

Programming 1 / Object Oriented Programming

Introduction to Java:
Lecture 1: The basics of the Java
language

CLICKER CHANNEL: 82

Part 1: The Logistics of Java

A question...

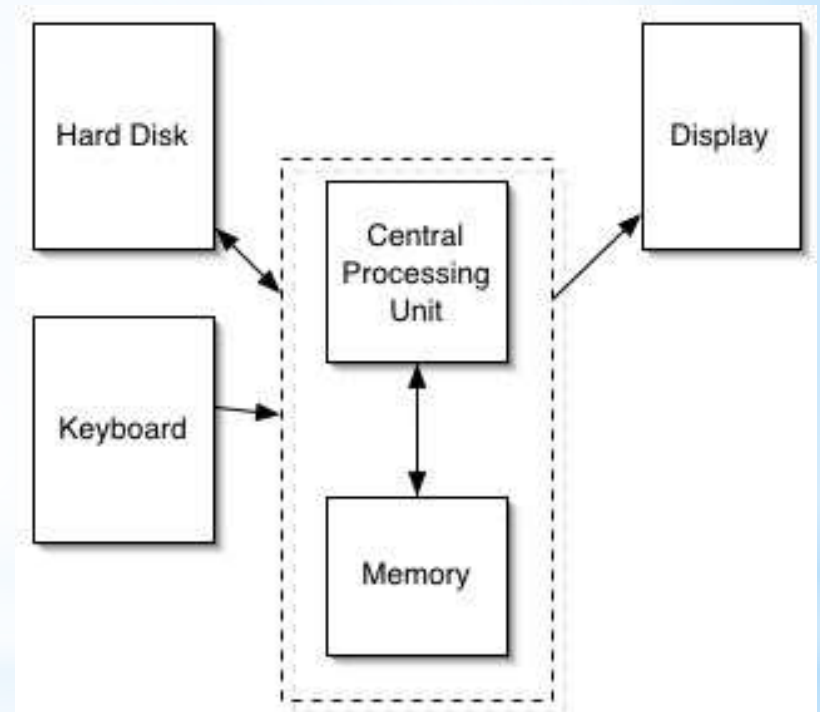
What is a computer?

What is a computer?

- ⦿ At its simplest, a computer is a collection of wires, transistors and electrical components that are connected in a specific way
- ⦿ When you send certain sequences of voltages down the wires, you will get other sequences of voltages back out again
- ⦿ We, as humans, choose to interpret the presence or absence of a voltage as a 1 or a 0
- ⦿ We, as humans, choose to interpret certain sequences of voltages as "adding", others as "subtracting" and so on

What is a computer?

- ◉ The part that does the adding and comparing is the *Central Processing Unit (CPU)*
- ◉ The CPU stores data in *Random Access Memory (RAM)*, often simply referred to as just *memory*
- ◉ The *hard disk* provides permanent storage - the content of memory is lost when the computer is turned off
- ◉ The hard disk also provides substantially more storage than memory - but is millions of times slower
- ◉ The display is the screen (or LCD display, or whatever)

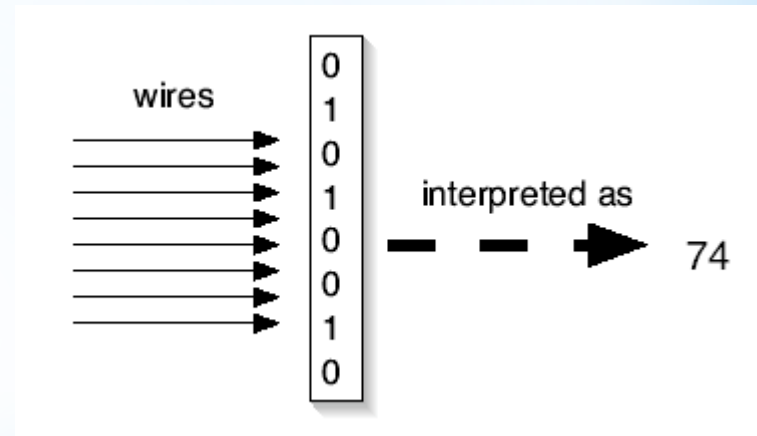


What is a computer?

- ⦿ A computer can also be said to be a layer cake of encodings:
 - ⦿ Consider eight wires where the second and eighth wire have a voltage, and all others have no voltage
 - ⦿ We would represent this as the binary number 01000001
 - ⦿ This is 65 in decimal...
 - ⦿ ...but it might also be the letter "A" if we were storing text in the ASCII encoding system!

Key Concept: Encodings

- We can *interpret* the 0's and 1's in computer memory any way we want.
 - We can treat them as numbers.
 - We can *encode* information in those numbers
 - Each 0 or 1 is a bit
 - Eight of these is a byte
 - Which we can, in turn, interpret as a decimal number
- (Remember that even the notion that the computer understands numbers at all is an interpretation!
 - We always talk about 0's and 1's but the reality is that computer data is voltage on wires. If there is a voltage, we say there's a 1 - if not, a 0.)



Key concept: number systems

- ⊙ Humans work in decimal notation
 - ⊙ We count from 1 to 9, then the next number is 10
 - ⊙ We might also refer to decimal as "base 10" - i.e. we get to ten before we increment the leading digit
- ⊙ There are other number systems apart from decimal
 - ⊙ Base 16 or hexadecimal (hex)
 - ⊙ We count 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F before incrementing and getting to "10".
 - ⊙ In hex, "10" does not mean ten - it is actually sixteen!
 - ⊙ Base 2 or binary
 - ⊙ We only use the digits 0 and 1
 - ⊙ We count 0, 1, 10, 11, 100, 101, 110, 111, 1000...
 - ⊙ which is zero, one, two, three, four, five, six, seven, eight...

Converting from binary

- Consider the binary number 10110101

Place values
(multiply this number by the 1 or 0 in its place)

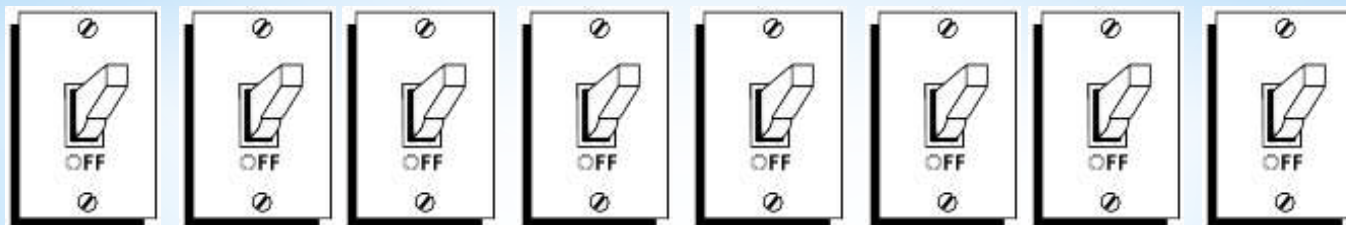
128	64	32	16	8	4	2	1
x	x	x	x	x	x	x	x
1	0	1	1	0	1	0	1
=	=	=	=	=	=	=	=
128	+ 0	+ 32	+ 16	+ 0	+ 4	+ 0	+ 1

