VOL. 3, NO. 5, May 2012 ISSN 2079-8407
Journal of Emerging Trends in Computing and Information Sciences

http://www.cisjournal.org

Analyses of Student Programming Errors In Java Programming Courses

Ioana Tuugalei Chan Mow
Computing Department.Faculty of Science
National University of Samoa
Apia.Samoa
Email: i.chanmow@nus.edu.ws

ABSTRACT

Computing students' difficulties in understanding Java programming provided a practical context for a critical investigation of why this continues to exist. An exploratory study was conducted by Computing department staff at the National University of Samoa to investigate what were the most common errors students made in Java programming classes. Program code from undergraduate Java programming classes were analysed for errors. Errors were categorized and the frequency of occurrence also tabled. Results of the analyses were used to form recommendations to inform course development and improve teaching practices.

Keywords: *Programming errors; Java; syntax; semantic; logical errors*

1. INTRODUCTION

Teaching computer programming skills within the university context has proven to be a difficult and challenging task [1] and Computer science educators have shown growing concern over the difficulties with which novice Computer programmers learn programming principles. Computer programming is a difficult and challenging subject area which places a heavy cognitive load on students [2]; [3]. Research has found that even after 2 years of learning programming, most novice programmers are still struggling to be proficient [3];[4]. According to [5], it takes 10 years for a novice programmer to become proficient.

Most novice programmers have had no programming background knowledge. Programming requires students to hold a wide range of information in working memory. These include the details of syntax and semantics specific to the programming language being used, some mental model of how to solve each problem, and the ability to differentiate between *solving the problem* and *specifying the solution* [6]. Computer programming also requires that the user be proficient in the use of (a) the development environment, (b) the programming language, and (c) compiler/interpreter, which are separate levels of the programming interface that the user must master in order to program successfully [6]. Consequently, these demands would impose a heavy cognitive load on the student, making the learning of programming a complex and cognitively challenging task. With the cognitively challenging nature of computer programming, one avenue to improve the teaching of introductory programming is by analyzing programming errors, novice programmers make.

Analyses of novice programming errors can help improve the teaching of novice programming in various ways. Motivation to undertake this study arose from the need to improve the teaching of programming and also problem solving strategies used by novice programmers. Findings from these analyses can identify the most problematic areas which students need to bring attention to. This would also help in allocation of time for topic coverage devoting the most time and attention to topics and areas students have the most difficulties with. It may even be feasible to use such information to identify at risk students at an early stage and provide targeted intervention.

This paper describes a study on novice programming errors in Java programming classes at the National University of Samoa (NUS). The study aimed at identifying common programming errors made by students in the Java programming courses and to analyse these errors to identify problematic areas which curriculum need to focus on. The research attempted to answer the question "What are the common types of programming errors students make based on the categorization as syntax, semantic and logical errors?"

Computer programming courses at NUS are taught as part of the undergraduate programs in Computing (certificate, diploma and bachelors). Except for one course in Visual basic, the rest of the courses are in Java programming. There are 4 Java programming one-semester courses in the undergraduate programming strand: HCS181, HCS281, HCS286 and HCS381. The prerequisite to HCS181 is HCS081 which contains a section on Java programming. The HCS081 Java section provides an introduction to programming concepts, introduces the integrated developer environment (IDE) JBuilder (java program editor, compiler and debugger), teaches simple java applications using projects, classes, methods and attributes, basic sequential and conditional executions. HCS181

provides an introduction to programming concepts, link to software development, introduces the integrated developer environment (IDE) JBuilder (java program editor, compiler and debugger), UML modeling, teaches simple java applications using projects, classes, methods and attributes, arrays, sequential, conditional and loop executions. HCS281 is the sequel to HCS181 which includes conditional and loop executions, exception handling, reading/writing from keyboard, reading/writing text files. HCS286 which is the sequel to HCS281 continues with exception handling, reading/writing from keyboard, reading/writing text files, access levels and then introduces inheritance, static objects, polymorphism, overriding, arraylists, linked lists, queues and stacks, hash tables, sorting, searching and binary trees. HCS381 the final Java course extends concepts learnt in HCS286 and students also focus on programming projects and developing applications.

A considerable number of studies have been conducted on the difficulties of novice programmers in learning programming [2]; [7 - 12]. The rationale has been that if we can understand the process of learning a first programming language, then we can create more effective learning environments. Furthermore, an analysis of programming problems and errors could inform teaching practice in terms of teaching approaches and time allocations of topics.

