Sanchez, R.J.P. and Roda, M.D.S., *Relationships between Self Concept and Academic Achievement in Priamry Students*, Electronic Journal of Research in Pschology and Psychopedagogy, Vol 1 No 1, pp 95-120, 2002, Available from: http://www.investigacion-psicopedagogica.org/revista/articulos/1/english/Art_1_7.pdf, [Accessed June 19th 2004]

Sangwan, R.S., Korsh, J.F. and LaFollette, P.S., *A System for Program Visualization in the Classroom*, ACM SIGCSE Bulletin , Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education, Vol 30 No1, pp 272-276, March 1998, ACM Press, New York, NY, USA , 1998, Available from: http://portal.acm.org/citation.cfm?id=274311&jmp=cit&coll=portal&dl=ACM&CFID=22937284&CFTOKEN=24337902#CIT, [Accessed June 14th 2004]

Schollmeyer, M., *Computer programming in high school vs. college*, ACM SIGCSE Bulletin , Proceedings of the twenty-seventh SIGCSE technical symposium on Computer science education, Philadelphia, PA, USA,Vol 28, No 1, pp 378-382, March 1996, Available from: http://portal.acm.org/citation.cfm?id=236584&jmp=cit&coll=portal&dl=ACM&CFID=22976789&CFTOKEN=19298574#CIT, [Accessed June 22nd 2004]

Seneviratne, A., *Uses of Interface javax.swing.event.ChangeListener*, MobQos Group, University of New South Wales, Sydney, Australia, Available from: http://mobqos.ee.unsw.edu.au/java-2/docs/api/javax/swing/event/class-use/ChangeListener.html [Accessed August 22nd 2003]

Sheehan,R., *Children's perception of Computer Programming as an Aid to Designing Programming Environments*, Proceeding of the 2003 conference on Interaction design and children, Preston, Engalnd, July 2003, pp 75-83, ACM Press, New York, NY, USA, Available from: http://portal.acm.org/citation.cfm?id=953548&jmp=cit&coll=portal&dl=ACM&CFID=23055039&CFTOKEN=48384762#CIT, [Accessed June 21st 2004]

Sigel, I.E., Brodzinsky, D.M. and Golinikoff, R.M., Sigel, I.E., *New Directions in Piagetian Theory and Research: An Integratice Perspective*, in Brodzinsky, D.M. and Golinikoff, R.M., (Eds.), *New Directions in Piagetian Theory and Practice,* Lawrence Erlbaum Associates, Hillsdale, NJ, USA, 1981

Smith, D.C., Cypher, A and Tesler, L., *Programming by Example: Novice Programming Comes of Age*, Communications of the ACM, March, 2000, Vol 43, No 3, pp75-81, Available from: http://portal.acm.org/citation.cfm?id=330544&jmp=cit&coll=portal&dl=ACM&CFID=23129667&CFTOKEN=27710340#CIT, [Accessed June 22nd 2004]

Smock, C. D., *Constructivism and Educational Practices*, in Brodzinsky, D.M. and Golinikoff, R.M., (Eds.), *New Directions in Piagetian Theory and Practice*, Lawrence Erlbaum Associates, Hillsdale, NJ, USA, 1981

Solomon, C. J,. *Teaching Young Children to Program in a LOGO Turtle Computer Culture*, ACM SIGCUE Outlook archive, Vol 12 No 3, pp 20-29, July, 1978, ACM Press, New York, NY, USA, 1978, Accessible from: http://portal.acm.org/results.cfm?coll=ACM&dl=ACM&CFID=23572506&CFTOKEN=8602147 [Accessed July 3rd 2004]

Soloway, E., *Learning to Program = Learning to Consruct Mechanisms and Explanations*, Communications of the ACM, Vol 29 No 9, pp 850-858, September, 1986, ACM Press, New York, NY, USA, Available from: http://portal.acm.org/citation.cfm?id=6594&jmp=cit&coll=Portal&dl=GUIDE&CFID=23463740&CFTOKEN=11586355#CIT , [Accessed June 12th 2004]

Staats, W.J. and Blum, T., *Enhancing an Object-Oriented Curriculum: Metacognitive Assessment and Training*, 29th ASEE/IEEE Frontiers in Education Conference 13b, pp7-13, November 10 - 13, 1999 San Juan, Puerto Rico, Available from: http://fie.engrng.pitt.edu/ fie99/papers/1263.pdf, [Accessed June 13th 2004]

Storey, M-A, Dr., *Turtle Programming Assignment*, Department of Computer Science, University of Victoria,Victoria, BC, Canada, Assignment 3, CSC 115 Fundamentals of Programming: II, Fall, 2002 Available from: http://www.cs.uvic.ca/~mstorey/teaching/csc115/assignments/assignment_3.html, [Accessed 28th August 2003]

