A Course Portfolio for CS-141, Principles and Practice of Programming.

Computer Science Department
University of Wales
Swansea

C. J. Whyley
June 2006

Contents

1 Introduction

Between October 2005 and June 2006 I was involved in a Disciplinary Commons in the Teaching of Introductory Programming, a programme whose aim was:

- To document and share knowledge about teaching and student learning on introductory programming courses in the UK.
- To establish practices for the scholarship of teaching by making it public, peer-reviewed, and amenable for future use and development by other educators: creating a teaching-appropriate document of practice equivalent to the research-appropriate journal paper.

Membership of the Commons involved two commitments:

- To attend a meeting once a month in London with the other members of the Commons.
- To create a course portfolio, which would be made available for public use.

This document is the resulting portfolio.

In section 2 we take a look at the context of the module, where it stands in relation to the teaching environment at Swansea University, in relation to the BSc course in Computer Science, in relation to my personal philosophy of teaching, the course structure etc.

In section 3 we take a look at the actual content of the course, what I teach, my personal aims and the aims of the department, the way the course is taught and assessed etc.

In section 4 we look at my findings about the course and how each of the aims described in section 3 is, in my opinion, achieved or not achieved by the way I teach the course.

Finally in section 5 we take a look at the things I have learned as a member of the Commons, and how I expect the course to change in the future as a result of the Commons.

For many of the sections there is an associated "artifact", a document or item which aims to demonstrate in some way that the stated objective is, or is not being achieved. These artifacts can be found as appendices at the end of the portfolio.

I have thoroughly enjoyed the experience of being a member of the commons. To have my teaching aims and abilities scrutinised by several people far more experienced than me has been a huge learning experience. I would like to thank Sally Fincher and all the other members of the Commons for their comradeship during what has been a wonderful year.

2 Context

2.1 Institute

The University of Wales, Swansea is a single campus of the University of Wales, whose sister campuses include Cardiff, Aberystwyth, Lampeter and some smaller campuses. Recently Swansea have followed the example set by the biggest campus, Cardiff, in applying for - and getting - degree awarding powers in its own right. This means that it will almost certainly in time be known simply as Swansea University. It is a medium sized university with about 12,000 - 13,000 students. The campus itself has a magnificent location, set in Singleton park with the beach of Swansea Bay just over the road (See Appendix One). Unfortunately its location has the drawback in that it is impossible to expand laterally, meaning that space is restricted and likely to remain so for the foreseeable future. In recent years a new Vice-Chancellor has allowed unpopular departments such as Chemistry and Philosophy to close, with the money saved being given to the better performing departments to allow them to expand. Computer Science has been a major beneficiary of this largely unpopular policy, almost doubling its numbers of teaching staff in just two years.

2.2 Department

To a great extent the lecturers and tutors are autonomous. When we design a new course we are very much left to our own devices as to what we intend to teach, how we teach it and how we assess it. Having said that, we have to produce formal documentation in a specific format. This document is then discussed by the Department's Teaching and Learning Committee, then by that of the University. The choice of language is, of course, not nearly as simple. When I took over introductory programming I introduced the "portfolio" system of assessment, the language was chosen by the entire department after, what I now understand to be the normal "language wars"! We are assessed in our teaching by senior members of staff. We submit Module Assessment forms to the students at the end of each module and the results of these, together with the exam results are discussed by the Department's Teaching and Learning Committee at the end of each teaching year. The students organise, and run, their own forum. This is extremely useful for picking up early indications that something may be wrong with the way we are teaching our modules. Students are also encouraged to discuss individual modules at compulsory weekly tutorial meetings. If we decide to run lab classes for the students in addition to our lectures we are allowed to employ postgraduate students to supervise these classes. We also employ post-graduates to run "program advisory" sessions each week at which students are encouraged to discuss with the post-graduates any problems they are having with any programming courseworks. Our typical entry requirement is 280 UCAS tariff points, equivalent to B,B,C at A-level.

The module which is the subject of this portfolio, CS-141, Principles and Practice of Programming (hereafter PPP), is a 20 credit module, making it 1/6 of the first year. Unlike other 20 credit modules it is however taught in just one of the teaching blocks, rather than being spread over both teaching blocks. The Department views programming as essential to all other modules in the three year course, thus all students must be able to program in Java by Christmas of their first year. Students who do not convince me that they can program, but who have not necessarily failed the course can "retrospectively" be given credit by passing successfully the coursework part of a Teaching Block 2 module, Data Structures. To pass PPP they must pass both a practical and a theoretical element. Students failing either part need only redeem that part to gain credit. (Course content and assessment is discussed in more detail in section 3).