Various studies [13 - 15] have shown that there are two challenges which impact most on learning programming. The first, program formulation, is in essence the "notation" referred to by [16] and the "language constructs" of [14]. It is the syntax and semantics of the individual program parts that are crucial to the formulation of any program. The second is problem formulation, which is the concept of "structures" from [16] and the "plan compositions" of [14]. Problem formulation is the challenge of identifying and structuring the necessary program components to formulate a programmed solution to a specific problem [13].

[17] Conducted a survey on topics or areas of difficulty in teaching Java, to determine the effectiveness of specific pedagogical initiatives undertaken (i.e. interactive whiteboard, online notes, and live coding of programs by the lecturer). Results of surveys indicated that the most problematic topics to students were the following: arrays, threads, polymorphism, looping exceptions and inheritance. Online notes (Compweb) were rated as the most effective of the pedagogical initiatives. Furthermore, the biggest obstacles students faced were the complex nature of Java, and problems with technical access.

Another study by [18], based on an international survey of opinions from more than 500 students and teachers, evaluated the difficulties experienced and perceived when learning and teaching programming. The most difficult issues in programming as reported by students were: (a) understanding how to design a program to solve a certain task; (b) dividing functionality into procedures; (c) and finding bugs from their own programs. These are all issues where the student needs to understand larger entities of the program instead of just some details about it. In addition, the most difficult programming concepts were recursion, pointers and references, abstract data types, error handling and using the language libraries. It needs to be noted that, error handling requires understanding the program comprehensively. Using the language libraries requires independent searching of the information, which can make it difficult for the novices. Recursion, pointers and references, and abstract data types are abstract concepts and thus cognitively complex to understand without a similar phenomenon in the daily life for comparison. The teachers' opinions on the most difficult course contents were almost the same as the students. In addition, the teachers perceived understanding programming structures as difficult, in issues about programming.

Another study of novice programmers [10] was conducted in an introductory Java course at the Otago Polytechnic. As part of the study, all of the problems encountered by students during the laboratory sessions were recorded, while students worked through a series of 25 labs for the course. Students were supervised by a number of demonstrators who provided help on any problems upon the request of students. These problems were recorded, analysed and classified against a list of problem definitions. The results showed that the commonest problems encountered involved: (a) basic mechanics; (b) program design; (c) problems with basic program structure; and (d) understanding the task. The results were consistent with trends noted in the literature, and highlight the significance of both fundamental design issues and the procedural aspects of programming. Different problem distributions were observed for high and low performing students.

An important tool which not only locates the source of errors in the program code but also aids program comprehension is the debugger. A study by [7] on the use of the debugger introduced many activities which included debugging exercises, debugging logs, development logs and reflective memos, and collaborative assignments. Results of the study indicated that: (a) formal training in debugging helps students develop skills in diagnosing and removing defects from computer programs; and (b) students who completed the optional debugging exercises spent significantly less time on debugging their programs than those who did not. Furthermore, they developed a model of debugging abilities and habits based on students' comments in their debugging logs, development logs, reflective memos, and evaluation surveys. Such a model could be used by students and educators to diagnose students' current debugging skills and take actions to enhance their skills.

In Samoa, the only study on computer programming was conducted by [1] in which over a three-year period, an instructional program for teaching computer programming was developed, and referred to as CABLE (Cognitive Apprenticeship Based Learning Environment).

From this investigation, a learning environment CABLE was designed which made use of cognitive apprenticeship, collaborative learning, meta-cognition, and technologies through the use of tele-apprenticeship and online or computer-mediated communication (CMC). From the field trials of CABLE, it was found that: (a) CABLE provided a viable instructional model which could be introduced into the normal conduct of university programming courses, (b) students exposed to CABLE evidenced increased achievement on Java programming scores relative to those taught in the traditional mode, (c) there were no differences in student attitudes towards the learning environment, between students taught with CABLE, and those taught in the traditional university mode, and (d) students taught under CABLE reported higher levels of mental engagement when compared to students taught via traditional mode. Students taught programming in CABLE showed positive attitudes towards the collaborative elements and also towards the online learning elements of CABLE.

The study by [1] evaluated the effectiveness of CABLE as a teaching environment for programming within a university context, but did not investigate specifically the types of errors students make in programming. The current study proposes to do this and thus contributes to improving the teaching of programming at the university by providing data on problematic areas within the curriculum which need urgent attention and targeted assistance.

