

Lab Week10. Console Applications

Last Week

- Designing & Implementing Multiple Classes for a more complex problem
 - Constructors & methods
- Creating instance objects and using them for a simple game
- Detecting a crash using pixel colours over an area

Learning Objectives

- Familiarisation with Eclipse
- Importing a Processing sketch into Eclipse
- Console applications
- Main
- Static methods (procedures and functions)
- Input and Output Streams


Resources

- Lecture Notes – Console Applications
- Oracle Java Tutorials
<https://docs.oracle.com/javase/tutorial/java/index.html>
- Lynda Courses (see moodle)


[open any portfolios for signing off later]

Start **Eclipse** using your spotlight. Ensure the workspace is set to your Programming directory. Follow instructions below, or watch screencast on moodle.

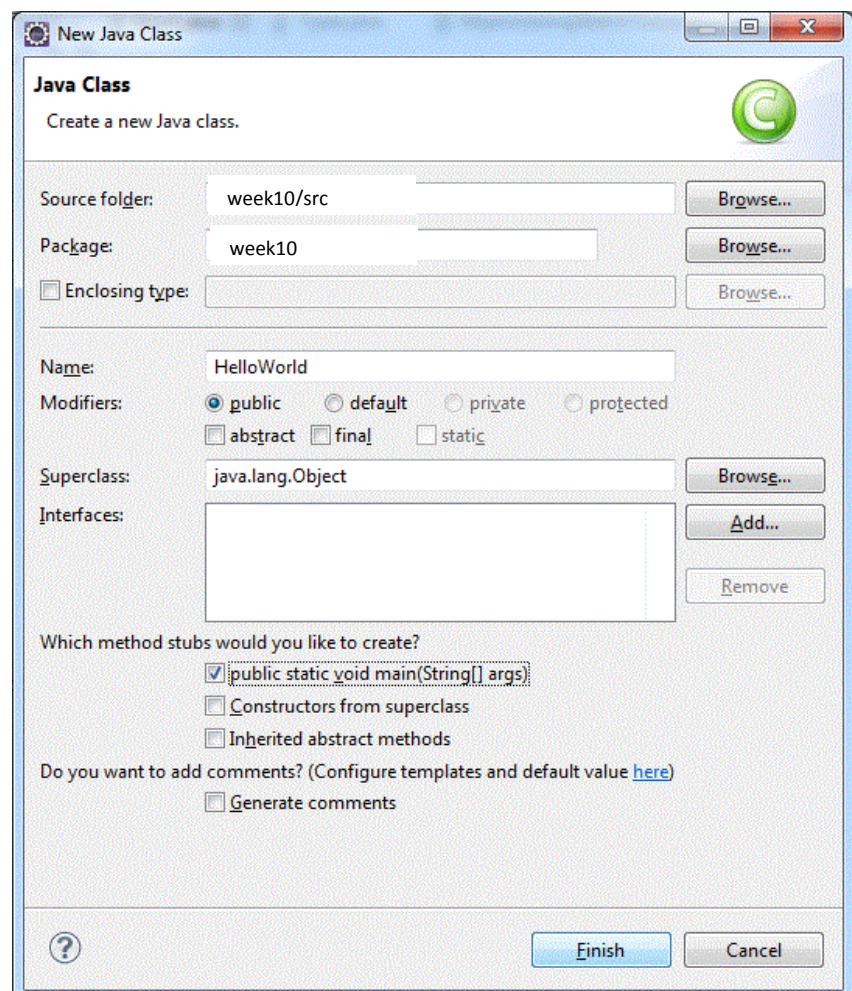
(File - menu) select **New** then **Java Project**

 **Java Project** call it '**week10**' (we'll do this for each consecutive week from now on)

Right click on **week10** select **New** then

Class  **Class** call it **HelloWorld** (see image right) and tick the **public static void main box**.

Then **Finish** button (see image right)



Amend your file to include the two print lines below.

```
public class HelloWorld {  
  
    public static void main(String[] args) {  
        System.out.print("Hello ");  
        System.out.println("World");  
    }  
}
```

Save and run your console application.

Exercise 1. Introduce **3 variables** for name, age, height – initialise these with values to represent you and produce an output of the form:

My name is Fred Bloggs

I am 18 years old.

I am 1.75 meters tall.

Where the values displayed come from the variables you set.