2.3 Teaching Environment

Lectures can take place more or less anywhere across campus in a wide variety of lecture theatres. Recently a campaign to replace all blackboards with whiteboards was scuppered by some lecturers hatred of whiteboards. Most theatres are well appointed with projectors, computers, OHPs etc and any problems with the technology are generally solved quickly by Media Services. Lecture theatres come in many shapes and sizes, long and thin, wide and short and even amphitheatres. There are about 80 - 100 students taking the module each year, almost all of whom are Computer Science students, though there are always a few from other departments doing joint honours. For Computer Science students the module is both compulsory and core, meaning that a student cannot continue to the second year without passing the module. The Computer Science Department has three labs for the exclusive use by students taking Computer Science modules, of which only one is accessible to first year students. Each lab has about 30 computers. Thus we don't have anywhere near one computer per student. The labs tend to be overcrowded only when a deadline

for coursework is close. Most students realise this at some time during their first year and start earlier next time. The Department can easily afford more machines but do not at present have any more space for them, a local manifestation of the wider University problem of lack of space to expand. The machines in the lab accessible to the first years always have the latest version of Microsoft Windows installed. Students are introduced to other operating systems early in their second year. Many more "open access' machines are available for use by all students across campus, mostly in the library. The Department itself is very cohesive, existing over four floors of the same building. All the student labs are on the same floor, along with notice boards, students pigeon - holes and a general office. Some lab classes are held in our own labs, but we generally prefer to keep these free for general use so most lab classes are held in the library. Students have access to printers in the labs, but since these are free, unlike other printers across campus their use is strictly monitored for abuse.

2.4 Teaching Philosophy

I began my undergraduate degree as a mature student aged 36 in September 1996. I graduated with a 2/1 in July 1999. At that stage, reluctant to re-enter the "real world" which I had abandoned for my degree I applied for, and was accepted for a PhD in Volume Graphics. It soon became clear that, much as a student will clean their bathroom rather than do their coursework, so I would rather do anything than research. The department released a longstanding member of staff because his research profile was not up to standard. This left them with two modules - Compilers, and Object Technology - to teach and nobody to teach them. I had already taught a module in Artificial Intelligence at a branch college as a favour to the department and, despite teaching it badly had actually enjoyed the experience. I volunteered to take the two courses and thoroughly enjoyed them, despite learning compilers (again!) as I went along. Many of the lectures in that first year were prepared, and even learnt by me, the night before I delivered them. I don't believe that I have ever taught them as well since! I was originally given a four month fixed-term contract which was twice extended by one month each time. I interviewed for a new one year contract and was awarded a three year contract! By this time the Department, and the University, had started to change. A new Vice-Chancellor had arrived and the Department was given huge new sums of money to expand, and improve its research profile. The Head of Department decided that to give his lecturers more time to research he should use some of the money to employ three tutors (non researching). I was one of the beneficiaries. After two of the three years I once again interviewed (this time for my own job) and was awarded a permanent contract.

In essence my philosophy of teaching is simple to state:

1. To know my subject.
2. To know my audience.
3. To make my audience want to learn my subject.

Of course implementing this philosophy within the confines and rules of University is far more difficult. I have an underlying theme which can be generally stated as:

Happier students are more likely to work harder and less likely to drop out/fail.

For this reason I try to involve as many students as possible in various ways in the day-to-day running of the department. Apart from helping post-graduates to organise regular monthly social evenings I employ several students each year to assist with recruitment events. I also enter three teams each year into the BCS annual programming competition.

Teaching programming is difficult and very important. I regard programming for a Computer Science course in a similar vein to teaching reading and writing at primary school. You cannot teach primary school children topics like history and geography until they can read and write. Similarly our students can scarcely be expected to study algorithm complexity until they can program. I depend absolutely on feedback. If all students get a particular exam question wrong then it's my fault, not theirs. I also constantly ask my colleagues whether or not they think the students can program, and if there are particular problems my students have. I also listen to the students very carefully, both when they speak to me and when they post comments on their forum. I am always happy to alter my teaching methods in response to this feedback, and am currently doing so as a direct result of this commons.

3 Content

The following is the official description of CS-141, Principles and Practice of Programming, Computer Science Department, University of Wales, Swansea:

Synopsis: The aim of this module is to enable students to reach a level of skill in program design and practice such that they will be able to carry out assignments in Levels 2 and 3 of a Computer Science degree with a minimum of stress.

Syllabus: The nature of programs and the variety of programming languages. Programming control structures - conditionals, iteration. Parameter passing. Simple and structured data types; classes and objects; fields and methods; accessor and mutator methods; constructors. Program design techniques: top-down and bottom-up. Debugging.

Learning Outcomes: Students will be able to design, implement and document working programs in a modular maintainable style to the standard required in the rest of the degree scheme. They will have the ability to read and understand code written by others, and be able to adapt such code. They will be able to identify and correct, errors and bugs using debugging tools and standard debugging techniques.

Transferable Skills: Problem solving. Ability to learn and use computer systems and software packages effectively.

Course Text: D J Barnes, M Kolling, Objects First with BlueJ, 3rd Ed, Prentice-Hall, 2006.

The student is given a task book (See section 7) containing seventeen tasks which they have to solve in order to pass the course. Each task is a small, self-contained problem which requires the understanding of a particular element of the course. They also have to pass three assignments (See section 8). The assignments can be done in the students' own time and they have access to our Program Advisory sessions, as well as being able to see me if they are struggling.

I currently give three one - hour lectures each week for the first eleven weeks of the year. Each student also has the opportunity to attend a two hour lab - class each week during which they can attempt the tasks in the task book. Lab classes are not compulsory but the tasks cannot be assessed outside these sessions.

The assignments have deadlines in week 4, week 7 and week ten. Students have two weeks for each coursework and they are weighted 20% each. The task book is weighted 40%. This makes up the practical part of the assessment. Students are told that they will receive no credits for coursework until their task-book is complete.

