Semester 1

CI101/CI177

Java

Object oriented programming

This is a copy of the notes used in the module CI01.

It is presented to you in the hope that it will be useful and reduce some of the burden of taking notes during the lectures.

However, it will not cover all the material discussed and explored in the lectures, seminars and tutorials.

Information about the course, electronic copy of notes, coursework details, tutorial details, seminar details, programs seen in lectures, Charon system link, extra material, extra programs, etc. is found under Study materials in the module area for CI101 on student central.

In the presentation of material some simplification of ideas has been done to aid initial understanding.

0. Java summary

Core

`char int double boolean`
 	
`short long float`
	primitive types;

`int a = 2;` `int b = 2, c = 4;`	Declaration of a variable

`a = b;` `a = b + c;`	Assignment

`b + c;  // Addition`
`b - c;  // Subtraction`
`b * c;  // Multiplication`
`b / c;  // Division`
`b % c;  // Delivers remainder`
	Arithmetic operators

`class Main`
`{`
`  public static void`
`  main( String args[] )`
`  {`
`  System.out.println("Hello");`
`  }`
`}`
	Complete Java application

Iteration

```
int i = 1;
while ( i <= 10 )
{
  // same code
  i++;
}
```

while loop / counting

repeats code 10 times varying i from 1 .. 10.

```
String name = BIO.getString();

while ( !name.equals("END") )
{
  // same code

  name = BIO.getString();
}
```

while loop /
end on condition

keep reading in a string until the string read in is equal to "END"

```
for (int i=1; i<=10; i++)
{
  // same code
}
```

for loop / counting
repeats code 10 times varying i from 1 .. 10.

Decision

```
if ( price < 10.00 )
{
  // Buy
}
```

if making a decision

```
if ( price < 10.00 )
{
  // Buy
} else {
  // Do not buy
}
```

if making a decision with else part.

```
switch ( month )
{
  case 1 : // Month is Jan
           break;
  case 2 : // Month is Feb
           break;

}
```

switch
An alternate way of writing if.
break must be used to exit, otherwise control will drop to the next case label.

```
switch ( month )
{
  case 1 : // Month is Jan
           break;
  case 2 : // Month is Feb
           break;

  default : // not Jan Feb

}
```

switch

With a default case which is matched when there is no match to a case label.

Arrays

```
class Main
{
  public static void main( String args[] )
  {
    int peopleInRoom[] = new int[3];

    //Allocate a new array initialised
    // of a different size

    peopleInRoom = new int[] {20,30,15,40 };

    //conventional for loop
    for ( int i = 0; i<peopleInRoom.length; i++ )
    {
      System.out.printf("Room %d has %d people\n",
                        i, peopleInRoom[i] );
    }

    //foreach loop
    int totalPeople = 0;
    for ( int people: peopleInRoom )
    {
      totalPeople += people;
    }

    System.out.printf("Total = %d\n", totalPeople );

    String colours[] =
      new String[] { "Red", "Green", "Blue" };

    //for eachloop
    for ( String colour: colours )
    {
      System.out.printf( "%s ", colour );
    }
    System.out.println();
  }
}
```

```
Room 0 has 20 people
Room 1 has 30 people
Room 2 has 15 people
Room 3 has 40 people
Total = 105
Red Green Blue
```

Class, Method, Object, Message

`class NoticeBoard`
`{`
 	
`  private String message="";`
 	
`  public NoticeBoard( String aMes )`
`  {`
`  setMessage( aMes );`
`  }`
 	
`  public void setMessage(String mes)`
`  {`
`  message = mes;`
`  }`
 	
`  public String getMessage()`
`  {`
`  return message;`
`  }`
`}`
	Class encapsulating code & data.
	
	Instance variable
	
	Constructor
	Sets the first message on the notice board.
	
	Method to set a message on an instance of a NoticeBoard.
	
	Method to return the message displayed on the notice board.

`NoticeBoard wall =`
`  new NoticeBoard( "Blank" );`
 	
`String read = wall.getMessage();`
 	
`wall.setMessage("Do not Disturb");`
`read = wall.getMessage();`
	Creation of an instance of the class `NoticeBoard`
	
	Send message `getMessage` to object `wall`.
	
	Set a new message, and retrieve from object `wall`

1. CI101

1.1. What is it about

- Problem solving
- Creating a computer based solution using an imperative programming language

1.2. Strategy

- Assume that you have no programming knowledge
- Use Java as the programming language to illustrate programming ideas
- Concentrate on core ideas/ concepts in programming first
- Initially lots of small exercises to help you build your confidence in writing computer programs
- Try to avoid hiding underlying detail

1.3. Learning computer programming

- It can be difficult at first.
- Programming is in part a skill and needs to be practised.
- Like learning a foreign language it takes time and effort.
- It can be frustrating, as the smallest of 'errors' will cause a program to fail to compile or to produce the wrong answer

1.4. CI101

- This course in not a distance learning course.
- Practice of the concepts explored in the lectures is vital.

  It is very unlikely that you would be able to successfully revise for the CI101 exams in the last month. In some subjects this may be possible, but not for CI101 (programming).

1.5. Careers in software development

- How much does a programmer earn?

  `www.itjobswatch.co.uk`

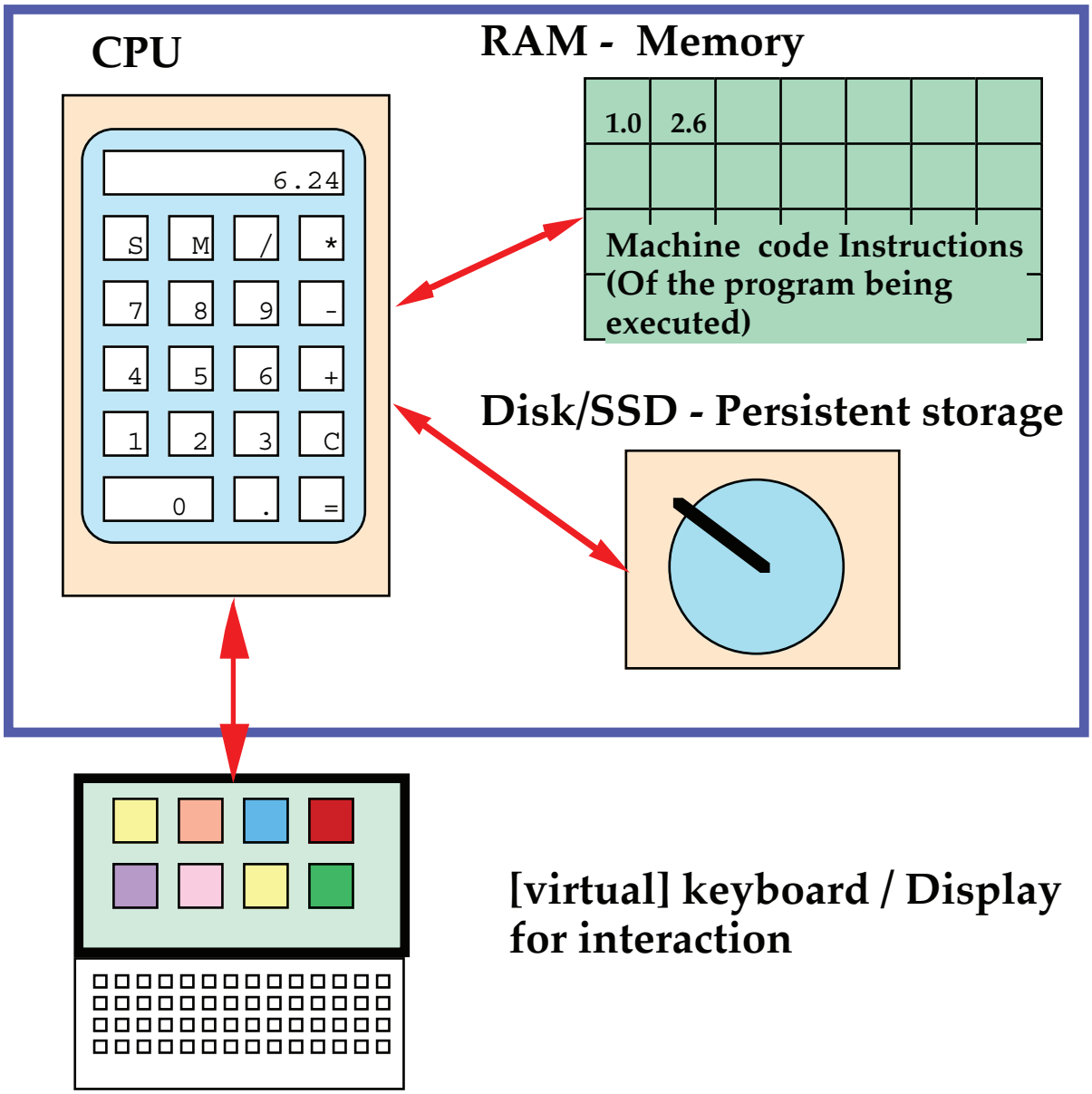
1.6. Why Java

- Used in industry
- Object Oriented
- Many tools, many free
- Has evolved to include many new ideas in programming.

- Influenced the languages:
  Ada 2005, C#, JavaScript, PHP, Python, Scala

- Machine independent
  Can run on 'any' machine
  `Write once run anywhere`

1.7. Simple model of a computer
Many many many simplifications



- **Fetch next instruction cycle**
  Repeat
    Fetch instruction
    Obey instruction
- CPU - Central Processing Unit
- RAM - Random Access Memory

1.8. What is a computer program

- A sequence of instructions that when executed by a computer implements a solution to a problem.

A computer program in the computer programming language **Java** to write the message `"I Like Java"`

```
class Main
{
  public static void main(String args[])
  {
    System.out.println("I Like Java");
  }
}
```

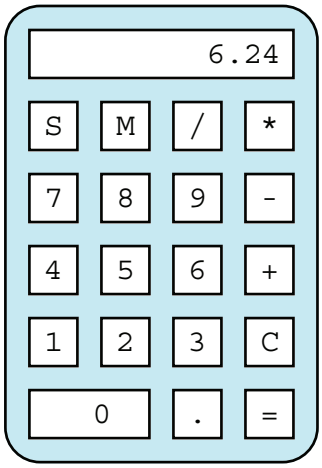
1.9. Initial thoughts

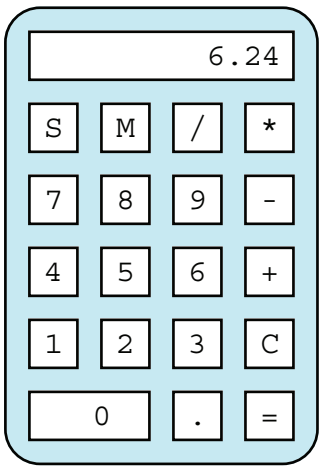
- Not like a natural language (English, French ..)
- Lots of unusual symbols: `{ } [ ] ()`
- Lots of unusual words: **class public static void**
- `System.out.println("I Like Java");`
  Writes `I Like Java`

1.10. Programming

A local orchard sells some of its surplus apples in its farm shop. A customer buying apples, fills a bag full of apples and takes the apples to the shop assistant who weighs the apples to determine their weight in kilograms and then multiples the weight by the price per kilogram to find there cost.

If the shop assistant is good at mental arithmetic they can perform the calculation in their head, or if mental arithmetic is not their strong point they can use an alternative means of determining the cost of the apples.

Pocket calculator	Step	Steps performed, the algorithm
	1	Enter the cost of a kilo of apples: **C 1.2 0**
	2	Enter the operation to be performed: *
	3	Enter the number of kilos to be bought: **5.2**
	4	Enter calculate to display the result =

Pocket calculator	Step	Steps performed, the algorithm
	1	Enter the cost of a kilo of apples: C 1.20
	2	Enter the operation to be performed: *
	3	Enter the number of kilos to be bought: 5.2
	4	Enter calculate to display the result =

1.11. Solution in the programming language Java

Step	Java statements

	`double pricePerKilo;`
	`double kilosOfApples;`
	`double cost;`

1	`pricePerKilo  = 1.20;`
2	`kilosOfApples = 5.2;`
3	`cost = pricePerKilo * kilosOfApples;`
4	`System.out.println( cost );`

1.12. Execution of the Java statements

Step	Java statements

	`double pricePerKilo;`
	`double kilosOfApples;`
	`double cost;`

1	`pricePerKilo  = 1.20;`
2	`kilosOfApples = 5.2;`
3	`cost = pricePerKilo * kilosOfApples;`
4	`System.out.println( cost );`

	Java statements	pricePerKilo	kilosOfApples	cost

	`double pricePerKilo;`
	`double kilosOfApples;`
	`double cost;`
		0.0	0.0	0.0

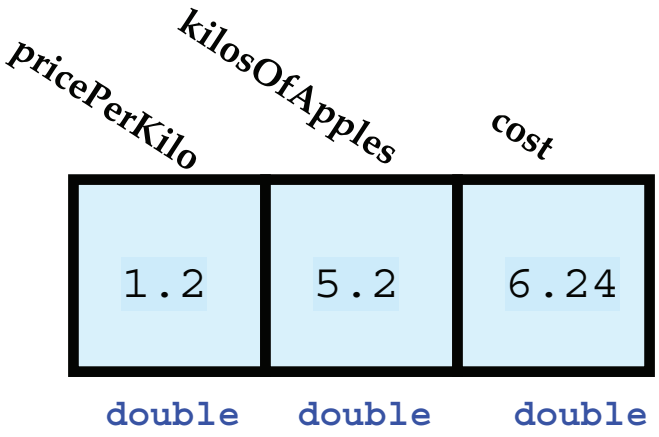
1	`pricePerKilo = 1.20;`	1.20	0.0	0.0
2	`kilosOfApples = 5.2;`	1.20	5.2	0.0
3	`cost = pricePerKilo *` `kilosOfApples;`	1.20	5.2	6.24
4	`System.out.println( cost );`	1.20	5.2	6.24

Note: Meaning full names for variables: **`pricePerKilo`, `kilosOfApples`, ...**

1.11.1. Declaration of a variable

```
double pricePerKilo;
```

Allocates an area of memory called `pricePerKilo` that will hold a number with decimal places. The initial value will be 0.0 (defined by the Java language).



The type of the memory location `pricePerKilo` is **`double`**.

Basically a variable is a memory location into which can be put values which may be changed during the life time of the program. There are many different types of variables. The type of the variable controls what can be put there.

In the above example, the type of the variables, `pricePerKilo`, `KilosOfApples` etc. is of type **`double`**. A variable of type **`double`** may only have numbers with decimal places stored in it. However, the decimal places could be all 0.

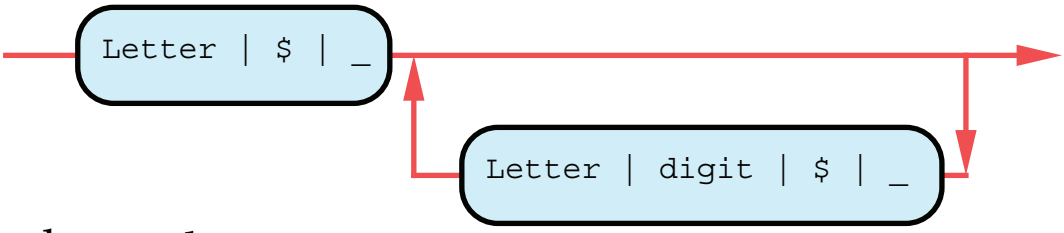
1.12.1. Assignment statement

1.12.2. Definition of the syntax of a variable name

- Simplified definition. Starts with a `letter`, `$` or `_` and then if longer than a single character can have following, only the characters (`letter`, `digit`, `$`, `_`,)

  The convention is that variables start with a lower case letter.

  - As a syntax diagram



  where a letter is:
  `A - Z` or `a - z` or a Unicode character over `0xC0`

  Upper and lower case letters are treated as different characters.

  - As a regular expression

  (letter | $ | _) (letter | digit | $ | _)*

  * zero or more occurrences

Things to investigate

Unicode characters, hexadecimal numbers

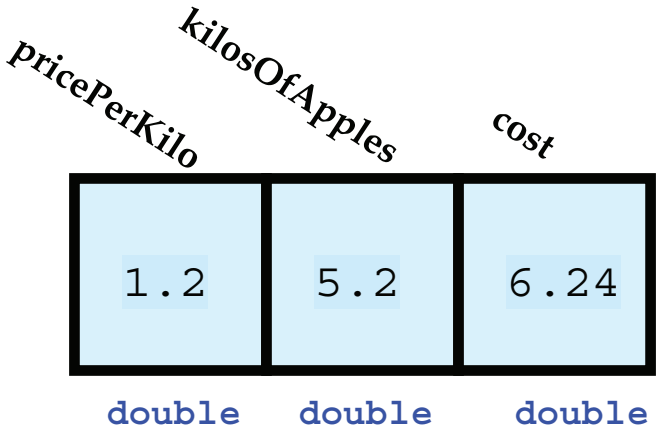
1.13. As a complete application (program)

```
class Main
{
  public static void main( String args[] )
  {
    double pricePerKilo  = 1.20;                  //Memory locations
    double kilosOfApples = 5.2;
    double cost = pricePerKilo * kilosOfApples;  //Calculate

    System.out.print( "Cost of apples is (GBP) " ); //Output answer
    System.out.println( cost );
  }
}
```

```
Cost of apples is (GBP) 6.24
```

Note: Simplified the code by assigning initial values to allocated variables.
Added comments `// Comment`

Memory locations



1.14. Components of the application (program)

```
class Main
{
  public static void main( String args[] )
  {
    double pricePerKilo  = 1.20;              //Initial values
    double kilosOfApples = 5.2;

    double cost = pricePerKilo * kilosOfApples; //Calculate

    System.out.print( "Cost of apples is (GBP) " );
    System.out.println( cost );
  }

}
```

- `class Main`
  Simple explanation: is used as a container for code and data. In the above example the container is called `Main`
  A program may be made up of many instances of these containers. Forms the bases for reusing code from a program into other programs
- `public static void main( String args[] )`
  Simple explanation: is a code fragment (method) that is called when you execute the Java application `Main`
- `System.out.print( "Hello" );`
  Simple explanation: is a call to the library method `print` that in this case will print `Hello`.

  `System.out.println( cost );`
  Simple explanation: is a call to the library method `println` that in this case will print the contents of the variable `cost` followed by a new line

1.15. Layout and comments are important

- The code below (though it will compile) is very very difficult/ impossible to follow even for the author.

```
class Main{public static void main(String args[]){double
pricePerKilo=1.20;double kilosOfApples=5.2;double
cost=pricePerKilo*kilosOfApples;System.out.print(
"Cost of apples is (GBP) " );System.out.println(cost);}}
```

- Though laid out the indentation is so poor that it makes it again very difficult to follow even for the author.

```
class Main
{
    public static void main( String args[] )
{
double pricePerKilo  = 1.20;//Initial values
    double kilosOfApples = 5.2;
double cost = pricePerKilo * kilosOfApples;      //Calculate

System.out.print( "Cost of apples is (GBP) " );
      System.out.println( cost );
}

}
```

1.16. Quick quiz

- Write a Java application that implements the following:

  Calculates the cost of 15 computer memory sticks that cost £5.98 each.

1.17. Quick quiz answers

- Calculate the cost of 15 computer memory sticks that cost £5.98 each.

```
class Main
{
  public static void main( String args[] )
  {
    double number           = 15;          //Initial values
    double memoryStickPrice = 5.98;

    double cost = number * memoryStickPrice; //Calculate cost

    System.out.print( "Cost of memory sticks is (GBP) " );
    System.out.println( cost );
  }
}
```

```
Cost of memory sticks is (GBP) 89.7
```

- Is this alternative solution better or worse

```
class Main
{
  public static void main( String args[] )
  {
    System.out.print( "Cost of memory sticks is (GBP) " );
    System.out.println( 15 * 5.98 );
  }
}
```

1.18. Background / Terminology

- A **double** is one of the primitive types in Java. An instance of a primitive type is called a variable.

  The primitive type **double** can be thought of a description of the way a number is represented. The instance of the **double** is a 'physical' representation of that description in the computers memory called a variable.

  A variable is simply a memory location that holds a value that may be changed (mutable).

```
double kilosOfApples = 1.2;
```

  The above Java statement declares a variable of type **double** called `kilosOfApples` that can hold a number that has decimal places.

- **int** is also a primitive type. However, an instance of an **int** may only hold a whole number (A number without any decimal places)

- Instances of **int**'s and **double**'s are represented in a very different way inside the memory of a computer.

Things to investigate

binary numbers and Floating point numbers.

1.19. Mixing instances of `int` and `double`

```
double price = 2.34
int pay = price;  // Not allowed (As loose precision)
                  // and hence will fail to compile
```

```
int classSize = 20;
double students = classSize;  // OK no loss of precision
```

- This is because Java is a strongly typed language.

? *Is this a good or bad feature*

1.20. casts
Converting between types (ints & doubles)

```
System.out.println(  (6+1) / 3  );
System.out.println(  ( (double) (6+1) ) / 3  );
System.out.println(  (double) (6+1) / 3  );
```

```
2
2.3333333333333335
2.3333333333333335
```

```
System.out.println( (int) 2.4 );
System.out.println( (int) 2.5 );
System.out.println( (int) 2.6 );
```

```
2
2
2
```

1.21. Input of data

The static method BIO.getDouble() will read from the keyboard a number (may have decimal places) typed by the user. This value returned is of type **`double`** and must be assigned to a variable of type **`double`**.

```
double kilosOfApples = BIO.getDouble();
```

Will input into the variable `kilosOfApples` a number typed by the user.

```
class Main
{
  public static void main( String args[] )
  {
    double pricePerKilo = 1.20;                    //Price

    System.out.print( "#Input Kilos of apples: " );
    double kilosOfApples = BIO.getDouble();        //How many

    double cost = pricePerKilo * kilosOfApples;    //Calculate

    System.out.print( "Cost of apples is (GBP) " ); //Result
    System.out.println( cost );
  }
}
```

```
#Input Kilos of apples: 10
Cost of apples is (GBP) 12.0
```

Note: `BIO.getDouble()` is not part of the Java language, but part of a very very simple class library created at Brighton.

All output lines containing a # are ignored by the Charon system.

1.22. Arithmetic operators

Integer arithmetic operators (**int**)

Operator		Example	Delivers
+	Plus	`1 + 2`	`3`
–	Minus	`3 - 2`	`1`
*	Multiplication	`2 * 3`	`6`
/	Division	`7 / 3`	`2`
%	Remainder	`7 % 3`	`1`

1.23. Floating point arithmetic operators (**double**)

Operator		Example	Delivers*
+	Plus	`1.2 + 2.2`	`3.3`
–	Minus	`3.2 - 2.1`	`1.1`
*	Multiplication	`2.1 * 3.0`	`6.3`
/	Division	`2.1 / 4.2`	`0.5`
%	Remainder	`7.3 % 3.1`	`1.1`

* Not the whole truth
Ignores the effects on precision of binary to decimal conversions

1.24. Input of an integer number

The static method `BIO.getInt()` will read from the keyboard an integer number typed by the user. This value must be assigned to a variable of type `int`.

```
int iDays = BIO.getInt();  //Whole number
```

Will input into the variable `iDays` an integer number typed by the user.

A program in Java to input a number of days and calculate the number of weeks and days this represents.

```
class Main
{
  public static void main( String args[] )
  {
    System.out.print( "#Input number of days  " );
    int iDays = BIO.getInt();

    System.out.print( iDays );
    System.out.print( " days is equal to " );

    int weeks = iDays / 7;                //Number of weeks
    System.out.print( weeks );
    System.out.print( " week(s) " );

    int days = iDays % 7;                 //Remaining days
    System.out.print( days );
    System.out.println( " day(s) " );
  }
}
```

```
#Input number of days  20
20 days is equal to 2 week(s) 6 day(s)
```

1.25. Terminology

- Java
  A programming language

- primitive type
  double, **int** [there are others]
  The 'type' of the item:
  double May have decimal places
  int A whole number, no decimal places

- variable
  An instance of a primitive type , reference type.
  A location in the memory of the computer:
  the variable will hold a value
  the variables contents may be changed.

1.26. What you should know

- That you need to spend at least 5 hours per week on this module in addition to attending lectures, seminars and tutorials.
- That there will be assessed programming exercises to do throughout the year.
- That learning a computer programming language is not an activity that you can undertake in the last few weeks of the academic year.
- How to write a small Java application
- How to read in a data value and store it in a variable
- How to declare instances of `double`'s and `int`'s in a Java program
- How to print instances of a `String` **`double`** and **`int`** in Java
- How to compile and run a Java application using BlueJ (Tutorial this week)

1.27. What you need to do for next week

- Complete the first weeks tutorial.
- Start on the work for the first part of the portfolio that you will create throughout the course.

  Think like a programmer

2. Iteration

It would make writing a program very difficult and long winded if we could only write a program were each instruction was executed only once.

- Iteration (looping) constructs
  A statement to allow a section of program code to be executed whilst the condition (Boolean expression) remains true.

  ```
  while (Booleanexpression )statement
  ```

- Use of { } to bind several statements together

- `printf`
  An easy way of printing data items in a specific format (Number of decimal places, field width, alignment etc.)

- Why 1.2 + 1.2 + 1.2 does not equal 3.6

2.1. Core ideas in an imperative programming language

- Sequence

```
double pricePerKilo = 1.20;

System.out.print( "#Input Kilos of apples: " );
double kilosOfApples = BIO.getDouble();

double cost = pricePerKilo * kilosOfApples;

System.out.print( "Cost of apples is (GBP) " );
System.out.println( cost );
```

- Iteration
- Selection
- Abstraction

2.2. A better way of printing data in Java

In Java it is expected that I/O will be via a GUI, hence input and output to the screen is primitive.

There are library classes to help you, but these are not initially straightforward to use.

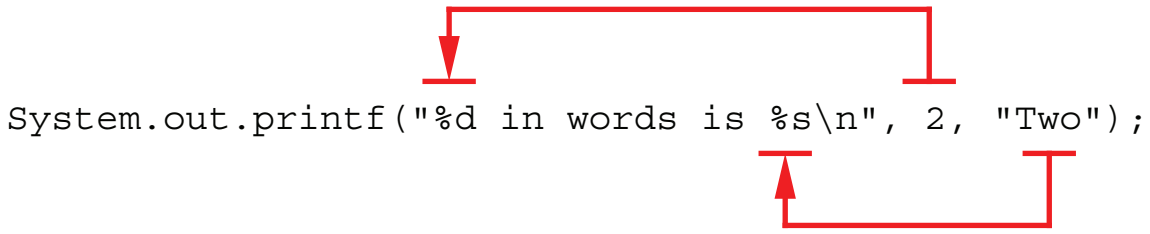
However, there is a relatively simple method that can help, this is `System.out.printf`

2.3. An example of `System.out.printf`

```
System.out.printf("%d in words is %s\n", 2, "Two");
```

Which when compiled and run as part of a program will print:

```
2 in words is Two
```



- Normally characters in the format string are simply printed. However:
- `%d` is replaced by the contents of the next parameter that occurs after the format string. (Which must be an instance of an **`int`**)
- `%s` is replaced by the contents of the next parameter.
  (Which must be an instance of a `String`)
- `\n` represents the new line character.

However be careful as the following will fail at run-time.

```
System.out.printf("%s is %d\n", 2, "Two" );
```

- An instance of an **int** (in this case 2) can not be printed as a string, neither can an instance of a String be printed as an **int**. (A rather unfriendly run time error will be generated, see below)

```
2 is
Exception in thread "main"
java.util.IllegalFormatConversionException: d != java.lang.String
        at java.util.Formatter.failConversion(Formatter.java:4302)
        at java.util.Formatter.printInteger(Formatter.java:2793)
        at java.util.Formatter.print(Formatter.java:2747)
        at java.util.Formatter.format(Formatter.java:2520)
        at java.io.PrintStream.format(PrintStream.java:970)
        at java.io.PrintStream.printf(PrintStream.java:871)
        at Main.main(j00_f01.java:5)
```

2.3.1. The \ character in a string or char constant

- The character \ is used to introduce several non printing characters:

  \n A new line
  \t A tab character

  But also, to allow the characters " and ' to be part of a String or character constant respectively.

  \" Allows the " character to be in a String
  \' Allows the ' character to be in a char constant
  \\ Allows the \ character to be in either a String or char constant.

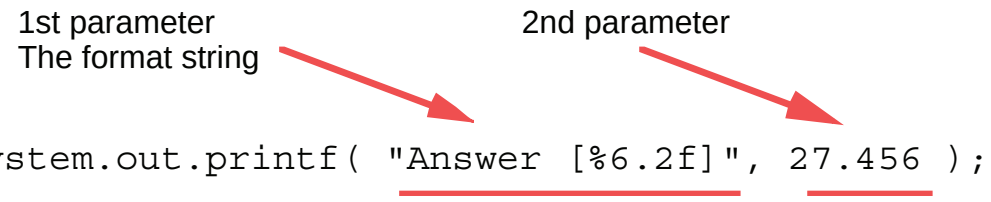
2.4. Controlling how a number etc. is printed

```
System.out.printf( "Answer [%6.2f]", 27.456 );
```

when compiled and run would print

```
Answer [ 27.46]
```

- The first parameter of printf `%6.2f` is replaced when printed by the contents of the second parameter formatted (in this case) in a field width of 6 character positions with 2 decimal places . (`f` is an abbreviation for floating point number)



- Note the number is rounded to 2 decimal places in the 'printed' output.

  `printf` is like `print` no new line is printed.

- Several values can be interpolated into the format string:

```
System.out.printf("Answer [%5.2f]  [%6.2f]", 0.4, 0.479 );
```

when compiled and run would print

```
Answer [ 0.40]  [  0.48]
```

- `%5.2f` is replaced by the contents of the second parameter and `%6.2f` is replaced by the contents of the third parameter.

2.4.1. Printing integers

```
System.out.printf("[%d]\n",      42 );
System.out.printf("[%3d]\n",     42 );
System.out.printf("[%-3d] left justified\n",    42 );
System.out.printf("[%-3d] left justified\n",    4200 );
```

Which when compiled and run as part of a program will print:

```
[42]
[ 42]
[42 ] left justified
[4200] left justified
```

- `%3d`
  print an instance of an int in a field width of 3 character positions

- `%-3d`
  print an instance of an int in a field width of 3 character positions left justified

2.4.2. Printing instances of a `String`

```
System.out.printf("[%s]\n",     "Brighton" );
System.out.printf("[%10s]\n",   "Brighton" );
System.out.printf("[%-10s]\n",  "Brighton" );
System.out.printf("[%6s]\n",    "Brighton" );
System.out.printf("[%6.5s]\n",  "Brighton" ); //Width.trunc
System.out.printf("[%-6.5s]\n","Brighton" ); //Width.trunc
```

Which when compiled and run as part of a program will print:

```
[Brighton]
[  Brighton]
[Brighton  ]
[Brighton]
[ Brigh]
[Brigh ]
```

- `%10s`
  Print an instance of a `String` in a field width of 10 character positions

- `%6.5s`
  Print an instance of an `String` in a field width of 6 character positions with the string truncated to 5 characters.

- `%-6.5s`
  As above but left justified

2.4.3. Printing an instance of `float` or `double`

```
System.out.printf("[%9.4f]\n",  1.54 );     //Width.places
System.out.printf("[%-9.4f]\n", 1.54 );     //Width.places
System.out.printf("[%-9.3f]\n", 1.54321 );  //Width.places
```

```
[   1.5400]
[1.5400   ]
[1.543    ]
```

- `%9.4f`
  Print an instance of an float or double in a field width of 9 character positions to 4 decimal places (rounded)

- `%-9.3f`
  As above but left justified and to 3 decimal places

2.5. Adding to a variable

```
int count = 1;
```

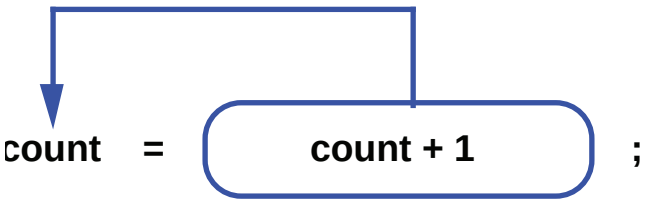
```
count = count + 1

System.out.println( count )
```

will print

```
2
```

How to read and understand

```
count = count + 1
```



To the contents of the variable `count` add `1` and then assign the result to the variable `count`.

Remember = means assigned, **not** equals

```
int count = 1;
```

```
count = count + 1
System.out.println( count )
```

```
count = count + 1
System.out.println( count )
```

```
count = count + 1
System.out.println( count )
```

If you executed the above code it would print

```
2
3
4
```

2.6. Simple problem (revisited)

A local orchard sells some of its apples in its local farm shop. A customer buying apples, fills a bag full of apples and takes the apples to the shop assistant who weighs the apples to determine their weight in kilograms and then multiples the weight by the price per kilogram to find there cost.

If the shop assistant is good at mental arithmetic they can perform the calculation in their head, or if mental arithmetic is not their strong point they can use an alternative means of determining the cost of the apples.

2.7. Table of cost of apples for various weights

```
#Cost of apples per kilo : 1.20
Kilos     Price
  0.1       0.12
  0.2       0.24
  0.3       0.36
  0.4       0.48
  0.5       0.60
  0.6       0.72
  0.7       0.84
  0.8       0.96
  0.9       1.08
  1.0       1.20
  1.1       1.32
  1.2       1.44
  1.3       1.56
  1.4       1.68

etc.
```

Java code to print a table of the cost of apples from 0.1 Kilograms to 10 kilogram in steps of 0.1

```
class Main
{

  public static void main( String args[] )
  {
    double pricePerKilo;                          //Price of apples
    double kilosOfApples;                         //Apples required
    double cost;                                  //Cost of apples

    System.out.print( "#Cost of apples per kilo : " );
    pricePerKilo = BIO.getDouble();

    System.out.println("Kilos  Price");
```

```
    kilosOfApples           = 0.1;
```

```
    cost = pricePerKilo * kilosOfApples;
    System.out.printf("%5.1f   %6.2f",
                      kilosOfApples, cost );
    System.out.println();
```

```
    kilosOfApples           = 0.2;
```

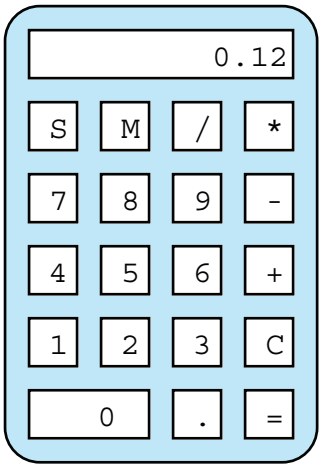
```
    cost = pricePerKilo * kilosOfApples;
    System.out.printf("%5.1f   %6.2f",
                      kilosOfApples, cost );
    System.out.println();
```

etc.

```
  }
}
```

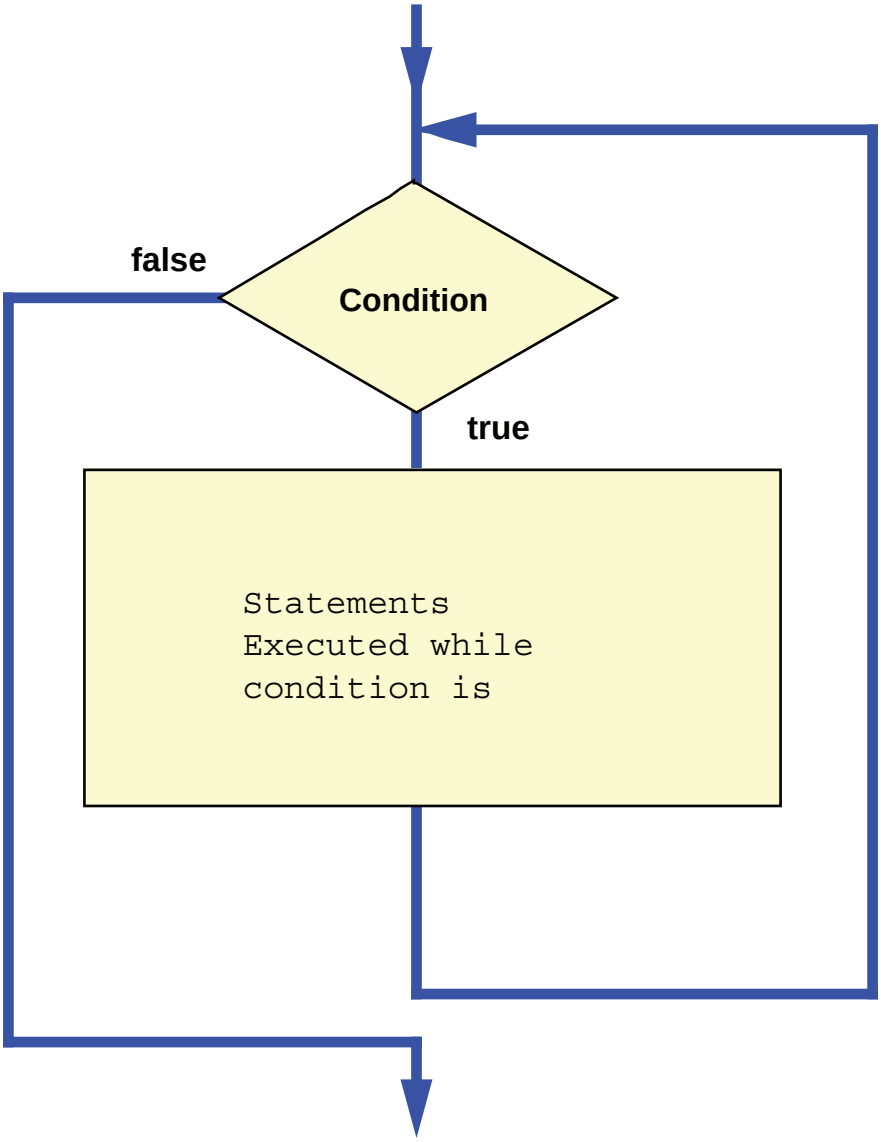
? Why is this not a good solution

2.8. Working out the table on a pocket calculator

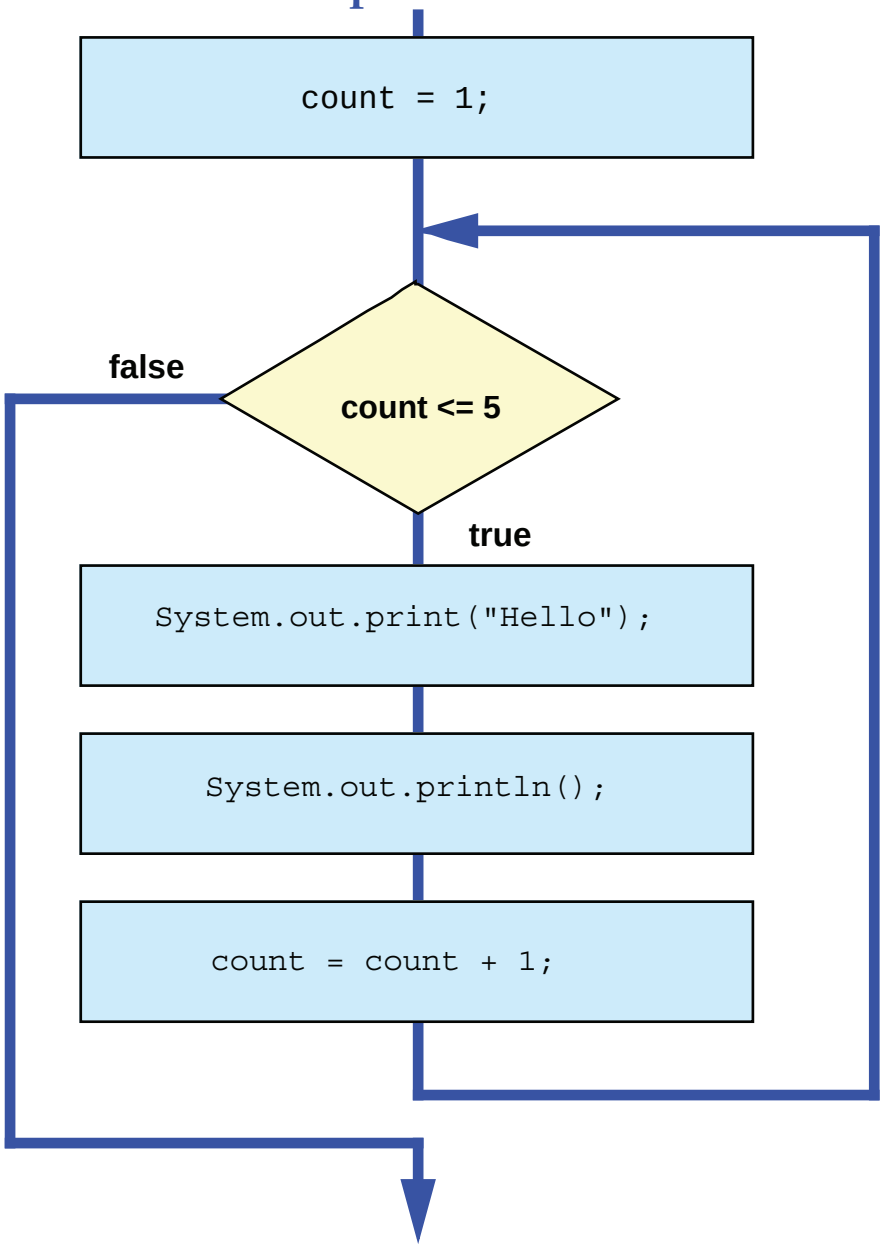
Pocket calculator	Step	Steps performed
	1.	Enter the cost of a kilo of apples: **C 1 . 2 0**
	2.	Save this value to the calculator's memory: **S**
	3.	Retrieve the value from memory: **M**
	4.	Enter the operation to be performed: *
	5.	Enter the number of kilos to be bought: **0. 1**
	6.	Enter calculate to display the result **=**

To calculate the cost of the apples for the next weight repeat steps 3 - 6 (replace the value in line 5 with the next value for the number of kilo's required)

2.9. while construct (as a flowchart)



2.10. A flow chart to print "Hello" 5 times



- count <= 5
  Read as: count less than or equal to 5

A fragment of Java code to write Hello 5 times

```
int count = 1;                    //Set count to 1
while ( count <= 5 )              //While count less than or equal 5
{
  System.out.print( "Hello" ); // Print Hello
  System.out.println();
  count = count + 1;              // Add 1 to count
}
```

2.11. Introduction to a boolean expression

- A Boolean expression evaluates to true or false. there are no other possibilities

You must know these

>	Greater than
<	Less than

then the others logical expressions follow

2.12. Symbols used

Symbol	Read as
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
==	Equal
!=	Not equal
!	Not

logical <expression>	Read as
`money < 20`	money is less than 20
`seats > 30`	seats are greater than 30
`rooms <= 10`	rooms are less than or equal to 10
`cars >= 60`	cars are greater than or equal to 60
`seats == 35`	seats equals 35
`rooms != 4`	rooms not equal to 4

2.13. Quick quiz

- Which of the following are true if:
  `money` contains 10
  `size` contains 10
  `mark` contains 80

Boolean expression	Read as	True / False
`money == 20`		
`size != 20`		
`size <= 5`		
`mark >= 70`		

- Write a loop in Java to print Happy birthday 10 times

2.14. Quick quiz answers

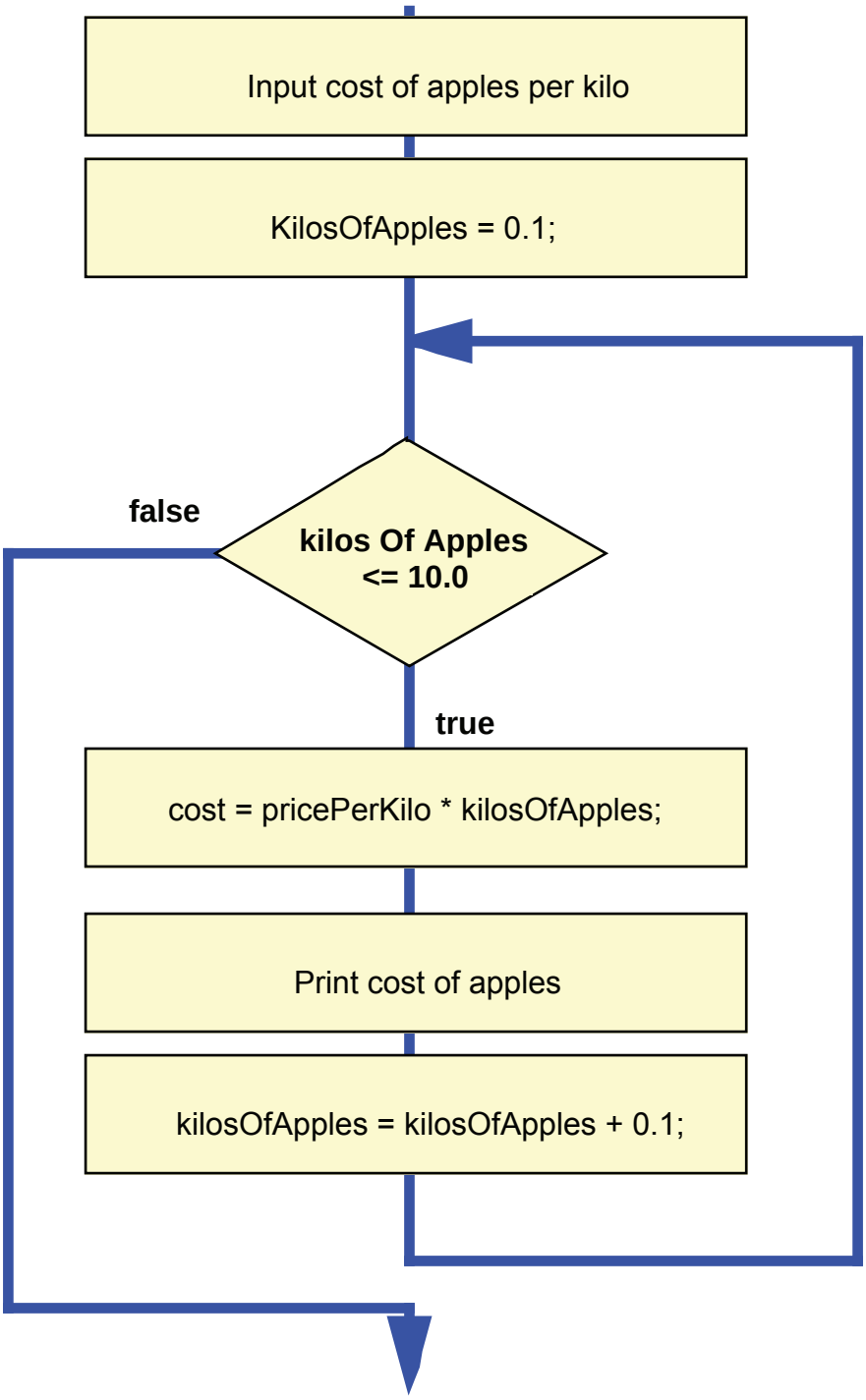
- Which of the following are true if:
  `money` contains 10
  `size` contains 10
  `mark` contains 80

Boolean expression	Read as	T/F
`money == 20`	money equal to 20	F
`size != 20`	size not equal 20	T
`size <= 5`	size less than or equal to 5	F
`mark >= 70`	mark greater than or equal to 70	T

- Write a loop in Java to print Happy birthday 10 times

```
class Main
{
  public static void main( String args[] )
  {
    int i=1;
    while ( i <= 10 )
    {
      System.out.println("Happy birthday");
      i = i + 1;
    }
  }
}
```

2.15. Table (A better solutions) as a flowchart



2.16. Table (A better solution)

To print a table of the cost of apples from 0.1 Kilograms to 10 kilogram in steps of 0.1

```
class Main
{
  public static void main( String args[] )
  {
    double pricePerKilo;                          //Price of apples
    double kilosOfApples;                         //Apples required
    double cost;                                  //Cost of apples

    System.out.print( "#Cost of apples per kilo : " );
    pricePerKilo = BIO.getDouble();

    System.out.println("Kilos  Price");
```

```
    kilosOfApples          = 0.1;

    while ( kilosOfApples <= 10.0 )               //less than 10.0
    {
      cost = pricePerKilo * kilosOfApples;        //New cost
      System.out.printf("%5.1f      %6.2f",
                        kilosOfApples, cost );
      System.out.println();

      kilosOfApples = kilosOfApples + 0.1;
    }
```

```
  }
}
```

```
Cost of apples per kilo  : 1.20
Kilos     Price
  0.1       0.12
  0.2       0.24
  0.3       0.36
  0.4       0.48
  0.5       0.60
  0.6       0.72
  0.7       0.84
  0.8       0.96
  0.9       1.08
  1.0       1.20
  1.1       1.32
  1.2       1.44
  1.3       1.56
  1.4       1.68

etc.
```

2.17. Do not always believe a computer

```
class Main
{
  public static void main( String args[] )
  {
    double value = 1.2 + 1.2 + 1.2;

    System.out.println( value );
  }

}
```

```
3.5999999999999996
```

Simple explanation

In a computer an instance of a **`double`** like an **`int`** is held as a binary number. There is not always an exact equivalence between a number with decimal places and a number with binary places. Internally the computer uses binary to represent numbers. Hence when converted back from binary to decimal the answer may not be exactly the original decimal number.

-> floating point numbers, binary and decimal fractions

? *Should you hold a monetary amount in an instance of a double*

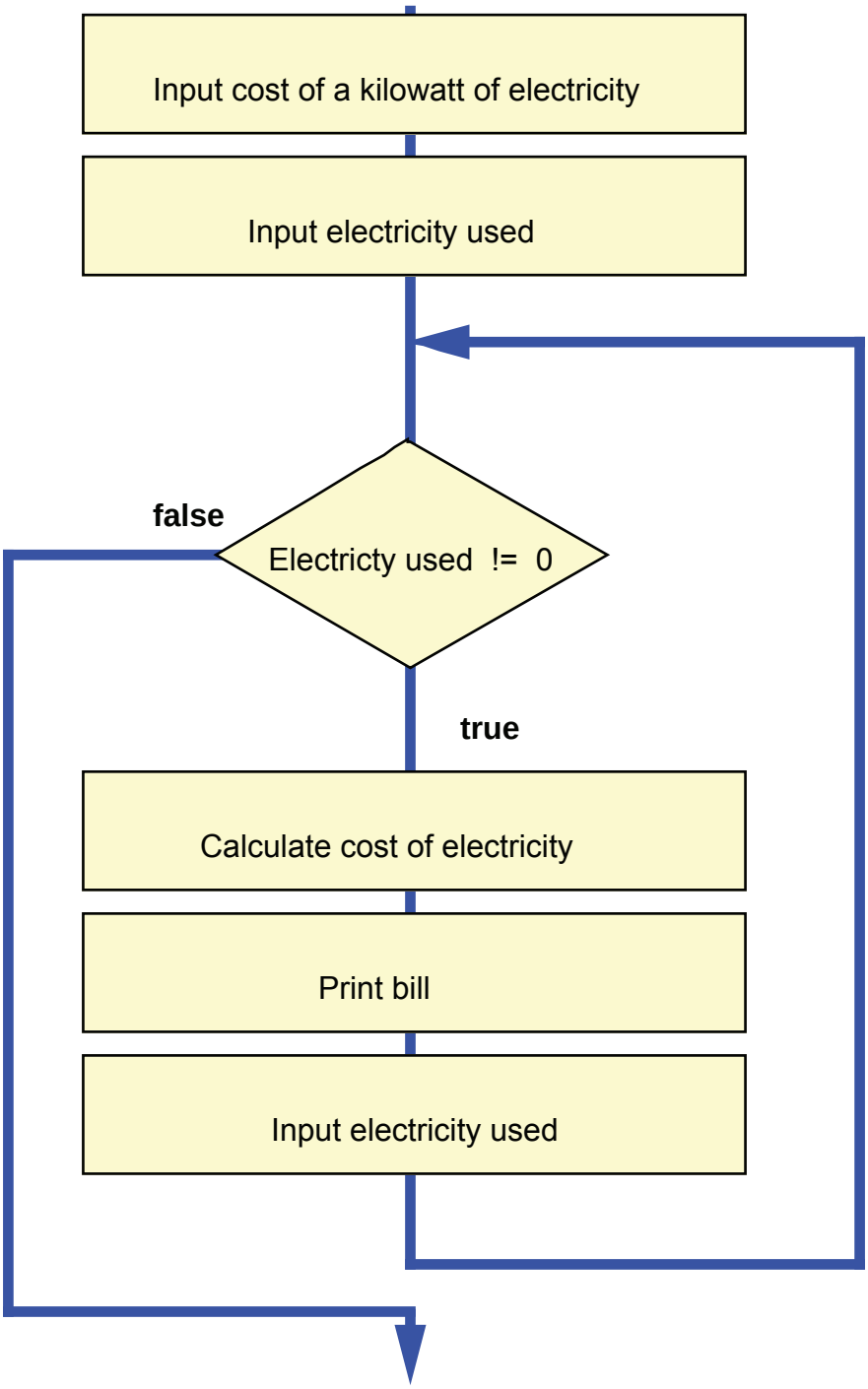
2.18. Electricity cost

A program to allow a user to work out the cost of running several different electrical devices.

