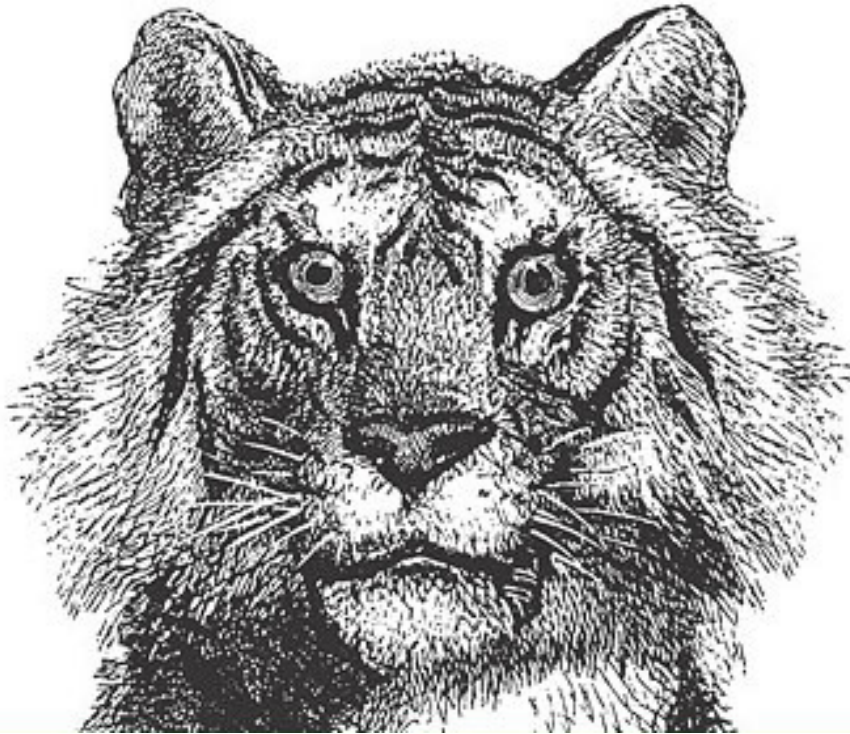


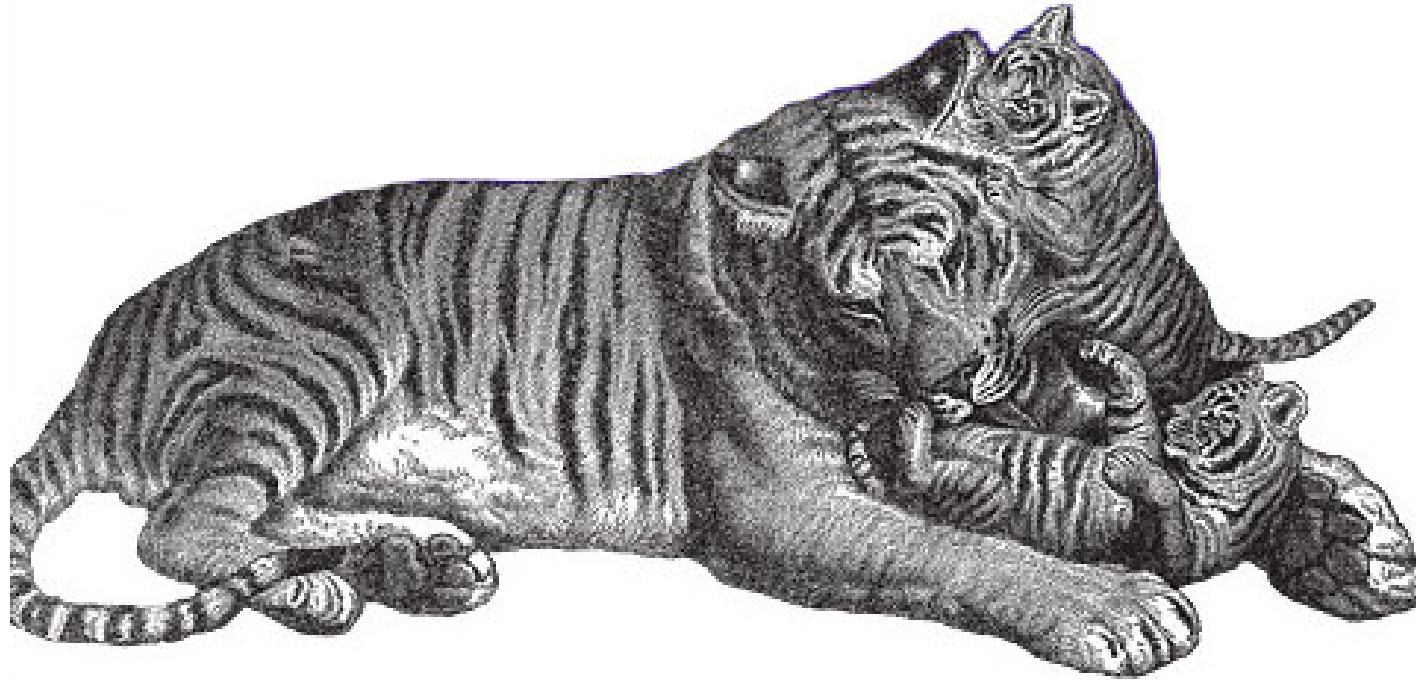
BASIC JAVA

Today

- We will kick off by getting you in a position to write some simple Java programs.
- Then we'll go next door and you'll write some.
- We'll start by looking at some of the things you can do with Java.
- Then we'll stress some of the differences with C/C++.



- Lots of this material is drawn from *Java in a Nutshell*, David Flanagan, O'Reilly ...



- Other material is drawn from *Learning Java*, Patrick Niemeyer, Jonathan Knudsen, O'Reilly.



- Java was developed in the early 1990s (released 1995) by Sun Microsystems.
- Sun is the second most famous spinoff from Stanford University.



- Originally developed for interactive TV, but couldn't run on the simple hardware at the time.
- Now runs on an estimated 3 billion mobile phones.