The task-book is a new idea to the Department, pioneered by me especially for this module. By completing the tasks weekly the students are able to see that they are making progress. The assignments are more difficult and require more thought and time. I teach "according to the task-book", meaning that I assume that an average student will complete two - three tasks each week and should therefore be able to tackle the tasks in order. I take the view that programming is very much an engineering discipline, consisting of several essential detailed concepts (iteration, loops etc.) which are then "bolted" together to create problem solving programs. The task book then is intended to teach the student the basic concepts while the assignments provide the problem solving tasks, requiring the bolting together of ideas they already understand.

4 Evaluation

It has already been stated that any module is evaluated before being taught, both by the Department and by the University. All new modules are "monitored" by other members of staff and modules being taught by a new lecturer are also monitored. This monitoring usually takes the form of observing one or two lectures and by informal gathering of feedback from students. There is also a formal method of treating feedback from students. At the conclusion of each module students are encouraged to fill in a questionairre (See section 9). The results of these questionairre are collated by the lecturer and discussed in some detail at a special meeting of the Department's Teaching and Learning Committee. Students are also involved in a student staff meeting during their final year at which modules are discussed in great detail in a relaxed, yet semi-formal environment. The results of these feedback mechanisms give me confidence that the module is in general going well and achieving its aims.

With regard to the official description of the module (given in section 3 above), students are generally able to program well enough to be able to perform any tasks set by lecturers for later modules. As far as the syllabus is concerned it is all covered with the possible exception of "Simple and structured data-types". I do not go into data structures in any great detail as this is covered in a separate Level One module, but discussions with the lecturer of that module suggests that the students are able to understand the basics required for that module. The learning outcomes and transferable skills are likewise covered, but there remains a serious lack of commenting of code! It is clear that I do not impress upon the students the importance of commenting. I will probably increase the weighting of comments in future assignments and maybe add some tasks to the task book in an attempt to correct this.

The task book is, I am convinced a good idea. At least two other lecturers have adopted the idea or a variation of it for their modules. It is very easy for a lecturer to see from the task book what skills each student should have, especially since it is impossible for a student to progress without having completed the task book. Several lecturers have asked me to add certain tasks which they feel important for their modules. The task-book idea makes it very easy for any new lecturer to see at a glance the sort of things we teach and regard as important. The task book idea also works for the students. From discussions with the students it is clear that they enjoy seeing their task book fill up week by week as it gives them confidence that they are learning and progressing. Able students can complete the book at their own pace. Every year we have at least one student who completes the task book during the first week of term! This means that they no longer need to attend the lab sessions, freeing up space and time for us to be able to deal with the less able students. It is also very easy for the course lecturer to see at a glance if a student is falling behind and take the student, or his/her tutor aside for a brief chat to see if anything is wrong. Several times we have identified a student with a serious problem in this manner, problems which otherwise might have resulted in the student dropping out. The task book also forms an informal guide to the lectures. It would be easy for a lecturer standing in for sickness to see at a glance whereabouts the lecturer has got in the course and take a lecture on a specific theme, such as loops, constructors etc.

The assignments are altogether a different thing! Taking into account the five things I believe to be most important in ITP assessment (See section 10) there are clearly some problems.

4.1 Assigment One

Many students achieve a good mark for this work without having to stretch themselves. It is also not very exciting. It does test students ability to construct and use different objects, parameter passing etc. A poor mark does indeed show which parts of the course are not understood and it does demand understanding of all the parts of the course taught to date. It is difficult to make such a trivial task exciting and motivating, and, since it is only one of three assignments I'm not too worried about it as yet. I do have some ideas about the assignments in general for which see the next chapter.

4.2 Assigment Two

This is much better in terms of achieving the top five! Unfortunately the students seem to find it too difficult. The main problem is in reading input from files, traditionally one of the most difficult and tedious things to do in Java. The new version of Java has a vastly improved series of classes for input/output and I intend to leave the assignment more or less as it is for a year or two to see if this fixes the problem.

4.3 Assigment Three

This is in many ways the best of the three assignments. It fulfils most of the top five and the students seem to do fairly well. It also (deliberately) contains some ambiguities which encourages the students to question themselves and the tutor, something we regard as important in Software Engineering. It doesn't however contain any inheritance or polymorphism, which violates the fifth of the top five importantcies. For an earlier second year module I developed an alternative agency which addressed this, but the problem became completely artificial.

4.4 The Exam

The exam is intended to examine the students' theoretical knowledge and as such I am quite happy with it. Aside from the problem of asking the same question a different way each year it tests that the students have a deep understanding of the concepts they will require later in the course. I do have a problem in writing good exams, in as much as I tend to have an "inverse" bell curve. That is to say that students who achieve a pass mark tend to get sixty percent or more. This is a problem I have with all exams and I am looking at other lecturers' exams to try to rectify the problem.

4.5 The Module

The assessment of the module has already been explained and this also causes some problems. Students are awarded 40% for the task-book and 20% for each assignment. This comprises the 50% practical content of the module and the exam (See section 11) makes up the other 50%. Unfortunately some of the weaker students feel that, having earned 40% for completing the task-book they can relax and make minimal effort on the assignments. Although the task-book and other literature makes it clear that they can fail the module this way most years there are one or two appeals. To avoid these problems we will in future, starting this year, award no credit for the task-book (but still make its completion compulsory) and weight the assignments 30%, 30% and 40%.

