## APROC Coursework – Notes

These notes are intended as guidance for the coursework development and also to answer some of the most frequently asked questions - they are not intended to be a step by step guide or to indicate the only way of the application development.

* You must adopt an Object Oriented approach to your application - simply writing all of the code within the main method and perhaps calling a few methods within one class is not acceptable.
* The user interface part of the assignment should be in a separate class, so that changes to the user interface do not affect the rest of the application. This will allow you to develop the program as an application and later change it to an applet using the AWT/Swing to construct the user interface.
* A possible design approach is to view the box types as different classes.
* When you create an object you pass the necessary information to initialize it through a constructor, for example, to create an object that represents a box, which the customer wants to order, you can pass to a constructor the necessary information that the customer specifies.
* Your application must determine the type of the ordered box, based on the characteristics/parameters specified by the customer. The clients must not be asked what type of box they want to order.
* When creating your class hierarchy pay attention which methods and instance variables of your superclass should be inherited. If your sub-classes inherit methods or instance variables that make no sense to have them inherited, your class hierarchy is incorrect. If, for example, you have a class that provides your user interface, it is incorrect to use it as a superclass of the classes representing the box types, since these classes do not need their own user interface.
* Adopt an incremental approach to the development of your application. When you have one part working, make a copy of the Java files, so that you can return to this point later if you run into a problem.

One possible design of your application (to start with, not the best one) is to create 5 classes (corresponding to each type) and one interface class (could be even your main class) in which you will prompt the client to enter the necessary order parameters (attributes such as, size (length, width, height), card grade, colour, reinforcements, quantity, etc.).

You should validate and handle I/O exceptions in this class.

To validate your input, you have to accept some intervals (boundaries) for the input data (which you should describe in your report). For example, minimum and maximum size of a box. Say, the *FlexBox* company does not produce boxes with length less than 0.1 meters (10 centimetres) and greater than 2 meters (you can assume different values of course), and say, it can not produce more than 100 boxes of one type for a given order, etc.

When you are designing your interface class, you have to decide what attributes and methods you will include in it. For example, if you decide to work in millimetres, your ‘size’ variable (length) could be of type ***integer*** (but later, when calculating the cost, you will have to convert the area into square meters, because the cost is given per square meter of card (Table 2 and Table 3 of the coursework)). If you decided to work in meters, then the type should be *double*, or *float*, etc.

Once you have validated the user order, your program should determine, based on Table 1, what is the type of the box that was ordered.

***Table 1.*** *Types of cardboard boxes available.*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Type** | **Grade of card** | **Colour print** | | | **Reinforced bottom** | **Reinforced corners** |
| **0** | **1** | **2** |
| I | 1 – 3 | YES | NO | NO | NO | NO |
| II | 2 – 4 | NO | YES | NO | NO | NO |
| III | 2 – 5 | NO | NO | YES | NO | NO |
| IV | 2 – 5 | NO | NO | YES | YES | NO |
| V | 3 – 5 | NO | NO | YES | YES | YES |

How to determine the type?

You can use ‘brute force’ approach, for example, *if (****grade*** *greater than 0) and (****grade*** *less than 4)* *and (no* ***colours****) and all others are ’No’*, then the ordered box is of **type I**.

(There are more intelligent ways to do this (to implement Table1), and you will get additional points if you have found such. For a start, you can work with the ‘brute force’ approach, and later, if you have time, you could try to improve it.)

Once this is done, you can create one box object of the corresponding type. In other words, you may have five classes, each one representing one type from Table1.

For example, for class *TypeI*, you may have attributes *length, width*, *height* and *grade.* You may also have at least two additional methods (apart from the constructors, access, and update methods), one to calculate the total area of a box, and one to calculate the cost.

When you are calculating the area of a box, make sure that you have converted it to squire meters. The area of a box is the sum of the areas of all box sides.

Say, if in your main class you have determined that the client order is a box of type I, then you can create an object of *TypeI* and for this object you can call the *cost()* method to calculate the cost and prompt the user.

At this point, you should have a working version of your coursework and you are on the ‘save side’.

Nevertheless, your application is far from perfect. You will notice that your ‘*type’* classes contain attributes and methods that are repeated, for example, the size data members will be needed in all of them, also, the method for calculating the box area will be the same for all of them, etc.

So, designing appropriate hierarchy (data methods and data fields at the appropriate level of abstraction) will allow you to fully implement and use Java inheritance and polymorphism techniques in your application.

What left is to develop your GUI interface class using AWT/Swing classes and to add it to your application (if you prefer, you can also start with developing your interface first).

Don’t forget to hand in your coursework report on **6.XII.2016**, and to give a demonstration and submit a disk (with your group student numbers on it) with your Java source code and project files (unless it has been already submitted to the CAM office with your group report).