SEAtS: 812900

University of Southampton

My Engagement Class Code: 812900

SEAtS: 812900

University of Southampton

Introduction to Java

Son Hoang
(adapted from Prof David Millard's slides)

COMP1202 (AY2023-24)

University of Southampton

Content

- How Java Works: The JVM
- Writing a Class in Java
  - Class
  - Member Variables
  - Methods
  - Statements
- Magic incantations
  - The main() method
- A First Example
  - Defining an `Account` class
  - `if` and Boolean operations
- Introducing the Toolbox

SEAtS: 812900

SEAtS: 812900

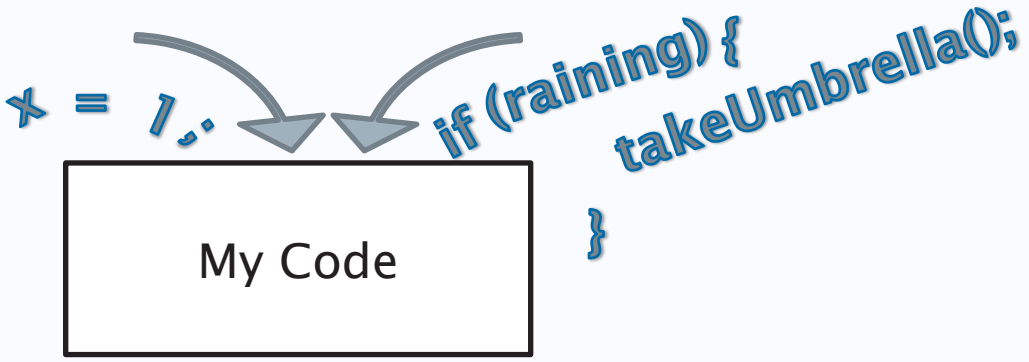
University of Southampton

Part 1

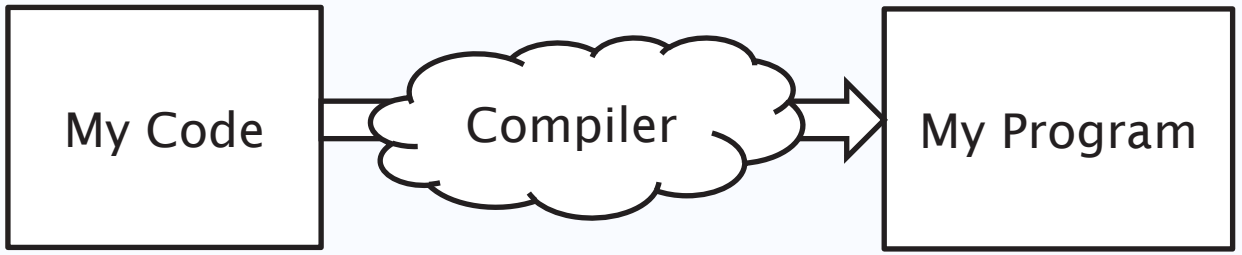
How Java Works

From Code to Program

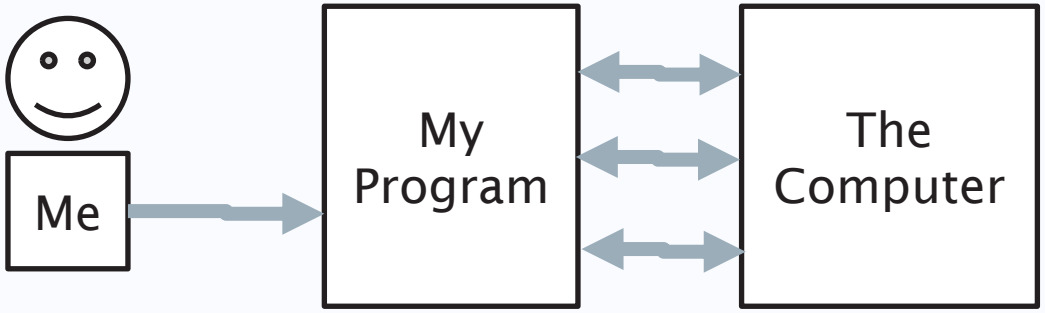
- You **write** code...