5 Reflections

The Commons' main effect on me was to force me to evaluate my teaching methods for the first time. I have always until now been satisfied that, as long as the students achieved what was expected then I was achieving my task as a tutor. A year spent discussing teaching with so many different people has persuaded me that I could in fact do much better than I am and a number of ideas have been growing throughout the year.

5.1 Teaching

I have always believed, or at least stated that I believe, that students can only learn to program by doing it, not by listening to me talking about it. Why then do I spend three hours each week in a lecture theatre? For the forthcoming year I will be reducing my teaching hours to two lectures each week and will also make the lab classes compulsory. At the start of each lab class I will go through a practical example with the students. Once they have completed the practical task associated with this class they will be expected to complete at least two tasks from the task book. This should have the effect of ensuring that more of the students complete their task books as well as giving them even more experience of actually programming. In future I intend to reduce the lectures to once each week and possibly have some or all of the lectures video taped and made available online.

5.2 Assessment

A major problem has emerged this year with masters students who, broadly speaking are taught the same module. Since they achieve 40% for completing their task books they tend to put much less effort into the assignments despite the warnings given in the module literature that all assignments must be attempted to a reasonable standard. It is clear that, although the task book idea is a good one I didn't think carefully enough about the assessment. Starting this year students will not achieve any credit for the task book but its completion is compulsory. This also gives me the opportunity to weight the third assignment more heavily than the first two. It is still the case that the tasks and the assignments are not very exciting and stimulating. In future I hope to create a large program which the students will gradually "accumulate" over the period of the module. This will require a great deal of thought and very careful design but I am already looking at examples from the Commons.

5.3 Observing

I have always been interested in watching other people teach. I try to attend at least one of somebody else's module each year, as much to watch their teaching as to learn the module. Attending seminars and colloquia is also very valuable. Bering observed was a whole new experience for me and one which I found extremely useful. My observer was able to spot several things which I was missing and I found her comments very helpful. I will try to arrange with some of my colleagues to observe each other at regular intervals. This may be difficult as the emphasis at the Department is very much on research and most of my colleagues are not particularly interested in improving their teaching abilities.

5.4 Space

Observing somebody else's lecture was equally useful and I spotted something I had never before considered. He was giving a lecture in a very wide shallow theatre which meant that some students at each end of the theatre were "excluded" from much of the lecture. These students tended to be self selecting, meaning that the weaker students who ought to be the focus of more attention were in fact more likely to miss out through their own choice. In discussing this phenomenon with the other Commons members it became clear that many of us had the same idea and that it was part of a larger concept. Lecturing style differs from one lecturer to another and different styles are encouraged or discouraged by the size and shape of the theatre. A lecturer who tends to be interactive and encourages students to come to the board would need lots of aisles and would be very unlikely to deliver a good lecture in the theatre described above. Another lecturer found herself in a small theatre directly above the students' dining room at 9:00 a.m! She changed her style for those lectures into a seminar type lecture, working in small groups. It is clear that we are affected in many ways by the theatre in which we are obliged to teach and we should attempt to derive several different styles to match our current environment (unless of course we are free to choose where we will lecture). In another

module which I teach I have two lectures which can be "dropped in" at any point during the module. These come in useful if I feel the students want a break from the routine, and also if for any reason I want to show off. It would be useful to create several such lectures for each module and I will be looking into this.