2. METHODOLOGY

The current study consisted of analyses of computer programs from 3 levels of Computer Science undergraduate programming courses HCS181, HCS281 and HCS286 at the National University of Samoa. The 3 courses were taught using 4 lecture /tutorial hours weekly and were 1 semester in duration. Assessments for these courses were in the form of programming tasks, tests and homework. The programming language used in these courses was Java and the integrated developer environment (IDE) used was JBuilder. For the current study, 2 sets of programs were used in the analyses for each course. Programs from 25 HCS181 students, 28 HCS281 students and 15 HCS286 students were used in the analyses. Copies of these assessment tasks appear in Appendices 1 to 3. Computer programs from the 3 classes were loaded into JBuilder and from compilation and running of the programs, the errors generated by the compiler were logged and categorized according to the type of errors which emerged. Samples of the compiler window for JBuilder and compiler messages (these appear in red) appear in Figure 1.

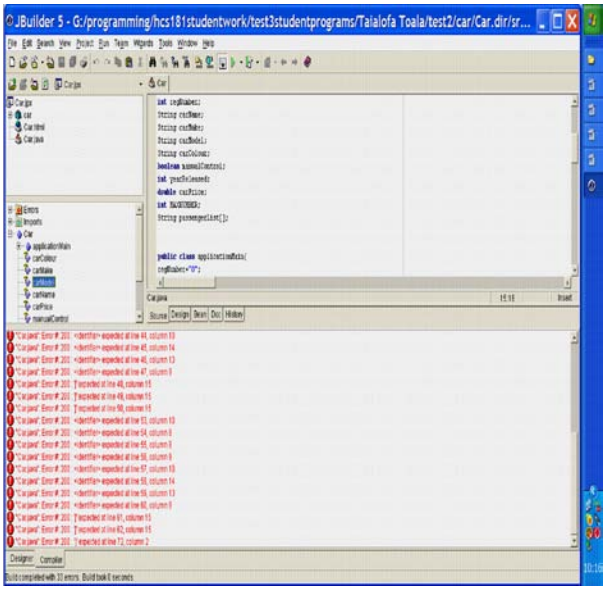

Figure 1: Jbuilder Compiler window with error messages in red

3. ANALYSES AND RESULTS

The frequency distribution of programming errors for each of the 3 courses were generated. The list of categorized errors for each level appear in the tables below. As mentioned earlier, HCS181 provides an introduction to programming concepts, link to software development, introduces the integrated developer environment (IDE) JBuilder (java program editor, compiler and debugger), UML modeling, teaches simple java applications using projects, classes, methods and attributes, arrays, sequential, conditional and loop executions. Analyses of HCS181 errors revealed the following. Of the 171 different types of error encountered, the five most common errors accounted for 90% of all errors generated by students while programming. These are: variable not found (59.1%), identifier expected (13.5%), class not found (7%), mismatched brackets (6.4%), and invalid method declaration (7%). The results appear in Table 1 below.

TABLE I: SUMMARY OF ERRORS FOR HCS181

Programming error	error frequency	%
Variable not found	101	59.1
Identifier expected	23	13.5
Class not found	12	7.0
Mismatched brackets/parentheses	11	6.4
invalid method declaration	7	4.1
Illegal start of type	5	2.9

Method not found	4	2.3
; expected	2	1.2
: operator < cannot be applied	1	0.6
not an expression statement	1	0.6
duplicate definition of variable	1	0.6
: 'class' or 'interface' expected	1	0.6
: cannot access class; neither class nor source found	2	1.2
Total	171	100

HCS281 is the sequel to HCS181 and topics covered include conditional and loop executions, string functions and string handling, parsing text, exception handling, reading and writing from keyboard, reading and writing text files. From the analyses of the 80 errors generated, the 5 most common errors which account for 71.25% of all errors generated were: identifier expected (27.5%), variable not found (22.5%), illegal start of type (7.5%), mismatched bracket/parentheses (7.5%) and invalid method declaration (6.3%). The results of the analyses appear in Table 2.

TABLE II: SUMMARY OF ERRORS FOR HCS281

Try without catch	1	1.3
java lang. Array Index Out Of Bounds Exception	2	2.5
operator * cannot be applied to (<any>, int[])	1	1.3
malformed declaration	1	1.3
possible loss of precision	1	1.3
unclosed character literal	1	1.3
Method not found	1	1.3
malformed expression	1	1.3
Total	80	100

HCS286 which is the sequel to HCS281 continues with an extension of topics covered in HCS281 such as exception handling, reading and writing from keyboard, reading and writing text files, access levels and then introduces inheritance, static objects, polymorphism, overriding, arraylists, linked lists, queues and stacks, hash tables, sorting, searching and binary trees. The analyses revealed that of a total of 70 errors generated, the 5 most common errors which accounted for 81.4% of all errors generated were: variable not found (58.6%), constructor not found (7.1%), arraytype required (5.7%), Nosuchelement exception (5.7%) and returning a value from method whose result type is void (4.3%). The results of the analyses appear in Table 3.