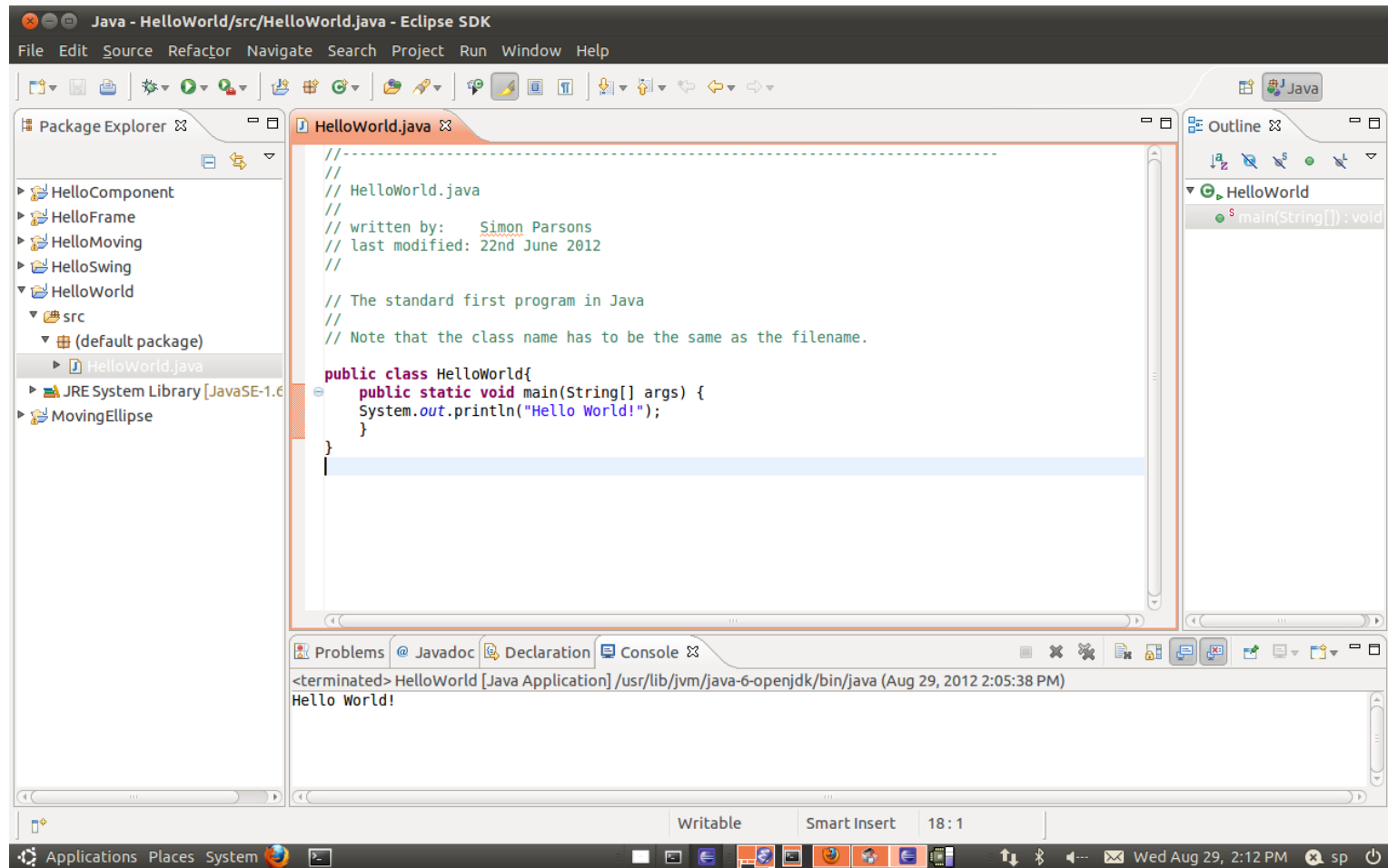
- In January 2010, Sun (and hence Java) was acquired by Oracle.

Hello world

- Here is the standard “hello world” program written in Java:

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

- The next slide shows how it looks in Eclipse, the IDE that we'll be using in this course.



- And now, let's look at the IDE in action

Interpretation

- One of the key advantages of Java is that it is an *interpreted* language.
 - Compiled code runs on a virtual machine that runs on the computer hardware.
- This abstracts away the detail of the hardware.
 - Leads to highly portable programs.
 - “Write Once, Run Anywhere”
 - Leads to some differences with C and C++
- The compiler generates *byte code* that the virtual machine executes.