The program should be able to:

- Allow the user to enter the cost in pence per kilowatt hour for electricity.
- Allow the user to enter the used amount of electricity for a device.
- Allow the user to find the cost of running several different devices.
- Allow the user to stop the program (enter 0 for used)

2.19. Electricity cost - As a flowchart



2.20. Electricity cost - As a program in Java

```
class Main
{
  public static void main( String args[] )
  {
    System.out.print("#Enter cost per kilowatt (pence)   : " );
    double pricePerKilowatt = BIO.getDouble();

    System.out.print("#Enter kilowatts used :            : " );
    double used = BIO.getDouble();

    while ( used != 0.0 )
    {
      double cost = pricePerKilowatt/100.0 * used;
      System.out.printf(
                "used kilowatts = %6.2f cost GBP = %6.2f\n\n",
                 used, cost );

      System.out.print("#Enter kilowatts used :             " );
      used = BIO.getDouble();
    }
  }
}
```

```
#Enter cost per kilowatt (pence)   : 14.23
#Enter kilowatts used :              100
used kilowatts = 100.00 cost GBP =  14.23

#Enter kilowatts used :              0
```

- But what if the electricity used is a negative number?

2.21. Electricity billing system

A program to print customer bills

Requirements.

- Be able to set the cost for a kilowatt hour of electricity use.
- Be able to print an electricity bill for each customer
- Stop when a customer number of 0 is entered.

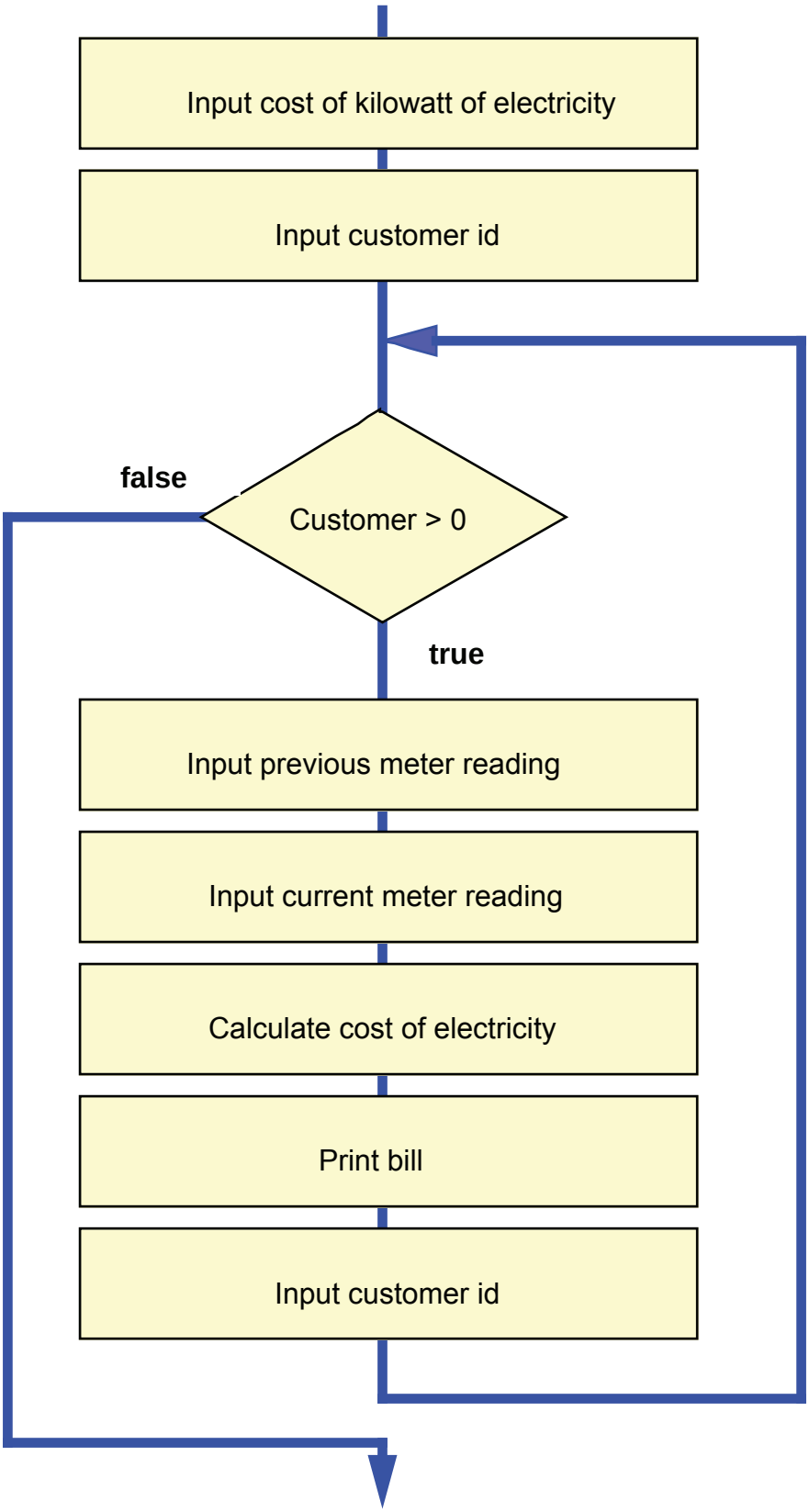
The data provided will be:

The cost in pence per kilowatt hour for electricity.

Then for each customer:

Customer number:	an integer number (7 digits)
Previous usage:	A meter reading (6 digits 1 DP)
Current usage:	A meter reading (6 digits 1 DP)

2.22. Electricity billing system - As a flowchart



2.23. Electricity billing system - As a program

```
class Main
{
  public static void main( String args[] )
  {
    System.out.print("#Cost per kilowatt hour (Pence) : " );
    double pricePerKilowatt = BIO.getDouble();

    System.out.print("#Enter customer id              : " );
    int    customer = BIO.getInt();

    while ( customer > 0 )
    {
      System.out.print("#Enter previous meter reading   : " );
      double previous = BIO.getDouble();
      System.out.print("#Enter current meter reading    : " );
      double current  = BIO.getDouble();

      double used = current - previous;
      double bill = pricePerKilowatt/100.0 * used;

      System.out.printf("Customer [%8d] ", customer);
      System.out.printf(" Meter now %8.1f last %8.1f \n",
                        current, previous );
      System.out.printf("Units used %8.1f  ", used );
      System.out.printf("amount now due (GBP) %8.2f",  bill );

      System.out.println();
      System.out.println();

      System.out.print("#Enter customer id              : " );
      customer = BIO.getInt();
    }
  }
}
```

The data for the program

```
14.23
9934567
10567.3
10975.7
9900001
50300.4
52546.9
9957777
32384.7
32340.7
9954258
56444.5
56444.5
0
```

The results produced

```
Customer [ 9934567]  Meter now  10975.7 last  10567.3
Units used    408.4  amount now due (GBP)    58.12


Customer [ 9900001]  Meter now  69546.9 last  50300.4
Units used  19246.5  amount now due (GBP)  2738.78


Customer [ 9957777]  Meter now  32340.7 last  32384.7
Units used    -44.0  amount now due (GBP)    -6.26


Customer [ 9954258]  Meter now  56444.5 last  56444.5
Units used      0.0  amount now due (GBP)     0.00
```

2.24. Reflection on the results produced

```
Customer [ 9934567]  Meter now  10975.7 last  10567.3
Units used    408.4  amount now due (GBP)    58.12

Customer [ 9900001]  Meter now  69546.9 last  50300.4
Units used  19246.5  amount now due (GBP)  2738.78

Customer [ 9957777]  Meter now  32340.7 last  32384.7
Units used    -44.0  amount now due (GBP)    -6.26

Customer [ 9954258]  Meter now  56444.5 last  56444.5
Units used      0.0  amount now due (GBP)     0.00
```

The data was

```
14.23
9934567
10567.3
10975.7
9900001
50300.4
52546.9
9957777
32384.7
32340.7
9954258
56444.5
56444.5
0
```

The programmer did not consider

- Data validation
  GIGO - Garbage in Garbage out
- Ways of refunding a customer.
- The case when no electricity was used.

- Is this important?

2.25. What you should know

- That you need to spend at least 5 hours per week on this module in addition to attending lectures, seminars and tutorials.
- That there will be assessed programming exercises to do throughout the year.
- That learning a computer programming language is not an activity that you can undertake in the last few weeks of the academic year.
- How to write a small Java application using a while loop.
- Be aware of the need for checking as the data may not be the normal expected values.

2.26. What you need to do for next week

- Complete the first practical exercises which will form part of your portfolio.
- Consider what data items need to be checked so that the electricity billing program will be able to produce sensible results.

  Think like a programmer

3. Algorithms

- Algorithms
- Analysis of Algorithms

3.1. Core ideas in an imperative programming language

- Sequence

```
double pricePerKilo = 1.20;

System.out.print( "#Input Kilos of apples: " );
double kilosOfApples = BIO.getDouble();

double cost = pricePerKilo * kilosOfApples;

System.out.print( "Cost of apples is (GBP) " );
System.out.println( cost );
```

- Iteration

```
int count = 1;

while ( count <= 5 )
{
  System.out.println( "Hello" );
  count = count + 1;
}
```

- Selection
- Abstraction

3.2. What is an algorithm

A step by step process for calculating a solution to a particular problem.

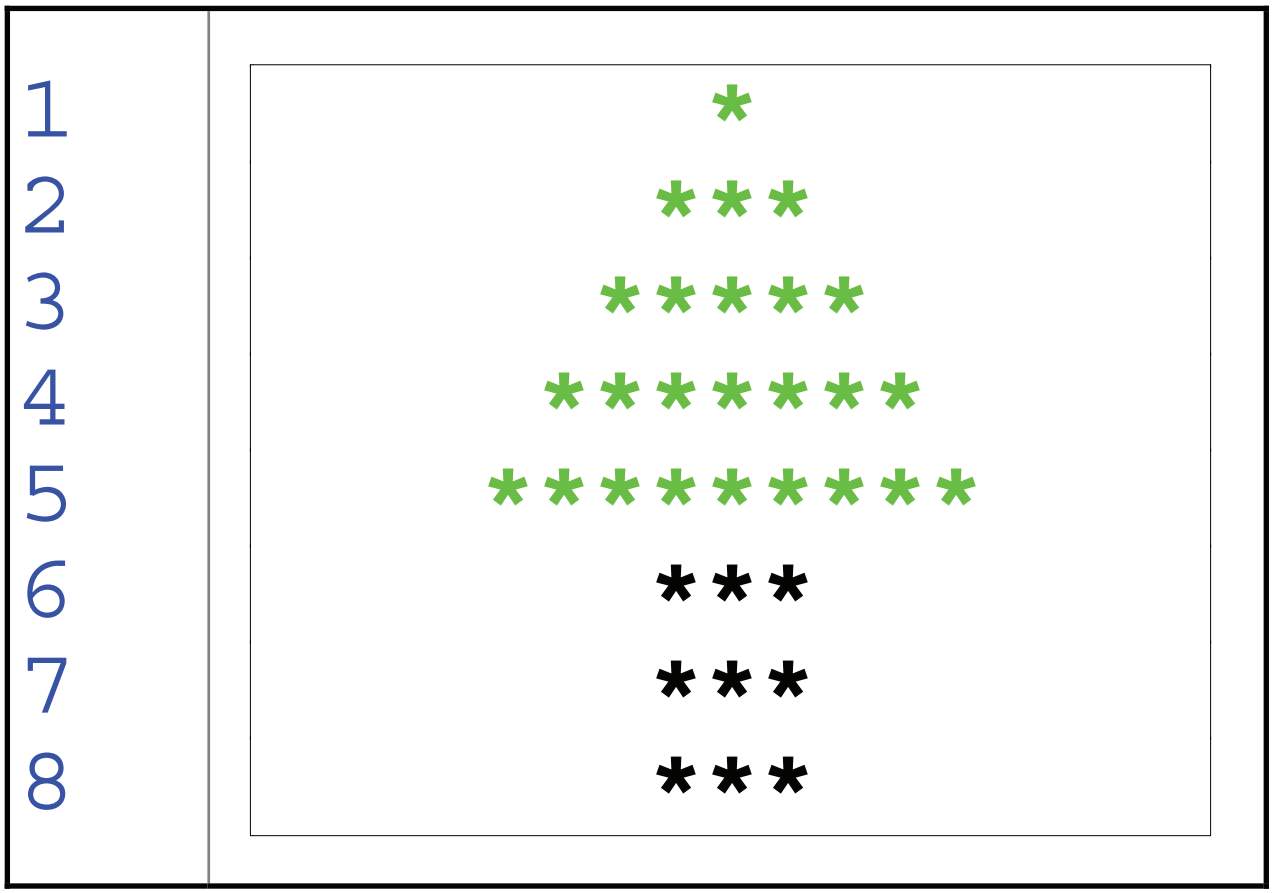
Example algorithms

- Calculate the compound interest on a loan
- Recipe for baking a chocolate cake
- Determine if a bullet has hit its target in a computer game (collision detection).
- Find where the web server for `brighton.ac.uk` is on the internet.
- Decide if a movie will be a hit / make money `www.epagogix.com`

The Java class library contains many algorithms that can be used in the solution of a problem.

Seminal book by Niklaus Wirth
Algorithms + data structures = programs

3.3. Printing a tree using a raster graphics approach



In a raster graphics approach all of line 1 is printed then all of line 2, then all of line 3, etc.

3.3. An algorithm to print a stylised tree

The algorithm will print the tree to any height of foliage. However, we have to do this by printing ASCII characters.

Note: the foliage is represented by stars and the font is monospaced (like an ancient mechanical typewriter) Colour for illustration only.

First attempt! - (Expressed as a Java code) What is wrong

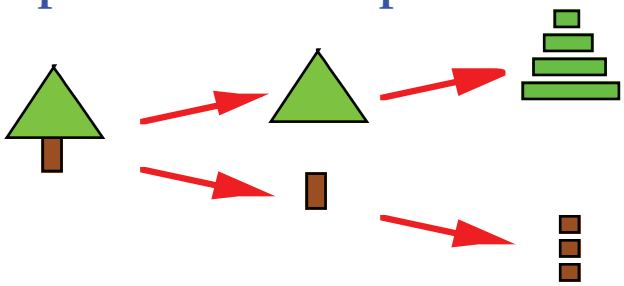
```
class Main
{
  public static void main( String args[] )
  {
    System.out.println( "         *"          );
    System.out.println( "        ***"         );
    System.out.println( "       *****"        );
    System.out.println( "      *******"       );
    System.out.println( "     *********"      );
    System.out.println( "    ***********"     );
    System.out.println( "   *************"    );
    System.out.println( "  ***************"   );
    System.out.println( "*****************"  );
    System.out.println( "        ***"         );
    System.out.println( "        ***"         );
    System.out.println( "        ***"         );
  }
}
```

3.4. Parts of a tree (Height of foliage 4 lines)

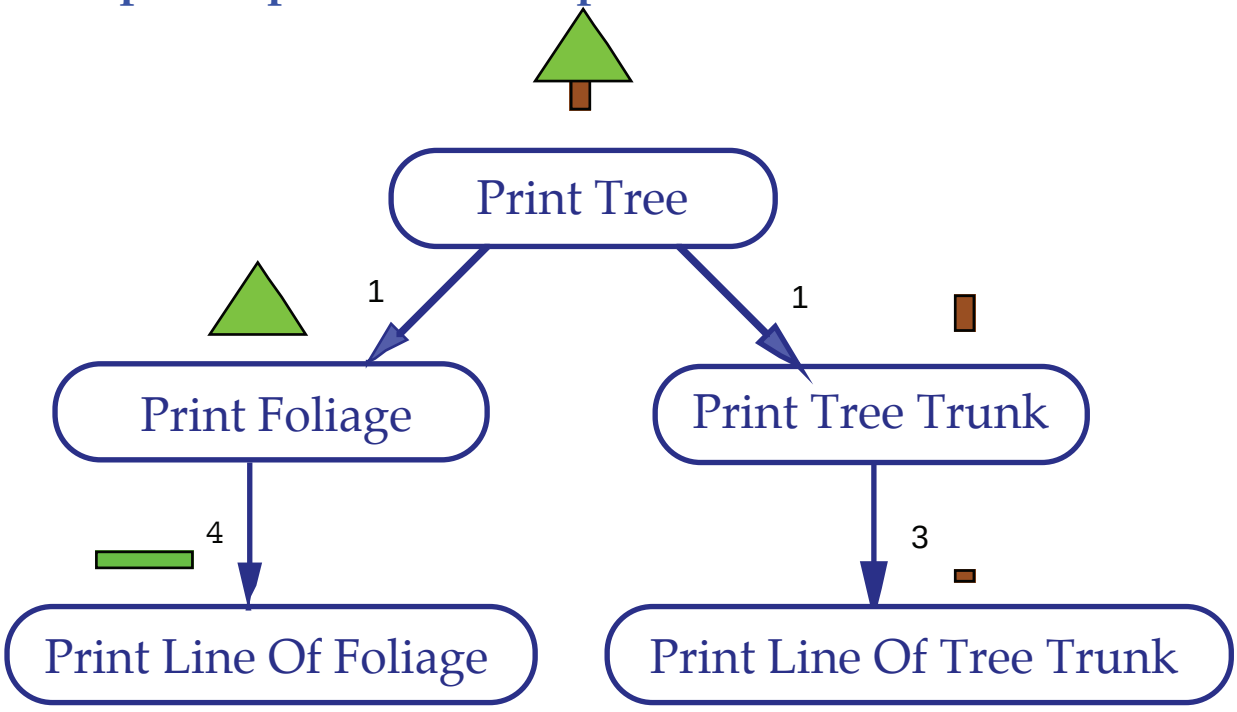
```
   *
  ***
 *****
*******
  ***
  ***
  ***
```

Top down approach

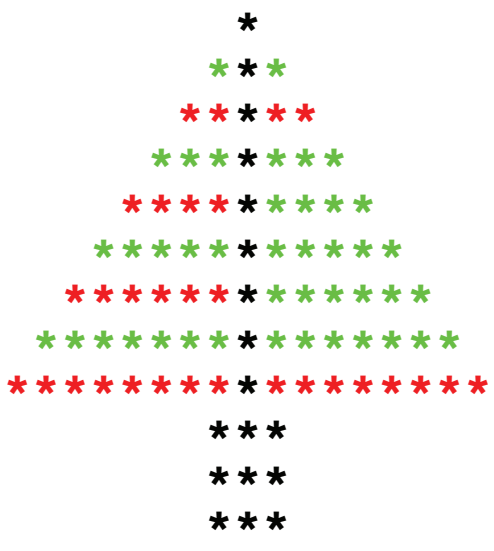
Diagrammatic split of tree into parts



Conceptual split of how to print a tree



3.5. Looking at each line of a tree (9 lines of foliage)

Tree Colour for illustration of algorithm	line	spaces in front
	1 2 3 4 5 6 7 8 9 10 11 12	8 7 6 5 4 3 2 1 0 7 7 7

3.5.1. Algorithm to print spaces at start of `line`

- Need to know the height (the line number of the last line of foliage **`foliageHeight`**).

Pseudo code:

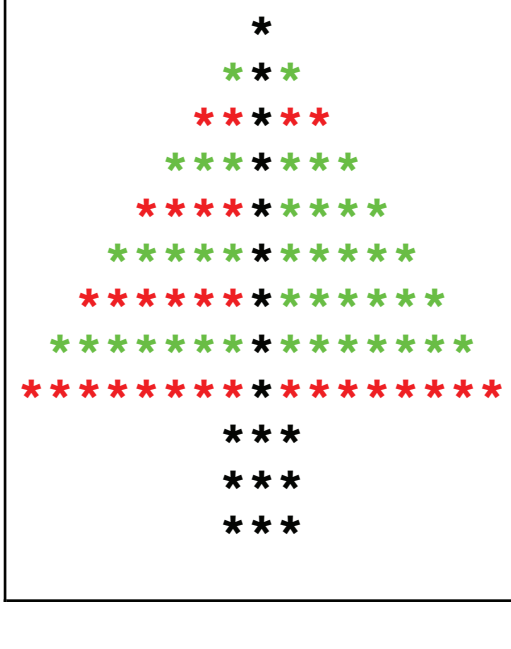
```
print foliageHeight - line spaces
```

3.5.2. Java code to print the spaces at start of `line`

```
int spaces = 1;
while ( spaces <= foliageHeight-line )
{
  System.out.print( " " );
  spaces = spaces + 1;
}
```

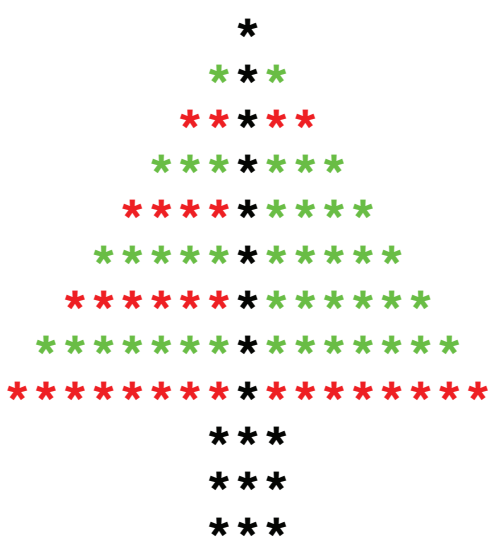
- Note only considering the spaces before the foliage part for each line of the tree

3.6. Looking at each line of the foliage part of the tree

Tree Colour for illustration of algorithm	line	spaces at start of line	stars in the line
	1	8	1
	2	7	3
	3	6	5
	4	5	7
	5	4	9
	6	3	11
	7	2	13
	8	1	15
	9	0	17
	10	7	3
	11	7	3
	12	7	3

3.7. Algorithm to print the stars at `line`

- Need to know the height (line number of the last line of foliage.

Tree Colour for illustration of algorithm	line	stars
	1 2 3 4 5 6 7 8 9 10 11 12	1 3 5 7 9 11 13 15 17 3 3 3

3.7.1. Code to print the foliage at `line`

Pseudo code:

```
print  line * 2 - 1   stars
```

Java code

```
int i = 1;                              // display foliage segment
while ( i <= line*2-1 )
{
  System.out.print( "*" );
  i = i + 1;
}
```

- Only consider the actual foliage part

3.7.2. To print the tree for `line`

To print a picture of the tree at `line`

Pseudo code:

```
print  foliageHeight - line   spaces
print  line * 2 - 1   stars
```

Java code

```
int spaces = 1;                    //Spaces before foliage
while ( spaces <= foliageHeight-line )
{
  System.out.print( " " );
  spaces = spaces + 1;
}
int stars = 1;                     // display foliage segment
while ( stars <= line*2-1 )
{
  System.out.print( "*" );
  stars = stars + 1;
}
```

- The code above could be simplified by using a user written function [this will be covered later]
- If we repeat the above code varying **line** from 1 - 9 then we get the picture shown below

```
        *
       ***
      *****
     *******
    *********
   ***********
  *************
 ***************
*****************
```

3.7.3. Code to print the foliage part of the tree

Pseudo code:

For each **line** of foliage [1 .. 9]

```
print  foliageHeight - line   spaces
print  line * 2 - 1   stars
```

Java code:

```
int foliageHeight = 9;

int line = 1;
while ( line <= foliageHeight )    //For each line of foliage
{

  int spaces = 1;                  //Spaces before foliage
  while ( spaces <= foliageHeight-line )
  {
    System.out.print( " " );
    spaces = spaces + 1;
  }

  int stars = 1;                   // display foliage segment
  while ( stars <= line*2-1 )
  {
    System.out.print( "*" );
    stars = stars + 1;
  }

  System.out.println();            // New line
  line = line + 1;
}
```

```
        *
       ***
      *****
     *******
    *********
   ***********
  *************
 ***************
*****************
```

3.7.4. Code to print the trunk

Tree Max. width at line 9	Line	Spaces in front
*	1	8
***	2	7
*****	3	6
*******	4	5
*********	5	4
***********	6	3
*************	7	2
***************	8	1
*****************	9	0
***	10	7
***	11	7
***	12	7

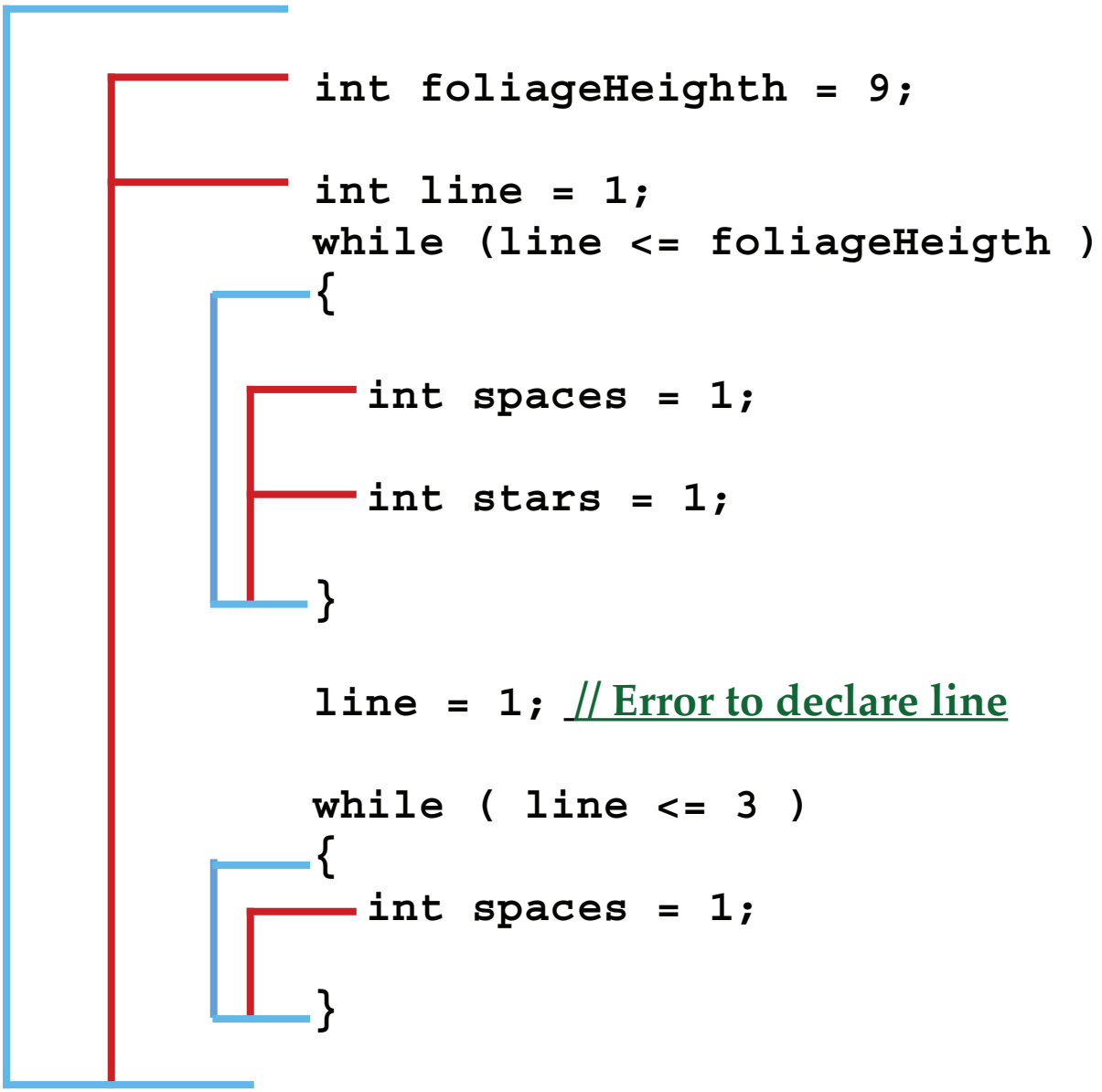
```
line = 1;                              //Display trunk
while ( line <= 3 )
{
 int spaces = 1;                       //Spaces before trunk
 while ( spaces <= foliageHeight-2 )
 {
   System.out.print( " " );
   spaces = spaces + 1;
 }

 System.out.println( "***" );     //Trunk segment

 line = line + 1;
}
```

```
        ***
        ***
        ***
```

3.8. Algorithm - to print a  in Java

```
int foliageHeight = 9;

int line = 1;
while ( line <= foliageHeight )//For each line of foliage
{

 int spaces = 1;                  //Spaces before foliage
 while ( spaces <= foliageHeight-line )
 {
  System.out.print( " " );
  s = s + 1;
 }

 int stars = 1;                   // display foliage segment
 while ( stars <= line*2-1 )
 {
  System.out.print( "*" );
  stars = stars + 1;
 }

 System.out.println();            // New line
 line = line + 1;
}


line = 1;                          //Display trunk
while ( line <= 3 )
{
 int spaces = 1;                   //Spaces before trunk
 while ( spaces <= foliageHeight-2 )
 {
   System.out.print( " " );
   spaces = spaces + 1;
 }

 System.out.println( "***" );     //Trunk segment

 line = line + 1;
}
```

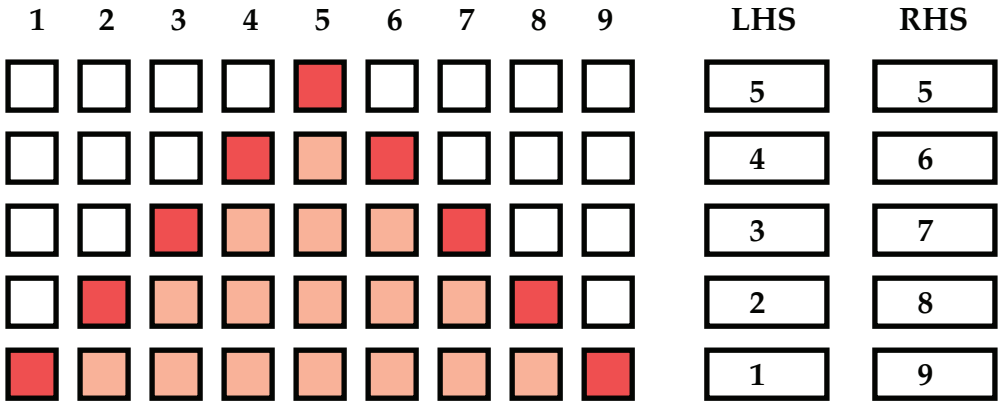
3.9. Visibility of variables in a program



```
int foliageHeighth = 9;

int line = 1;
while (line <= foliageHeigth )
{

    int spaces = 1;

    int stars = 1;

}

line = 1; // Error to declare line

while ( line <= 3 )
{
    int spaces = 1;

}
```

Note You can not have two different variables with the same name visible at the same time within a method

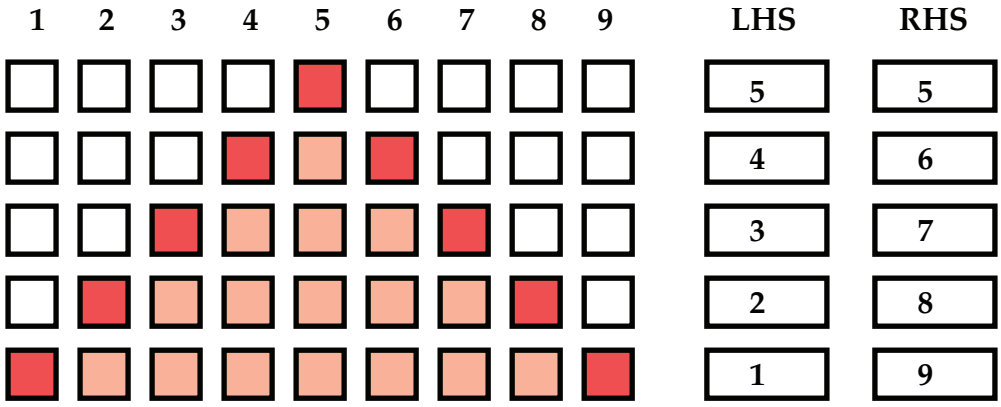
3.10. Another approach



- LHS -> Column index of Left Hand Side 'star'
- RHS -> Column index of Right Hand Side 'star'

```
while LHS is greater than 0
        print LHS - 1 spaces
        print RHS - LHS + 1 stars

        Subtract 1 from LHS, Add 1 to RHS
```

3.11. Yet another approach



- Scan across each row up to and including the column number of the RHS.
  As you scan across the row output a space if the column number is less than the LHS otherwise output a star.

```
while LHS > 0                          // So we will stop
      set column = 1
      while column <= RHS
            if the column < LHS
            print a space
            else
                  print a star
            add 1 to column

      Subtract 1 from LHS, Add 1 to RHS
```

- You can create the effect of an **if** statement out of a **while** loop, but the code is very inelegant. More on **if** soon.

- The **while** loop could be replaced by a **for** loop. More on **for** soon.

3.12. Quick quiz

- Devise an algorithm to print (to any height) a triangle made up of stars. Using the same raster approach as taken to draw the tree.

```
*
**
***
****
*****
******
*******
********
*********
```

The above triangle is printed to a height of 9 units.

Hint
Line 1 has 1 star
Line 2 has 2 stars
Line 3 has 3 stars

3.13. Quick quiz answer

- Devise an algorithm to print (to any height) a triangle made up of stars. Using the same raster approach as taken to draw the tree.

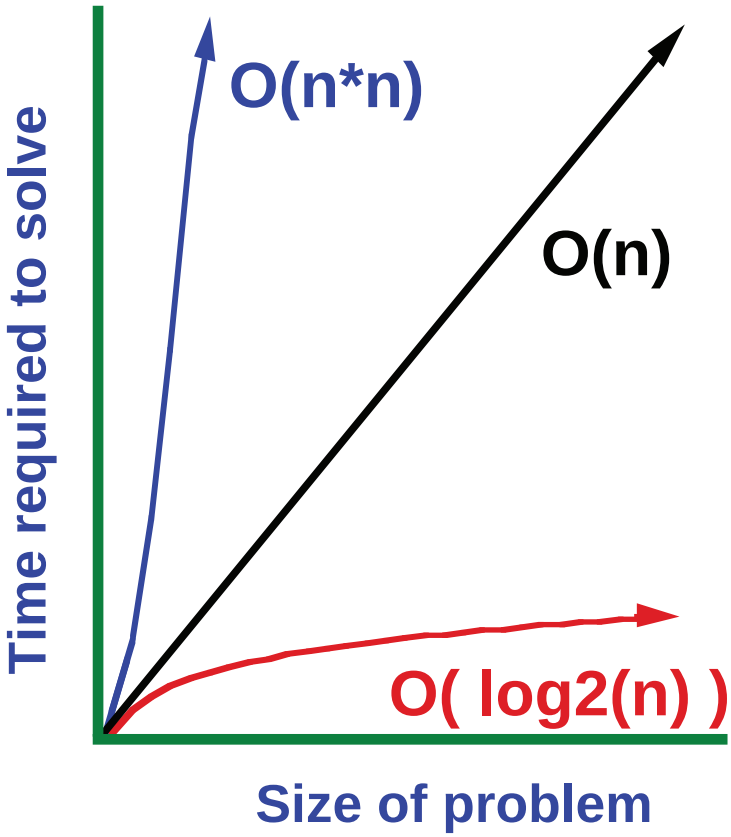
Pseudo code:

```
read in height of triangle
For each line [1 .. height ]
  print  line  stars
  print a new line
```

Java code

```
class Main
{
  public static void main( String args[] )
  {
    System.out.print( "#Input size of triangle: " );
    int height = BIO.getInt();

    int line = 1;
    while ( line <= height )          //For each line of triangle
    {
      int stars = 1;                  // display line of triangle
      while ( stars <= line )
      {
        System.out.print( "*" );
        stars = stars + 1;
      }
      System.out.println();           // New line

      line = line + 1;                //Next line of triangle
    }

  }
}
```

3.14. Effort (time) required to solve a problem



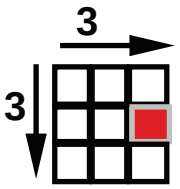
The big O notation gives an estimate of how long it will take to solve a problem with n items.

So O(n) means that if you have **n** items it will take **n** units of time to solve the problem.

Some examples (some what contrived)

O(n) Stamp your name on the front page of the **n** books that you own.

O(n*n) Cut the grass on a square lawn whose side is **n** units long. So a square of side 3 units has 9 squares, a square of side 5 units has 25 squares etc.



Approximately

$O(\log_2(n))$ A book contains **n** pages and on each page is the name of a single person. The names are in alphabetic order with each name on a unique page.

Look up a persons name in the book.

3.15. How long does an algorithm take to run

The steps required to find a name in a book of names, with each name on a individual page

Algorithm 1	Algorithm 2
Step 1: Start at page 1	Step 1: Select all the pages in the book and find the middle page (the current page).
Step 2: If the name is on the current page there then have found the name.	Step 2: If the name is on the current page there then have found the name.
Step 3: Turn over the current page to reveal the next page (forward direction) .	Step 3: If the name collates lower, then select the pages that are to the left of the current page in the selected pages. If the name collates higher, then select the pages that are to the right of the current page in the selected pages. Find the middle page of these selected pages. This is the new current page
Go back to step 2.	Go back to step 2.

Which is the better algorithm?

Each page of the book contains a single name. There are N pages in the book.

Algorithm 1	Algorithm 2
On average takes about: `N/2` comparisons. Because on average you will have to look through half the pages	On average takes about: $\log_2 N$ comparisons. Because each comparison halves the number of pages to look through. Remember the names are in order.

	Comparisons (Approximately)	
Pages in book	Algorithm 1	Algorithm 2
10	5	5
100	50	7
1,000	500	10
10,000	5,000	14
100,000	50,000	16
1,000,000	500,000	20
1,000,000,000	500,000,000	30

3.16. Is the time an algorithm takes important

- public key encryption uses the knowledge that all (publicly) known algorithms (for conventional computers) take a very long time to factor a large number into two primes. Hence difficult to break in a reasonable amount of time.

- When you login to the University system the time to check your username and password has to be reasonably fast otherwise it could take a long time to login.

- Hence if you use a 'poor' algorithm the program you write may be unusable.

  As when a large set of data is processed, the program takes a very long time to deliver the result, even though on a small set of data the processing is almost instantaneous.

3.17. What you should know

- What an algorithm is.
- Be confident in expressing a problem using the while statement.

3.18. What you need to do for next week

- Complete the first practical exercises which will form part of your portfolio by the end of next week.

- Consider what data items need to be checked so that the electricity billing program (From week 2) will produce sensible results.

  Think like a programmer

- Work out how to replace an if statement in Java by a while loop.
  Hint: you only want to go round the loop once, if the condition is true.

4. Selection

To select at run-time which statement(s) are executed

- Selection constructs

```
if ( <Boolean expression> )
    <Statement>

if ( <Boolean expression> )
    <Statement>
else
    <Statement>
```

Statement :
{ [<StatementExpression> ;] }

- Other selection constructs

4.1. Core ideas in an imperative programming language

- Sequence

```
double pricePerKilo = 1.20;

System.out.print( "#Input Kilos of apples: " );
double kilosOfApples = BIO.getDouble();

double cost = pricePerKilo * kilosOfApples;

System.out.print( "Cost of apples is (GBP) " );
System.out.println( cost );
```

- Iteration

```
int count = 1;

while ( count <= 5 )
{
  System.out.println( "Hello" );
  count = count + 1;
}
```

- Selection
- Abstraction

4.2. Recap Boolean expression (Condition)

Symbols used

You must know these

>	**Greater than**
<	**Less than**

then the others logical expressions follow

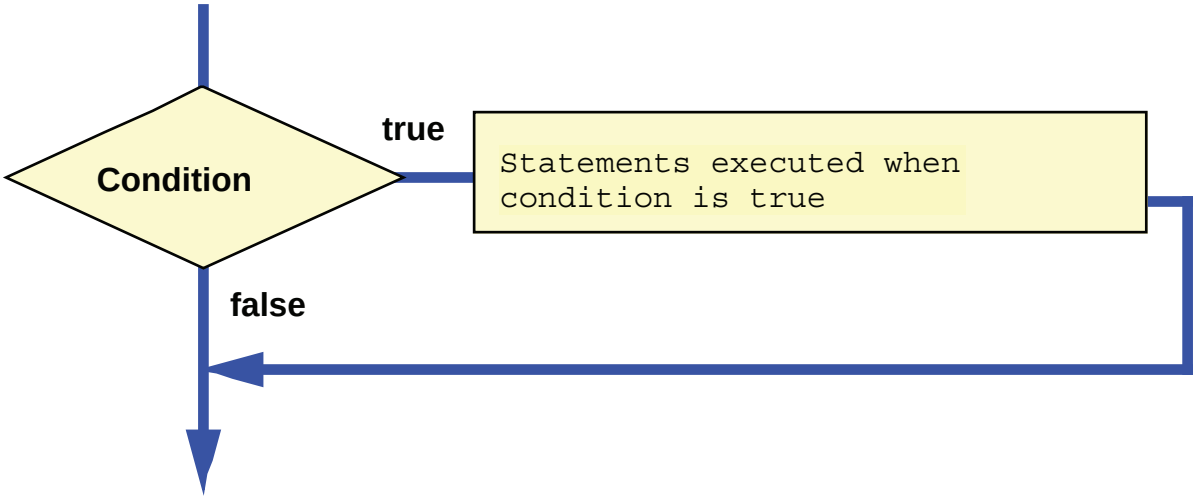
4.3. Symbols used

Symbol	Read as
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
==	Equal
!=	Not equal
!	Not

Boolean expression	Read as
`money < 20`	money is less than 20
`seats > 30`	seats are greater than 30
`rooms <= 10`	rooms are less than or equal to 10
`cars >= 60`	cars are greater than or equal to 60
`seats == 35`	seats equals 35
`rooms != 4`	rooms not equal to 4

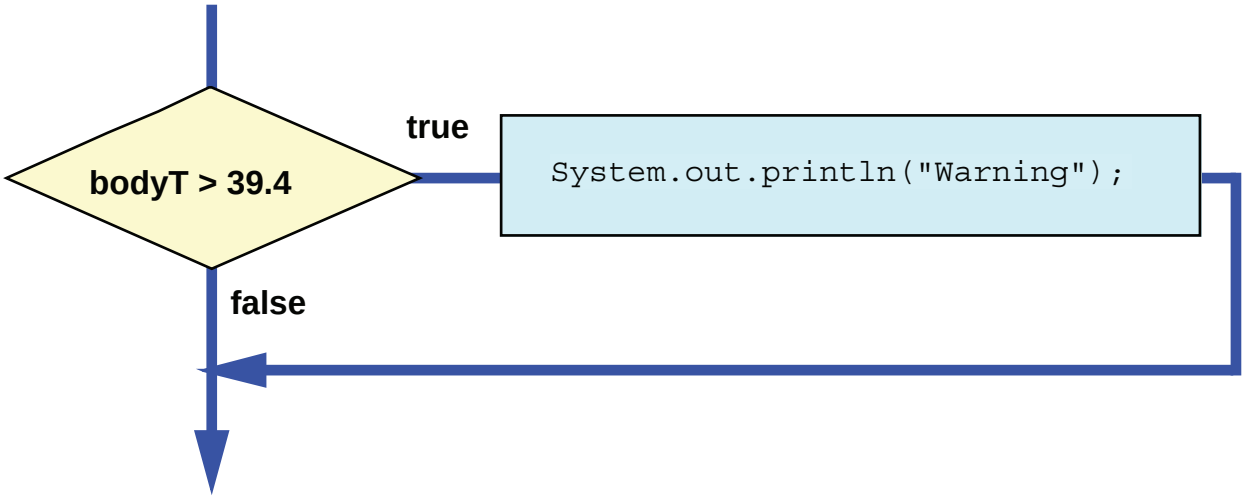
4.4. The `if` statement

4.4.1. flow chart for the if statement



A dog will likely suffer heat stroke if its internal temperature goes over 39.4 degrees Centigrade. The following fragment of code (Perhaps as part of an embedded program in a temperature probe will display a warning if a dogs temperature goes over this value.

4.5. Solution as a flow chart



4.6. Solution as code

```
if ( bodyT > 39.4 )  // For dogs
{
  System.out.println( "Warning" );
}
```

4.7. if then else

```
if ( month != 2 )
  System.out.println("The month is not February");
```

```
if ( month == 2 )
  System.out.println("The month is February");
else
  System.out.println("The month is not February");
```

```
if ( month == 2 )
{
  System.out.println("The month is February");
} else {
  System.out.println("The month is not February");
}
```

Note Without the enclosing { } there is the danger that you might write:

```
if ( month == 2 )
  System.out.println("The month is February");
else
  System.out.println("The month is not February");
  System.out.println("and has 30 or 31 days")
```

Which will print `and has 30 or 31 days` regardless of what the month is.

4.8. Quick quiz

Write an if statement to:

- Print `expensive` if the value contained in `cost` is greater than 10.00

- Print `cheap` if the value contained in `cost` is less than or equal to 5.00 otherwise print `not so cheap`

4.9. Quick quiz answer

- Print `expensive` if the value contained in `cost` is greater than 10.00

```
if ( cost > 10.00 )
  System.out.println("Expensive");
```

```
if ( cost > 10.00 )
{
  System.out.println("Expensive");
}
```

- Print `cheap` if the value contained in `cost` is less than or equal to 5.00 otherwise print `not so cheap`

```
if ( cost <= 5.00 )
  System.out.println("cheap");
else
  System.out.println("Not so cheap");
```

```
if ( cost <= 5.00 )
{
  System.out.println("cheap");
} else {
  System.out.println("Not so cheap");
}
```

4.10. Electricity billing system

A program to print customer bills

Requirements.

- Be able to set the cost for a kilowatt hour of electricity use.
- Be able to print an electricity bill for each customer
- Stop when a customer number of 0 is entered.

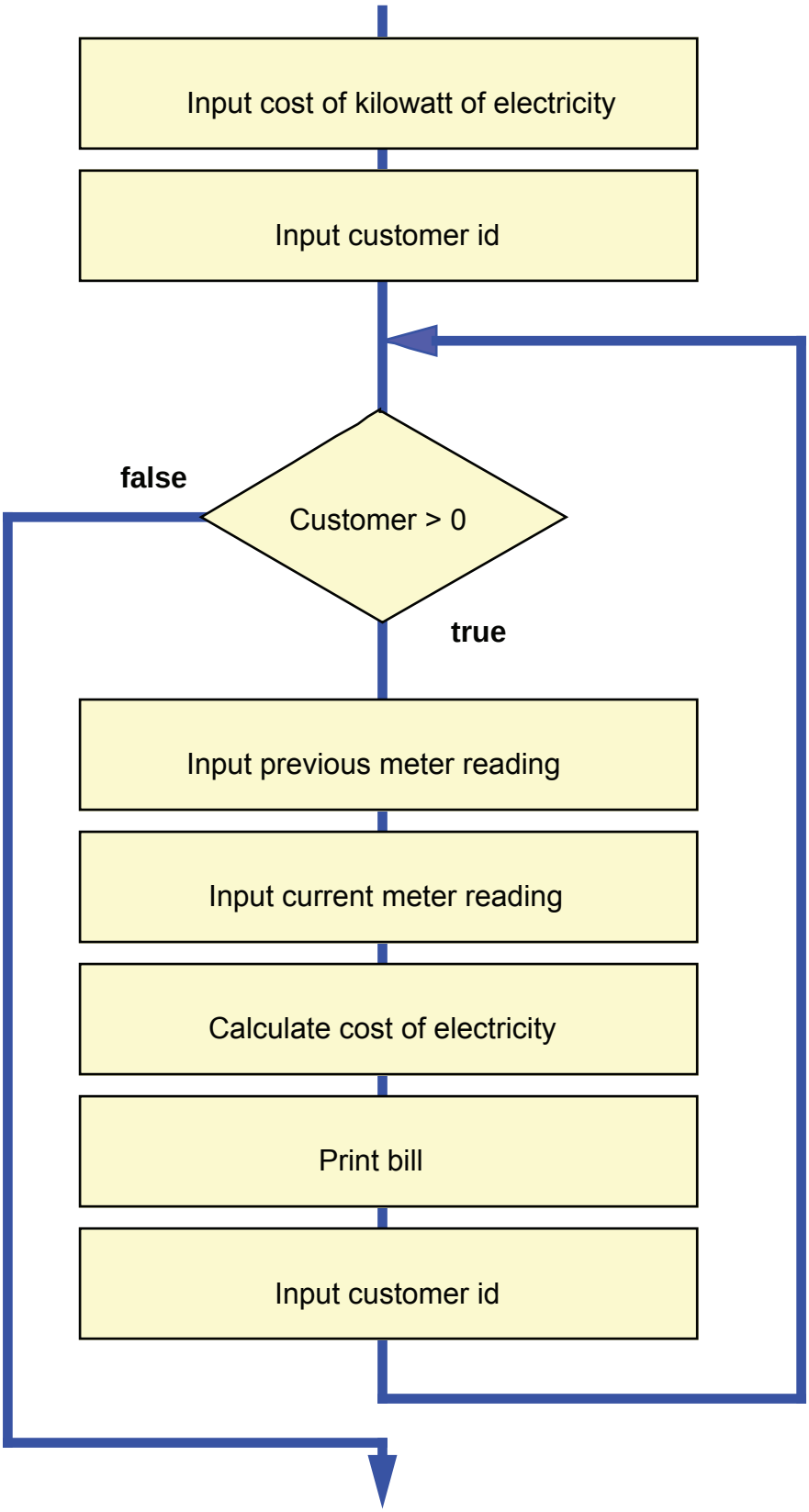
The data provided will be:

The cost in pence per kilowatt hour for electricity.