TABLE III: SUMMARY OF ERRORS FOR HCS286

Programming error	error frequency	%
Variable not found	41	58.6
Constructor not found	5	7.1
Arraytype required	4	5.7
java.util.NoSuchElementException	4	5.7
can't return a value from method whose result type is void	3	4.3
object type required	2	2.9
malformed expression	2	2.9
system cannot find the path specified	2	2.9
does not define any classes or interfaces	1	1.4
check sourcepath; source	1	1.4
;expected	1	1.4
incompatible types; found	1	1.4
Class not found	1	1.4
Duplicate definition of variable	1	1.4
illegal start of expression	1	1.4
Total	70	100

The errors generated across the 3 courses (HCS181, HCS281 and HCS286) were combined in the various categories and a summary of the top 8 errors across the 3 levels appear in Table 4. The 8 most common errors were: variable not found (49.8%), identifier expected (14%), class not found (5%), mismatched brackets/parentheses (5.3%), invalid method declaration (3.7%), illegal start of type (3.4%), expected (2.2%) and method not found (1.6%).

TABLE 4: SUMMARY OF THE TOP 8 ERRORS ACROSS THE 3 LEVELS HCS181, HCS281 AND HCS286

Programming error	HCS181	HCS281	HCS286	Total	% of Total
Variable not found	101	18	41	160	49.8
Identifier expected	23	22	0	45	14.0
Class not found	12	3	1	16	5.0
Mismatched brackets/parentheses	11	6	0	17	5.3
Invalid method Declaration	7	5	0	12	3.7
Illegal start of type	5	6	0	11	3.4
Method not found	4	1	0	5	1.6
Expected	2	4	1	7	2.2

4. DISCUSSION

An inspection of the cause of these programming errors was carried out by looking at the context in which these errors occurred in the program. The analyses pointed to the following as underlying causes. Most of the simple syntax errors were due to carelessness of students. Unknown variable errors such as variable not found which accounted for 49.8% of errors were generated by typographic errors or by students failing to declare a variable. Identifier expected errors were generated for similar reasons, as well as failing to import a package containing the class in question. Class not found errors were often caused by either failing to import a package containing the class or confusion between a class and other data structures in the program. Mismatched bracket errors included all types of brackets (‘( )’, ‘{ }’, and ‘[ ]’) and caused by failure to match the open and close brackets or parentheses. Invalid method declaration errors were mainly caused by failure to include the return data type in the declaration. Illegal start of expression errors were caused by bracketing and missing semicolons. Method not found and invalid method declaration errors were caused by typographic error, omitting the “( ) “after the method name, incorrectly inserting a semicolon after the method name or failing to declare or include the method in the program.

Programming errors can be categorized as syntax, semantic and logic [19]. A syntax error is an error due to incorrect grammar. Syntax errors are often detected by a program called a compiler, if the language is a compiled language such as Java. A semantic error is an error due to misuse of a programming concept, despite correct syntactic structure. Semantic errors are caught when the program code is compiled. A logic error occurs when the program does not solve the problem that the programmer meant for it to solve.

Categorization of errors of the present study into syntax, semantic, runtime and logic revealed that syntax errors made up 94.1 %, semantic errors 4.7% and logic errors 1.2%. The categorization of errors and the relative percentages appear in Table 5. Hence errors in the current study were predominantly syntax errors (94.1%). This is consistent with findings of similar studies by [2] and [4].

Semantic errors made by students in the current study included: a) non-static method printBools() cannot be referenced from a static context = 3,b) arrayindex out of bounds exception, c) java.utils NoSuchElement exception, d) 'class' or 'interface' expected, e) : cannot access class; neither class nor source found, and f) system cannot find the path specified. Non-static method cannot be referenced from a static method error is mainly caused by trying to use a class property "statically" without instantiating an instance of that class. Arrayindexout of bounds exception is mainly caused by trying to access an array element beyond the declared size of the array. NoSuch element exception is caused by trying to access an element which does not exist. Class or interface expected errors occur when there is code outside of a class declaration and usually caused by declaring a few objects at the start of the class, and mistakenly placing them before the class has started.

Logic errors made by students in the current study were: a) possible loss of precision and b) can't return a value from method whose result type is void. Possible loss of precision errors were mainly caused by storing data of type double into a variable of type integer. Can’t return a value from method whose result type is void is caused by students trying to return a value from a method declared to be void.

