# Java CPD (I)

*Frans Coenen*
Department of Computer Science

# Content

- Session 1, 12:45-14:30 (First Java Programme, Inheritance, Arithmetic)
- Session 2, 14:45-16:45 (Input and Programme Constructs)
- Materials at:

http://www.liv.ac.uk/computer-science/continuing-professional-development/

# Logging On and Materials

- Log on using your password.
- All materials are in the directory called `H:\JavaCPD`.
- This presentation is in `H:\JavaCPD\Presentation` (suggest you open it in a window on your computer).
- You will also find all the example problems (plus some additional problems) in `H:\JavaCPD\JavaExampleProblems`.

# Background and Introduction

# High-Level Programming Languages

- In the early days of computing programming was done in byte (machine) code, however this is both extremely time consuming and error prone.
- A solution to speeding up the programming process, and reduce the associated risk of errors, is to use a high level programming language such a Java.
- High-level languages tend to use natural language constructs and/or automate certain aspects of programming (such as memory management), hence easier to use.
- However, a program written in a high-level language cannot be *run* directly, it must be either *compiled* or *interpreted*.

# Compilers v. Interpreters

- A compiler translates (converts) source code written in a high level language into a machine executable form.
- The advantage is that the executable form runs much faster than if it were interpreted.
- The disadvantage is that different machines and operating systems have different machine codes associated with them, consequently to compile a program under (say) windows would require a different compiler to that needed to do the same under (say) Apple OS.
- An interpreter steps through each line of the high-level source code and "decodes" it, the source is never translated into machine code. Different interpreters are required for different languages (and different machines). Interpretation is much slower than compilation.
- Java combines the two!

# The Java Virtual Machine

- Java “compiles” java source code into *Java Byte Code*.
- The Java Byte Code is then “interpreted” using a Java Virtual machine.
- The operation of the Java Virtual Machine, with respect to Java Byte Code, is independent of the machine it is located on.
- This means the Java Virtual Machined can be incorporated into www browsers which can then run specially constructed Java applications (Apps).
- This www capability is one of the strengths of Java.

# Object Orientation

- Object orientation is about a number of things:
  1. A way of thinking about computer solutions to problems using the concept of “objects”.
  2. The efficient generation of software solutions to problems by allowing the definitions of objects to be reused with respect to many applications.
  3. The effective writing of code by bundling elements of a solution into objects (“information hiding”).
- Java is an object oriented language and we will be using it in this way.

# Creating Java Programmes

UNIVERSITY OF
LIVERPOOL

# Creating Java Programmes

- Java Integrated Development Environments (IDEs) exist, such as NetBeans.
- We will simply be making use a of text editor.
- Windows comes with a number of these. We will be using `NotePad++`.
- Note:

1. Java source code file names (by convention) start with an upper case letter.
2. Java source code file names always take the postfix `.java`.

# Running Java Programmes

UNIVERSITY OF LIVERPOOL

# Running Java Programmes

- We will be running Java from a terminal window.
- To run a java programme we (from a terminal window) need to:
  - Compile it into Java Byte Code using the Java compiler (`javac`), and then,
  - Interpret it (`java`) by invoking the Java Virtual Machine (JVM).

# Defining a Java Class

UNIVERSITY OF
LIVERPOOL

# Defining a Java Class

- The objects we typically wish to use when programming in Java are defined using what is known as a *class*.
- A class has *members*,
- Class members can be *fields* or *methods*.
- Once we have our class definition we can use it to create objects, we say that an object is an *instance of a class*.
- We create an instance of a class using a special method called a *constructor*.

# Anatomy of a Java Class

```
Class <CLASS_NAME> {

  // Fields

  // Constructors

  // Methods

  }
```

- The class name must always be the same as the file name (without the `.java` postfix).

# Problem Example 1: Giant Letters

UNIVERSITY OF LIVERPOOL

# Giant Letters Requirements

Design and implement a Java program that writes "JAVA" vertically down the screen using giant letters made up of strings of * characters and blank spaces. (Do not use the "tab" character!)

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

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

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

# Compiling The Giant Letters Source Code

- Go to `H:\JavaCPD\JavaExampleProblems\Sequence\GiantLeters\GiantLetters` and Compile the two source files. Using:

```
javac GiantJava.java
javac GiantJavaApp.java
```

- Or:

```
javac *.java
```

- In your `GiantLetters` directory you will now find the relevant `.class` files.

