CS620c
Static methods and static class variables

Joe Duffin

Callan Building Room 2.108

Email: Joseph.Duffin@nuim.ie

Static or Class variables or methods

- Static should really have been called “class”
- Methods and attributes which are Static can have one “version” of them. (E.g. A static variable which counts the number of objects created for a given class).
- We will see later that non-Static items can relate to any number of specific items

Java wants a main method to run an application.

- The main method is that start for all Java programs so it must be publically accessible hence the **public** access modifier

- The word **static** specifies how to access the main method. With a **non-static** method you must do some extra work prior to accessing it (create an instance of the Class, instantiate a class == an object)

- A static method on the other hand can be accessed immediately without doing any extra work. (Use the Class name followed by "." followed by the method name.)

Static variables (Class variables)

- So far all variables were contained within a method. (This includes the main method)

- However, it can be helpful to have a variable accessible to all methods in a class.

  - Without being passed as an argument
  - Variables declared in this way have scope which covers the whole **Class body { }**

Remember this example from Lesson 5: Class Scope of a class variable (use now)

```
public class TestScope
{
    // class variable
    // This variable x will be in scope through out the class.
    // This means that it can be accessible in any method defined in this
    // class TestScope; including the main method and any other methods defined
    // inside TestScope. Notice the colour scheme BlueJ uses to indicate variable scope.
    public static int x=0;

    public static void main(String []args){

        x  = 25;

        System.out.println("the value of x is : " + x);

        printValue();

    }

    public static void printValue(){

        System.out.println("In the printValue method the value of x changed to : " + x);

    }
}
```

Global scope of the class variable x.

- The ***variable x*** is now a shared variable that is usable by all methods in that class.
- The main method changes the value of x to 25
- The print method just prints out the value of x to the screen.
- **Notice** we did not have to pass the variable x to the method. It was accessible due to its **global scope** or scope throughout the Class.

Using other Classes and **public static** Methods

- In lectures and in the lab assignments we have written many different methods.
- These are not just dead archives of completed work!
- We can re-use these classes and methods in other programs.
- See the examples in Moodle of how TestStringProcessing uses the StringProcessing class and MyTester uses MyMaths class.