The current study has highlighted the common errors students in the 3 courses make while programming. These errors are predominantly syntactic and seem to arise mostly from careless typographic errors. Although this study is exploratory and is a preliminary stage of an investigation into student programming errors, the common errors highlighted in this study need to be taken into account in

improving teaching. Strategies to remedy these errors need to be employed in future work and could be the subject of future research. Such strategies were recommended in [4] which evaluated the CABLE learning environment for teaching Java programming. To resolve syntax and semantic errors the following techniques were recommended:1) Teaching only a subset of the language, and then gradually including the full language together with the use of contextualised examples of practical programs; 2) Make students aware of the limits of analogies, by highlighting and correcting misconceptions during the course of programming instruction;3) Emphasising and highlighting common errors; 4) Supporting role expressiveness by explaining the meaning of each syntactic term in a language; 5) Instructing novice programmers in the use of secondary notation in order to enhance the readability and comprehensibility of programs, and 6) Use of scaffolding, coaching, and modelling to help students master intellectually challenging operations.

Another aspect that needs to be looked at are the main reasons of why students are making these programming errors. As pointed out in [14], computer programming is a challenging task which places a heavy cognitive load on the student and requires that the user be proficient in the use of (a) the development environment, (b) the programming language, and (c) compiler/interpreter. Furthermore, programming requires students to hold a wide range of information in working memory. These include the details of syntax and semantics specific to the programming language being used, some mental model of how to solve each problem, and the ability to differentiate between *solving the problem* and *specifying the solution* ([20]). The formation of these mental models of what the problem is and the translation into plans which specify the solution has been found by as one of the main reasons why students struggle with programming [2];[3];[12].

To alleviate these difficulties, further recommendations in terms of instructional techniques as stated in [1] include :1) Repetition of examples illustrating the concepts vital for the construction of viable mental models; 2) The use of examples, partial solutions, and visualisation techniques to reduce cognitive overload; 3) Emphasis on the need for precision by extensive modelling, coaching, scaffolding, and the use of a debugger;4) The model should address 'orientation' by incorporating a good introductory section on programming, and its different types; 5) Programming should be related to its application in software development; 6) Instruction in computer programming must also include training in file management, editing, compiling, and debugging.

TABLE 5: BREAKDOWN OF ERRORS INTO SYNTAX, SEMANTIC AND LOGIC CATEGORIES

Error category	Frequency	% of total
Syntax	302	94.1
Semantic	15	4.7
Logic,	4	1.2
Total	321	100

5. CONCLUSION

The study is a preliminary and exploratory investigation into novice programming errors and further research is needed. The current study summarized common programming errors at the level of courses and overall class but did not look at errors at the level of the individual student. The study ascertained common programming errors but did not look at the time taken by students to fix their errors. Such information is important in finding out which errors students were struggling with the most. Further research is also needed to trial and gauge the effectiveness of solutions recommended from the current study and the earlier study in [1]. The current study categorized programming errors based on 3 categories. However, other forms of categorization of programming errors are found in the literature [21 - 22] and it is not known whether such categorizations are more useful and informative or what kind of information these other categorization of errors can provide. The current study did not provide information on topics of difficulty in Java programming. A study on student perceptions of their programming experience should provide in-depth information on areas or topics students have difficulty with and what assistance is perceived by them as necessary for improvement in their learning environment and ultimately in their programming skills.

The results from the current study will be used to refocus our course content and teaching approaches to include clear and specific solutions for the errors identified in this study. This will also be the subject of further study. It is hoped that after implementing these changes that they would have a positive effect on student programming skills at the National University of Samoa.

REFERENCES

[1] I.T. Chan Mow. "The Effectiveness of Cognitive Apprenticeship based Learning Environment (CABLE) in Teaching Computer Programming". Unpublished PHD dissertation, University of South Australia, 2006.

[2] M. Ahmadzadeh, D. Elliman, and C. Higgins, "An analysis of patterns of debuggi among novice computer science students," in ITiCSE 2005:

Proceedings of the10th annual SIGCSE conference on Innovation and technology in computer scienceducation. NewYork,NY,USA: ACMPress, pp.84–88, 2005.
Retrieved from http://dx.doi.org/10.1145/1067445.1067472, June 2011

[3] T. Flowers, C. A. Carver, and J. Jackson, "Empowering students and building confidence in novice programmers through gauntlet," in Frontiers in Education Conference, vol. 1. ASEE/IEEE, October 2004, pp. T3H/10–T3H/13

[4] AECT. The handbook of Research for Educational Communications and Technology, 2001. Retrieved June 12, 2011 from
http://www.aect.org/intranet/publications/edtech/24/24-05.html

[5] L.E. Winslow "Programming pedagogy- A psychological overview". SIGCSE Bulletin, Vol 28, 17–22, 1996.

