Laboratory Sheet 6

This Lab Sheet contains material based on Lectures 1 – 12 (*up to 29 October 2014*), and contains the submission information for Laboratory 6 (week 7, *3 – 7 November 2014*).

The deadline for submission of the lab exercise is 48 hours after the end of your scheduled laboratory session in week 7 *(3 – 7 November 2014)*.

Of course you may submit work that is incorrect or incomplete. In order to stretch the stronger members of the class, some of the laboratory exercises are quite challenging, and you should not be too discouraged if you cannot complete all of them.

Aims and objectives

- Using `ArrayLists` and the Collections framework in Java
- Designing classes and consolidation of earlier object-oriented concepts
- Implementing interfaces in classes

Set up

When you download Laboratory6.zip from moodle, please unzip this file. You will obtain a folder Laboratory6, containing a subfolder entitled fillingcrates.

In the folder fillingcrates will be the following files:

- FillableContainer.java which defines an interface
- TestCrate.java which provides JUnit tests for the Crate class you must implement
- InsufficientCapacityException.java which defines an Exception subclass
- PackingStrategy.java which defines an interface
- AbstractFit.java which defines an Abstract class that implements `PackingStrategy`
- CratesProgram.java which defines a `main` method

In Eclipse, you should create a new project entitled Lab6. In this project, create a new package called fillingcrates. Drag the Java source code files into this new project.

You will need to create new Java classes as part of your solution to this exercise. These are:

- Crate.java which implements FillableContainer
- FirstFit.java which extends AbstractFit
- BestFit.java which extends AbstractFit
- WorstFit.java which extends AbstractFit

Submission material

Preparatory work for this programming exercise, prior to your scheduled lab session, is expected and essential to enable you to submit a satisfactory attempt.

Submission exercise 6

You are to design and implement a program that will assign items to crates based on the weights of the items and the capacity of the crates.

Each crate has a capacity of 100 kg, and items can have any non-negative integer weight up to 100 kg. Items arrive in an unpredictable order, represented by successive integers in the input. An item is placed in a new (empty) crate only if it is too heavy for any existing (non-empty) crate. The used crates are maintained in the order in which they were first used. Up to three separate algorithms are to be considered, as follows:

The *first-fit algorithm*: an item is placed in the first non-empty crate that will accommodate it, or in a new crate if it is too heavy for any existing crate.

The *best-fit algorithm*: an item is placed in a crate whose spare capacity is nearest to the weight of the item, or in a new crate if it is too heavy for any existing crate.

The *worst-fit algorithm*: an item is placed in a crate with the greatest spare capacity, or in a new crate if it is too heavy for any existing crate.

For example, given the sequence of values

```
75   50   20   60   40   50
```

the output from the required program should be as follows:

```
Number of crates using first-fit = 4
Crate 0 has load 95
Crate 1 has load 90
Crate 2 has load 60
Crate 3 has load 50

Number of crates using best-fit = 3
Crate 0 has load 95
Crate 1 has load 100
Crate 2 has load 100

Number of crates using worst-fit = 4
Crate 0 has load 75
Crate 1 has load 70
Crate 2 has load 100
Crate 3 has load 50
```

For this set of data, the best-fit algorithm wins, in the sense that it uses fewest crates, i.e. 3 as compared to the 4 crates required by the other two algorithms. (It's not hard to see that any one of the algorithms might give the best solution, depending on the nature of the data.)

Hint 1. You should create a class Crate to represent a crate. The required methods for Crate are specified in the interface FillableContainer, which the Crate class should implement. Eclipse can help to generate the signatures and empty bodies for these methods.

Hint 2. Use the provided TestCrate JUnit tests to check that your Crate class is working properly. (Note – this class will initially give compile-time errors because no Crate class is supplied. These should disappear once you create your own Crate class.)

Hint 3. The AbstractFit class has an `ArrayList` instance variable `containers` that represents a stock of crates. You should provide subclasses of `AbstractFit` that implement specific packing algorithms as described above (FirstFit, BestFit and WorstFit). ***Initially aim to implement just one of these, say*** FirstFit – the others can be added later, if you have time. Meanwhile just comment their constructors out in the `main` method.

Hint 4. The CratesProgram class has a `main` method that can be used to check the behaviour of your packing algorithms. When the main method is executed, it should send output to the console like that shown above.

Submission

You should submit your work before the deadline no matter whether the programs are fully working or not.

When you are ready to submit, go to the JOOSE2 moodle site. Click on Laboratory 6 Submission. Click 'Add Submission'. Open Windows Explorer and browse to the folder that contains your Java source code and drag *only* the four Java files Crate.java, FirstFit.java, BestFit.java and WorstFit.java into the drag-n-drop area on the moodle submission page. **Your markers only want to read your java files, not your class files.** Then click the blue save changes button. Check the four .java files are uploaded to the system. Then click submit assignment and fill in the non-plagiarism declaration. Your tutor will inspect your file and return feedback to you via moodle.

Outline Mark Scheme

Here is a rough mark scheme that your tutor will use when assessing your work.

A: Excellent attempt. Code compiles and runs. Code passes TestCrate unit tests. Correct output for main method.
B: Very good attempt. Code compiles and runs, although may not give correct answers in all cases. FirstFit algorithm implementation is correct.
C: Good attempt at solution. Code may not compile, but you have attempted to implement all four specified classes.
D: Minimal attempt at solution. Code may not compile, but you have attempted at least Crate and one of the AbstractFit subclasses.

Within these general guidelines, the tutors are also checking to see whether:

- you use sensible variable names and method names
- you use `@Override` annotations where appropriate
- you write appropriate Javadoc comments for classes and 'interesting' methods
- you write clean and efficient control flow structures (e.g. for each loops) in your code.
- you include an `@author` tag in each source code file
- you are using packages correctly