Programming in JAVA

lecture 7

Object oriented programming
(static variables and methods, abstract classes)

Static variables

1. A static variable is a variable that is shared by all objects of given class.
2. A static variable can be used even before an object of the class has been created.
3. We use a static variable by preceeding the dot operator by the name of the class (not the name of the object).
4. The **static** keyword is used to specify the variable is static.

Static variable - example

```
public class Test
{
    public static int ITERATION_NUMBER = 8;
}
```

to use static variable :

```
Test.ITERATION_NUMBER
```

Static methods

1. Static methods can be used even before the object of the class is initializated.
2. We call a static method by preceeding the dot operator by the name of the class.
3. The **static** keyword is used to specify the method is static.

Static methods - example

```
public class MathFunctions
{
  public static double pow(double number ,
                           int power)
  {
        double result = number;
        for(int i=0; i<(power-1) ; i++)
            result = result*number;

        return result;
  }
}
```

to call static method :

```
MathFunctions.pow(5,2);
```

Static methods

1. Static methods can be used to program in a typical procedural manner.
2. The function main is static, as it is required that it is called before any object is created.
3. Static method can't operate on non-static class fields.
4. You used static methods and variables to create MessageBoxes in lecture 4.

Abstract classes

1. One or more class methods can be left undefined.
2. Undefined methods can are named as abstract methods.
3. We use abstract keyword to define an abstract method.

Example 1

Abstract class - example

```
abstract class Vehicule
{
   public abstract void move();
}
```

Abstract class - example part. 2

```
class Boat extends Vehicule
{
   public void move(){
      System.out.println("Sailing");
   }
}

class Van extends Vehicule
{
   public void move(){
      System.out.println("Driving");
   }
}

class Excavator extends Vehicule
{
   public void move(){
      System.out.println("Digging");
   }
}
```

Abstract class – example – part 3

```
Boat       vehicule1    = new Boat();
Van        vehicule2    = new Van();
Excavator  vehicule3    = new Excavator();
vehicule1.move();
vehicule2.move();
vehicule3.move();
```

Abstract classes – example information

1. The Vehicule class is the base class and contains one abstract method **move()**.
2. The method move() is left undefined in the Vehicule class. You can see a semicolon just after the list of parameters. There is no function code inside the brackets and the keyword abstract is used.
3. Classes Boat, Van and Excavator inherit from Vehicule class, so they must implement the method move().
4. By using abstract methods, we force a group of classes to have a common interface (method ,move() in this example).

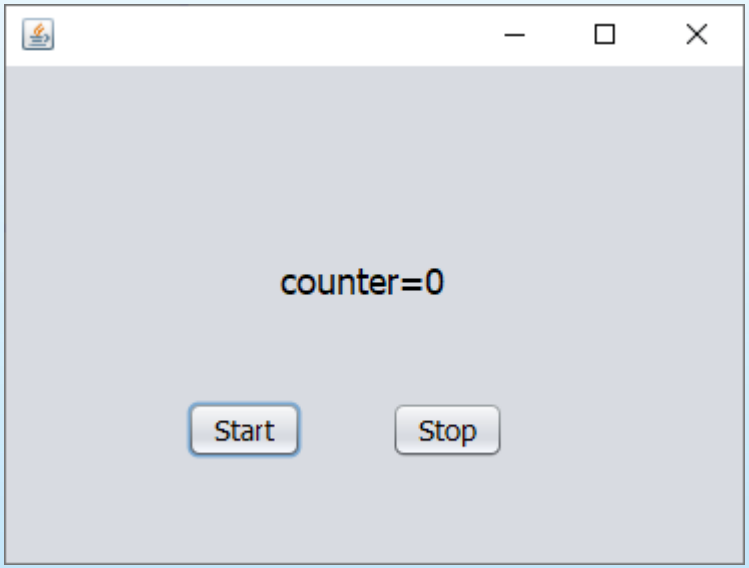
Example 2

using the Timer class

Timer class

1. The `Timer` class is used to measure elapsing real time.
2. The task of measuring time is done to execute some code after some defined time.
3. Code to be executed should be defined as code inside implementation of abstract method.
4. This abstract method is called `run()` and it is declared inside a class `TimerTask`.
5. An object of a class that extends `TimerTask` should be created and a reference to this object should be passed to the method `schedule` of `Timer`.

Timer example



Timer example

```
8  import java.util.Timer;
9  import java.util.TimerTask;
10
11 public class NewJFrame extends javax.swing.JFrame {
12
13     Timer timer;
14     int counter=0;
15
16     private class MyTimerTask extends TimerTask
17     {
18         @Override
19         public void run() {
20             counter++;
21             jLabell.setText("Counter="+String.valueOf(counter));
22         }
23     }
24
```

Note!

When choosing FixImports , make sure that java.util.Timer is selected

(not java.swing.timer)

Timer example

Event handling functions for Start and Stop buttons :

```
98
99      //Start button
100     private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
101         if (timer==null) {
102             timer = new Timer();
103             MyTimerTask myTask = new MyTimerTask();
104             timer.schedule(myTask,0,1000);
105         }
106     }
107     //End button
108     private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
109         timer.cancel();
110         timer=null;
111     }
112
```

Paremeters of schedule methods are:

1. Reference to TimerTask object.

2. Offset (in milliseconds), meaning: after what time the Timer starts.

3. Interval (in milliseconds), meaning: time interval for the Timer to execute code in TimerTask