[6] J.F.Pane & B.A. Myers. "Usability Issues in the Design of Novice Programming system", Carnegie Mellon University, School of Computer Science Technical Report CMU-CS-96-132, 1996, Pittsburgh, PA, August 1996.

[7] R. Chmiel & M.C.Loui. "Debugging: from Novice to Expert" : Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education, SIGCSE 2004, Norfolk, Virginia, USA, March 3-7, 2004. ACM 2004, ISBN 1-58113-798-2, 2005.

[8] A. Ebrahimi. "Novice programmer errors: language constructs and plan composition". International Journal of human-Computer Studies, Vol 41, pp. 457-480, 1994.

[9] S. Garner " Cognitive load reduction in problem solving domains", Edith Cowan University, 2000.

[10] S.Garner, P. Haden. & A. Robins. "My program is correct but it does not run. Teaching students how to be computer scientists through filling in the gap ", Programming Exercises Proceedings of the Seventh Australasian Computing Education Conference (pp.173 – 180). Newcastle, Australia: ACS, pp. 173 – 180, 2005.

[11] M. Hristova, A. Misra, M. Rutter, R. Mercuri: "Identifying and Correcting Java Programming Errors for Introductory Computer Science Students." Proceedings of the 34th SIGCSETechnical Symposium on Computer Science Education, Reno, Nevada, USA, ACM Press, vol 34, pp153- 156, 2005.

[12] M. C. Jadud, "A first look at novice compilation behavior using bluej," in 16th Annual Workshop of the Psychology of Programming Interest Group (PPIG 2004). Carlow, Ireland: Institute of Technology, April 2004. [Online]. available:http://www.jadud.com/people/mcj/files/2004-PPIG-flcbBlueJ.pdf

[13] N. Coull, I. Duncan, J. Archibald, and G. Lund, "Helping novice programmers interpret compiler error messages," in 4th Annual LTSN-ICS Conference. National University of Ireland, Galway, August 2003, pp. 26–28

[14] Du Boulay."Some difficulties of learning to program". In Soloway & Spohrer: Studying the Novice Programmer, pp. 283-300, 1989.

[15] L. McIver. "Syntactic & Semantic Issues in Introductory Programming education". Unpublished doctoral dissertation, Monash University, January, 2001.

[16] E.Soloway & K. Ehrlich. "Empirical Studies of Programming Knowledge" IEEE Trans. on Software Engineering SE-vol 10, No 5, pp595-609, 1984

[17] E. Hyland & G. Clynch. "Initial experiences gained and initiatives employed in the teaching of java programming in the Institute of Technology". Proceedings of Conference on Principles and Practice of Programming in Java, Dublin: ACM pp.101-106, 2002.

[18] E. Lahtinen, K. Ala-Mutka, and H.M. Jarvinen. "A study of the difficulties of novice programmers ", ACM SIGCSE Bulletin. - New York, NY, USA : ACM Press, - 3 : Vol.37. pp. 14--18, 2005.

[19] S.M.Thompson. "An Exploratory Study of Novice Programming Errors and Experiences". Unpublished MSc dissertation. Victoria University 2004.

[20] D.G.Moursound. "Increasing your expertise as a problem solver: Some roles of computers". Eugene, OR: ISTE. Copyright (C) David Moursund 2002. Retrieved August 11th, 2004 from

http://www.uoregon.edu/~moursund/PSBook1996/chapter_9.htm.

[21] Naidoo, R, & Ranjeeth, S, " Errors made by Students in a Computer
Programming Course" Proceedings of the Computer Science and IT
Education Conference 2007

[22] W,F.J.Cheng. "Teaching and Learning to Program: A Qualitative study of Hong Kong sub degree students", Unpublished PHD dissertation, University of Sydney, 2010.

.

APPENDICES

Appendix 1: programs for hcs181

Program 1

Purpose

The purpose of this software application is to print out information of car objects to the screen that the customer has requested…

Background

No background

Object information

A Car object includes a regNumber which is an int
A Car object includes a carName which is a String
A Car object includes a carMake which is a String
A Car object includes a carModel which is a String
A Car object includes a carColour which is a String
A Car object includes a manualControl which is a boolean
A Car object includes a yearReleased which is an int
A Car object includes a carPrice which is a double
A Car object includes a MAXNUMBER which is an int = 4
A Car object includes an passengerList which is a String[ ]
A Car object includes an getcarName() method
A Car object includes an setcarName() method
A Car object includes a printInformation() method
Car is in hcs181Test2Sem12006 package
An applicationMain object includes a numberOfCars which is an int
An applicationMain object uses a Car1 which is a Car
An applicationMain object uses a Car2 which is a Car
An applicationMain object uses a Car3 which is a Car
An applicationMain object includes a constructor method ApplicationMain()
An applicationMain object includes a main() method which is used to start the java application software
applicationMain is in hcs181Test2Sem12006 package

