UNIVERSITY OF LIVERPOOL

PHYS 488

Modelling Physical Phenomena

Lecture 2

Phys488: What we learned in week 1
General structure Java program

In last weeks exercises we practiced: initialising variables, writing to the screen, reading from the keyboard and using some methods from the Math class.

Import any external package you might need.

Definition of the class. (Here the program itself is the class)

```
import java.io.*;
class HelloWorld
{
      static PrintWriter screen = new PrintWriter(System.out,true);
      public static void main(String[] args)
      {
            // this part of the program is called the main method
            screen.println(" Hello World!");
      }
}
```

Definition of the main method. Every program need a "main" method

{ .. } curly brackets to mark the code belonging to a class or the main method
; semi-colon at the end of every line of code
"…" quotation marks to define a text string
// to write comments in the code
Use of indentation to clarify where brackets start and end.

Phys488: What we learned in week 1 (cont.)
We encountered primitive data types

int	**32 bit integers between -2,147,483,648 (-2^{31}) and 2,147,483,647 ($2^{31}-1$)**
long	**64 bit integers between -9,223,372,036,845,775,808 (-2^{63}) and 9,223,372,036,845,775,807 ($2^{63}-1$)**
float	**32 bit real number …. (~ 7 digit precision)**
double	**64 bit real number …. (~16 digits precision)**
char	**Unicode characters (16 bits)**
boolean	**true or false (1 bit)**

Note, the String class we encountered last week is NOT a primitive data type.

Phys488: What we learned in week 1 (cont.)
First encounter with Fields

These are **Field** declarations:

```
int a;
double momentum = 1000.;
```

Sometimes a "**modifier**" comes before the **field** declaration.
For example:

```
final double c = 3E8;
```

Where "**final**" signifies that the value of this field must be set and cannot be changes (appropriate for a physical constant).

Example of a more complicated **Field** declaration:

```
static PrintWriter screen = new PrintWriter(System.out,true);
```

Phys488: What we learned in week 1 (cont.)
First encounter with Classes and *Methods*

We have written 5 different classes (**program ≡ class with a main() method**):

- **InputDataExample**, **PrimitiveDataTypes**, **MathExamples**, **RelativisticDynamics** and **FourVectors**

But we've also used several standard Java **classes** and some of their ***methods***:

- **PrintWriter**: ***println***()
- **BufferedReader**: ***readLine***()
- **Double**: ***doubleValue***()
- **Integer**: ***parseInt***()
- **Math**: ***sqrt***(), ***Pi***, ***atan2***()
- **String**

These are all standard Java classes and methods.

Today we'll write some of our own methods. We'll come to write our own classes as well.

Classes (and their ***methods***) are at the heart of **Object-Oriented** programming.

A very flexible way to have generic bits of software that can be used by multiple programs. These can be:

- tools that are useful in various programs
- Complex data types created for some dedicated purpose (think of Particle example Lecture 1)

Methods (focus this weeks exercises)

The first step in making programs more **modular** is through the use of methods.

For any operation/calculation/manipulation that is done more than once, writing a method makes sense.

The operation is defined **only once** in the program. This is done **outside** the main and can be called from the main many times.