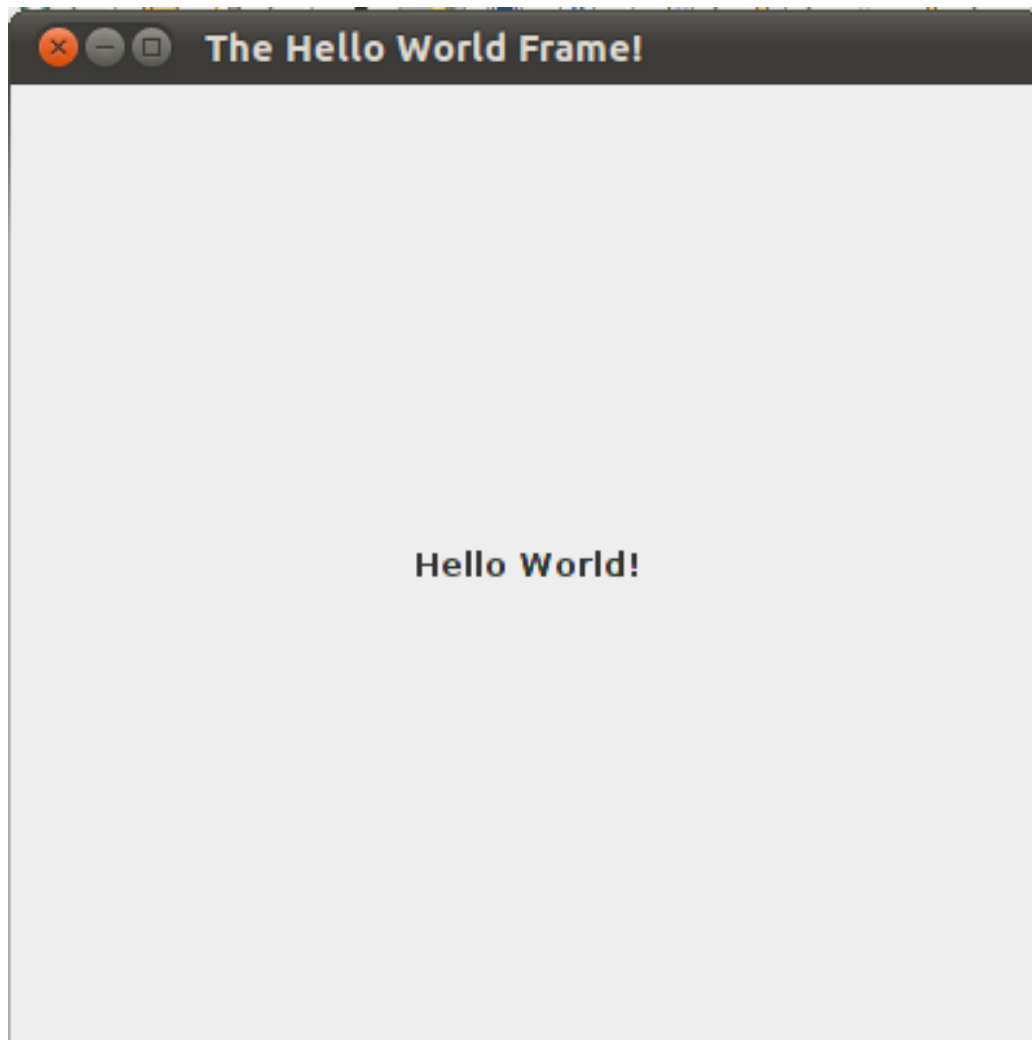
Program structure

- A Java program consists of one or more class definitions.
 - Java is a pure object-oriented language.
 - It is not possible to write programs without objects.
(You can, of course, write a program that is one great big object, not recommended).
- In our “hello world” program, there is one class:
 - `HelloWorld`.
- Somewhere in one of the classes there must be exactly one `main` method/function.
 - Here it is (naturally) in `HelloWorld`.
 - Later we’ll see examples with more than one class — they still have just one `main`.

- Java classes are contained in files called `<something>.java`.
- The convention that Eclipse enforces is that each class must be in a file that has the same name as the class.
 - In other words, our class `HelloWorld` has to be in a file called `HelloWorld.java`
- When the compiler is run on this file, it creates the byte code file `<something>.class`.
 - In other words, when our class `HelloWorld` is compiled, the byte code file is called `HelloWorld.class`

Why Java?

- Java makes a number of things very easy to do.
 - The language itself is small and simple.
 - However, there is a very large set of libraries.
- As an example, consider the program `HelloSwing`.
 - This uses the `swing` library to prints the `Hello World` text in a separate window.
 - Java calls this window a *frame*.
- The output is on the next slide.



- HelloWorld is a simple example of a *graphical user interface* (GUI)
 - a very simple example :-)
- Java also makes it easy to generate *applets*
 - Programs that run in a web browser
- Graphics and networking are also straightforward, because of all those helpful libraries,
- That is why we use it for this course.