Then for each customer:

Customer number:	an integer number (7 digits)
Previous usage:	A meter reading (6 digits 1 DP)
Current usage:	A meter reading (6 digits 1 DP)

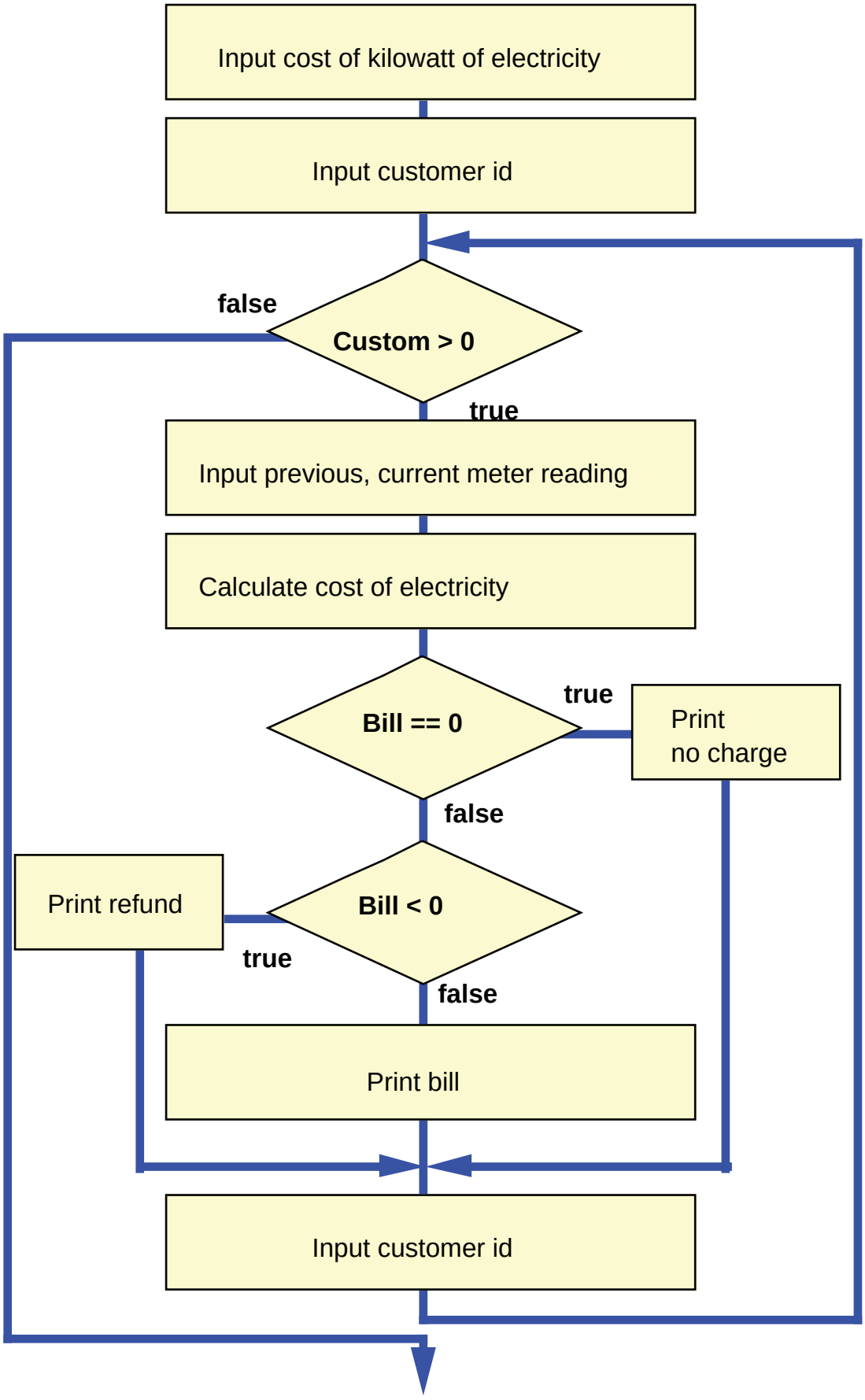
4.11. Electricity billing system - no checks flowchart



4.12. Electricity billing system - As a program no checks

```
class Main
{
  public static void main( String args[] )
  {
    System.out.print("#Cost per kilowatt hour (Pence) : " );
    double pricePerKilowatt = BIO.getDouble();

    System.out.print("#Enter customer id              : " );
    int    customer = BIO.getInt();

    while ( customer > 0 )
    {
      System.out.print("#Enter previous meter reading   : " );
      double previous = BIO.getDouble();
      System.out.print("#Enter current meter reading    : " );
      double current  = BIO.getDouble();

      double used = current - previous;
      double bill = pricePerKilowatt/100.0 * used;

      System.out.printf("Customer [%8d] ", customer);
      System.out.printf(" Meter now %8.1f last %8.1f \n",
                        current, previous );
      System.out.printf("Units used %8.1f  ", used );
      System.out.printf("amount now due (GBP) %8.2f",  bill );

      System.out.println();
      System.out.println();

      System.out.print("#Enter customer id             : " );
      customer = BIO.getInt();
    }
  }
}
```

4.13. Electricity billing system - with checks flowchart



4.14. Electricity billing system - with checks

```
class Main
{
  public static void main( String args[] )
  {
    System.out.print("#Cost per kilowatt hour (pence) : " );
    double pricePerKilowatt = BIO.getDouble();

    System.out.print("#Enter customer id              : " );
    int    customer = BIO.getInt();

    while ( customer > 0 )
    {
      System.out.print("#Enter previous meter reading   : " );
      double previous = BIO.getDouble();
      System.out.print("#Enter current meter reading    : " );
      double current  = BIO.getDouble();

      double used = current - previous;
      double bill = pricePerKilowatt/100.0 * used;

      System.out.printf("Customer [%8d] ", customer);
      if ( bill == 0.0 )
      {
        System.out.printf(" No charge" );
      } else {
        if ( bill < 0.0 )
        {
          System.out.printf(" Refund of (GBP) %8.2f",
                            -bill );
        }
        else
        {
          System.out.printf(" Meter now %8.1f last %8.1f \n",
                            current, previous );
          System.out.printf("Units used %8.1f  ", used );
          System.out.printf("amount now due (GBP) %8.2f",  bill );
        }
      }
      System.out.println();
      System.out.println();

      System.out.print("#Enter customer id              : " );
      customer = BIO.getInt();
    }
  }
}
```

```
Customer [ 9934567]  Meter now  10975.7 last  10567.3
Units used    408.4  amount now due (GBP)    58.12

Customer [ 9900001]  Meter now  52546.9 last  50300.4
Units used   2246.5  amount now due (GBP)   319.68

Customer [ 9957777]  Refund of (GBP)     6.26

Customer [ 9954258]  No charge
```

Q Are there any other checks that should be made?

Observations

Flowchart is getting large and 'complicated'. For a larger program would not fit on a page.

However, useful to show the execution flow in a simple program.

4.14.1. Alternatives to `if`

The code below prints the month which internally is represented by a number in the rage 1-12 held in the variable `month`.

```
if ( month == 1 )
  System.out.print( "January" );
else if ( month == 2 )
  System.out.print( "February" );
else if ( month == 3 )
  System.out.print( "March" );
else System.out.print( "Not January, February or March" );

System.out.println();
```

This code can be 'simplified' by the use of a switch statement.

```
switch ( month )
{
  case 1 :
    System.out.print("January");
    break;
  case 2 :
    System.out.print("February");
    break;
  case 3 :
    System.out.print("March");
    break;
  default:
    System.out.print( "Not January, February or March" );
}
System.out.println();
```

Warning If you do not put in the statement **break**, then the code for the next case label will also be executed.

Note You do not need to put the default case in, but it is usually good programming practice to put it in.

The value switched must be a byte, int, short, char, enumerated type or `String` (JDK 7 and later)

4.14.2. Conditional expression statement

```
<expression> ? <expression> : <expression>
```

To simplify

```
if ( month == 2 )
  System.out.println( "The month is February" );
else
  System.out.println( "The month is not February" );
```

You could write

```
System.out.println( "The month is " +
                    (month == 2 ? "" : "not ") +
                    "February" );
```

4.15. More complex Boolean expressions

```
seats < 20 && day == 2
```
(seats less than 20) and (day equals 2)

```
seats < 20 || day == 2
```
(seats less than 20) or (day equals 2)

4.15.1. && (and)

A	B	A && B
False	False	False
False	True	False
True	False	False
True	True	True

4.15.2. || (or)

A	B	A || B
False	False	False
False	True	True
True	False	True
True	True	True

4.16. Quick quiz

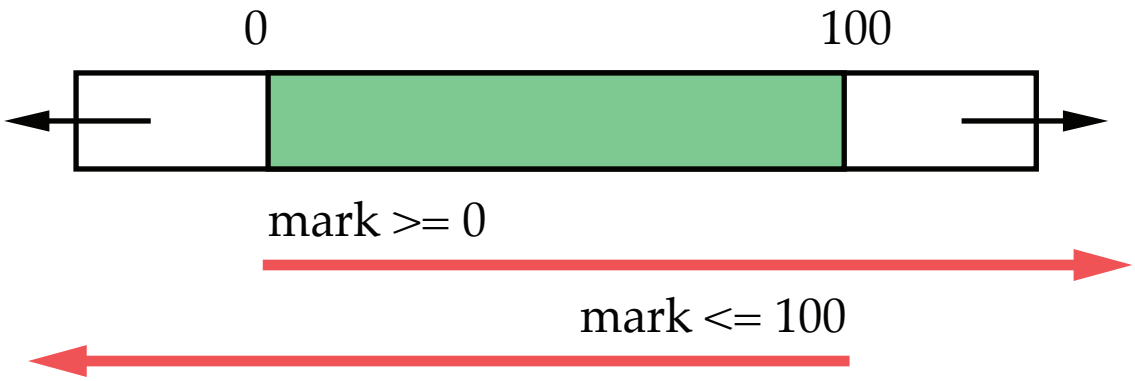
Write an if statement to:

- Print `valid` if the value contained in `mark` is in the range 0 .. 100.

4.17. Quick quiz answers

- Print `valid` if the value contained in `mark` is in the range 0 .. 100.

```
if ( mark >= 0 && mark <= 100 )
{
  System.out.println("valid");
}
```



4.18. Using if

A program to read a list of data consisting of a place name and then the temperature. The list is terminated by the input of a place name STOP.

For each place print if warm or cold and at the end the number of places that are warm and cold.
It is warm if the temperature is greater than or equal to 20.

```
class Main
{
  public static void main( String args[] )
  {
    System.out.print("#Enter place name    : " );
    String  place = BIO.getString();

    int warm = 0;
    int cold = 0;

    while ( ! place.equals("STOP")  )
    {
      System.out.print("#Enter temperature   : " );
      int temp = BIO.getInt();

      if ( temp >= 20 )
      {
        warm = warm + 1;
        System.out.printf( "%-15.15s : warm\n", place );
      } else {
        cold = cold + 1;
        System.out.printf( "%-15.15s : cold\n", place );
      }

      System.out.print("#Enter place name    : " );
      place = BIO.getString();
    }
    System.out.println();
    System.out.printf("Warm places %4d cold places %4d\n",
                      warm, cold );
  }
}
```

```
Liverpool
19
Manchester
20
Preston
18
Blackburn
20
STOP
```

```
#Enter place name     : Liverpool
#Enter temperature    : 19
Liverpool         : cold
#Enter place name     : Manchester
#Enter temperature    : 20
Manchester        : warm
#Enter place name     : Preston
#Enter temperature    : 18
Preston           : cold
#Enter place name     : Blackburn
#Enter temperature    : 20
Blackburn         : warm
#Enter place name     : STOP

Warm places     2 cold places     2
```

```
Liverpool         : cold
Manchester        : warm
Preston           : cold
Blackburn         : warm

Warm places     2 cold places     2
```

4.19. Conventions

- variables always start with a lower case letter

```
double cost = 20.45;

double vatRate = 20.0;
```

- class names always start with a capital

```
class Main
{
  public static void main( String args[] )
  {
    // Some code
  }
}
```

- method names/ functions always start with a lower case letter

```
class Main
{
  public static void main( String args[] )
  {
    // Some code
  }
}
```

4.20. What you should know

- That you need to spend at least 5 hours per week on this module in addition to attending lectures, seminars and tutorials.
- That there will be assessed programming exercises to do throughout the year.
- That learning a computer programming language is not an activity that you can undertake in the last few weeks of the academic year.
- How to write a small Java application
- How to use selection statements in a program
- Be confident what the symbols:

>=	>	!=	==	<	<=

  mean

- Understand the working of the electricity billing program

5. Functional decomposition

- Why.
- Parameter passing

5.1. Core ideas in an imperative programming language

- Sequence

```
double pricePerKilo = 1.20;

System.out.print( "#Input Kilos of apples: " );
double kilosOfApples = BIO.getDouble();

double cost = pricePerKilo * kilosOfApples;

System.out.print( "Cost of apples is (GBP) " );
System.out.println( cost );
```

- Iteration

```
int count = 1;

while ( count <= 5 )
{
  System.out.println( "Hello" );
  count = count + 1;
}
```

- Selection

```
if ( month == 2 )
   System.out.println("The month is February");
```

- Abstraction

- Why split a program into several smaller functions.
  - re-use.

    Possibility of low level code re-use. Functions could be re-used in other Java programs, usually using cut and paste.

    There is a better way the `class` construct
  - Abstraction

    Do not need to know how the code works to use.

    Easier to develop a program by creating it out of several functions.

    Each function has a specific task. The implementation of the function is "hidden" from the user of the code. Thus reducing the overall complexity of the program for the implementer.

5.2. Example

A method (function) to print "`Happy birthday`" several times.

```
class Main
{
  public static void main( String args[] )
  {
    System.out.println("Print Happy birthday 2 times");
    printHappyBirthday( 2 );

    System.out.println("Print Happy birthday 3 times");
    printHappyBirthday( 3 );
  }


  //@param times        How many times to print
  public static void printHappyBirthday( int times )
  {
    int i = 1;
    while ( i <= times )
    {
      System.out.println("Happy birthday");
      i = i + 1;
    }
  }
}
```

```
Print Happy birthday 2 times
Happy birthday
Happy birthday
Print Happy birthday 3 times
Happy birthday
Happy birthday
Happy birthday
```

```
class Main
{
  public static void main( String args[] )
  {
    System.out.println("Print Happy birthday 2 times");
    printHappyBirthday( 2 );

    System.out.println("Print Happy birthday 3 times");
    printHappyBirthday( 3 );
  }


  //@param times        How many times to print
  public static void printHappyBirthday( int times )
  {
    int i = 1;
    while ( i <= times )
    {
      System.out.println("Happy birthday");
      i = i + 1;
    }
  }
}
```

Call of `printHappyBirthday( 2 );`

The actual parameter `2` is assigned to the formal parameter `times`, then the body of the function is executed. Which will print happy birthday 2 times. At the end of the function control is returned to the line after the function call.

Call of `printHappyBirthday( 3 );`

The actual parameter `3` is assigned to the formal parameter `times`, then the body of the function is executed. Which will print happy birthday 3 times. At the end of the function control is returned to the line after the function call.

5.3. Returning a result from a function

An application to print the new price of a product after a price reduction.

The function `newPrice` has two formal parameters

price	The original price
reduction	The percentage reduction to be applied

and returns as its result the new price after the reduction has been applied.

```
class Main
{
  public static void main( String args[] )
  {
    System.out.println("Price      % Off  New price");
    int reduction = 10;
    while ( reduction <= 90 )
    {
      System.out.printf( "%-8.2f %3d      %-8.2f\n",
                         100.0, reduction,
                         newPrice( 100.0, reduction )
                       );
      reduction = reduction + 10;
    }
  }

  //@param price        original price of item
  //@param reduction    percentage reduction
  //@return             reduced new price
  public static double newPrice( double price, double reduction )
  {
    return price * (100.0 - reduction ) / 100.0;
  }
}
```

```
Price      % Off  New price
100.00     10     90.00
100.00     20     80.00
100.00     30     70.00
100.00     40     60.00
100.00     50     50.00
100.00     60     40.00
100.00     70     30.00
100.00     80     20.00
100.00     90     10.00
```

Terminology

Actual parameter

The variable passed to the method (function)

Formal parameter

The name of the parameter in the methods signature

Method (function) Signature

The name of the method (function) plus the names of any formal parameters and there type, plus the type of any returned value.

Call by value

The contents of the actual parameter are copied to the formal parameter.

5.4. Print a triangle re-visited

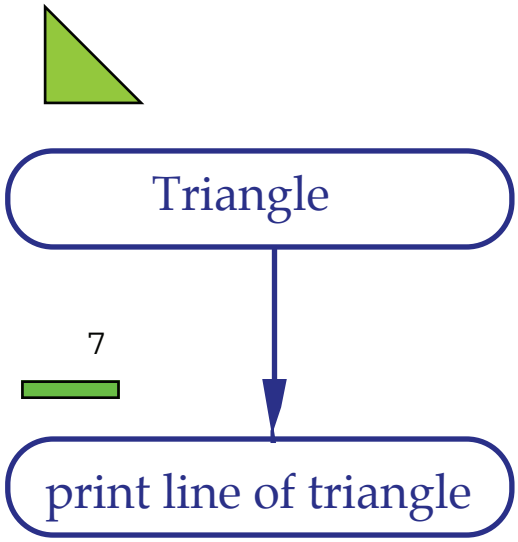
- An algorithm to print (to any height) a picture of a triangle using a raster graphics approach. Triangle of height 7 shown below

```
*
**
***
****
*****
******
*******
```

```
class Main
{
  public static void main( String args[] )
  {
    System.out.print( "#Input size of triangle: " );
    int height = BIO.getInt();

    int line = 1;
    while ( line <= height )          //For each line of triangle

      int stars = 1;                  // display line of triangle
      while ( stars <= line )
      {
        System.out.print( "*" );
        stars = stars + 1;
      }
      System.out.println();           // New line

      line = line + 1;                //Next line of triangle
    }

  }
}
```

5.5. Print a triangle, using a function

```
*
**
***
****
*****
******
*******
```



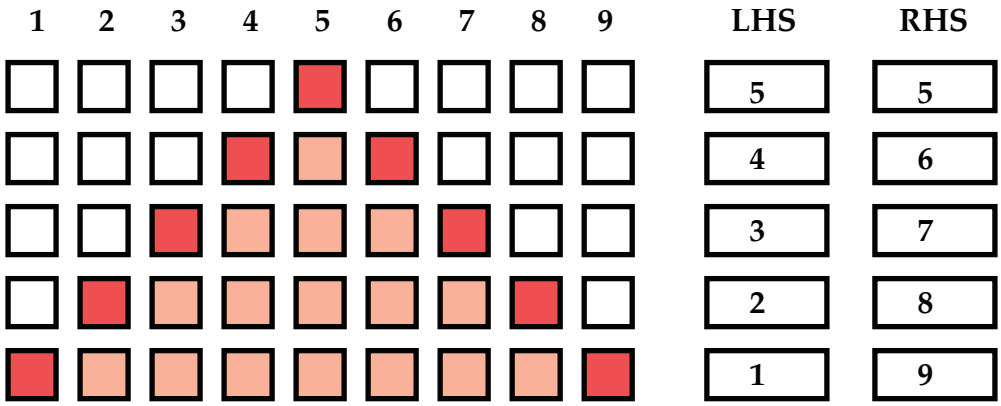
5.6. Java code to print a triangle, using a function

```
class Main
{
  public static void main( String args[] )
  {
    System.out.print( "#Input size of triangle: " );
    int height = BIO.getInt();

    int line = 1;
    while ( line <= height )          //For each line of triangle
    {
      printLineOfTriangle( line );
      line = line + 1;                //Next line
    }

  }

  private static void printLineOfTriangle( int numberOfStars )
  {
    int stars = 1;                    // display line of triangle
    while ( stars <= numberOfStars )
    {
      System.out.print( "*" );
      stars = stars + 1;
    }
    System.out.println();             // New line
  }
}
```

```
*
**
***
****
*****
******
*******
```

5.7. Print a tree, using functions

```
   *
  ***
 *****
*******
*********
  ***
  ***
  ***
```



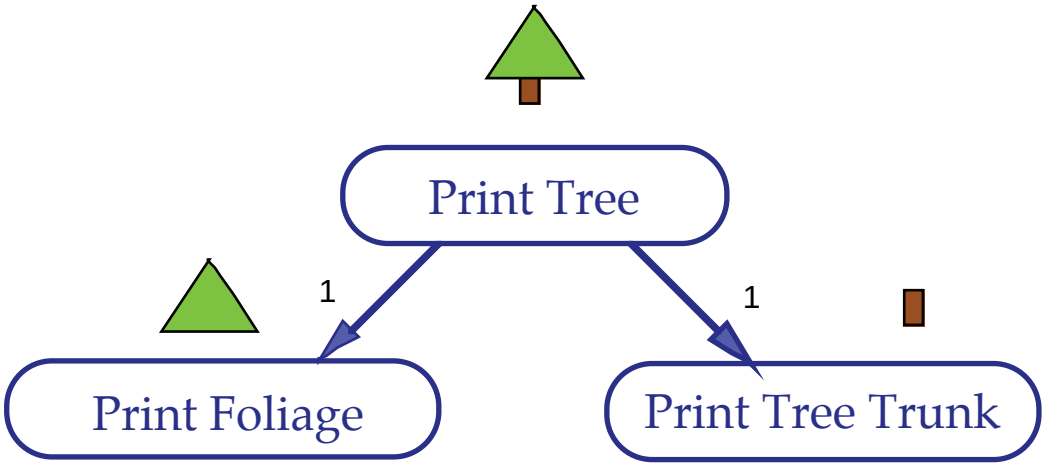
Algorithm

LHS -> Index of Left Hand Side 'star'

RHS -> Index of Right Hand Side 'star'

```
while LHS is greater than 0
        set column = 1
        while column <= RHS
                if the column is  less than LHS
                        print a space
                else
                        print a star
                add 1 to column

        Subtract 1 from LHS, Add 1 to RHS

Print tree trunk
```

5.8. Functional decomposition



5.9. Shortcuts for adding/ subtracting 1

Long form	Short form
`count = count + 1;`	`count++;`
`count = count - 1;`	`count--;`

5.10. functions (static methods in Java)

```
class Main
{
  public static void main( String args[] )
  {
    printTree( 9 );
  }

  /*
   * Display all of tree
   */
  public static void printTree( int foliageHeight )
  {
    printFoliage( foliageHeight );
    printTrunk( foliageHeight, 3 );
  }
```

```
class Main
{
  public static void main( String args[] )
  {
    printTree( 9 );
  }

  public static void printTree( int height )
  {
    printFoliage( height );
    printTrunk( height, 3 );
  }
```

Actual parameter

Formal parameter

- Actual parameter
  Value that is passed to the function
- Formal parameter
  Name of the 'variable' inside the function that the value is passed to
- On a function call the value in the actual parameter is assigned to the formal parameter.

```
class Main
{
  public static void main( String args[] )
  {
    printTree( 6 );
    printTree( 9 );
    printTree( 15 );
  }

  /*
   * Display all of tree
   */
  public static void printTree( int height )
  {
    printFoliage( height );
    printTrunk( height, 3 );
  }

  /*
   * Display foliage of tree
   */
  public static void printFoliage( int foliageHeight )
  {
    int lhs = foliageHeight;
    int rhs = foliageHeight;

    while ( lhs > 0 )                    //For each line of foliage
    {
      int col = 1;                       //start at col 1
      while ( col <= rhs )
      {
        if ( col < lhs )
          System.out.print( " " );       //Spaces
        else
          System.out.print( "*" );       //Stars
        col++;
      }
      lhs--; rhs++;
      System.out.println();
    }
  }
```

```
/*
 * Display trunk of tree
 */
public static void printTrunk( int foliageHeight,
                               int trunkHeight )
{
  int lineOfTrunk = 1;
  while ( lineOfTrunk <= trunkHeight )
  {
    int stars = 1;                    //Spaces before trunk segment
    while ( stars < foliageHeight-1 )
    {
      System.out.print( " " );
      stars++;
    }

    System.out.println( "***" );     //Trunk segment

    lineOfTrunk++;
  }
}
```

5.11. Functional decomposition

- Now the code is easier to read/ modify/ maintain
- However, the code is not always easily re-usable in another program.

  As the data used by the function may not always be able to be contained within the function.

  In essence a group of functions working together may wish to share common data.

  For this shared data and the methods that use it to be bound together the **class** construct would need to be used.

5.12. What you should know

- That you need to spend at least 5 hours per week on this module in addition to attending lectures, seminars and tutorials.
- What an algorithm is.
- Be confident in expressing a problem using the while and if statements.
- Understand functional decomposition.
- Know what a formal and actual parameter are.
- Understand what happens when you call a function.

6. Java system

- The Java system
  `javac`
  `java`
- More on primitive types:
  `int`, `double` etc.
- All about the type `String` a reference type: class

6.1. Core ideas in an imperative programming language

- Sequence

```
double pricePerKilo = 1.20;

System.out.print( "#Input Kilos of apples: " );
double kilosOfApples = BIO.getDouble();

double cost = pricePerKilo * kilosOfApples;

System.out.print( "Cost of apples is (GBP) " );
System.out.println( cost );
```

- Iteration

```
int count = 1;

while ( count <= 5 )
{
  System.out.println( "Hello" );
  count = count + 1;
}
```

- Selection

```
if ( month == 2 )
   System.out.println("The month is February");
```

- Abstraction

```
public static void main( String args[] )
{
  printTree( 9 );
}

public static void printTree( int maxFoliageHeight )
{
  printFoliage( maxFoliageHeight );
  printTreeTrunk( maxFoliageHeight );
}
```

You can write any program by just using the above type of constructs plus arrays.

? So is this nearly the end of the course

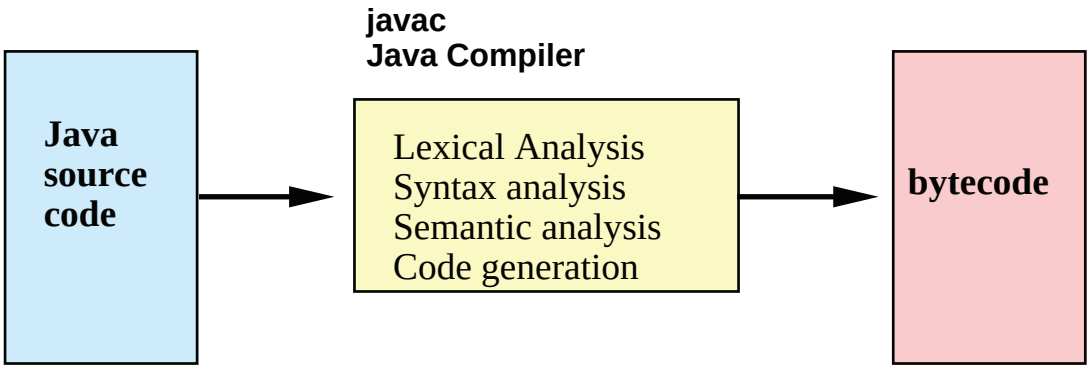
6.2. The Java Compiler

Converts a source Java program into a series of artificial instructions called the bytecode. Individual bytecode instructions are very simple, for example:

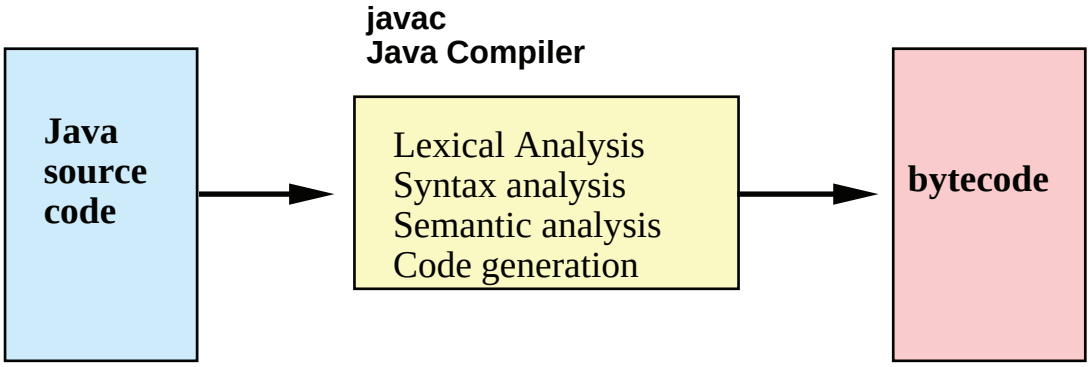
Instruction	Explanation
`istore arg`	Pop the integer off the stack and store into location arg
`iload arg`	Push onto the stack contents of location arg
`imul`	Replace top 2 integer values on the stack by their product
`iadd`	Replace top 2 integer values on the stack by their sum

Each line of a Java program will normally be converted into several bytecode instructions.

6.3. Overview of the Java compiler



What language is the Java compiler written in?



- Lexical analysis
  Breaks the program into tokens
  `class`, `Main`, `{`, `public`, `static`, `void`, `main`, `(`, `String`, `args`, `[`, `]`, `)`
- Syntax analysis

  Check the tokens occur in the correct order

  Syntax error

```
if (vatRate==20  category = "Full";
```

- Semantic analysis
  Check the semantics of the program are correct
  Not checking the program is correct

  Semantic error

```
int category;
if (vatRate== 20) category = "Full";
```

- Code generation
  Generate the bytecode to represent the program

  Still the possibility of logic errors (program does not do what was intended) in program.

```
int number = 2;
if (number%2 == 0)
  System.out.print("Number is odd");
```

6.4. Types of error in a program

- When compiling the program

Syntax	`// Syntactical rules broken`
	`//  missing ;`
	
	`int a = 2`
	`int b = 2;`

Semantic	`// Syntax OK, but meaning wrong`
	`//  color is not declared`
	
	`int colour = 2;`
	
	`color = 2;`

- When running the program

Logical	`// Program produces wrong result`
	
	`String result;`
	
	`if ( mark > 40 )`
	`  result = "Fail";`

run-time error	`// Program fails with a run-time`
	`// error-message`
	`// Exception`
	`//  String index out of range:10`
	
	`String m = "Hello";`
	
	`char c = m.charAt(10);`

6.5. Quick quiz

What type of error is in each of the following programs:

```
class Main
{
  public static void main(String args[])
  {
    int a = "I like Java";
  }
}
```

```
class Main
{
  public static void main(String args[])
  {
    String message = "I like Java"
  }
}
```

```
class Main
{
  public static void main(String args[])
  {
    int value = 0;
    int a = 1 / value;
  }
}
```

6.6. Quick quiz answer

What type of error is in each of the following programs:

```
class Main
{
  public static void main(String args[])
  {
    int a = "I like Java"; // semantic
  }
}
```

```
class Main
{
  public static void main(String args[])
  {
    // syntax missing ;
    String message = "I like Java"
  }
}
```

```
class Main
{
  public static void main(String args[])
  {
    int value = 0;
    // run-time Division by zero
    int a = 1 / value;
  }
}
```

6.7. Bytecode

Java Code	Bytecode

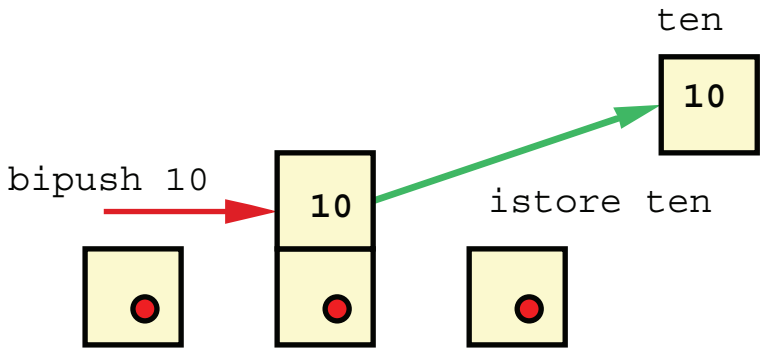
`class Main`
`{`
`  public static void main( String args[] )`
`  {`
`  int ten  = 10;`
	`bipush  10`
	`istore  ten`

`  int twenty = 20;`	`bipush  20` `istore  twenty`
`  int thirty = 30;`	`bipush  30` `istore  thirty`

`  int result = ten + twenty * thirty;`	`iload  ten`
	`iload  twenty`
	`iload  thirty`
	`imul`
	`iadd`
	`istore  result`

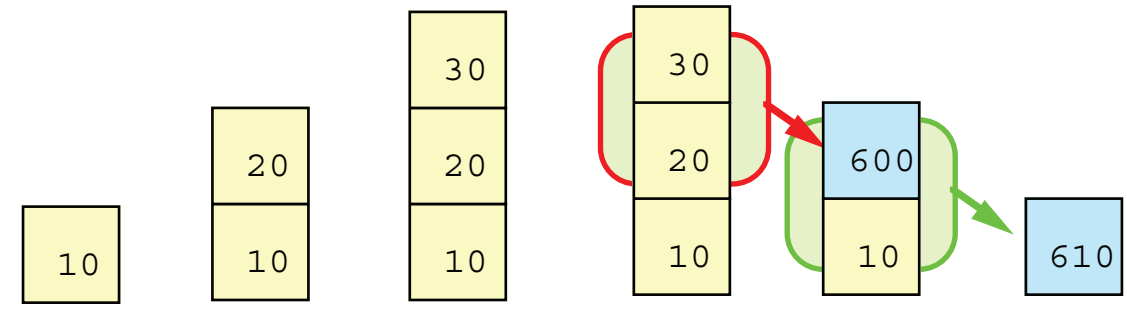
`  }` `}`	`return`

```
opCode    operand
bipush    10        // Put on the stack the integer 10
istore    ten       // Take the top item off the stack
                    //  and put in the variable ten
```



```
opCode      operand
iload       ten         // Put on the stack the contents of ten
iload       twenty      // Put on the stack the contents of twenty
iload       thirty      // Put on the stack the contents of thirty
imul                    // Multiply the top two items on the stack
                        //   replacing with the result
iadd                    // Add the top two items on the stack
                        //   replacing with the result
istore      result      // Take the top item off the stack and store in
                        //  the memory location result
```

Execution of `result = ten + twenty * thirty;`



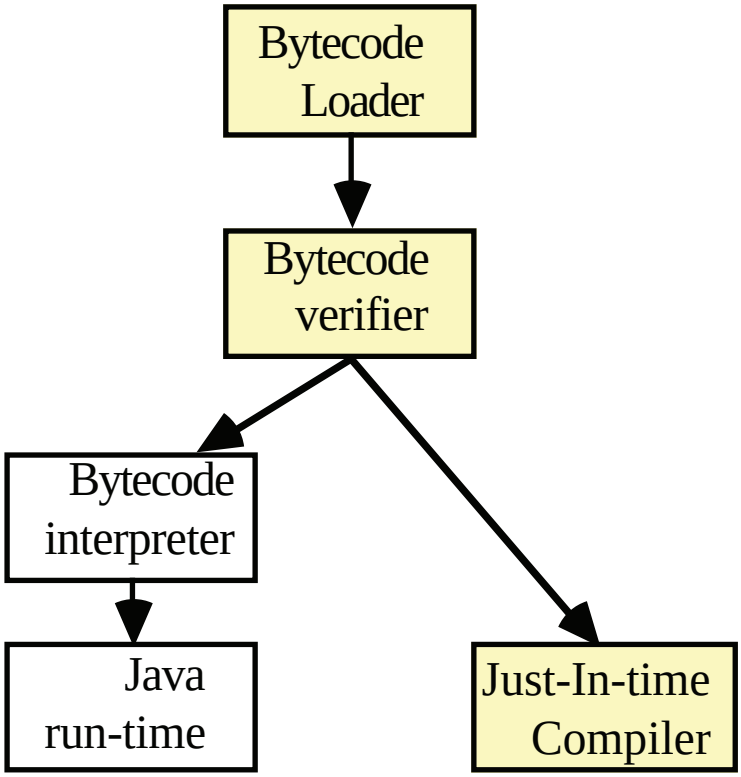
`iload 10` `iload 20` `iload 30` `imul` `iadd` `istore result`

6.7.1. The JVM (Java Virtual Machine)

This is a program that 'executes' the byte code instructions and hence realises the Java program.

As the bytecode can be run on any machine that has a bytecode interpreter. Java uses the slogan:
Compile once run anywhere.

6.8. Running a Java program



6.9. Scala

Scala is a programming language that also compiles in to byte code and hence can be executed using the JVM.

6.10. Java bytecode interpreter (JVM)

JVM - Java Virtual Machine

```
// Illustrative some declarations are missing

class Main
{
  public static void main()
  {
    while ( true )
    {
      //Fetch next instruction : opCode operand

      //Execute instruction

      switch ( opCode )
      {
        case bipush :
          //Push operand onto the top of the stack
          break;
        case istore :
          //Take top item of the stack and store
          // into the operand at memory location ...
          break;

        //etc.

        default :
          // Unrecognised instruction
      }
    }
  }
}
```

- Stylised possible code for a JVM
  Different implementations of the JVM written in a variety of languages: C, C++, Java.

  So how can a JVM be written in Java?

6.11. What is important

- Phases of the compiler
  Discover of any syntax errors
  Discovery of any semantic errors
  Bytecode generation

- The bytecode as a machine independent representation of a Java program.

  Hence the bytecode from a Java program can be run on any machine that has a Java bytecode interpreter (The Java Virtual Machine [JVM])

  Write once run anywhere

- Idea of Fetch next instruction cycle
  Repeat
  Fetch bytecode instruction
  Execute bytecode instruction

6.12. JIT Compiler (Just In Time)

- Before executing the bytecode instructions converts them to real machine code instructions and then executes this code. The next time round the loop the compiled code is executed.

- Will be fast executing: as code executed more than once will be run as real machine code and not require translation.

- Over simplification
  10% of code is executed 90% of the time
  90% of code is executed 10% of the time

  Write code that works and is easy to follow, worry about efficiency later.

- The Java HotSpot JVM targets the frequently executed instructions to convert into real machine code.

6.13. A Java program running in a web browser

On the web visiting a web page may result in a Java applet being run.

See: *JVM, Java bytecode, portability, applet*

6.14. Primitive types in Java

There are several primitive types in Java. However, it is very likely that you will only use a few.

Integer data types	`byte, short, int, long`
Real data types (Have decimal places)	`float, double`
Character data type	`char`
Boolean data type	`boolean`

6.14.1. Example declarations

```
int     classSize = 50;
double  payRate   = 10.00;
char    grade     = 'A'
boolean sunnyDay  = true;

boolean hot  = temperature > 30;
```

6.15. Memory

- 8 bits = 1 byte, $^1/_2$ byte is a nibble
- Typical size of RAM memory
  4 GiB = $4 * 2^{30}$ = 4,294,967,296 bytes
- Typical size of rotating disk drive
  2TB = $2 * 10^{12}$ = 2,000,000,000,000 bytes
- 2^{10} (1024) is approximately equal to 10^3 (1000)

6.16. Technical details: Types

- Exact representation

Type	Range of values that an instance of the type can hold	Size bits	size bytes
byte	-128 ... +127	8	1
short	-32768 ... +32767	16	2
int	-2147483648 ... +2147483647	32	4
long	-9223372036854775808 ... +9223372036854775807	64	8

- `float` Held to approx. 7 decimal digits
- `double` Held to approx. 16 decimal digits

Type	Range of values that can be held (approximately)	Size bytes
float	$+/- \quad 1.4 * 10^{-45} \quad ... \quad 3.4 * 10^{38}$	4
double	$+/- \quad 4.9 * 10^{-324} \quad ... \quad 1.8 * 10^{308}$	8

- Atoms in the observable universe $\sim 10^{80}$

Type	Comment	Representation	Initial value
boolean	A boolean value	1 bit	false
byte	A whole number	8 bit 2's complement no.	0
short	A whole number	16 bit 2's complement no.	0
char	A character	16 bit Unicode character	'\u0000'
int	A whole number	32 bit 2's complement no.	0
float	A number with decimal paces	32 bit IEEE 754	0.0
long	A whole number	64 bit 2's complement no.	0
double	A number with decimal places	64 bit IEEE 754	0.0

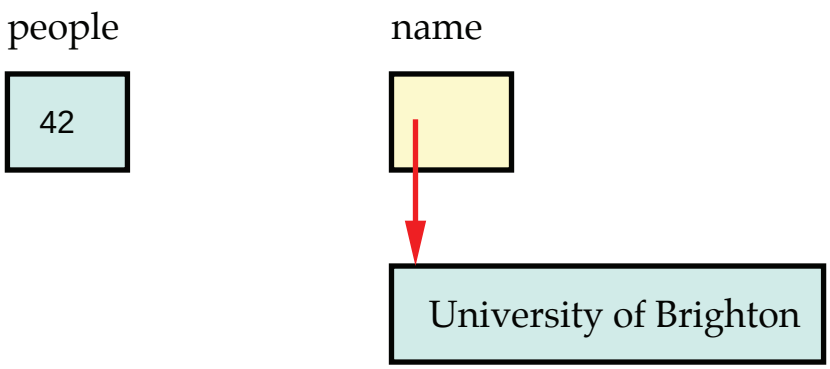
It is a requirement of Java that all implementations use these definitions. So for example, the size of an `int` will be the same for any Java program run on any machine.

6.17. Reference type: `String (A Class)`

Java has as part of its language a type `String` that is used to declare variables that can hold a text message.

It is called a reference type, as an instance of a `String` holds a reference to the characters of the text message. Recall that an instance of a primitive type holds the actual value.

```
int people = 42;
String name = "University of Brighton";
```



This distinction has many consequences that we will look at later. It also means that converting a program written in Java to another language is not always easy. However, it does in many cases make a programming solution simpler.

6.17.1. Concatenating (joining) instance of a `String`

```
class Main
{
  public static void main( String args[] )
  {
    String language   = "Java";
    String aboutCI101 = "Module CI101 uses " + language;

    System.out.println( aboutCI101 );
  }
}
```

```
Module CI101 uses Java
```

6.17.2. Conversion of variable to a `String`

```
class Main
{
  public static void main( String args[] )
  {
    int    year   = 2016;
    String cohort = "Class of " + year;

    System.out.println( cohort );
  }
}
```

```
class of 2016
```

6.17.3. "\n" in a String

```
class Main
{
  public static void main( String args[] )
  {
    String address = "University of Brighton\nBrighton";

    address = address + "\nEast Sussex";

    System.out.println( address );
  }
}
```

```
University of Brighton
Brighton
East Sussex
```

Sequence in a string or char	Is replaced by
\t	A tab character
\n	A new line character
\"	A double quote character
\'	A single quote character
\\	The \ character

6.17.4. Comparing Strings

```
String name = "me";
```

```
if ( name.equals( "me" ) )
{
  System.out.println("Thats me" );
}
```

```
if ( ! name.equals( "me" ) )
{
  System.out.println("Thats not me" );
}
```

- Do not use == to compare strings
  The code does not do what you expect it to do.

```
if ( name == "me" )
{
  System.out.println("Will not always work" );
}
```

6.18. What you should know

- That you need to spend at least 5 hours per week on this module in addition to attending lectures, seminars and tutorials.
- That there will be assessed programming exercises to do throughout the year.
- That learning a computer programming language is not an activity that you can undertake in the last few weeks of the academic year.
- The difference between syntax and semantic errors.
- The concept of the fetch next instruction cycle.
- Other primitive data types.
- Simple uses of the reference type `String` (a class).

7. Strings - and OO ideas

- Shortcuts / Idioms
- Manipulating Strings
- `String` methods
- Introduction to sorting

7.1. Core ideas in an imperative programming language

- Sequence

```
double pricePerKilo = 1.20;

System.out.print( "#Input Kilos of apples: " );
double kilosOfApples = BIO.getDouble();

double cost = pricePerKilo * kilosOfApples;

System.out.print( "Cost of apples is (GBP) " );
System.out.println( cost );
```

- Iteration

```
int count = 1;

while ( count <= 5 )
{
  System.out.println( "Hello" );
  count = count + 1;
}
```

- Selection

```
if ( month == 2 )
   System.out.println("The month is February");
```

- Abstraction

```
public static void main( String args[] )
{
  printTree( 9 );
}

public static void printTree( int maxFoliageHeight )
{
  printFoliage( maxFoliageHeight );
  printTreeTrunk( maxFoliageHeight );
}
```

7.2. Arithmetic shortcuts

History
Used originally in the language C

Commonly used

Long form	Short form
`count = count + 1;`	`count++;`
`count = count - 1;`	`count--;`

Not so commonly used

Long form	Short form
`count = count + 2;`	`count += 2;`
`count = count - 3;`	`count -= 3;`
`count = count * 4;`	`count *= 4;`
`count = count / 5;`	`count /= 5;`
`count = count % 6;`	`count %= 6;`

Not so commonly used

`++` `--` in an expression

Expression	Original contents of count	Contents of a after assignment	Contents of count after assignment
`a = ++count;`	10	11	11
`a = count++;`	10	10	11
`a = --count;`	10	9	9
`a = count--;`	10	10	9

7.3. Specialised form of `while` - `for`

```
int count = 1;                    //Set count to 1
while ( count <= 5 )              //While count less than or equal 5
{
  System.out.println( "Hello" );// Print Hello
  count++;                        // Add 1 to count
}
```

7.4. Which is 'equivalent' to

```
for ( int count=1; count <= 5; count++ )
{
  System.out.println( "Hello" );// Print Hello
}
```

However, the scope of `count` is limited to the `for` statement, so is actually equivalent to the `while` statement:

```
{
  int count = 1;                  //Set count to 1
  while ( count <= 5 )            //While count <= 5
  {
    System.out.println( "Hello" );// Print Hello
    count++;                      // Add 1 to count
  }
}
```

7.5. Use of a for loop

A Java application to write Hello 5 times

```
class Main
{
  public static void main( String args[] )
  {
    for ( int count=0; count < 5; count++ )
    {
      System.out.printf( "%3d Hello\n" , count );
    }

  }
}
```

```
  0 Hello
  1 Hello
  2 Hello
  3 Hello
  4 Hello
```

7.6. An endless loop

```
for ( ; ; ) ;
```

- You are allowed to miss out all or 2 or 1 of the elements of a **`for`** statement, or even the body.

7.7. `do while` loop

If you know you are to execute some code at least once, then you can use a `do while` loop

```
do {
  // Some code
} while ( condition );
```

However, be warned, there are many cases when you actually wish to do nothing for certain sets of data.

- Print a square of side `N` where `N >= 0`.
  A 0 sided square is nothing.
- Process student results for individual modules.
  There may be cases when in one year no students take a module.

7.8. break

Used to break out of a loop. Considered by some to be bad programming practice.

```
class Main
{
  public static void main( String args[] )
  {
    int warm = 0;
    int cold = 0;

    while ( true )
    {
      System.out.print("#Enter place name    : " );
      String  place = BIO.getString();

      if ( place.equals( "STOP" ) )
        break;                          //Break out of loop -->

      System.out.print("#Enter temperature   : " );
      int temp = BIO.getInt();

      if ( temp >= 20 )
      {
        warm = warm + 1;
        System.out.printf( "%-15.15s : warm\n", place );
      } else {
        cold = cold + 1;
        System.out.printf( "%-15.15s : cold\n", place );
      }
    }

    System.out.println();
    System.out.printf("Warm places %4d cold places %4d\n",
                      warm, cold );
  }
}
```

```
#Enter place name    : Liverpool
#Enter temperature   : 19
Liverpool       : cold
#Enter place name    : STOP

Warm places    0 cold places    1
```

7.9. Introduction to `String` methods

- Overview
  We can 'ask' an instance of a `String` about itself

```
String name = "Brighton";
```

the expression

```
name.length()
```

Will send the message `length` to `name` an instance of a `String`. This will result in the number of characters (or length) of the `String` being returned. For example:

```
String name = "Brighton";
System.out.println( name.length() )
```

will print

```
8
```

There is a lot more to this, but this will be for later lectures and later modules.

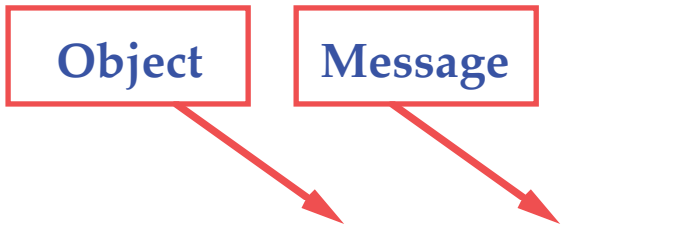
- An instance of a `String` is immutable
  Hence you can not change an instance of a `String`. You can obtain a new string as a result of sending a message to a `String`, but not to change the string.

- Terminology

```
String name = "Brighton";
```

```
int length = name.length()
```

Which is read as: send the message `length` to the object `name` returning an `int` value.



An instance of the reference type class is an Object

- Returning a string all in lower case

```
System.out.print("CEM".toLowerCase());
```

```
cem
```

7.10. `String` methods

Method	Behaviour
`s.charAt( pos )`	Returns the character at position `pos` in the String `s`. The first character is at position `0`.
`s.equals( str )`	Returns true if `s` equals the String `str`.
`s.length()`	Returns the length (number of characters in the String).
`s.trim()`	Returns a new string which has all leading and trailing white space characters in `s` removed.

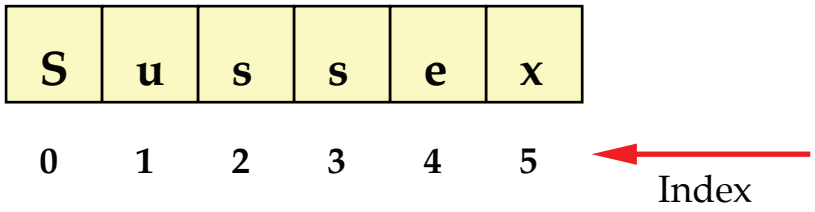
`String.format( fmt, args )`	Like `printf`, but returns a String containing the result. Note format is a static method. String result = String.format("Answer = %d", answer);

Method	Behaviour
s.charAt(pos)	Returns the character at position pos in the String s. The first character is at position 0.
s.compareTo(b)	Compares s lexicographically using the Unicode character set. For s.compareTo(b) the value returned is an int that is: = 0 if s == b. < 0 if s < b. > 0 if s > b.
s.equals(str)	Returns true if s equals the String str.
String.format(fmt, args)	Like printf, but returns a String containing the result. Note format is a static method. String result = String.format("Answer = %d", answer);
s.IndexOf(ch)	Returns the index of the first character ch in the String s. If not found return -1.
s.IndexOf(str)	Returns the index of the first occurrence of the first character of the substring str in String s. If the substring str is not found return -1.
s.length()	Returns the number of characters in the String.
s.replace(o, n)	Returns a new string with all occurrences of character o in the String s replaced with the character n.
s.substring(from)	Returns a substring of s that extends from the character at position from to the end of the string. The string starts at position 0.
s.substring(from, to)	Returns a substring of s that extends from the character at position from to the character at position to-1.
s.toCharArray()	Returns a char array of the characters in the string s.
s.toLowerCase()	Returns a new string which has all the characters of s converted to lower case.
s.toUpperCase()	Returns a new string which has all the characters of s converted to upper case.
s.trim()	Returns a new string which has all leading and trailing white space characters in s removed.

Note: in a string the first character is at position 0, the second character at position 1 etc.
The methods do not change `s`.
Do not use == to compare strings

7.11. Looking at individual characters in a string

```
String place   = "Sussex";
```



- The first character in the string is at index 0.

  There are very good reasons why the first character is at index `0`.

- The last character in the string is at index `length-1`.

A `for` loop to 'visit' every 'index' in the instance of the string object `place` would be as follows:

```
for ( index = 0; index < place.length(); index++ )
{

}
```

7.11.1. Count the number of s's in place names

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

    System.out.print( "#Enter place name : " );
    String place = BIO.getString();

    while ( ! place.equals( "END" ) )
    {
      String placeLC = place.toLowerCase();      //make life easy

      int count = 0;
      for ( int i = 0; i < placeLC.length(); i++ )
      {
        if ( placeLC.charAt(i) == 's' )          //If an s
          count++;                               // add 1 to count
      }

      System.out.printf("Number of s's in %s is %d\n",
                        place,  count );

      System.out.print( "#Enter place name : " );
      place = BIO.getString();
    }
  }
}
```

```
#Enter place name : Sussex
Number of s's in Sussex is 3
#Enter place name : Seaford
Number of s's in Seaford is 1
#Enter place name : END
```

- Remember a string is immutable, so to convert all the characters in the String `place` to lower case you must write:

```
place = place.toLowerCase();
```

Which returns a new string in which all the upper case characters of the String `place` have been converted to lower case and then assigns this newly created string to the variable `place`

If you just write:

```
place.toLowerCase();
```

The result is thrown away, and `place` is left unchanged

7.11.2. Count s's in a string implemented as a function

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

    System.out.print( "#Enter place name : " );
    String place = BIO.getString();

    while ( ! place.equals( "END" ) )
    {
      System.out.printf("Number of s's in %s is %d\n",
                        place,  countCh( place, 's' )      );

      System.out.print( "#Enter place name : " );
      place = BIO.getString();
    }
  }


  public static int countCh( String s, char c )
  {
    String sLC = s.toLowerCase();

    int count = 0;
    for ( int i = 0; i < sLC.length(); i++ )
    {
      if ( sLC.charAt(i) == c )            //If equal to c
        count++;                           // add 1 to count
    }
    return count;
  }

}
```

```
#Enter place name : Sussex
Number of s's in Sussex is 3
#Enter place name : Seaford
Number of s's in Seaford is 1
#Enter place name : END
```

7.12. Quick quiz

- Write a Java code fragment to print out the first character of the string held in the variable `name`.
- Write a Java code fragment to print out the `n`'th (index held in the variable `n`) character of the string held in the variable `name`.
- Write a Java code fragment to print out the last character of the string held in the variable `name`.