(add all these together to get the decimal number)

$$= 181$$

Tripping the right switches

- ⦿ The transistors inside a computer can be considered a collection of electrical switches that make electricity flow this way or that way
 - ⦿ The presence of electricity (or not) represents data - our 1's and 0's
- ⦿ A program is simply a combination of switch settings that cause the computer to produce a particular combination of voltages
- ⦿ As we've seen, we *interpret* those voltages to be numbers, text, or even multimedia such as images or sound.
- ⦿ We interpret a particular combination of switch settings to be instructions to do addition, subtraction, loading, storing, etc



Assembler

- ⦿ Machine language if represented in a human readable form looks like a load of numbers
- ⦿ Assembler is a set of words, symbols and letters that correspond to the machine language
 - ⦿ It's a one-to-one relationship
 - ⦿ Usually, a word of assembler equals one machine language instruction - often just a single byte

Assembler -> Machine

LOAD 10,R0	; Load special variable R0 with 10
LOAD 12,R1	; Load special variable R1 with 12
SUM R0,R1	; Add special variables R0 and R1
STOR R1,#45	; Store the result into memory location #45

Might appear in memory as just 12 bytes:

01 00 10

01 01 12

02 00 01

03 01 45

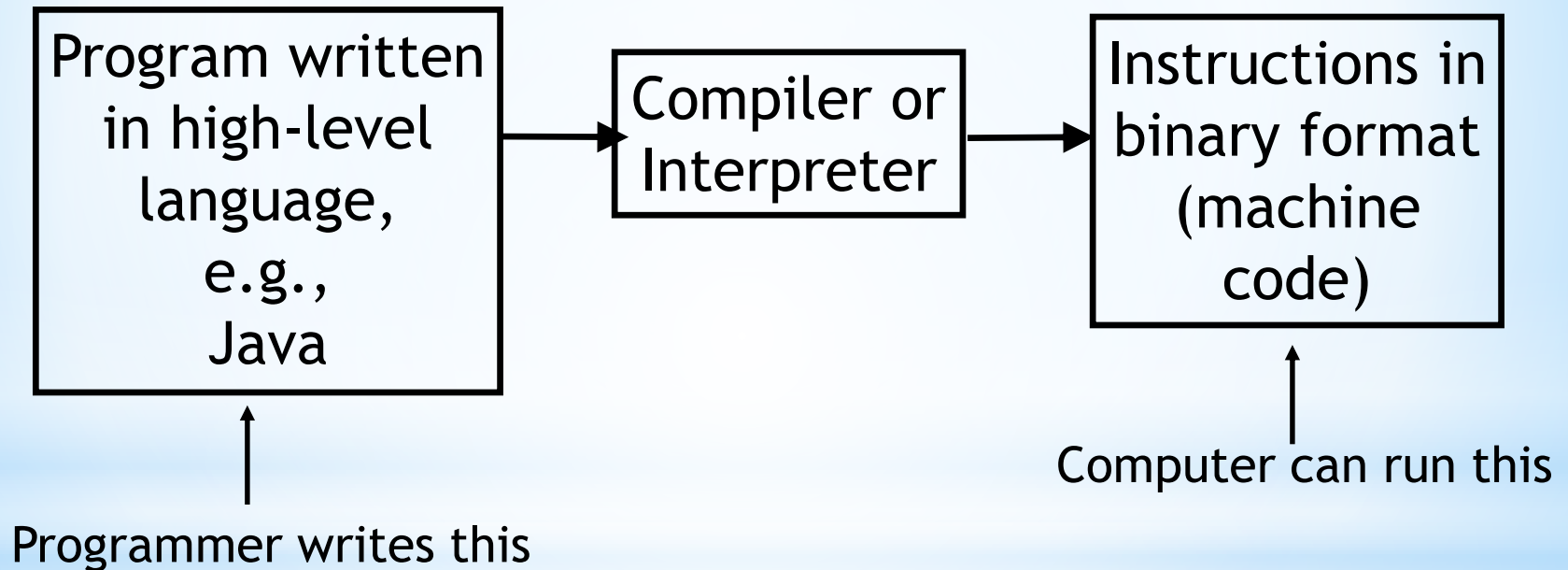
Programming Languages

- ⊙ Writing all of our programs in binary would be very inconvenient!
- ⊙ Even writing our programs in assembler would be a drag
 - ⊙ The instructions (or *operation codes*) are often inscrutable
 - ⊙ Each CPU has a different instruction set
- ⊙ Instead, one option is to use *high level* programming languages such as Java
- ⊙ High level languages are more like English and are (in theory) easier for human beings to understand
- ⊙ However, programs in high level languages must be converted into machine code before they can be run

Levels of programming languages

- ⦿ Machine code is very difficult to understand - even if translated into assembler
- ⦿ *High level* programming languages, such as Java, Javascript, C, C++, C# etc are easier to understand, and use commands that are more like English
- ⦿ Because computers do not understand high level languages, they must be translated into machine code before they can run
 - ⦿ ...so programming languages, ultimately, are just another suite of encoding layers
- ⦿ High level languages might be *interpreted* or *compiled*
 - ⦿ *Compiled* means that the entire program is converted into machine code before it is run
 - ⦿ *Interpreted* means that the program is translated into machine code while it is being run

Converting a computer program into machine code



Interpreted v compiled

- ⦿ Compiled languages perform *much* faster once a program is running, but you have to compile the code first
- ⦿ Interpreted languages *start* running more quickly because there's no initial need to compile the code, but are much slower once they're running
- ⦿ Interpreted languages can be easier to write programs for, not because the languages themselves are necessarily "easier" to understand, but because you can make changes and corrections to the code and try them out much more quickly
- ⦿ The compiler in a compiler language "sees" the whole program at once, so might give information about all the errors in a program before the program runs

The Java Programming Language

- ⦿ Java is a general purpose language
- ⦿ Java is **platform independent**
- ⦿ Java is an **object-oriented** language
- ⦿ There are different 'editions' of Java - we are using Java SE

Which are we: interpreter or compiler?

- ⦿ Recap
 - ⦿ High level languages might be *interpreted* or *compiled*
 - ⦿ *Compiled* means that the entire program is converted into machine code before it is run
 - ⦿ *Interpreted* means that the program is translated into machine code while it is being run
- ⦿ Java is a bit different: Java programs are compiled to Java *bytecode*
- ⦿ Bytecode is not machine code for a particular CPU type, but instead runs on the *Java Virtual Machine* or JVM
- ⦿ The JVM is available for all major desktop computing platforms, embedded devices, mobile devices, etc
- ⦿ The JVM interprets the bytecode into the machine code of the target platform
 - ⦿ thus Java is both compiled AND interpreted!

