Introductory Java Programming

Queen Mary University of London
School of Electronic Engineering and Computer Science

Course Code: EBU4201

Lab Sheet 3: OO Programming – Objects and Classes

---

1. Download the files **`Cat.java`** and **`CatTest.java`** from the course website.

   i) Compile and run the two Java files above.

   ii) Add access modifiers to the **`Cat`** class, conforming to the principles of information hiding (see lecture notes)[1]. Change the code in the **`CatTest`** class to assign the **`name`** and **`speed`** values for the cat object, and print out the **`name`** and **`speed`** of the cat object.

   iii) Write a constructor for the **`Cat`** class. This constructor should initialise all instance variables to the values passed in as parameters to the constructor. Rewrite the code in the file **`CatTest.java`** to use the new constructor.

   iv) Create the two **`Cat`** objects described below, using the constructor you wrote in *part iii)* and print out:
   - the **`name`** and **`colour`** of **`cat1`** and call **`cat1`** to run in a straight line for **`10`** minutes;
   - the **`name`** and **`colour`** of **`cat2`** and call **`cat2`** to run in a zigzag for **`5`** minutes.

`cat1`

```
name = "Tom"
tail = true
speed = 500
furType = "short"
colour = Color.BLACK
```

`cat2`

```
name = "Moggy"
tail = false
speed = 400
furType = "long"
colour = Color.WHITE
```

Note: The string representation of a colour (when using the **`java.awat.Color`** class as above), is usually[2] displayed to the screen in the format **`java.awt.Color[r=x,g=y,b=z]`**, where **`x`**,**`y`**,**`z`** are values in the range **`0-255`** representing the amount of **Red**, **Green** and **Blue** in the chosen colour. For example, the string representation of **`Color.RED`** is **`java.awt.Color[r=255,g=0,b=0]`**.

2. Create a class **`Rectangle`** (and store it in a file called **`Rectangle.java`**), such that:

   i) It should have two instance variables **`l`** (the rectangle's length) and **`w`** (the rectangle's width).

   ii) It should have one method that calculates the area of the rectangle and returns it to the caller.

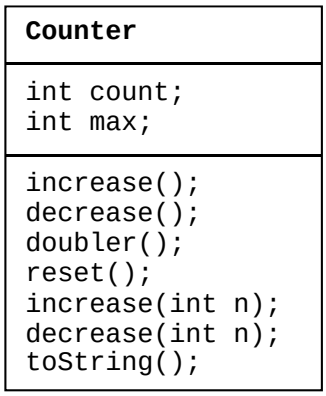
   iii) Create a **`main()`** method *in the same class* for testing purposes. In the **`main()`** method, create one rectangle with the dimensions **`(l,w)=(8,6)`**, and another rectangle with the dimensions **`(l,w)=(7,7)`**. Call your area method on each rectangle and print the corresponding result.

---

[1] You will have to declare instance variables as **`private`** and provide *getters* and *setters* in the **`Dog`** class.

[2] The representation format may be slightly different when running the code on computers with a non-Windows OS.

3. Write a class **`Counter`** that represents a simple counter (as defined in the UML class diagram below) and write a class **`CounterTest`** to test it. Following the steps below may help you.



i) Write the **`Counter`** class which should have two **`private`** instance variables: **`count`** and **`max`**. Add a default constructor to the **`Counter`** class, such that:
- it has no parameters;
- it assigns value **`0`** to **`count`** and value **`10`** to **`max`** as their default values.

ii) Add *getters* and *setters* for the two private instance variables. The *getter* methods should be named **`getCount()`** and **`getMax()`**, whereas the *setter* methods should be named **`setCount(int n)`** and **`setMax(int n)`**.

iii) Write the **`CounterTest`** class with a **`main()`** method. In this class, create a **`Counter`** object and call the *getter* and *setter* methods to test them. Compile and run your program.

iv) Add four methods to the **`Counter`** class:
- **`increase()`** method to increase the **`count`** value by **`2`**;
- **`decrease()`** method to decrease the **`count`** value by **`1`**;
- **`doubler()`** method to double the **`count`** value;
- **`reset()`** method to reset the **`count`** value to **`0`** and print the message **`"Counter Reset!"`**.

Test these four methods by invoking them in the **`CounterTest`** class.

v) Add a **`toString()`** method in the **`Counter`** class and test this method in the **`CounterTest`** class. This could simply display the current values of **`count`** and **`max`**.

vi) Modify the **`increase()`** method so that it resets the counter when the **`count`** value reaches the **`max`** value. Modify the **`decrease()`** method so that it does NOT decrease the **`count`** value when **`count`** reaches **0**. Test both of these methods in the **`CounterTest`** class[3].

vii) *Method overloading*: Add another two methods, **`increase(int n)`** and **`decrease(int n)`**. These methods increment and decrement the **`count`** value by an amount **`n`**, respectively. Test both of these methods in the **`CounterTest`** class.

viii) Check that you have written sufficient comments in your code; you need to write:
- *Javadoc* comments (using notation: **`/** ... */`**) for the class and for each method;
- some inline comments (using notation: **`//`** or **`/* ... */`**) to explain your code if necessary.

Ensure that all your programs contain both internal comments and *Javadoc* comments.

[3] **Hint**: You may want to use a loop in the **`CounterTest`** class to invoke the **`increase()`** and **`decrease()`** methods many times.