Imperial College of Science, Technology and Medicine	University of London
Computer Science (CS) / Software Engineering (SE)	BEng and MEng Examinations Part I
Department of Computing	Integrated Laboratory Course

Laboratory work is a continuously assessed part of the examinations and is a required part of the degree assessment.

Laboratory work must be handed in for marking by the due date.

Late submissions may not be marked.

Exercise: 11	Working: Individual
Title: Name Sorting	
Issue date: 26th January 2004	Due date: 2nd February 2004
System: Linux	Language: Java

Aims

- To introduce object-oriented programming using Java syntax.
- To exercise writing programs that are made up of a number of classes.
- To familiarise you with the object-oriented concepts of superclass, subclass, constructor, inheritance, polymorphism and method overriding.
- To explore some of Java's modifiers, in particular, public, protected, final and static.
- To reinforce your understanding of parameter passing in Java

The Problem

- This is a simple exercise requiring you to read in and sort into alphabetical order a group (database) of Imperial College student records. However, the database will contain two types of student - a 'generic' student defined by their College-wide login, forename and surname, and a special type of student (DoC student) who has an additional attribute - their DoC login. You need to define four classes: a **Student** class which needs to be extended to accommodate DoC students (**DoCStudent** class), a **Group** class representing a group of students and a main class **GroupSort**. The first three classes will form a package called **student**.
- Skeleton files are provided for all four classes. Copy these files with the command: **exercise 11**. Create a sub-directory, **student** and move the skeleton files **Student.java DoCStudent.java Group.java** (but not the main program Java file **GroupSort.java**) into this sub-directory.

- Your program will need to achieve the functionality described in the spec and it *must* contain the classes and packages named in the spec. You are free, however, to define and name your own methods inside the specified packages and classes. You **cannot add additional classes or packages to your program**.
- A **student record** consists of a **login-name**, a **forename** and a **surname**.[1] Each of these *fields* is simply a string, and can be assumed to contain no spaces or tabs ("whitespace").

  A **group** of student records is any number of student records from 0 (an empty group) to an upper limit of 100. Student records can be sorted alphabetically by surname with students sharing the same surname being sorted by *forename*, and students sharing the same forename and surname being sorted by *login-name*.
- Your program should **not** do any other I/O, such as prompts, apart from a message requesting a group of students to be input and an output message as they appear in the skeleton main program part.
- If your program contains assertions you will need to compile your program with the following flag **javac -source 1.4 FILENAME(S).java** and run it with the following flag **java -ea FILENAME**.

Submit by Monday 2nd February 2004

What To Do

A Student

- Define a class **Student** which contains *protected* variables for the above data along with the void method **getStudentData()** for reading a student record and the **toString()** method for formatting (rendering as a string suitable for printing) a student record. This *class* is defined as part of the *package* **student** so the corresponding class file should be stored in a sub-directory called **student**. The same applies to the **DoCStudent** and **Group** classes, as noted earlier. **Note:** we have used the methods provided in *kenya.io.InputReader* for reading input from the keyboard. You can change this if you want to use other publicly available classes but we recommend that you *import* and use this class for the first Java exercises.
- Now define the **DoCStudent** sub-class of **Student** and include an additional private variable for the DoC login (a String). Override the **getStudentData ()** and **toString ()** methods respectively to read and format the extra DoC login. You may if you wish use one or more of the superclass methods to help define these methods.
- Define the **Group** class. This should contain a private variable of type **Student[]** for storing a group of students and a private integer variable for recording the number of students stored.

[1] In real life there is a great variety of patterns of names. For example, people can have middle names, or "double-barrelled" surnames (like de Souza or von Neumann); or their surname can come before their other names, as in Chinese names. However, to keep things simple, for the purposes of this exercise everyone will be assumed to have exactly one forename and one surname, in that order.

The **Group** constructor should take no arguments and create an array capable of containing up to 100 students. Define a method **addStudent (Student s)** that adds a student to the array of students. Define methods **putGroupList ()** that will print the items in the student array (using **toString ()**) and **sortGroupList ()** for sorting the student array lexicographically. You will need to define an auxiliary method say, **lessThan(Student s)** which gives true iff the student is lexicographically smaller than the given student **s** under the above ordering.

The Main Program

- The main program should read and print a group (list) of student records, then sort the records and print them. The records are read from a single source file which comprises a mixture of DoC student and general (default) student records. You may assume the format of the input file is one student per line, with each lines beginning with a (String) tag which distinguishes DoC students from other students. The line format is:

  DoC <*login-name*> <*forename*> <*surname*> <*DoC-login*>

  for DoC students and

  Def <*login-name*> <*forename*> <*surname*>

  for the other (default) students. The fields are assumed to be separated by one or more whitespace characters (the Kenya I/O system will resolve this for you). You can assume that there will be no more than 100 students and that all login names will be unique.

- Your main program should print both the unsorted group and the sorted group by creating an instance of **Group** and calling the **Group** constructor and methods accordingly.

- Note: generic student records should be printed one per line, tab separated, in the format:

  <*login-name*> <*tab*> <*forename*> <*tab*> <*surname*>.

  DoC students should be printed similarly but with the DoC login added (on the same line) in parentheses, thus:

  (DoC login = <*DoC-login-name*>).

I/O in Java

We recommend that you use the I/O classes that you used in Kenya. This is because the Java I/O classes are not well designed - a topic you can discuss with your PPT. The input is provided by the class **kenya.io.InputReader**, you will need to import this into any classes that use it. The class contains methods that allow you to read in values of different types. The methods return values and can be used in assignment statements, e.g.

```
String word;
word = InputReader.readString();
```

The names of the other methods for reading in values of other types follow the same pattern as above.

For output you can use **print ()** and **println ()** as with Kenya but you need to preface this with **System.out**, as in:

```
System.out.println( word );
```

The String Class in Java

The **String** type is expressed as a class in Java and contains methods for performing operations on **Strings**. For a full list of the provided methods you should refer to the Java class library, but for the purposes of this exercise you will only need to use the method `int compareTo( String s )` for comparing strings lexographically. If the current is the same as the argument 0 is returned, if the current is less than the argument less than 0 is returned or if the current is greater than the argument greater than 0 is returned. Again, this is highly questionable design!

Sorting Algorithms

- You may use any sorting algorithm you wish.

Testing

- It is **extremely important** to thoroughly test your program and you should **test your code thoroughly** before submitting.
- Test your program on a variety of input groups. Remember *you* are responsible for the behaviour of your program, and you should try your hardest to detect and correct any errors in your program.
- You can test your program using linux's *input re-direction* to read from a prepared test file. e.g. **java -ea GroupSort < NAME_OF_TEST_FILE**.
- Note that an *empty* group is perfectly valid and your program should handle it.

Unassessed

- You could further extend the **Student** class or **DoCStudent** class to include different types of student.
- You could add code to check for errors such as duplicate logins or too many students for the array.

Submission

- Submit your program by copying the file **GroupSort.java** into the **student** subdirectory and then *cd* into the **student** subdirectory and then type: **submit 11**. Check your mail to ensure that you have submitted your program files correctly.

Assessment

Reading in the class	2
Sorting the class	2
Printing out the class	1
Design, style, readability	5
Total	10