**(Make sure you are in the right directory!)**

# Running The Giant Letters Application

- In the same terminal window type:

```
java GiantJavaApp
```

# Giant Java Source Code

- Open the `GiantJava.java` source file (not the `.class` file) in a text editor editor such as `notepad++`.

# Giant Java Comments (1)

- Comments are important from a Software Engineering perspective (readability leads to understandability which leads to maintainability).
- In Java single line comments are indicated using a `//`, multi line comments using a `/* ... */` or a `/** ... */`.
- Note that the class has:
  - No fields.
  - A default constructor (which in this case we do not have to specifically specify).
  - Three methods: `giantLetterA()`, `giantLetterJ()`, `giantLetterV()`.

# Giant java Comments (2)

- Anatomy of a Java method:

```
<Modifiers> <ReturnType> <Name>
                ( <ArgumentList ) {
  <Statements>
  }
```

- In the case of our `GiantJava` source code:
  - The return type for each method is `void` (return nothing).
  - The argument list for each method has been omitted (there are no arguments).

# Giant Java Comments (3)

- Modifiers can be:
  - `public` visible from outside of the class.
  - `private` visible only from within the class.
  - `protected` visible from within the class and by sub classes of the current class (more on this later).
- In our case all the methods are all public (can be called from anywhere).

# Giant Letters Application

- Having created the source code for our `GiantJava` class we also need an *application class* that allows us to use it (we need to be able to create an instance of this class).
- An application class has a special method in it called `main` from where the JVM starts "interpreting".

```
public static void main(String[] args) {
   <Statements>
   }
```

# Giant Letters Application

```
public static void main(String[] args) {
   <Statements>
   }
```

- A closer look at the `main` method:
  - It is public (can be called from anywhere)
  - It is static (to use the method we do not need to create an instance of the class in which it is defined).
  - It returns no value (it has nowhere to return it to).
  - It has an argument (we will simply have to accept that this is what is required).

# Giant Java Application Comments (1)

- We create an instance of the class GiantJava as follows:

```
GiantJava newGJ = new GiantJava();
```

- A default constructor is one that has no arguments and is created automatically if we do not specify our own constructor.

# Giant Java Application Comments (2)

- We call methods in the `GiantJava` class by linking them to an instance of the class:

```
newGJ.giantLetterJ();
```

# Editing The Giant Letters Application

- Try editing the `GiantJavaApp.java` application class source (you will have to recompile).
- In the main method try reordering the method calls.
- In the main method try creating a second instance of the class `GiantJava`.

# Problem Example 2: Landscape Gardening Quote Item

UNIVERSITY OF LIVERPOOL

# Landscape Gardening Quote Item Requirements*

Customers provide a landscape gardening company with a plan detailing lawns, concrete patios and water features. Unit material costs and installation times are as shown in the table. Create a Java class that can be used to store unit material costs and installation times.

| Work to be done | Unit cost of materials | Unit time to install |
|---|---|---|
| Laying a lawn | £15.50 per $m^2$ | 20 mins per $m^2$ |
| Laying a concrete patio | £20.99 per $m^2$ | 20 mins per $m^2$ |
| Installing a water feature (e.g. a fountain) | £150.00 each | 60 mins each |

* (Taken form AQA HCSE Specimen Controlled Assessment v1.0)

# Compiling The Landscape Gardening Quote Item Application

- Go to the directory `H:\JavaCPD\JavaExampleProblems\Sequence\QuoteItem` and compile the source files there:

```
javac *.java
```

- Run the code by typing:

```
Java QuoteItemApp
```

# Quote Item Source Code

- Load `H:\JavaCPD\JavaExampleProblems\Sequence\QuoteItem\QuoteItem.java` into the `NotePad++` editor.

# Quote Item Comments (1)

- The QuoteItem class has three fields: (i) `itemName` (ii) `unitMaterialCost`, and (iii) `unitInstallationTime`.
- The first is of type `String`, the other two are of type `double` (another popular Java type is the type `int`).

# Quote Item Comments (2)

- There is one constructor.
- Recall that constructors are special methods that are used to create instances of classes.
- Constructors have the same name as their class and are of necessity public.
- They do not have a return type.
- The `QuoteItem` constructor has three arguments which are assigned to the three fields.

# Quote Item Comments (3)

- The + operator in the `toString` method is a concatenation operator.
- Note that `String`, because it is written with a uppercase letter, is class.
- Because the string class is used very frequently Java provides lots of “short cuts” to facilitate its usage (for example we did not have to use a constructor).

