PHYS 488

Modelling Physical Phenomena

Joost Vossebeld

Lectures developed by Mike Houlden

Lecture 1

PHYS488: The main objectives

Introduction to the use of an Object Oriented (OO) programming language (Java).

To learn how to set up a realistic model of a given physical problem.

To gain experience in testing/verification of a model.

Working in a group, reporting and presentation of the work.

PHYS488: Schedule & Assessment

Weeks 1-5: lectures and 5 weekly exercises, with bunched hand-in for weeks: (1+2), (3+4) & 5. Initial training in Java and general computer techniques

- lecture on Tuesday (10am, BROD-106)
- practical sessions on Tuesday mornings (10am-1pm, CTL6-PCTC-Orange), Tuesday afternoons (2pm-5pm, CTL6-PCTC-Orange) and Wednesday mornings (10am-1pm, CTL6-PCTC-Orange). **Room allocations can change, so keep an eye on ORBIT and my emails**

Weeks 6-10:Significant programming project within your project group that will form the basis of your project report and presentation.

- Week 6: short lecture on Tuesday 10am in room BROD-106 to start of the projects.
- Weeks 6-10: work on projects during practical sessions on Tuesdays and Wednesday mornings and in your own time.

Week 11: presentations and handing-in of reports

Assessment: The proportion of marks awarded for each part is as follows;

• 5 sets of weekly exercises (week 1+2 12%, week 3+4 12%, week 5 6%)	30%
• Group project report	20%*
• Individual project report (2x15%)	30%*
• Oral presentation	20%

* Group and individual reports will be double-marked, as with other project reports.

PHYS488: Some technical points.

Attendance:

Students are expected to attend **all** lectures and practical sessions

Only exception is the practical sessions during weeks 1-5, where attendance is only required until the work on that weeks exercise has been finished and the report is handed-in (in weeks where a hand-in is due!). In the weeks 6-10, during the project work, all practical sessions should be attended.

Demonstrators will be available to help during the practical sessions. Working "at home" for extended periods without discussing your work and progress with the module teachers is not tolerated.

Please scan your student ID for our attendance record for both lectures and practical sessions.

Book:

For the duration of the semester you can borrow the book on programming in Java. John R Hubbard's book ***"Programming with Java"*** Second Edition. This covers most of the Java we will meet in this module. You should bring the book with you to the lab sessions.

PHYS488: Some technical points (cont.)

Exercises weeks 1 to 5:

Reports should be written in a comprehensible manner. Assume the same standard applies as for the year 1, 2 and 3 lab reports. The reports should always include

- An introduction
- A clear **description of the work** done, including
  - Java aspects used,
  - Physics aspects of the programmes
  - Mathematical aspects of the programmes
- **Print-outs, input and output** of the code you've written (make sure these are readable!) ,**tables** , **graphs** with appropriate axis labels, error bars etc.,
- Any other requested information
- **Description and results of Desk Checks,** i.e. evidence that you checked your code.
- **Descriptions of any Problems** or **Errors** you encountered, or **Questions** you asked the demonstrators, and their **solutions**.
- A brief summary

*The reports for the weekly exercises **should be submitted for marking as soon as the work is finished and no later than <u>before the next lecture</u>.** Exercises must be submitted electronically as a single WORD document using the VITAL system.*

*Exercises handed in late, after marked work has been returned to other students, **will not be awarded any marks.***

***Save all work** you produce on a memory stick or on your own disk space on the University system. **You are responsible** for ensuring you always can **always** retrieve a copy of your work from **any week**.*

PHYS488: Some technical points (cont.)

<u>**Group Projects:**</u>

- In **week 5** you will be divided into groups of up to 4, to work together on the final weekly exercise, and subsequently on the project. You are welcome to choose the people you want to work with and tell me at the appropriate time.
- In **week 11** you will EACH give an individual 12-15 minute continuous presentation, using PowerPoint, about the group project and your contribution to it.
- The deadline for handing in the final individual and group report is **the end of week 11**; University Rules apply for late submission.
- More details on what is expected in the project presentation and reports will come in due time.

The Java Programming language

Java is a commonly used *High Level Programming* language based on the idea of *Object Oriented Programming*.

Less High-Level then MathLab $\Rightarrow$ bit harder to learn, but more flexible

Used commonly for internet applications. (“applets”)

Other high level OO languages are C++, …

High Level programming languages

Computers make simple manipulations of binary numbers.

To use computers for solving more complex problems we need to translate these problems into (many) much simpler steps. This is called *programming.*

Luckily we don't have to start from scratch.