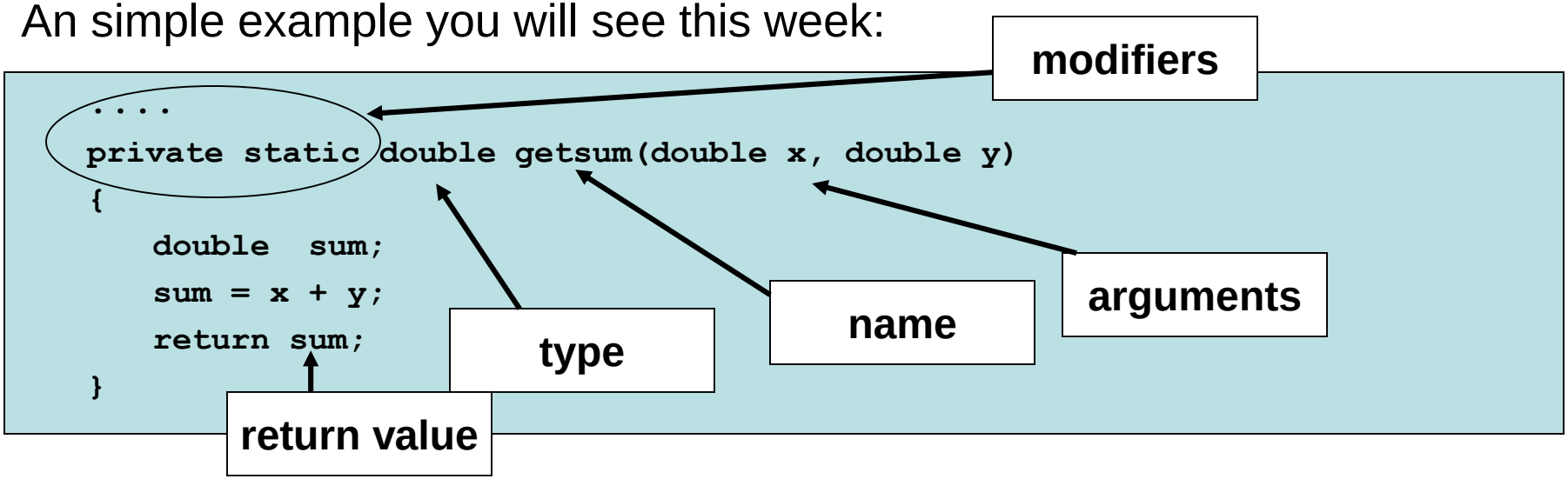
The method becomes a **tool** that can be used many times.

This avoids writing the same code more often than necessary, thus keeping your program as short as possible.

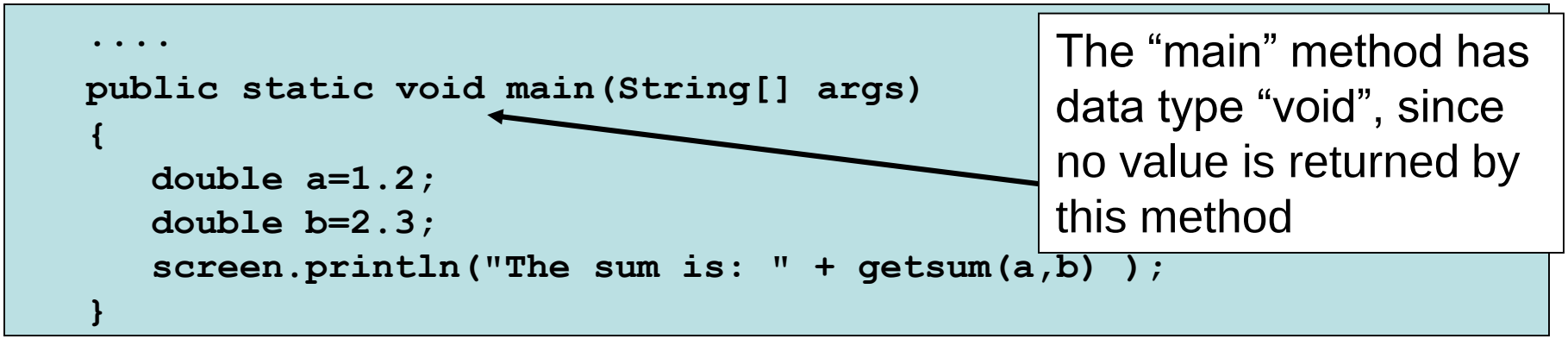
*The more radical step to making code more **modular** (and which defines **Object Oriented Programming**) is the use of additional Classes. We discuss this next week.*

The structure of Methods

An simple example you will see this week:



We have already encountered the “main” method:



In addition to the “main” method a program can have many other methods, which could for example be invoked from the main method. We will see several examples this week:

Arrays

An array is a series of objects of the same type. For example:

```
double a =1.23;
double [] b = new double[5];
b[0] = 1.23;
b[1] = 2.34;
b[2] = 3.45;
b[3] = 4.56;
b[4] = 5.67;
screen.println(" the first value is:, b[0]);
```

Note that in java the position in the array start at position 0.