- You **compile** code



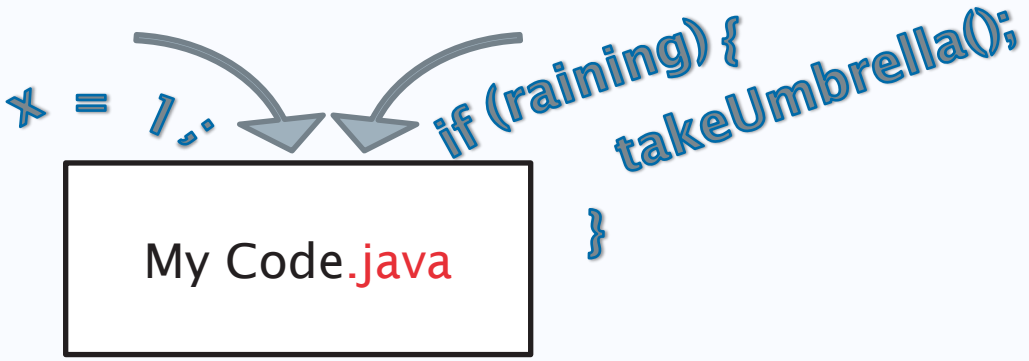
- You **run** the program



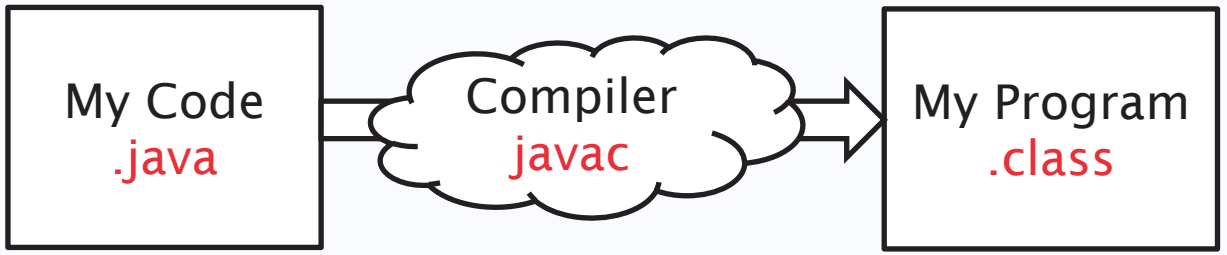
SEAtS: 812900

From Java Code to Program

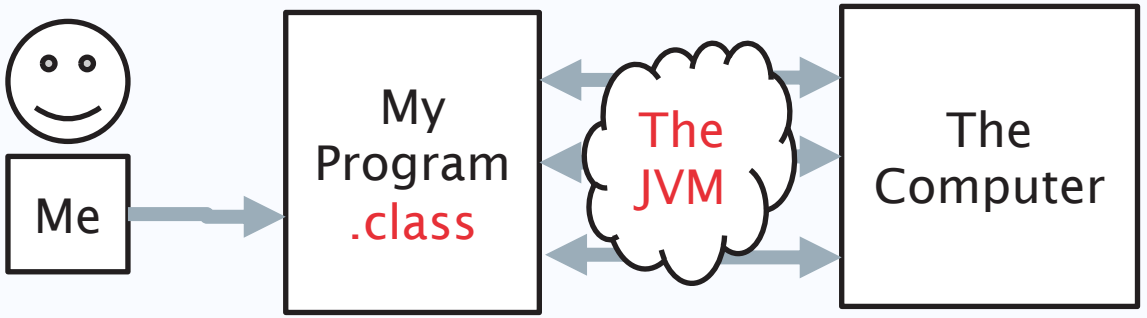
- You **write** code...



- You **compile** code



- You **run** the program



SEAtS: 812900

University of Southampton

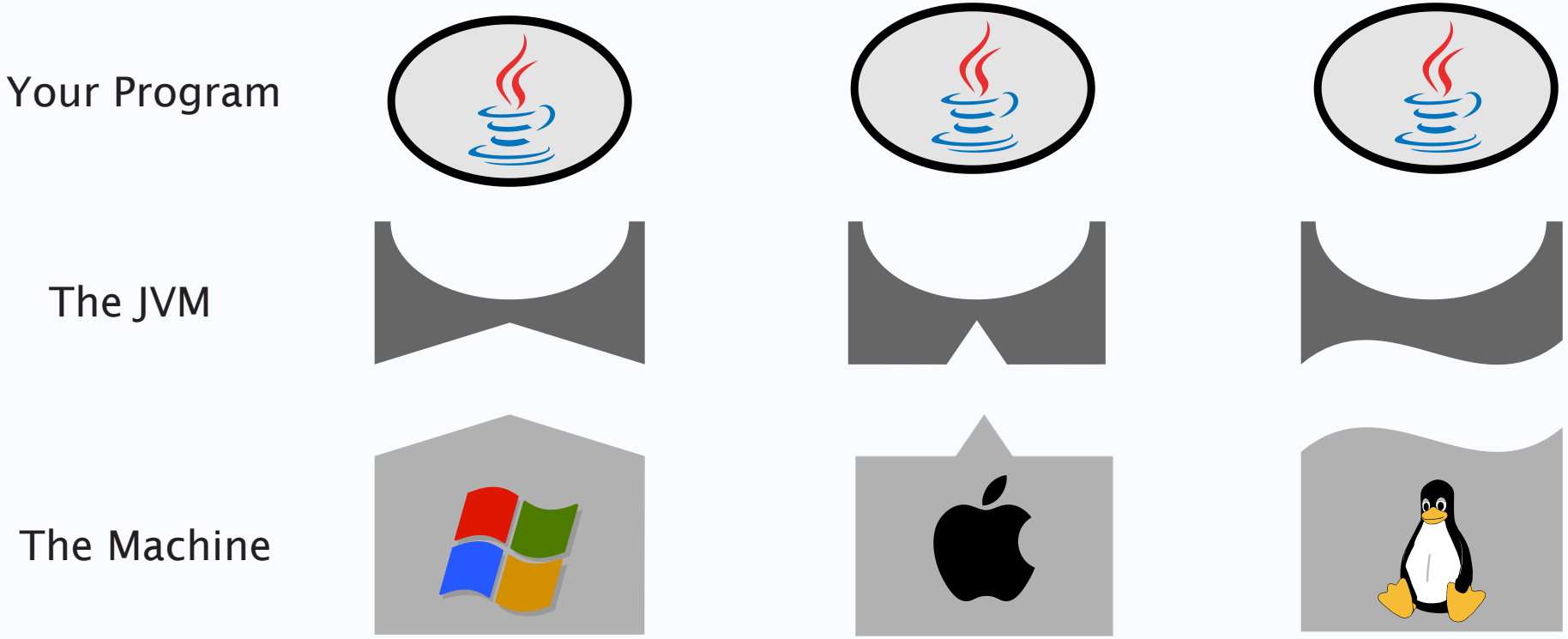
What is a JVM?