# Quote Item Application Source Code (Ver. 1)

- Load `QuoteItemApp.java` into your text editor.

- Note:

1. The main method.
2. The way that we create an instance of the `QuoteItem` class using the constructor.
3. The `System.out.println` output method.

# Quote Item Application Comments

- `System.out.println(newQI)` outputs its content to the screen. In this case its content is an object, `newQI`, hence the associated `toString` method (another special method) is called.
- `System` is a "built-in" class that comes with Java which contains an instance `out` which in turn is use to call the `println` method.
- There is also a `print` method (no new line at the end of the output).

# Editing The Quote Item Application (1)

- Try changing the values in the application class, recompiling and running again. Example:

```
QuoteItem newQI =
      new QuoteItem(”patio",10.0,5.0);
```

# Editing The Quote Item Application (2)

- Try creating two `QuoteItem` objects. For Example:

```
QuoteItem newQI_1 =
     new QuoteItem(”decking",10.0,5.0);
QuoteItem newQI_2 =
     new QuoteItem(”pond",10.0,5.0);
System.out.println(newQI_1);
System.out.println(newQI_2);
```

# Java Inheritance

UNIVERSITY OF
LIVERPOOL

# Java Class Hierarchies and Inheritance

- Java class hierarchies are a mechanism whereby "*child*" classes can inherit from "*parent*" classes.
- It allows for code reuse by child classes and the extension of parent classes.
- A child class can have only one parent class, but a parent can have many child classes.
- We indicate that a class is a child of another class using the keyword `extends`.

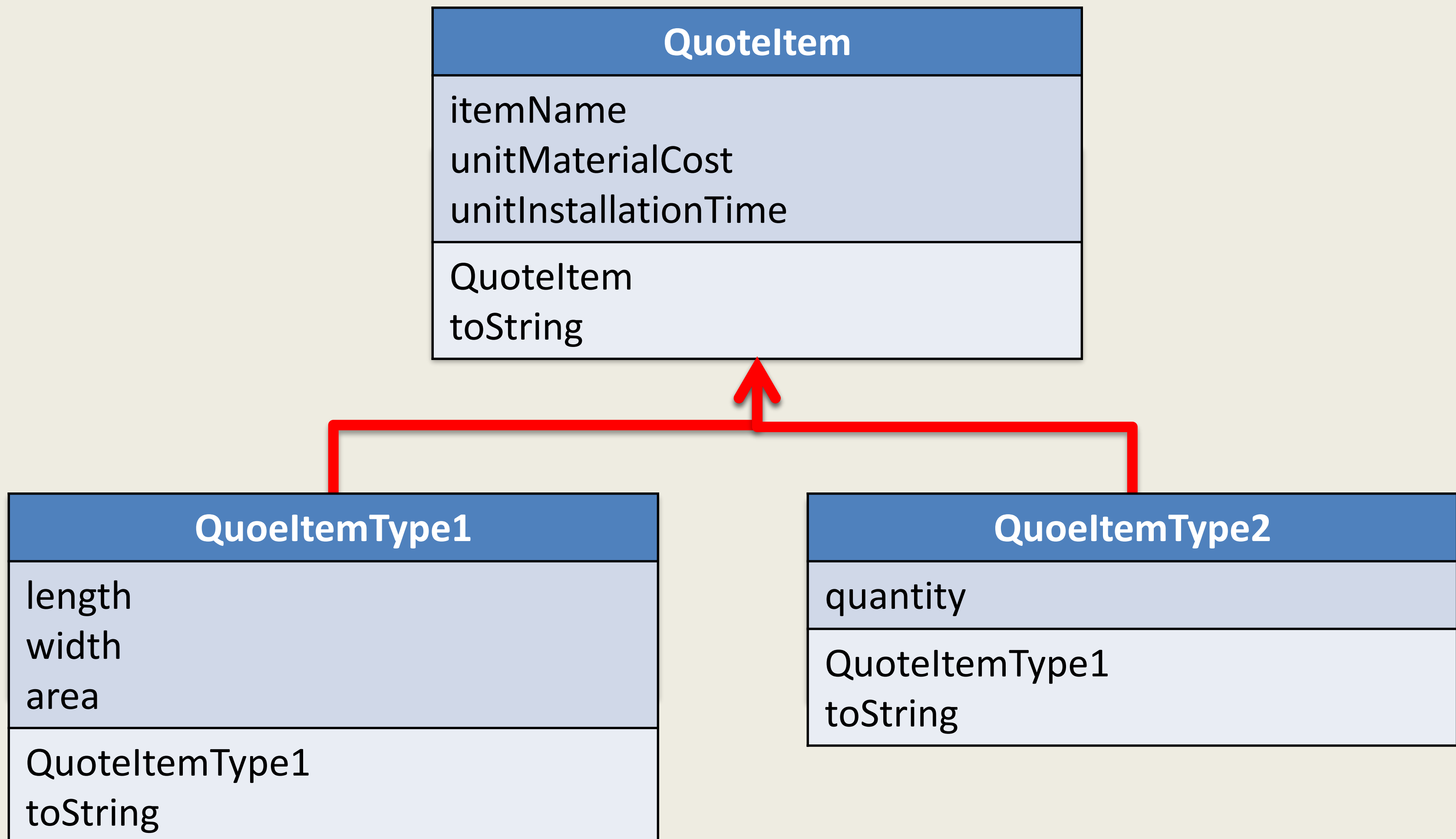
# Problem Example 3: Landscape Gardening Quote With Inheritance 1