The following also works:

```
double [] b = {1.23, 2.34, 3.45, 4.56, 5.67};
```

Loops

Useful for things that need to be done more than once.

“for”-loops: for instructions that need to be executed a fixed number of times

```
for (int n = 0 ; n < 10 ; n++)
{
    screen.println(" n = " + n + "  then n^n =  " + Math.pow(n,n) );
}
```

“while”-loops: when it is not known (in advance) how often instructions need to be executed

```
int sum = 0;
while (sum < 100)
{
    screen.println(" Type in a number: ");
    int number= new Integer(keyboard.readLine() ).intValue();
    sum += number;
    screen.println(" The sum stands at:  " + sum);
}
```

“if” (“else”) statements

If the execution of a set of statements depends on a condition

```
screen.println(" Type in a number: ");
int number= new Integer(keyboard.readLine() ).intValue();
if ( number >= 0 )
{
    screen.println(" The number you typed is positive.");
}
else
{
    screen.println(" The number you typed is negative.");
}
```

The “else { ...}“ part is optional.

Phys488: Week 2 Examples and work.

Class SimpleSum. This shows how to divide your program into *independent* blocks of code called *Class Methods*. This is a very important way to write programs.

Class FindRoots. This is a similar program to example (1) but with a more interesting Class Method, called nthRoot.

Class SimpleLoops. Give two examples of constructing a loop. The first way, using the 'for' statement, is useful when you know how many times the loop has to be executed. The second example, using the 'while' statement, is useful when you don't know how many times the loop is to be executed.

Class SimpleArray. Introduces the concept of storing numbers in consecutive locations in memory with one generic name. **This is a very important idea.**

Class Interpolate. This combines techniques seen in examples 3 and 4 to interpolate on a set of data. This means to make an estimate of a number when you only have some nearby values available.

Class GenerateHistogram. This example shows how to generate a large number of random numbers, r, in the range $0 < r < 1$, and make a histogram (frequency distribution) of them.

Class method writeToDisk Class method to save the histogram data in a file on disk to import into Excel to make presentation quality graphs and charts.

(The last three examples are in VITAL so you don't have to type these in completely.)

Example: class SimpleSum

```
import java.io.*;
class SimpleSum
{
     // define the identifiers with class scope.
     static BufferedReader keyboard = new
     BufferedReader (new InputStreamReader(System.in)) ;
     static PrintWriter screen = new PrintWriter( System.out, true);
     //--------Class methods start here------------------------------
     private static double getsum(double x, double y)
     {
          double  sum;
          sum = x + y;
          return sum;
     }
     //------------ End of class methods --------------------------
     // Example of program with a  main method and Class method, simpleSum
     public static void main (String [] args ) throws IOException
     {
          screen.println( "Please type in the first number    " );
          double first = new Double(keyboard.readLine()).doubleValue();
          screen.println( "Please type in the second number   " );
          double second = new Double(keyboard.readLine()).doubleValue();
          double ans = getsum( first, second);
          screen.println(" The sum of these two numbers = " + ans);
     }
}
```

Example: class FindRoots

```
import java.io.*;
class FindRoots
{
     // define the identifiers with class scope.
     static BufferedReader keyboard = new
     BufferedReader (new InputStreamReader(System.in)) ;
     static PrintWriter screen = new PrintWriter( System.out, true);
     //--------Class methods start here------------------------------
     private static double nthRoot(double x, double n)
     {
          // This class method finds the nth root of any real number
          double  logofanswer;
          // warning: this method is unprotected against x <= 0
          logofanswer = Math.log(x)/n;
          return Math.exp(logofanswer);
     }
     //------------ End of class methods -------------------------
     public static void main (String [] args ) throws IOException
     {
          screen.println(" The cube root of 27 =     " + nthRoot(27,3));
          screen.println(" The cube root of 9*9*9 =  " + nthRoot(9*9*9,3));
          screen.println(" The 1.5 root of 9*9*9 =   " + nthRoot(9*9*9,1.5));
          // notice from the output, computers work to a finite accuracy with real numbers
          // QUESTION. Does the following line produce the right answer?If not, why not?
          screen.println(" The 3/2  root of 9*9*9 =  " + nthRoot(9*9*9,3/2));
     }
}
```