6 Appendix One

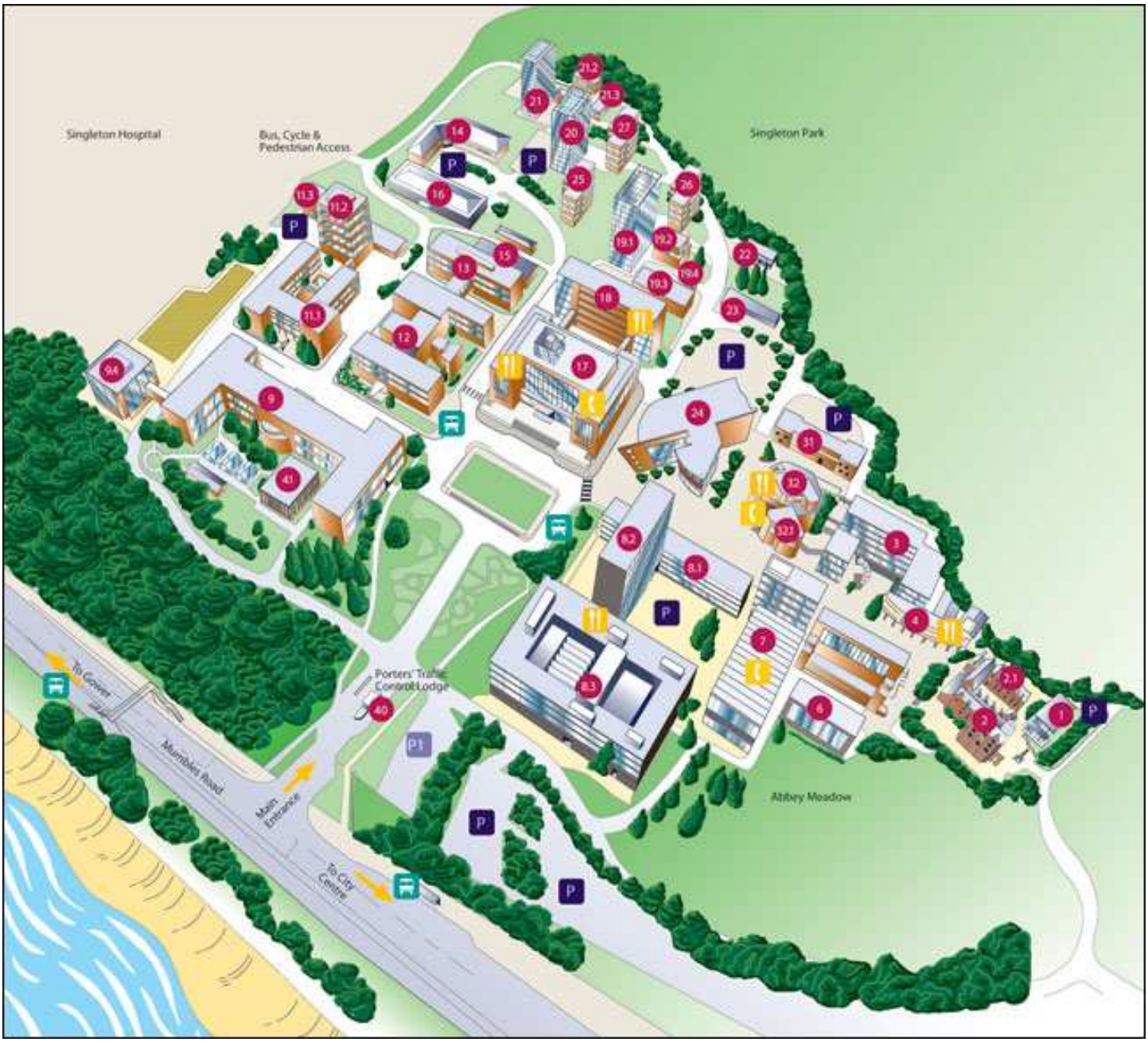

Figure 1: Map of Swansea Campus

7 Appendix Two

Level One Programming.

CS_141 Principles and Practice of Programming.

C. J. Whyley

Student Name	Student Number

Introduction

This booklet is essential for each student studying this course. It gives a brief description of the course, explains the aims and expected learning outcomes and, most importantly, contains a *checklist* which **must** be retained by each student. Please write your name and student number on each page where it is required and take it to all your laboratory classes.

Course Structure and Assessment

The aim of this course is to teach you to program. This is an essential skill which you will require throughout your 3 or 4 years here and therefore we teach it quite intensively during teaching block 1 (from now until Christmas). The course is split into two for assessment purposes, a written exam will take place in January to test your knowledge of theory. The practical element of the course will be assessed completely by coursework. This assessment takes place throughout teaching block one during the laboratory sessions. There are three lectures each week, each of about 45 minutes duration. You will also attend a two hour laboratory session once a week. In addition to this booklet you have been given a form on which you can state your preferred choice of lab class. You must return this form by Friday October 1st if you wish to state a preference, otherwise a lab session will be allocated for you.

Laboratory Sessions

The laboratory sessions exist for two purposes.

- There will be two people at each lab session to help you with any problems you may be having with the programming. Remember, they are there to help you, not to do your work for you.
- When you feel you have completed a task in the checklist in section six you should ask one of the assistants to examine it. If he/she agrees that the task is completed he/she will sign the appropriate space for you. When all the tasks have been completed you will have completed this part of the assessment. You can **only** have these forms signed during lab sessions (It may be possible to have some tasks assessed electronically. Details will be given on the course web-site and during lectures).

Assessment

The practical part of the course is assessed continously. To complete the course you must complete **all** the tasks in the checklist in section six and also make a satisfactory effort in three pieces of coursework which will be given out during TB1. Completing the checklist will score you 40%. The remaining 60% will be given for the three assignments. You will not receive any marks at all until the checklist is completed. You will also not receive any marks if you do not attempt the courseworks to a satisfactory standard. If you fail to convince the examiners that you have demonstrated capability in all tasks you may be obliged to complete the unfinished tasks under exam conditions during the supplementary exam period in August. When teaching the course we will assume that you have never programmed a computer before. If in fact you have some experience then you can complete the course more quickly. Each student can work at his/her own pace. You are quite free to work upon the tasks in your own time and merely attend the lab sessions to have them checked. Help can be given during Program Advisory sessions in our own PC lab (see notice boards for details), however, remember that you can only have your checklist signed during the normal weekly sessions (or electronically).

In addition to the practical part of the course there will be an exam to test your knowledge of the theoretical aspects of the course. This exam will take place in January. In order to complete the course you must pass both the practical and theoretical parts.

The tasks

The tasks are intended to be of increasing difficulty in each group and so should be attempted in the order given in each group. Before describing the tasks in each group there is a summary of the main learning outcomes for that group of tasks. In addition to completing the tasks you should make sure that you understand the concepts described and are able to answer exam questions about those concepts.

The Checklist

Chapter One. Objects and Classes.

After completing these tasks you should have a clear understanding of :-

- The difference between an **object** and a **class**.
- Mutator methods and accessor methods.
- Invoking methods, with and without parameters.
- Returning values from methods.

	Task	Assessed
1	Download the "Task_1" package from the web-site. Alter it so that there are two suns, one red and one yellow, one on each side of the house. Write a method called **sunset** which, when invoked, makes the yellow sun slowly descend.	
2	Open the "Task_2" package. Write a new method called **getInstructor** which, when invoked, returns the name of the instructor for that class. Create a new **labClass** object and fill it with several new **student** objects before asking for this task to be assessed.	