High-Level Programming languages provide us with many standard, complex operations, ready for us to use:

- variable declaration
- mathematical functions
- input and output of data
- .. and much more

Object Oriented Programming

Object Oriented (OO) programming is a modern programming style that allows/forces/encourages to develop software in a structured manner.

Users add their own tools to the already available ones.

For example:

• further mathematical functions

• more complex programming structures (“Classes”), adapted to the physical entities or systems they represent.

Practical Advantages:

• write code in such a way that functional parts of it can easily be used elsewhere or re-used later. You should never have to write the same code twice.

• hide unnecessary technical detail! So programmers/users don’t need to see (or even know!) all the details.

(I will also skip a lot of technical detail in this module)

The basics of writing and running a program

Writing the program itself (*code* or *source code*). Mostly we would use some dedicated text editor to do this.

Compiling the program. A compiler is used to translate our *high-level code* into something the computer can deal with (*machine code*).
Most compilers will also warn us in case of any errors in our code.

Execution. Once a program has successfully been compiled we can *run* or *execute* the program.

For our Java course we use the *BlueJ IDE (Integrated Development Environment)* . This free software package acts as an intelligent editor to *write* Java code, *compile* it and *run* it.

PHYS488: Java information on the Web

Link to the main Java site: **http://www.oracle.com/technetwork/java**

There is lots of useful stuff on this site. e.g. the mathematical facilities available in the class Math can be found there; at URL:

http://docs.oracle.com/javase/8/docs/api/java/lang/Math.html

For more advanced information on other Java classes/packages you can look at

http://docs.oracle.com/javase/8/docs/api/index.html

Another excellent place to find more information on Java is the URL

http://www.csc.liv.ac.uk/~frans/OldLectures/COMP101/comp101.html

This is the material presented to Year 1 students in Computer Science. It contains a lot of examples that are clearly explained. (Note this course is a much more extensive treatment of various aspects of object oriented programming than the current module)

http://math.hws.edu/javanotes/ (a free e-book on programming Java)

PHYS488: First Steps, working in BlueJ

In this module we use the BlueJ IDE (Integrated Development Environment) as an intelligent editor to write code, compile it, run it and print off program listings.

Proceed as follows to put in a new class (program) in a new project.

1. Start BlueJ running
2. Under the Project menu choose "*New Project*"
3. Change the "*Look in*:" option to point at the disk where you want to store your files.
4. In the "*File Name*" box type in the name of the Folder you want to create, e.g. *Project1*. Click the "*CREATE*" option and that folder will be created. **All java files for this project will be stored here.**
5. In the BlueJ window click the "*New Class*" option: when prompted, type in the name of the class you want to create; click the *OK* option.
6. In the BlueJ window an orange box with the class name now appears. Double click on this, and **an editor window** will open *showing a template for the new class*. **I suggest you select all this text by dragging the mouse and then delete it.** Note the *Options > Preferences > Editor* menu at the top lets you change some options, such as the font size shown on the screen.
7. Now type in the Java code for the program, as shown in the examples.
8. To print out a hard copy of the code choose the "*Print*" option under the "*Class*" menu.

PHYS488: First Steps, working in BlueJ

9. Once the code is typed in, click *“Compile”* to compile the code. It will prompt you if it finds errors. **Compiling will automatically save the file for you**.
10. Once the code has compiled correctly you run it as follows
    a) Close the editor window
    b) RIGHT CLICK with the mouse on the orange class box in the BlueJ window, then select the *“void main (String[] args)”* option.
    c) When the Method Call box pops up, click “*Ok*”.
    d) A Terminal window will open showing the program output (if there is any).
    e) You can print **this directly** or to save it to a file, under the *Options* menu in this window choose “*Save to file*…”.
    f) Choose the folder (click on the name) then Type in the file name you want to save this as. It will be saved in that folder as a .txt file. This can be opened with MS Wordpad and printed if needed.
11. To modify the code in your program, open the editor again by double clicking on the orange class name box in the BlueJ window.
12. Go around this loop (points 7 to 12) until you have finished this class.
13. Choose the *New Class* option to make another program and start from option (5) above. This new class will be included in the same project.
14. If you close BlueJ and re-start it again, on your own machine it will open the same project you were in when it was closed.
15. To delete a class from the project, **right click** on the orange class box and choose the *REMOVE* option

The structure of a simple Java Program

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 needs a “main” method

The structure of a simple Java Program (cont.)

```
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!");
     }
}
```

A few more things to notice:

{ .. }	**curly brackets to mark the code belonging to a class or the main method**
;	**semi-colon at the end of (most) lines 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.