7.13. Quick quiz answer

- Write a Java code fragment to print out the first character of the string held in the variable `name`.

```
System.out.print( name.charAt( 0 ) );
```

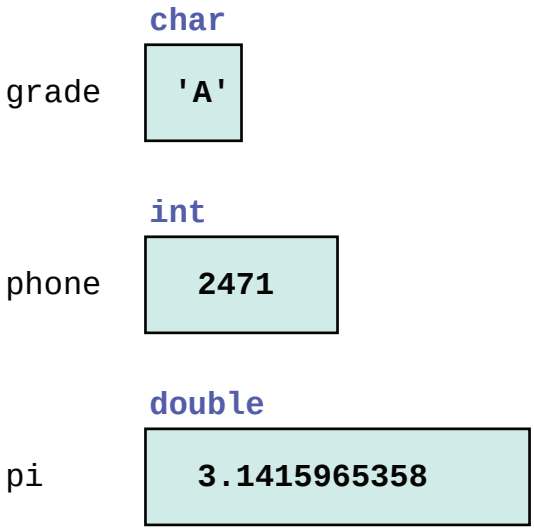
- Write a Java code fragment to print out the `n`'th (index held in the variable `n`) character of the string held in the variable `name`.

```
System.out.print( name.charAt( n ) );
```

- Write a Java code fragment to print out the last character of the string held in the variable `name`.

```
char c = name.charAt( name.length()-1 );
System.out.print( c );
```

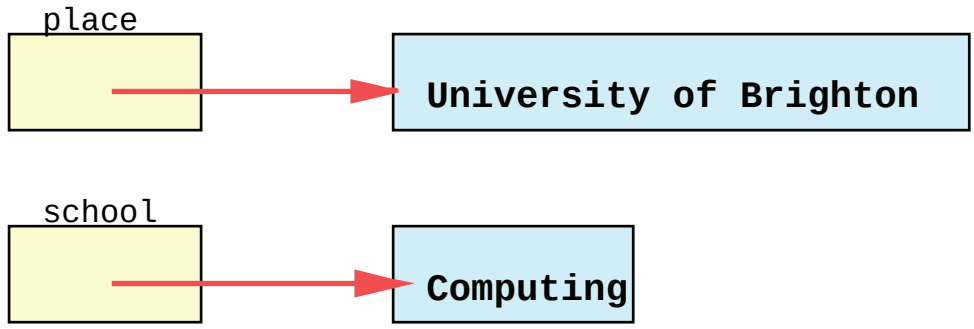
7.14. primitive types



- The size of instances of different primitive types may not be the same. For example, an instance of a `char` is 2 bytes, an instance of an `int` is 4 bytes, an instance of a `double` is 8 bytes.

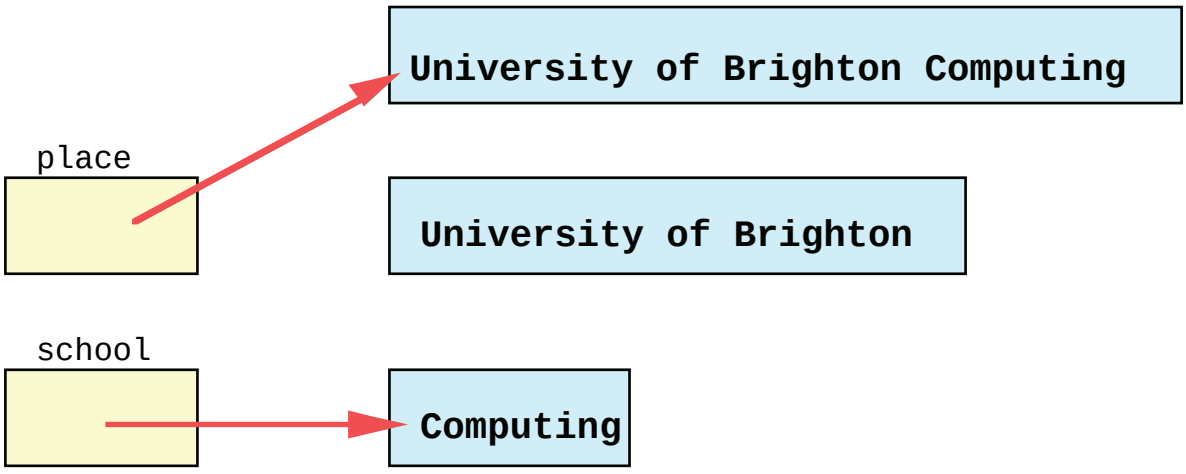
  An instance of a primitive type will hold directly the value stored in it.

7.15. reference types: `class`



- The size of an instance of a reference type is always the same. An instance of the reference type: `class` holds a 'handle' which 'points' to the storage for the object. This storage contains the physical characters of the stored string. This storage is held in an area of memory called the heap.

```
place = place + school;
```



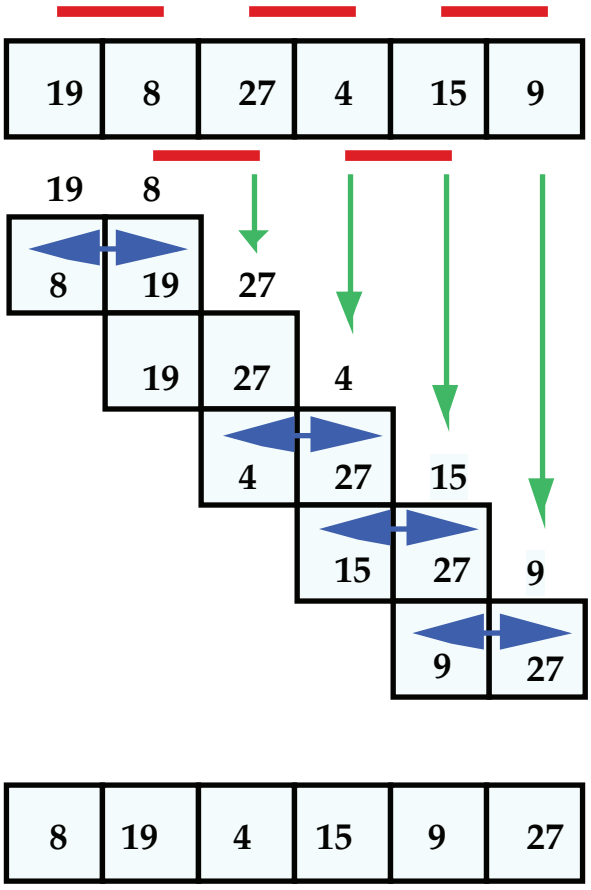
When the stored data is no longer accessible the memory used to store the data can be garbage collected (joined with other garbaged collected memory and made available for newly created objects etc.). This garbage collection is done automatically by the Java run time system.

7.16. Sorting (Bubble sort ascending)

Algorithm

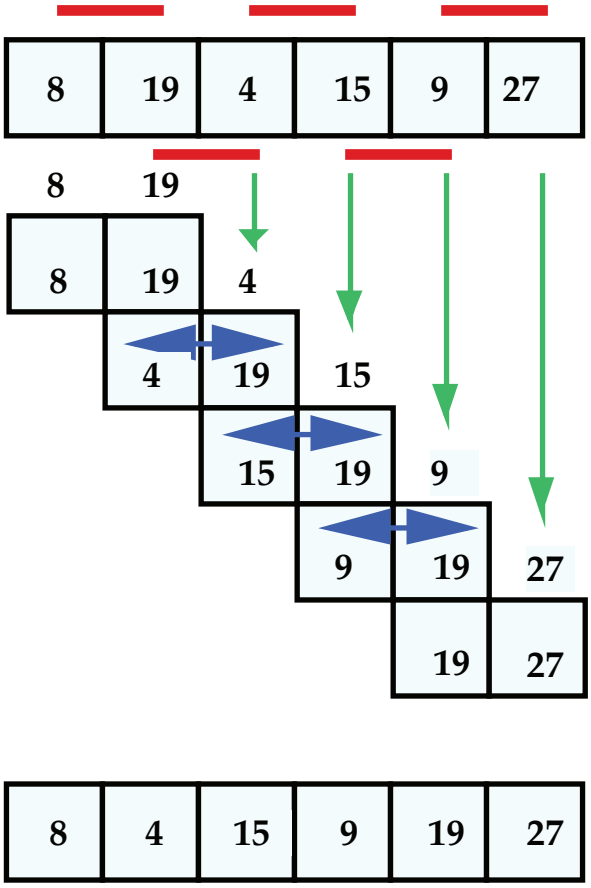
Step 1 Look at successive pairs of numbers in the list, for each pair put in correct ascending order

The largest number bubbles up to the end

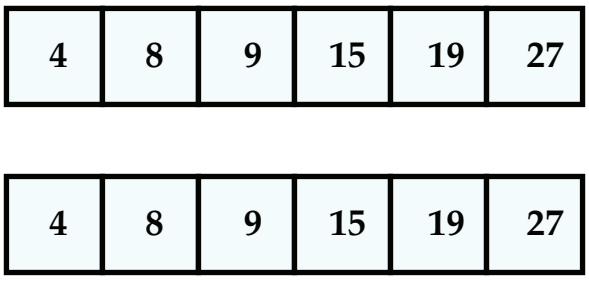


Step 2 Repeat 'Step 1' till there are no interchanges of numbers.
The numbers are now sorted

Step 1 Second pass



Step 1 Third pass & fourth pass



Step 1 In this case after the third pass the numbers are in order. So as the fourth pass does not swap any numbers, we know that the numbers are in order.

7.17. What you should know

- That you need to spend at least 5 hours per week on this module in addition to attending lectures, seminars and tutorials.
- That there will be assessed programming exercises to do throughout the year.
- That learning a computer programming language is not an activity that you can undertake in the last few weeks of the academic year.
- How to declare and use strings
- Idea of Object and message
- How to write a method that returns a result
- How the bubble sort works

8. Arrays

- Reference type: array

```
String greethamStreet[] = new String[25];
```

An array is accessed by using an index. The index is an integer number. This is very similar to naming an individual house in a street or road of like houses by a number.

0 Greetham Street / / Streets have no number 0
1 Greetham Street
2 Greetham Street
3 Greetham Street
4 Greetham Street

- Why use arrays
  In many cases an application needs to store and process several items of the same type. An array make this task easier.

- Both the array and its contents are stored on the heap.

  When the array can no longer be accessed the array and its contents may be garbage collected (and re-used again when storage is required).

8.1. Core ideas in an imperative programming language

- Sequence

```
double pricePerKilo = 1.20;

System.out.print( "#Input Kilos of apples: " );
double kilosOfApples = BIO.getDouble();

double cost = pricePerKilo * kilosOfApples;

System.out.print( "Cost of apples is (GBP) " );
System.out.println( cost );
```

- Iteration

```
int count = 1;

while ( count <= 5 )
{
  System.out.println( "Hello" );
  count = count + 1;
}
```

- Selection

```
if ( month == 2 )
   System.out.println("The month is February");
```

- Abstraction

```
public static void main( String args[] )
{
  printTree( 9 );
}

public static void printTree( int maxFoliageHeight )
{
  printFoliage( maxFoliageHeight );
  printTreeTrunk( maxFoliageHeight );
}
```

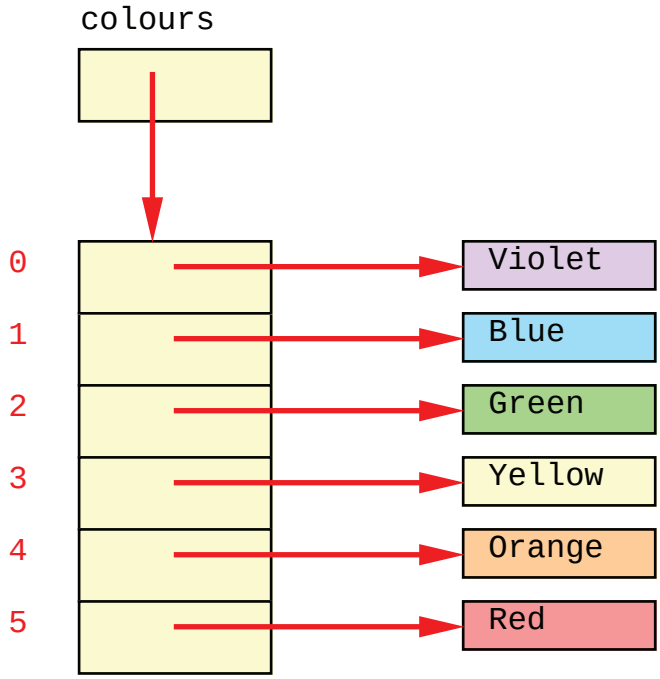
8.2. Arrays

A collection of like things accessed using an index (instance of an `int`, `byte`, `short`, `char`).

```
String colours[] = new String[6];
```

`colours` is a collection of 6 strings.

After populating the array with strings representing the colours of the rainbow the internal structure would be of the form.



Individual strings in the array `colours` are accessed using an array index in the range `0 .. 5`.

Array indexes in Java always start at 0.

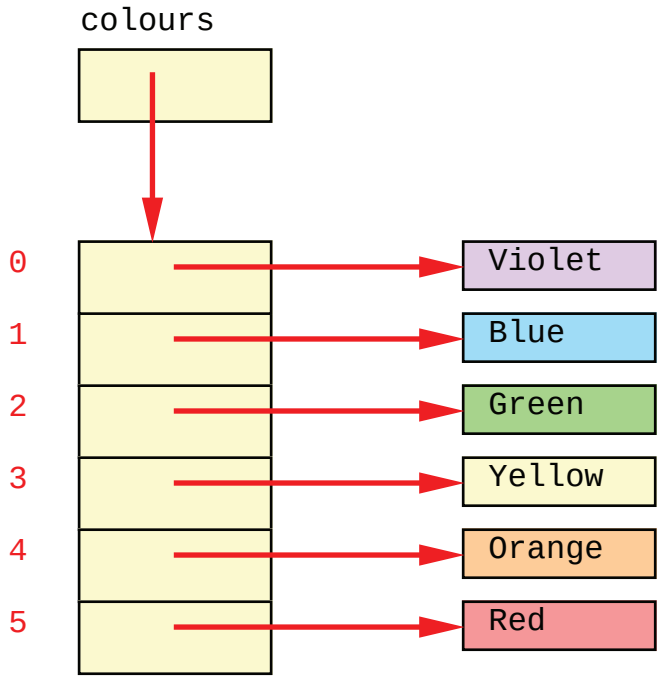
```
colours[2] = "Green";
```

would set the element indexed by `2` to `"Green"`

The number of elements in an array is given by using the field `length` which is immutable (read only).

```
String colours[] = new String[6];
```

After populating the array with strings representing the colours of the rainbow the internal structure would be of the form.



Then the number of elements in `colours` is given by

```
System.out.println( colours.length );
```

- Note `length` is a field of the array.
  It is a declared as `final` so can not be assigned too.

8.3. Example program using arrays

A program to store and then print out the colours of the rainbow.

```
class Main
{
  public static void main( String args[] )
  {
    String colours[] = new String[6];
    colours[0] = "Violet";
    colours[1] = "Blue";
    colours[2] = "Green";
    colours[3] = "Yellow";
    colours[4] = "Orange";
    colours[5] = "Red";

    System.out.println("Colours of the rainbow are:");

    for ( int i =0; i<colours.length; i++ )
      System.out.print( colours[i] + " " );

    System.out.println();
  }
}
```

? Is it good or bad not to use { } with the `for` statement

```
Colours of the rainbow are:
Violet Blue Green Yellow Orange Red
```

8.4. Same program

With explicit declaration and initialisation of the array `colours`.

```
class Main
{
  public static void main( String args[] )
  {
    String colours[] = { "Violet", "Blue", "Green",
                         "Yellow", "Orange", "Red" };

    System.out.println("Colours of the rainbow are:");
    for ( int i =0; i<colours.length; i++ )
      System.out.print( colours[i] + " " );
    System.out.println();
  }
}
```

Now with split into two functions (methods)

```
class Main
{
  public static void main( String args[] )
  {
    String rainbow[] = { "Violet", "Blue", "Green",
                         "Yellow", "Orange", "Red" };

    printColoursOfRainbow( rainbow );

  }

  public static void printColoursOfRainbow( String [] colours )
  {
    System.out.println("Colours of the rainbow are:");
    for ( int i =0; i<colours.length; i++ )
      System.out.print( colours[i] + " " );
    System.out.println();
  }
}
```

Now with the declaration of array split into two statements. Note the use of the keyword **new**.

```
class Main
{
  public static void main( String args[] )
  {
    String rainbow[];

    rainbow = new String [] { "Violet", "Blue", "Green",
                              "Yellow", "Orange", "Red" };

    printColoursOfRainbow( rainbow );

  }

  public static void printColoursOfRainbow( String [] colours )
  {
    System.out.println("Colours of the rainbow are:");
    for ( int i =0; i<colours.length; i++ )
      System.out.print( colours[i] + " " );
    System.out.println();
  }
}
```

The statement

```
rainbow = new String [] { "Violet", "Blue", "Green",
                          "Yellow", "Orange", "Red" };
```

Could be used later in the program perhaps with different values, or number of values. This replaces all the contents of the array `rainbow` with 'new' values.

8.5. Technical details: A Java program

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

When you invoke a Java program it is called with a list of instances of a `String`. This list is passed as an array to the `main` method. The following code will print out these strings

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

    for ( int i=0; i<args.length; i++ )  //Arg 0 is the 1st arg
    {
      System.out.print( args[i] );       //Write argument
      if ( i != args.length-1 )
        System.out.print( " " );         //Separator
    }
    System.out.println();                //New line

  }
}
```

Example call from the command line

```
  java Main    Passed to the Java application as individual Strings
```

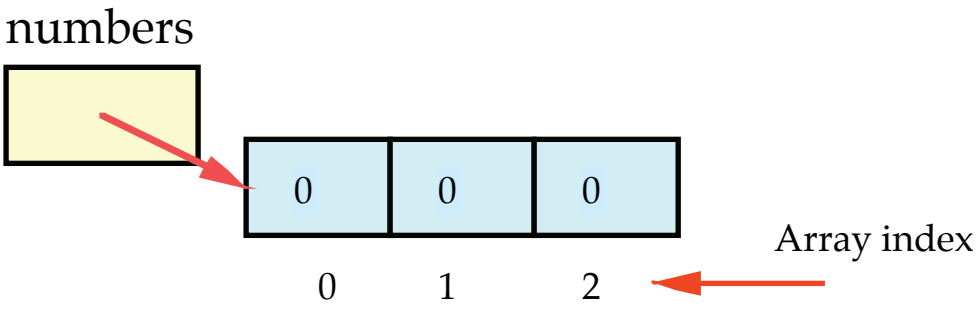
8.6. Technical details: reference type: Array

```
int numbers[] = new int[3];
```

The array declaration shown above consists of two distinct components:

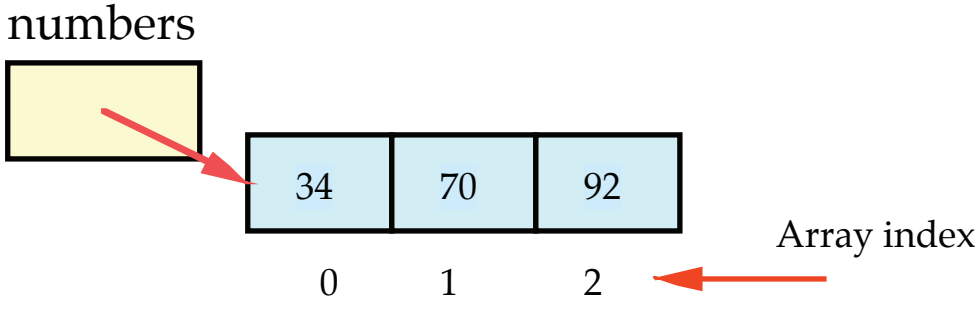
1	Declaration of a handle for the array	`int numbers[]`
2	Allocation of storage for the array	`= new int[3]`

```
int numbers[];                  // Allocate handle
numbers = new int[3];           // Allocate storage for array
```




```
numbers[0] = 34;
numbers[1] = 70;
numbers[2] = 92;
```

The storage for the array after assigning the values 34, 70, 92 to individual elements is illustrated below:



8.7. Quick quiz

- Write a Java declaration of an array of 6 `int`'s called `modules`.

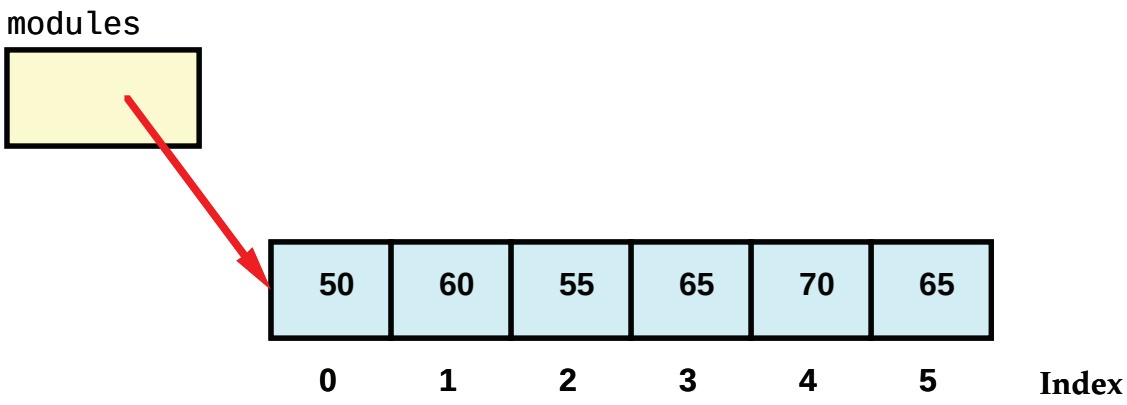
- Assume that this a new declaration of the array contains marks for modules taken by a student. Write a fragment of code to print the average of the marks in the array `modules`

8.8. Quick quiz answers

- Write a Java declaration of an array of 6 ints

```
int modules[] = new int[6];
```

- Write a fragment of code to print the average of the marks in the array `modules`




```
int sum = 0;
for ( int i=0; i<modules.length; i++ )
{
  sum += modules[i];
}

double average =
    sum / (double) modules.length;

System.out.println( average );
```

8.9. Using arrays

A program to find a persons e-mail address.

The data is stored in two parallel arrays, one array contains the persons name the other their e-mail address.

	name	eMail
0	Thomas	t27@brighton.ac.uk
1	Gordon	g56@brighton.ac.uk
2	Clarabelle	c99@brighton.ac.uk
3	Annie	a94@brighton.ac.uk
4	Henry	h11@brighton.ac.uk

The user of the application enters a person name and the application returns their e-mail address.

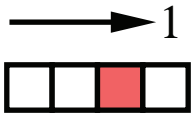
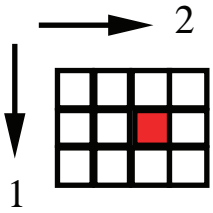
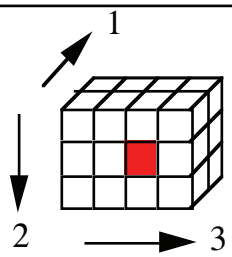
The program simply searches the array `name` for a match and the corresponding entry in the `eMail` array is their e-mail address.

A more sophisticated solution is to use one of the `map` classes in the Java library, where the searching will be implemented in a very effective way.

```
#Find e-mail address
#Enter name to lookup : Henry
Email address for Henry is h11@brighton.ac.uk
#Enter name to lookup : Annie
Email address for Annie is a94@brighton.ac.uk
#Enter name to lookup : Thomas
Email address for Thomas is t27@brighton.ac.uk
#Enter name to lookup : END
```

```
class Main
{
  private static String name[];  //Visible to all in the class
  private static String eMail[]; //Visible to all in the class

  public static void main( String args[] )
  {
    populate();

    System.out.println("#Find e-mail address");
    System.out.print("#Enter name to lookup : ");
    String aName = BIO.getString();

    while ( ! aName.equals( "END" ) )        //End of data
    {
      boolean found = false;
      for ( int i=0; i<name.length; i++ )
      {
        if ( aName.equals( name[i] ) )
        {
          System.out.printf("Email address for %s is %s\n",
                            aName, eMail[i] );
          found = true;
          break;                  //Break out of loop now
        }
      }
      if ( ! found )
         System.out.printf("Name %s not known\n", aName );

      System.out.print("#Enter name to lookup : ");
      aName = BIO.getString();
    }
  }

  public static void populate()
  {
    name  = new String[5];
    eMail = new String[5];

    name[0] = "Thomas";      eMail[0] = "t27@brighton.ac.uk";
    name[1] = "Gordon";      eMail[1] = "g56@brighton.ac.uk";
    name[2] = "Clarabelle"; eMail[2] = "c99@brighton.ac.uk";
    name[3] = "Annie";       eMail[3] = "a94@brighton.ac.uk";
    name[4] = "Henry";       eMail[4] = "h11@brighton.ac.uk";
  }
}
```

8.10. Arrays overview

Declaration of array handle	Conceptual model of storage	Allocation of physical storage for array an access to shaded element
`int vector[];`		`vector = new int[4];` `int v = vector[2];`
`int table[][];`		`table = new int[3][4];` `int v = table[1][2];`
`int cube[][][];`		`cube = new int[2][3][4];` `int v = cube[0][1][2];`

8.11. Information hiding

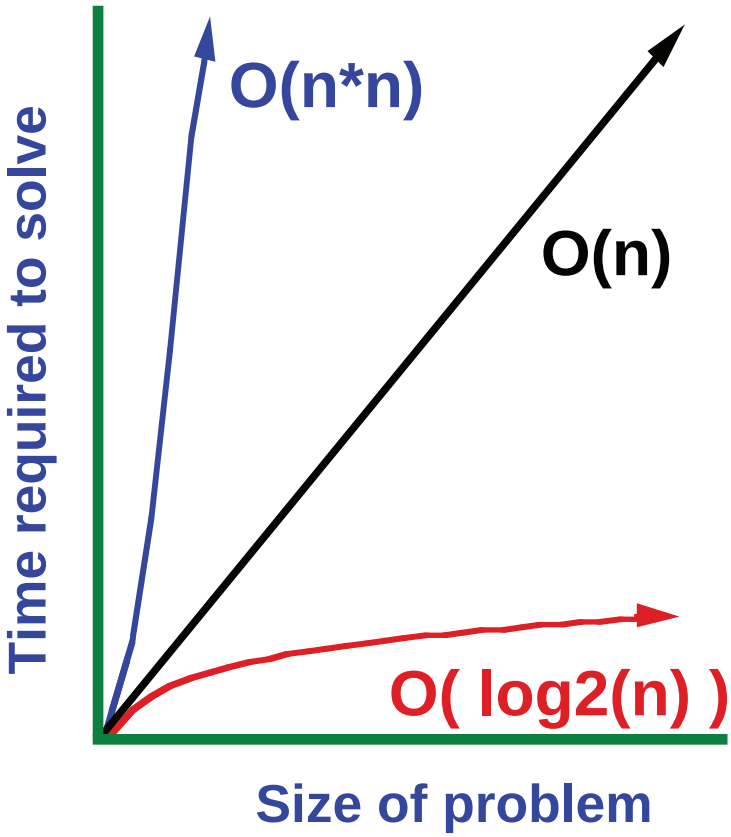
Scope of variables in a class

```
class Main
{
  private static String name[] ;


  public static void main( String args[] )
  {
    Sting aName = BIO.getString();

  }



  public static void popuulate()
  {
    name = new String[5];

    name[0] = "Thomas";

  }

}
```

- Declared outside of a method in a class.
  The scope of is from the declaration (`name`) to the end of the class.

- Declared within the 'outer block' of the method
  The scope of is from the declaration (`aName`) to the end of the method.

8.12 Effort (time) required to solve a problem



The big O notation gives an estimate of how long it will take to solve a problem with n items.

So O(n) means that if you have **n** items it will take **n** units of time to solve the problem.

8.12. About the search algorithm

- On average it will take approximately N/2 comparisons to find a name.

- OK, if only a few names to search. But if 1,000,000 names then it will take 500,000 comparisons on average to find a name.

  The above algorithm is said to be of order [Big O notation] O(n/2). In that it will take time O(n/2) to find an item. A better searching algorithm will find the item in time O($\log_2$ n)

  If you are looking up a lot of names then this will be a slow process.

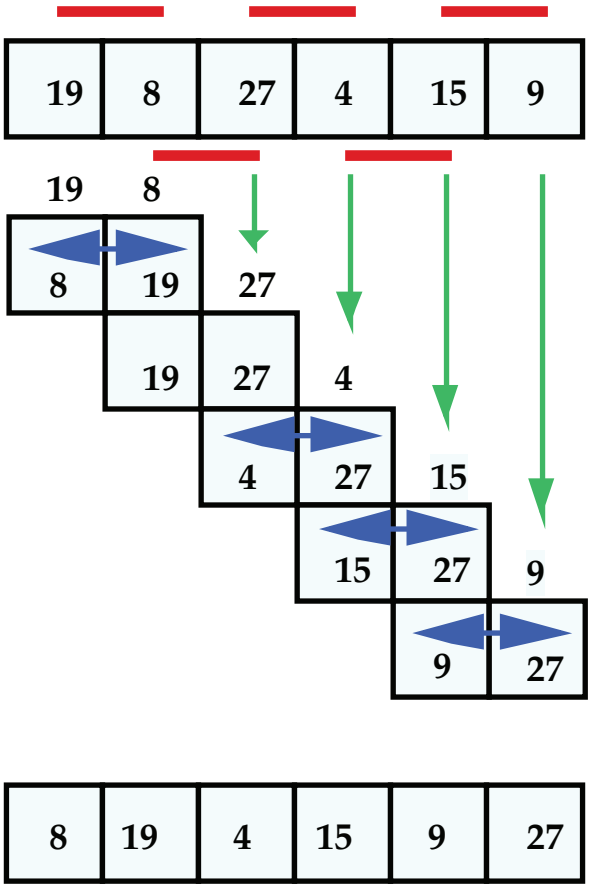
  - Domain name to IP address
  - LDAP lookup of your username so the password can be checked.
  - Details about a user on facebook.

8.13. Sorting (Bubble sort ascending)

Algorithm

Step 1 Look at successive pairs of numbers in the list, for each pair put in ascending order if not already

The largest number bubbles up to the end



Step 2 Repeat 'Step 1' till there are no interchanges of numbers

```
class Main
{
  public static void main( String args[] )
  {
    int numbers[] = new int[6];           //Array
    numbers[0] = 30;                      //Populate
    numbers[1] = 20;
    numbers[2] = 60;
    numbers[3] = 50;
    numbers[4] = 10;
    numbers[5] = 40;

    boolean haveSwapped = true;           //Assume not in order

    while ( haveSwapped )                 //Are in order
    {
      haveSwapped = false;                //Assume in order

      for (int i=0; i<numbers.length-1; i++)
      {                                   //Check pair
         if ( numbers[i] > numbers[i+1] ) // Not in order
         {
           int tmp = numbers[i];          // Swap
           numbers[i] = numbers[i+1];
           numbers[i+1] = tmp;
           haveSwapped = true;            // record not in order
         }
      }                                   //End pass
    }

    for ( int i=0; i<numbers.length; i++ )//Print result
    {
      System.out.print( numbers[i] + " " );
    }
    System.out.println();

  }
}
```

8.14. About the algorithm

- In the worst case it will take n-1 passes to put the numbers in order plus 1 pass to find that they are in order.

  Each pass will involve n-1 comparisons and there will be n passes through the data.

  Total number of comparisons is n . (n-1) which is of the order of n^2

Numbers	Comparisons (order of n^2)
10	100
100 (10^2)	10000 (10^4)
1000 (10^3)	1000000 (10^6)
Million (10^6)	Trillion (10^{12})

Actual clock times (Intel Core 2 2.83Ghz)

Numbers Random	Com-parisons	Time Seconds	Time Hours
10	100	0.000	0.000
100	10000	0.000	0.000
1000	1000000	0.003	0.000
10000 (10^4)	10^8	0.343	0.000
10^5	10^{10}	33.298	0.009
10^6	10^{12}	3342.886	0.929

Estimate

10^7	10^{14}	4 days	93
10^8	10^{16}	1.1 years	9,300

8.15. What you should know

- That you need to spend at least 5 hours per week on this module in addition to attending lectures, seminars and tutorials.
- That there will be assessed programming exercises to do throughout the year.
- That learning a computer programming language is not an activity that you can undertake in the last few weeks of the academic year.
- How to declare and use arrays
- How the bubble sort algorithm works
- Why the bubble sort is not the best algorithm to use.

9. The software crisis

- Why
- Ideas of Class, Object, Method, Message.
- The class construct
- Creating and using a class in Java
- Constructors
- Why use OO ideas

9.1. Software crises and beyond

- Software not being delivered on-time
- Software costs escalating
- Software not working as expected
- Rise of OO programming
  Software re-use
  Components

9.2. Core ideas in an imperative programming language

- Sequence

```
double cost = pricePerKilo * kilosOfApples;
System.out.print( "Cost of apples is (GBP)" );
System.out.print( cost);
```

- Iteration

```
for ( int count = 1; count <= 5; count++ )
{
  System.out.println( "Hello" );
}
```

- Selection

```
if ( month == 2 )
  System.out.println("The month is February");
else
  System.out.println("The month is not February");
```

- functional decomposition

```
public static void main( String args[] )
{
  printTree( 9 );
}

public static void printTree( int maxFoliageHeight )
{
  printFoliage( maxFoliageHeight );
  printTreeTrunk( maxFoliageHeight );
}
```

How to write unmaintainable code.

`http://thc.org/root/phun/unmaintain.html`
(accessed 27 November 2014)

Lines of code to create:

Windows XP	45,000,000
Linux Kernel 2.6.35	13,500,000
Android OS	12,141,638
MySQL	1,900,000

Professional programmer will produce 100 - 1000 lines of code per month.

Debugged, documented

- One of my most productive days was throwing away 1000 lines of code.

  Ken Thompson
  Early developer of Unix, Plan 9 and Co-creator of the programming language Go.

- Simplicity is the ultimate sophistication.

  Leonardo da Vinci

- Simplicity is prerequisite for reliability.

  Edsger W. Dijkstra
  1972 Turing Award winner

- The computing scientist's main challenge is not to get confused by the complexities of his own making.

  Edsger W. Dijkstra

- The competent programmer is fully aware of the limited size of his own skull. He therefore approaches his task with full humility, and avoids clever tricks like the plague.

  Edsger W. Dijkstra

- The central enemy of reliability is complexity.

  Geer et al.
  Computer security expert

- There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies and the other way is to make it so complicated that there are no obvious deficiencies.

  C.A.R. Hoare,
  The 1980 ACM Turing Award Lecture

- Beauty is more important in computing than anywhere else in technology because software is so complicated. Beauty is the ultimate defence against complexity.

  David Gelernter
  Computer scientist

- The most amazing achievement of the computer software industry is its continuing cancellation of the steady and staggering gains made by the computer hardware industry.

  Henry Petroski
  Professor of Civil engineering & Professor of History Duke University

9.3. Software

Fred Brooks (Mid 1980's)

The Mythical Man-month

> Adding people to a late project will make matters worse

No silver bullet

> Hardware 6 orders of magnitude in performance gain in 30 years

Argument

> Difficulties in software production are inevitable, arising from the very nature or essence of software. Not by accident but by how software engineers build software today (Mid 1980's).

Features of software

Complexity

- Each program is different
- Complexity increases non-linearly with size

Changeability

- As a software product is found to be useful, people use it in new ways and demand new facilities.
- Software survives beyond the life of the machine it was originally written for.

9.4. There is a silver Bullet

Brad Cox (Late 1980's)

There is a silver bullet

Culture change

- Away from a craft industry
- Build software from re-usable and interchangeable parts

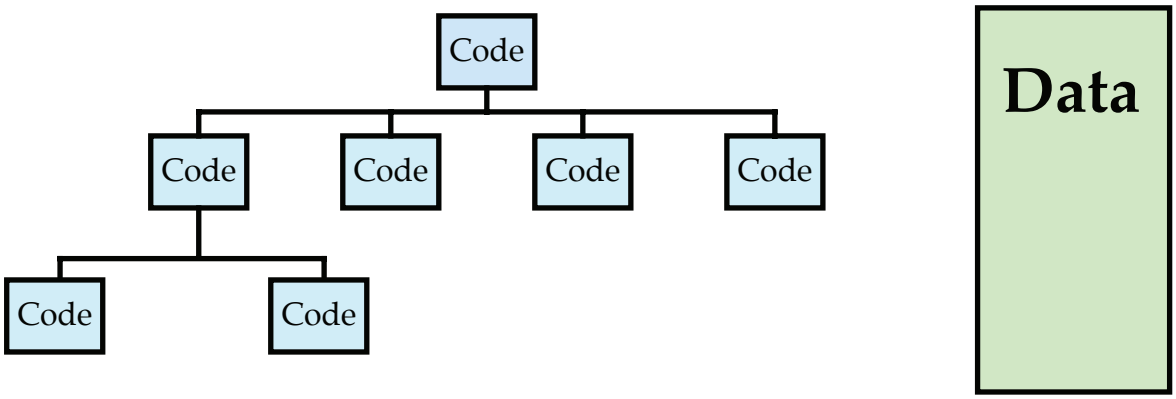
Change how we view software

From the processes use to build software to the objects we build.

- A mobile phone is made from many standard parts
- A TV is made from many standard parts
- A computer is made from many standard parts
- A computer program is made from many standard parts.

9.5. Functional decomposition

- Find all things that need to be done
- Decompose into smaller tasks
- Stop when task can easily be implemented in a few programming language statements

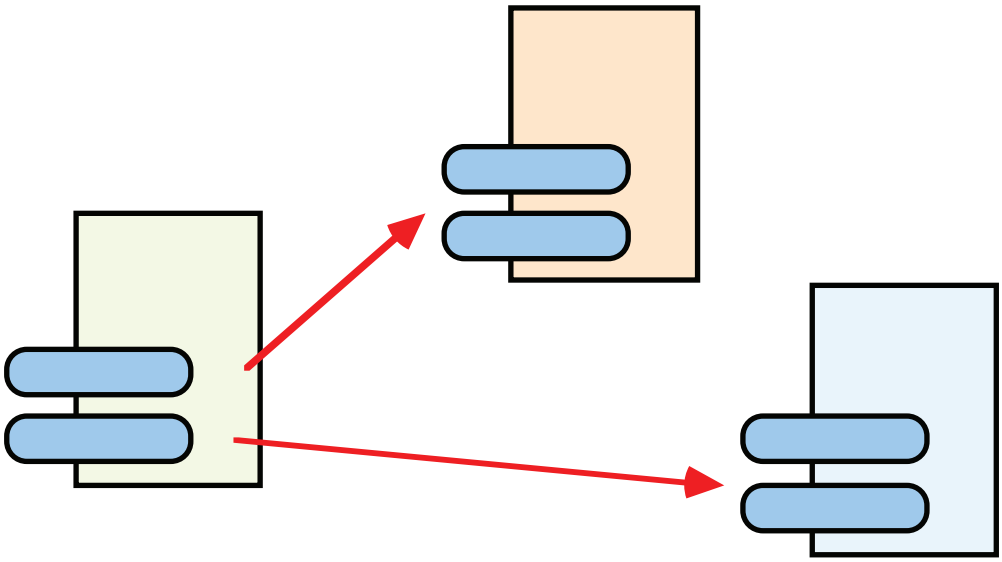


- Difficult to re-use part of a program written this way.
  Code is separated from the data that it uses.
  Hence difficult to isolate a section of program (code + data) to re-use.

9.6. Object Oriented programming

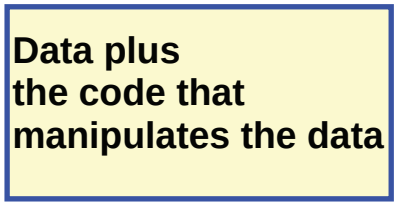
- Asks first about intent - what
- Goals
  - To find the objects and the connections
  - What operations need to be performed

    Each object knows how to perform its own operations and remember its own data



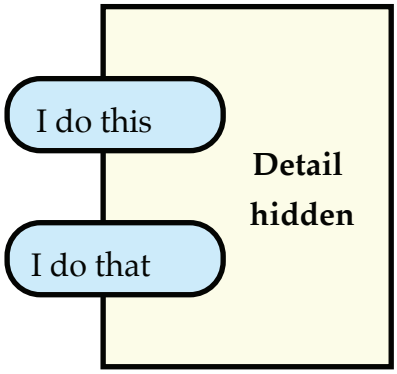
9.7. What are objects

- Encapsulation of code and data



Example
Code: Operations on a bank account
Data: Overdraft limit and the account balance for an individual

- Information Hiding



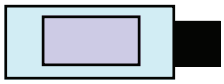
- I do these things but hide the implementation.

  Public interface of **what** I can do
  Private representation & implementation of **how** it is done

9.8. Classes, Objects, Methods, Messages

Class

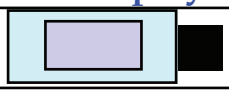
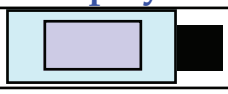
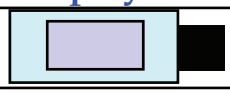
All objects that share the same responsibilities and state information (not necessarily there values) belong to the same class.



MP3 player

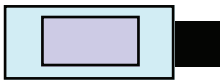
Object

An instance of a class

Corinna's player	Mike's player	Miri's player
		

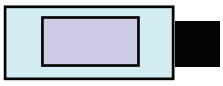
Message

Invokes a method in an object a result may be returned.

Mike's player	Example messages
	Play Next track Display current track Delete track

Method

Inspects or mutates the object. Implementation of the message .

Corinna's player	Example method
	Play [decompress in real-time the mp3 track to a form that represents sound waves]

Inspects Returns information about the state of the object

Mutates Changes the state of the object, may also return information about the object.

9.9. Quick quiz

Classify the following into classes and objects

- University, Dog, Cat
- University of Brighton, Cleo the dog,
- Ship, RMS Titanic, Cutty Sark, Tugboat
- `String`, `"Hello world", "class"`

Which of the following are inspector methods and which are mutator methods (in relationship to an mp3 player).

- Delete track
- Display current track

Class A 'collective name' for similar things. An instance of which will be an object.

Object Is an instance of a class

Inspects Returns information about the state of the object

Mutates Changes the state of the object, may also return information about the object.

9.10. Quick quiz answers

Classify the following into classes and objects

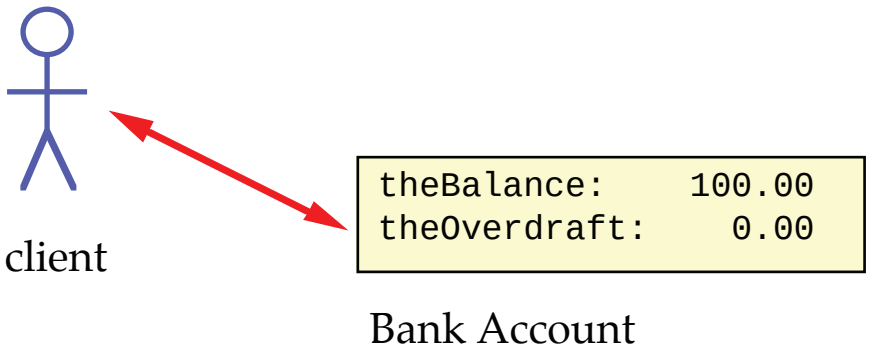
- University - **Class**, Dog - **Class**, Cat - **Class**
- University of Brighton - **Object**,
  Cleo the dog - **Object**,
- Ship - **Class**, RMS Titanic - **Object**,
  Cutty Sark - **Object**, Tugboat - **Class**
- `String` - **Class**, `"Hello world"` - **Object**,
  `"class"` - **Object**

Which of the following are inspector methods and which are mutator methods (in relationship to an mp3 player).

- Delete track - Mutates
- Display current track - Inspects

9.11. A Bank Account

Operations to perform on a bank account



- Deposit money into the account
- Get the balance of the account
- Get the overdraft limit allowed
- Set the overdraft limit allowed on the account
- Withdraw money from the account

9.12. A class to represent a Bank account

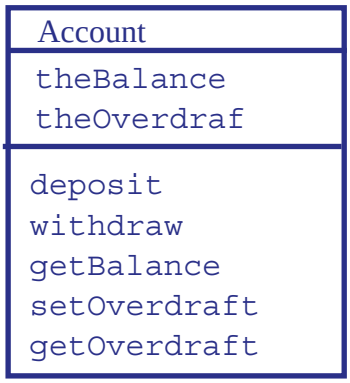
9.12.1. Public components

Method	Behaviour
`void deposit( double m )`	Deposits money `m` into the account.
`double getBalance()`	Returns the current balance of the account.
`double getOverdraft()`	Returns the overdraft limit for the account.
`void setOverdraft(double ov)`	Sets the overdraft to `ov` for the account. A negative amount indicates an overdraft is allowed.
`double withdraw( double m )`	Withdraws money `m` from the account. Naturally money can only be withdrawn if the account balance stays above or equal to the minimum balance allowed.

9.12.2. Private components

Instance variable	Used to hold
`theBalance`	Holds the current balance of the account, may be negative.
`theOverdraft`	The overdraft that the account is allowed to go to. If the account holder was allowed an overdraft of 200.00 then `theOverdraft would` be set to -200.00

9.12.3. UML Diagram



9.13. Sending a message to an object in Java

To send the message `deposit` with a parameter of `100.00` to the object `mike`.

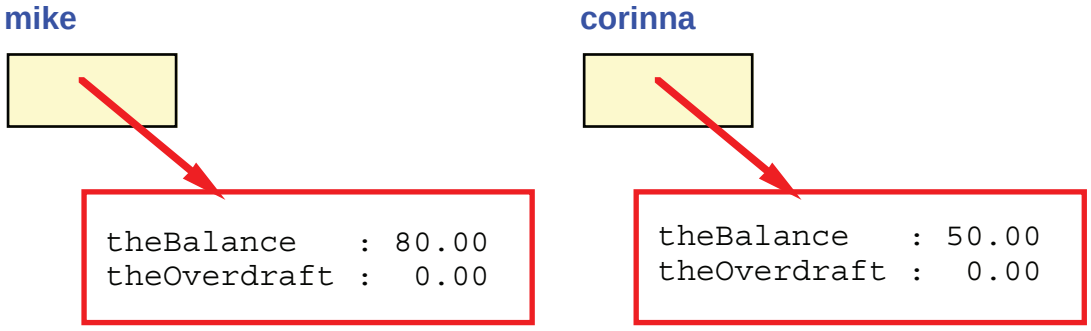
```
mike.deposit( 100.00 );
```

Components of:	
`mike.deposit(100.00);`	object, message sent, parameter to message `mike.deposit(100.00)`

9.14. An example program

```
class Main
{
  public static void main( String args[] )
  {
    Account mike    = new Account();
    Account corinna = new Account();

    double obtained;

    System.out.printf( "Mike's Balance      = %10.2f\n",
                       mike.getBalance() );

    mike.deposit(100.00);
    System.out.printf( "Mike's Balance      = %10.2f\n",
                       mike.getBalance() );

    obtained = mike.withdraw(20.00);
    System.out.printf( "Mike has withdrawn  : %10.2f\n", obtained);
    System.out.printf( "Mike's Balance      = %10.2f\n",
                       mike.getBalance() );

    corinna.deposit(50.00);
    System.out.printf( "Corinna's Balance   = %10.2f\n",
                       corinna.getBalance() );
  }
}
```

```
Mike's Balance      =       0.00
Mike's Balance      =     100.00
Mike has withdrawn  :      20.00
Mike's Balance      =      80.00
Corinna's Balance   =      50.00
```



9.15. The class Account

```
class Account
{
  private double theBalance    = 0.00;   //Balance of account
  private double theOverdraft  = 0.00;   //Overdraft allowed

  public double getBalance()
  {
    return theBalance;
  }

  public double withdraw( final double money )
  {
    if ( theBalance - money >= theOverdraft )
    {
      theBalance = theBalance - money;
      return money;
    } else {
      return 0.00;
    }
  }

  public void deposit( final double money )
  {
    theBalance = theBalance + money;
  }
  public void setOverdraft( final double money )
  {
    theOverdraft = money;
  }

  public double getOverdraftLimit()
  {
    return theOverdraft;
  }
}
```

WARNING

Holding a monetary amount in a double is not a good idea, as rounding errors will lead to discrepancies in the 'money' held.

9.16. Quick quiz

In the above class write a method:

- `inCredit`
  That returns true if the account is in credit otherwise if overdrawn would return false.

  An account is in credit if its balance is greater than or equal to zero.

```
if ( miri.inCredit() )
{
  //All ok
}
```

- `watchOverdraft`
  That returns true if the customers current balance is nearer than the supplied parameter `margin` to the overdraft limit:

```
Account bank[] = new Account[100];

// Code that sets up bank etc

for ( int i=0; i<bank.length; i++ )
{
  if ( bank[i].watchOverdraft(200.00) )
  {
    //Send warning letter
  }
}
```

9.17. Quick quiz answers

In the above class write a method:

- `inCredit`
  That returns true if the account is in credit otherwise if overdrawn would return false.

  An account is in credit if its balance is greater than or equal to zero.

```
public boolean inCredit()
{
  return theBalance >= 0.00;
}
```

- `watchOverdraft`
  That returns true if the customers current balance is nearer than the supplied parameter `margin` to the overdraft limit:

```
public boolean watchOverdraft( double margin )
{
  return inCredit() ? false :
                      theBalance - margin < theOverdraft;
}
```

9.18. Calling a method (what actually happens)

As seen by the programmer:

```
public void deposit( final double money )
{
  theBalance = theBalance + money;
}
```

```
  mike.deposit(100.00);
```

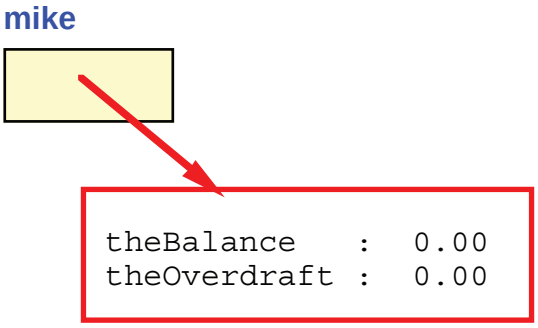
The method `deposit` is actually implemented by the compiler as:

```
public void deposit( Account this, final double money )
{
  this.theBalance = this.theBalance + money;
}
```

The secret 1st parameter is the object the message will be sent to (know as **`this`**). The object **`this`** (an instance of Account contains the instance variables `theBalance` & `theOverdraft`).

The method to send the message `deposit` to the object `mike` is actually implemented as:

```
deposit( mike, 100.00 );
```



In the process of setting up the environment to execute `mike.deposit(100.00)` the handle `mike` is assigned to `this` and the value `100.00` is assigned to the parameter `money`.

Note the formal parameter `money` has been declared as **`final`** and thus can not be changed within the method

8.18. More about the use of classes

Information hiding

```
class Main
{
  public static void main( String args[] )
  {
    Account mike    = new Account();
    Account corinna = new Account();

    double obtained;

    System.out.printf( "Mike's Balance      = %10.2f\n",
                       mike.getBalance() );


    mike.theBalance = 10000.00;  // Will not compile

    obtained = mike.theBalance;  // Will not compile
  }
}
```

By making the instance variables of the class **`private`** a user of an instance of the class can not 'cheat' and directly change the balance of the account.

If this were allowed then in a later revision of the class, the balance of the account could be held in a database, and hence any code that directly accessed the balance would now not work.

9.19. Technical details: Class

```
class Account
{
  private double theBalance    = 0.00;    //Balance of account
  private double theOverdraft  = 0.00;    //Overdraft

  public double getBalance()
  {
    return theBalance;
  }

  public void deposit( final double money )
  {
    theBalance = theBalance + money;
  }
}
```

9.19.1. Terminology: Classes

- Instance variables
  [theBalance, theOverdraft]
  Variables in the class. Each instance of the class has its own instance variables. They should be declared as **private**.

  Access to the instance variables may also be written as: **this**.theBalance

- **private double** theBalance = 0.00;
  Access not allowed to this instance variable outside the class as **private**.

- **public void** deposit(**final double** money)
  Access to method allowed as **public**.

9.19.2. Visibility

Client view (A user of the class)	Only sees public components
	
	`Account mike = new Account();`
	`mike.deposit();`

Implementor view (The person who wrote or maintains the class)	Sees both public and private components inside the class
	
	`public double getBalance()`
	`{`
	`  return theBalance;`
	`}`

9.20. Technical details: Objects

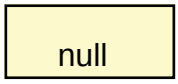
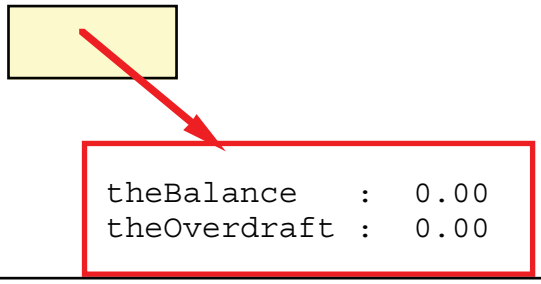
```
Account mike     = new Account();
```

The object declaration shown above consists of two distinct components:

1	The declaration of the handle for the object	`Account mike`
2.	The allocation of storage for the object.	`mike = new Account()`

```
Account mike;                    // Allocate handle
mike   = new Account();          // Allocate storage for object
```

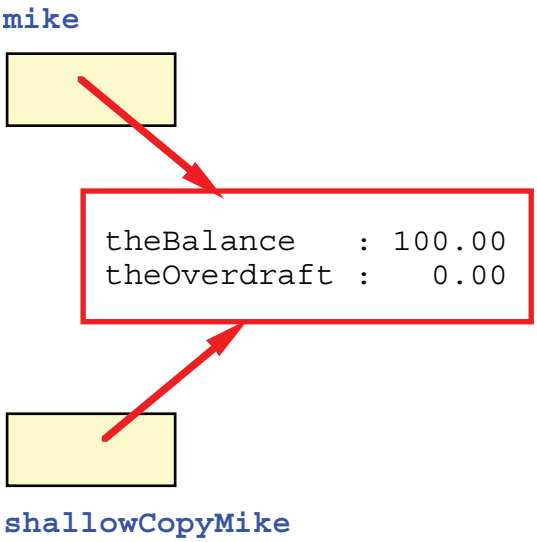
Symbolic representation of the relationship between the handle and the storage for the object `mike` is shown below

1	`Account mike`	**mike** 
2.	`mike = new Account()`	**mike** 

9.21. Consequence

```
Account mike              = new Account();
Account shallowCopyMike   = mike;

mike.deposit(100.00);

System.out.println("mike              = %10.2f\n",
                   mike.getBalance() );

System.out.println("shallowCopyMike = %10.2f\n",
                   shallowCopyMike.getBalance() );
```

```
mike             =     100.00
shallowCopyMike =     100.00
```



9.22. The Java class library

Database access (Relational)	java.sql
GUI java.swing	java.awt
I/O	java.io
Networking	java.net
Remote Method Invocation	java.rmi
Utilities	java.util

9.23. An example class from the class library

`java.io.PrintStream`

Method	Behaviour
`void`	`print( int anInt )` Write the integer `anInt` to the stream in character format.
`void`	`print( double aDouble )` Write the double aDouble to the stream in character format.
`void`	`print( char aCharacter )` Write the char aCharacter to the stream. *Also to print an instance of other standard types.*
`void`	`println()` Write a newline to the stream.
`void`	`println( int anInt )` Write the integer `anInt` to the stream in character format followed by a newline. *Also to print an instance of other standard types.*
`void`	`close()` Close the stream
	`PrintStream( OutputStream out )` Creates a new printstream.