Example: class SimpleLoops

```
import java.io.*;
class SimpleLoops
{
     static PrintWriter screen = new PrintWriter( System.out, true);
     static BufferedReader keyboard = new
     BufferedReader (new InputStreamReader(System.in)) ;
     public static void main (String [] args ) throws IOException
     {
          // example 1 , "for" loop.
          int maxn = 10;
          for(int n = 0; n <= maxn; n++)  // n++ means add 1 to current value of n
          {
               screen.println(" When n = " + n + "  then n^n =  " + Math.pow(n,n) );
          }
          // example 2 , "while" loop
          // guessing game.. try to guess the number I'm thinking of
          int guess= 17;
          screen.println(" Hello.. try to guess the integer < 20  I'm thinking about");
          screen. println(" Type in your first guess");
          int yourGuess = new Integer(keyboard.readLine() ).intValue();
          while (yourGuess != guess)  // != means NOT EQUAL
          {
               screen.println(" No, your guess = " + yourGuess + " is not correct, try again");
               yourGuess = new Integer(keyboard.readLine() ).intValue();
          }
          screen.println("Correct! , at last you did it");
     }
}
```

Example: class SimpleArray

```
import java.io.*;
class SimpleArray
{
     // define the identifiers with class scope.
     static BufferedReader keyboard = new
     BufferedReader (new InputStreamReader(System.in)) ;
     static PrintWriter screen = new PrintWriter( System.out, true);
     public static void main (String [] args ) throws IOException
     {   // Program to store integer numbers in an array
          screen.println(" How many numbers do you want to store?");
          int numberToStore = new Integer(keyboard.readLine() ).intValue();
          int [] myStore = new int [numberToStore];
          // create an array called 'myStore' in memory with this number of elements
          // warning..arrays start at ZERO, not 1.
          int m;
          for( int n = 0; n < numberToStore; n++)
          {
               screen.println( "Please type in the " + (n+1) + " number    " );
               m = new Integer(keyboard.readLine()).intValue();
               myStore[n] = m;
          }
          screen.println("\n\n All finished..please check this list");
          for( int n = 0; n < numberToStore; n++)
          {
               screen.println( n + "  location,  value stored = " + myStore[n] );
          }
     }
}
```

Example: class Interpolate

```
import java.io.*;
class Interpolate
{
     static BufferedReader keyboard = new
     BufferedReader (new InputStreamReader(System.in)) ;
     static PrintWriter screen = new PrintWriter( System.out, true);
     public static void main (String [] args ) throws IOException
     {
          /* cross-section in mb for hypothetical process in 12 equal energy steps of 10 MeV
          starting from 10 MeV hence: lowest energy= 10 MeV. Compute highest energy */
          // this is how to put the data directly into an array.
          double [] sigmaOfProcess= { 20, 40, 80, 160, 130, 120 ,110,105,100,100,100,100};
          //  this is  element number        0   1   2   3    4    5    6   7   8   9   10  11
          //    energy                      10  20  30  40   50   60   70  80  90  100  110 120  MeV
          int ndata = 12;
          int binBelow;
          double deltaE  = 10;     // interval in MeV between each entry in the array 'sigmaOfpProcess'.
          double minE =    10;     // lowest energy for which measurements exists.
          double maxE;             // Highest energy for which data exists.
          double inputEnergy;      // Trial energy typed in by the user  (MeV).
          double gradient;         // local gradient d(sigma)/dE  at energy 'inputEnergy'.
          double cross_section ;   // output cross-section computed from table using linear interpolation.
          // as the maximum energy of the array of data is  not given, calculate it in case it is needed.
          maxE = minE + (ndata-1)*deltaE;
          // ask user to type an energy, stop calculating cross-sections when this is zero
          screen.println(" Type in an energy in MeV, zero to finish   ");
          inputEnergy = new Double(keyboard.readLine() ).doubleValue();
          // Now follows an example of using the 'while' statement to put Numeric Sentinel on code
```