Chapter Two. Constructors, Methods, Variables and Flow Control.

After completing these tasks you should know :-

- What a **constructor** is.
- What is meant by a **variable**.
- What the **scope** and **lifetime** of a variable are.
- How to print to the screen
- The concept of **fields** and how they affect objects.
- What a **boolean** expression is and how to make code execute in different ways depending upon the boolean value of a test.

Both tasks in this chapter require the "Task_3" package from the course web-site.

	Task	Assessed
3	Add two new fields to the class, a **String** field called refNumber, and an **int** field called **pages**. refNumber should be set to a value of "" (the empty string) by the constructor (which will need to be altered accordingly). The pages, representing the number of pages this book contains, should be passed into the constructor as an extra parameter. Add another method to the class called **setRef** which alters the refNumber field	
4	Add another method which prints the details of the book to the screen **in a neat, tidy format**, together with some textual information for the reader explaining each piece of information. Your method should only print the reference number if it has been set. Otherwise the method should print "Ref No. not known".	

Chapter Three. Comments, debugging and testing.

After completing these tasks you should know :-

- How objects can create other objects.
- The difference between **internal** and **external** method calls.
- What method **overloading** is.
- How to use a debugger to find errors in your programs.
- The importance of commenting your code.
- The importance of testing your programs thoroughly.

	Task	Assessed
5	Download the "Task_5" package from the web-site. Carefully read the **readme** file which explains in detail what the program should do. Use the debugger to find the errors in the program and fix them.	

Chapter Four. Collections, iteration and loops.

After completing these tasks you should understand the following concepts :-

- The **import** statement.
- Collections, **arrays** and **arrayLists**.
- Iteration, the **for** and **while** loops.
- Casting

	Task	Assessed
6	Download the "Task_6" package. Add a method which, when invoked, will iterate through the collection printing each entry.	
7	Alter the **showNote** and **removeNote** methods so that they print an error message to the screen if invoked with an invalid parameter.	
8	Download the "task_8" package. Rewrite the printHourlyCounts method so that it uses a **while** loop instead of a **for** loop.	

Chapter Five. Libraries.

After completing these tasks you should understand the importance of :-

- Using library classes.
- Reading and writing correct, accurate documentation.
- Access modifiers **private** and **public**
- **Static** and **Final**

	Task	Assessed
9	Download the "task_9" package. Add a method which converts a string typed in by the user to lowercase. (You should use a library method from the **String** class to do this).	
10	Download the "task_10" package. Add the appropriate access modifier for each method and each field. Add correctly formatted **javadoc** comments to document the code.	
11	Create a new class called **Student**. The constructor should take a String as a parameter representing the student's name. The class should have a **String** field called **teacher**.Instantiate 4 objects of type **Student**, each with a different name. Each student's teacher should be **"Mr. Whyley"**. This name should not be alterable by the program.	

Chapter Six. Testing.

After completing these tasks you should understand the importance of testing thoroughly and be able to create and automate a series of tests.

	Task	Assessed
12	Downloas "Task12". Read the documentation and "Readme" file carefully to understand what the methods do. Write a test class using the unit testing tools which test the following features: ParUpTo(3), ParUpTo(8) ParUpTo(18) and the par for hole 10. To ensure that you fully understand the unit testing paradigm you should experiment with placing different values in the array and watching the tests fail. Do this after you have had this task signed off.	

Chapter Seven. The "Main" keyword.

After completing this section you will understand how to write, compile and run programs from different **IDE's** and from the command line. After this section your assessors will only sign your book if the programs compile and run from the command line.

	Task	Assessed
13	Download the program you wrote for task 11 and alter it to allow it to run from the command line. The program should now instantiate a new student after asking the user to type the student's name. The program should then print the student's name and teacher's name to the terminal. Download the program you altered for task 1 and alter it again so that the sunset method is called automatically by running the program from the command line.	

Chapter Eight. Inheritance and Polymorphism.

The tasks in this section are designed to ensure that you understand the concepts of :-

- Inheritance.
- Polymorphism.
- Code reuse.
- Sperclasses and subclasses.
- Superclass constructor calls.
- Polymorphic containers and collections.
- method overriding.

	Task	Assessed
14	Write a class called "Item". This should have fields for **Name** (a String), and **cost** (an int).These fields should be passed to the Item constructor. Write a **print** method which prints these details to the screen in a tidy format. Write a class called **Library** which contains an ArrayList of Items. Write a **printall** method for the class which iterates through the arraylist calling the Item print method on each object.	
15	Derive two **subclasses** of **Item** called **Book** and **Video**. The Book class will contain an extra field called **NumberOfPages** and the Video class will contain a field called **Length**, both of type int. Each class should override the **print** method of its superclass so that when its print method is called it prints out all its details to the screen.	

Chapter Nine. Algorithms

In order to complete these tasks you will have to find some algorithms from books and write code to implement them.

	Task	Assessed
16	Write a program which reads 100 integers from a file (numbers.txt). Store these integers in an array in the order in which they are read. Print them to the screen. Sort the integers in place using **bubble sort**. Print the sorted array to the screen. Now implement the **sieve of Eratosthenes** to place all prime numbers in an ArrayList. Print that list to the screen.	
17	Write a program which reads 50 strings from a file and places them in a sorted array using **Insertion sort** according to their lengths.	