Dissecting HelloSwing

- Now, what do the various bits of HelloSwing do?
- `JFrame frame = new JFrame("The Hello World Frame!");`

This creates a variable `frame` which is an object of type `JFrame`

`frame` is then initialised, and its title set.

- `JLabel label = new JLabel("Hello World!", JLabel.CENTER);`

This creates a variable `label` which is an object of type `JLabel`

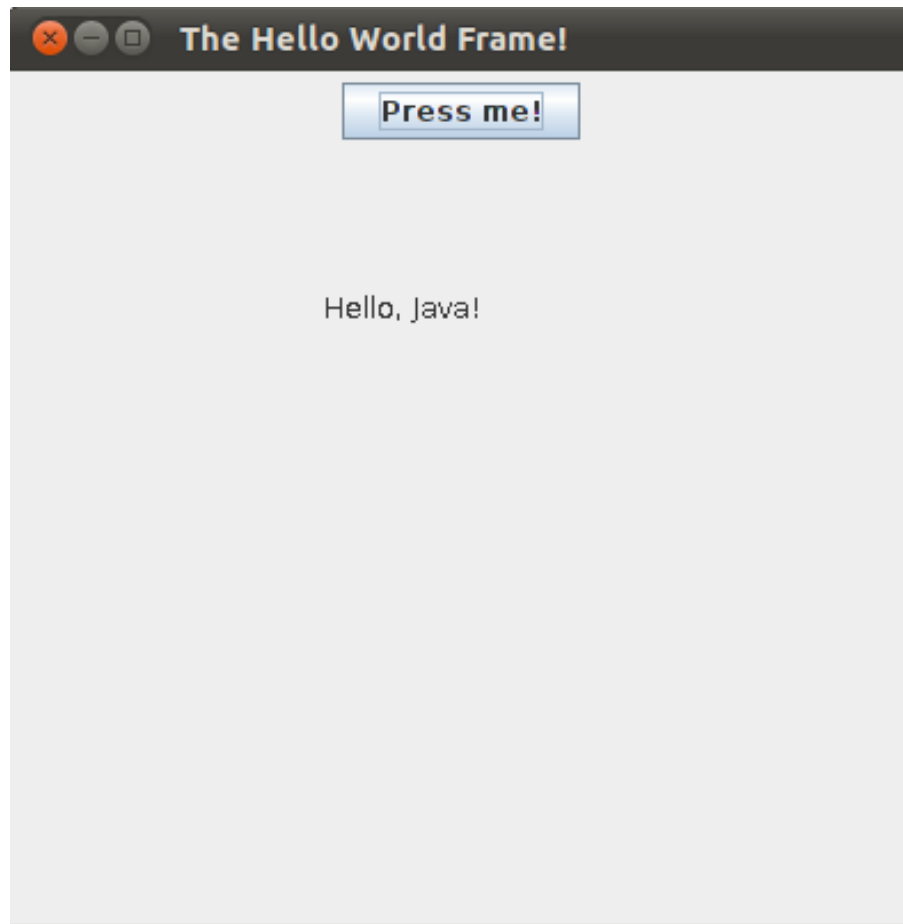
`label` is then initialised to hold the text `Hello World!`

- `frame.getContentPane().add(label);`
 - `label` is then added to `frame`
 - Note that the dot notation is similar to that of C++
 - We pass the output of the method `add` to the method `getContentPane()`.
- `frame.setSize(400, 400);`
 - the size of `frame` is set.
- `frame.setVisible(true);`
 - `frame` is made visible.

A multiclass program

- The examples we have seen so far are *extremely* simple.
- One aspect of the simplicity is that they only have a single class.
- Our next example has more
 - Well, two.
- The Eclipse project for this example is called `HelloMoving` since it takes the previous example and makes the text move.
- Since each class has to be in a file with the same name, naturally this means that we have multiple files.
- A neat thing about the Java compiler is that it figures out all the dependencies.
 - A nice change for those used to handling this in C or C++

- Here's what the window looks like:



- This is more of a GUI since it responds to what the user does.