Note: the default floating point number type in Java is a double, so if you want to assign a value to a variable of type float we must cast it, e.g.

```
float number = (float) 5.67;
```

we met casting before with the get command in processing.

Exercise 2. Importing a processing file into Eclipse

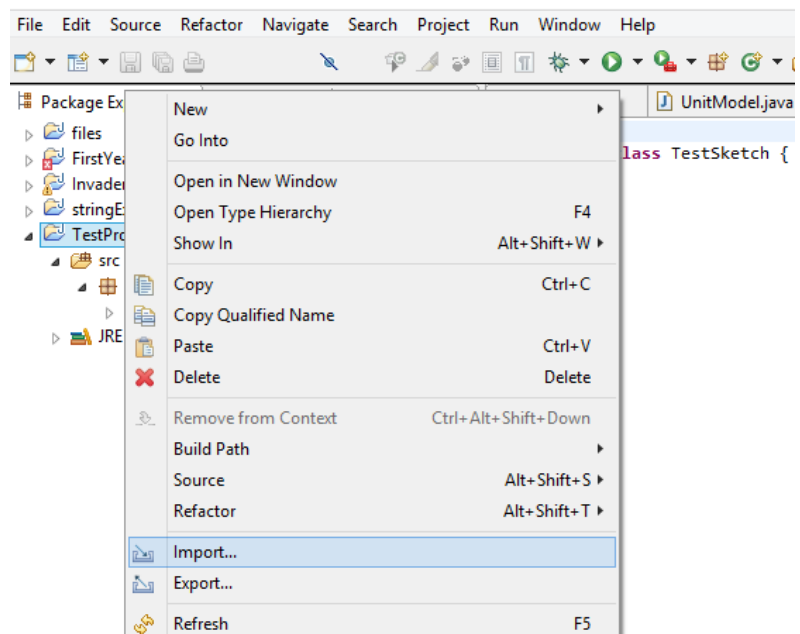
You can use Eclipse with our existing Processing code by importing a library and adding a few extra statements to the code (which Processing hides, for simplicities sake).

In your programming directory create a new **Java project** as we did earlier, called **TestProcessing**. Now click on 'File' and 'New', and select 'Class' (fourth from the top). This will bring up the 'New Java Class' window. Type the word **TestSketch** in the 'Name:' text-field.

Download file **core.jar** from Moodle and place this file inside your Programming directory. This is a Java library that contains the Processing drawing commands.

In Eclipse Right click on the project name (in the package explorer, left) and select Import (see right).