```
System.out.println( "Hello world" );
System.out.println( 24.6 );
```

```
public final class System
{
  public static final PrintStream out;
  public static final InputStream in;

  private System() {}


}
```

- `System` can not be instantiated

9.24. Terminology: Classes

- Overloading [print]
  Several methods with the same name. Each method must have a unique signature so that the compiler can work out which version to use.

9.25. Why use Object-Oriented ideas

- Better organisation of inherent complexity
- Reduces development effort through code re-use
- Systems are easier to maintain
- Systems are easier to extend

9.26. Constructors

- Code executed when an object is created
- A class may have multiple constructors

```
class Account
{
  private double theBalance  = 0.00;  //Balance of account
  private double theOverdraft= 0.00;  //Overdraft

  public Account( final double openingBalance,
                  final double overdraft )
  {
    theOverdraft = overdraft;
    theBalance   = openingBalance;
  }

  public Account( final double openingBalance )
  {
    this( openingBalance, 0.00 );
  }

  public Account()
  {
    this( 0.00, 0.00 );
  }

  public double getBalance() {}
  public double withdraw( final double money ) {}
  public void   deposit( final double money ) {}
  public void   setOverdraft( final double money ) {}
  public double getOverdraft() {}
}
```

```
Account mike    = new Account();
Account corinna = new Account( 100.00, -200.00 );
```

9.26.1. Why have a constructor

- Guarantees initialisation
- Allows a user to specify the initialisation of an object, rather than being 'hard wired'.

```
class Main
{
  public static void main( String args[] )
  {
    Account mike = new Account(0.00, -100.00);
    Account cori = new Account(0.00);
    Account miri = new Account();

    double obtained;

    System.out.printf( "Mike's Balance      = %10.2f\n",
                       mike.getBalance() );

    obtained = mike.withdraw(20.00);
    System.out.printf( "Mike has withdrawn  : %10.2f\n", obtained);
    System.out.printf( "Mike's Balance      = %10.2f\n",
                       mike.getBalance() );

    cori.deposit(50.00);

    obtained = cori.withdraw(30.00);
    System.out.printf( "Cori has withdrawn  : %10.2f\n", obtained);
    System.out.printf( "Cori's Balance      = %10.2f\n",
                       cori.getBalance() );
  }

}
```

```
Mike's Balance      =       0.00
Mike has withdrawn  :      20.00
Mike's Balance      =     -20.00
Cori has withdrawn  :      30.00
Cori's Balance      =      20.00
```

9.26.2. Technical details: Constructors

- If a class does not have an explicit constructor then a constructor with no parameters with a null action is created for it. This is called the default constructor

  For example, in the case of the original class `Account` which no constructor, the system generated the following constructor.

```
public Account()
{
}
```

- If a class has a constructor the default constructor is not automatically created.

  This will cause problems (Compile time error) if you try an create an object which would call the non-existent default constructor

```
class Account
{
  private double theBalance    = 0.00;  //Balance of account
  private double theOverdraft  = 0.00;  //Overdraft

  public Account()
  {
     this( 0.00, 0.00 );    //Call constructor with 2 parameters
  }

  public Account( final double openingBalance,
                  final double overdraft )
  {
    theOverdraft = overdraft;
    theBalance   = openingBalance;
  }

  //Other methods
}
```

9.27. Key points: Classes

- Instance variables (`theBalance`, `theOverdraft`) are private.
- The constructor does not have a return type
- Methods (`getBalance, ...`) are public
- A client of the class must use a method to read or write to an instance variable
- Overloading: Methods/Constructors with same name but different signature.
- Each object (An instance of the class) has its own instance variables that may contain different values.

9.28. What you should know

- What is a class, object, method and message
- How to write a simple class containing several methods in Java. The methods to have multiple parameters and return a result.
- Why a class is important in software production
- The role of final used in a function/ method specification
- How to write simple Java applications (10-50 lines) using classes.
- The role of a constructor in a class.

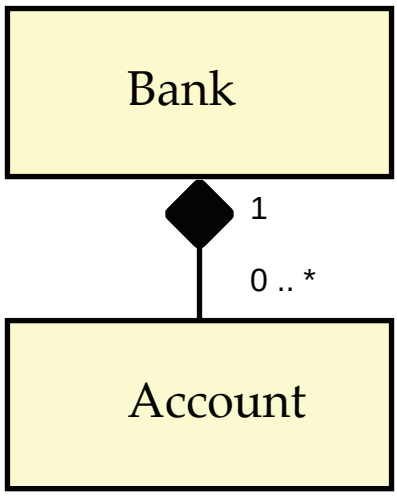
10. OO continued

- Using several classes
- Handling failure in a program
  The exception mechanism
- Dynamically increasing the size of an array.
- A more flexible alternative to an array
- UML notation

10.1. Composition / Aggregation

- Composition

  A bank contains accounts. In this case the bank has no other branches and accounts can not be transferred to another bank.

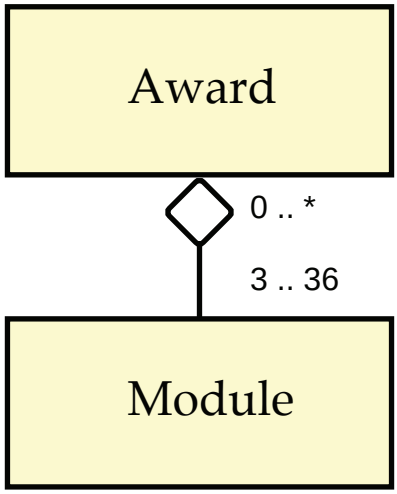


If the instance of the bank ceases to exist so will the accounts in the bank.

In the implementation the class `Bank` instances of the accounts are created inside the class `Bank` and are not referenced anywhere else.

- Aggregation

  An award contains modules.



If the instance of the award ceases to exist then the modules will still exist as they are (usually) delivered to other awards.

In the implementation of the class `Award` a reference to the modules used would be held inside the class, but these references would also be held in other awards.

It is likely that as part of the implementation of the class `Award` methods would be provided to `add`/`delete` an instance of a module.

10.2. Code to use an instance of the Bank class

```
class Main
{
  public static void main( String args[] )
  {
    Bank piggy    = new Bank();
    int mikeAN    = piggy.newAccount(); //returns account number
    int corinnaAN = piggy.newAccount();


    piggy.deposit( mikeAN, 100.00 );

    double obtained;

    System.out.printf( "Mike's Balance    = %10.2f\n",
                       piggy.getBalance(mikeAN) );

    piggy.deposit( mikeAN, 100.00 );
    System.out.printf( "Mike's Balance    = %10.2f\n",
                       piggy.getBalance(mikeAN) );

    obtained = piggy.withdraw( mikeAN, 20.00 );
    System.out.printf( "Mike has withdrawn: %10.2f\n",
                       obtained );
    System.out.printf( "Mike's Balance    = %10.2f\n",
                       piggy.getBalance(mikeAN) );


    System.out.printf( "Corinna's Balance = %10.2f\n",
                         piggy.getBalance(corinnaAN) );
  }
}
```

- Note the above code assumes that a new account will always be able to be created.

```
Mike's Balance    =     100.00
Mike's Balance    =     200.00
Mike has withdrawn:      20.00
Mike's Balance    =     180.00
Corinna's Balance =       0.00
```

10.3. Account class

```
class Account
{
  private double theBalance    = 0.00;   //Balance of account
  private double theOverdraft  = 0.00;   //Overdraft allowed

  public double getBalance()
  {
    return theBalance;
  }

  public double withdraw( final double money )
  {
    assert money >= 0.00;
    if ( theBalance - money >= theOverdraft )
    {
      theBalance = theBalance - money;
      return money;
    } else {
      return 0.00;
    }
  }

  public void deposit( final double money )
  {
    assert money >= 0.00;
    theBalance = theBalance + money;
  }


  public void setOverdraft( final double money )
  {
    theOverdraft = money;
  }

  public double getOverdraftLimit()
  {
    return theOverdraft;
  }
}
```

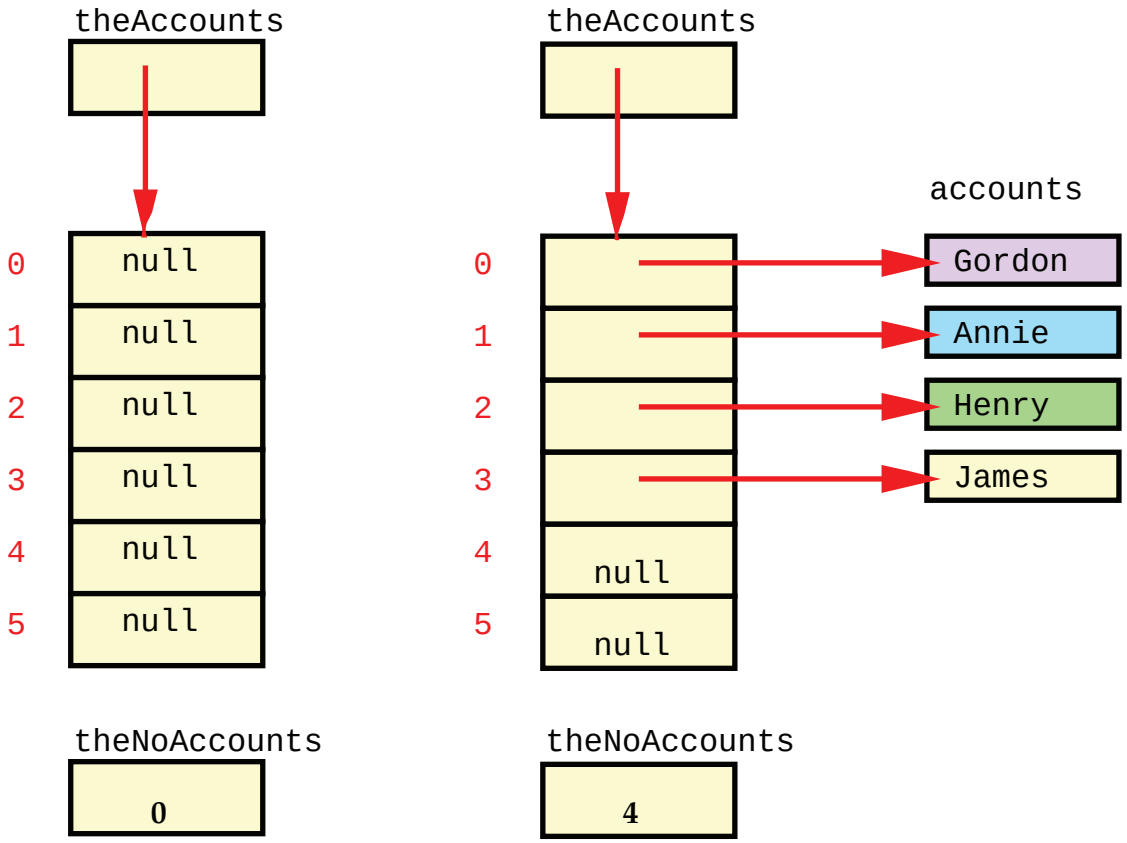
- Implementation details
  The bank contains accounts that are held in an array

  - Problems
    Have to allocate the size of the array when it is declared, so how big should the array be:
    10, 100, 1000, 10000, 100000, 1000000 .. elements.

  - The bigger (more elements) in the array the less likely there will be of running out of room to hold an additional account. However, this array will consume a large amount of memory.

    A small optimisation that we can make is only to allocate the storage for an instance of an account when a new account is requested.

10.4. Storage for the array of accounts (bank)



- `Account theAccounts[] = new Accounts[6];`

  Only allocates the array and not the individual storage for each account.
  Initially each element is set to `null`.
  No objects in the array will be created by the above declaration.

- `theAccounts[0] = new Account();`

  Allocates the storage for an individual account, in this case at element 0, then add 1 to `theNoAccounts`

10.5. Bank class

```
class Bank
{
  private static final int MAX_ACCOUNTS=6; //Maximum # of Accounts

  private Account theAccounts[] =          //Collection of Accounts
                    new Account[MAX_ACCOUNTS];
  private int theNoAccounts=0;             //# of accounts

  public int newAccount()
  {
    if ( theNoAccounts < MAX_ACCOUNTS )
    {
      theAccounts[ theNoAccounts] = new Account();
      return theNoAccounts++;              //return value then add 1
    }
    else
      return -1;                           //Hope user checks
  }

  public boolean valid( final int no )
  {
    return no >= 0 && no < theNoAccounts;
  }

  public double getBalance( final int no )
  {
    return theAccounts[no].getBalance();
  }

  public double withdraw( final int no, final double money )
  {
    return theAccounts[no].withdraw( money );
  }

      //etc
}
```

- If no more accounts can be created, return -1 as an error condition and hope the user can recover.
- It would be wrong to write an error message from the class. Why is this?

10.6. Handling error conditions

```
class Main
{
  public static void main( String args[] )
  {
    Bank piggy    = new Bank();

    int mikeAN    = piggy.newAccount();
    if ( mikeAN < 0 )
    {
       //Could not create account

    }

    int corinnaAN = piggy.newAccount();
    if ( corinnaAN < 0 )
    {
       //Could not create account

    }

  }
}
```

- It is a bit tedious to keep having to check.
- Easy to forget to check.
- How can we guarantee to prevent the program missing a failure (exceptional condition) in code that it calls.

10.7. Handling exceptional conditions

```
class Bank
{
  private static final int MAX_ACCOUNTS=6; //Maximum # of Accounts
  private Account theAccounts[] =          //Collection of Accounts
                    new Account[MAX_ACCOUNTS];
  private int theNoAccounts=0;             //Next account number

  public int newAccount() throws Exception
  {
    if ( theNoAccounts < MAX_ACCOUNTS )
    {
      theAccounts[ theNoAccounts] = new Account(); //Allocate
      return theNoAccounts++;          //Return value then add 1
    }
    else
      throw new Exception( "No space for new account" );
  }
```

- Now if we had written

```
class Main
{
  public static void main( String args[] )
  {
    Bank piggy    = new Bank();
    int mikeAN    = piggy.newAccount();
    int corinnaAN = piggy.newAccount();
  }

}
```

- It would not compile

```
s10_bank_0E1.java:6: error:
unreported exception Exception;
must be caught or declared to be thrown
```

- Now put in a **try catch** block in the client code

```
class Main
{
  public static void main( String args[] )
  {
    try
    {
      Bank piggy    = new Bank();

      for ( int i=0; i<100; i++)
      {
       int number = piggy.newAccount();
       // Process new account

      }
    }
    catch ( Exception e )
    {
      System.out.println("Error : " + e.getMessage() );
      System.out.println( "Recovering" );
    }

  }
}
```

- Forces the client of the class to take account of the failure.

```
Executing java Main - data.txt
Error : No space for new account
Recovering
```

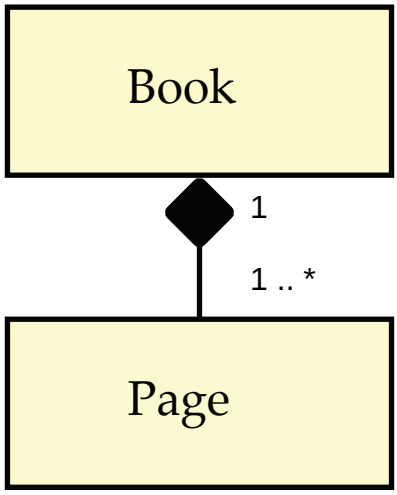
- The exception `Exception` is special in Java as no special declaration is required to say the exception is exported out of the class that contained its raising.

10.8. Quick quiz

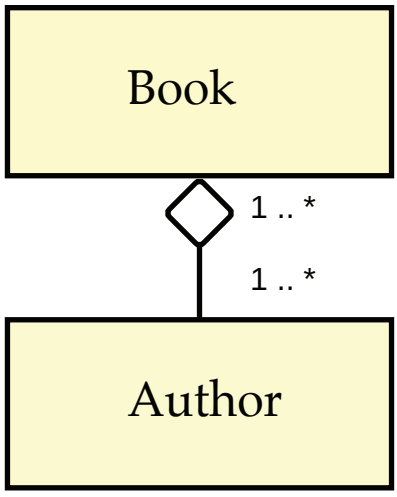
- Draw a class diagram for the relationship between a book class and a page class.
- Draw a class diagram for the relationship between a book class and an author class

10.9. Quick quiz answers

- Draw a class diagram for the relationship between a book class and a page class.



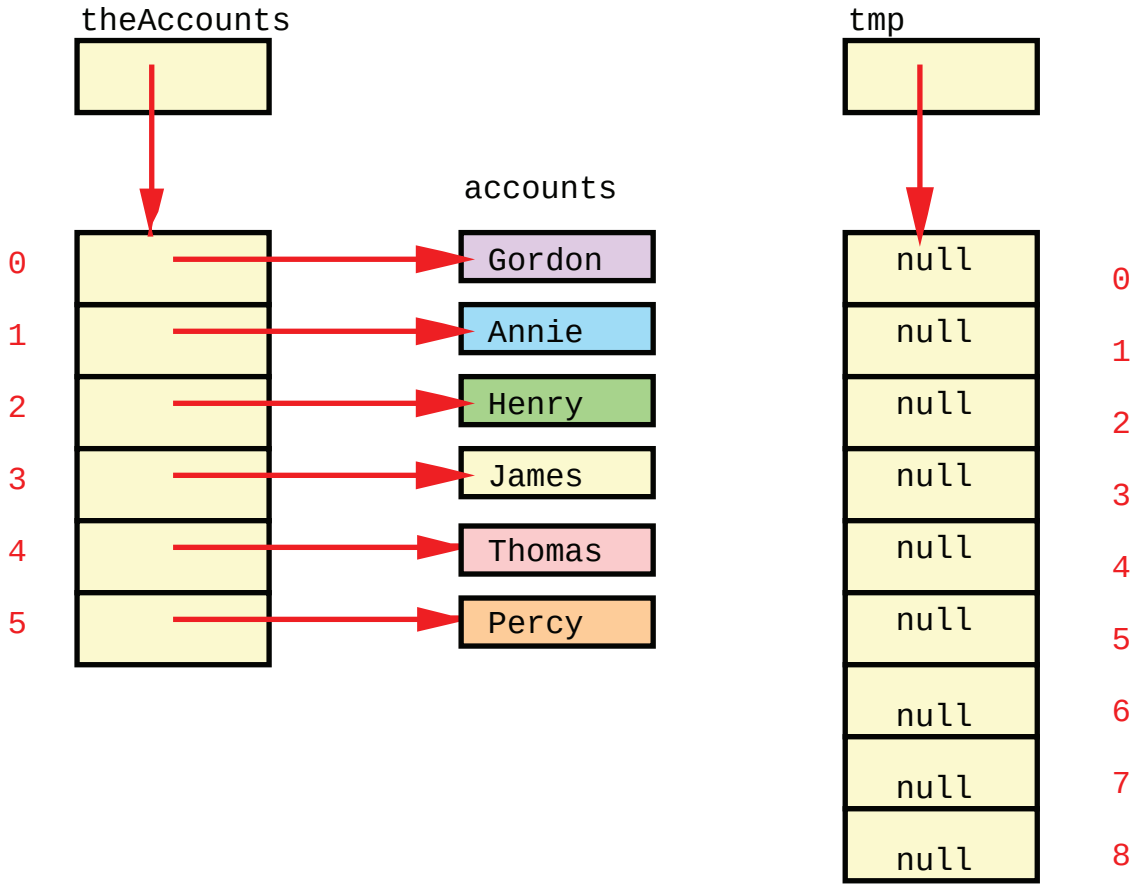
- Draw a class diagram for the relationship between a book class and an author class



10.10. A solution: to the array problem

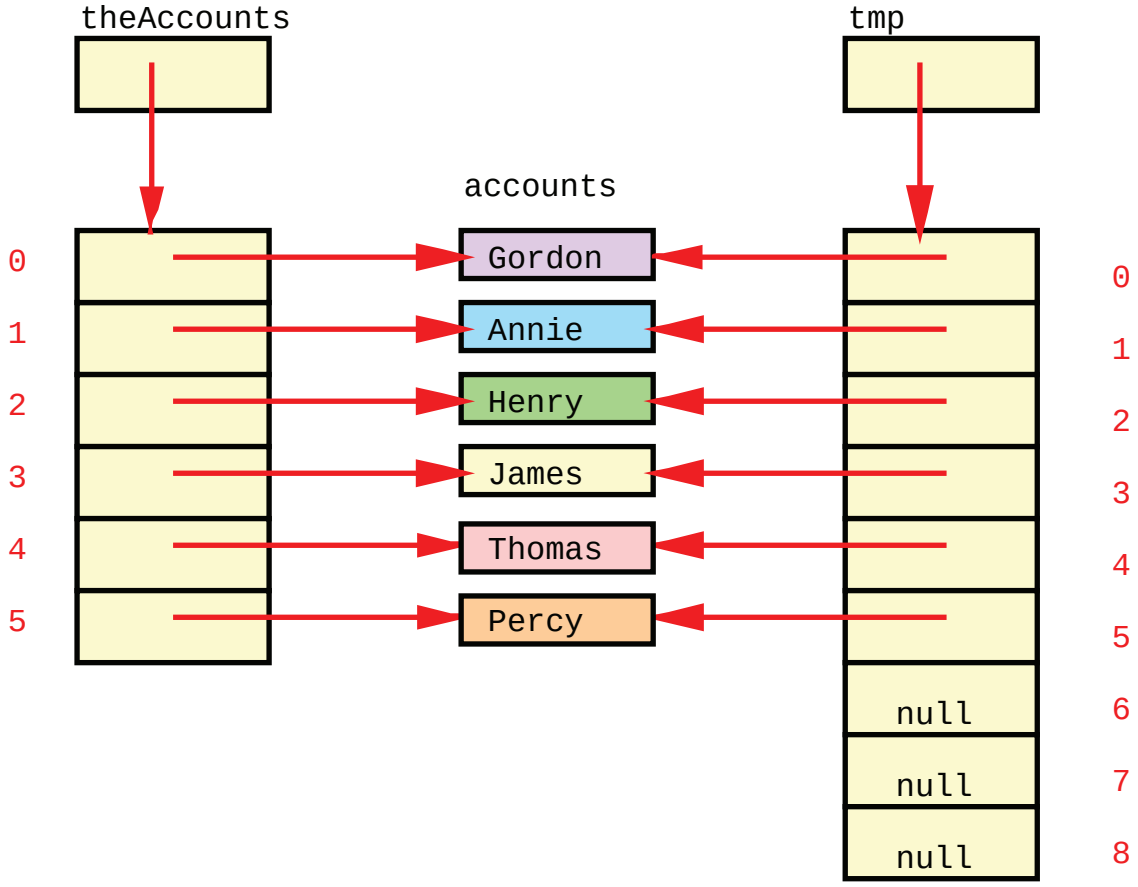
- Required to store accounts in an array, but do not know how many, and can not run out of space.

Create a new larger array (`tmp`) to hold accounts. In this case the array tmp is an extra 3 elements longer.

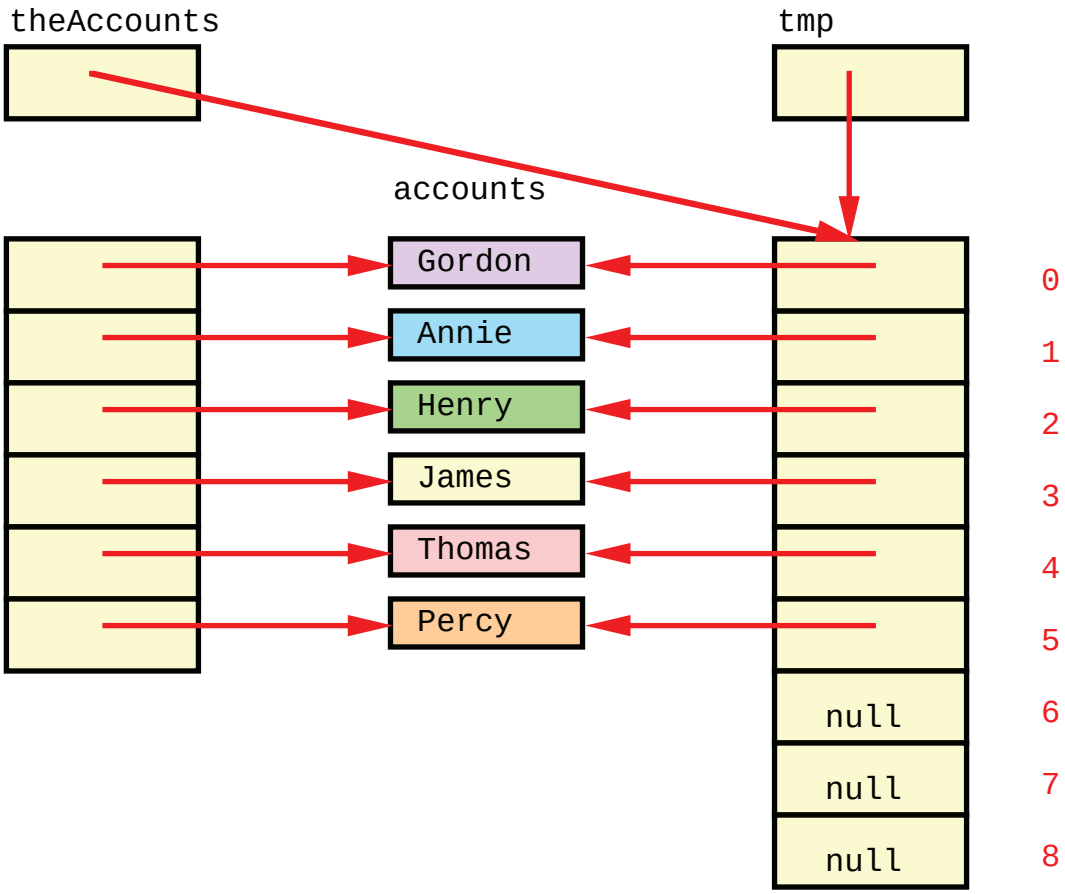


Note as `tmp` is an array of objects the initial default value is 'null'. null can be tested for and simply means there is no handle to a physical object.

Copy the handle of the accounts in the array `theAccounts` to the array `tmp` (Shallow copy)



Copy the handle in `tmp` to `theAccounts`. Now 'both' `theAccounts` and `tmp` represent the same structure.



The storage for the original array can no longer be accessed, so eventually it will be garbage collected. The variable `tmp` will cease to exist when an exit is made from the method in which it is declared.

10.11. The solution (Code)

- The new version of the method `newAccount`.

```
class Bank
{
  private static final int INCREMENT = 3; //Increase by # accounts
  private static final int INIT_SIZE = 6; //Initial size

  private int size = INIT_SIZE;            //Current size
  private Account theAccounts[] =          //Collection of Accounts
                    new Account[size];
  private int theNoAccounts=0;             //# of accounts

  public int newAccount()
  {
    if ( theNoAccounts < size )             //T All OK
    {
      theAccounts[ theNoAccounts] = new Account();
      return theNoAccounts++;
    } else {                                //F the difficult bit
      int oldSize = size;                   //Remember
      size = theNoAccounts+INCREMENT;       //new Size of array
      Account tmp[] = new Account[ size ]; //Allocate new array
      for ( int i=0; i< oldSize; i++ )
      {
         tmp[i] = theAccounts[i];           //Copy old -> new
      }
      theAccounts = tmp;                    //Now have larger array
    }
    return newAccount();                    //Just call again
                                            // to allocate account
  }

//as before
```

- Now have a bank were the number of accounts can increase to an arbitrary size. Provided of course that storage can be allocated.

10.12. Fine tuning

- It is a lot of work, (copying elements to a new array) and will be done many times if the eventual size of the array is large.

  Make the size of `INCREMENT` larger

- Make the initial size of the array `INIT_SIZE` larger.

10.13. The real solution

- Use the collection class `ArrayList`.
  Which uses many sophisticated programming techniques to make the cost [time to perform operations] of working with an arbitrary sized 'array' efficient. Thought it will not be as fast and efficient as if an array had been used.

```
class Bank
{
  private ArrayList<Account> theAccounts = //Collection
                new ArrayList<>();          // of accounts


  public boolean valid( final int no )      //Valid a/n
  {
    return no >= 0 && no < theAccounts.size();
  }

  public int newAccount()
  {
    theAccounts.add( new Account() );
    return theAccounts.size()-1;
  }

  public double getBalance( final int no )
  {
    return theAccounts.get(no).getBalance();

  }

   //etc.

}
```

- An `ArrayList` is a part of the collection classes, a series of standard library classes to hold and allow access to objects.

10.14. What you should know

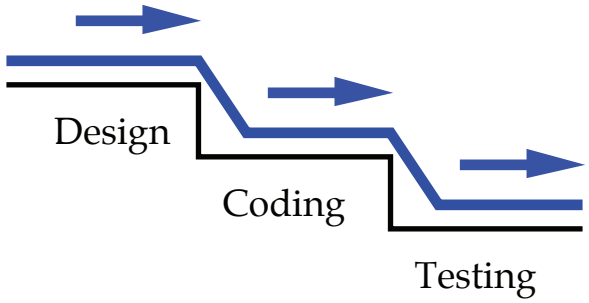
- That you need to spend at least 5 hours per week on this module in addition to attending lectures, seminars and tutorials.
- That learning a computer programming language is not an activity that you can undertake in the last few weeks of the academic year.
- Idea of splitting a program into several classes.
- The difference between the relationships of aggregation and composition.
- How an array of strings is stored in memory
- Ways of handling failure in a program

11. OO Continued

- A game in Java
- ArrayList
- Comparison Array vs. ArrayList

11.1. Program construction

Waterfall model



- 40% of effort - Requirements / Design
- 20% of effort - Coding
- 40% of effort - Testing
- Maintenance of system
  several times the total time above

\+ Find problems early / Big stable software
\- Clients change software requirements / Complex problems have interdependencies solving 1 part causes problems in another

11.2. Software fallacies / misconceptions

- My program compiles so there is little left to do
- It will be easy to remove the last few bugs from my code.
- The first thing I should do in software development is start coding.

```
public class Main
{
}
```

- Simple explanation
  Must be contained in a file called Main.java

```
class Main
{
}
```

- Simple explanation
  No special name required, but must have extension .java

11.3. The game of noughts and crosses

```
Enter move for X:
1

X |   |
---------
  |   |
---------
  |   |
```

```
Enter move for O:
2

X | O |
---------
  |   |
---------
  |   |
```

```
Enter move for X:
3

X | O | X
---------
  |   |
---------
  |   |
```

```
Enter move for O:
5

X | O | X
---------
  | O |
---------
  |   |
```

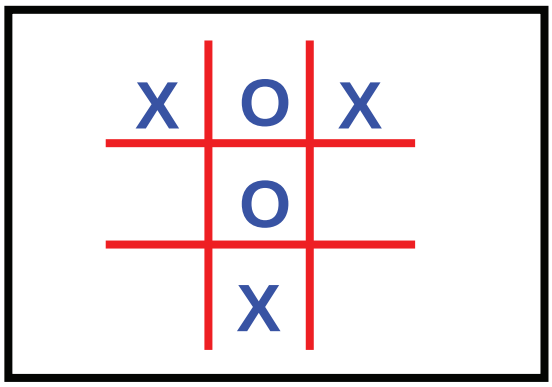
```
Enter move for X:
8

X | O | X
---------
  | O |
---------
  | X |
```

```
Enter move for O:
6

X | O | X
---------
  | O | O
---------
  | X |
```

```
Enter move for X:
4

X | O | X
---------
X | O | O
---------
  | X |
```

```
Enter move for O:
7
Its a draw

X | O | X
---------
X | O | O
---------
O | X |
```

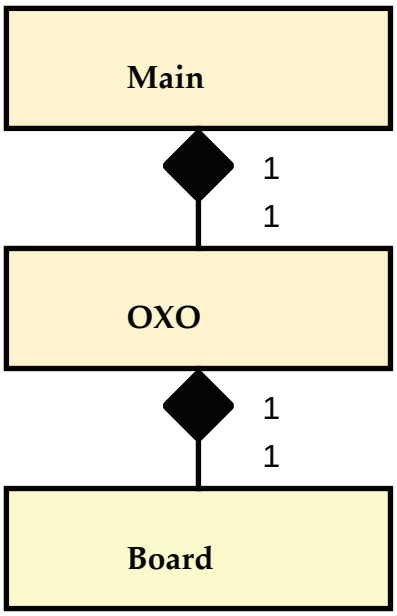
11.4. Structure



Board

playable - Game still playable?
valid - Check if a move is valid
add - Add a counter
situation - win, draw, playable ?
position - contents of cell

```
public boolean playable()
public boolean valid( final int pos )
public void add( final int pos, final char piece )
public BoardState situation()
public char position( int i )
```



Main
Start the application

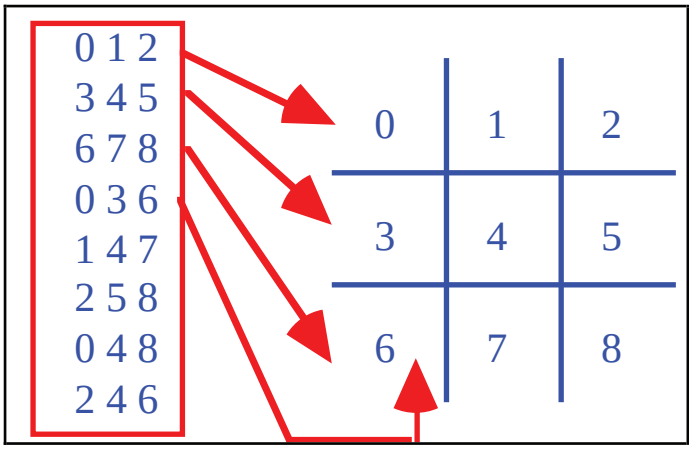
Logic of the game

Playing board

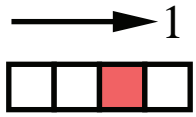
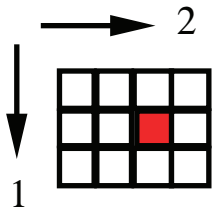
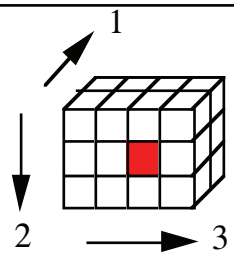
```
class Main
{
  public static void main( String args[] )
  {
    (new OXO()).newGame();
  }
}
```

```
class OXO
{
  private Board board = null;

  public void newGame()
  {
    Board.BoardState  theGameIs= Board.BoardState.PLAYABLE;
    board = new Board();

    char player = 'X';

    while ( board.playable() )
    {
       System.out.printf("Enter move for %c: ", player);
       int move = BIO.getInt();
       if ( board.valid( move ) )
       {
          board.add( move, player );
          switch( board.situation() )
          {
            case  WON  :
              System.out.printf("Player %c wins\n", player );
              break;
            case  DRAWN :
              System.out.printf("Its a draw\n" );
              break;
            default     :
              player = (player == 'X') ? 'O' : 'X' ;
          }
          drawBoard();
       } else {
         System.out.printf("Invalid move try again\n" );
       }
    }
  }
```

```
  private void drawBoard()
  {
    System.out.print( "\n" +
      returnLine( 1 ) + "\n---------\n" +
      returnLine( 4 ) + "\n---------\n" +
      returnLine( 7 ) + "\n" );
  }

  private String returnLine( final int pos )
  {
    return board.position(pos)   + " | " +
           board.position(pos+1) + " | " +
           board.position(pos+2);
  }

}
```



Each row of the 2D array represents the position of the three cells in one of the eight win lines.
Note: in Java arrays start at element 0.

Declaration of array handle	Conceptual model of storage	Allocation of physical storage for array an access to shaded element
`int vector[];`		`vector = new int[4];` `int v = vector[2];`
`int table[][];`		`table = new int[3][4];` `int v = table[1][2];`
`int cube[][][];`		`cube = new int[2][3][4];` `int v = cube[0][1][2];`

```
class Board
{
  public  enum BoardState { WON, DRAWN, LOSE, PLAYABLE };
  private char    theSqrs[];                //Playing grid
  private int     theMoves;                 //Moves made
  private boolean theGamePlayable = true;   //Game finished

  private static final int SIZE_TTT = 9;    //Squares on board

  public Board()
  {
    theSqrs = new char[SIZE_TTT];           //Allocates storage
    for ( int i=0; i<theSqrs.length; i++ ) //Populate with ' '
    {
      theSqrs[i] = ' ';
    }
    theMoves = 0;                           //0 moves so far
  }

  public boolean valid( final int pos )     //Is move valid
  {
    return (pos>=1 && pos<=SIZE_TTT) &&     //Between 1 - 9
           theSqrs[pos-1] == ' '     &&     // and not made
           theGamePlayable;                 //
  }

  public void add( final int pos, final char piece )
  {
    theSqrs[ pos-1 ] = piece;               //Add to board
    theMoves++;                             //Increment moves
  }

  public char position( int i )             //Return contents
  {                                         // of square i (1-9)
    return theSqrs[i-1];                    // on the board
  }

  public boolean playable()
  {
    return theGamePlayable;
  }
```

```
  public BoardState situation()              //Current game state
  {
    int WL = 8;                              //Number of win lines
    int win[][] =                            //All 8 win lines
      { {0,1,2}, {3,4,5}, {6,7,8},           // 3 cell positions
        {0,3,6}, {1,4,7}, {2,5,8},           //
        {0,4,8}, {2,4,6} };
    int drawn = 0;                           //Count of lines drawn
    for ( int i=0; i<WL; i++ )               //For each win line
    {
      char c1 = theSqrs[ win[i][0] ];        // First cell in a pos
      if ( c1 != ' ' &&                      //  win line occupied so
           c1 == theSqrs[ win[i][1] ] &&     //  if same as 2nd and
           c1 == theSqrs[ win[i][2] ] )      //  3rd cell ?
      {                                      // T
        theGamePlayable = false;             //  Found a win line
        return BoardState.WON;               //  Game won
      }                                      //
      boolean isX = false, isO = false;
      for ( int j=0; j<3; j++ )
      {
        char curSquare = theSqrs[ win[i][j] ];
        if ( curSquare == 'X' ) isX = true;
        if ( curSquare == 'O' ) isO = true;
      }
      if ( isX && isO ) drawn++;             //Drawn line
    }

    if ( drawn >= 8 )                        //Lines drawn
    {                                        //T
      theGamePlayable = false;               //Game drawn
      return BoardState.DRAWN;
    }

    return BoardState.PLAYABLE;              //Still playable
  }

}
```

11.5. What you should know

- That you need to spend at least 5 hours per week on this module in addition to attending lectures, seminars and tutorials.
- That learning a computer programming language is not an activity that you can undertake in the last few weeks of the academic year.
- That program construction does not start with writing code.
- That coding (writing code) is only a small part of program construction.
- The fundamental differences between an `ArrayList` and an array.

12. Review/ reflection/ revision/ + Java

- As a desk calculator
- Loops
- Making decisions
- functions / methods
- Arrays
- Collections

12.1. Core ideas in an imperative programming language

- Sequence

```
double pricePerKilo = 1.20;

System.out.print( "#Input Kilos of apples: " );
double kilosOfApples = BIO.getDouble();

double cost = pricePerKilo * kilosOfApples;

System.out.print( "Cost of apples is (GBP) " );
System.out.println( cost );
```

- Iteration

```
int count = 1;

while ( count <= 5 )
{
  System.out.println( "Hello" );
  count = count + 1;
}
```

- Selection

```
if ( month == 2 )
   System.out.println("The month is February");
 else
   System.out.println("The month is not February");
```

- Abstraction

```
public static void main( String args[] )
{
  printTree( 9 );
}

public static void printTree( int maxFoliageHeight )
{
  printFoliage( maxFoliageHeight );
  printTreeTrunk( maxFoliageHeight );
}
```

12.2. As a desk calculator

- Printing values

```
System.out.println( 27 );
System.out.println( 27 + 2 );
System.out.println( 2 + 3 * 4 );
```

```
27
29
14
```

```
System.out.print( "Hello" );
System.out.println( " world" );
```

```
Hello world
```

```
System.out.print( 27 );
System.out.println( 27 + 2 );
```

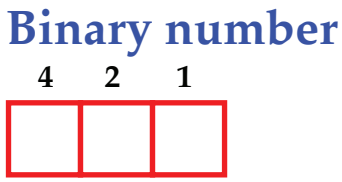
```
2729
```

```
System.out.printf( "%d * %d = %d", 2, 3, 6 );
System.out.println();
```

```
2 * 3 = 6
```

- Whole numbers

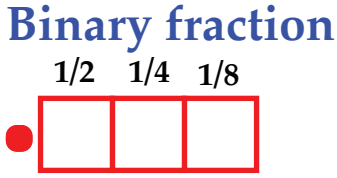
Binary	Decimal
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7



  - Exact translation from a binary to a decimal number and vice versa

- Fractions

Binary	Decimal
.000	.000
.001	.125
.010	.250
.011	.375
.100	.500
.101	.625
.110	.750
.111	.875

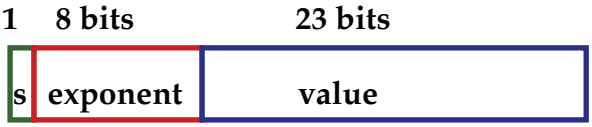


  - Not always an exact translation for all fractions to a decimal or binary number.

- Scientific notation

7	$7.000 * 10^0$	7.000	1	Shift . 0 place to right
15.2	$1.520 * 10^1$	1.520	1	Shift . 1 places to right
179	$1.790 * 10^2$	1.790	2	Shift . 2 places to right

- Representation of a float in binary



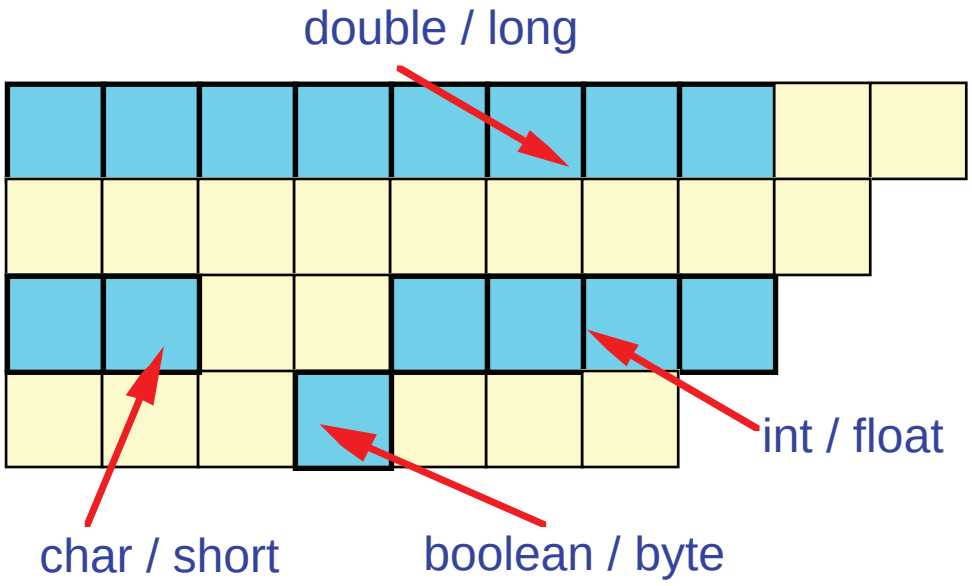
value	is between 1.000... & 1.111... (binary)
exponent	is where the decimal place goes, how many places to shift left or right.
s(ign)	number +ve or -ve

  - As the value is between 1.000... & 1.111... the leading 1 does not need to be stored.
  - In many cases the number held in a `float` will only be an approximation to the actual number that the user wishes to be stored.
  - In many cases when performing arithmetic with an instance of a float some precision will be lost (rounding errors).
  - Is this important?
    Yes, if you are working with money, etc. where absolute precision is required.

- Memory locations (variables) used to store values

```
int     aWholeNumber = 27;
double  aNumberWithDecimalPlaces = 27.1234;

char    aCharacterFromTheUnicodeCharacterSet = 'A';

boolean aLogicalValue = true;
```

Memory organised as 'array' of bytes (8 bits)



- Representation is different.
  - `int` 32 bit 2's compliment number
  - `double` 64 bit IEEE 754 standard
  - `char` 16 bit Unicode character
- The Java convention is that variable names (of mutable variables) always start with a lower case letter.

- Java is a strongly typed language.

```
int    rainfall = 27.2;  // Will not compile
```

```
double rainfall = 27;    // Fine 27 widening conversion
                         // to 27.0
```

  - Widening conversion (Promotion) of an `int` to a `double` occurs.

12.3. Simple calculation

```
double pricePerKilo  = 1.20;
double kilosOfApples = 5.2;

double cost = pricePerKilo * kilosOfApples;

System.out.printf( "Cost of apples is (GBP) %5.2f\n",
                   cost );
```

12.3.1. A simple program (straight line code)

```
class Main
{
  public static void main( String args[] )
  {
    double pricePerKilo  = 1.20;                    //Variables
    double kilosOfApples = 5.2;
    double cost = pricePerKilo * kilosOfApples;     //Calculate

    System.out.printf( "Cost of apples is (GBP) %5.2f\n",
                                  cost );           // answer
  }
}
```

```
Cost of apples is (GBP)  6.24
```

- `javac`
  Converts the source program into bytecode
- `java`
  Obeys the bytecode instructions to run the program
- `BlueJ`
  Is simply a visual front end that when you press compile runs the program `javac` and when you ask to execute your code runs the program `java`.

12.4. Input

- BIO
  A very simple class library that allows the input of an integer value, a double value, and an instance of a string

```
double kilosOfApples = BIO.getDouble();
```

Will input into the variable `kilosOfApples` a number typed by the user.

```
class Main
{
  public static void main( String args[] )
  {
    double pricePerKilo = 1.20;                    //Price

    System.out.print( "#Input Kilos of apples: " );
    double kilosOfApples = BIO.getDouble();        //How many

    double cost = pricePerKilo * kilosOfApples;    //Calculate

    System.out.print( "Cost of apples is (GBP) " ); //Result
    System.out.println( cost );
  }
}
```

```
#Input Kilos of apples: 10
Cost of apples is (GBP) 12.0
```

12.5. Iteration

- Repeat a section of code a number of times

```
for ( int count=1; count <= 5; count++ )
{
  System.out.println( "Hello" );// Print Hello
}
```

e.g. print n stars on a line

```
for ( int count=1; count <= n; count++ )
{
  System.out.print( "*" );
}
System.out.println();
```

- Repeat a section of code till a condition is met

```
  System.out.print("#Enter data     : " );
  String  data = BIO.getString();

  while ( ! data.equals("STOP")  )
  {
     // Process

     System.out.print("#Enter data     : " );
     data = BIO.getString();
   }
```

12.6. Selection

```
if ( cost <= 5.00 )
  System.out.println("Cheap");
else
  System.out.println("Not so cheap");
```

```
if ( cost <= 5.00 )
{
  System.out.println("Cheap");
} else {
  System.out.println("Not so cheap");
}
```

```
int month = 2;

switch ( month )
{
  case 1 :
    System.out.print("January");
    break;
  case 2 :
    System.out.print("February");
    break;
  case 3 :
    System.out.print("March");
    break;
  default:
    System.out.print( "Not January, February or March" );
}
System.out.println();
```

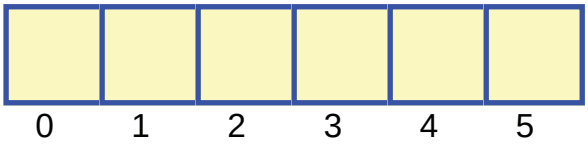
12.7. functions / methods

```
class Main
{
  public static void main( String args[] )
  {
    System.out.print("#Enter year       : " );
    int year = BIO.getInt();

    while ( year > 0 )
    {
      System.out.printf( "%4d was%s a leap year\n",
                         year, leapYear( year ) ? "" : " NOT" );

      System.out.print("#Enter year       : " );
      year = BIO.getInt();
    }
  }

  public static boolean leapYear( int year )
  {
    if ( year % 4 == 0 )
    {
      if ( year % 100 != 0 || year % 400 == 0 )
      {
        return true;
      } else {
        return false;
      }
    }
    return false;
  }
}
```

```
1900
2000
2015
2016
2017
```

```
1900 was NOT a leap year
2000 was a leap year
2015 was NOT a leap year
2016 was a leap year
2017 was NOT a leap year
```

12.8. Data structure: Array : a reference type



Collection of like things
(instances of primitive types or reference types),
items selected from the collection by a numeric index in the range `0 .. length-1`.

Advantages

- Random access good
- Sequential access good

Problems

- Can not extend (Easily)
- Can not remove element (Easily)

12.9. functions / methods

```
class Main
{
  public static void main( String args[] )
  {
    int years[] = {1752, 1899, 1900, 1901, 2000, 2015, 2016, 2017};

    for ( int i=0; i<years.length; i++)
    {
      System.out.printf( "%4d was%s a leap year\n",
                         years[i],
                         leapYear( years[i] ) ? "" : " NOT" );
    }
  }

  public static boolean leapYear( int year )
  {
    if ( year % 4 == 0 )
    {
      if ( year % 100 != 0 || year % 400 == 0 )
      {
        return true;
      } else {
        return false;
      }
    }
    return false;
  }
}
```

```
1752 was a leap year
1899 was NOT a leap year
1900 was NOT a leap year
1901 was NOT a leap year
2000 was a leap year
2015 was NOT a leap year
2016 was a leap year
2017 was NOT a leap year
```

Alternatively leapYear could have been written as:

```
public static boolean leapYear( int year )
{
  return  year%4 == 0 && ( year%100 != 0 || year%400 == 0 );
}
```

1899	1900	1901	1996	2000	2004	year	
	✓		✓	✓	✓	year%4 == 0	&&
✓		✓	✓		✓	year%100 != 0	||
				✓		year%400 == 0	

	✓		✓	✓	✓	year%4 == 0	&&
✓		✓	✓	✓	✓	year%100 != 0 || year%400 == 0	

			✓	✓	✓	leapYear(year)

lazy evaluation

If year%4 == 0 is false, then the whole result is taken as false

12.10. Yet another `Account` class

```
public class Account
{
  private double theBalance    = 0.00;   //Balance of account
  private String theName       = "";

  public Account( String name, double openningBalance )
  {
    theBalance = openningBalance;
    theName    = name;
  }

  public double getBalance()
  {
    return theBalance;
  }

  public String getName()
  {
    return theName;
  }

  public double withdraw( final double money )
  {
    if ( theBalance >= money  )
    {
      theBalance = theBalance - money;
      return money;
    } else {
      return 0.00;
    }
  }

  public void deposit( final double money )
  {
    theBalance = theBalance + money;
  }
}
```

12.11. Calculate total money in all accounts

```
class Main
{
  public static void main( String args[] )
  {
    Account accounts[] = new Account[5];

    accounts[0] = new Account( "Gordon", 100.0 );
    accounts[1] = new Account( "James",  100.0 );
    accounts[2] = new Account( "Edward", 100.0 );
    accounts[3] = new Account( "Percy",  100.0 );
    accounts[4] = new Account( "Thomas", 100.0 );

    double total = 0.0;
    total = total + accounts[0].getBalance();
    total = total + accounts[1].getBalance();
    total = total + accounts[2].getBalance();
    total = total + accounts[3].getBalance();
    total = total + accounts[4].getBalance();

    System.out.printf( "Total amount of money = %10.2f\n", total );
  }
}
```

```
Total amount of money =     500.00
```

```
class Main
{
  public static void main( String args[] )
  {
    Account accounts[] = new Account[5];

    accounts[0] = new Account( "Gordon", 100.0 );
    accounts[1] = new Account( "James",  100.0 );
    accounts[2] = new Account( "Edward", 100.0 );
    accounts[3] = new Account( "Percy",  100.0 );
    accounts[4] = new Account( "Thomas", 100.0 );

    double total = 0.0;
    for ( int i=0; i < accounts.length; i++ )
    {
      total = total + accounts[i].getBalance();
    }

    System.out.printf( "Total amount of money = %10.2f\n", total );
  }
}
```

```
Total amount of money =     500.00
```

Alternate `for`

```
    double total = 0.0;
    for ( Account acc : accounts )
    {
      total = total + acc.getBalance();
    }
```

12.12. ArrayList

```
ArrayList<Account> accounts = new ArrayList<>(0);
accounts.add( new Account( "Gordon", 100.0 ) );
accounts.add( new Account( "James",  100.0 ) );
accounts.add( new Account( "Edward", 100.0 ) );
accounts.add( new Account( "Percy",  100.0 ) );
accounts.add( new Account( "Thomas", 100.0 ) );

double total = 0.0;
for ( int i=0; i<accounts.size(); i++ )
{
  total = total + accounts.get( i ).getBalance();
}

 System.out.printf( "Total amount of money = %10.2f\n", total );
```

```
Total amount of money =     500.00
```

- Alternate `for`

```
double total = 0.0;
for ( Account acc : accounts )
{
  total = total + acc.getBalance();
}
```

12.13. Replace an item in the ArrayList

```
ArrayList<Account> accounts = new ArrayList<>(0);
accounts.add( new Account( "Gordon", 100.0 ) );
accounts.add( new Account( "James",  100.0 ) );
accounts.add( new Account( "Edward", 100.0 ) );
accounts.add( new Account( "Percy",  100.0 ) );
accounts.add( new Account( "Thomas", 100.0 ) );

accounts.set( 4, new Account( "Henry", 200.0 ) );

double total = 0.0;
for ( int i=0; i<accounts.size(); i++ )
{
  total = total + accounts.get( i ).getBalance();
}

System.out.printf( "Total amount of money = %10.2f\n", total );
```

```
Total amount of money =     600.00
```

12.14. Remove an item in the ArrayList

```
ArrayList<Account> accounts = new ArrayList<>(0);
accounts.add( new Account( "Gordon", 100.0 ) );
accounts.add( new Account( "James",  100.0 ) );
accounts.add( new Account( "Edward", 100.0 ) );
accounts.add( new Account( "Percy",  100.0 ) );
accounts.add( new Account( "Thomas", 100.0 ) );

accounts.remove( 3 );

double total = 0.0;
for ( int i=0; i<accounts.size(); i++ )
{
  total = total + accounts.get( i ).getBalance();
}

System.out.printf( "Total amount of money = %10.2f\n", total );
```

```
Total amount of money =     400.00
```

12.15. ArrayList vs. Array

Metric on collection	ArrayList	Array
Easy to add an extra item	✓	✗
Easy to remove an item	✓	✗
Can store objects and instances of primitive types	✗ **[1]**	✓
Fast access to random item	✓ (slower)	✓
Syntax is the same	✗	✗
Syntax elegance when using the collection (personal view)	inelegant at times	Elegant & concise

[1]

```
ArrayList<Double> ages;
ages = new ArrayList<>();

ages.add( 22.4 );
```

However, the above works as the Java 'compiler' generates hidden extra code to (box) put the double 22.4 in an instance of the wrapper class `Double` and stores this object in the instance of the ArrayList `ages`.