Program 2

JBuilder Project name: hcs181Program4

Root path:
w:/hcs181/hcs181Programs

JBuilder Project name: hcs181Program4DIR

Class Name: Employee
ApplicationMain

Purpose

The purpose of this software application is to print out information of employees to the screen that the customer has requeste

- **Provide UML diagrams**

REQUIREMENTS DOCUMENT

Object Information

An Employee object includes an employeeCode which is an int
An Employee object includes a firstName which is a String
An Employee object includes a lastName which is an String
An Employee object includes an employeeAge which is an int
An Employee object includes an employeeSalary which is a double
An Employee object includes an employeeGender which is a char
An Employee object includes a MAXNUMBER =
An Employee object includes an favouriteFoodList which is a String[ ]

An Employee object includes a getemployeeCode() method
An Employee object includes a setemployeeCode() method

An Employee object includes a printInformation() method
Employee object is in the hcs181Program4 package
An ApplicationMain object includes a numberOfEmployees which is an int
An ApplicationMain object uses a Emp1 which is an employee
An ApplicationMain object uses a Emp2 which is an employee
An ApplicationMain object uses a Emp3 which is an employee
An ApplicationMain object includes a main() method which is used to start the java application software
ApplicationMain object is in the hcs181Program4 package

- **Employee methods**
  - public Employee() – a constructor method that initializes the values of Employee objects' attributes to the following
    - employeeCode=0
    - firstName="No name"
    - lastName="No Name"
    - employeeAge=0
    - employeeSalary=0.0
    - employeeGender='m'
    - instantiate an favouriteFoodList array with MAXNUMBER elements
    - use a for loop to assign favouriteFoodList elements as following:
      - favouriteFoodList [0]="Food0"
      - favouriteFoodList [1]="Food1"
      - etc

      (Remember you must use a for loop)
  - public void printInformation() – this method prints out all the values of the employee object attributes
  - public int getemployeeCode() – getter method for the employeeCode
  - public void setemployeeCode(int newId) – setter method for employeeCode

- **ApplicationMain methods**
  - public ApplicationMain() – do the following
    - print to stdout "start: constructor: applicationMain"
    - instantiate Emp1
      - set the value of Emp1.firstName equal to the value "Aurora"
      - set the value of Emp1.lastName equal to the value "Elisaia"
      - using appropriate setter method, assign employeeCode=11111
        hint: Emp1.setemployeeCode(11111);
      - set the value of Emp1.employeeAge equal to the value 25
      - set the value of Emp1.employeeSalary equal to the value 26000.00
      - set the value of Emp1.employeeGender equal to the value 'F'
      - set the value of Emp1. favouriteFoodList [0] equal to the value "Chicken"
      - set the value of Emp1. favouriteFoodList [1] equal to the value "Burger"
      - set the value of Emp1. favouriteFoodList [3] equal to the value "Sashimi"
    - create a local variable named temp, with datatype int; set the value of temp equal to the value returned from Emp1.getemployeeCode()
    - instantiate Emp2
      - set the value of Emp2.firstName equal to the value "Edna"
      - set the value of Emp2.lastName equal to the value "Temese"
      - using appropriate setter method, assign employeeCode=22222
      - set the value of Emp2.employeeAge equal to the value 26
      - set the value of Emp2.employeeSalary equal to the value 25000.00
      - set the value of Emp2.employeeGender equal to the value 'F'
      - set the value of Emp2. favouriteFoodList [1] equal to the value "Fish"
      - set the value of Emp2. favouriteFoodList [2] equal to the value "Banana"
      - set the value of Emp2. favouriteFoodList [4] equal to the value Emp1.employeeFavouriteFoodList[2]
      - instantiate Emp3
        - set the value of Emp3.employeeCode equal to the value of temp * 3
      - set the value of numberOfEmployees equal to value 3
      - print to stdout "end: constructor: applicationMain "
  - public static void main(String args)
    - print to stdout "start: main"

- create a local variable named ApplicationMain1, with datatype ApplicationMain; instantiate ApplicationMain1
- invoke ApplicationMain1.Emp1.printInformation() method
- invoke ApplicationMain1.Emp2.printInformation() method
- invoke ApplicationMain1.Emp3.printInformation() method
- print to stdout “end: main”
-

Appendix 2: programs for hcs281

Program 1

Write the program code for the following: [50 marks]

Project name: arrayExercise

Package name: arrayExercise

Class name: SomeArrayExamples

Attributes: Five arrays as follows:
String[] strings;
int[] integers;
float[] floats;
boolean[] booleans;
Byte[] bytes;

Constructor: Write a constructor that initializes the arrays all to have four elements. Initialize the elements of all of the arrays.