- Click on the **General**, then the **File**

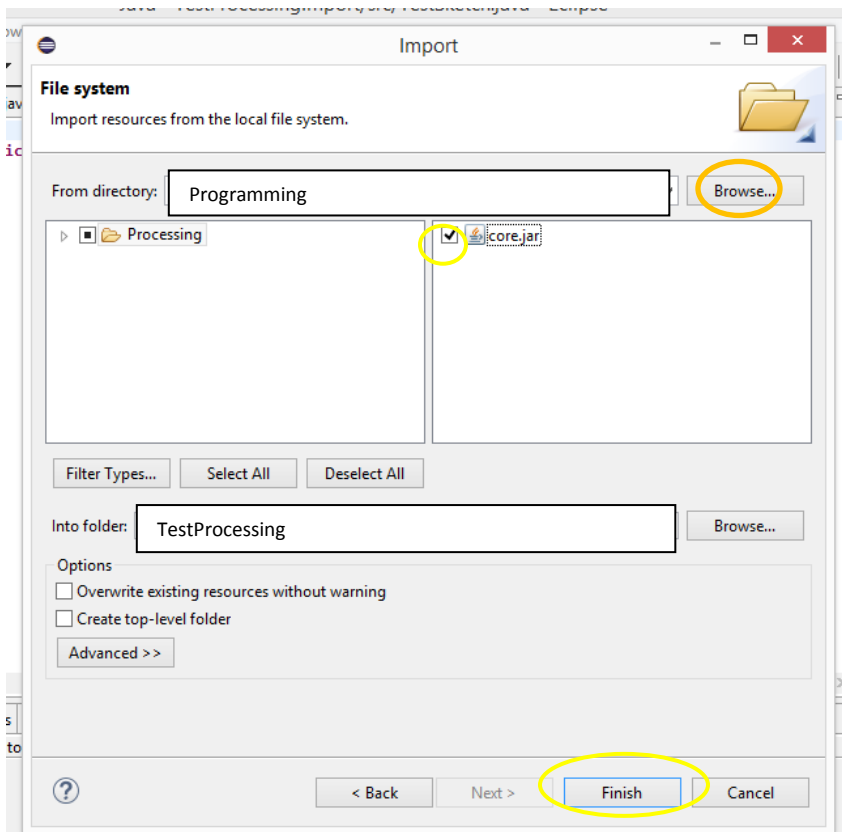


System directory and click on **Next**

- Click the browse button (see image below right) and navigate to your Programming directory.
- core.jar will appear in the window on the right – select the tick box next to it (see image below), then click finish

- **core.jar** will appear in the package explorer.

Right click it, select **Build Path** and **Add to build path**



In the week8 lab we created a Walking person sketch. There is a **zip** file of a completed sketch and the image files on moodle called walkingMan.zip. Download, unzip and open your sketch file which included a class called Walker. Copy and paste ALL the code into the class file testSketch.

Add the import line to the top of your file, and add **extends PApplet** to testSketch as below

```
import processing.core.*;
public class testSketch extends PApplet{
```

Find void setup() and void draw(), add the word **public** in front:

```
public void setup()
```

do the same for draw()

```
public void draw()
```

Now we will import the images the class makes use of walk1.gif .. walk4.gif, either copy these files into your **Programming/TestProcessing/bin** directory, or as before you can use the import file system and import them into the directory above (has the same effect).

Save and run your application. If prompted select **Java Applet**

Ex3. User Input. Using the scanner class (as discussed in the lecture) produce a robust program (input validation) that will allow a user to enter their Name, Age and Height and display these values to the screen.

Ex4. Averages. Write a program that will allow a user to enter 5 integer numbers and display the total and the average of the 5 (both as floating point values).

Ex4. B) Amend your code to allow any number of inputs. You can use a sentinel value of -99 to indicate that input has ended.

Ex 5. Create a console application that will allow the user to enter two validated integers and output the sum, the product and the difference. You should use functions or procedures with parameters to :

- validate the input (1 parameter, return the result)
- Calculate the sum (2 parameters, return the result)
- Calculate the product (2 parameters, return the result)
- Calculate the difference (2 parameters, return the result)

Extension exercises. Add a power function to your code

$$X^1 = x$$

$$X^2 = x * x$$

$$X^5 = x * x * x * x * x \text{ etc.}$$

Add a factorial function to your code (google it if unsure) above, use the first number as the parameter.

Note: Using Multiple Processing Class files in Eclipse

Normally when we have multiple classes in a Java program we would use separate class files for each. This aids reusability (a core principle of OO programming). So we will go through the steps necessary to move Walker into a new Class file. Walker code contains some commands specific to Processing, e.g. **image** and **loadImage**. When we move our Walker class outside of a sketch file, it needs to know which PApplet window its drawing to.

In your Walker file amend to include the import statement and add the new member called sketch as below

```
import processing.core.*;

class Walker {
    PApplet sketch;
```

Sketch is a variable that contains an instance of a PApplet class (actually a reference to an object) the one we will be drawing to.

Our constructor will also be amended, here we'll add a new parameter, telling sketch which PApplet instance the Walker instance belongs to (see bold below).

```
Walker(PApplet parent, int x, int y, int dx){
    this.sketch = parent;
    this.x = x;
    this.y = y;
    this.velocityX=dx;
    image1 = sketch.loadImage("walk1.gif");
    image2 = sketch.loadImage("walk2.gif");
    image3 = sketch.loadImage("walk3.gif");
    image4 = sketch.loadImage("walk4.gif");
}
```

Note where we have used a Processing command **loadImage** we now use **sketch.loadImage** instead.

Also in your render method – any processing commands will also need to be preceded by **sketch**. As below (complete and check it runs).

```
void render(){
    if (counter>=0 && counter<10)
    {
        sketch.image(image1,x,y);
    }
}
```

Finally as we altered our Walker constructor with an extra parameter we need to add an extra value when we call this constructor, e.g.

```
walter = new Walker(this, 10,100,3);
```

this refers to this PApplet the sketch screen we are drawing to (draw() and setup() refer to).