```
double firstAge = ages.get(0);
```

Works as the Java 'compiler' generates hidden extra code to take the double out of the instance of the wrapper class `Double` (Unbox).

12.16. ArrayList vs. Array

Syntax to	ArrayList	Array
add at end add at position 2	`c.add(item);` `c.add(2,item);`	✗ ✗
delete item delete item @ pos. 2	`c.remove(item);` `c.remove(2);`	✗ ✗
overwrite @ pos 2.	`c.set(2, item);`	`c[2]=item;`
get item at pos. 2	`item=c.get(2);`	`item=c[2];`
find # elements	`c.size();`	`c.length;`

- `remove` & `set` also returns removed item

  For `c.remove(item);` the first item for which

```
item == null
   ? c.get(i)==null
   : item.equals( c.get(i) )
```

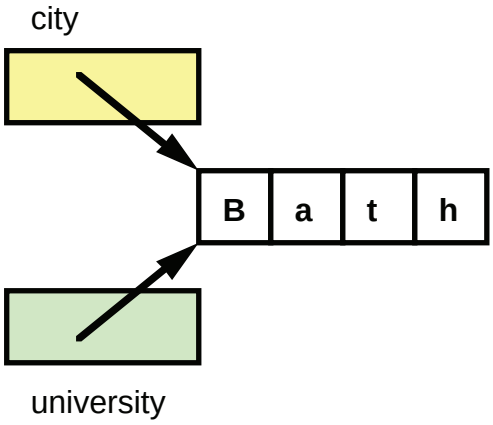
  is true is removed.

Be careful the method `equals` may not always be what you expect. It could compare the actual contents of the object (deep equality) , or it may be a default implementation which simply compares handles (Shallow equality)

For an instance of a String all is ok

Shallow equality

```
String city = "Bath";
String university = city;


if (city==university)
  // is True
if(city.equals(university))
  // is True
```

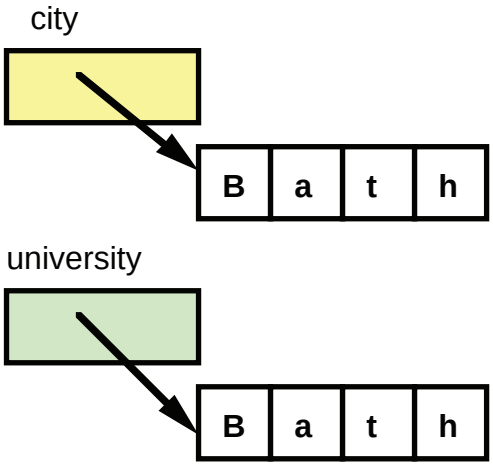



```
String city = "Bath";
String university ="Bath";
```

As strings are immutable the compiler if it can will share strings.

Deep equality

```
String city = "Bath"
String university =
   new String(city);


if (city==university)
  // is False
if(city.equals(university))
  // is True
```



12.17. Widening of variables

Items of type	May be widened to any of the following types
byte	short int long float double
short	int long float double
char	int long float double
int	long float double
long	float double
float	double

This will happen when for example an int is assigned to an instance of a double.

```
double aDoubleValue = 1;                    // Will widen 1 -> 1.0
```

? Is it good or bad to have widening

Think about

```
class Main
{
  public static void main( String args[] )
  {
     int   i = 2_000_000_123;
     float f = i;
     System.out.printf("int = %10d float= %10.0f\n", i, f );

     i = 2_123_456_789;
     f = i;
     System.out.printf("int = %10d float= %10.0f\n", i, f );
  }
}
```

```
int = 2000000123 float= 2000000128
int = 2123456789 float= 2123456768
```

12.18. Narrowing conversions

- Cast

  Used to forces a narrowing conversion between types

  Use of a cast:

```
int    anIntValue = (int) 7.6;          // Will compile
```

  - 7.6 truncated to 7
    If you want to round the value add 0.5 first.

The Java compiler will not automatically perform the narrowing of the result of an expression so that it may be assigned.

For example:

Items of type	May be narrowed to any of the following types by using a cast. e.g. `(int) a_double;`
short	byte char
char	byte short
int	byte short char
long	byte short char int
float	byte short char int long
double	byte short char int long float

- Rank

Primitive type	Rank
byte	Low
short / char	
int	
long	
float	
double	High

- In an expression, instance of a **byte**, **short**, **char** promoted to an **int**;
  If the highest ranked type used is a:
  long rest are promoted to **long**
  float rest promoted to **float**
  double rest promoted to **double**

- **Warning** the widening conversion of an int to a float or a long to a double may lose precision.

12.19. Binary/ dyadic numeric promotions

Consider `Operand1 op Operand2`

If either `Operand1` or `Operand2` is a:

`double`	then the other operand is converted to a	`double`
`float`	then the other operand is converted to a	`float`
`long`	then the other operand is converted to a	`long`

otherwise both operands are converted to an `int`. For example, with the following declarations of instances of the standard types:

```
byte    a_byte;
int     an_int;
float   a_float;
```

The effect of the following expressions is as follows:

Expression	Results in
`a_byte + an_int`	The contents of `a_byte` promoted to an `int`.
`a_byte + a_byte`	The contents of both operands promoted to an `int`.
`a_float + a_byte`	The contents of a `a_byte` promoted to a `float`.

This means that any operation delivering a numeric result involving instances of `byte`, `short` or `char` will result in the delivery of an `int` value as the result of the computation.

Assignment expression	Results in
`an_int = a_byte+b_byte`	`a_byte` and `b_byte` are promoted to an `int` and the result is assigned to an_int.
`a_double = a_byte+b_byte`	`a_byte` and `b_byte` are promoted to an `int` and the result is widened to a `double`

12.20. Summary of binary/ dyadic operators "a + b"

		Allowed on instances of				
Operator	Description	boolean	char	byte / short	int / long	float / double
+	Addition		✓ [1]	✓ [1]	✓	✓
-	Subtraction		✓ [1]	✓ [1]	✓	✓
/	Division		✓ [1]	✓ [1]	✓	✓
*	Multiplication		✓ [1]	✓ [1]	✓	✓
%	Remainder		✓ [1]	✓ [1]	✓	✓ [2]
<<	Left shift		✓ [1]	✓ [1]	✓	
>>	Right shift [3]		✓ [1]	✓ [1]	✓	
>>>	Right shift [4]		✓ [1]	✓ [1]	✓	
&	bitwise and		✓ [1]	✓ [1]	✓	
|	bitwise or		✓ [1]	✓ [1]	✓	
^	bitwise xor		✓ [1]	✓ [1]	✓	
&&	logical and	✓				
||	logical or	✓				

Key:
- ✓ Allowed.
- [1] Both sides promoted to an `int` before the operation is performed. Thus an `int` result is returned from the expression.
- [2] The floating point remainder.
- [3] The right shift operator `>>` propagates the sign bit.
- [4] The right shift operator `>>>` does **not** propagate the sign bit.

12.21. Summary of monadic arithmetic operators "+a"

		Allowed on instances of				
Operator	Description	boolean	char	byte / short	int / long	float / double
+	Monadic plus		✓ [1]	✓ [1]	✓	✓
-	Negation		✓ [1]	✓ [1]	✓	✓

Key:
- ✓ Allowed.
- [1] The result will be converted to an `int`.

12.22. What you should know

- That you need to spend at least 5 hours per week on this module in addition to attending lectures, seminars and tutorials.

- That learning a computer programming language is not an activity that you can undertake in the last few weeks of the academic year.

- The first semester exam will soon take place, it is vital that you are familiar with the system that will be used in the exam.

  Look in the exam area (link `About exam`) of the on-line notes

Semester 2

CI101

Java

Object oriented programming

13. Java summary

Core

`char int double boolean`
 	
`short long float`
	primitive types;

`int a = 2;` `int b = 2, c = 4;`	Declaration of a variable

`a = b;` `a = b + c;`	Assignment

`b + c;  // Addition`
`b - c;  // Subtraction`
`b * c;  // Multiplication`
`b / c;  // Division`
`b % c;  // Delivers remainder`
	Arithmetic operators

`class Main`
`{`
`  public static void`
`          main(String args[])`
`  {`
`    System.out.println("Hello");`
`  }`
`}`
	Complete Java application

Iteration

```
int i = 1;
while ( i <= 10 )
{
  // same code
  i++;
}
```

while loop / counting

repeats code 10 times varying i from 1 .. 10.

```
String name = BIO.getString();

while ( !name.equals("END") )
{
  // same code

  name = BIO.getString();
}
```

while loop /
end on condition

keep reading in a string until the string read in is equal to "END"

```
for (int i=1; i<=10; i++)
{
  // same code
}
```

for loop / counting
repeats code 10 times varying i from 1 .. 10.

Decision

```
if ( price < 10.00 )
{
  // Buy
}
```

if making a decision

```
if ( price < 10.00 )
{
  // Buy
} else {
  // Do not buy
}
```

if making a decision with else part.

```
switch ( month )
{
  case 1 : // Month is Jan
           break;
  case 2 : // Month is Feb
           break;

}
```

switch
An alternate way of writing if.
break must be used to exit, otherwise control will drop to the next case label.

```
switch ( month )
{
  case 1 : // Month is Jan
           break;
  case 2 : // Month is Feb
           break;

  default : // not Jan Feb

}
```

switch

With a default case which is matched when there is no match to a case label.

Arrays

```
class Main
{
  public static void main( String args[] )
  {
    int peopleInRoom[] = new int[3];

    //Allocate a new array initialised
    // of a different size
    peopleInRoom = new int[] {20,30,15,40 };

    //conventional for loop
    for ( int i = 0; i<peopleInRoom.length; i++ )
    {
      System.out.printf("Room %d has %d people\n",
                        i, peopleInRoom[i] );
    }

    //foreach loop
    int totalPeople = 0;
    for ( int people: peopleInRoom )
    {
      totalPeople += people;
    }

    System.out.printf("Total = %d\n", totalPeople );

    String colours[] =
      new String[] { "Red", "Green", "Blue" };

    //for eachloop
    for ( String colour: colours )
    {
      System.out.printf( "%s ", colour );
    }
    System.out.println();
  }
}
```

```
Room 0 has 20 people
Room 1 has 30 people
Room 2 has 15 people
Room 3 has 40 people
Total = 105
Red Green Blue
```

Class, Method, Object, Message

`class NoticeBoard`
`{`
 	
`  private String message="";`
 	
`  public NoticeBoard( String aMes )`
`  {`
`  setMessage( aMes );`
`  }`
 	
`  public void setMessage(String mes)`
`  {`
`  message = mes;`
`  }`
 	
`  public String getMessage()`
`  {`
`  return message;`
`  }`
`}`
	Class encapsulating code & data.
	
	Instance variable
	
	Constructor
	Sets the first message on the notice board.
	
	Method to set a message on an instance of a NoticeBoard.
	
	Method to return the message displayed on the notice board.

`NoticeBoard wall =`
`  new NoticeBoard( "Blank" );`
 	
`String read = wall.getMessage();`
 	
`wall.setMessage("Do not Disturb");`
`read = wall.getMessage();`
	Creation of an instance of the class `NoticeBoard`
	
	Send message `getMessage` to object `wall`.
	
	Set a new message, and retrieve from object `wall`

An example class

```
class Account
{
  //private   visible only within class
  //protected visible only in a class and subclass
  //public    visible to all who can see object

  private double theBalance     = 0.0d;   //Balance of account

  public Account()                        //Constructor
  {
    theBalance = 0.00;
  }

  public Account( double openingBalance ) //Constructor
  {
    theBalance = openingBalance;
  }

  final public double getBalance()  //Method may not be overridden
  {
    return theBalance;
  }

  public double withdraw( final double money ) //money is read only
  {
    if ( theBalance - money >= 0.00 )
    {
      theBalance = theBalance - money;
      return money;
    } else {
      return 0.00;
    }
  }

  public void deposit( final double money )
  {
    theBalance = theBalance + money;
  }
}
```

Collection class `ArrayList`

```
class Main
{
  public static void main( String args[] )
  {
    ArrayList<Integer> peopleInRoom = new ArrayList<>();
    //Populate an ArrayList
    peopleInRoom.add( 20 ); //Implicit boxing
    peopleInRoom.add( 30 );
    peopleInRoom.add( 15 );
    peopleInRoom.add( 40 );

    //foreach loop to sum # of people in peopleInRoom
    //Note unboxing of values in ArrayList
    int totalPeople = 0;
    for ( int people: peopleInRoom )
    {
      totalPeople += people;
    }
    System.out.printf( "Total = %d\n" , totalPeople );

    //Populate an ArrayList
    ArrayList<String> colours =
      new ArrayList<>( Arrays.asList( "Red", "Green", "Blue" ) );

    //foreach loop to print colours in an ArrayList
    for ( String colour: colours )
    {
      System.out.printf( "%s " , colour );
    }
    System.out.println();
  }
}
```

1. GUI

- History
- GUI in Java
- Demonstration of coursework 4
- Demonstration of testing a class

1.1. Historic view

1970's

Originally many mainframe computers used a teleprinter (teletype or tty) as the output device from a multi-user computer. (Printing on a continuos roll of paper at 10 characters a second)

Then VDU's (Visual Display Unit) displayed characters on a CRT screen at a faster rate 10-960 (110 - 9600 baud) characters per second.

1973 Xerox Alto
GUI, mouse, ethernet (Not commercial product)

1979 Many people in the 1970's saw the Xerox Alto, including Steve Jobs (Apple) in 1979

1980's

WYSIWYG What You See Is What You Get
WIMP Window Icon Menus Pointer

1981 Xerox Star workstation
[First commercial system to have GUI, mouse, Ethernet]
Xerox 8010 Information Systems

1083 Apple Lisa (100,000 sold, cost $10,000)

1984 Apple Macintosh (Cheap Lisa)
Psion organiser (Pocket computer)

1987 Newton
(Apple PDA, with handwriting recognition)

1989 Xerox tried to protect there user interface in 1989, but lost due to 3 year statue of limitations? had passed.

1990's

1994 Apple lost lawsuit started in 1989? against Microsoft for using a similar user interface. (Vast oversimplification)

1995 Windows 95 (Consumer Market)
Runs on top of DOS

1996 Window NT (Professional Market)

1998 Windows 98 (Consumer Market)
Runs on top of DOS

2000's

2000 Windows 2000 (Professional, Market)

2001 Windows XP (Professional, Consumer market)

2007 iPhone (touch screen)
Windows Vista (Professional, Consumer market)

2007 Nov Android

2008 Oct HTC dream launched (first Android phone)

2009 Windows 7 (Professional, Consumer market)

2010's

2009 Android tablets

2010 Apple iPad
PC sales flat(ish) 2010-2012 ~ 350 million per year

2012 Windows 8 (Professional market?, Consumer market)

2014 PC sales 2014 308 Million units (Includes apple)

2015 Windows 10
PC Sales 2015 276 (DC) 299 (Gartner) Million
Down 10.6% 8.3%
Tablet sales overtake PC sales (?)
PC Sales expected to pick up in 2016 as Businesses move to Windows 10.
Apple up 2.8% Q4 2015 sales ≈ ASUS in Q4

2016

Object Oriented programming Summary

Class

The blueprint definition of the state and behaviour in a particular kind of object (e.g. an ATM).

state

How much money is in the ATM.
Instance variables: `int` notes_10; `int` notes_20;

behaviour

The methods: `withdraw(10), getBalance()`
Which may change or interrogate state

Object

An instance of a Class

The `mezzanine-atm` (object) is an instance of an ATM

Method

The implementation of behaviour in a class
`getBalance() { return theBalance; }`

Message

Invokes a method on an object by sending the object a message. "**`object.message(parameters* )`**"
`mezzanine-atm.withdraw( 10 )`

1.2. Early (CLI) Command Line Unix system

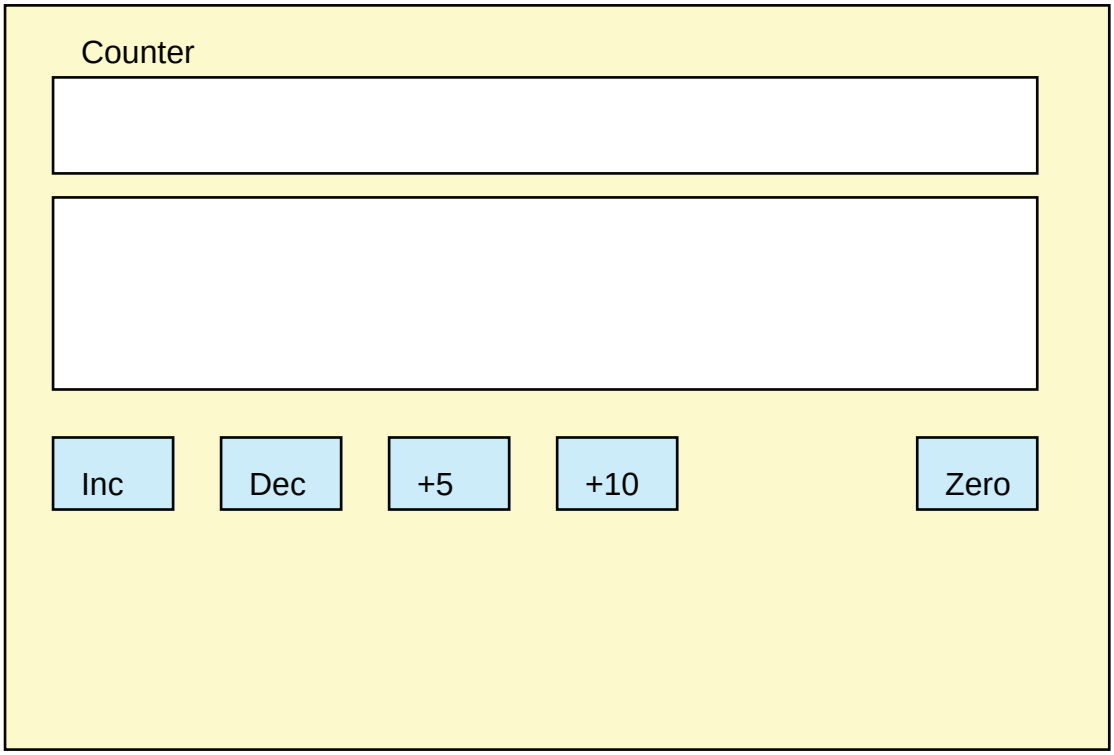
- `http://pdp11.aiju.de`
  Emulation of Unix system 6 (Released May 1975) on a PDP-11 in JavaScript.
  Acessed 05-Feb 2016

1.3. Early (GUI) Graphical User Interface Mac OS

- `http://minivmac.sourceforge.net`
  `http://www.gryphel.com/c/minivmac`
  Emulation of early Mac O/S System 1-6 (1984-1996)
  Accessed 23-Dec 2014

1.4. Demonstration of coursework 4.1

- See 4.1 in Semester 2 programs C/W link

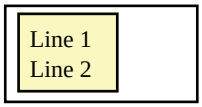
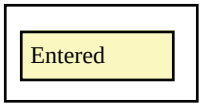
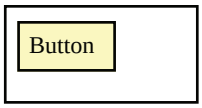
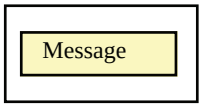


1.5. GUI classes in Java

Several GUI class libraries in Java

- AWT
  Abstract Windowing Tool kit, the original way of creating A GUI in Java
- Swing
  Since Java 1.2, The swing library makes heavy use of the MVC (Model View Controller) software design pattern.
- Java FX (version 1.0 from December 2008)

1.6. GUI in Java (Swing)

Class	An instance of implements	Example
`JFrame`	A window container for visual components in an application or applet.	
`JTextArea`	A multiline area on the screen into which results from the application are written. May also be written into by the end user.	
`JTextField`	A single line on the screen into which the application may write text. May also be written into by the end user.	
`JButton`	The representation of a labelled button on the screen. The button may be pressed to affect an action inside the application.	
`JLabel`	An area on the screen into which results from the application are written. Can not be changed by the end user.	

1.7. lambda expressions

In GUI programming you often need to nominate a method to be called when a user interacts with a visual component.

In Java originally this was originally not directly possible, and a function object had to be created.

> This required the creation of a class that contained the method that is required to be passed as a parameter.
>
> Then an instance of this class is passed as the actual parameter, inside the body of the called method the formal parameter containing the object is stored and later used to call the passed method.

Now (since, Java 1.8) using a lambda expression, this can now be expressed in a more elegant way. In addition the code associated with the action is more easily seen to be associated with the process.

Explicit function object

```
class Store
{
  FO store;

  public void set( FO fo )
  {
    store = fo;
  }

  public void obey()
  {
    store.action();
  }
}

class FO
{
  public void action()
  {
    println("Action!!");
  }
}

class Main
{
  public static void
         main( String args[] )
  {
    Store store = new Store();
    store.set( new FO() );


    store.obey();
    store.obey();
  }
}
```

Using a lambda function

```
class Store
{
  FO store;

  public void set( FO fo )
  {
    store = fo;
  }

  public void obey()
  {
    store.action();
  }
}

interface FO
{
  public void action();
}



class Main
{
  public static void
         main( String args[] )
  {
    Store store = new Store();
    store.set( () ->
          println("Action!!") );

    store.obey();
    store.obey();
  }
}
```

Note: `System.out.println` has been written as `println` to make the example more readable.

Output for both code sequences

```
Action!!
Action!!
```

Creating a GUI - vastly simplified (Swing)

```
class GUI
{
```

```
  public void display()
  {
    // Other GUI components

    JButton button = new JButton( "Name" );

    // Position on window x,y, then width, height
    button.setBounds( x, y, width, height );

    button.addActionListener( e -> {
       //code to perform action on button press
      }
    );

    // add button to visual window
  }


}
```

A complete Java application to render raw HTML into formatted text.

The user enters raw HTML into the bottom window and then presses the button and the formatted text is displayed in to the top window.

Things to be aware of:

- Many library methods / classes used
  So can obscure the issue of learning a programming language.

- Object-oriented approach to programming is extensively used as the GUI is built from a 'large' class hierarchy.

- Can use tools to create the GUI or let a graphic designer create the interface using high level tools.

- Can separate the actual GUI code from the back end code that does the processing of the data.

  MVC - Model View Controller

  This approach of using the MVC pattern will be used in the mini projects

Core idea

An event handler (listener) will execute Java code when an interaction is initiated by a user interacting with the visual display.

For example, a user 'pressing' a button on the screen, causes the event handler for the button to execute code written by the programmer.

Consequences

The programmer does not know the order in which code is executed.

Non GUI program

Do this, Then this, finally this.

GUI program

What it does depends on the order the user interacts with the program. This of course may be different for different users.

Some programs will disallow certain actions unless something else has been done first.

For example, a banking application, will not let you perform a transaction on an account until you have verified that you have permission to do this. This will involve extra program code.
[ATM mini project]

- See programs lectures week 1

Very simple program to demonstrate a GUI program

```
import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.text.html.*;

import java.io.IOException;

class Main
{
  public static void main( String args[] )
  {
    (new Application()).setVisible(true);
  }
}
```

```
class Application extends JFrame
{
  private static final int H = 500;        //Height of window
  private static final int W = 300;        //Width  of window
  private static final int TM = 10;        //Top margin

  private static final int MH=H/2;         //Mid Height
  private static final int HSIZE=MH-80;    //Height of sub window
  private static final String NAME =
          "Press to view rendered HTML";

  private JTextArea   input  = new JTextArea();   //Visual Comp.
  private JEditorPane output = new JEditorPane(); //Visual Comp.
  private JButton     button = new JButton(NAME); //Visual Comp.

  private String message = "";
```

Using a lambda function to pass code to event handler

```
  public Application()
  {
    Container cp = getContentPane();           //Content pane
    cp.setLayout(null);                        //No layout manager
    setSize( W, H );                           //Size of Window
    Font font =
      new Font("Monospaced",Font.PLAIN,12);    //Font Used

    JScrollPane theSP = new JScrollPane();     //Scrolling window
    theSP.setBounds(10, TM, W-40, HSIZE );     //Size of window
    theSP.setFont( font );                     //             Font

    cp.add( theSP );                           //Add to c. pane
    output.setFont(font);
    output.setEditorKit( new HTMLEditorKit() );
    theSP.getViewport().add( output );         //Add output area

    button.setBounds( 10, MH-30, W-40, 40 );

    button.addActionListener(
      e -> {                                   //On BT press
             System.out.println( e.getActionCommand() );
             String text = input.getText();    //get and display
             output.setText( text );           //entered text
           }
     );

    cp.add( button );

    input.setBounds(10, MH+30, W-40, HSIZE); //Input Area Size
    input.setFont( font );                   //             Font
    cp.add( input );                         //Add to c. pane
    input.setText("Type HTML here");         //Message


    setDefaultCloseOperation(EXIT_ON_CLOSE); //Exit on close
  }

}
```

Using a function object to pass code to event handler

```
  public Application()
  {
    Container cp = getContentPane();          //Content pane
    cp.setLayout(null);                       //No layout manager
    setSize( W, H );                          //Size of Window
    Font font =
      new Font("Monospaced",Font.PLAIN,12);   //Font Used

    JScrollPane theSP = new JScrollPane();    //Scrolling window
    theSP.setBounds(10, TM, W-40, HSIZE );    //Size of window
    theSP.setFont( font );                    //             Font

    cp.add( theSP );                          //Add to c. pane
    output.setFont(font);
    output.setEditorKit( new HTMLEditorKit() );
    theSP.getViewport().add( output );        //Add output area

    button.setBounds( 10, MH-30, W-40, 40 );
    button.addActionListener( new  ActionListener()
      {
        public void actionPerformed( ActionEvent e )
        {                                     //On BT press
          System.out.println( e.getActionCommand() );
          String message = input.getText();   //get entered text
          output.setText( message );          //and display
        }
      }
    );

    cp.add( button );

    input.setBounds(10, MH+30, W-40, HSIZE); //Input Area Size
    input.setFont( font );                    //            Font
    cp.add( input );                          //Add to c. pane

    setDefaultCloseOperation(EXIT_ON_CLOSE); //Exit on close
    input.setText("Type here");
  }

}
```

1.8. Demonstration of creating a GUI

- The demonstration uses the WindowsBuilder plug-in for the Eclipse IDE (Integrated Development Environment)

```
http://www.eclipse.org/downloads
https://eclipse.org/windowsbuilder
```

OverView

- Select Layout Manager
  Absolute
- Select visual components used in GUI
- Add call-back code to backend code that processes data.
- Now Run GUI application

1.9. What you should know

- That you need to spend at least 5 hours per week on this module in addition to attending lectures, seminars and tutorials.

- That learning a computer programming language is not an activity that you can undertake in the last few weeks of the semester/ academic year.

2. Coursework - Continued

- All about coursework 4.1, 4.2, 4.3
- Java documentation
  `javadoc`

Java documentation

- [Most] programmers dislike writing documentation.
- Issues

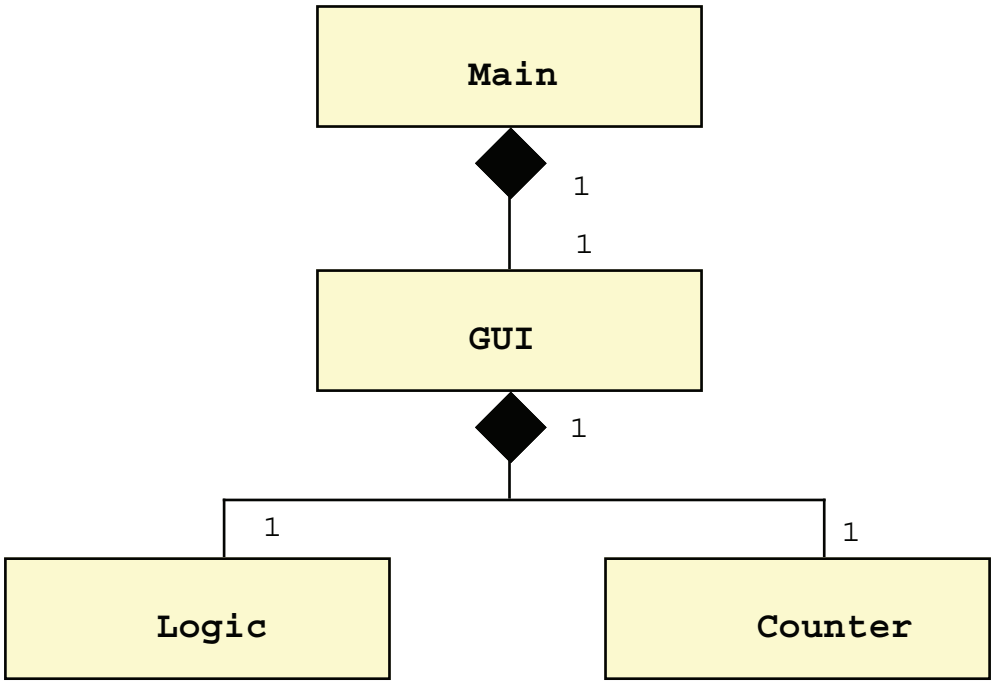
  [Usually] not done consistently
  Plus not [usually] kept up to date
- A solution

  Make documentation integral to the programming process.
  Documentation generated automatically from the Java source code.
- Look at Java documentation for

  `java.lang.System`
  `java.io.PrintStream`
  `java.util.ArrayList<E>`

2.1. Coursework 4.1

Class diagram, showing the relationship between the classes in the program



What is important

- Concept of splitting a program into several classes. Each class implements a self contained part of the program.

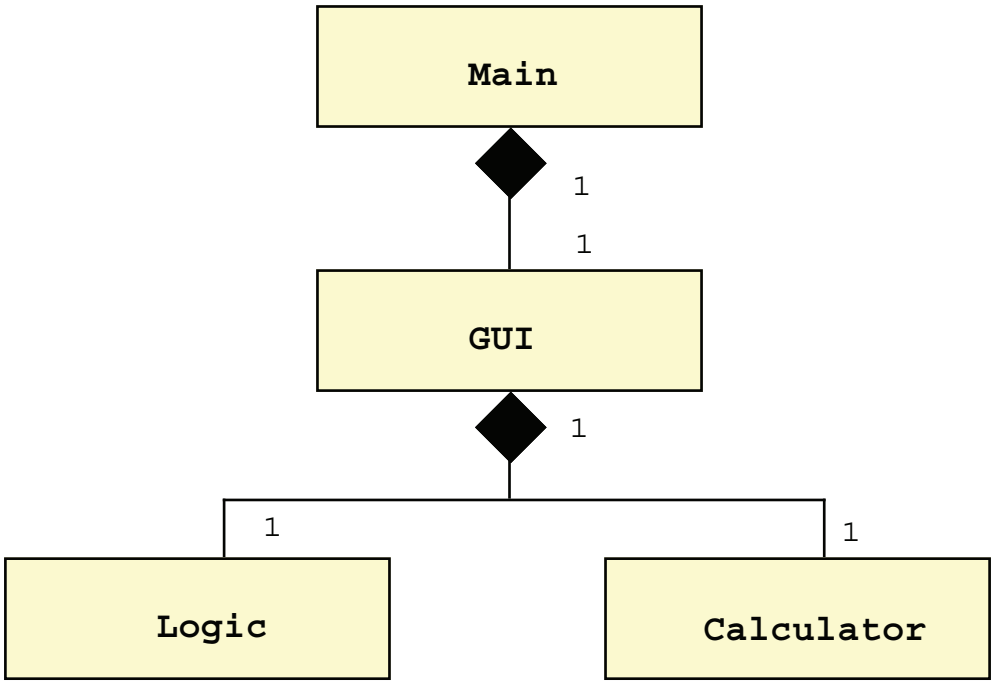
  A class is composed of Data (State) & methods (Behaviour) that use the data.

```
public class Counter
{
  //Declaration of a long instance variable to hold
  // the value of the counter
  /**
   * Return the current value of the counter
   * @return The value of the counter
   */
  public long getValue()
  {
    //Fill in code for the method
  }

  /**
   * Reset the counter to zero
   */
  public void reset()
  {
    //Fill in code for the method
  }

  /**
   * Add 1 to the counter
   */
  public void inc()
  {
    //Fill in code for the method
  }

  /**
   * Subtract 1 from the counter
   */
  public void dec()
  {
    //Fill in code for the method
  }
```

```
  /**
   * Add 5 to the counter
   */
  public void add5()
  {
    //Fill in code for the method
  }

  /**
   * Add 10 to the counter
   */
  public void add10()
  {
    //Fill in code for the method
  }

}
```

2.2. Coursework 4.2

Class diagram, showing the relationship between the classes in the program



- Concept of spliting a program into several classes. Each class implements a self contained part of the program.

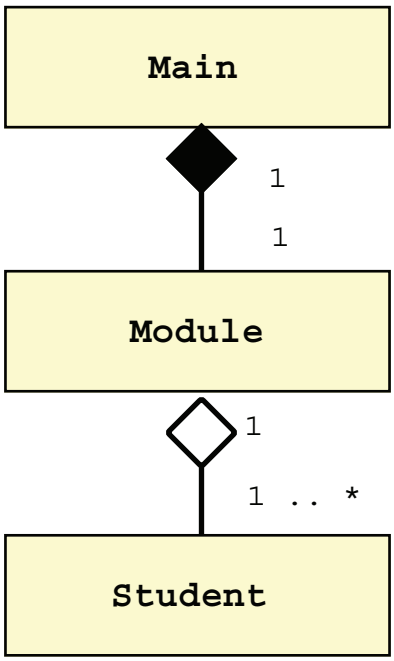
  A class is composed of Data (State) & methods (Behaviour) that use the state information.

```
class Calculator
{
  //Declaration of a long variable to hold the stored result
  private long theResult = 0;

  //Evaluate an arithmetic operation on the stored result
  // E.g evaluate( '+', 9) would add 9 to the stored result
  //     evaluate( '/', 3) would divide the stored result by 3
  //     actions are '+'. '-', '*', '/'
  //Note: if the operation is
  //     evaluate( '/', 0 ) the theResult returned must be 0
  //     (Not mathematically correct)
  //     You will need to do a special check to ensure this
  /**
   * perform the operation
   *  theResult = theResult 'action' number
   * @param action An arithmetic operation + - * /
   * @param number A whole number
   */
  public void evaluate( char action, long number )
  {
    //Fill in code for the method
  }

  /**
   * Return the long calculated value
   * @return The calculated value
   */
  public long getValue()
  {
    //Fill in code for the method
  }

  /**
   * Set the stored result to be number
   * @param number to set result to.
   */
  public void setValue( long number )
  {
    //Fill in code for the method
  }
```

```
  /**
   * Set the stored result to be 0
   */
  public void reset()
  {
    //Fill in code for the method
  }
}
```

2.3. Coursework 4.3

Class diagram, showing the relationship between the classes in the program



- Though not actually incuded in the class Main, a reference to the student object could be retained so if an instance of Module was destroyed, the reference to the student could live on (hence aggregation shown as would allow for expected future modification).

- Concept of spliting a program into several classes. Each class implements a self contained part of the program.

  A class is composed of Data (State) & methods (Behaviour) that use the data.

```
import java.util.*;

class Module
{
  private ArrayList<Student> people = new ArrayList<>();

  /**
   * Add a student
   * @param student A Student
   */
  public void add( Student student )
  {
    people.add( student );
  }

  /**
   * Return the number of students who pass
   * @return Number of students who have passed
   */
  public int pass()
  {
    //Fill in code for the method
  }

  /**
   * Return the number of students who fail
   * @return Number of students who have failed
   */
  public int fail()
  {
    //Fill in code for the method
  }
```

```
  /**
   * Return the name of the student with the highest mark
   *  There will only be 1 students who has the highest mark
   * @return Name of the student with the highest mark.
   */
  public String top()
  {
    //Fill in code for the method
  }


  /**
   * Return the average mark of all the students
   * @return The average mark
   */
  public double average()
  {
    //Fill in code for the method
  }
}
```

The class Student

```
public class Student
{
  private String name;
  private int mark;

  /**
   * Construct a student
   * @param aName Name of student
   * @param aMark The students mark
   */
  public Student( String aName, int aMark )
  {
    name = aName; mark = aMark;
  }

  /**
   * Return the students name
   * @return The students name
   */
  public String getName()
  {
     return name;
  }

  /**
   * Return the students mark
   * @return The students mark 0 .. 100
   */
  public int getMark()
  {
    return mark;
  }
}
```

2.4. Use of an `ArrayList`

- Like an array
  Plus: can insert and delete items (objects).
  However: A different syntax to an array.

```
public class Account
{
  private double theBalance    = 0.00;   //Balance of account
  private String theName       = "";

  public Account( String name, double openningBalance )
  {
    theBalance = openningBalance;
    theName    = name;
  }

  public double getBalance() { /* Some code */ }
  public String getName()    { /* Some code */ }
  public double withdraw( final double money ) { /* Some code */ }
  public void deposit( final double money )    { /* Some code */ }
}
```

2.4.1. Creating and accessing items in an ArrayList

```
ArrayList<Account> accounts = new ArrayList<>(0);
accounts.add( new Account( "Gordon", 100.0 ) );
accounts.add( new Account( "James",  100.0 ) );
accounts.add( new Account( "Edward", 100.0 ) );
accounts.add( new Account( "Percy",  100.0 ) );
accounts.add( new Account( "Thomas", 100.0 ) );

double total = 0.0;
for ( int i=0; i<accounts.size(); i++ )
{
  total = total + accounts.get( i ).getBalance();
}

System.out.printf( "Total amount of money = %10.2f\n", total );
```

```
Total amount of money =     500.00
```

- The foreach loop is an alternative to the conventional `for` loop.

  However, you cannot in general change a collection (Delete or add an element) that is accessed by the control variable of a foreach loop

```
ArrayList<Account> accounts = new ArrayList<>(0);
accounts.add( new Account( "Gordon", 100.0 ) );
accounts.add( new Account( "James",  100.0 ) );
accounts.add( new Account( "Edward", 100.0 ) );
accounts.add( new Account( "Percy",  100.0 ) );
accounts.add( new Account( "Thomas", 100.0 ) );

double total = 0.0;
for ( Account acc : accounts )
{
  total = total + acc.getBalance();
}

System.out.printf( "Total amount of money = %10.2f\n", total );
```

`ArrayList`

Implemented as an array which grows by copying elements.

- Fast random access
- Could be slow to append, insert or delete an element.

`LinkedList`

Implemented as a chain (Daisy chain) of elements

- Could be very slow for random access to individual elements if the collection is large.
- Fast to append at end, insert or delete an element at the current position.

- An `ArrayList` and `LinkedList` are usually interchangeable in a program. As they both implement the interface `List`. (More on this later)

2.4.2. Removing an element from a collection

- Concept of an iterator

  An implementation independent way of moving through the elements of a collection.

```
System.out.println("LinkedList using ListIterator");
LinkedList<Account> accounts = new LinkedList<>();
accounts.add( new Account( "Gordon", 100.00 ) );
accounts.add( new Account( "James",  100.00 ) );
accounts.add( new Account( "Edward", 100.00 ) );
accounts.add( new Account( "Percy",  100.00 ) );
accounts.add( new Account( "Thomas", 100.00 ) );

System.out.print( "Accounts held:" ); //Print items in collection
for ( Account acc : accounts )
{
  System.out.print( " " + acc.getName() );
}
System.out.println();

ListIterator<Account> it = accounts.listIterator();
while ( it.hasNext() )                          //Has next element
{
  Account cur = it.next();                      //Access next element
  if ( cur.getName().equals( "Edward" ) )
  {
    it.remove();                                //Remove last element
  }
}

System.out.print( "Accounts held:" ); //Print items in collection
for ( Account acc : accounts )
{
  System.out.print( " " + acc.getName() );
}
System.out.println();
```

```
LinkedList using ListIterator
Accounts held: Gordon James Edward Percy Thomas
Accounts held: Gordon James Percy Thomas
```

2.4.3. Removing an element from a collection

- Using a foreach loop
  You must break out of the foreach loop immediately you remove an element. Otherwise you will get a `ConcurrentModificationException`.

```
System.out.println("LinkedList using foreach");
LinkedList<Account> accounts = new LinkedList<>();
accounts.add( new Account( "Gordon", 100.00 ) );
accounts.add( new Account( "James",  100.00 ) );
accounts.add( new Account( "Edward", 100.00 ) );
accounts.add( new Account( "Percy",  100.00 ) );
accounts.add( new Account( "Thomas", 100.00 ) );


System.out.print( "Accounts held:" );
for ( Account acc : accounts )
{
  System.out.print( " " + acc.getName() );
}
System.out.println();

for ( Account cur : accounts )
{
   if ( cur.getName().equals( "Gordon" ) )
   {
      accounts.remove( cur );
      //Must exit loop as changed collection
      break;
   }
}

System.out.print( "Accounts held:" );
for ( Account acc : accounts )
{
  System.out.print( " " + acc.getName() );
}
System.out.println();
}
```

```
LinkedList using foreach
Accounts held: Gordon James Edward Percy Thomas
Accounts held: James Edward Percy Thomas
```

2.5. What you should know

- That you need to spend at least 5 hours per week on this module in addition to attending lectures, seminars and tutorials.
- How to use an ArrayList to store and manipulate objects.
- The relationships between classes of aggregation and composition.
- The need for documentation.
- That learning a computer programming language is not an activity that you can undertake in the last few weeks of the academic year.

3. Mini project

- BreakOut- A computer game

3.1. Demonstration

3.2. Tasks

Making the game work

- class `Model`
  Add bricks to the model
  Add code at point [1]

- class `Model`
  Stop the bat moving off the screen. Currently you can move the bat off the screen.
  Add code at point [2]

- class `Model`
  If a ball bounces off a brick the brick is destroyed
  Add code at point [3] to remove brick(s).

- class `View`
  Display the (remaining) bricks on the screen.
  Add code at point [4] to display individual bricks. (Same way as the bat and ball are displayed)

Improving the game

- Allow extra levels of play
  An elegant way of doing this is to factor out into methods:
  - code that creates and manages the brick layout.
  - Code that decides what happens when a ball hits a brick

  Signatures of these methods are then used to create the interface `Level`.

  Then create implementations of the interface `Level_01`, `Level_02` etc.

  Then using an array of instances of these levels select at run-time the `i`'th member of the array to allow the `i`'th level to be played. At the end of playing a level `i` can be increased to allow the next level to be played.

  You may also wish to look at inheritance to help reduce the need to duplicate code.

- Make the game more exciting, addictive, fun. by improving the graphics displayed.
- Improve the score display
- Have sound, so an appropriate sound is made when the ball hits a bat, brick etc.
- Maintain a high score table on a server to allow persistence of data.
- Create a 'user' level designer.

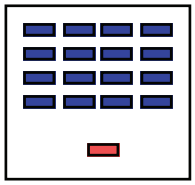
3.3. Overview of the game

Model

State
`bat, ball, bricks`

Behaviour
How to calculate next position in game.

View

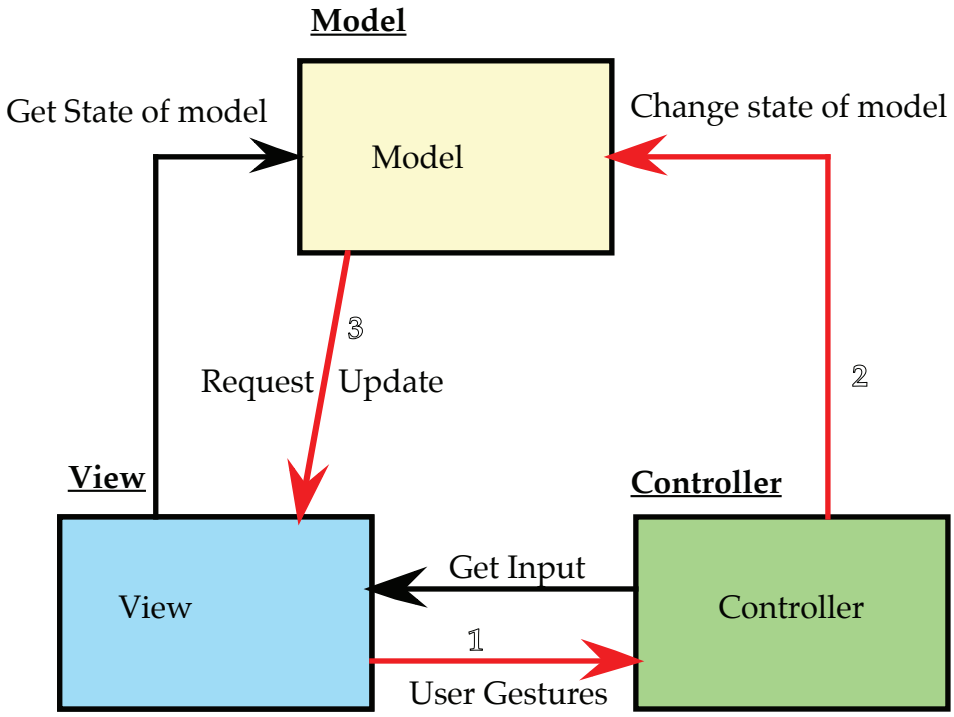


- Model
  Model of the game, holds its current position (state) and how to calculate what happens next (behaviour).

- View
  Responsible for a graphical representation of the current position of the game (obtained from the model)

3.4. MVC - Model View Controller

- 1978/9 Xerox PARC: Smalltalk group

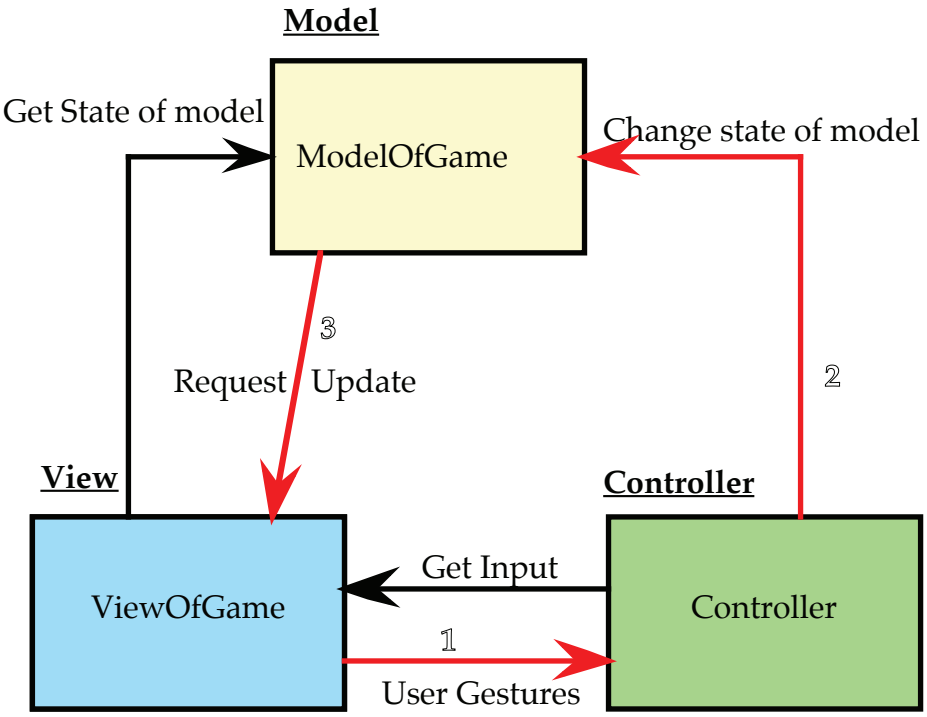


- **Controller**
  Deals with input to the artefact
  User requests to change the model

- **View**
  Deals with the display of the artefact
  The GUI for the artefact displays the artefact and what can be done to it. Captures Buttons pressed, text typed, gestures made by the user. Calls the controller to implement the users request.

- **Model**
  Computer model of the artefact
  A 'data structure' used to represent the model. Plus code to change the model.

3.4.1. Model View Controller

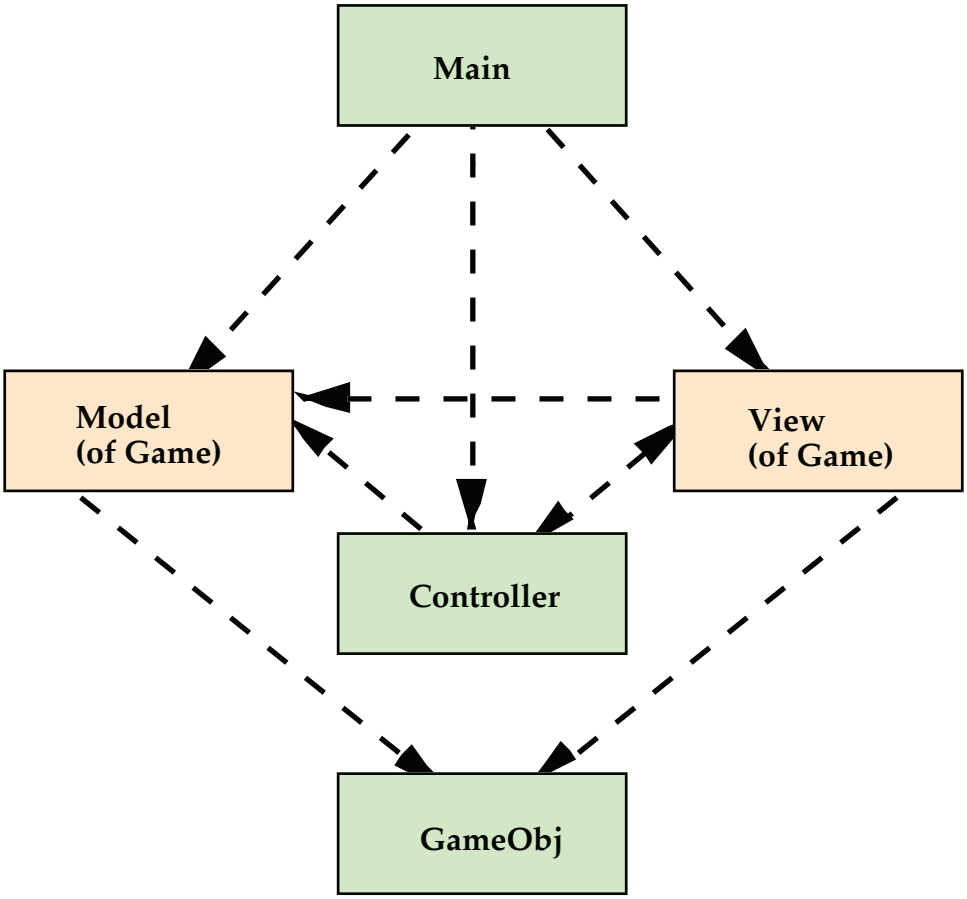


1 User interacts with the view of the game. Code in view calls code in the controller to process action.

2 Controller actions request by calling code in the model to update the game. May access view code, for more information.

3. The model has changed so asks for all views of the model to update there representation. `update` method called in view code.

14.1.1 All of the application



class `GameObj`

An object in the game, will have:

- A position
- A size and colour, may be different for each object (Bat, Ball & Brick are each different sizes and colour)
- A possible direction of movement in the x and y axis (Ball only, Bat handled separately)
- A distance to move

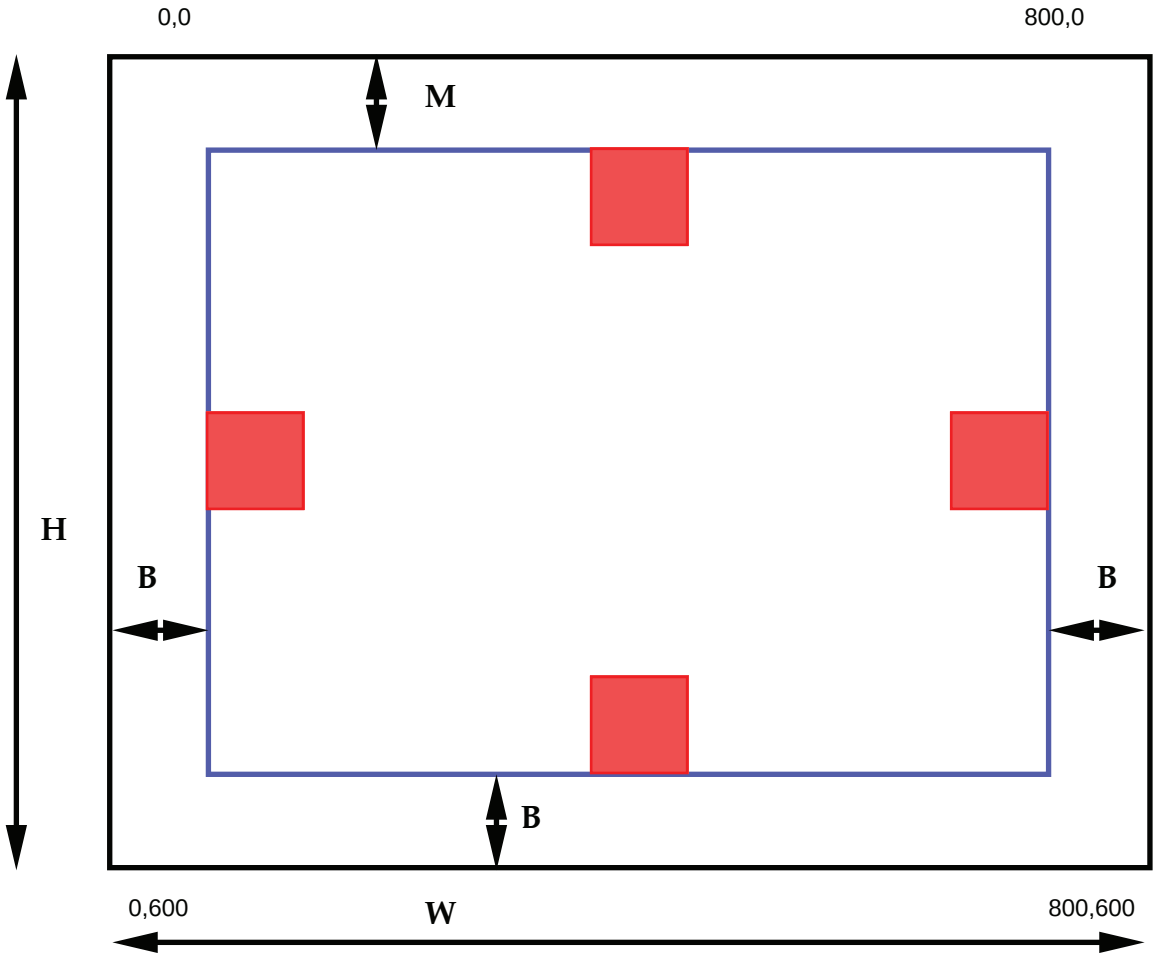
```
public class GameObj
{
  public GameObj( float x, float y,
                  float widthIs, float heightIs, Colour c )
  {
    topX   = x;       topY = y;
    width  = widthIs; height = heightIs;
    colour = c;
  }