- Here we have two classes:
 - HelloMoving
 - MovingComponent
- helloMoving is the same as HelloSwing, but instead of:

```
JLabel label = new JLabel("Hello World!",  
JLabel.CENTER);  
frame.getContentPane().add(label);
```

It has:

```
frame.getContentPane().add(new  
MovingComponent());
```
- So it replaces the JLabel object with a new kind of object MovingComponent

- So, rather than using the library object `JLabel`, we define our own component.
- This is what is in the `MovingComponent` class.
- The key parts of `MovingComponent` are explained on the next couple of slides.

```
public class MovingComponent extends JComponent  
implements ActionListener
```

- Our class definition is a modification of JComponent.
- Our class responds to user actions.

```
public MovingComponent() {  
    theButton = new JButton("Press me!");  
    setLayout(new FlowLayout());  
    add(theButton);  
    theButton.addActionListener(this);  
}
```

- This is code that gets run when a `MovingComponent` is created.
 - It is a *constructor*
- It creates a button that says “Press me!” and associates a listener with it.

```
public void paintComponent(Graphics g) {  
    g.drawString("Hello, Java!", msgX, msgY); }
```

- Put the message “Hello, Java!” at a specific point in the frame.

```
public void actionPerformed(ActionEvent e) {  
    if (e.getSource() == theButton) {  
        msgX++;  
        msgY++;  
        repaint();  
    }  
}
```

- If an action happens, and it is the pressing of the button then change the coordinates of the message and repaint the window.

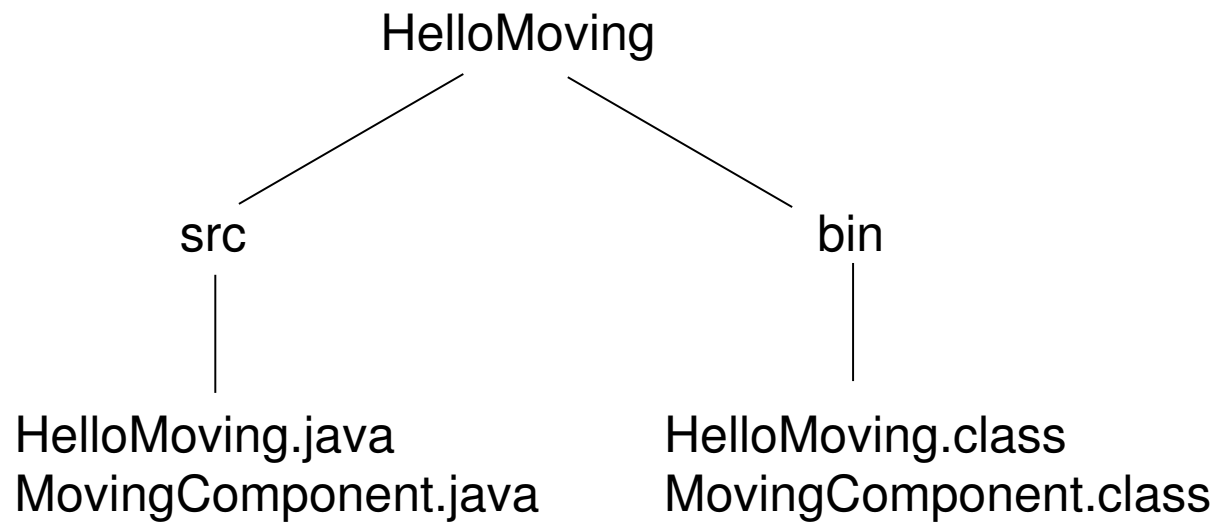
Onwards and upwards

- As I (should have) already said, this course does not aim to teach you all of Java.
- Rather it is going to prod you to pick up some aspects of Java for yourselves.
 - By having you learn them to finish the homeworks
- To help you, there is lots of information online.
- A good place to start is the Oracle Java documentation:

