University of Stirling
Computing Science and Mathematics
CSCU9P5 Software Engineering I **Autumn 2018**

Practical 1 (for practicals in the week starting 17 September 2018)

In these practicals you should log on using your Computing Science username and password. Your files and all shared materials will be on Computing Science servers.

This practical consists of some basic exercises in object-oriented programming in Java. In later practicals, you will be moving on to Together Architect.

Note: This exercise can be carried out using BlueJ, Eclipse or any other Java IDE (or none). The instructions below refer to BlueJ. If you prefer to use a different IDE, adapt the instructions to suit.

Exercise 1. Outline **on paper** a Java class `BankAccount` that will be used to implement a simple bank account. The class must have an attribute `balance` and the methods `getBalance()`, `deposit()` and `withdraw()`. The attribute will contain information about the current balance of the account and the methods will return the current balance, make a deposit (hence updating the balance) and withdraw some money (hence updating the balance), respectively.

- You need to decide whether the attribute and the methods are public or private, and the parameters and type of returned value of the methods. Also, you need to define one or more suitable constructors for the class.
- Implement the class in Java (either on paper or by using any editor of your choice). Include useful comments in the code documenting what is the meaning/functioning of each method/attribute!
- Check the class you have written – do you spot any potential problems?
- Launch BlueJ, create a new project in a new folder somewhere on your H: drive, create a new class `BankAccount`, and enter the class implementation that you have designed.
- At this point there will not be a "main program" to make use of this class. However, you *could* test your `BankAccount` class using BlueJ's "object workbench" (you may have done something like this in CSCU9A2/3): Right click the class icon in the BlueJ main window and select the constructor, fill in the constructor parameters (if any!), and OK to obtain an instance of the class on the workbench. (You can make several separate instances if you wish.) You can double-click the icons on the object workbench to view their current attribute values using the "object inspector". Now you can run the separate methods within the instance(s) by right clicking on the instance in the workbench and selecting the method that you want to run – again fill in parameters (if needed) and OK to run. Try fetching the balance, making a deposit, fetching the balance again, etc.

  Did you detect any problems?

Exercise 2. Now implement another class, `BankAccountTest` say, that "knows about", and will be used as a main program to test, the `BankAccount` class. This test class will *not* have a Graphical User Interface (too awkward to set up); instead it will simply be a `main` method that carries out a sequence of method calls and display the results on BlueJ's Terminal window using `System.out.println`

Here is an outline for the test class:

```
public class BankAccountTest {

   public static void main(String[] args)
   {
     BankAccount myAccount = new BankAccount(...);
     System.out.println("Initial balance is "
                                 + myAccount.getBalance());
     ...
     ...
   }
}
```

The test class should create a `BankAccount` object, should deposit 1000 pounds and then make two withdrawals of 400 pounds. After each of these operations the test class should print a message containing the current balance, such as

```
The current value of your account is 1000
The current value of your account is 600
The current value of your account is 200
```

Exercise 3. Change the set of tests performed by `BankAccountTest` so that after the initial deposit of 1000 pounds, ***three** withdrawals* of 400 pounds will be executed (each time printing the current balance). Does this create some behaviour that requires some special action? (Would you imagine that you could do *any* number of withdrawals after a 1000 pounds deposit in the real world?).

Think about how such a problem can be solved/how such unexpected behaviour can be dealt with. Implement a possible solution.

Exercise 4. Try to print a `BankAccount` object – by passing it as a parameter to the method `System.out.println`. What happens? How can you explain it? Do it again – is the result different? How and why?

Design and implement a `public String toString()` method in the `BankAccount` class. This method has to return a string describing the current state of the `BankAccount` object, such as `"The current balance is _n_"`, where `_n_` is the current value of the `balance` attribute.

Run the `System.out.println` to print the `BankAccount` object again. Do things change? Can you explain/figure out why?