COM6516 Object oriented programming and software design: Practical session 2

The aim of this practical is to introduce you gently to the object oriented features of the Java language. You might like to work with another student during the practical session, but make sure that you write your own code and understand the material for yourself.

Part 1: Review of last week's code

Take a look at the example solutions to last week's practical exercises on Blackboard. Compare the solution to the `CycleComputer` task to your own. Do they produce the same output? What are the good features of each program? How could you improve on your own program?

If you are new to Java then make sure you look through the material on the main features of the Java programming language that are covered in the slides from last week's lecture.

Part 2: OO programming (basics)

Java is an object oriented (OO) language. A Java program therefore consists of objects that interact to deliver the program functionality. If you have not programmed in an OO language before, it may take you a while to get used to an OO approach to programming. However, you will find that it is a powerful and flexible way to write good quality software. If OO programming is new to you, then Core Java chapter 4 is a good introduction, and you should also read the tutorial at – http://download.oracle.com/javase/tutorial/java/javaOO/

A quick introduction to classes and objects in Java

The basic idea behind OO programming:

- An OO program is made up of objects with certain properties and which can perform certain operations.
- The state of the program changes by objects interacting with each other in well-specified (and documented!) ways.
- You can build your own object (programming) or use a pre-existing object.
- In OO programming you care about an object's interface to the outside world, not its inner workings.

Some important vocabulary associated with OO programming is

- **Class:** A class is the blueprint for an object, specifying the data it contains and the methods that make up its interface to the outside world.
- **Object:** An object is an instance of a class, and is constructed by calling the class constructor.
- **Encapsulation:** This means combining data and behaviour in an object, such that the implementation of the data is hidden from users of the object. Users must access the object using its publicly available methods.
- **Extension:** A class that is built on top of another class extends it.

The best way to understand these concepts is to experiment with some simple objects, and that is what we will do next.

Programming task 1

Your task is to study and modify a simple class that represents a storage container of 'food' (e.g. grain). This class initially has an attribute for the current amount in the storage container. This attribute is set by the class constructor (a method that instantiates new objects of the class), and can be displayed and modified using the class methods.

Open the file `FoodStore.java` (available in Blackboard), and study the code carefully. The comments explain what each part of the class does. Make sure you understand what the different parts of the class do.

This class has no `main` method to provide an entry point to the program, and so we need to write another class to provide this. Your next task is therefore to write a class called `TestFoodStore.java`, which creates `FoodStore` objects, and calls their methods. Your new class should look something like the following:

```
public class TestFoodStore {
    public static void main(String[] args){
        // create a new FoodStore object called MyFoodStore
        // by invoking the constructor
        FoodStore MyFoodStore = new FoodStore(10);

        // display the amount stored by calling the getAmountStored
        // method associated with the MyFoodStore object
        System.out.println("Contains " + MyFoodStore.getAmountStored());
    }
}
```

Compile and run your `TestFoodStore` class.

You should now go on to do the following tasks:

- Modify your `TestFoodStore` class so that it tests the other class methods in `FoodStore`, showing that the amount of food associated a `FoodStore` object is updated correctly.
- Modify your `FoodStore` class so that the number of deposits and withdrawals are recorded as instance variables. If you are not sure what an instance variable is, then look at – http://docs.oracle.com/javase/tutorial/java/javaOO/classvars.html
  Modify `TestFoodStore` to make sure that these attributes are updated correctly.
- Write a new Java program `TestFoodStore2` that uses the `FoodStore` class and processes a set of transactions typed at the keyboard, to simulate deposits (additions) and withdrawals (subtractions) from the food store. You may import the 'sheffield package'.
- The above program should begin by reading the current amount of food in the storage container. Then a loop should be used to read and process a series of transactions. Each transaction should take the form of an integer, `n`. If `n` is positive, the transaction is a deposit of food and the amount in the storage container should be increased by `n`. However, if `n` is negative then the transaction is a withdrawal of food, and the amount in the storage container should be decreased accordingly. For each transaction, your program should display a message indicating whether it was a withdrawal or deposit. Any transaction that would result in the amount of food in the storage container falling below `0` should be refused. When all the transactions have been processed, the program should display the total number of withdrawals, the total number of deposits, and the total number of refused transactions. It should also report the total amount of food deposited and the total amount of food withdrawn from the storage container.
- How would you test whether two `FoodStore` objects are equal?

Part 3: OO programming (a bit more advanced)
The Blackboard page also contains code for the following classes:

- `Basket.java` – this class represents a collection (an array) of Items that are held in a shopping basket.
- `Item.java` – this class represents a single item with fixed price (e.g. a tin of baked beans).

Take some time to look at these classes, and the associations between them. Are these classes correctly documented with comments? Add comments so that the code is understandable, and make sure that the indenting is consistent. Why does the `Item` class have a `main` method and what does it do? Note that the `Item` class is *immutable*, so once an `Item` is created, it is not possible to change the instance fields. Is this a good idea?

Programming task 2
Write a new class called `TestBasket.java` that stores an array of `Item` objects in a `Basket` object. This class should have a `main` method, and you could use code such as the following to initialise the array of `Items`:

```
Item[] shopping = {
    new Item("baked beans", 0.3),
    new Item("tomato soup", 0.4)
};
```

Use the `toString` method to print out the instance field of each element in the array. Then use the `total` method in the `Basket` class to work out the total cost of the items in the shopping basket, and display this to the user. Check that the result you obtain is the result that you expect.

Modify the `toString` methods in the `Item` class to display the class type as well as the object name and instance fields. (Hint: You can use the `getClass` method of the `Object` superclass to do this.)

Programming task 3
(Do not use IDE for this – IDE can do most of this task automatically and you need to learn what the `equals` method does.)

Write an `equals` method for the `ItemWithEquals` class that:

- Returns `true` if the current (`this`) object and the parameter passed to the `equals` method are identical,
- Returns `false` if the parameter passed is `null`,
- Returns `false` if the current (`this`) object and the parameter passed have different classes,
- Tests for equality of the instance fields.

You will need to write a `TestItemEquals.java` class to test that these new equals methods work properly.