- Java Virtual Machine
- Each operating system / machine is different at a very low level:
  - Different way of putting things on the screen
  - Different way of making sound
  - Different way of taking input from keyboard
  - etc…
- So how can Java work on all these platforms?

SEAtS: 812900

University of Southampton

What is a JVM?



So the JVM is software that allows your Java program to run on any platform where there is a JVM

SEAtS: 812900

To Compile and Run a Class

- Save with same name as class
  - e.g., `Dog.java`
  - (Rule: Must do this)
- Command line to folder
- Compile (create the `.class` file)
  - `javac Dog.java`
- Run (execute the `.class` file)
  - `java Dog`

SEAtS: 812900

SEAtS: 812900

University of Southampton

YOUR QUESTIONS

Summary on how Java works

- The process of from code to program
- The purpose of a JVM
- What is `javac` command for?
- What is `java` command for?

SEAtS: 812900

University of Southampton

Part 2

Writing a Class in Java

Structure

Source file

Dog.java

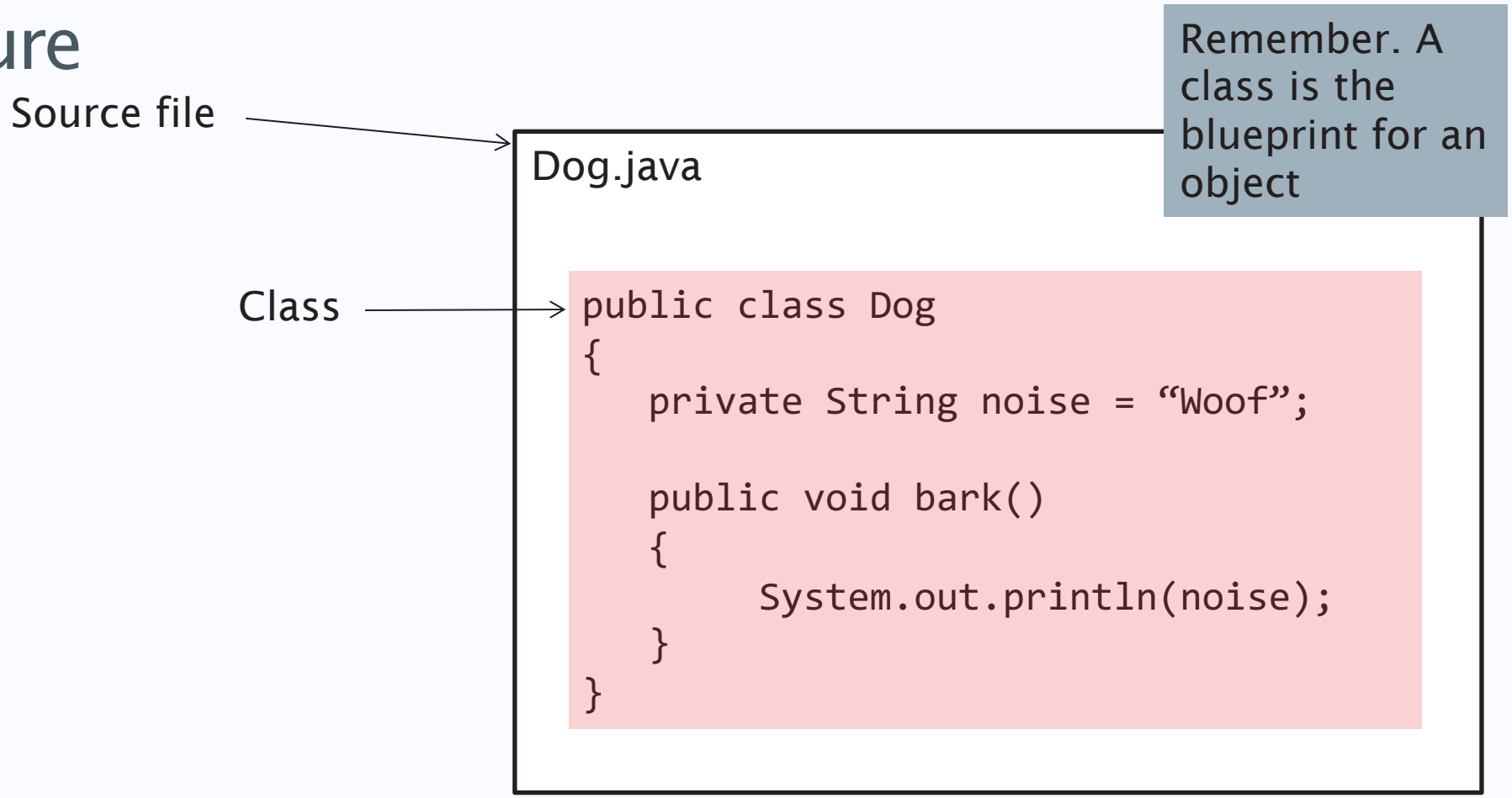
```
public class Dog
{
    public String noise = “Woof”;

    public void bark()
    {
        System.out.println(noise);
    }
}
```

