Course Information

- **Purpose**
  - Learned Java before – getting refreshed
  - New to Java - getting started
- **Useful for other courses**
  - OOD, CRT, PSM etc.
- **Mode of study**
  - 3 x 1 hour lectures + 3x2 hours supervised lab practicals
  - Self-study throughout the first term
- **Assessment**
  - An hour unseen written exam (week 9)
  - Not counted as any credits for your degree

Plan for this Session

- Getting started
- Objects and classes
- Data types in Java
- Operators and control structures
- Java methods
- Terminal input and output
- Object encapsulation

Getting Started

```java
public class First {
    public static void main(String args[])
    {
        System.out.println("Welcome to the University of York");
    }
}
```

Java source code ends with a `java` command to compile: `javac xxx.java`
A First Program

Run the program, not very exciting really!!

Java bytecode code ends with a .class
Command to run the Java program - java xxx (without .class postfix)

Installing Java

• You must install a Java 2 Standard Edition (J2SE) Software Development Kit (SDK)
  You can find the J2SE SDK from http://java.sun.com/j2se/
• Can you distinguish SDK, JDK and JRE?

Development Environment

• Command line interface
  – E.g. Notepad
  – Need to set Java environment variables
  – E.g. path=.;C:\Program Files\Java\jdk1.6.0\bin
• IDE - Integrated development environment
  – an editor (very like Word)
  – a compiler and an interpreter (JDK)
  – a debugger
  – there are many, e.g. Eclipse, JCreator, JBuilder, JGrasp etc.

Structured Programming

• Traditional approach, large tasks were broken down into procedures or functions.

Objects and Classes

A structured approach to modular design
Structured Programming

• However, this approach did not always isolate or identify the data that was being acted upon. Data could be changed by many functions.

[Diagram showing data and functions]

What are the consequences here?

Object-Orientation Programming

• Modern approach

[Diagram showing objects and methods]

An object-oriented approach to modular design

Classes and Objects

• Closely related to the idea of an object, is the concept of a class.
• A class is the blueprint from which objects are generated.

[Diagram showing classes and objects]

Classes and Objects

- Break down into two main parts:

  public class SomeClass
  {
    // Inner part
    Attributes (fields or information stores)
    Methods
  }

  a small outer wrapping that simply names the class

  a much larger inner part that does all the work

Classes and Objects

  public class Product
  {
    // Attributes
    double costPrice;
    final double VAT_RATE = 17.5;
    /* Methods*/
    public void setCostPrice()
    {
      costPrice = 400;
      System.out.print("Cost price set to ");
      System.out.println(costPrice);
    }
    public void displayPriceDetails()
    {
      System.out.println("*Product Price Check *");
      System.out.print("Cost Price = ");
      System.out.println(costPrice);
      System.out.print("VAT rate = ");
      System.out.println(VAT_RATE);
    }
  }

  public class Driver
  {
    public static void main (String args[])
    {
      Product p1=new Product();
      p1.setCostPrice();
      p1.displayPriceDetails();
      Product p2=new Product();
      p2.displayPriceDetails();
    }
  }

  See any Java conventions?

Data Types in Java

Scalar, object and collection

[Diagram showing data types]

See any Java conventions?
## Scalar Data Types

**Scalar types** as they refer to a single piece of information

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
<th>Range of values</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>1 byte</td>
<td>(2^7 ) to (2^8 - 1)</td>
<td>(5)</td>
</tr>
<tr>
<td>short</td>
<td>2 bytes</td>
<td>(2^{15} ) to (2^{16} - 1)</td>
<td>(555555555555)</td>
</tr>
<tr>
<td>int</td>
<td>4 bytes</td>
<td>(2^{31} ) to (2^{32} - 1)</td>
<td>(5)</td>
</tr>
<tr>
<td>long</td>
<td>8 bytes</td>
<td>(2^{63} ) to (2^{64} - 1)</td>
<td></td>
</tr>
<tr>
<td>float</td>
<td>4 bytes</td>
<td>(\pm 1.4 \times 10^{45}) to (3.4 \times 10^{38})</td>
<td>(0.5F)</td>
</tr>
<tr>
<td>double</td>
<td>8 bytes</td>
<td>(\pm 4.9 \times 10^{324}) to (1.8 \times 10^{308})</td>
<td>(0.5D)</td>
</tr>
<tr>
<td>char</td>
<td>2 bytes</td>
<td>Unicode characters set</td>
<td>('b')</td>
</tr>
<tr>
<td>boolean</td>
<td>1 bit</td>
<td>true, false</td>
<td></td>
</tr>
</tbody>
</table>

## Scalar Type Casting

Allows for all except a boolean

- **Widening:** nothing to lose

  ```java
  float f=0.55f;
  double d=f;
  ```

- **Narrowing:** possible loss of precision

  ```java
  double d=0.55;
  float f=(float)d;
  ```