Main Method: Instantiate one instance of SomeArrayExamples and call each method below after you create it.

Other Methods:

1. Write a method called "concStrings" that concatenates all the strings into one string.
2. Write a method called "printBools" that prints each boolean value separated by spaces all on the same line.
3. Write a method called "avgInts" that prints out the average of all of the integers.
4. Write a method called "multFloats" that multiplies all of the floats together and prints the result.

Program2

Write a program that has a class called "tryingExceptionsMain"[50marks]

It will have a private integer called result

It will have a private String called textVersionOfNumber
It will have a constructor and a static main method (like normal)
Inside the constructor, first initialize textVersionOfNumber to "3.2"
After this, enter the line:
result = Integer.parseInt(textVersionOfNumber);

Your goal is to make the program work using exceptions. Do not change the string value of “3.2”. You need to fix it by using try and catch. The program should print out the text:
"The number <textVersionOfNumber> was not a properly formatted integer."
when parseInt() throws the exception that is for that purpose. <textVersionOfNumber> is whatever was in the variable textVersionOfNumber. The program should continue to work for any value put into this string variable, whether it’s a valid string representation of an integer or not.
The program should print “The number was a good integer.” if no exception is thrown.

Appendix 3: programs for hcs286

Program 1

Due Date: Thursday, March 1st, 2007

Purpose:

This program will show your knowledge of the following areas

- Reading Text Files
- Parsing Data using the StringTokenizer class
- Using Conditional (if) statements

Your program will read a text file which contains one or more lines. Each line will have either a “+” , “-“, “*”, or “/” symbol, followed by two numbers. These three “tokens” will be separated by tab characters. Here is an example:

-	32	4
+	12	9
+	3	82
/	36	4
*	3	50
+	23	45

Your program must read each line from the file and perform the correct mathematical operation based on the given operator. You must then output the full equation along with the solution.

Given the above text file, your output should be:

```
32 - 4 = 28
12 + 9 = 21
3 + 82 = 85
36 / 4 = 9
3 * 50 = 150
23 + 45 = 68
```

Note: This is just an **example**! Your program should work with any input file that matches the above specifications.

Program 2

Objective: In this project, you will create a class that stores a 3 by 3 matrix and allows the user of the class to add matricies, multiply matricies, and multiply a matrix by a constant. It will be worth one project grade and you should work on this project on your own.

5. Create a project called "matrixMath" on your student drive.
6. Create an ApplicationMain class with a main method.
7. Create a class called Matrix that contains a private member called myMatrix that is a 2D array of integers. It will hold a 3x3 matrix and thus 9 values total.
8. Create a constructor for the Matrix class that allocates the 2D array.
9. Create another constructor for the Matrix class that allocates the 2D array and accepts a 2D array of integers as its only parameter. It will use this array to initialize the internal array.
10. Create a method in the class Matrix that is called "multiply" that accepts a Matrix as its only input. It will return a Matrix. You will need a local variable in this method that is of type Matrix. This will be the hardest method to write and will be extra credit.
11. Create a method in the class Matrix that is called "multiply" that accepts an integer as its only input. This is an example of overloading this method name! It will return a matrix. This method will require a local variable of type Matrix as well.
12. Create a method in the class Matrix that is called "add" that accepts a Matrix as its only input. It will return a Matrix. This method will require a local variable of type Matrix too.
13. Create a method called "print" that prints out the matrix to standard output.
14. Using the Matrix class you have created, print to the screen the results of the following equations:

$$\begin{bmatrix} 3 & 4 & 5 \\ 10 & 8 & 10 \\ 11 & 18 & 31 \end{bmatrix} \square \begin{bmatrix} 3 & 4 & 6 \\ 2 & 8 & 10 \\ 2 & 8 & 3 \end{bmatrix}$$

$$\begin{bmatrix} 3 & 2 & 14 \\ 1 & 8 & 10 \\ 6 & 8 & 10 \end{bmatrix} \times \begin{bmatrix} 3 & 4 & 1 \\ 8 & 34 & 10 \\ 5 & 4 & 5 \end{bmatrix}$$

$$15 \begin{bmatrix} 3 & 7 & 7 \\ 2 & 8 & 10 \\ 10 & 8 & 10 \end{bmatrix}$$