Tuckman, B.W. *Measuring Educational Outcomes: Fundamentals of Testing*, Harcourt Brace Jovanovich, New York,1975

Van der Linden, P., *Simple Input from the Keyboard for All Primitive Types*, May, 1997, Available from: http://pvdl.best.vwh.net/EasyIn.txt, [Accessed July 5th 2004]

Van Hiele, P.M., Structure *and Insight A Theory of Mathematics Education*, Academic Press, Inc, Harcourt Bruce Janovitch, Orlando, San Diego, New York, Austin, London, Montreal, Sydney, Tokyo, Toronto, 1986

Van Roy, P., Armstrong, J., Flatt, M. and Magnusson, B., *The Role of Language Paradigms in Teaching Programming*, Proceedings of the 34th SIGCSE technical symposium on Computer science education, ACM SIGCSE Bulletin , Vol 35 No 1, pp 269-270, January, 2003, ACM Press, New York, NY, USA , 2003, Available from: http://portal.acm.org/citation.cfm?id=611908&jmp=cit&coll=GUIDE&dl=ACM&CFID=22782190&CFTOKEN=87597448#CIT , [Accessed June 15th 2004]

Ventura, P., *Turtle Source Code*, Department of Computer Science and Engineering, University at Buffalo, The State University of New York, Buffalo, NY, USA, FA2001, November, 2000, Available from: http://www.cs.buffalo.edu/faculty/pventura/Courses/FA2001/115/ SourceCode/Turtle/Turtle.java [Accessed 28th August 2003]

Walrath, K, Campione, M., Huml, A., and Zakhour, S., *Swing Tutorial*, From "The JFC Swing Tutorial", Addison-Wesley Pub Co; 1999, Available from: http://java.sun.com/docs/books/tutorial/uiswing/mini/fifthexample.html, [Accessed July 25th 2003]

Weston, D., *The Second Logo Book: Advanced Techniques in Logo*, Scott, Foresman & Co, Glenview, IL, USA, 1985

Weissinger, M., Sentilles, I., Collison, C., Ryan, G., Vorhaus, D., and Clifton, T., *Turtle Source Code,* April, 2001, CPS108 Final Project, Computer Science Department, Duke University, Durham, NC, USA, Available from: http://www.duke.edu/~tmc6/final108/logofat/code/Turtle.java, [Accessed 26th August 2003]

Wilcock, G., Kench, D. and Pournara, C., *Exploring Java Level One Workbook*, Holt Software Associates Inc, Ontario, Canada, 2002

Winder, R. and Roberts, G., *Simple Java I/O*, October, 2003, in Developing Java Software, John Wiley & Sons, Chichester, UK, 1998, Available from:http://www.cs.ucl.ac.uk/staff/g.roberts/code/ keyboardinput.html, [Accessed July 5th 004]

Winer, B.J., *Statistical Principles in Experimental Design*, McGraw-Hill, NewYork, NY, USA, 1962

Winograd, T. and Flores, F. *Understanding Computers and Cognition*, Ablex Publishing Co., Norwood, NJ, USA, 1987

Wolz, Ursula *Turtle.java*, Department of Computer Science, The College of New Jersey, Ewing, NJ, USA, Class Lab, CMSC 250, November, 2002, Available from: http://www.tcnj.edu/~wolz/CMSC250/Assignments/TurtleStuff/recursion2.html [Accessed 28th August 2003]

Wolz, U. and Koffman, E., *simplelo: A Java package for Novice Interactive and*

Graphics Programming, ACM SIGCSE Bulletin , Proceedings of the 4th annual SIGCSE/SIGCUE ITiCSE conference on Innovation and technology in computer science education, Vol 31 No 3, pp139-142, June 1999, Available from: http://portal.acm.org/citation.cfm?id=305963&jmp=cit&coll=portal&dl=ACM&CFID=22994987&CFTOKEN=787634#CIT, [Accessed June 22nd 2004]

Zhu, H. and Zhou, M., *Methodology First and Language Second: A Way to Teach Object-Oriented Programming,* Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, pp 140-147, October, 2003, ACM Press, New York, NY, USA , 2003, Available from: http://portal.acm.org/results.cfm?coll=portal&dl=ACM&CFID =22777812&CFTOKEN=91008658, [Accessed June 15th 2004]

Ziegler, U. and Crews, T., *An Integrated Program Development Tool for Teaching and Learning How to Program*, The proceedings of the thirtieth SIGCSE technical symposium on Computer science education, ACM SIGCSE Bulletin, Vol 31 No 1, pp 276-280, March, 1999, ACM Press, New York, NY, USA , 1999, Available from: http://portal.acm.org/citation.cfm?id=299786&jmp=cit&coll=portal&dl=ACM&CFID=22937284&CFTOKEN=24337902#CIT, [Accessed June 13th 2004]