A Source file is a basic text file. Normally these end .txt, but for java we always save them as .java

SEAtS: 812900

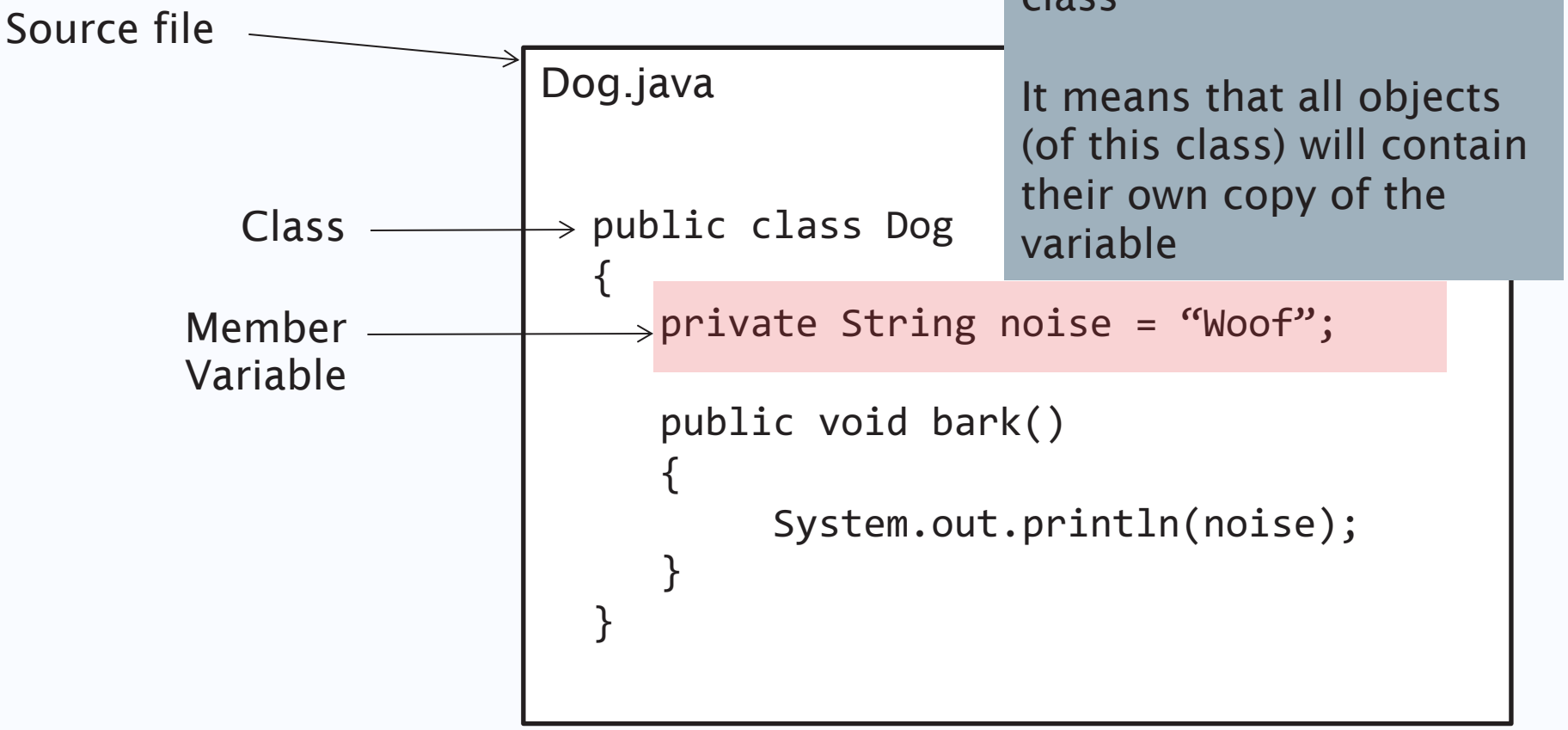
Structure



Remember. A class is the blueprint for an object

SEAtS: 812900