The Java Virtual Machine

- ⦿ Think of bytecode as being like the machine code for a fictional* CPU
- ⦿ This fictional CPU has its own instruction set - just as an x86 processor has an instruction set, or an ARM processor has an instruction set
- ⦿ The JVM is basically a program that simulates this fictional CPU - so your bytecode runs within the simulation

A fun tangent

- ⦿ Anyone ever play vintage computer games on emulator programs?
- ⦿ Anyone ever hear of the Sinclair Spectrum?
- ⦿ Vital statistics:
 - ⦿ RAM:
 - ⦿ 48K
 - ⦿ Processor:
 - ⦿ 8 bit, Z80, 3.5Mhz

The average modern handheld calculator probably has better specs than the poor old Speccy...! 😊



A fun tangent

- ⦿ Point your browser at
 - ⦿ <http://torinak.com/qaop>

A fun tangent

- ⦿ The QAOP website provides an authentic simulation of EVERY aspect of a Sinclair Spectrum
- ⦿ It simulates
 - ⦿ the CPU
 - ⦿ the graphics hardware
 - ⦿ the keyboard
 - ⦿ the display hardware
 - ⦿ everything that was part of a Sinclair Spectrum
- ⦿ Spectrum programs can't (ordinarily) run on a PC - the CPU is different, the graphic hardware is different, the way a PC gets signals from input devices (e.g. keyboard, mouse, etc) is different
- ⦿ ...however, Spectrum programs CAN run on our *simulated* Spectrum
 - ⦿ ...and our simulated Spectrum is just a Javascript program, that runs in our browser, which in turn runs on our PC!
 - ⦿ (that's at least three levels of interpretation going on)

The Java Virtual Machine

- ⦿ Think of bytecode as being like the machine code for a fictional* CPU
- ⦿ This fictional CPU has its own instruction set - just as an x86 processor has an instruction set, or an ARM processor has an instruction set
- ⦿ The JVM is basically a program that simulates this fictional CPU - so your bytecode runs within the simulation
 - ⦿ ...just as our Spectrum games ran within our simulation of the Spectrum computer!

* actually, there ARE some hardware implementations of this "fictional" processor - i.e. CPUs whose instruction set is the same set of instructions as Java bytecode.

How to "get" Java

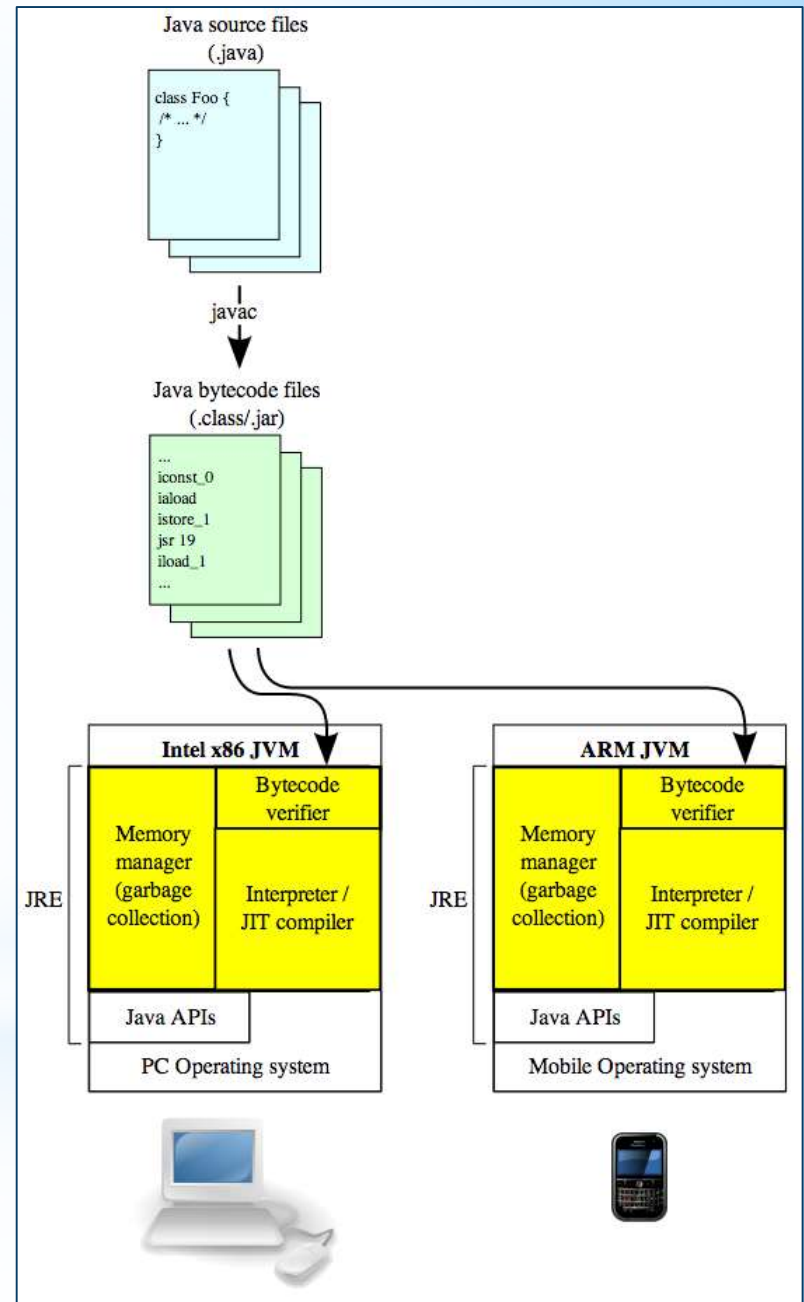
- ⦿ You can download either the Java Runtime Environment (JRE) or the Java Development Kit (JDK) from Oracle's website
- ⦿ Just Google **Java JRE** or **Java JDK**
- ⦿ The latest version at the time of writing is version 7
 - ⦿ (just to make things confusing they went version 1.0, 1.1, 1.2, 1.3, 1.4, 5, 6 and 7 - i.e. they dropped the "1." bit after 1.4...!)
- ⦿ The JRE gives you the JVM along with everything else you need to *run* Java applications
 - ⦿ it does NOT include **javac**, the Java compiler
- ⦿ The JDK includes the JRE along with everything additional you need to *develop* Java applications...
 - ⦿ ...such as (among other things) the compiler

Creating and running a Java program

- ⦿ So, ordinarily, you would need
 - ⦿ An editor, to author your **source code**
 - ⦿ Examples might be Notepad, TextPad, TextWrangler on a Mac - anything that can create a text file
 - ⦿ The Java **compiler** which produces **byte code**
 - ⦿ the *javac* program, included with the Java development kit (JDK)
 - ⦿ The Java application launcher
 - ⦿ the *java* program, included with both the Java runtime environment (JRE) and also with the JDK

The Java compilation process

- ⦿ A Java program begins as *source code*.
- ⦿ This is the human-readable code that the programmer composes
- ⦿ The java compiler, **javac**, produces an object file which contains Java *bytecode*.
 - ⦿ E.g. `javac Hello.java`
 - ⦿ The result will be a file called `Hello.class` - our bytecode
- ⦿ The bytecode can be run on the JVM on any platform that has a JVM available
- ⦿ We might invoke the JVM to run our bytecode with
 - ⦿ `java Hello`



The simplest Java program: Hello World!