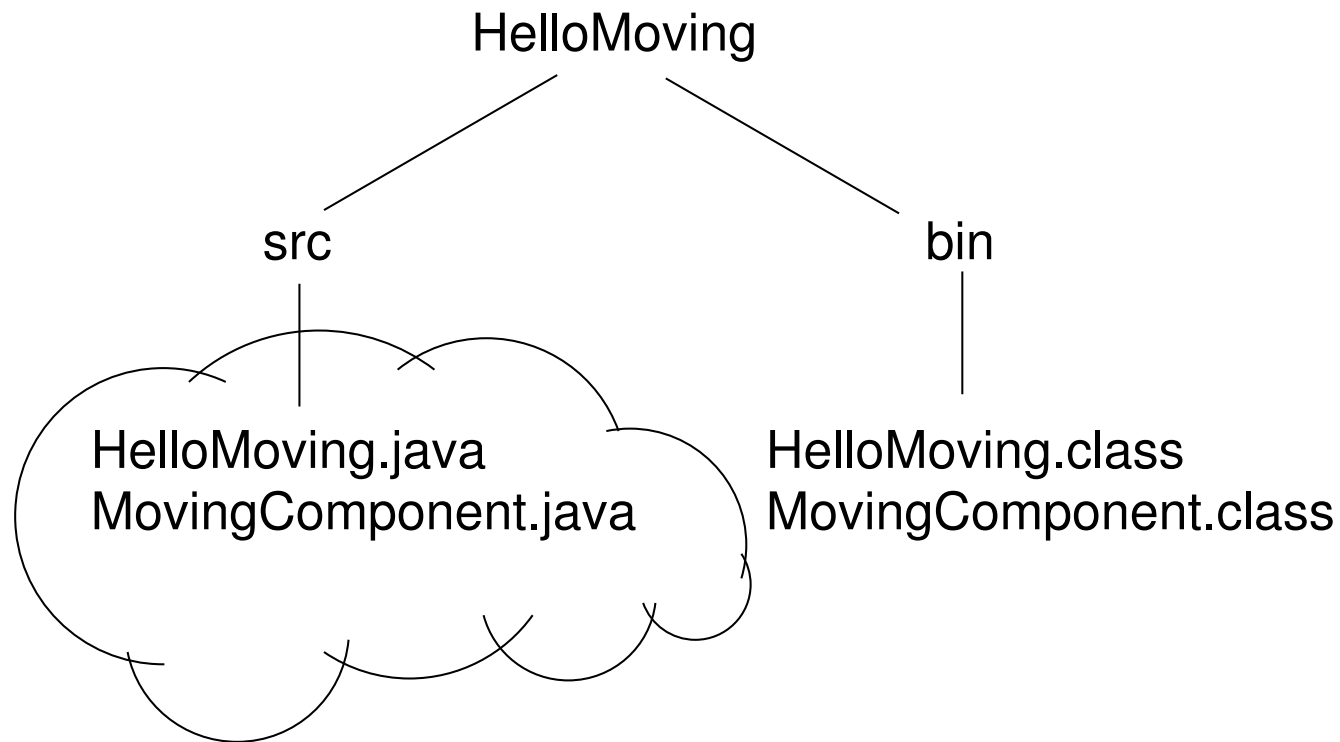
`http://docs.oracle.com/javase/7/docs/api/`

A note on homework

- I will post the first homework over the weekend.
- It will involve taking `MovingComponent` and modifying it.
- You will submit your work to me electronically.
 - Instructions on what you need to do will be included with the homework.
- To do this you will need to extract the `.java` files from the Eclipse project.
- The next slide shows you what you need to do.



- I need the `.java` files for every homework.



These are the ones I need

- I need the . java files for every homework.

Summary

- Today we covered some Java basics
 - A few simple programs
 - A very basic GUI
- This is enough to get you writing simple Java programs
 - Which you will have to do for the homework
- More next week.