Structure

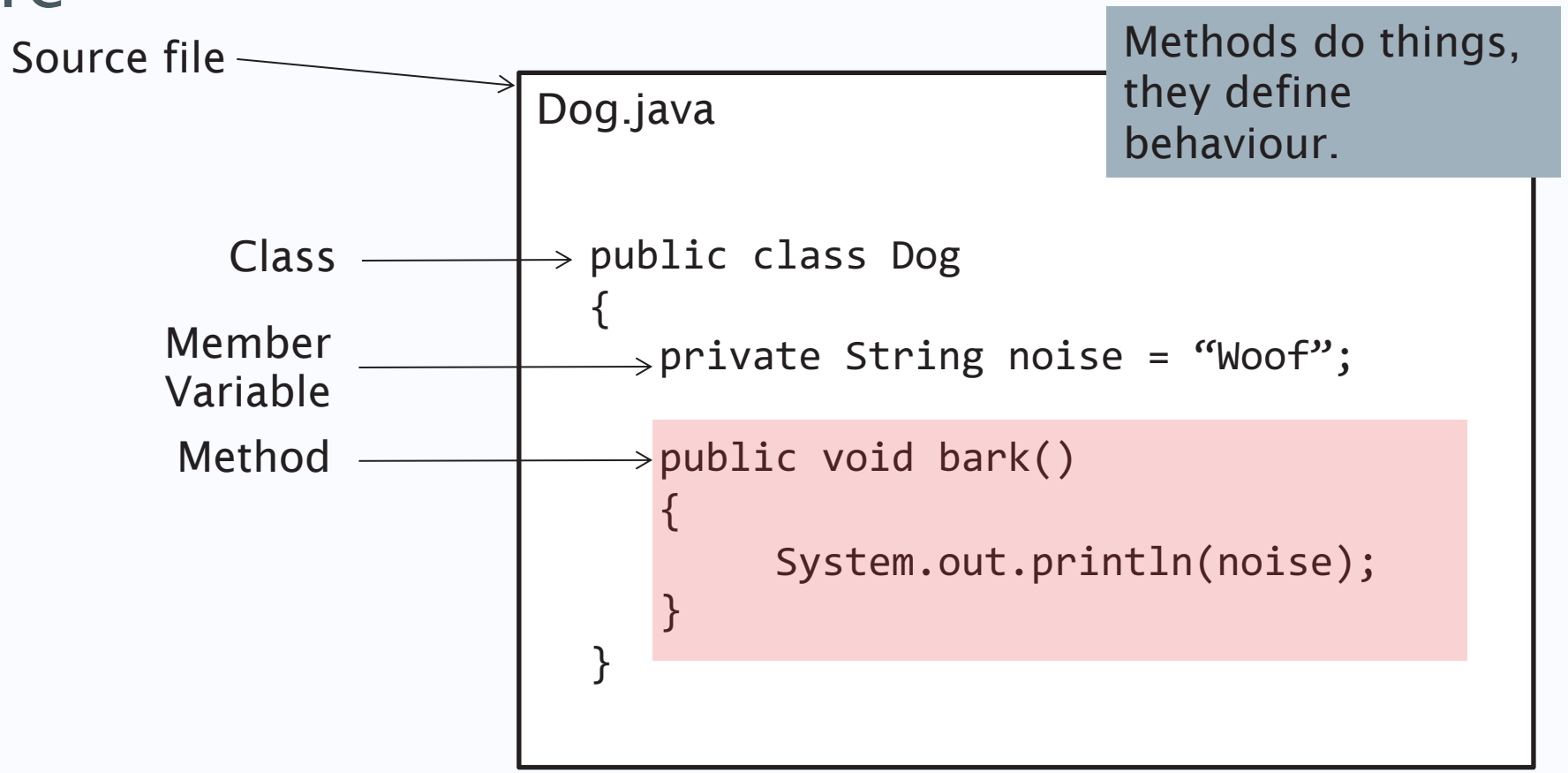


Member variables are how Java handles properties, they store data about the class

It means that all objects (of this class) will contain their own copy of the variable