The HelloWorld program: writing output to screen

```
import java.io.*;
class HelloWorld
{
     static PrintWriter screen = new PrintWriter(System.out,true);
     public static void main(String[] args)
     {
          screen.println(" Hello World!");
     }
}
```

PrintWriter : example of the use of a pre-existing Java class with some useful high level tools (for writing output).
(Some more explanation at the back of the lecture notes. More on the use of classes, like this one in the following weeks)

"`System.out`" is the standard output stream of the computer, usually the screen.

Adding some variables

```
import java.io.*;
class OutputExample
{
     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!");
          int numweights =  2;
          screen.println(" The number of weights is " + numweights);
          double mass =10.5;         // mass of something in kg
          double velocity = 3345;   //velocity in meters per second
          double momentum = mass*velocity;
          screen.println(" The momentum is " + momentum + " kg m/s");
     }
}
```

“int” and “double” are *<u>primitive data types</u>* in Java.

This program will print the following lines to the screen:

```
Hello World!
The number of weights is 2
The momentum is 35122.5 kg m/s
```

Get a number from the keyboard

```
import java.io.*;
class ReadNumber
{
     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
     {
          screen.println( "Please type in the mass in kg " );
          double mass = new Double(keyboard.readLine() ).doubleValue();
          screen.println(" Mass input was " + mass +  " kg");
     }
}
```

BufferedReader: example of use of a pre-existing Java class with some useful high level tools (for reading input).

String: class to store and manipulate text

Double: class with some commands to manipulate variable of type “double”. For example to interpret a string of characters “1.234e5” as a “double” value. (note upper/lower case “d”)

“`System.in`” is the standard input stream of the computer, usually the keyboard.

“`throws IOException`” warns the computer the program must be interrupted if input error occurs.

Work for Week 1

Please do the following exercises and submit a report, as a Word document, **to VITAL as soon as the work is finished**, and no later than before the lecture next week.

The computers to be used are the **University machines** belonging to the Managed Network. There should be demonstrators on hand on Tuesday, and on Wednesday morning in the allocated PC centres.

1) Ensure you can start the BlueJ IDE, type in a simple program, run it and print out the output. To compile code click the 'compile' button. To run code **RIGHT CLICK** on the orange box with the chosen class in the initial BlueJ window, and choose the *void main(args)* option. (if you cannot find BlueJ, you can download it from www.bluej.org)

2) Save any programs you produce on a memory stick, or your own disk space on the University system. **You are responsible** for ensuring you always can **always** retrieve a copy of your work from **any week**.

3) You should visit the official BlueJ site at www.bluej.org and download a copy of the BlueJ manual for yourself from http://www.bluej.org/doc/bluej-ref-manual.pdf
   You can also download BlueJ and the Java development kit onto your own home machines from the web if you have an internet connection.

PTO

Work for Week 1 (cont.)

4) Programs to write:

a) Type in the InputDataExample.java program, run it and ensure you are happy with what it is doing. Visit the URL http://docs.oracle.com/javase/8/docs/api/java/lang/Math.html and download or print off a copy of this documentation. Type in and run the Classes *MathExamples* and the *PrimativeDataTypes* that you have been given. Again, ensure you appreciate what these examples are showing you about Java.

[1 mark]

b) Produce a Java program to calculate the examples a) to g) given on pages 2 and 3 of the hand-out "**Notes on Relativistic Dynamics**". Write the program so that you can type in things like the mass or speed of the particle. Ensure you can do relativistic calculations using MeV, MeV/c and MeV/c^2 units.

[3 marks]

c) Produce a simple Java program that asks the user for the E, px, py and pz values of the four-vectors of two particles (in MeV, MeV/c and MeV/c^2 units). Use these values to calculate and print to the screen , the absolute momentum and the mass (using $E^2=p^2+m^2$) of the two particles. Then add the two four-vector together to get a third four-vector and calculate (and print to screen) the absolute momentum and mass of the later as well. Run the program using the following input values for four-vector 1=(5, 4, 0, 0) and four-vector 2=(5, -4, 0, 0)

[2 marks]

Write a report on the work done that satisfies the criteria listed on slide 5. (NOTE: the work for week 1 will be handed in together with that for week 2 as 1 word document, at the end of week 2. **It is strongly recommended to discuss your write-up for week1 with the module organiser during week 2 so you have some feedback, before handing in the combined week 1+2 report for marking.**