8 Appendix Three

Assignment One

Outline Up until now in the lectures and your task books you have been mainly concerned with learning the individual parts of the Object Oriented language Java. It is now time to "bolt" these components together to create a complete working programme.

You are required to create a library capable of holding a collection of books.

1. Create a new project called **Library** in Bluej.
2. Create a new class called **Book**. Your book class should contain the following fields :

- Author – String.
- Title – String.
- Cost – String.
- NumberOfPages – String.
- enjoyability – Integer between 0 and 10.
- Genre – String.

3. Write a constructor for your book class which accepts parameters for these fields and sets the fields accordingly.
4. Write an accessor method for each field.
5. Write a **print** method which will print all the details of the fields to a terminal in a neat tidy format.
6. Create a new class called **Library**. The library class should contain a suitable collection object as a field. The constructor should construct this object.
7. Write a method which calls the Book class constructor to construct a new book and add it to the end of the collection.
8. Write a method for the Library class which prints all details of all books to the screen. (Using the Book's print() method).
9. Write methods which print details of only those books which:

- Have more than x pages (where x is given as a parameter).
- Were written by a particular author.
- Had an enjoyability score of x or above.
- Are of a specific genre.
- Cost less than x pounds.

10. Write correct, meaningful documentation for both classes.
11. Write a method called *alphaAdd* which, after constructing a new book, adds it to the collection in alphabetical order by the title.

Marking

1. A program which runs and has some functionality – Up to 50% (depending upon how complete the functionality is).
2. Good use of meaningful names for methods, fields, parameters and variables – 10%
3. Good class documentation – 20%.
4. Task 11 – 20%

Submission

Deadline 4th November 2005. You should archive the files you have created and submit them via the course's Blackboard site. You will be told how to do this in the lectures. Coursework received after the deadline will incurr a penalty of 2 marks / day (including weekends). Coursework submitted more than one week late will not be marked except by prior arrangement.

Help

You may not work on the assignment during the lab classes, these are for your task books. You are allowed to ask for help during the usual Program Advisary sessions, and you can come and see me also. If you ask me for help I will expect to see that you have made a reasonable attempt for the amount of time, i.e. do not ask me for help one week before the deadline unless you have nearly finished.

Assignment Two

Date set : 8/11/2004.

Deadline : 24/11/2004

Introduction

The aim of this assignment is to give you practice in inheritance and polymorphism. You will need to design your classes very carefully to achieve a high mark.

Overview The greatest manager in football history (Harry Redknapp of Portsmouth), has asked you to write a program to assist him in selecting his team each week. He keeps records of each player's performance to date and also keeps a record of the most recent match. If a player's performance in the most recent match was better than that player's overall performance to date then the player's performance is improving, otherwise it is deteriorating. If a player's performance is deteriorating then he will be dropped for the next match and be replaced by the best player for that position who didn't play in the last match. Your program will:

- Read in the details to date from the file "Summary.txt".
- Read in the report for the last match from the file "LastMatch.txt".
- Write to the screen for each player in the last match,his name, his performance in that match and whether he is improving or deteriorating.
- Write to the screen the team for the next match with each player's performance rating (including the last match if he played and his position.
- Write to a file called "Summary_1.txt" the new details of each player.

Details

Each player will play in one position, **striker, midfielder, defender** or **goalkeeper**. Each position has a different way of calculating performance rating.

- Striker: (Number of goals scored + 0.5 × number of non–scoring shots on target)/total shots.
- Midfielder: Number of accurate passes / total passes made.
- Defender: Number of successful tackles / (number of fouls + (0.2 × number of goals conceded by the team)).
- Goalkeeper: Number of saves / (number of saves + (0.5 × number of goals conceded)).

Implementation

The first class to write will be called "Player". This will be inherited by four sub–classes : "Striker", "Midfielder", "Defender", and "Goalkeeper". Think carefully about what details all players will have. (Read the file format below). The player class will certainly have a field in which to store the number of goals conceded by the team. Think carefully about the declaration of this field. There will be several methods in this class including "calculateForm". Think carefully about what sort of method this should be.

Then derive the four subclasses, each with its own method of calculating the rating.

Write a class called "Team". This class will read in the details from the summary file, construct players and store them in a suitable collection. Then it will read in the details of the last match. Each time it reads in a player you will need to:

- Calculate their rating for the match.
- Find that player in the collection.
- Decide whether he is improving or deteriorating and hence whether he will play the next match (storing that information somehow).
- Write the necessary information to the screen.
- Recalculate his overall rating.

When the match file has been processed this class will write the new information to the "summary_1.txt" file. (In a real scenario it will overwrite the "summary.txt" file but write a new one to assist with my marking). The next job is to select the team for the next match. Iterate through the collection and find the players who played the last match. For each player: If they are to play then write them to the screen as stated above. If they are to be dropped then iterate through the collection finding the best player for that position to replace him.

Finally write a class called "TeamSelector" which will create a "Team" object and call one of its methods.

File format

There will be a different number of lines in the summary file for each player:

The first line for each player will be the player's name.

The next line will be "s", "m", "d", or "g", depending upon his position.

Depending upon the player's position there will be two or three more lines, (see below).

The next line will be that player's current overall rating.

The last line in the file will be the number of goals the team has conceded in the season so far.