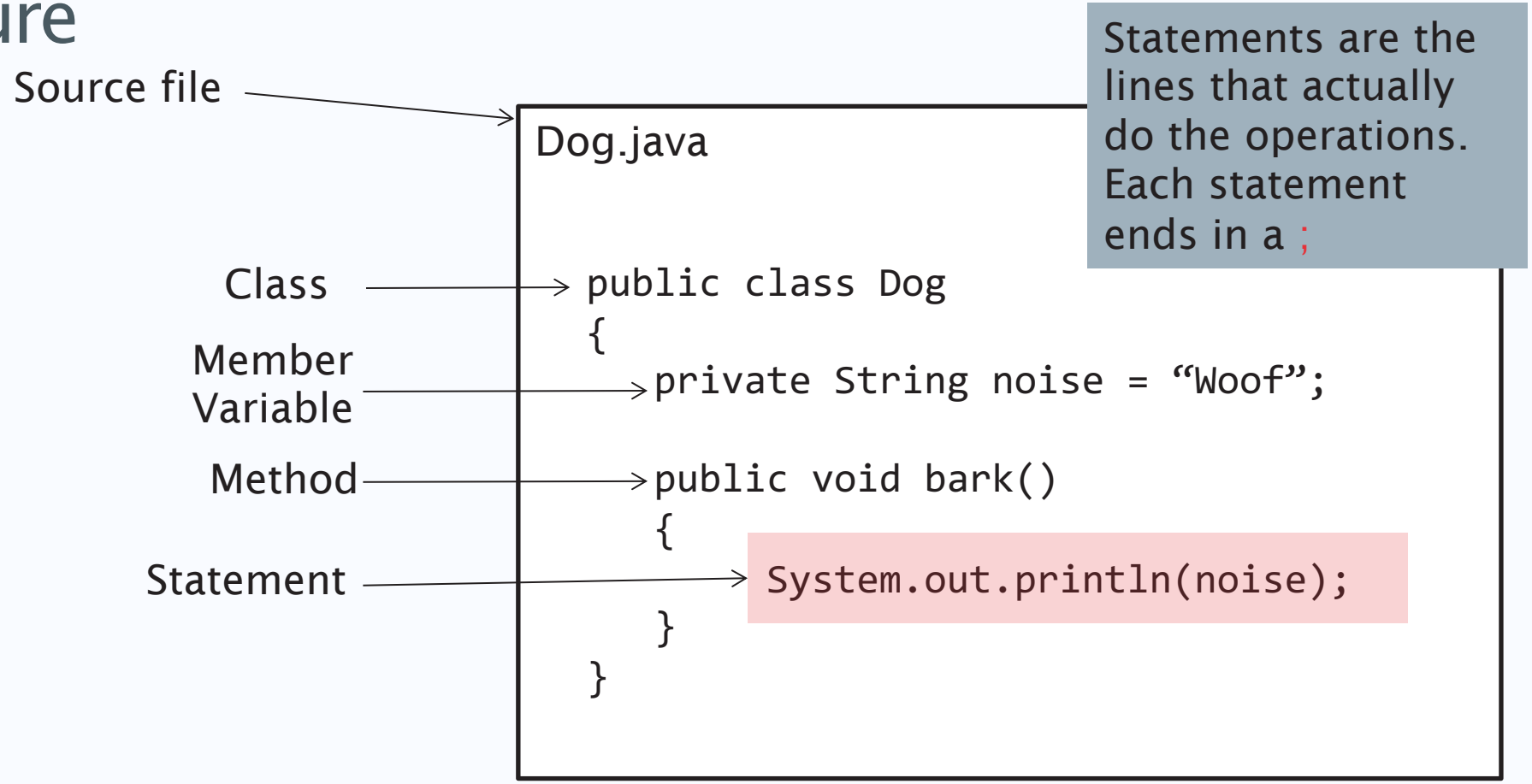
SEAtS: 812900

Structure



SEAtS: 812900

Structure



SEAtS: 812900

Part 3

A Detour. Magic Incantations

Magic Incantations

Programming is a complex study

You start learning something…
and find out you need to know
something else to understand it

and something else to understand that

and something else to understand that

SEAtS: 812900

Magic Incantations

Dog.java

```
public class Dog
{
   private String noise = “Woof”;

   public void doBark()
    {
          System.out.println(noise);
    }
}
```

SEAtS: 812900

Magic Incantations

You can start going nowhere fast if you try and understand **everything at once**!

We will explain all these things

Go with the flow

You may not understand the ‘incantations’
But don’t let it bother you, trust that they work

You will understand later

SEAtS: 812900

Magic Incantation #1

- Where does a program start?

- The program is made up of lots of classes, and those classes have methods, but which method is called **first?**

- In a special method called '`main`'

```
public static void main(String[] args){

}
```

SEAtS: 812900

The Java `main` method

- Creates the objects you need and tells them what to do, a bit like a conductor in an orchestra.

- It does not matter which class you put the main method in. You can put it in its own one if you like.

Dog.java

```
public class Dog
{
   private String noise = “Woof”;

   public void bark() {
        System.out.println(noise);
   }

   public static void main(String[] args){
         Dog d = new Dog();
         d.bark();
   }
}
```

SEAtS: 812900

Remember …

- Your class is just a template
- To make your program work you need objects
  - The instances of your class

```
public static void main(String[] args){
        Dog d = new Dog();
        d.bark();
}
```

SEAtS: 812900

SEAtS: 812900

University of Southampton

YOUR QUESTIONS

Summary of Writing a Java Class

- Java classes
- member variables
- methods
- statements
- Magic incantation: the `main()` method
- Creating an instance of a class

Part 4

A First Example

Banking Example

```
public class Account{

   int balance = 10;          //the bank balance
   boolean active = false;    //true if the account is active

   public void withdrawFiver(){
        balance = balance - 5;
   }
}
```

int and **boolean** are variable **types**, they tell Java what sort of thing is stored in the variables

Note that we can use **//** to show that everything that follows on that line is a **comment** and should be ignored by Java compiler

SEAtS: 812900

Making Decisions

- Often we want to make the result of a program conditional on something

- If the bank account is not active
    - Don't allow a withdrawal
  - Else, if the account doesn't have enough money
    - Don't allow a withdrawal
  - Else
    - Allow the withdrawal

- For this, we need a control structure called **if/else**

SEAtS: 812900

Withdrawal

```
public class Account{

    int balance;                    //the bank balance
    boolean active;                 // true if the account is active
    active = true;                  //set active to true

    // some code omitted

    public void withdrawFiver(){
        if (active != true)
            System.out.println("Your account isn't active");

    }
}
```

a != b is a **conditional operator**, it performs a logical test on a and b and returns true if they are not equal.