File: HelloWorld.java

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

javac HelloWorld.java

-> results in HelloWorld.class

Java HelloWorld

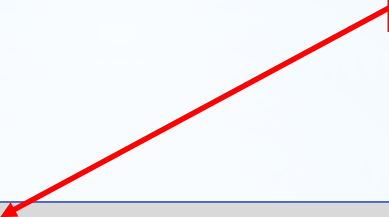
-> runs HelloWorld.class

Part 2: The basics of the Java language

The simplest Java program: Hello World!

File: HelloWorld.java

class name



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

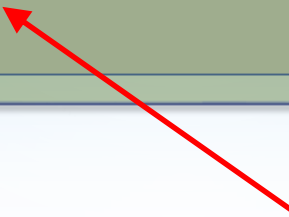
The simplest Java program: Hello World!

File: HelloWorld.java

class name

A red arrow points from the 'class name' box to the 'public class HelloWorld' line in the code block.

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

A red arrow points from the 'code block (class declaration)' box to the entire code block.

code block (class
declaration)

The simplest Java program: Hello World!

File: HelloWorld.java

semi-colon denotes
end of statement

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

The simplest Java program: Hello World!

File: HelloWorld.java

class name

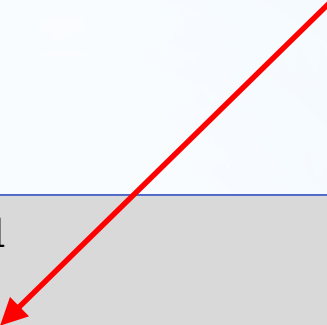
```
public class HelloWorld
{
    public static void main (String[] args)
    {
        System.out.println("Hello, World!");
    }
}
```

inner code block
(method declaration)

The simplest Java program: Hello World!

File: HelloWorld.java

main method



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

The simplest Java program: Hello World!

File: HelloWorld.java


a method that
displays text on the
screen

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

string (text) to display

The simplest Java program:

NoobLab takes this as read in the early exercises



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



and this

Adding to our Hello World program

```
public class SomeJavaCode
{
    public static void main (String[] args)
    {
        System.out.println("Hello, World!");
        System.out.println("Good here isn't it?");
    }
}
```

Syntax errors

⦿ For example:

```
System.out.println("Hello, World!")
```

Error!

Punctuation is wrong -
Semi-colon is missing

Error!
spelling is not right

```
Zystem.out.println("Hello, World!");
```

Error!
"spelling" is not right - should be an uppercase 'S'

```
system.out.println("Hello, World!");
```

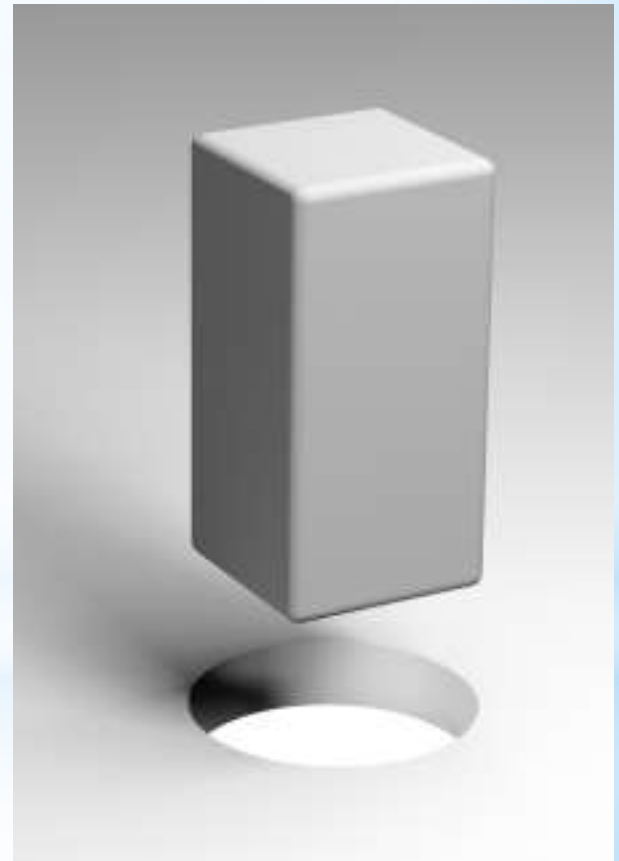
Variables in Java and data types

What is meant by a data type?

- ⦿ The clue is in the name...
 - ⦿ ... it means which *type* of information a particular piece of data is
- ⦿ Consider the following pieces of information:
 - ⦿ "Paul Neve"
 - ⦿ 57
 - ⦿ 57.347531
 - ⦿ 11/7/80
- ⦿ They are all different data types

Java is a strongly typed language

- ⦿ This means that types of variables are strongly enforced
 - ⦿ You can't put a number in a string variable
 - ⦿ You can't put a string in a numeric variable



Java is a strongly typed language

- ⦿ So if variables are like boxes, then the boxes in Java are like this
- ⦿ You can only put things into them that "fit in the box's hole"