The last match report file will contain a different number of lines depending upon the player's position.

The first line will be the player's name.

If the player is a striker there will be three more lines, one for the number of goals he scored, one for the number of shots on target and one for the number of shots off target.

If he is a midfielder there will be two more lines, one for the number of passes which went to a plaer on his side (successfull) and one for the number of passes which went astray.

If he is a defender there will be two more lines, one for the number of tackles he made and one for the number of fouls he committed.

If he is a goalkeeper there will be only one more line, the number of saves he made.

The last line in the match report file will be the number of goals conceded by the team in the match.

Submission

The deadline for this assignment is 24/11/2004. The program must compile and run from the command line, though you are encouraged to use BlueJ for testing your classes. Attach your seven files to an email and put "coursework" in the subject box. Don't forget to put your name and student number in your email.

Marking

A program which contains the bulk of the code and compiles on the command line – up to 50%

A program which runs without crashing and produces output – 10%

Correct, well formatted output to screen and file – 10%

Meaningful variable names and well written code – 10%

Well designed classes, good use of inheritance and good programming style – 10%

Goog JavaDoc comments – 10%

Help

Stsrt EARLY. Come and ask me for help. Go to program advisory for help. Keep an eye on your emails as I will answer general problems that way, and of course in my lectures.

CS_141 Assignment 3 2004/2005
Set 17/11/2004
Deadline 10/12/04

Outline

For the first two assignments you were given help in deciding how to tackle the problem. For this assignment you are on your own. You must read the problem **carefully** and make your own design decisions, although you will receive help in a lecture.

The Problem

You are the organiser of a dating agency. Each customer has a forename, a surname, a gender, a list of pass-times which they enjoy and a list of pass-times which they hate. You must find a partner of the opposite gender for each customer. You will be paid by each of your customers depending upon how happy they are with their assigned partner. Their happiness depends upon how many pass-times they have in common and whether or not either of them enjoys a pass-time which the other hates. For each of the pass-times which they both enjoy you will receive £10 but if their partner enjoys something which they hate they will deduct £5. The more profitable your company is the more marks you will receive.

Input

There is only one file for this program, a text file called **Customers.txt**. The first line for each customer contains their gender (m or f). The second line contains their surname, the third line contains their forename, both as **strings**. The fourth and fifth lines are a series of characters (without spaces). There will be one or more of the following :-

```
f = football
o = opera
t = theatre
c = cinema
l = literature
a = alcohol
```

The fourth line is their list of likes, the fifth a list of their dislikes. The file ends with a single carriage return after the last line of the last customer. Sample entries in the file may look like the following :-

```
...
m
Whyley
Chris
fota
cl
f
Whyley
Helen
tca
fo
...
```

Marking

- A program which contains the bulk of the necessary code which compiles and runs without crashing - pass. (The program MUST compile and run from the command line).
- Printing details of each customer's partner to the screen in a neat tidy format - 2/2
- Computing and printing how much each customer pays the company - 2/1.
- Attempting to maximise the total income - 1st.

Extra marks as always can be obtained for well documented, easily readable code, meaningful variable names, good use of classes, sensible declarations etc.

Submission

As usual you should submit your program (just the **java** files) as an attachment to an email. You **must** put **coursework** in the subject box of your email. Failure to do this will result in a deduction of marks. The deadline is noon on 10th December. Late submissions will be penalised, I will not mark any work submitted more than one week late except by prior arrangement. Help will be available in the usual way and under the usual conditions.

9 Appendix Four

10 Appendix Five

The top five things I believe to be important in itp assessment

1. A good solution earning a high mark must demonstrate clear understanding of the theory and implementation of the task.

2. A good project/assignment should be exciting and motivating.

3. A poor answer earning a low mark should show which parts of the course the student had not understood.

4. A good mark should require a suitable amount of effort. It should not be possible to gain a good mark without working hard.

5. A particular assignment should demand understanding of all elements taught in the module to date.

11 Appendix Six

CS_141 Principles and Practice of Programming

(Attempt 2 questions out of 3)

Question 1.

(a) Explain the Object–Oriented concept of *encapsulation*. Your answer should include references to the four privacy modifiers. **[10 marks]**

(b) Write a *while* loop and a *for* loop, both of which will print the integers from 1 to 100 to the screen. **[6 marks]**

(c) Explain what is meant by a *static* field, giving an example of its use. **[6 marks]**

(d) Where is a field visible if it is declared as *protected*? **[3 marks]**

Question 2.

(a) Explain the Object–Oriented concept of *polymorphism*. Your answer should include references to *static* and *dynamic* binding, *collection* objects, and *casting*. **[10 marks]**

(b) Describe the format of a *switch* statement. When would you use a switch statement instead of an *if-then-else* statement? **[7 marks]**

(c) What is the difference between *accessor* method and *mutator* methods. **[5 marks]**

(d) What does it mean for a field to be declared as *final*? **[3 marks]**

Question 3.

(a) Explain the Object–Oriented concept of *inheritance*. Give a simple example of an inheritance structure. Your answer should clearly show the role of *constructors* in inheritance.

[10 marks]

(b) What is the difference between method *overloading* and method *overriding*? **[5 marks]**

(c) Where is a field visible if it is declared as:

1. Private
2. Public

[4 marks]

(d) What is a static method? Give an example of a static method and explain how it is invoked.

[6 marks]