# Landscape Gardening Quote With Inheritance 1 Requirements*

Customers who engage our landscape gardening company can specify two types of landscape gardening item: (i) Type 1, items specified by length and width (lawns and patios) and (ii) Type 2, items specified by quantity (water features). Both have unit material costs and installation times associated with them. Create a Java class hierarchy that can be used to store details concerning Type 1 and Type 2 landscape gardening items.

* (Taken form AQA HCSE Specimen Controlled Assessment v1.0)

# Landscape Gardening Class Hierarchy (Ver. 1)



# Compiling The Quote Item Application

- Go to the directory `H:\JavaCPD\JavaExampleProblems\Inheritance\LandscapeGardQuoteInheritance1` and compile the four source files in a terminal window type:

```
javac *.java
```

- In a terminal window type:

```
java QuoteItemApp
```

# Quote Item Source Code (Ver. 2)

- Load the `QuoteItem.java` source file into the text editor.

- Note:

1. That the fields that we want the child classes to inherit now have the modifier `protected` (if we had kept the `private` modifier the child classes would not be able to “see” them).
2. Otherwise the class is identical

# Quote Item Type 1 and Type2 Source Code

- Load `QuoteItemType1.java` and `QuoteItemType2.java` into the text editor.

- Note:

1. We indicate that the class `QuoteItemType1` is a child of the class `QuoteItem`:

```
public class QuoteItemType1 extends QuoteItem
```

2. The method `super()` calls the parent constructor.

# Quote Application Source Code (Ver. 2)

- Load `QuoteItemApp.java` into the text editor.

- Note:

1. We can still create instances of the class `QuoteItem` in the same way as before, but in this example we create instances of the classes `QuoteItemType1` and `QuoteItemType2` (using the appropriate constructors).

# Editing The Quote Item Application Source Code

- Try adding another instance of the class `QuoteItemType1` the specifies a patio measuring `3x4` with unit cost of £`20.99` and unit installation time of `20.0` minutes per $m^2$.

# Java Arithmetic

UNIVERSITY OF LIVERPOOL

# Simple Arithmetic in java

- Java features all the usual arithmetic functions.
- Care must be taken with respect to “mixed mode” arithmetic.
- For example integer division means that `5/2=2`!