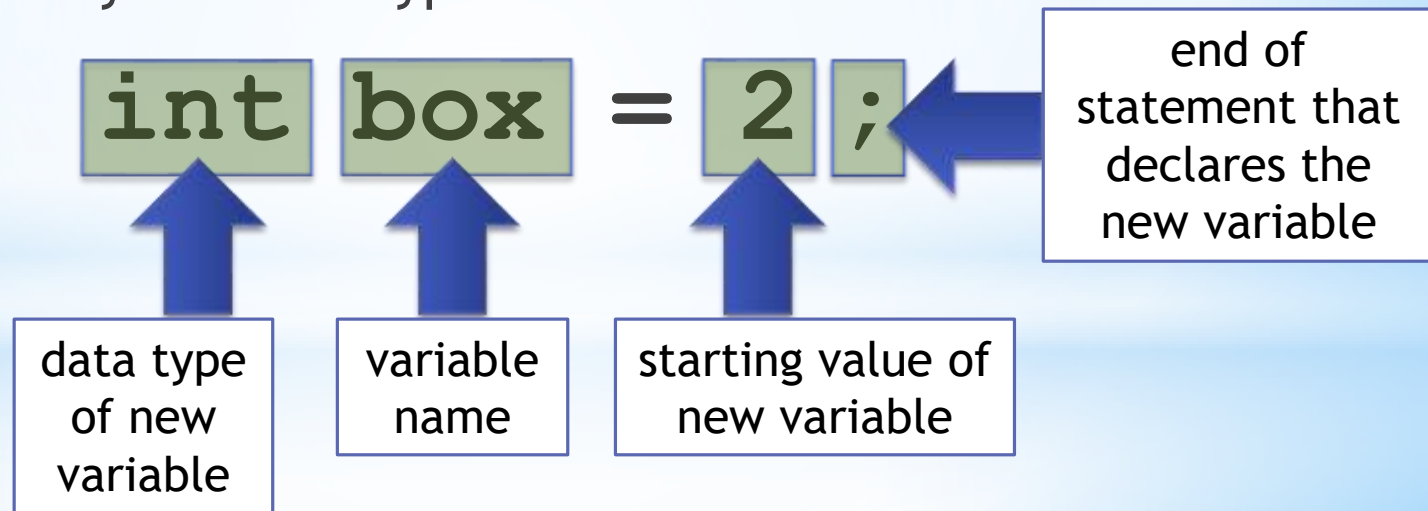
Declaring a variable in Java

- ⦿ Unlike in pseudocode...
 - ⦿ `set box = 2;`
- ⦿ ..or in Javascript
 - ⦿ `var box = 2;`
- ⦿ ...or in PHP
 - ⦿ `$box = 2;`
- ⦿ ...Java requires that when you declare a new variable, you have to specify it's data type

`int box = 2;`

Declaring a variable in Java

- ⦿ Unlike in pseudocode...
 - ⦿ `set box = 2;`
- ⦿ ..or in Javascript
 - ⦿ `var box = 2;`
- ⦿ ...Java requires that when you declare a new variable, you have to specify it's data type



Declaring a variable in Java

- ⦿ We can split up the declaration of a new variable and assigning a value into two separate lines
- ⦿ The first time we declare a variable, we must specify its data type

```
int box;
```

```
System.out.println("Box has been declared");
```

```
box = 17;
```

```
System.out.println("Box now contains "+box);
```

- ⦿ When we subsequently use it - either to assign a value, or to access what's stored inside it, we do NOT specify the data type, we just refer to it by name alone

The integer data type

- ⦿ Integers are whole numbers, e.g. 1, 7, 27, -7 4394, -500, 475893 - anything between -2147483648 and 2147483647 inclusive

- ⦿ In Java, we use the keyword `int` to indicate an integer:

```
int numberOfStudents;
```

```
numberOfStudents = 65;
```

or

```
int numberOfStudents = 65;
```

The string "data type"

- ⦿ Strictly speaking, strings aren't a data type in Java - but for now, you can treat them like they are
- ⦿ So to declare a string variable in Java:

```
String box1 = "Paul";
```



The string "data type"

- ⦿ Strictly speaking, strings aren't a data type in Java - but for now, you can treat them like they are
- ⦿ So to declare a string variable in Java:

```
String box1 = "Paul" ;
```

Note capital S!
Remember C-like
languages are CASE
SENSITIVE!

Remember that if
you want to put an
explicit value into
your string, you
must put it in
quotes!



Which of the following would work?

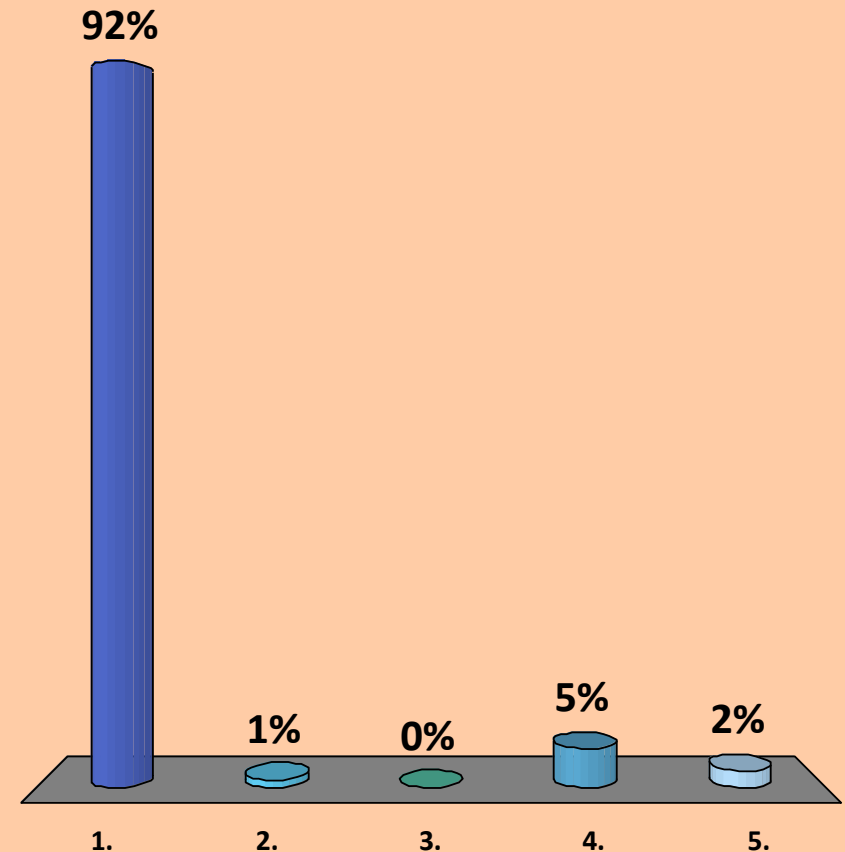
1. `String myBigBox;
myBigBox = "Paul";`

2. `String myBigBox;
mybigbox = "Paul";`

3. `String myBigBox;
MYBIGBOX = "Paul";`

4. `String myBigBox;
MyBigBox = "Paul";`

5. All of them



Beware case-sensitivity!

```
String myBigBox;  
myBigBox = "Paul";
```



```
String myBigBox;  
mybigbox = "Paul";
```



```
String myBigBox;  
MYBIGBOX = "Paul";
```