Other conditionals include <, > and ==

The one statement immediately following the if will only be run if the condition between the () brackets is true

SEAtS: 812900

Withdrawal

```
public class Account{

    int balance;                    //the bank balance
    boolean active;                 // true if the account is active
    active = true;                  //set active to true

    // some code omitted

    public void withdrawFiver(){
        if (active != true) {
            System.out.println("Your account isn't active");
            System.out.println("Withdrawal is not allowed");
        }
    }
}
```

We can use {} to group several statements together so that the if applies to all of them together

SEAtS: 812900

Withdrawal

```
public class Account{

    int balance;                    //the bank balance
    boolean active;                 // true if the account is active
    active = true;                  //set active to true

    // some code omitted

    public void withdrawFiver(){
        if (!active) {
            System.out.println("Your account isn't active");
            System.out.println("Withdrawal is not allowed");
        }
    }
}
```

We can simplify

! Is a **logical operator**, it reverses a logical value (so !a returns true if a is false)

Other logical operators are OR (a || b), AND (a && b)

SEAtS: 812900

Withdrawal

```
public class Account{

    int balance;                        //the bank balance
    boolean active;                     // true if the account is active
    active = true;                      //set active to true

    // some code omitted

    public void withdrawFiver(){
        if (!active) {
            System.out.println("Your account isn't active");
            System.out.println("Withdrawal is not allowed");
        } else {
            balance = balance - 5;
        }
    }
```

Optionally we can add an **else** clause.

SEAtS: 812900

Withdrawal

```
public class Account{

    int balance;                //the bank balance
    boolean active;             // true if the account is active
    active = true;              //set active to true

    // some code omitted

    public void withdrawFiver(){
          if (!active) {
                    System.out.println("Your account isn't active");
                    System.out.println("Withdrawal is not allowed");
        } else {
                  if (balance < 5) {
                                        System.out.println("Not enough money!");
                  } else {
                        balance = balance - 5;
                  }
        }
}
```

A whole if/else block is actually a single statement, so we can chain them together like this

SEAtS: 812900

Withdrawal

```
public class Account{

    int balance;                //the bank balance
    boolean active;             // true if the account is active
    active = true;              //set active to true

    // some code omitted

    public void withdrawFiver(){
          if (!active) {
                    System.out.println("Your account isn't active");
                    System.out.println("Withdrawal is not allowed");
        } else if (balance < 5) {
                    System.out.println("Not enough money!");
        } else {
                    balance = balance - 5;
        }
    }
}
```

We can simplify

Because an `if/else` block is actually considered a single statement we can get rid of some of these brackets and tidy up

SEAtS: 812900

SEAtS: 812900

University of Southampton

YOUR QUESTIONS

Summary of Conditional Statements

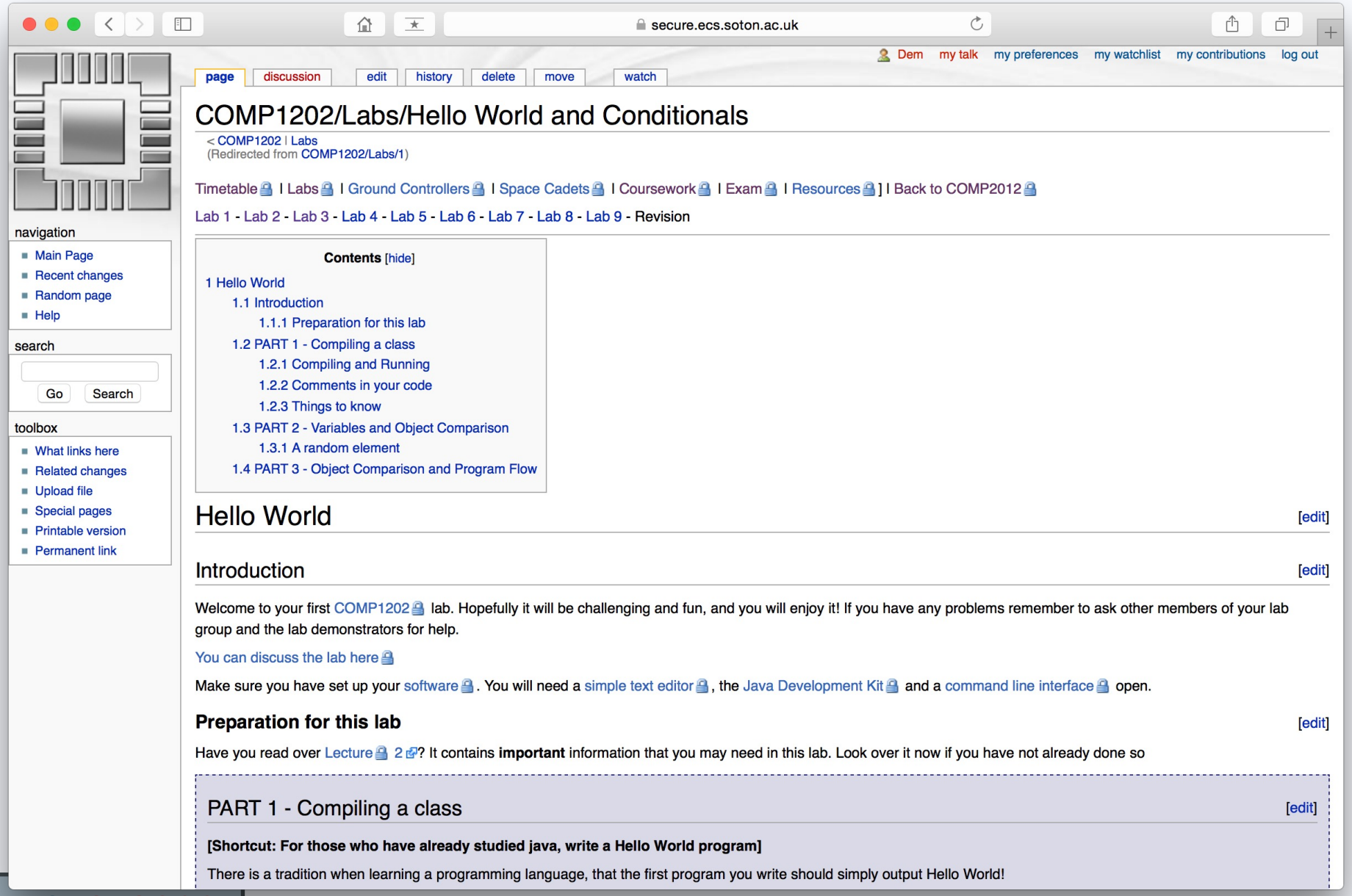
- `if/else` condition statement
- When `{}` is required, when `{}` is not required
- Nested `if/else` statement