}
```

```
public class Model extends Observable
{
  private GameObj ball;          //The ball
  private List<GameObj> bricks;  //The bricks
  private GameObj bat;           //The bat
```

```
  /**
   * Create in the model the objects that form the game
   */
  public void createGameObjects()
  {
    synchronized( Model.class )
    {
      ball   = new GameObj(W/2, H/2, BALL_SIZE, BALL_SIZE, Colour.RED );
      bat    = new GameObj(W/2, H - BRICK_HEIGHT*1.5f, BRICK_WIDTH*3,
                           BRICK_HEIGHT/4, Colour.GRAY);
      bricks = new ArrayList<>();
      //*[1]*********************************************************[1]*
      //* Fill in code to place the bricks on the board               *
      //*************************************************************
    }
  }
```

```
  /**
   * Move the bat. (-1) is left or (+1) is right
   * @param direction - The direction to move
   */
  public void moveBat( int direction )
  {
    //*[2]*********************************************************[2]*
    //* Fill in code to prevent the bat being moved off the screen *
    //*************************************************************
    float dist = direction * BAT_MOVE;    //Actual distance to move
    Debug.trace( "Model: Move bat = %6.2f", dist );
    bat.moveX(dist);
  }
```

Playing surface

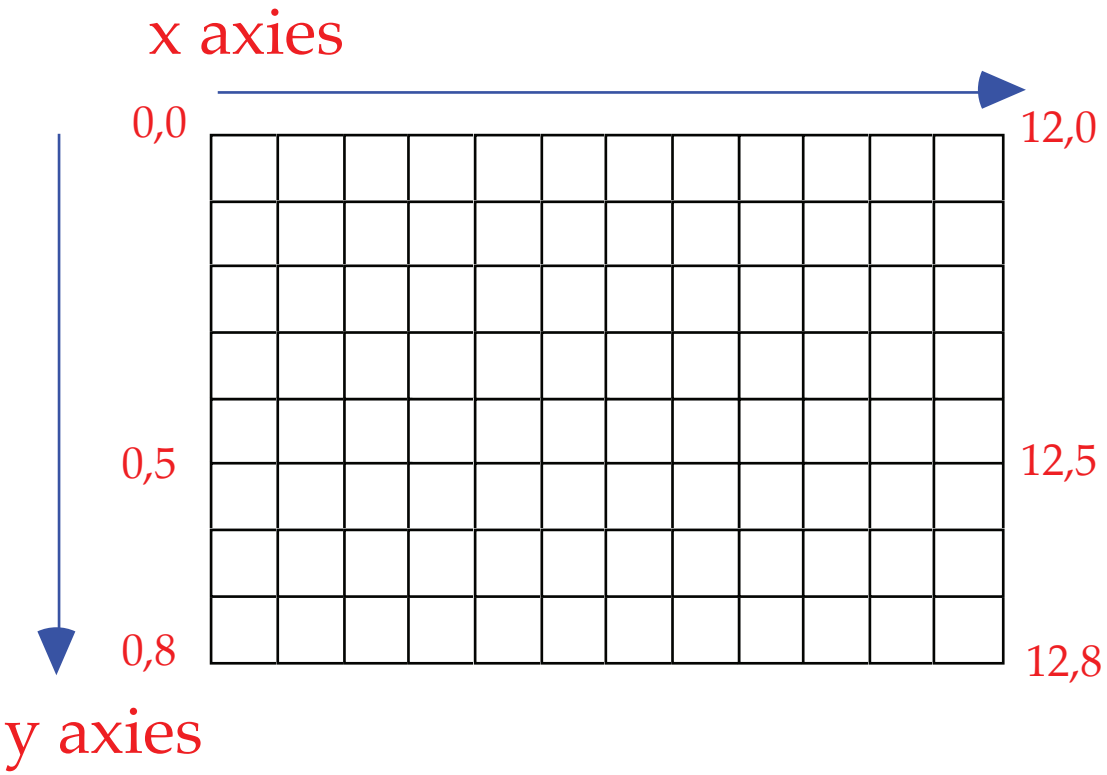


`W`	Width of the displayed surface
`H`	Height of the displayed surface
`M`	Top Margin size
`B`	Border size
`BALL_SIZE`	Size of the ball Height or Width from top left hand corner.

Active part continually updates models state

```
public void runAsSeparateThread()
{
  final float S = 3; //Units to move (Speed)
  try
  {
    synchronized ( Model.class ) //Make thread safe
    {
      GameObj       ball   = getBall();     //Ball in game
      GameObj       bat    = getBat();      //Bat
      List<GameObj> bricks = getBricks();   //Bricks
    }

    while (runGame)
    {
      synchronized ( Model.class ) //Make thread safe
      {
        float x = ball.getX();  //Current x,y position
        float y = ball.getY();
        //Deal with possible edge of board hit
        if (x >= W - B - BALL_SIZE)  ball.changeDirectionX();
        if (x <= 0 + B            )  ball.changeDirectionX();
        if (y >= H - B - BALL_SIZE)  //Bottom
        {
          ball.changeDirectionY(); addToScore( HIT_BOTTOM );
        }
        if (y <= 0 + M            )  ball.changeDirectionY();

        //As only a hit on the bat/ball is detected it is
        // assumed to be on the top or bottom of the object.
        //A hit on the left or right of the object
        // has an interesting affect

        boolean hit = false;
        //*[3]*********************************************************[3]*
        //* Fill in code to check if a visible brick has been hit      *
        //*      The ball has no effect on an invisible brick          *
        //*****************************************************************
        if (hit)
          ball.changeDirectionY();

        if ( ball.hitBy(bat) )
          ball.changeDirectionY();
      }
      modelChanged();      //Model changed refresh screen
      Thread.sleep( fast ? 2 : 20 );
      ball.moveX(S);  ball.moveY(S);
    }
  } catch (Exception e)
  {
    Debug.error("Model.runAsSeparateThread - Error\n%s",
                e.getMessage() );
  }
 }
}
```

3.5. Swing (Graphics primitives)



Parameters	Are used to denote
`x, y`	The top left hand corner of the shape to be drawn.
`width, height`	The width and height of the shape to be drawn.

Shape	Represented by
	**new** Line2D.Float(x, y, width, height)
	**new** Rectangle2D.Float(x, y, width, height)
	**new** Ellipse2D.Float(x, y, width, height)

`g` The object g is an instance of the graphics context.
See the Java documentation for:
`java.awt.Graphics2D`

Drawn by

`g.fill( shape )` A filled shape
`g.draw( shape )` The outline of the shape

Java FX (Uses another set of graphics primitives)

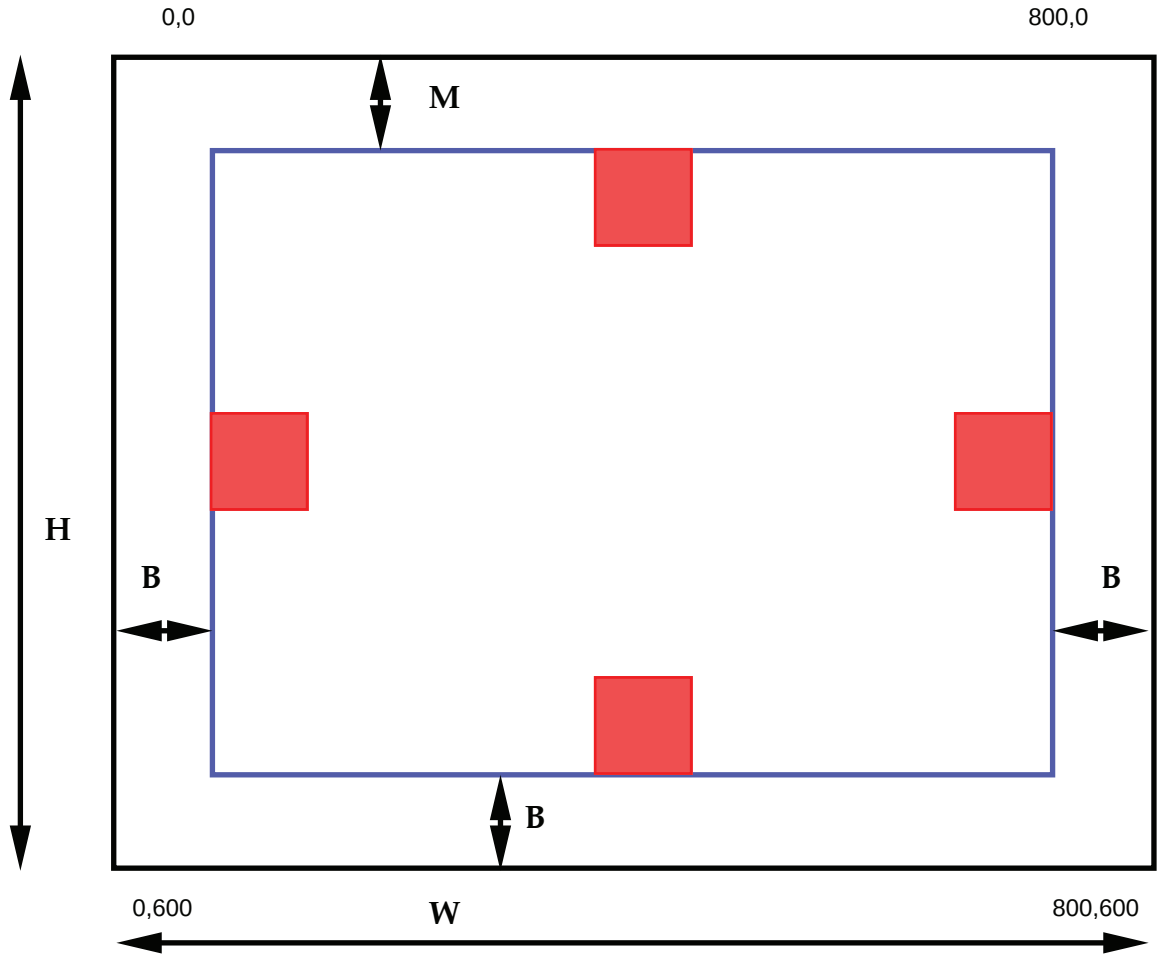
Parameters	Are used to denote
`x, y`	The top left hand corner of the shape to be drawn.
`width, height`	The width and height of the shape to be drawn.

`g` The object g is an instance of the graphics context.
See the Java documentation for
`javafx.scene.canvas.GraphicsContext`

Shape	Drawn by
	g.strokeLine(x, y, width, height)
	g. strokeRect(x, y, width, height)
	g.strokeOval(x, y, width, height)

	g.setFill(Color.RED); g. fillRect(x, y, width, height)
	g.setFill(Color.RED); *// If already set RED not required* g.fillOval(x, y, width, height)

```
while (runGame)
{
  float x = ball.getX();  //Current x,y position
  float y = ball.getY();
  //Deal with possible edge of board hit
  if (x >= W - B - BALL_SIZE)  ball.changeDirectionX();
  if (x <= 0 + B     )         ball.changeDirectionX();
  if (y >= H - B - BALL_SIZE)  //Bottom
  {
    ball.changeDirectionY(); addToScore( HIT_BOTTOM );
  }
  if (y <= 0 + M            )  ball.changeDirectionY(); //Top
```



This is the code that is called to display the screen

```
public void drawActualPicture( Graphics2D g )
{
  final int  RESET_AFTER = 200; //Movements
  frames++;
  synchronized( Model.class )   //Make thread safe
  {
    //White background
    g.setPaint( Color.WHITE );
    g.fill( new Rectangle2D.Float( 0, 0, width, height) );

    Font font = new Font("Monospaced",Font.BOLD,24);
    g.setFont( font );

    displayGameObj( g, ball );   //Display the Ball
    displayGameObj( g, bat  );   //Display the Bat


    //*[4]*****************************************************[4]*
    //* Display the bricks that make up the game                  *
    //* Fill in code to display bricks                            *
    //* Remember only a visible brick is to be displayed          *
    //************************************************************

    //Display state of game
    g.setPaint( Color.black );
    FontMetrics fm = getFontMetrics( font );
    String fmt = "BreakOut: Score = [%6d] fps=%5.1f";
    String text = String.format(fmt, score,
                                frames/(Timer.timeTaken()/1000.0)
                  );
    if ( frames > RESET_AFTER )
      { frames = 0; Timer.startTimer(); }
    g.drawString( text, width /2-fm.stringWidth(text)/2, 80  );
  }
}
```

3.6. When character typed at keyboard

```
  class Transaction implements KeyListener  //When character typed
  {
    @Override
    public void keyPressed(KeyEvent e)      //Obey this method
    {
      //Make -ve so not confused with normal characters
      controller.userKeyInteraction( -e.getKeyCode() );
    }

    @Override
    public void keyReleased(KeyEvent e)
    {
      //Called on key release including specials
    }

    @Override
    public void keyTyped(KeyEvent e)
    {
      //Send internal code for key
      controller.userKeyInteraction( e.getKeyChar() );
    }
  }
}
```

```
public class Controller
{
  private Model model;    //Model of game
  private View  view;     //View of game

  public Controller(Model aBreakOutModel,
                    View aBreakOutView )
  {
    model  = aBreakOutModel;
    view   = aBreakOutView;
    view.setController( this );    //View could talk to controller
  }

  /**
   * Decide what to do for each interaction from the user
   * Called from the interaction code in the view
   * @param keyCode The key pressed
   */
  public void userKeyInteraction(int keyCode )
  {
    //Key typed includes specials, -ve
    //keycode is ASCII value
    switch ( keyCode )                 //Character is
    {
      case -KeyEvent.VK_LEFT:          //Left Arrow
        model.moveBat( -1);
        break;
      case -KeyEvent.VK_RIGHT:         //Right arrow
        model.moveBat( +1 );
        break;
      case 'f' :
        model.setFast( true );         //Very fast ball movement now
        break;
      case 'n' :
        model.setFast( false );        //Normal speed
        break;
    }
  }

}
```

3.7. What you should know

- That you need to spend at least 5 hours per week on this module in addition to attending lectures, seminars and tutorials.

- That learning a computer programming language is not an activity that you can undertake in the last few weeks of the academic year.

4. Mini project

- Bank ATM

4.1. Demonstration

4.2. Tasks

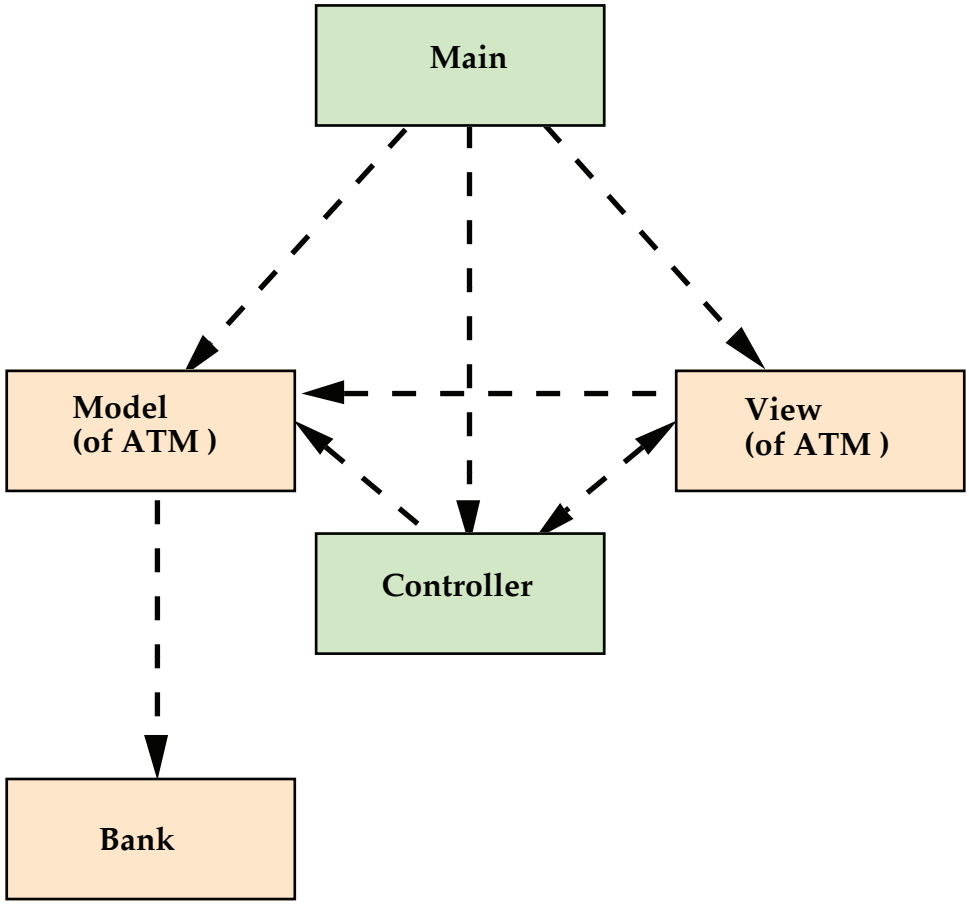
Making the ATM work

- Design the tables(s) that you will use in your database.
- Create a Java application to create and populate the database with account information.
- Add appropriate code to the class `Bank` to access information in the database.

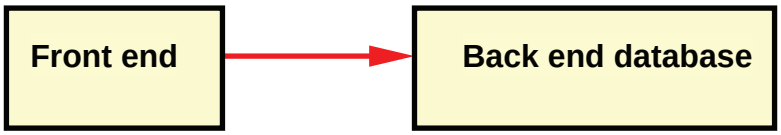
Improving the ATM

- Allow a customer to change their password
  Need to add new button & action
- Management access
  Separate program.
- Improve the interaction to the ATM.
- Audit trail.

4.3. Overview ATM



4.4. Concept 2 tired architecture



- Each tier is on a different machine

4.5. Security

- There is no security.

4.6. Real systems

- Use a client server approach with strong data encryption between the two tiers (see: Two Tier Architecture).

  Ensured by a secure data encryption mechanism.

  Possibilities:
  SSL (Secure Socket Layer)

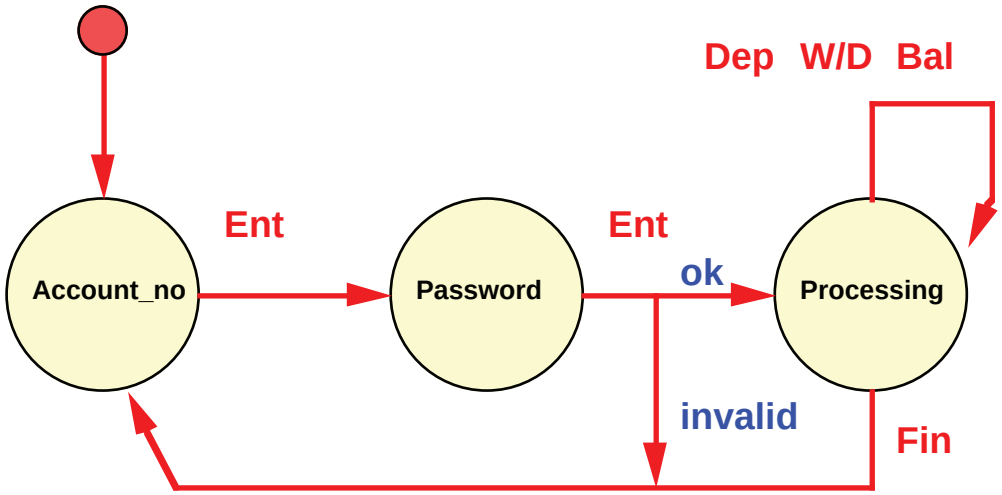
- Use a 1 way function to create a cryptographic hash (Known as a digest) of the password, the digest of the password is stored on the server, not the plain text of the password.

  Then when the user sends their login name & password the password is converted by the 1 way function to a digest and compared against the stored digest of the users password.

  Hence users passwords are never stored in plain text on the banks server.

  MD5 and SHA-1 are no longer considered secure. SHA-2 & SHA-3 still considered secure.

4.7. State diagram for ATM



	Action	Next State
Number	Ent	`Password`
Number	Ent	`Processing/ Account_NO`
Number	Dep, W/D, Bal	`Processing`
	Fin	`Account_NO`

4.8. Connection jdbc

jdbc Java DataBase Connectivity

```
private Connection con = null;       //Connection to database
private Statement  stm = null;      //Used to talk to database

private static final String urlDB =
                "jdbc:derby:derby.db;create=true";
private static final String DRIVER =
                "org.apache.derby.jdbc.EmbeddedDriver";
```

```
private void createDB()
{
  try
  {
    // Driver to access database
    Class.forName(DRIVER).newInstance();
    con = DriverManager.getConnection( urlDB, "", "" );
  }
  catch ( SQLException e )
  {
    System.err.println( "Problem with connection to " + urlDB );
    System.out.println("SQLException: " + e.getMessage());
    System.out.println("SQLState:     " + e.getSQLState());
    System.out.println("VendorError:  " + e.getErrorCode());
    System.exit( -1 );
  }
  catch ( Exception e )
  {
      System.err.println("Can not load JDBC/ODBC driver.");
      System.exit( -1 );
  }

  try {
      theStmt = con.createStatement();
  } catch (Exception e) {
      System.err.println("problems creating statement object" );
  }
}
```

4.9. Database table

productNo	department	description	price

```
private void createTable()
{
  try
  {
    //Remove (drop) table as create in next section
    stm.execute( "drop table StockList" );     //May fail
  } catch (Exception e)
  {
    //Table did not exist - just ignore
  }


   String sql = "create table StockList (     " +
                "  productNo      Char(6) ,    " +
                "  department     Char(6),     " +
                "  description    Varchar(40)," +
                "  price          Float )";
  try
  {
     stm.execute( sql );

  } catch (Exception e)
  {
    System.out.printf("problems with SQL statement:\n %s\n %s\n",
                      sql, e.getMessage() );
    System.exit(-1);  //Give up
  }
}
```

4.10. Populate

productNo	department	description	price
0001	Comp	Intel i5-6600	176.22

```
private void populate()
{
  try
  {
    //Populate with some values

    stm.execute( "insert into StockList values " +
              "('0001', 'Comp', 'Intel i5-6600',    176.22)" );
    stm.execute( "insert into StockList values " +
               "('0002', 'Comp', '4GB DDR4 memory',  21.81)" );
    stm.execute( "insert into StockList values " +
               "('0003', 'Comp', '32GB memory Stick', 7.55)" );
    stm.execute( "insert into StockList values " +
                "('0004', 'Comp', '4TB Disk Drive', 109.99)" );
    stm.execute( "insert into StockList values " +
                "('0005', 'Home', 'Clock',  15.00)" );

  } catch (Exception e)
  {
    System.err.println("problems with SQL statement:\n" +
                       e.getMessage() );
  }
}
```

4.11. processing a database transaction

```
  private void process()
  {
    try
    {
      //Find and print all products less than 60.00

      ResultSet res = null;
      res = stm.executeQuery(
                  "select productNo, price from StockList " +
                  "where price < 60.00 " );
      while ( res.next() )
      {
        System.out.printf( "ProductNo = %s Price = %6.2f\n",
                           res.getString("productNo"),
                           res.getDouble("price") );
      }

    } catch (Exception ex)
    {
      System.err.println("problems with SQL statement:\n" +
                         ex.getMessage() );
    }
}
```

```
ProductNo = 0002   Price =  21.81
ProductNo = 0003   Price =   7.55
ProductNo = 0005   Price =  15.00
```

Contents of `res`

productNo	price
0002	21.81
0003	7.55
0005	15.00

Use

`res.getString("productNo")`
Will return as a string the contents of the value in the column `productNo` for the current row.
`res.next()`
Will return true if there is a next row and advance to it.. Otherwise will return false.

Also `getInt, getLong, getFloat, getDouble`

4.12. What you should know

- That you need to spend at least 5 hours per week on this module in addition to attending lectures, seminars and tutorials.
- That learning a computer programming language is not an activity that you can undertake in the last few weeks of the academic year.

5. Testing / Interfaces & inheritance

- Types of software testing
- Do you need to test using all possible data values?
- Some software errors and the consequences.
- The keywords `interface` & `implements`
  In essence a promise to implement method(s) in a class. The `interface` provides the signature of the methods that are to be implemented.
- The keywords `extends`
  A way of modifying (Changing the behaviour of) an existing class.

5.1. Is testing important

- 1st August 2012
  Knight Capital (US market [financial] trading company)
  Software problem led to the company loosing approximately $10 million a minute on financial trades for 45 minutes.

  Which was approximately 3 times there annual profit.

  Short term outcome:

  - Share price of company falls by 75%
  - Injection of $400 million into the company, for 70% of company stock (value $1.50 a share, before event $10.33 a share)

5.2. Types of testing

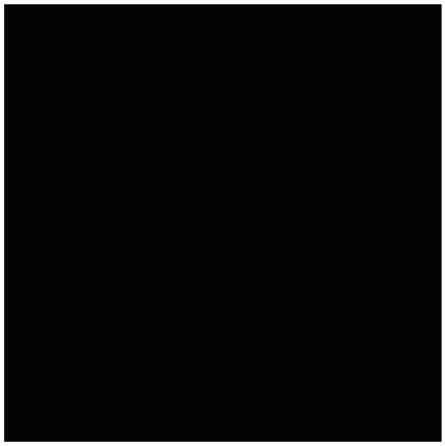
- White box
  tests internal structures. Testers devise tests to test the paths through the code.

- Black box
  Test the functions, testers do not (normally) see the code / structure of the application been tested.

White box testing

```
class Account
{
  private double theBalance    = 0.00;
  private double theOverdraft  = 0.00;
  public double getBalance()
  {
    return theBalance;
  }
  public double withdraw( final double money )
  {
    assert money >= 0.00;
    if ( theBalance - money >= theOverdraft )
    {
      theBalance = theBalance - money;
      return money;
    } else {
      return 0.00;
    }
  }
```

Black box testing



5.3. Aim to find (all) errors

- Devise test data to:
  - Check that valid data is handled correctly
  - Check that invalid data is handled correctly

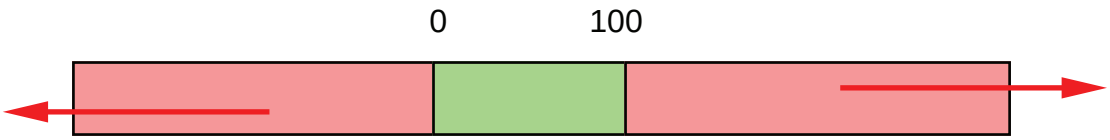
- Early program
  Process exam results

  To pass a module you need to gain a module mark of at least 40% and a mark of at least 30 in every component (exam or coursework mark) that you take. The module mark is the average of the exam and coursework components.

  There are several outcomes

Pass	Module mark greater than 40 component marks greater than 30.
Fail	Module mark less than 40
Fail [Threshold]	Component mark less than 30, but the overall module mark is a pass

- You could do exhaustive testing
  There are 101 * 101 (approx. 10,000) possible valid data inputs consisting of an exam mark followed by a coursework mark.



The invalid data input (assume using an `int`) is approximately 4 billion * 4 billion - 10,000 data values.
Which is approximately 16 * 10^{18} different values
If you could test 1 billion of these a second it would take you 16 billion seconds which is approximately 500 years.

- Be like a detective, ask the important questions.

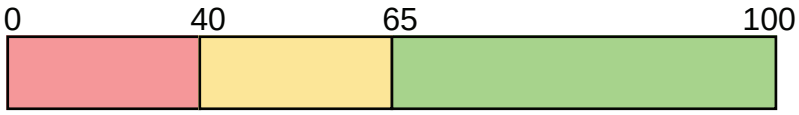
Valid data check

Pass: (looking at boundaries)

c/w	exam	
40	40	Bare pass
30	50	Threshold just on c/w
50	30	Threshold just on exam
100	100	Max. mark

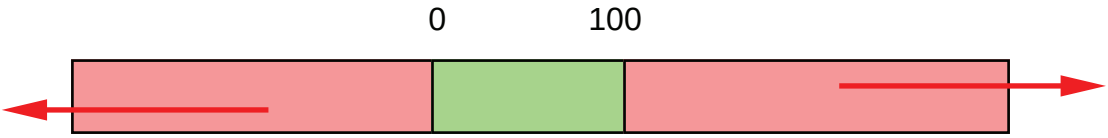
Fail: (looking at boundaries)

c/w	exam	
0	0	Minimum mark
39	40	Just fail because of c/w
40	39	Just fail because of exam



Threshold fail: (looking at boundaries)

c/w	exam	
29	51	Mark >= 40 threshold c/w
51	29	Mark >= 40 threshold ex.
29	100	Mark >= 40 threshold c/w
100	29	Mark >= 40 threshold ex.



Invalid data check (looking at boundaries)

c/w	exam	
-1	50	Invalid cw (low)
101	50	Invalid cw(high)
50	-1	Invalid exam (low)
50	101	Invalid exam (high)

- If the program passes all these tests (using black box testing), is the program correct?

Could someone have written

```
if (cwmark ==  0  || cwmark == 1  ||
    cwmark ==  2  || cwmark == 3  ||
    cwmark ==  4  || cwmark == 5  ||

       // etc to

    cwmark ==  92 || cwmark == 93 ||
    cwmark ==  94 || cwmark == 95 ||
    cwmark ==  98 || cwmark == 99 ||
    cwmark == 100 )
{
      // Process valid mark
      // But unfortunately missed out
      //  the check for marks 96 & 97
} else {

      // Process invalid mark
}
```

- Remember:
  Testing only shows that the program works with the test data that you give it.

- Testing (Outline)

- Test each class (unit testing)
  Write a program to test each class.
  So test can be repeated if code is changed, if in testing an error is found, correct class and re-run test.

- Test the whole program
  This may show up errors in individual classes, fix errors in individual classes, add to class tests so the discovered failure can be checked for.

  Re test the whole program

- Unit testing
  Usually at the class level, tests written by developers as they write the code (white box testing)

- Integration testing
  Testing the interfaces between components in the software design.
  Used to expose issues in the interface and interaction between components

- System testing
  Test the system to make sure it meets the requirements.

- Regression testing
  Testing the system after a 'software update' to make sure that old bugs/ new bugs do not appear.

- Some programs are safety critical, the cost of failure is greater than the cost of the system.

  1985-7 **The Therac-25 accidents**

  A medical electron linear accelerator essential contained a software error that causes massive radiation overdoses. As a result several people died and others were seriously injured.

  Causal factors (From: Nancy Leveson, Safeware: Systems Safety and Computers, Addison-Wesley 1995)

  - Overconfidence in software
  - Confusing reliability with safety
  - Lack of defensive design
  - Failure to eliminate root causes
    Belief that it was a hardware issue (Micro switch issue)
  - Complacency
  - Inadequate software engineering practices
  - Safe versus friendly user interfaces

- 1962 Mariner 1 space probe
  Error in transcribing formula into program code cause space craft to divert from intended flight plan. (Controlled destruction of probe)

- 1983 Russian defence system
  Sunlight reflecting of cloud tops, taken as alerts of missile launches, software did not filter out these spurious alerts. Fortunately duty officer flagged alerts as faulty as only '5 missiles launched'. Protocol in such events was to respond decisively, launching soviet missiles before they would be destroyed.

- 1990 Network outage AT&T 60,000 people left with no long distance telephone services for 9 hours after software update.
  Fix, load the old software.

- 1993 Pentium bug
  4195835.0/3145727.0 gives 1.33374 instead of 1.33382
  Cost to Intel $475 million

- 1996 Ariane 5 Flight 501
  Re-use of code from Ariane 4, but Ariane 5 faster causing an overflow condition resulting in the flight computer crashing.
  Resulting in engines to overpower and rocket to disintegrates.
  http://www.youtube.com/watch?v=gp_D8r-2hwk

5.4. What is an interface

An `interface` is used to define a protocol that a class will implement.

```
interface AccountProtocol
{
  public double getBalance();

  public void deposit( final double money );

  public double withdraw( final double money );
}
```

```
class Account implements AccountProtocol
{
  private double theBalance    = 0.00;     //Balance of account


  public double getBalance()
  {
    return theBalance;
  }

  public void deposit( final double money )
  {
    theBalance += money;
  }

  public double withdraw( final double money )
  {
    if ( theBalance - money >= 0.0 )
    {
       theBalance -= money;
       return money;
    } else {
       return 0.0;
    }
  }
}
```

The compiler checks that the class has implemented all of the methods in the interface `AccountProtocol`.

An `interface` in Java is a promise that the programmer who creates a class that implements an `interface` will provide concrete methods for all the 'abstract' methods that have been defined in the `interface`.

If you implement an interface in a class and do not provide a method for the 'abstract' method(s) in the interface the class will not compile.

5.5. Why use an interface

- Guarantees that a class will implement a method
- In Java you can use an interface where a class name would be used.
  For example,

```
public static double transfer( final double money,
  AccountProtocol from, AccountProtocol to )
{
  if ( money > 0.00 )
  {
    double obtain = from.withdraw( money );
    if ( obtain == money )
    {
      to.deposit( money );
      return money;
    }
  }
  return 0.00;
}
```

the method `transfer` could be called with its second and third parameters being an instance of any class that implements the interface `AccountProtocol`.

5.5.1. Example

- A SpecialAccount only allows 3 withdraws in any time period. This limit can be reset by calling the method `resetMaxWithdraws`.

```
class SpecialAccount implements AccountProtocol
{
  private final static int MAX_WITHDRAWS = 3;
  private double theBalance       = 0.0d;   //Balance of account
  private int    theMaxWithdraws = MAX_WITHDRAWS;

  public double getBalance()
  {
     return theBalance;
  }

  public double withdraw( final double money )
  {
    if ( theMaxWithdraws > 0  &&
         theBalance - money >= 0.00 )
    {
      theMaxWithdraws--;
      theBalance = theBalance - money;
      return money;
    }
    return 0.00;
  }

  public void deposit( final double money )
  {
     theBalance = theBalance + money;
  }

  public void resetMaxWithdraws()
  {
    theMaxWithdraws = MAX_WITHDRAWS;
  }
}
```

5.5.2. Using classes that implement `AccountProtocol`

- 4 transfers using an instance of an `Account`

```
class Main
{
  public static void main( String args[] )
  {
    Account mike    = new Account();
    Account corinna = new Account();

    double obtained;

    mike.deposit(100.00);
    corinna.deposit(150.00);

    transfer( 10.00, mike, corinna );
    transfer( 10.00, mike, corinna );
    transfer( 10.00, mike, corinna );
    transfer( 10.00, mike, corinna );

    System.out.printf( "Corinna's Balance    = %6.2f\n",
                       corinna.getBalance() );

    System.out.printf( "Mike's Balance       = %6.2f\n",
                       mike.getBalance() );
  }
}
```

```
Corinna's Balance    = 190.00
Mike's Balance       =  60.00
```

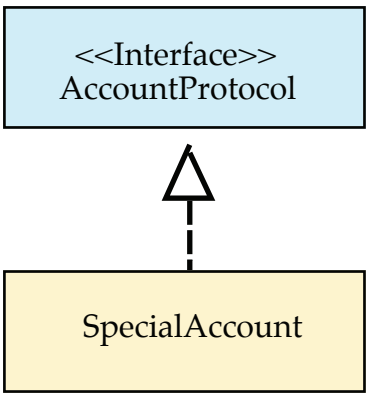
- Now try 4 transfers using an instance of a `SpecialAccount`

```
class Main
{
  public static void main( String args[] )
  {
    BetterSpecialAccount mike    = new BetterSpecialAccount();
    Account              corinna = new Account();

    double obtained;

    mike.deposit(100.00);
    corinna.deposit(150.00);

    transfer( 10.00, mike, corinna );
    transfer( 10.00, mike, corinna );
    transfer( 10.00, mike, corinna );
    transfer( 10.00, mike, corinna );

    System.out.printf( "Corinna's Balance    = %6.2f\n",
                       corinna.getBalance() );

    System.out.printf( "Mike's Balance       = %6.2f\n",
                       mike.getBalance() );

  }
}
```

- Limit applied only 3 allowed (Different behaviour)

```
Corinna's Balance    = 180.00
Mike's Balance       =  70.00
```

UML Diagram (Interface)



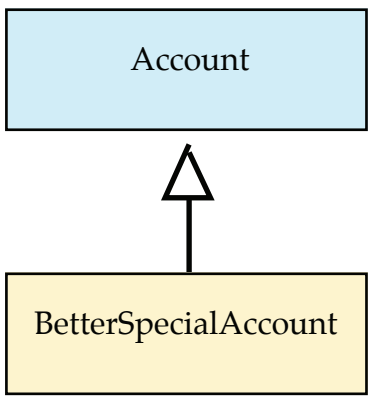
Interface

Class

5.6. Inheritance (An advanced concept)

```
class BetterSpecialAccount extends Account
{
  private final static int MAX_WITHDRAWS = 3;
  private int theMaxWithdraws= MAX_WITHDRAWS;

  public void resetMaxWithdraws()
  {
    theMaxWithdraws = MAX_WITHDRAWS;
  }

  public double withdraw( final double money )
  {
    if ( theMaxWithdraws > 0 )
    {
      double get = super.withdraw( money );
      if ( get > 0.0 )
      {
        theMaxWithdraws--;
        return money;
      }
    }
    return 0.00;
  }
}
```

UML Diagram (Inheritance)



The Superclass

The Subclass

Inheritance example Summary

- Inherits all state (variables) and behaviour (Methods) from an existing class (called the super class).
- As Account implements `AccountProtocol` `BetterSepecialAccount` will also implement `AccountProtocol`.
- The class Account implicitly inherits behaviour and state from the class `Object`
- Then you can add extra state (variables) and behaviour (methods) to the new class (called the sub class). In the case of BetterSpecialAccount its the variables, `MAX_WITHDRAWS` and `theMaxWithdraws` and the methods `resetMaxWithdraws` and `withdraw` (which overrides `withdraw` in the super class)
- Override is when you override an existing method that you inherited with a new method. (Must have the same signature as the overridden method)

  prefix **`super`** is used to call the overridden method (in the super class). Otherwise you would get endless recursion.

- If a class does not explicitly inherit from another class then the compiler implicitly inherits from the class `Object`.

  Hence all classes contains the methods and instance variables of the class `Object`.

- The **is-a** relationship

  ```
  class SmartPhone extends Phone
  ```

  A `SmartPhone` **is-a** `Phone`.

  ```
  SmartPhone aSmartPhone;
  Phone      aPhone
  ```

  `aSmartPhone` can be assigned to `aPhone`

  > As a `SmartPhone` can do all that a `Phone` can do.

  ```
  aPhone = aSmartPhone;   // OK ✔
  ```

  However, `aPhone` **is-a** `SmartPhone` is false.

  `aPhone` can not be assigned to `aSmartPhone`

  > As a `Phone` can not do all that a `SmartPhone` can do

  ```
  aSmartPhone = aPhone;   // Wrong ✘
  ```

5.7. Why use inheritance

- Simplifies code production
  Working smarter.

- Builds on existing code that has already been written.

  An existing class may not do exactly what you want but by using inheritance you can modify the behaviour to do what you want the class to do.

  The class inherited from (the super class) may have been written by you, or more likely to have been written by someone else.

- Only need to test the new features in the class, not the inherited behaviour (Simple explanation, lookup factory method pattern).

5.8. What you should know

- That you need to spend at least 5 hours per week on this module in addition to attending lectures, seminars and tutorials.
- Why testing is important
- Concept of an interface. Used next week.
- Concept of inheritance.
  UML notation for inheritance.
- That learning a computer programming language is not an activity that you can undertake in the last few weeks of the academic year.

6. Data structures

- Data structure
  A structure to store data together with methods to allow efficient:
  - access to data items
  - ability to add new data items
  - ability to delete data items

- Helps solve problems like:
  In a spreadsheet how is the data stored in memory

  Array

  + Simple
  - Very expensive in memory, as have to pre-allocate the maximum size
    What happens when you insert a row or column

- Speed / time trade-off
  There is often a choice between space used by a data structure and the speed of access to the data in the data structure.

Smaller space used	Usually results in a slower access to the data.
Larger space used	Usually allows a faster access to the data.

- In Java
  The collection classes provide a range of common data structures that can be used in many situations.

