SCHOOL OF COMPUTER SCIENCE

PRIFYSGOL
BANGOR
UNIVERSITY

Object Oriented Programming in Java

Laboratory 8
Using Linked Lists

Dave Perkins

Introduction

This laboratory follows on from the lectures dealing with the part of the Java library known as the Java Collections Framework (JCF).

In this session you will gain experience dealing with the class **`LinkedList`** and using the accompanying API documentation to obtain an understanding of the functionality available to the programmer.

Make sure that you study the relevant Java API carefully as it will provide you with most of what you will need. Note that you are only allowed to use the fundamental **`LinkedList`** methods described in *Big Java*, Chapter 14. If you use other **`LinkedList`** methods you will be penalised.

Exercise 1: Creating the Class Student

Write a class to represent students. For the purposes of this exercise assume that a student is defined as follows:

```
public class Student
{
      private String foreName;
      private String surName;
      private String studentID;
      private String degreeScheme;
      ...
}
```

The class **`Student`** should contain a constructor, appropriate getters and setters, and the usual string methods. Compile the java source to obtain a **.class** file and then write a tester class which creates three instances of **`Student`**. For this exercise, supply student details as hard-coded parameters. As always, ensure that your test program provides 100% method coverage.

Exercise 2: Using the Class LinkedList

The data type **`ArrayList`** in Java offers a dynamic data structure with fast *read access*. Whilst this is ideal for many problems, the structure is not suitable for every eventuality. In some contexts, *write access* to a data structure (i.e. inserting anddeleting elements) is time critical and must therefore be implemented as efficiently as possible. Unfortunately, when re-organising, e.g. removing or adding elements in a large **`ArrayList`**, a lot of computational time can be taken up by having to “shuffle” elements about.

This is where linked lists come in. One of their main advantages is the ability to add and remove elements with little disruption to the rest of the data. For this exercise you will use the **LinkedList** class provided in the **java.util** package to create a very basic student registration system. Refer to the Java API to identify the methods available in order to complete the exercise below. The**LinkedList** class provides all the necessary methods to complete the set tasks easily. To solve this problem create a class **Registry** which manages a linked list of students called **studentList**. This class is outlined below:

```
import java.util.*;
public class Registry
{
      LinkedList<Student> studentList;

      public Registry() {}

      public void addStudent(Student aStudent) {}

      public void deleteStudent(String studentID) {}

      public String toString() {}

      public String format() {}
}
```

For this exercise assume that the add and delete operations are not required to deal with errors or to "report back" to the user whether the operation has succeeded. In particular, assume that the ID value presented to **addStudent** is a value that is *not* already in the list and for **deleteStudent** assume that the ID value supplied corresponds to the identifier of a specific student in the list. You may, of course, if you wish ignore this simplification, and produce a more robust implementation but this is not part of the assessment.

Write a class to test all of the methods in **Registry**. Test runs should have on-screen narrative to explain the purpose of each test and to compare actual results with those expected. For example, part of the test output might look like this.

```
Registry Tester
***************

Test 1. Check student list contains three students.
Methods tested:  constructor and format

Expected:
Steve Marriott          1001PG      BSc Mathematics
Sean Crossan            1002UG      BSc Computer Science
Alan McLachlan          1003UG      BSc Computer Information Systems

Actual:
Steve Marriott          1001PG      BSc Mathematics
Sean Crossan            1002UG      BSc Computer Science
Alan McLachlan          1003UG      BSc Computer Information Systems
```

In reading this problem description you may have realised that the **Registry** class as described does not support permanent data storage because the linked list holding the data is not stored on file. The omission is deliberate because in this exercise I want you to focus on the use of linked list structures rather than file management. The issue of file processing is, however, covered in the ICP 1023 Programming Project.

Exercise 3: Adding a Command Line Interface(CLI)

Create a class called **RegistryCLI** to allow the user to interact with the registry via the keyboard and console screen. For example, when the application starts the user should be greeted with a simple menu system illustrated below:

```
Registry Main Menu
******************

1. Add a Student

2. Delete a Student

3. Print Registry

4. Quit

Select option [1, 2, 3, 4] :> 1
```

Here the user has selected option 1 so they should be presented with an input screen.

```
Add New Student
***************

Enter forename          :> Steve

Enter surname           :> Marriott

Enter student ID        :> 1001UG

Enter degree scheme     :> BSc Mathematics

Enter another (Y/N)     :> n
```

The effect of selecting options 2 and 3 should be obvious but if in doubt ask for clarification. Input for the main menu *must* be validated, if the user enters an *invalid option* they must be notified and then returned to the main menu. A code stub for the CLI class is presented below:

```
public class RegistryCLI
{
      private Registry theRegistry;
      public RegistryCLI(Registry theRegistry){}

      // Displays main menu and gets valid option from user
      public void doMenu() {}
      private void doAddStudent() {}
      private void doDeleteStudent() {}
```

```
    private void doPrintRegistry() {}
}
```

The private methods are "helper" methods invoked from within the **`doMenu()`** method. Note also that **`RegistryCLI`** requires a reference to a suitable **`Registry`** object, otherwise data received through the interface cannot be supplied to the University registration system.

Do *not* use recursion to implement the menu system. Submissions which involve recursion will be *penalised*.

Having completed the interface class – which also lacks a main method - we need a third class to initiate the application. This is listed below:

```
public class RegistryApp
{
    public static void main(String[] args)
    {
        // Create the registry object
        Registry theRegistry = new Registry();

        // Create an interface
        RegistryCLI theRegistryCLI
            = new RegistryCLI(theRegistry);

        // Display the menu
        theRegistryCLI.doMenu();
    }
}
```

****

Assessed Laboratory Work

For this piece of assessed work you are required to complete Exercises 1-3 and submit the following source files:

- `Student.java`
- `StudentTester.java`
- `Registry.java`
- `RegistryTester.java`
- `RegistryInterface.java`
- `RegistryApp.java`

All classes, apart from tester classes, should have Javadoc style comments.

Please see next page for *Submission Notes and Marking Scheme*.

Submission Notes and Marking Scheme

Use **Blackboard** to submit your source code files. The deadline for submission will be published on Blackboard. Late submissions will be penalised in line with School policy.

Marks for this laboratory exercise are awarded as follows:

- Implementation of the class **`Student`** and **`StudentTester`** 25%
- Implementation of the class **`Registry`** and **`RegistryTester`** 25%
- Implementation of the class **`RegistryCLI`** 30%
- Implementation of the class **`RegistryApp`** 10%
- Comments, layout and coding conventions 10%

Students who submit work but have a poor understanding of what has been submitted may be *heavily penalised*. When making a submission it is your responsibility to ensure that all code submitted via Blackboard is:

- Consistent with stated requirements
- Entirely your own work
- Submitted through Blackboard on time

Please note that there are **severe penalties** for submitting work which is not your own. *If you have used code which you have found on the Internet* or *from any other source* then you **must** signal that fact with appropriate program comments.

Note also that to obtain a mark you *must attend a laboratory session* and be prepared to demonstrate your program and answer questions about the coding. Non-attendance at labs will result in your work **not** being marked.