SEAtS: 812900

University of Southampton

Part 5

The Lab


secure.ecs.soton.ac.uk
Dem my talk my preferences my watchlist my contributions log out
page discussion edit history delete move watch
COMP1202/Labs/Hello World and Conditionals
< COMP1202 | Labs
(Redirected from COMP1202/Labs/1)
Timetable | Labs | Ground Controllers | Space Cadets | Coursework | Exam | Resources] | Back to COMP2012
Lab 1 - Lab 2 - Lab 3 - Lab 4 - Lab 5 - Lab 6 - Lab 7 - Lab 8 - Lab 9 - Revision
navigation
Main Page
Recent changes
Random page
Help
search
Go Search
toolbox
What links here
Related changes
Upload file
Special pages
Printable version
Permanent link
Contents [hide]
1 Hello World
1.1 Introduction
1.1.1 Preparation for this lab
1.2 PART 1 - Compiling a class
1.2.1 Compiling and Running
1.2.2 Comments in your code
1.2.3 Things to know
1.3 PART 2 - Variables and Object Comparison
1.3.1 A random element
1.4 PART 3 - Object Comparison and Program Flow
Hello World [edit]
Introduction [edit]
Welcome to your first COMP1202 lab. Hopefully it will be challenging and fun, and you will enjoy it! If you have any problems remember to ask other members of your lab group and the lab demonstrators for help.
You can discuss the lab here
Make sure you have set up your software. You will need a simple text editor, the Java Development Kit and a command line interface open.
Preparation for this lab [edit]
Have you read over Lecture 2? It contains important information that you may need in this lab. Look over it now if you have not already done so
PART 1 - Compiling a class [edit]
[Shortcut: For those who have already studied java, write a Hello World program]
There is a tradition when learning a programming language, that the first program you write should simply output Hello World!

Marking – Functional Correctness

- **5 (Excellent)** - Your code compiles and passes all
- **4 (Very Good)** - Your code compiles and passes al cases
- **3 (Good)** - Your code compiles and passes the ma cases
- **2 (Acceptable)** - Your code compiles and passes s cases
- **1 (Poor)** - Your code does not compile or very few passed
- **0 (Inadequate)** - No code submitted, not a serious

Automatically Marked

Using Test Harnesses
That we **give to you**
Lab 1 will show you the process

(remember no formal marking until Week 4)

SEAtS: 812900

Formative – Readability/Coding Style

- **Excellent** - Code is very easy to read and understand
  - i.e. Perfect indentation, placement of brackets and cam
    design decisions parameters class and method functior
- **Very Good** - Minor flaws in style, with an appropriate
  lacking some details
- **Good** - Code is mostly easy to understand with a goo
  some improvements possible
- **Acceptable** - Sound programming style but readabili
- **Poor** - Coding style and readability need significant i
- **Inadequate** - Expected coding style is not used, code
  - i.e. Incorrect indentations, the inconsistent placing of b
    appropriately and no comments

SEAtS: 812900

Given By Demonstrators

Ask for their feedback in the lab sessions.

We will also discuss coding style in Week 3.

(remember no formal marking until Week 4)

University of Southampton

The Toolbox

- Getting input and output can be a pain in Java.
- So can some other things
- ECS has provided a ‘toolbox’ for you to use.
- This toolbox is a class of useful methods
- Like readStringfromCmd()
- Has some other useful functions, we’ll introduce them as we need them
- But you don’t have to use them if you know what to do already.

SEAtS: 812900

Getting Input

```
Toolbox toolbox = new Toolbox();
String word;  //or whatever name you choose
word = toolbox.readStringFromCmd();
```

- Entering this in your code means when the program gets to that line
  - The command prompt will ask you for a string
  - You enter the string and press enter
  - The program keeps going, with the string you typed stored in the variable

SEAtS: 812900

SEAtS: 812900

University of Southampton

Good luck for the Lab!

YOUR QUESTIONS

Summary

- How Java Works: The JVM
- Writing a Class in Java (Class, Member Variables, Methods, Statements)
- Magic incantations (The `main()` method)
- A First Example
  - Defining an `Account` class
  - `If/else` and `Boolean` operations
- Introducing the `Toolbox`