# Problem Example 4: Landscape Gardening Quote With Inheritance 2

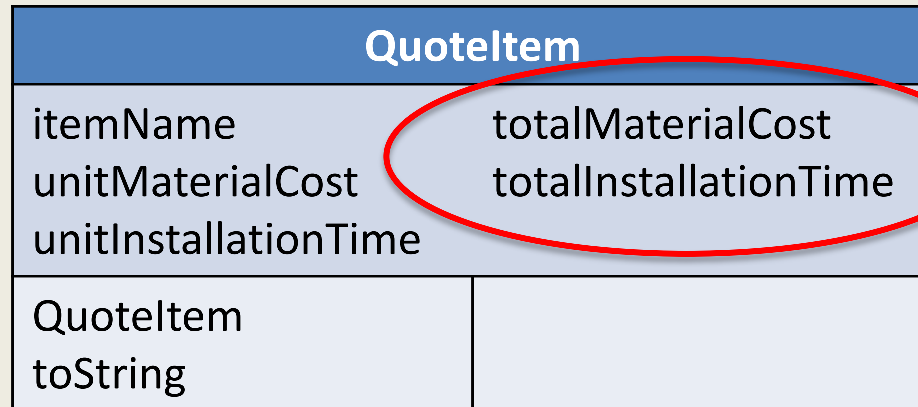
UNIVERSITY OF
LIVERPOOL

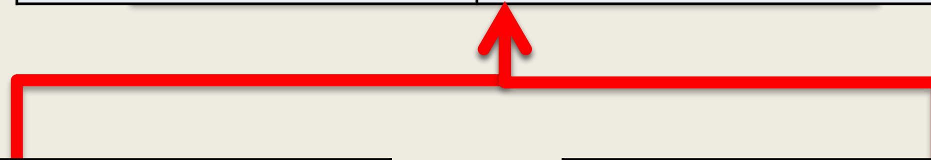
# Landscape Gardening Quote With Inheritance 2 Requirements*

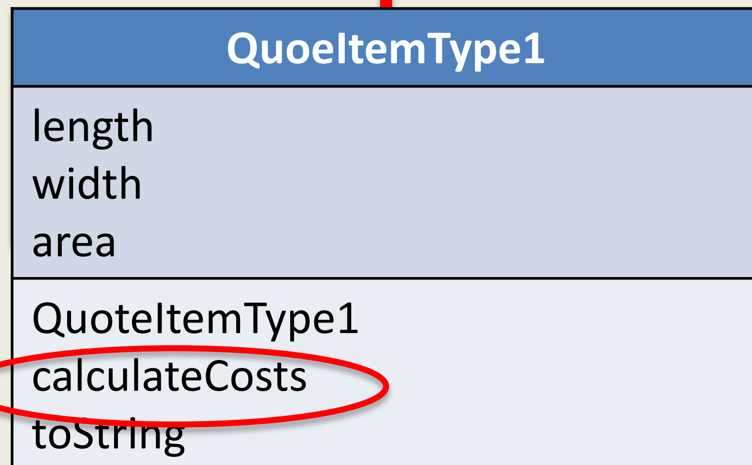
Our landscape gardening company, when generating quotes for customers, needs to determine the total material cost and installation time for each item. Create a collection of Java classes that will calculate individual total material costs and installation times per item given a specific quote.

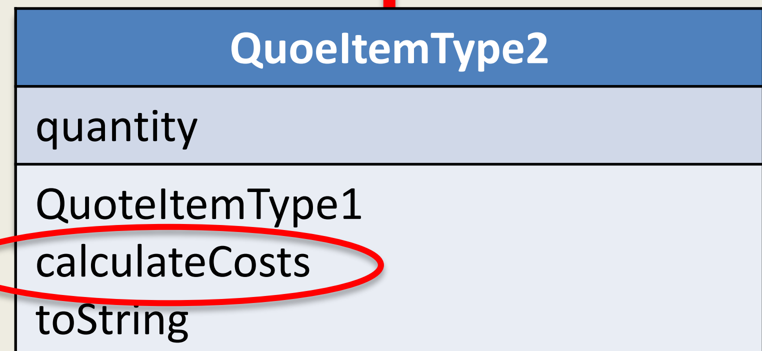
* (Taken form AQA HCSE Specimen Controlled Assessment v1.0)

# Landscape Gardening Class Hierarchy (Ver. 2)





# Compiling The Quote Item Application

- Go to the directory `H:\JavaCPD\JavaEaxampleProblems\Inheritance\LandscapeGardQuoteInheritance2` and compile the source files in the usual manner:

```
javac *.java
```

- Run the code by typing:

```
java QuoteItemApp
```

# Quote Item Source Code (Ver. 3)

- Load `QuoteItem.java` into the text editor.

- Note:

1. We have added two more fields `totalMaterialCost` and `totalInstallationTime` (both are protected fields, so can be inherited).

# Quote Item Type 1 and 2 Source Code (Vers. 2)

- Load `QuoteItemType1.java` and `QuoteItemType1.java` into the text editor.

- Note:

1. We have added a private method (can only be called from inside the class) to calculate the totals (called from the constructor).
2. We have also revised the `toString` method and included calls to the `twoDecPlaces` method in a `Utility` static class.

# Quote Application Source Code (Ver. 3)

- Load `QuoteItemApp.java` into the text editor.
- Note that the application class is entirely unchanged.

# Editing the Quote Item Application Source Code

- Again try adding another instance of the class `QuoteItemType1` that specifies a patio measuring `3x4` with unit cost of £`20.99` and unit installation time of `20.0` minutes per $m^2$.

- Remember to include an additional output statement.

# Tea Time?

UNIVERSITY OF LIVERPOOL