Example: class InputDataExample

```java
// Example of program to input some data
import java.io.*;
class InputDataExample
{
    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
    {
        String name; double height;int age;
        screen.println( "Please type in your name " );
        // this next line gets a string of characters from the keyboard
        name = keyboard.readLine();  // note the L in readLine()
        screen.println("\n\nHello  " + name );// this outputs the characters
        screen.println( "Please type in your height in metres " );
        // this next line converts the characters to a floating point variable
        height = new Double(keyboard.readLine()).doubleValue();
        screen.println(" Thank you, your height is " + 100*height +  " cm");
        screen.println(" Please type in your age in years");
        // the next line uses the new Java method parseInt()
        // to convert the characters to an integer number
        age = Integer.parseInt(keyboard.readLine());
        int yearOfBirth=2009-age;
        screen.println(" You were probably born in " + yearOfBirth);
    }
}
```

Example: class PrimativeDataTypes

```
import java.io.*;
class PrimitiveDataTypes
{
    static PrintWriter screen = new PrintWriter( System.out, true);
    public static void main (String [] args)
    {
        // declare some primative variable types
        boolean b = false;
        char c ='R';                      // need quotes '' to store the Unicode value
        byte j = 127;                     // this is maximum value allowed (8 bits)
        short k=32767;                    // short integer maximum value (16 bits)
        int m = 2147483647;               // integer maximum value (32 bits)
        long n =9223372036854775807L;     // maximum value for long integer (64 bits)
        float x =3.14159F;              // float variables are accurate to about 7 decimals
        double y =Math.PI;              // double variables are accurate to about 15 decimals.
        screen.println(" The value of b = " + b  );
        screen.println(" The value of c = " + c  );
        screen.println(" The value of j = " + j  );
        screen.println(" The value of k = " + k  );
        screen.println(" The value of m = " + m  );
        screen.println(" The value of n = " + n  );
        screen.println(" The value of x = " + x  );
        screen.println(" The value of y = " + y  );
    }
}
```

Example: class MathExamples

```
import java.io.*;
class MathExamples
{
      static PrintWriter screen = new PrintWriter( System.out, true);
      public static void main (String [] args)
      {
           // Find the square root of a number
           double r = 12.33;
           double ans = Math.sqrt(r);
           screen.println(" The square root of " + r + "  is = " + ans);
           // Pi is built in
           double area = Math.PI*r*r;
           screen.println(" Area of circle with radius " + r + " is = " + area);
           /* The method Atan2(y,x) gives the angle of line from the
           origin to (x,y) in the correct quadrant */
           double x = -1;
           double y = -2;
           double angle = Math.atan2(y, x);
           // returns angle in RADIANS in the range -PI < angle <PI
           screen.println(" Angle = " + angle + " rad, or " + angle*180/Math.PI + " degrees");
           // This also works if you have the sine and cosine of the angle
           double sinTheta = y/Math.sqrt(x*x + y*y);
           double cosTheta = x/Math.sqrt(x*x + y*y);
           double angleb   = Math.atan2(sinTheta, cosTheta);
           screen.println(" Angleb = " + angleb + " radians");
      }
}
```

Stuff to read for next week:

Chapters that discuss what is covered in week 1 lecture and exercises:

1.5, (1.6), (1.8), 1.11, 1.12, 1.13, 2.9 + some extra (advanced) notes on the following slides.

Chapters that discuss what is covered in week 2 lecture and exercises:

3.1, 4.1, 4.2, 5.1, 5.2, 5.3, 7.1

Some further information

In the program ReadNumber there was an example of the use of very condensed code, which occurs sometimes in Object-Oriented programming:

```
        . . . . .
        double mass = new Double(keyboard.readLine() ).doubleValue();
        . . . . .
```

is equivalent to

```
        . . . .
        String text = keyboard.readLine();
        Double x = new Double(text);
        double mass = x.doubleValue();
        . . . .
```

The first reduces the length of a program and in this case also reduces memory usage. However, it can also make the program harder to understand. Use sensibly…

PrintWriter and *BufferedReader* Classes

Example of a pre-existing Java classes with some high level tools we will use.

PrintWriter:

http://docs.oracle.com/javase/8/docs/api/java/io/PrintWriter.html

Set of basic commands (`printf,format,println,..`) for writing output. We can choose to send the output to the screen (`System.out`) or for example to a file.

BufferedReader:

http://docs.oracle.com/javase/8/docs/api/java/io/BufferedReader.html

Set of basic commands (`read,readLine,skip,..`) for reading character input. We can choose to read the input from the keyboard (`System.in`) or for example from a file.

String and *Double* Classes

Example of a pre-existing Java class with some high level tools we will use.

String:

http://docs.oracle.com/javase/8/docs/api/java/lang/String.html

Class to store and manipulate text

Double:

http://docs.oracle.com/javase/8/docs/api/java/lang/Double.html

Class with some commands to manipulate double variable (beyond the standard operations for the Java type double). For example to interpret a string of characters “1.234e5” as a floating point value.