## Object Data Types

- **Predefined types from Java library**
  - For example, `String` class defined in `java.lang` package
    ```java
    String st = "Hello";
    String st = new String("Hello");
    ```

- **User defined types**
  - For example the `Product` class we have seen earlier

## Object Type Casting

- **Widening:** super-type

  ```java
  Apple a=new Apple();
  Fruit f=a;
  ```

- **Narrowing:** explicit cast to a subtype

  ```java
  Apple a1=new Apple();
  Fruit f1=a1;
  Apple a2=(Apple)f1;
  ```
Arrays

• Array: viewed as a special kind of object
double[] temperature = new double[7];

• Array can also be created and initialised
an array can also be created and initialised this way

Arrays

• Arrays are fixed sized
• Fundamental, many flexible sized Java collection
classes were built on this, e.g. java.util.ArrayList
• More linear data structures
  - Linked List
  - Stack
  - Queue
• Non-linear data structures
  - Tree
  - Graph

Arrays

• Arrays are viewed as a special kind of object
double[] temperature = new double[7];

• Array: viewed as a special kind of object
double[] temperature = new double[7];

Objects and Control Structures

Mathematical, comparison, logical operators;
selection and iteration

Java Arithmetic Operators

• Mathematical Operators
  +, -, *, /, %, --, ++

• What do we get? And Why?
  System.out.println(30/7);
  System.out.println(30.0/7);

Java Comparison Operators

• Comparison Operators
  =, !=, <, <=, >, >=

• What do we get? And Why?
  String s1=new String("hello world");
  String s2=new String("hello world");
  System.out.println(s1.equals(s2));
  System.out.println(s1.equals(s2));
Java Boolean Operators

- Boolean Operators
  - Logical OR (!)
  - Logical AND (&&)
  - Logical NOT (!)

- Does instruction b get executed? Why?
  ```java
  a=false;
  if (a&&b)
    a=true;
  if (a||b)
  ```

Java Control Structures

- Selection
  - IF-ELSE
  - SWITCH
  
  It is good practice to always include a final else and default part

- Iteration
  - FOR loop
  - WHILE loop
  - DO-WHILE loop

Java Methods

- Constructor, accessor, mutator methods
- Parameter passing, method overloading

Java Methods

- Constructors
  - a special method for creating a new object
  - always has the same name as the class, and it does not return a value.
  - the compiler creates a 'default' constructor for you if no constructor method has been written for the object.

```java
public class Flower{
  private int petalCount;
  private String name;
  //Constructor taking name as parameter
  Flower(String nameIn){
    name = nameIn;
  }
  //Two parameters
  Flower(String nameIn, int petalCountIn){
    name = nameIn;
    petalCount = petalCountIn;
  }
  //Default constructor - no args
  Flower(){
    petalCount = 0;
    name = "Undefined";
  }
  ...
}
```

Java Methods

- Parameter passing
  - Passing by values
  - Passing by references

What do we get by calling the beforeAndAfter() methods?
Java Methods - Overloading

- Method overloading
  - Methods with the same name but different set of parameters
  - Some we have seen, e.g. '/', '+', and many constructors in the Flower class
  - An example of an important OO features known as polymorphism
- Check how many max methods in the java.lang.Maths class?

Terminal Input and Output in Java

- We have already seen the command to output a message on the screen.

```java
public void displaySettings()
{
    //With tabs
    System.out.println("Mass = \t" + mass);
    System.out.println("Level = \t" + warningLevel);
    //With 2 new line escape character
    System.out.println("\n\n" + "A double quote example");
}
```

Terminal Input In Java

- From Java 1.5, the JDK provides a special class called Scanner

```java
import java.util.*;

public void promptSettings()
{
    /* Purpose: Demonstrate input techniques using Scanner */
    //Create a new scanner object to receive input from the keyboard
    Scanner sc=new Scanner(System.in);
    System.out.println("Enter a warning level (L,M,H) and press return ");
    warningLevel = sc.next().charAt(0);
    System.out.println("Enter Warp Factor (2 - 10) ");
    warpFactor = sc.nextInt();
    System.out.println("Enter Star Speed ");
    starSpeed = sc.nextDouble();
}
```

Object Encapsulation

- 'Layers of an onion' model of an object:
  - An outer layer of operation signatures...
  - ...gives access to middle layer of operations...
  - ...which can access inner core of data
  - Message from another object requests a service.
  - Operation called only via valid operation signature.
  - An object's data is hidden (encapsulated).
  - Data accessed only by object's own operations.
Object Encapsulation

- Encapsulation is the ability of OO programs to ‘close off’ or isolate classes in order to protect an objects’ attributes from outside interference.

```java
public: +
protected: #
no modifier: -
private:
```

So, we have introduced...

- Getting started
- Objects and classes
- Data types in java
- Operators and control structures
- Java methods
- Terminal output and input in Java
- Object encapsulation

Questions?