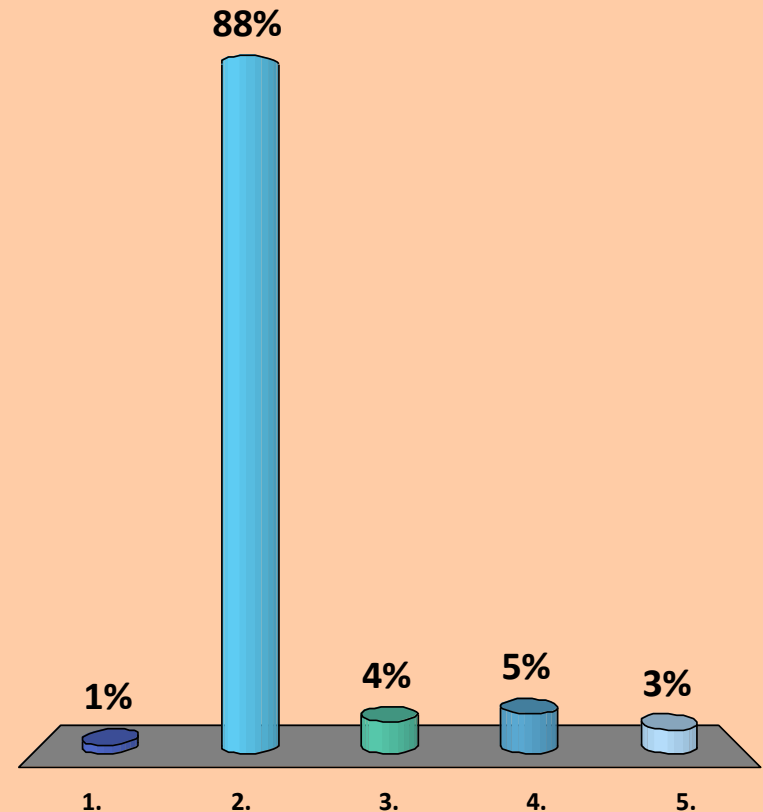
```
String myBigBox;  
MyBigBox = "Paul";
```



What's wrong here (assuming you wanted the result to be to print the text *Hello, World*)?

```
String greeting;  
greeting = "Hello, World";  
System.out.println("greeting");
```

1. The brackets should be removed from the third line
2. The quotes should be removed from the third line
3. The keyword **String** should be added at the start of the second line
4. There is some other error in the code
5. There is no error in the code, it would run fine, and it would print *Hello, World*



The double data type

- ⦿ Stores numbers with a decimal point or *floating point numbers*, e.g. 10.5, 1.0, 100.999, -0.1
- ⦿ Example usage:
`double totalMarks = 56.7;`

Things to note about doubles

- ⦿ doubles are frowned upon for use in "real world" applications that deal with money
 - ⦿ you'll see an example WHY in the practical...
- ⦿ An integer value will be automatically converted if you try to assign it into a double variable, e.g.

```
double width = 10;           // results in 10.0
```

or even

```
int roughWidth = 20;
```

```
double width = roughWidth; // results in 20.0
```

Things to note about doubles

- ⦿ However, a double will not "fit" into an integer, e.g.

```
double width = 10.7;  
int roughWidth = width;
```

will not work ☹

WARNING!

- ⦿ I have taught you about the **double** data type
- ⦿ Java has lots of other data types that can represent floating point numbers
- ⦿ The NoobLab exercises assume you've used what we teach on this module!
- ⦿ If you start using **floats** or other data types we've not mentioned in workshop exercises
 - ⦿ You won't win the medals
 - ⦿ I will laugh at you for
 - a) not paying attention during lectures
 - b) not coming to lectures
 - c) mindlessly Googling for answers rather than studying (or at least searching) the teaching material provided
 - d) all of the above

Java Arithmetic Operators

+ add
- Subtract
* multiply
/ divide
% remainder

```
int counter = 1000;  
int number1 = counter + 10;  
int number2 = number1 - 500;  
int number3 = number2 * 2;  
int number4 = number3 / 9;  
int number5 = number4 % 5;
```

operator

**assignment
operator**

operands

expression

Shorthand

```
int myNumber = 10;  
myNumber = myNumber + 5;
```

```
int myNumber = 10;  
myNumber += 5;
```

```
int myNumber = 10;  
myNumber = myNumber + 1;
```

```
int myNumber = 10;  
myNumber++;
```

⦿ Also for operators other than plus, e.g.

```
int myNumber = 10;
```

```
myNumber = myNumber--;           // myNumber becomes 9
```

```
myNumber *= 2;                   // myNumber becomes 18
```

Mixing your data types

- ⦿ You can mix data types when applying an operator to several operands
- ⦿ A rule of thumb is that the result will have the "simplest" data type required to hold the result of the operation:

```
String text = "Some text";  
String num = 27;  
String merged = text+num;  
// result is a string  
// i.e. "Some text27"
```

```
int num = 27;  
double merged = num * 0.5  
// result is a double  
// i.e. 13.5
```

```
int num = 27;  
double merged = num * 2  
// result is an int, i.e 54  
// but an int "fits" into a  
// double, so merged ends up  
// as 54.0
```

Variables **MUST** be declared before you can use them

```
String gender;  
gender = "male";  
System.out.println(gender);
```



```
String gender = "male";  
System.out.println(gender);
```



```
gender = "male";  
System.out.println(gender);
```



The moral of the story:

Whenever you see some Java code that has a **data type** followed by a variable name (such as the code that is underlined in our examples), the code is **declaring** a new variable.

You can only declare a variable **ONCE**

```
String gender;  
gender = "male";  
System.out.println(gender);  
gender = "female";  
System.out.println(gender);
```



```
String gender = "male";  
System.out.println(gender);  
gender = "female";  
System.out.println(gender);
```



```
String gender = "male";  
System.out.println(gender);  
String gender = "female";  
System.out.println(gender);
```



The moral of the story:

We've already seen that a **data type** followed by a variable name **declares** a new variable.

You cannot declare another new variable with the same name - and you cannot re-declare a variable

You are the judge...! Which of these would WORK?