Example: class Interpolate (cont.)

```
        while ( inputEnergy != 0 )  //  !=  means not equal to
        // will  only get to here if inputEnergy is > 0
        // identify the array element just below  the energy the user has typed in.
        // this assumes the energy interval between each bin is constant = deltaE.
        {
             // first time through, use numbers just typed in.
             // warning . This is not protected against illegal values of  inputEnergy.
             binBelow = (int) (  (inputEnergy - minE)/deltaE ) ;
             screen.println(" this is just above bin      " + binBelow  );
             // now use linear interpolation to estimate the cross section for this energy
             gradient = (   sigmaOfProcess[binBelow+1] - sigmaOfProcess[binBelow] )/ deltaE;
             // NOTE: above line needs square brackets, [binBelow],  to refer to array elements.
             cross_section = sigmaOfProcess[binBelow] + gradient * (inputEnergy-( binBelow*deltaE + minE));
             screen.println(" The cross-section at  "  +   inputEnergy + "  = " +  cross_section + " barns" );
             //note : it is VITAL to read the keyboard again INSIDE this 'while' loop
             screen.println(" Type in an energy in MeV, zero to finish   ");
             inputEnergy = new Double(keyboard.readLine() ).doubleValue();
        }
    }
}
```

Example: class GenerateHistogram

```
import java.io.*;
import java.util.Random; // notice this..needed to load the class Random.
class GenerateHistogram
{
     static BufferedReader keyboard = new
     BufferedReader (new InputStreamReader(System.in)) ;
     static PrintWriter screen = new PrintWriter( System.out, true);
     public static void main (String [] args ) throws IOException
     {
          final int SIZE = 20; // note : the array elements are numbered 0, 1 , 2 ,3 ,.... SIZE-1
          int [] hist1 = new int[SIZE]; //  The array is filled with zeros, see page 162 Hubbard
          double binlow  = 0; // this is the low edge of the first bin , hist1[0].
          double binhigh = 1; // this is the upper edge of the last bin, hist1[SIZE-1].
          double binsize;     // this is the width of each bin. It is calculated
          double sum;
          Random value = new Random(); /* value is a reference variable (page 30 Hubbard) which points
                                          to the random number class Random. This command
                                          'brings to life' or INSTANTIATES one copy of the class Random
                                          and puts it into memory.
                                          "value" refers to an INSTANCE of the class Random*/
          int trials;
          int bin;
          String anykey;
          double nextone;// primative variable to store each random number as it is generated.
          /* we will generate a large number of random numbers between 0 and 1 and make a histogram.
            The random numbers  should be uniformly distributed, so that all the bins will contain
            the same number of counts. However, there will be statistical fluctuations in the contents
            of the bins */
          binsize = (binhigh - binlow)/( (double)SIZE);  // note the cast (double)
          screen.println( " The width of each bin = " + binsize );
          screen.println( "Input the number of random numbers to generate ");
```

Example: class GenerateHistogram (cont.)

```
        trials = new Integer(keyboard.readLine()).intValue();
        if (trials > 1000000 ) trials =1000000;// trap stupid values that will take ages
        screen.println("\n\n\n   Working.. please wait");
        for ( int n=1; n <= trials; n++)
        {
             nextone = value.nextDouble();
             // the public method nextDouble() is found in the instantiation
             // of the class Random to which the reference variable 'value' points
             if( n < 4) screen.println(" The " + n + "   number is =  " + nextone);
             // calculate which bin of the array should be increased by 1
             bin =(int) ( (nextone - binlow)/binsize );
             hist1[bin]++; // add 1 to the location in the array hist1
        }
        //histogram has been filled. Show the contents on the screen.
        //Also, add up the contents of the bins to see if the sum equals trials
        sum = 0;
        for (int bins =0; bins <= SIZE-1; bins++) //
        {
             screen.println("Bin number " + bins + " contents =  " +  hist1[bins] + "\t");
             sum = sum + hist1[bins];
        }
        screen.println(" the number of trials =" + trials + " , the sum of the contents =" + sum );
    }
}
```