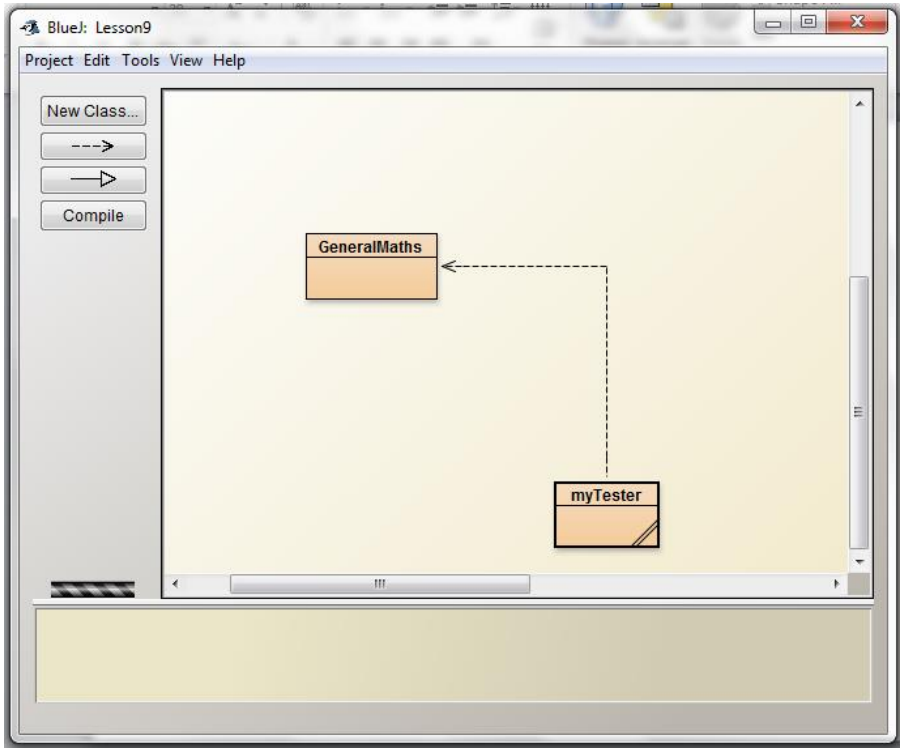
Reusing the `mySigma()` Class

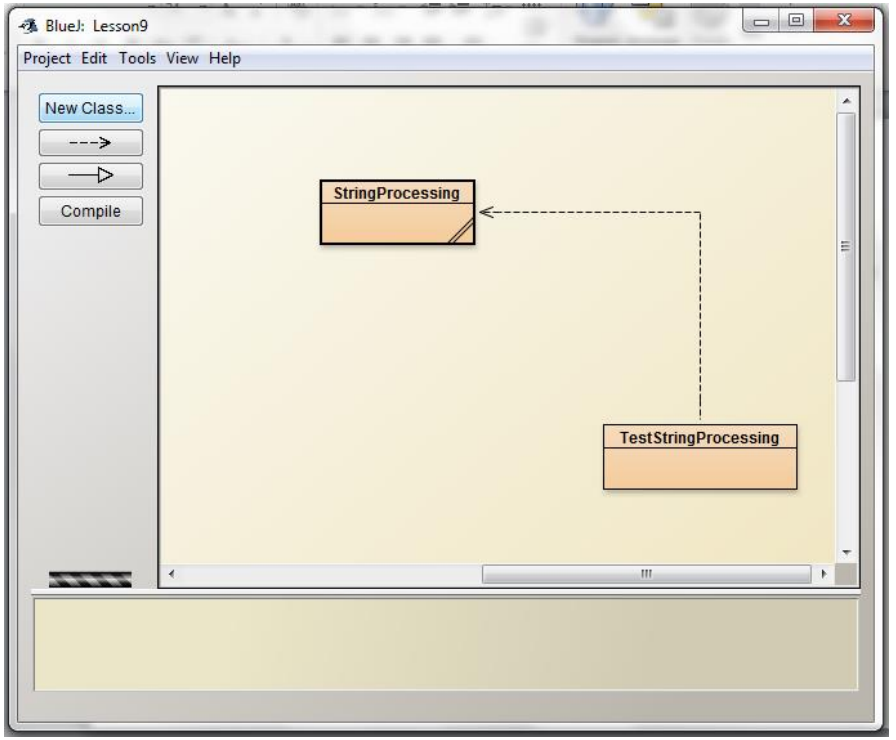
- This week we wrote a *mySigma(int x)* method in one of your programs (class files).
- Another class can easily make use of this method

```
public class MyTester{
  public static void main(){
    int result =0;
    result= YourClassName.mySigma(7);
    system.out.println(result);
  }
}
```

Using other Classes/Methods

- Public static methods written in one class can be used by another class by calling the method using the Class name "." method name





`public` vs `private`

- When writing method for one class, we can control what other classes use that method
- Method are usually defined as **public**
- Attributes are usually defined as **private**
- However, we might want to ensure some methods are NOT used by other classes
- private static *type name*()
- public and private are called Access Modifiers

Significance of public and private

- private methods can Not be seen by other classes
- However, other methods in the same class can still see & call private methods from the same class
- So “private” really only applies to other classes, not within a class

Can methods call other methods.

- One method can call any other method
- The main method of the *GeneralMaths* can call the *mySigma()* method
- Within a class, it does not matter if the other methods are public or private
- That only matters between classes

Reusing method names **(overloading)**

- You can define two or more methods within the same class that share the same name, as long as their parameter declarations are different.
- These are called **overloaded methods**
- Such methods can be distinguished by
  - **The types of their arguments/parameters**
  - **The number of arguments/parameters**

Overloaded Methods

```
public static void timesTwo(int y)
{
   System.out.println(y*2);
}
```

```
private static void timesTwo(double y)
{
   System.out.println(y*2.0);
}
```

Using overloaded Methods

- It is up to Java to make sure it calls the correct method
  - All it has to do is check the types of the arguments.

```
public static void main()
{
    twoTimes(2);
    twoTimes(2.0);
}
```