1

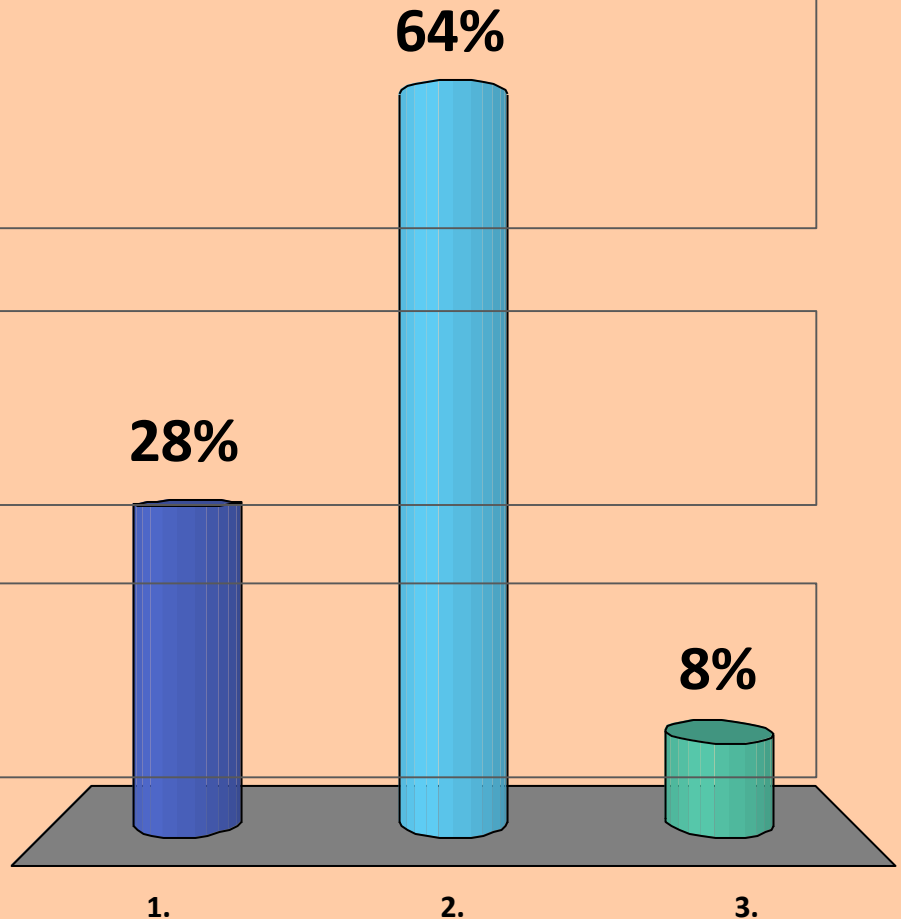
```
String name;  
name = "Paul";  
System.out.println(name);  
String name = "Neve";  
System.out.println(name);
```

2

```
String language;  
language = "Java";  
String Java = "Javascript";
```

3

```
String gender;  
sex = "male";  
System.out.println(gender);
```



You are the judge...! Which of these would WORK?

```
String name;  
name = "Paul";  
System.out.println(name);  
String name = "Neve";  
System.out.println(name);
```

```
String language;  
language = "Java";  
String Java = "Javascript";
```

```
String gender;  
sex = "male";  
System.out.println(gender);
```

You are the judge...!

```
String name;  
name = "Paul";  
System.out.println(name);  
String name = "Neve";  
System.out.println(name);
```



name is declared
twice

```
String language;  
language = "Java";  
String Java = "Javascript";
```

```
String gender;  
sex = "male";  
System.out.println(gender);
```

You are the judge...!

```
String name;  
name = "Paul";  
System.out.println(name);  
String name = "Neve";  
System.out.println(name);
```



name is declared twice

```
String language;  
language = "Java";  
String Java = "Javascript";
```



language and *Java* are two different variables

```
String gender;  
sex = "male";  
System.out.println(gender);
```

You are the judge...!

```
String name;  
name = "Paul";  
System.out.println(name);  
String name = "Neve";  
System.out.println(name);
```



name is declared twice

```
String language;  
language = "Java";  
String Java = "Javascript";
```



language and *Java* are two different variables

```
String gender;  
sex = "male";  
System.out.println(gender);
```



sex is never declared (*gender* is, but that's a different variable)

Variable naming rules

- ⦿ Variable names
 - ⦿ Must begin with a letter, the dollar sign \$, or the underscore character _
 - ⦿ May then contain letters, digits, dollar signs, or underscores
 - ⦿ May NOT contain any other characters - including spaces
 - ⦿ May not be the same as a Java keyword
 - ⦿ Google for "Java language keywords"

Variable naming conventions

- ⦿ Variable names should always start with a lower case (small) letter
 - ⦿ name, language, numberOfBananas ✓
 - ⦿ Name, Language, NUMBEROFBANANAS ✗
- ⦿ Variable names should describe what they store
 - ⦿ userName, programmingLanguage, meaningOfLife ✓
 - ⦿ stringData, x, fredsVariable ✗
- ⦿ If the name you choose consists of only one word, spell that word in all lowercase letters. If it consists of more than one word, capitalise the first letter of each subsequent word
 - ⦿ speed, name, userName, meaningOfLifeTheUniverseAndEverything ✓
 - ⦿ user_name, meaningoflifetheuniverseandeverything ✗

Variable naming conventions

```
String language;  
language = "Java";  
String Java = "Javascript";
```



Variable naming conventions

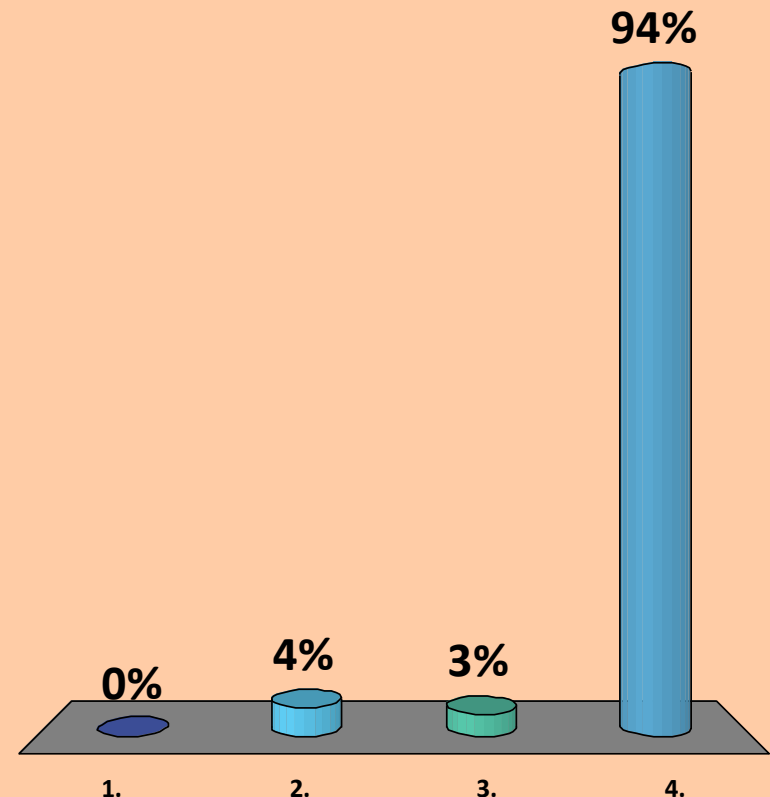
```
String language;  
language = "Java";  
String Java = "Javascript";
```



So what's wrong with this then?

```
String language;  
language = "Java";  
String Java = "Javascript";
```

1. In line 2, you need to add the keyword **String** to declare the variable
2. In line 3, it doesn't make sense - Java is not Javascript
3. The code breaks the rules of the Java language somewhere, so it wouldn't run
4. The code breaks the conventions of the Java language somewhere. It'd run, but experienced programmers would laugh at you



The Scanner

- ⦿ We can use this to get input from the end-user during execution of a program

```
1 Scanner userInput = new Scanner(System.in);  
2 System.out.println("Please type your name.");  
3 String name = userInput.nextLine();  
4 System.out.println("Hello there, "+name);
```