Example: class method writeToDisk

```
// Example of a class method to save the histogram data in a file on disk to import into Excel to make
     presentation quality graphs and charts.
     static BufferedReader keyboard = new BufferedReader (new InputStreamReader(System.in)) ;
     static PrintWriter screen = new PrintWriter( System.out, true);
     private static void writeToDisk(int nbins, int [] h , double low, double dx ) throws IOException
     {
          // This method handles the writing to disk
          String filename ="C:\\hist1.csv";
          FileWriter file1 = new FileWriter(filename);// this creates the file on the A: drive
          PrintWriter outputFile = new PrintWriter(file1);// this sends the output to file1
          // we chose to write the file as a comma seperated file (.csv) so you can read it into EXCEL
          screen.println("Writing to disk, please wait....");
          outputFile.println("Binlow , " + low   );     // note the comma in the text here
          outputFile.println("Binint , " + dx    );     // ditto the previous comment
          outputFile.println("nbins  , " + nbins );     // ditto the previous comment
          // now make a loop to write the contents of each bin to disk, one number at a time
          // together with the x-coordinate of the centre of each bin.
          for (int n = 0; n <= nbins-1; n++)
          {
               // calculate the x coordinate of the centre of each bin
               double binCentre = low + dx/2 + n*dx;
               outputFile.println( n + "," + binCentre + "," + h[n] );
               // note in the above line we specifically write the comma into the file
          }
          outputFile.close(); // close the output file. THIS IS AN IMPORTANT LINE
          screen.println(" Data written to disk in file  " + filename);
          return;
     }
```

Work for Week 2.

Look through the above examples, copy the code into BlueJ and run the programs. Make sure you **understand the points about Java and general computing techniques** that are being made. If you don't understand, ask!

Do you understand why the programm FindRoots give an unexpected result on the last line of output?

Take your first week's four-vector program and modify it to use a **for** loop to read in the input values as well as an **array** to store the vectors. Also **add three separate Class Methods** to this class:

1. a class method that returns the momentum of a given four-vector,
2. a class method that returns the invariant mass of a given four-vector,
3. a class method to add together 2 four-vectors.

Make sure you use these methods from the main.

[1.5]

Modify the **GenerateHistogram** class to histogram numbers in the range $0.4 < r < 0.9$, (i.e.change *binlow* and *binhigh*) and add code to count the number of random numbers that fall below (the underflows) or above (the overflows) the range of the histogram. Also **print out the statistical error** on the contents in each bin, both as a number of counts **and** as a percentage of the total in each bin.

[1.5]

Work for Week 2 (continued)

The class method **WriteToDisk** takes the histogram and writes *some* of the data to disk so that it can be imported more easily into EXCEL to make nice graphs/charts etc. The class method is accessed by a single line:

writeToDisk(SIZE, hist1, binlow, binsize);

to be added to the program at the very end of the main method. Notice how the array **hist1** is passed to the class method as a parameter in the argument list. Using an array like this is a convenient way of passing many parameters *either way*, to or from(!) a class method. The 'return' command can only return *one* value.

Work to do with writeToDisk method:

Add the **writeToDisk** method to the **GenerateHistogram** class. Modify it to also save, in the file, the **statistical errors** and the **number of trials, underflows** and **overflows**. Also modify it to ask the user to **type in the filename** rather than build it into the code as is done at present.

[1.5]

Import the output file from **writeToDisk** in to Excel and make a graph of the histogram you produced, using the bin centre as the x value, the contents of each bin as the y value and the statistical error as the error bar on the y value.

[1.5]

Submit the report for the exercises for weeks 1 & 2 to VITAL before the lecture in week 3!

Stuff to read for next week

Chapters that discuss what is covered in week 3 lecture and exercises:

6.1..6.4 (not a particularly good introduction however, so …)

Some further information:

"Modifiers" are placed before field, class or method declarations. (see also section 6.4 in Hubbard)

private/public: determines whether fields or methods can be accessed from outside or only from inside the class in which they are declared.

Example:

- the "main" method must always be "public"
- other methods we declare in our programs will often be "private".

final: the initial value given to this variable cannot be changed (use this when you define a constant)

static: the value can be accessed throughout the class. I.e. every method in the class has access to this variable. (there is a bit more to the meaning of **static**, to which we'll get later on..)