  ```
  java.util.*;
  ```

  Specifically they 'hide' the implementation details (often complex) and allow the programmer to concentrate on the rest of the program implementation.

  Can choose which collection to use based on metrics of:
  Space used
  Time taken to access/delete/add elements

- Why use the collection classes

  Speeds up code development as do not need to spend time re-implementing existing/known algorithms

  Allows re-use of existing code
  Do not re-invent the wheel

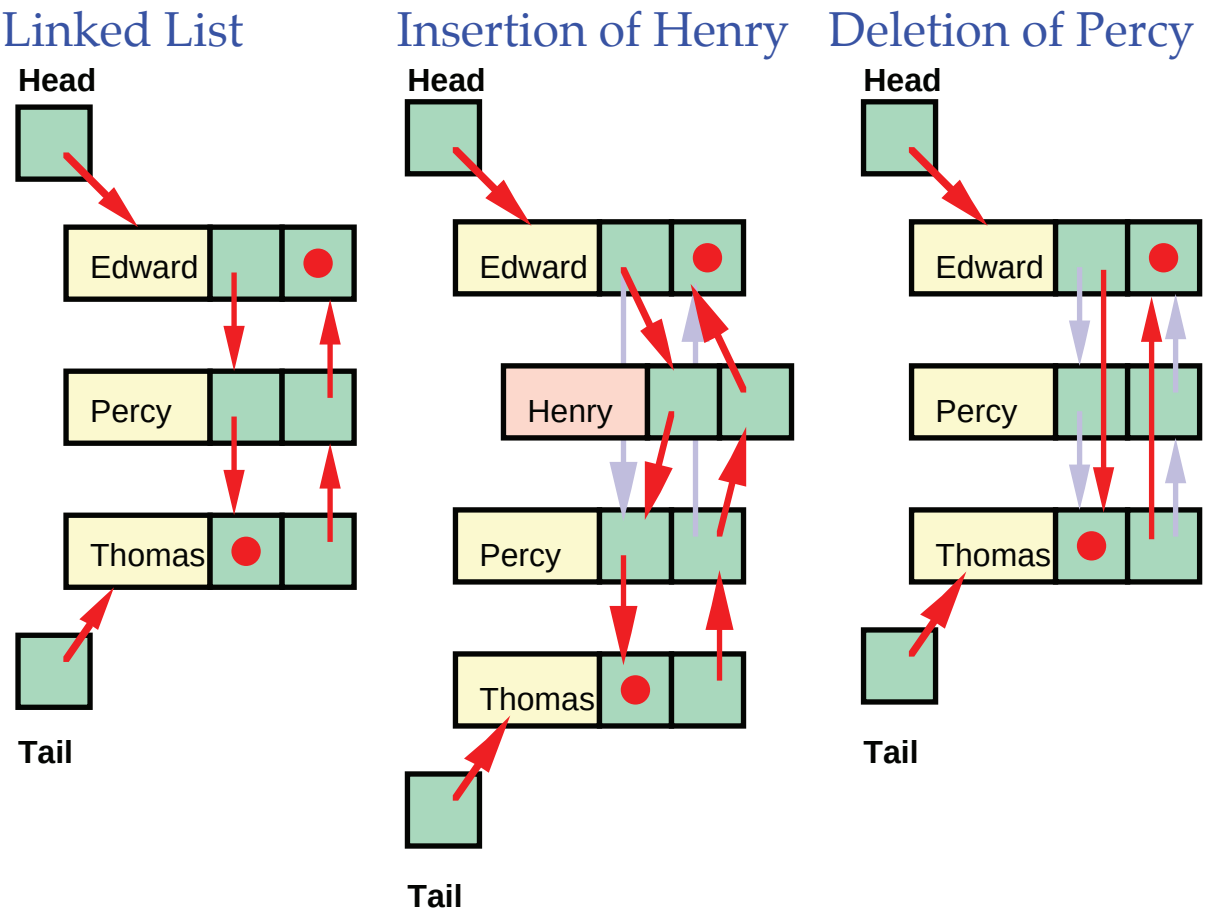
  Potentially allows a programmer to change the collections implementation without changing the code that uses the collection.

6.1. Selected Interfaces used by the collection classes

Interface	Properties of container that implement the interface
`Collection`	Can add objects to the collection Can iterate (visit each member) through the collection Can delete objects from the collection
`List`	Can add objects to the collection at arbitrary positions. Can iterate through the contents of the collection (forward or reverse direction) Can delete objects from the collection at arbitrary positions.
`Map`	Can add [Key, Data] object pairs into the collection Can retrieve a Data item by quoting its Key Can delete a [Key,Data] object pair by quoting the object Key

Interface	Implemented by (selected) classes
`Collection`	ArrayList, LinkedList, HashSet, TreeSet
`List`	ArrayList, LinkedList
`Map`	HashMap, TreeMap

6.2. LinkedList -Possible implementation



The red dot is the end of the double linked list

\+ Easy to add element
Easy to delete element
Access of next element (forward or backward) fast

\- Random access potentially very costly

When storage can no longer be accessed it is returned to the heap (Pool of unused storage) by the garbage collector. Any new storage required will be allocated from the heap of unused storage.

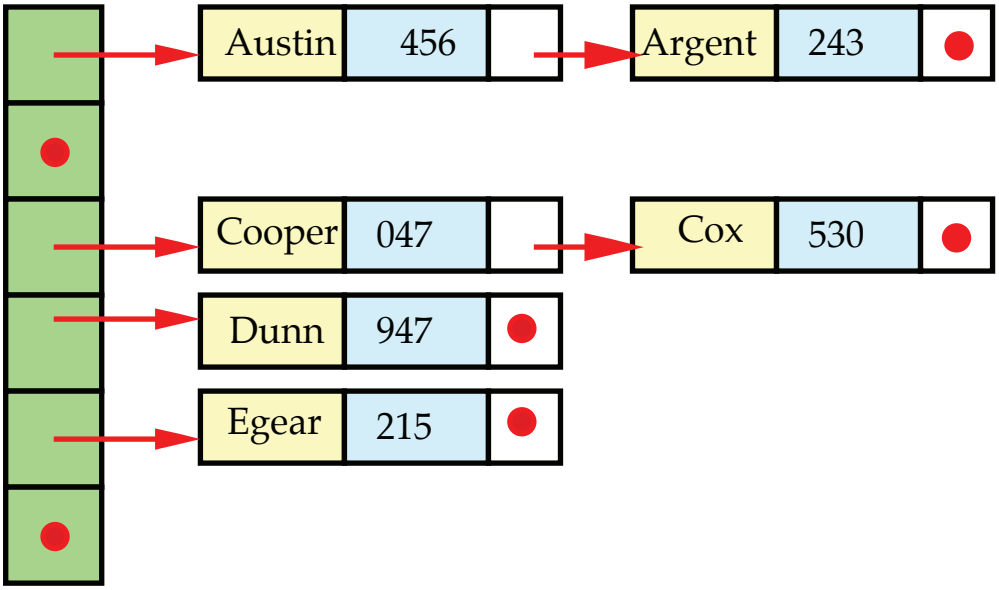
6.3. Data structure: Hash Table

Hash Tables are used to store [Key,Data] pairs. The value can be accessed quickly using the key:

The key (A String) is converted into a number (the hash value). Two or more keys may thus map to the same number. It is good to have the size of the hash table (number of entries) a prime number.

`[Austin, 456][Cooper, 047][ Cox, 530]` etc. entered into a hash table where the hash value (in this example) is simply the first letter of the name. Thus in this case the hash table is 26 elements.

(This of course would not be used for a real hashing algorithm)



- External chaining (As above)
- Internal chaining, find a free slot in the hash table and use this. (For example: add n to the hash value and check if this space is free, ...).

6.4. Efficiency of a hash table / Comparisons

Hash algorithm

An algorithm to turn key into a number, using a prime number for the size of the hash table is considered good.

Successful Search

%load factor	25%	50%	75%	80%	90%	95%
Hash with Chaining		1.25*		1.4*	1.45*	1.5*
Hash internal chaining	1.13*	1.30*	1.51*		1.67*	1.73*

Unsuccessful Search

%load factor	25%	50%	75%	80%	90%	95%
Hash with Chaining		1.11*		1.2*	1.31*	1.3*
Hash internal chaining	1.01*	1.18*	1.50*		1.81*	1.94*

- Notes

  * Approximately number of comparisons to find item

  Load factor % of hash table used.

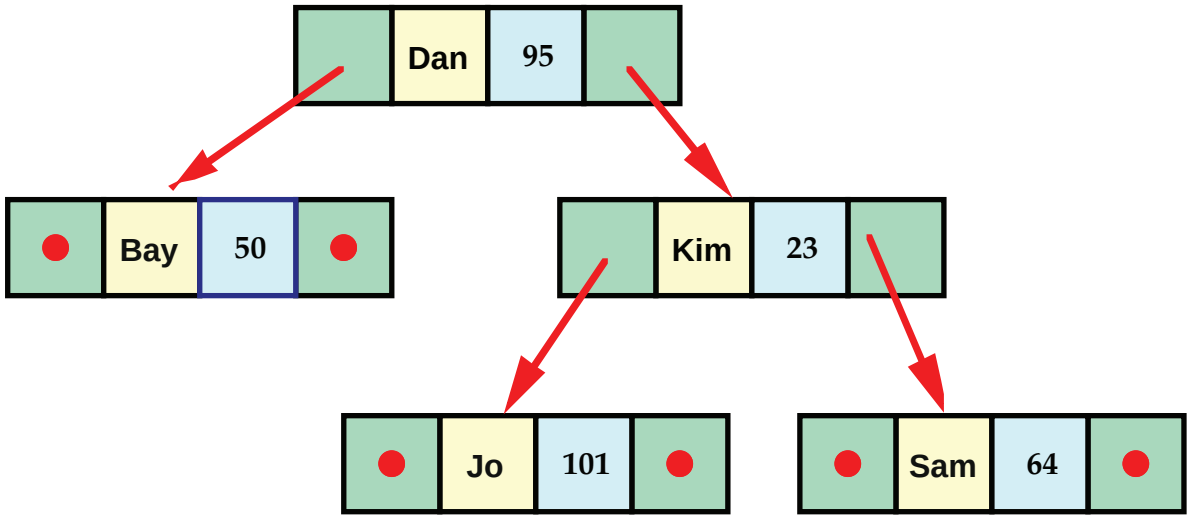
  The number of searches are independent of the table size.

6.5. Data structure: Tree

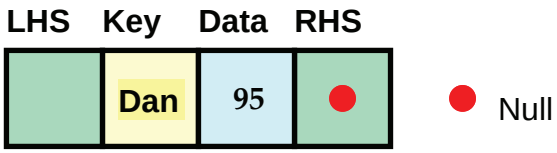
Trees are used to hold [Key, Data] pairs in order, so that reasonably fast access can be made to Data values using the Key. In addition the next item in order can be easily found.

Insertion of the **[key Data]** pairs:
`[Dan,95], [Kim,23], [Jo,101], [Bay,50], [Sam,64]`

into a tree:



Node:



Algorithm to print the contents of a tree:

```
PrintNode( Node n )
{
    if ( n == null ) return;
    PrintNode(n.LHS) // On Left Hand Side
    print( val )
    PrintNode(n.RHS) // On Right Hand Side
}
```

This is a **recursive** algorithm

6.5.1. Efficiency of a TreeMap (Tree balanced)

A balanced tree is when all but the bottom row have sub nodes on the left and right hand side.

Comparisons required (Worst case) to find a [key data] pair in a balanced tree of 1.. N items: is: (int) $\log_2 N + 1$

Data items (N)	Accesses required (Worst case) (int) $\log_2 N + 1$
10	4
100	7
1,000	10
1,000,000	20
1,000,000,000	30
1,000,000,000,000	40
1,000,000,000,000,000	50
1,000,000,000,000,000,000	60

$\log_2 N$ Is essentially how many times does 1 have to be doubled to get to N. N=7-> 3: 1+1=2, 2+2=4, 4+4=8

$\log_2 7$ is 2.8074 (to 4 decimal places)
$\log_2 8$ is 3 (Exactly)
$\log_2 10$ is 3.3219 (to 4 decimal places)
$\log_2 100$ is 6.6439 (to 4 decimal places)
$\log_2 127$ is 6.9887 (to 4 decimal places)
$\log_2 128$ is 7 (Exactly)

Quick Quiz

? Which structure is better for finding items in the fewest number of searches, a binary tree or a hash table?

6.6. Using the collection Interface

```
Collection<String> aRainbow = new ArrayList<>();
aRainbow.add("Violet");
aRainbow.add("Blue");
aRainbow.add("Green");
aRainbow.add("Yellow");
aRainbow.add("Orange");
aRainbow.add("Red");

System.out.println("The colours of the Rainbow are :");
for ( String cur : aRainbow )
{
  System.out.print( cur + " " );
}
System.out.println();
```

6.6.1. Results

```
The colours of the Rainbow are :
Violet Blue Green Yellow Orange Red
```

6.6.2. Signatures in the interface Collection<E>

Signature	Implements
`boolean`	`add( E element )` Append `element` to the end of the collection.
`void`	`clear()` Removes all the items from the collection.
`boolean`	`contains( E element )` Returns true if the collection contains `element`.
`Iterator`	Returns an iterator over the collection. Order depends on properties of the collection
`boolean`	`remove( E element )` Remove `element` from the collection, returns true if this is performed.
`int`	`size()` Returns the size of the collection.

6.7. Using the class LinkedList instead

```
Collection<String> aRainbow = new LinkedList<>();
aRainbow.add("Violet");
aRainbow.add("Blue");
aRainbow.add("Green");
aRainbow.add("Yellow");
aRainbow.add("Orange");
aRainbow.add("Red");

System.out.println("The colours of the Rainbow are :");
for ( String cur : aRainbow )
{
  System.out.print( cur + " " );
}
System.out.println();
```

6.7.1. Results

```
The colours of the Rainbow are :
Violet Blue Green Yellow Orange Red
```

6.7.2. Signatures in the interface List<E>

Signature	Behaviour
`void`	`add( int index, E element )` Inserts `element` at position `index` in the collection.
`boolean`	`add( E element )` Append element to the end of the collection. Returns true if the collection is changed
`void`	`clear()` Removes all the items from the list.
`boolean`	`contains( E element )` Returns true if the list contains `element`.
`E`	`get( int index )` Returns the `index` member of the collection.
`E`	`remove( int index )` Removes the object at position `index`. Returns the removed `element`.
`boolean`	`remove( E element )` Remove `element` from the list, returns true if this is performed.
`E`	`set( int index, E element )` Replaces the `index` member of the collection with `element`. Returns the original `element` at the position `index`.
`int`	`size()` Returns the size of the list.

6.8. What you should know

- That you need to spend at least 5 hours per week on this module in addition to attending lectures, seminars and tutorials.

- That learning a computer programming language is not an activity that you can undertake in the last few weeks of the academic year.

7. Data structures (Continued)

- Data structure HashMap & TreeMap

- Helps solve problems like:
  Need to store key data pair accessing the data item using the key.

7.1. The Map<Key,Data> interface

Signature	Implements
`boolean`	`containsKey( Key k )` Returns true if contains key `k`.
`Data`	`get(Key k)` Returns the object associated with key `k`.
`Data`	`put( Key k, Data d)` Puts the [Key,Data] pair in the map. Returns the previous element association with key, if none then return null
`Set<K>`	`keySet()` Returns the set of all the keys in the map.
`Data`	`remove(Key k)` Removes the key from the map, the data associated with the key is returned.
`int`	`size()` Returns the number of [Key,Data] pairs in the map

7.2. Map interface

Stores key, Data pairs, can retrieve data object by specifying key object.

```
import java.util.*;

class Main
{
  public static void main( String args[] )
  {
    Map<String,String> map = new TreeMap<>();

    map.put( "Gordon",   "Gordon@brighton.ac.uk" );
    map.put( "Thomas",   "Thomas@brighton.ac.uk" );
    map.put( "Edward",   "Edward@brighton.ac.uk" );
    map.put( "Annie",    "Annie@brighton.ac.uk" );
    map.put( "Clarabel", "Clarabel@brighton.ac.uk" );

    System.out.print("Enter a name: " );
    String name = BIO.getString();
    while ( ! name.equals( "END" ) )
    {
      if ( map.containsKey( name ) )
      {
        System.out.printf( "%s's e-mail address is: %s\n",
                           name, map.get( name )  );
      } else {
        System.out.printf( "%s name not know\n", name );
      }

      System.out.print("Enter a name: " );
      name = BIO.getString();
    }
  }
}
```

7.2.1. Results

```
Enter a name: Gordon
Gordon's e-mail address is: Gordon@brighton.ac.uk
Enter a name: OLiver
Oliver name not know
Enter a name: Thomas
Thomas's e-mail address is: Thomas@brighton.ac.uk
Enter a name: END
```

7.2.2. Components

```
Map<String,String> map = new TreeMap<>();
```

The [Key,Data] pair [Gordon","Gordon@brighton.ac.uk"] is added to the map as follows:

```
map.put( "Gordon",    "Gordon@brighton.ac.uk" );
```

Finally, the e-mail address for "Gordon" is retrieved from the map and printed. The variable `name` contains "Gordon".

```
if ( map.containsKey( name ) )
{
  System.out.printf( "%s's e-mail address is: %s\n",
                     name, map.get( name )  );
} else {
  System.out.printf( "%s name not know\n", name );
}
```

```
Gordon's e-mail address is: Gordon@brighton.ac.uk
```

7.3. Wrapper classes

The class `Integer` is a wrapper class for the primitive type `int`. By using implicit boxing and unboxing Java code can be written as if the arithmetic operations were defined when using instances of `Integer`.

Wrapper classes exist for all the other primitive types: `Double`, `Long`, `Float` etc.

```
class Main
{
  public static void main( String args[] )
  {
    System.out.print( "Enter size of box: ");
    Integer size = BIO.getInt();

    for ( Integer depth=0; depth<size; depth++ )
    {
      for ( Integer width=0; width<size; width++ )
      {
        System.out.print( "*" );
      }
      System.out.println();
    }

  }
}
```

- Warning there are dangers in doing this in the general case. So best not to do this.

7.3.1. Results

When the data was
4

```
****
****
****
****
```

7.4. Wrapper classes

Using an object to hold an instance of a primitive type so that it can be used like an object.

```
class Main
{
  public static void main( String args[] )
  {
    plain();
    wrapper();  //Using boxing / unboxing
    expanded(); //Without boxing/ unboxing
  }

  public static void plain()  //10 Bytecode instructions
  {
    int a = 1;
    int b = 2;

    int res  = a + b + 3;
    System.out.printf( "Result = %d\n", res );
  }

  public static void wrapper()  //15 Bytecode instructions
  {
    Integer a = 1;
    Integer b = 2;

    Integer res  = a + b + 3;
    System.out.printf( "Result = %d\n", res );
  }

  public static void expanded() //21 Bytecode instructions
  {
    Integer a = new Integer(1);
    Integer b = new Integer(2);

    Integer res = new Integer( a.intValue() + b.intValue() + 3 );
    System.out.printf( "Result = %d\n", res.intValue() );
  }
}
```

- Bytecode instructions not including return and IO instructions (`System.out.printf`)

7.5. Frequency of words

```
class Main
{
  public static void main( String args[] )
  {
    Map<String,Integer> map = new TreeMap<>();


    System.out.print("Enter a word: " );
    String word = BIO.getString();
    while ( ! word.equals( "END" ) )
    {
      if ( map.containsKey( word ) )              //in Map
      {
        map.put( word, map.get( word ) + 1 );     // Add 1
      } else {
        map.put( word, 1 );                       //Add word
      }
      System.out.print("Enter a word: " );
      word = BIO.getString();
    }

    for ( String aWord : map.keySet() )       //Tree order (sorted)
    {
      System.out.printf("%-10s occurs %d time(s)\n",
                        aWord, map.get( aWord ) );
    }

  }
}
```

7.5.1. Results

When the data was the following words each on a separate line: `the cat sat on the END`

```
cat        occurs 1 time(s)
mat        occurs 1 time(s)
on         occurs 1 time(s)
sat        occurs 1 time(s)
the        occurs 2 time(s)
```

Using this program on the text of the Tempest by William Shakespeare.

The top 20 words used are:

```
and         occurs 513 times
the         occurs 512 times
i           occurs 446 times
to          occurs 324 times
a           occurs 310 times
of          occurs 295 times
my          occurs 288 times
you         occurs 212 times
that        occurs 188 times
this        occurs 185 times
thou        occurs 184 times
in          occurs 162 times
me          occurs 158 times
not         occurs 154 times
is          occurs 150 times
prospero    occurs 139 times
with        occurs 137 times
be          occurs 132 times
it          occurs 131 times
```

and there are 3,286 unique words of which 1,980 occure once.

I used the Unix command line tools `tr` and `sed` to put each word of the play on a new line, strip out blank lines, punctuation characters and convert all words to lower case. Then I sorted the output into descending frequency order using the Unix command line tool `sort`.

The Unix command line tool `wc` was used to count the number of output lines.

7.5.2. Observation

If the line (in the frequency of words program) had been changed from

```
Map<String,Integer> map = new TreeMap<>();
```

to

```
Map<String,Integer> map = new HashMap<>();
```

Then the program would still work. As `HashMap` also implements the `interface Map`.

It would be faster as a `HashMap` is faster to access keys in the map.

However, the output would not be in sorted order. As the natural ordering of the keys in a `HashMap` is "random".

Fixed, by using a sorted set to navigate the keys in the map.

```
SortedSet<String> sorted = new TreeSet<>( map.keySet() );

for ( String aWord : sorted )  //sorted - A sorted set
{
  System.out.printf("%-10s occurs %d time(s)\n",
                    aWord, map.get( aWord ) );
}
```

7.6. Course work

- **Diary**
  Essentially a map
  Date -> Text

- **Could be either**

```
new HashMap <>();
```

or

```
new TreeMap <>();
```

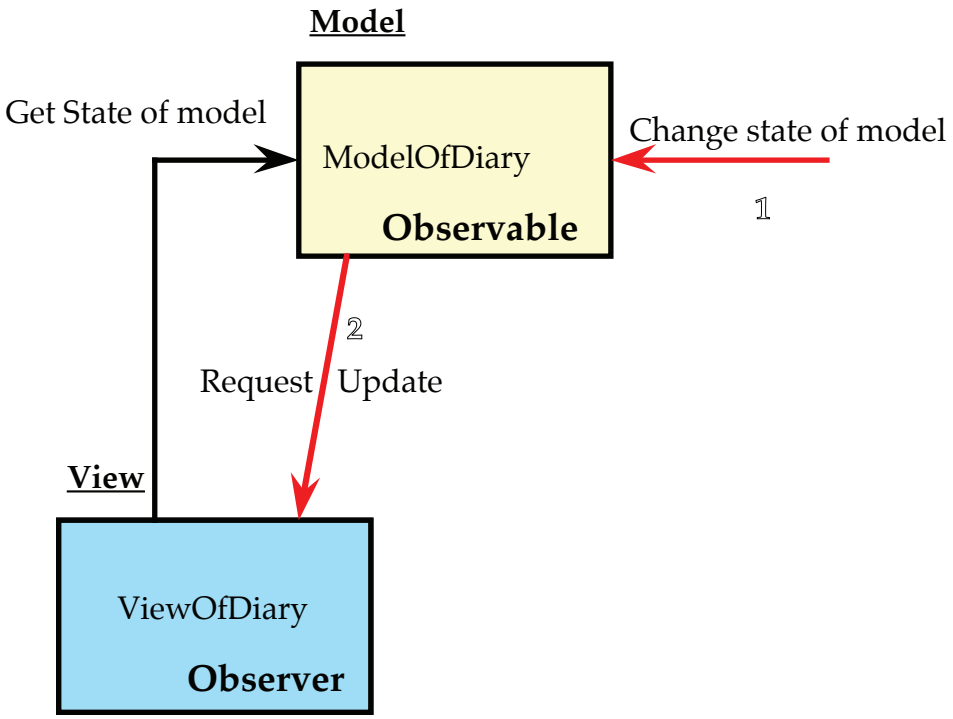
- **Uses an interface**

```
Map<String, String> diary = ...
```

so can choose which implementation to use/ or change the implementation of the map at a later stage.

7.7. Course work demonstration

7.6. Observe Observable

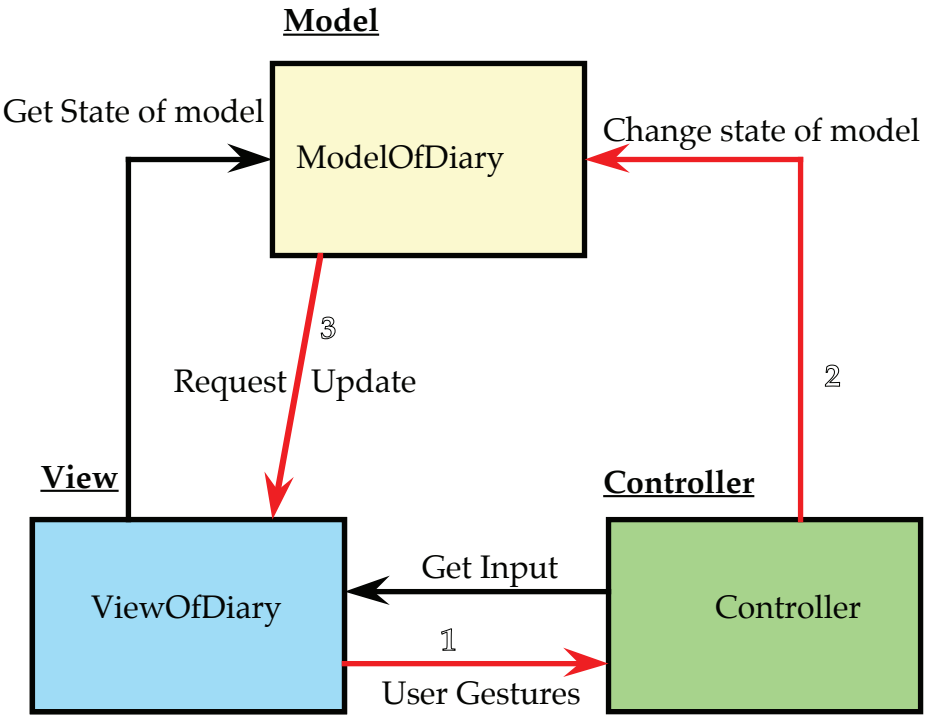


- **View**
  Deals with the display of the artefact (an observer)

- **Model**
  Model of the artefact (an observable object)

7.8. MVC - Model View Controller

- 1978/9 Xerox PARC: Smalltalk group



- **Controller**
  Deals with input to the artefact
  Buttons pressed, text typed into the diary

- **View**
  Deals with the display of the artefact
  The GUI for the diary, show text of day, allow the current day to be changed

- **Model**
  Model of the artefact
  Diary entries, current day, how to add/ subtract 1 to/ from the current year, month, day

```
class Controller
{
  private ModelOfDiary diary;
  private ViewOfDiary  view;

  public Controller( ModelOfDiary aModelOfDiary,
                     ViewOfDiary aViewOfDiary )
  {
    diary = aModelOfDiary;
    view  = aViewOfDiary;
    view.setController( this );
  }

  public void process( String buttonName, String message )
  {
    if ( message != null )
      diary.saveMessageForToday( message );
    diary.changeDate( buttonName );
  }

  public void close()
  {
    diary.close();
  }

}
```

7.9. Outline of code

```
import java.util.*;

/**
 *   Model of a diary
 */
public class ModelOfDiary extends Observable
{
  private DiaryIO dio = new DiaryIO();
  //The diary is represented by a map
  //The key is the diary entry date
  //and the data value is a string representing the diary entry
  private Map< String, String> diary;  //date, message

  private int year = 2016, month = 4, day = 23;

  /**
   * Construct the diary
   * By reloading if possible a saved version
   */
  public ModelOfDiary()
  {
    diary = dio.read();      //Read previous saved diary
  }

  /**
   * Retrieve the message for the current date, from the diary
   * If no event for the date then the
   *  empty "" string is returned
   * @return The diary entry for the current date.
   */
   //The date is held in the instance variables
   //  year, month, day
   //Use the date as the key (A String) to retrieve the message
   //from the map used to represent the diary entries
  public String getMessageForToday()
  {
    //*********************************************************
    //***  You need to add code here                        ***
    //*********************************************************
      return "";
  }
```

```
/**
 * Set in the diary the message for the current day
 * @param message The message to be set in the diary
 */
 //The date is held in the instance variables
 //  year, month, day
 //Use the date as the key (A String) to store the message
 //to the map used to represent the diary entries
 //Note: Map is an interface
public void saveMessageForToday( String message )
{
  //*********************************************************
  //***  You need to add code here                        ***
  //*********************************************************
}

/**
 * Return the number of days in the month for the year
 * month = 1 - 12, Jan = 1, Feb = 2, etc.
 * years in the range 1600 - 2400
 * @param year The year
 * @param month The month
 * @return The days in the month specified
 */
public int maxDaysInMonth( int year, int month )
{
  //*********************************************************
  //***  You need to add code here                        ***
  //*********************************************************
  return 31;
}
```

```
import javax.swing.*;

/**
 * View of a diary (Graphical)
 */
class ViewOfDiary extends JFrame implements Observer
{
  private final int YEAR  = 0;            //year area
  private final int MONTH = 1;
  private final int DAY   = 2;

  private static final int H = 620;       //Height of window pixels
  private static final int W = 280;       //Width  of window pixels

  private JLabel      theMes     = new JLabel();     //Message area
  private JTextArea   theOutput1 = new JTextArea(); //Input area
  private JTextArea   theOutput2 = new JTextArea();  //Result area
  private JScrollPane theSP      = new JScrollPane();
  private JTextField date[]      = new JTextField[3];//Y / M / D


  /**
   * Called when the model has changed
   * @param aModelOfDiary The model of the ticket machine
   * @param arg Any arguments
   */
  @Override
  public void update( Observable aModelOfDiary, Object arg )
  {
    ModelOfDiary model = (ModelOfDiary) aModelOfDiary;

    date[DAY].setText  (String.format("%02d", model.getDay()));
    date[MONTH].setText(String.format("%02d", model.getMonth()));
    date[YEAR].setText (String.format("%04d", model.getYear()));

    String message = model.getMessageForToday();
    String cMessage= message.replace('\n', ' ');

    int len = message.length();
    if ( len > 30) len = 30;
      theOutput1.setText( cMessage.substring(0, len) );
    theOutput2.setText( message );

    theMes.setText( String.format( "%04d  %02d  %02d",
                        model.getYear(), model.getMonth(),
                        model.getDay() )
                      );
  }
```

```
import java.util.*;
import java.io.*;

class DiaryIO
{
  /**
   * Write the map to a disk file called diary.dat
   * @param diary The map representing the diary
   */
  public void write( Map<String,String> diary )
  {
    try
    {
      FileOutputStream ostream = new FileOutputStream("diary.dat");
      ObjectOutputStream oos   = new ObjectOutputStream(ostream);

      oos.writeObject( diary );

      oos.flush();
      oos.close();
      ostream.close();
    }
    catch ( IOException e )
    {
      System.out.println("IOException : " + e.getMessage() );
    }
    catch ( Throwable e )
    {
      System.out.println("FAIL : " + e.getMessage() );
    }
  }
```

Note: Will write out the object `diary` to disk.

```
  @SuppressWarnings("unchecked")
  private static
     Map<String,String> readObject( ObjectInputStream in)
           throws java.io.IOException,
                  java.lang.ClassNotFoundException
  {
    return (Map<String,String>) in.readObject();
  }

  /**
   * Read the map representing the diary from the file diary.dat
   * If fails to read the file diary.dat create an empty map.
   * @return A map representing the diary
   */
  public Map<String,String> read()
  {
    try
    {
      FileInputStream istream = new FileInputStream("diary.dat");
      ObjectInputStream   ois = new ObjectInputStream(istream);

      Map <String,String> diary = readObject(ois);

      ois.close();
      istream.close();
      return diary;
    }
    catch ( Exception e )
    {
      //Failed to read diary, so create an instance of an object
      // that implements the map interface
      return new HashMap< String, String >( 1000 );
    }
  }
}
```

Note: If fails to read 'diary' from disk then creates an empty diary to use. This will be saved when the program exits

- Originally in Java the parameters to a class where always an instance of an `Object`. When the language was changed to allow parameterised types ([generics] `Map<String,String>`), the virtual machine was not updated (for compatibility reasons) and hence the information (run-time) about the type of the parameter to the class is nor held (type erasure).

- The serialization of the state of an object to a disk file/ stream is part of the Java system.

7.10. What you should know

- That you need to spend at least 5 hours per week on this module in addition to attending lectures, seminars and tutorials.
- That learning a computer programming language is not an activity that you can undertake in the last few weeks of the academic year.

8. Data Structures (Continued)

- Building a collection class
- Writing a generic class
- Unit testing: JUnit testing
- Java documentation of collection classes

8.1. Examples of a queue

A queue is a First in First out data structure

- When you buy products from a supermarket you stand in a queue waiting to be served.

- When you type characters at a terminal characters are entered into a queue to be read by the application that has focus in order of arrival.

- When you print a file in the 2nd floor labs, the request goes into a queue and the requests are processed in order of arrival.

8.2. Building a new Collection class [Simple Queue]

Method	Behaviour
`int`	`poll()` Returns the first element at the head the queue. If no first element return `SQueue.NO_DATA`
`boolean`	`offer(int element)` Offers to put an element at the end of the queue, if possible returns true, if not possible returns false.
`int`	`size()` returns the number of elements in the queue

```
class SQueue
{
  //Hope the value NO_DATA is not a valid data item
  public static final int NO_DATA = -99999999;
  private ArrayList<Integer> queue = new ArrayList<>();

  public boolean offer( int element )
  {
    return queue.add( element );
  }

  public int poll()
  {
    if ( queue.size() > 0 )
    {
      return queue.remove(0); //Removes & returns element 0
    }
    return NO_DATA;
  }

  public int size()
  {
    return queue.size();
  }
}
```

? What is wrong with this code

8.3. Creating a generic class

- A class to work with any object, an instance of a bank account, a button on a screen.

  For example, the class `ArrayList` is a generic class as it can hold instances of any class. (e.g. `ArrayList<Object>`) Though normally you want to restrict it to instances of a specific class.

Signature	Behaviour
`E`	`poll()` Returns the first element at the head the queue. If no first element return null
`boolean`	`offer(E element)` Offers to put an element at the end of the queue, if possible returns true, if not possible returns false.
`int`	`size()` returns the number of elements in the queue

8.4. Implementation

```
import java.util.ArrayList;

class SQueue<E>
{
  private ArrayList<E> queue = new ArrayList<>();

  public boolean offer( E element )
  {
    return queue.add( element );
  }

  public E poll()
  {
    if ( queue.size() > 0 )
    {
      return queue.remove(0); //Removes & returns element 0
    }
    return null;
  }

  public int size()
  {
    return queue.size();
  }
}
```

```
import java.util.ArrayList;

/**
 * <B><FONT COLOR="BLUE">Implements a Simple Queue</FONT></B>
 * <BR>The implementation uses an ArrayList for storing elements.
 * @author      Mike Smith
 * @version     1.0
 * @since       2012-03-25
 */
class SQueue<E>
{
  private ArrayList<E> queue = new ArrayList<>();

  /**
   * Adds an element at the 'end' of the queue
   * @param  element to be added
   * @return true if successfully added else false
   */
  public boolean offer( E element )
  {
    return queue.add( element );
  }

  /**
   * Remove the element at the 'head' of the queue
   * @return the element at the 'head' of the queue
   *         or null if the queue is empty
   */
  public E poll()
  {
    if ( queue.size() > 0 )
    {
      return queue.remove(0); //Removes & returns element 0
    }
    return null;
  }

  /**
   * Returns the number of elements in the queue
   * @return the number of elements in the queue
   */
  public int size()
  {
    return queue.size();
  }
}
```

Restricting the class that can be used (Not in exam)

```
// Things that can be put in a Dock

class Dock<Vessel extends Boat> extends ArrayList><Vessel>
{

}
```

```
class Boat                          {   }
class Schooner  extends Boat        {   }
class Catamaran extends Boat        {   }
class Plane                         {   }

class Main
{
  public static void main( String args[] )
  {
    Dock <Schooner>  dock_1 = new Dock<>();
    Dock <Catamaran> dock_2 = new Dock<>();
    Dock <Boat>      dock_3 = new Dock<>();

    // Works as a Schooner is-a Boat, Catamaran is-a Boat
    dock_3.add( new Schooner() );  // Can add a Schooner
    dock_3.add( new Catamaran() ); // Can add a Catamaran

    //Will not compile, as Plane is-a Boat is false
    Dock<Plane> d = new Dock<>();
  }
}
```

Can put into	Anything that **is-a**
dock_1	Schooner
dock_2	Catamaran
dock_3	Boat Schooner, Catamaran

8.5. Testing

```
import static org.junit.Assert.*;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;

public class SQueueTest
{
  static SQueue <Integer> sQueue;

  //Called before every test case method.
  @Before
  public void setUp()
  {
    sQueue = new SQueue<Integer>();
  }

  //Called after every test case method.
  @After
  public void tearDown()
  {
    assertEquals(0, sQueue.size());
    assertEquals(null, sQueue.poll() );
  }

  @Test
  public void test1()
  {
    assertEquals(true, sQueue.offer(1));
    int val = sQueue.poll();
    assertEquals(1, val );
  }

  @Test
  public void test2()
  {
    SQueue<Integer> sQueue = new SQueue<Integer>();
    int val;
    assertEquals(true, sQueue.offer(1));
    assertEquals(true, sQueue.offer(2));
    assertEquals(2, sQueue.size());
    val = sQueue.poll();
    assertEquals(1, val );
    assertEquals(1, sQueue.size());
    val = sQueue.poll();
    assertEquals(2, val );
  }
}
```

8.6. Using BlueJ to create a unit test

Demonstration

To see JUnit tools in BlueJ

Tools -> Preferences -> Miscellaneous

Tick `Show unit testing tools`

Consequence of the use of the method `assertEquals`

When comparing an Integer returned by the method poll in the class SQueue need to use:

```
new Integer( someInteger )
```

as there is no method:

```
assertEquals( int a, Integer b )
```

there is however the method:

```
assertEquals( Object a, Object b )
```

Example of code generated (From manual testing)

```
@Test
public void test1()
{
    SQueue<Integer> q1 = new SQueue<Integer>();
    assertEquals(0, q1.size());
    assertEquals(true, q1.offer(34));
    assertEquals(new Integer(34), q1.poll());
}

@Test
public void test2()
{
    SQueue<Double> q2 = new SQueue<Double>();
    assertEquals(true, q2.offer(3.1415926));
    assertEquals(1, q2.size());
    assertEquals( new Double(3.1415925), q2.poll());
}
```

8.7. What you should know

- That you need to spend at least 5 hours per week on this module in addition to attending lectures, seminars and tutorials.

- That learning a computer programming language is not an activity that you can undertake in the last few weeks of the academic year.

9. Data Structures (Continued)

- Interfaces and the collection classes
- Iterators
- Java documentation

9.1. Types of container

Actual Container	Properties	Implements interfaces

`ArrayList<E>`	Resizable array.
	The collection grows in `capacityIncrement` units
	Efficient random access to the collection using an array index.
		`Collection`
		`List`
		`RandomAccess`
		`Iterable`
		`Cloneable`
		`Serializable`

`LinkedList<E>`	A linked list.
	Efficient insertion and deletion of elements in the collection.
		`Collection`
		`List`
		`Queue`
		`Iterable`
		`Cloneable`
		`Serializable`

`Stack<E>`	Last In First Out (LIFO) data structure.	`Collection`
		`List`
		`RandomAccess`
		`Iterable`
		`Cloneable`
		`Serializable`

`Vector<E>`	The major properties of an ArrayList.
	Plus access to the array is synchronized.
		`Collection`
		`List`
		`RandomAccess`
		`Iterable`
		`Cloneable`
		`Serializable`

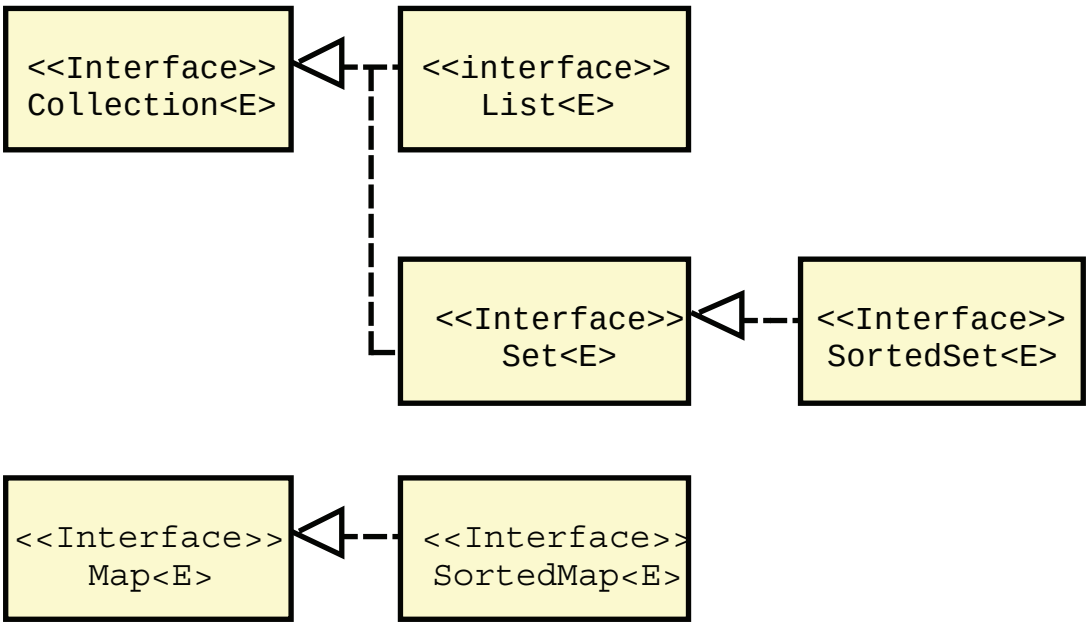
`HashSet<E>`	A set implemented using a hash table.
	The order that the elements are held in is not guaranteed.
		`Collection`
		`Set`
		`Iterable`
		`Cloneable`
		`Serializable`

`TreeSet<E>`	A set implemented using a Tree.
	The elements are held in a sorted order.
		`Collection`
		`Set`
		`SortedSet`
		`Iterable`
		`Cloneable`
		`Serializable`

`HashMap <Key,Data>`	A collection of [Key, Data] pairs.
	The order that the [Key, Data] pairs are held in is not guaranteed.
		`Map`
		`Cloneable`
		`Serializable`

`TreeMap <Key,Data>`	A collection of (Key, Data) pairs.
	The [Key, Data] pairs are held in ascending key order.
		`Map`
		`SortedMap`
		`Cloneable`
		`Serializable`

9.2. Interface Hierarchy



9.3. Interoperability of collection classes

Interface	Supported by
`Collection<E>`	`ArrayList<E>`, `LinkedList<E>`, `Stack<E>`, `Vector<E>`, `HashSet<E>`, `TreeSet<E>`
`List<E>`	`ArrayList<E>`, `LinkedList<E>`, `Stack<E>`, `Vector<E>`
`Set<E>`	`HashSet<E>`, `TreeSet<E>`
`SortedSet<E>`	`TreeSet<E>`
`Map<Key, Data>`	`HashMap<Key, Data>`, `TreeMap<Key, Data>`, `Hashtable<Key, Data>`
`SortedMap<Key, Data>`	`TreeMap<Key, Data>`

9.4. The `LinkedList` class

Implements interfaces `Collection<E>` & `List<E>`

Method	Behaviour	C	L
`void`	`add( int index, E element )` Inserts `t` at position `index` in the collection.		✓
`boolean`	`add( E element )` Append the `element` to the end of the collection.	✓	✓
`void`	`clear()` Removes all the items from the list.	✓	✓
`boolean`	`contains( E element )` Returns true if the list contains t.	✓	✓
`E`	`get( int index )` Returns the indexed member of the collection.		✓
`Iterator`	`iterator()` Returns an iterator, to the start of the collection	✓	✓
`ListIterator`	`listIterator( int index )` Returns a `ListIterator`, at position `index` in the collection.		✓
`ListIterator`	`listIterator()` Returns a `ListIterator`, to the start of the collection.		✓
`E`	`remove( int index )` Removes the object at position `index`. Return the removed object.		✓
`boolean`	`remove( E element )` Remove `element` from the list, returns true if this is performed.	✓	✓
`E`	`set( int index, E element )` Replaces the index member of the collection with `element`. Returns the original object at the position `index`.		✓
`int`	`size()` Returns the size of the list.	✓	✓

9.5. Accessing items in a collection

- ### Overview / Intent
  To access elements in the collection without exposing the collections underlying structure.

- ### Behaviour of an Iterator
  The Iterator object has the behaviour of:

  - Accessing items in the collection
  - Moving through the collection in a forwards or backwards direction.

    The iterator object will remember its 'current position' in the collection.

    `next();`
    A method in the class `Iterator` that will return the next item in the collection and advance the 'cursor' position.

    `hasNext();`
    A method in the class `Iterator` that will return `true` if there is a next item in the collection otherwise will return `false`.

- ### Creating an `Iterator`
  You ask the collection object for an iterator object that will allow you to iterate over the collection.

9.6. Using a `ListIterator`

```
List<String> names = new LinkedList<>();
names.add( "Lord" );  //Adds at end of linked list
names.add( "Jones" ); //Adds at end of linked list
names.add( "Shaw" );  //Adds at end of linked list
```

```
ListIterator<String> it = names.listIterator();
while ( it.hasNext() )
{
  String aName = it.next();
  System.out.print( aName + " " );
}
System.out.println();
```

9.6.1. Results

```
Lord Jones Shaw
```

- Allow a program to visit each item in a collection, but hides from the programmer how the items in the collection are stored.

Can also be written as:

```
for ( String name : names )
{
  System.out.print( name + " " );
}
System.out.println();
```

- But do remember not to try and remove an element from the collection `names`. It will not work.

9.6.2. The `ListIterator<E>` Interface

A `listIterator` can traverse the list in a forward or backward direction. The major methods in the interface `ListIterator` are as follows:

Signature	Implements
`boolean`	`hasNext()` Returns true if there is a next element .
`boolean`	`hasPrevious()` Returns true if there is a previous element .
`E`	`next()` Returns the next element
`E`	`previous()` Returns the previous element
`E`	`get( int index )` Returns the object at position index
`void`	`remove()` removes the previous item returned by `next()` or `previous()`
`void`	`add( object o )` Adds before the next element returned by `next()` or after the next element returned by `previous()`.

9.6.3. Traversing a list in the reverse order

The following code traverses the linked list in reverse order.

```
ListIterator<String> it = names.listIterator( names.size() );
while ( it.hasPrevious() )
{
  String aName = it.previous();
  System.out.print( aName + " " );
}
System.out.println();
```

9.6.4. Results

```
Shaw Jones Lord
```

9.6.5. The `Iterator<E>` Interface

An Iterator can traverse the list in a forward direction only

Signature	Implements
`boolean`	`hasNext()` Returns true if there is a next element .
`object`	`next()` Returns the next element
`void`	`remove()` removes the previous item returned by `next()`

```
Iterator<String> it = names.iterator();
while ( it.hasNext() )
{
  String aName = (String) it.next();
  System.out.print( aName + " " );
}
System.out.println();
```

9.6.6. Results

```
Lord Jones Shaw
```

9.6.7. Inserting an item into a linked list

```
ListIterator<String> it = names.listIterator();
while ( it.hasNext() )
{
  String aName = it.next();
  System.out.println( aName );
  if ( aName.equals( "Jones" ) )
  {
    it.add( "Smith" );    //before next() element
  }
}
```

9.6.8. Results

```
Lord Jones Smith Shaw
```

9.7. What you should know

- That you need to spend at least 5 hours per week on this module in addition to attending lectures, seminars and tutorials.
- That learning a computer programming language is not an activity that you can undertake in the last few weeks of the academic year.

10. Java syntax

- How do you know if a Java program is syntactically correct?
- Need an unambiguous definition of the correct syntax for the programming language Java
- Semantics
  Is the code semantically correct

Idea of a formal grammar

- *[x]* denotes **zero or one** occurrences of x.
- *{ x }* denotes **zero or more** occurrences of x.
- *x | y* means one of either x or y.

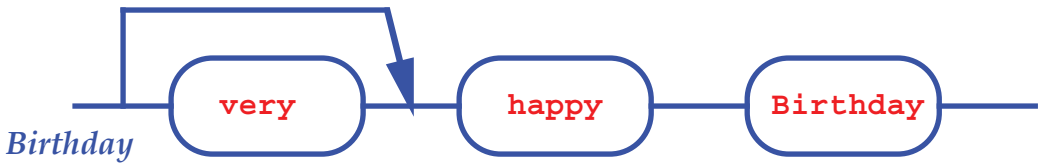
Birthday:

[`very` *]* `happy` `birthday`

```
happy birthday
very happy birthday
```

Key **terminal symbols are in red,**
non terminal symbols in *blue and italisized.*
[ ] { } | **are metasymbols in *blue and italisized.***

Syntax diagram



Idea of a formal grammar

- *[x]* denotes **zero or one** occurrences of x.
- *{ x }* denotes **zero or more** occurrences of x.
- *x | y* means one of either x or y.

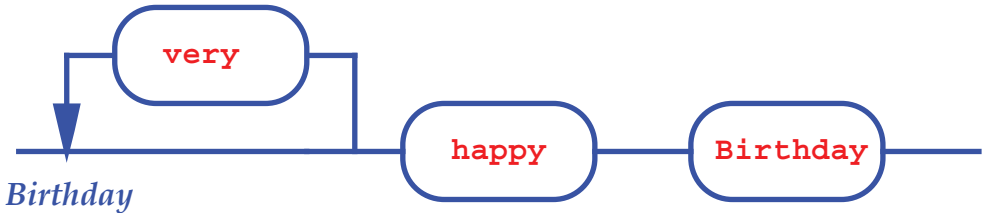
Birthday:
Very `happy birthday`

Very:
{ `very` *}*

```
happy birthday
very happy birthday
very very happy birthday
very very very happy birthday
etc.
```

Key **terminal symbols are in red,**
non terminal symbols in *blue and italisized.*
[ ] { } | **are metasymbols in *blue and italisized.***

Syntax diagram



Idea of a formal grammar

- *[x]* denotes **zero or one** occurrences of x.
- *{ x }* denotes **zero or more** occurrences of x.
- *x | y* means one of either x or y.

aboutanimals:
species speak .

species:
`dog` | `cat`

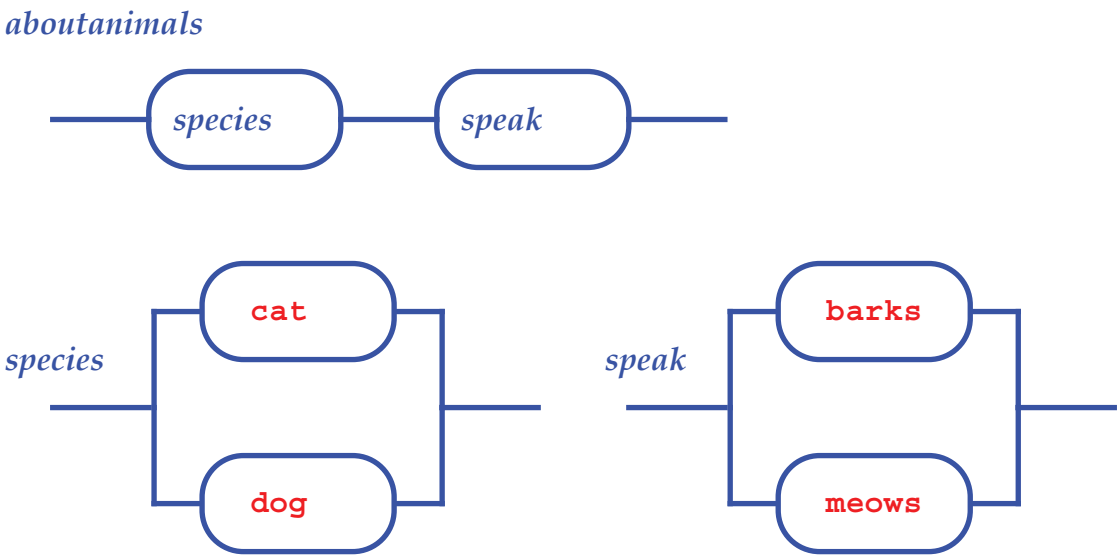
speak:
`barks` | `meows`

```
dog barks .
cat meows .
dog meows .
cat barks .
```

syntactically valid, but sometime semantically incorrect.

Key **terminal symbols are in red,**
non terminal symbols in *blue and italisized.*
[ ] { } | **are metasymbols in *blue and italisized.***

Syntax diagram



10.1. Quick Quiz

Write a grammar to represent the following an unsigned decimal number in the language Java.

An unsigned decimal number is made up of the digits

0, 1, 2 , 3 ,4, 5 , 6 , 7 , 8 , 9 but must not have a leading 0 (unless 0 on its own).

Examples of an unsigned decimal number in Java include:

1 100 12345678 0

A number with a leading 0 is treated as an octal number. An octal number is a number to base 8.

10.3. Ambiguity

Half baked chicken

Half baked chicken

Half baked chicken

10.4. Quick Quiz

What does the following Java program print

```
class Main
{
  public static void main( String args[] )
  {
    if ( 10 > 5 )
      if ( 20 > 30 )
        System.out.println("Branch 1a");
      else
        System.out.println("Branch 2a");


    if ( 10 > 5 )
      if ( 20 > 30 )
        System.out.println("Branch 1b");
    else
      System.out.println("Branch 2b");
  }

}
```

The Java language specification (Java SE8 Edition)
2014 March 18

- *[x]* denotes **zero or one** occurrences of x.
- *{ x }* denotes **zero or more** occurrences of x.
- *x | y* means one of either x or y.

Java Syntax (Fragments simplified)

Block:
{ [*BlockStatements*] }

BlockStatements:
BlockStatement { *BlockStatement* }

BlockStatement:
LocalVariableDeclarationStatement
ClassOrInterfaceDeclaration
[Identifier] Statement

LocalVariableDeclarationStatement:
{ *VariableModifier* } *Type VariableDeclaration* ;

Key **terminal symbols are in red,**
non terminal symbols in *blue and italisized.*
[ ] { } |* are metasymbols in *blue and italisized.

Statement:
Block
;
Identifier **:** *Statement*
StatementExpression **;**
if *ParExpression Statement* [**else** *Statement*]
assert *Expression* [**:** *Expression*] **;**
switch *ParExpression* **{** *SwitchBlockStatementGroups* **}**
while *ParExpression Statement*
do *Statement* **while** *ParExpression* **;**
for (*ForControl* **)** *Statement*
break [*Identifier*] **;**
continue [*Identifier*] **;**
return [*Expression*] **;**
throw *Expression* **;**
synchronized *ParExpression Block*
try *Block* **(** *Catches* | [*Catches*] *Finally* **)**
try *ResourceSpecification Block* [*Catches*] [*Finally*]

ParExpression:
(*Expression* **)**

Key **terminal symbols are in red,**
non terminal symbols in *blue and italisized.*
[ ] { } |* are metasymbols in *blue and italisized.

10.6. What you should know

- That you need to spend at least 5 hours per week on this module in addition to attending lectures, seminars and tutorials.
- That learning a computer programming language is not an activity that you can undertake in the last few weeks of the academic year.

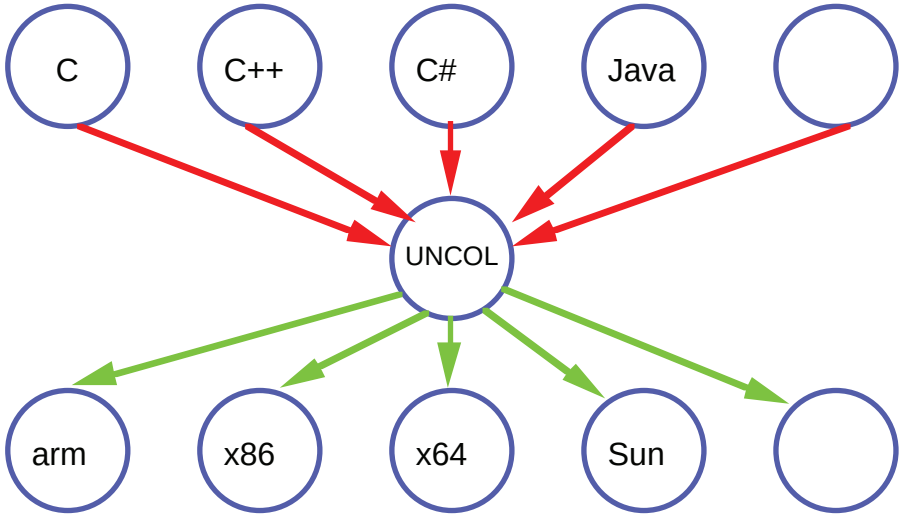
11. Java revisited

- New ideas / Old ideas

- UNCOL - 1954 - 58 (ACM paper) - 1961 proposal

POL	Problem Oriented Language - [Java, JavaScript, C, C++, C#, etc.]
UNCOL	Universal Computer Oriented Language -
ML	Machine Language - machine instructions

Using current languages and machines



Historic implementations

1969		
BCPL	-> OCode	-> Machine code
1978		
UCSD Pascal	-> PCode	-> Interpreted

- JVM - Java Virtual machine (1995)

  Java -> bytecode -> realised program

- .NET framework (2002)

  VB.NET
  C# ->CIL -> CLR
  J#

  CIL Common Intermediate Language
  CLR Common Language Runtime

- Web browser (September 1995 / Netscape Navigator)
  -> JavaScript -> realised program

- Languages translating into bytecode

  Ada, C, COBOL Pascal, Scala

- Languages translating into JavaScript

  Quby [Ruby like], Py2JS [Python like],

- Web browser
  Linux running in a web browser, using a x86 simulator written in JavaScript.

  `http://bellard.org/jslinux/index.html`

11.1. What you should know

- That you need to spend at least 5 hours per week on this module in addition to attending lectures, seminars and tutorials.

- That learning a computer programming language is not an activity that you can undertake in the last few weeks of the academic year.