The Scanner

- ⦿ You can use the same Scanner as many times as necessary...
- ⦿ ...so our program could continue thus:

```
Scanner userInput = new Scanner(System.in);  
System.out.println("Please type your name.");  
String name = userInput.nextLine();  
System.out.println("Hello there, "+name);  
System.out.println("Please type your age.");  
int age = userInput.nextInt();  
System.out.println(name+"'s age is "+age);
```

The Scanner

- ⦿ You can use the same Scanner as many times as necessary...
- ⦿ ...so our program could continue thus:

```
Scanner userInput = new Scanner(System.in);  
System.out.println("Please type your name.");
```

```
String name = userInput.nextLine();
```

```
System.out.println("Hello there, "+name);  
System.out.println("Please type your age.");
```

```
int age = userInput.nextInt();
```

```
System.out.println(name+"'s age is "+age);
```

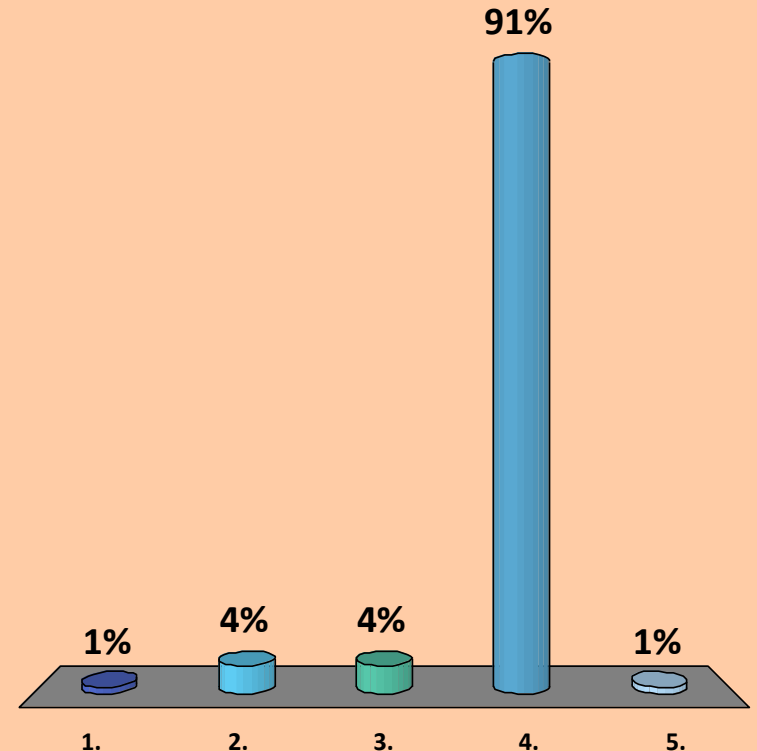
**IMPORTANT
QUESTION:
WHY DO YOU
THINK THESE ARE
DIFFERENT?**

Retro, retro, retro

(or: "let's see who was paying attention during the
first part of the lecture!")

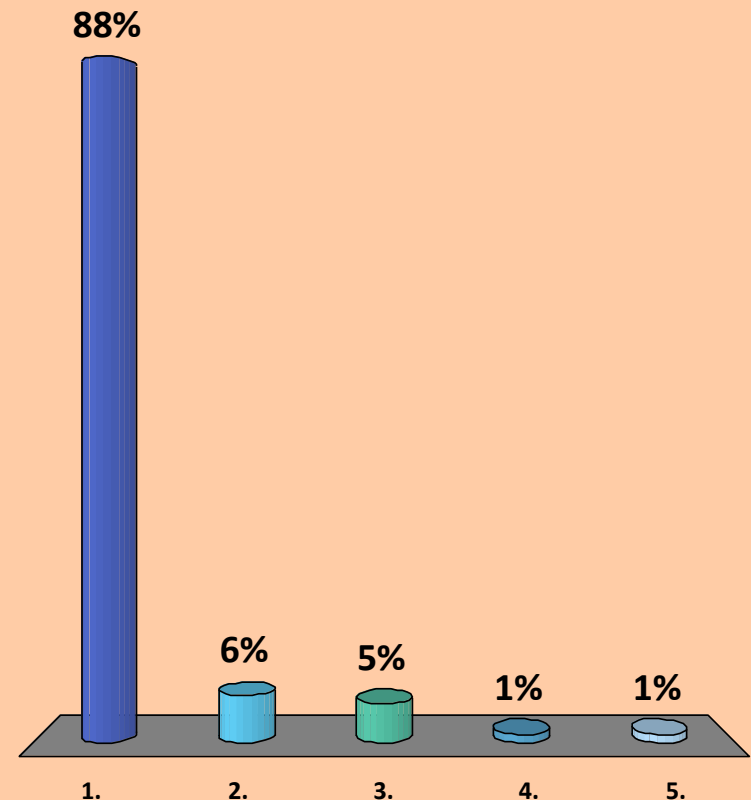
Which of these statements is *untrue*?

1. Java is a compiled language
2. Java is an interpreted language
3. Java is object oriented
4. Java is weakly typed
5. Java is a high-level language



What is the purpose of javac?

1. It is used to compile Java source code to Java byte code
2. It is used to run Java byte code
3. It is used to create and edit Java source code
4. It is used to clear (hence the "C") the screen in Java
5. Something else



Summary

- ⦿ Java is a programming language that is
 - ⦿ general purpose
 - ⦿ platform independent
 - ⦿ object oriented
 - ⦿ both compiled and interpreted
- ⦿ Java source code is compiled to Java bytecode, which runs on the Java Virtual Machine
- ⦿ The JVM is a simulation of a CPU with an instruction set that matches Java bytecode
- ⦿ Your compiled programs run in this simulation - so any platform that has a JVM available can run compiled Java programs

Summary

- ⦿ Java is a C-like language
 - ⦿ Code blocks are delimited with curly brackets { }
 - ⦿ The semi-colon indicates the end of a statement
- ⦿ Programs are comprised of *classes*
- ⦿ Every Java program must have a main *method* where the program starts from
- ⦿ Java is *strongly typed*
 - ⦿ Variables **MUST** have their data types specified when you declare them
 - ⦿ You cannot put data into a variable if the data type doesn't "fit"
 - ⦿ e.g. you couldn't put the text **Hello** into an integer variable!

Summary

- ⦿ Variables **MUST** be declared before you can use them
- ⦿ A variable is declared by specifying its data type and its name
- ⦿ You can only declare a variable once
- ⦿ There are variable naming *conventions* and there are variable naming *rules*
 - ⦿ The *rules* cannot be broken - your programs will not compile!
 - ⦿ The *conventions* are practices that Java programmers have agreed among themselves
 - ⦿ Your programs will compile if you break the conventions - but don't! 😊
- ⦿ Use a **Scanner** to get input from the end user in your programs
- ⦿ Oh, and finally... the Sinclair Spectrum was the best computer ever made 😊