CM10227/ CM50258: Lecture 5

Dr Rachid Hourizi and Dr. Michael Wright

November 14, 2016

Resources

- More help with this course
  - Moodle
  - E-mail - programming1@lists.bath.ac.uk
- Online C and Java IDE
  - `https://www.codechef.com/ide`
  - Remember to select Java from the drop down menu.
- Books
  - C by Dissection
  - Java by Dissection (Free pdf online)

- The places that you can get additional support if you are finding the pace of the course a little fast now include
  - The A labs
  - The B (catch up) lab
  - The Drop in Sessions
- please note that we have moved a couple of the labs
- please check the details on Moodle and let us know if you now cannot get to a lab that you wold otherwise have attended.

- If you struggling with the exercises, pace of the course and/or coding in general
- Please come and see Rachid or Michael

- If, on the other hand, you are finding the pace a little slow
- You can still sign up for the Advanced Programming Labs

Last Week

- Complex Collections (In C)
- Abstract Data Types (In C)

This Week

- Java
- First Java Program
- Objects

Java

- This week marks a major turning point in the course
  - We will move from using C
  - To using a second programing language: Java
- (We will talk more about why later in the lecture)

- That change means that you will have to learn
  - new ways to write the instructions that we want the computer to execute and
  - a new way to think about the programs that we write whilst using those instructions: Object Oriented Programming

- As you make these changes, however, it is worth noting that you are not starting from scratch
  - When writing small Java programs, the programs that you write can appear reasonably similar to those that you wrote in C
  - The Java version of HelloWorld, for example, bears some similarity to the C version

```
public class HelloWorld {

    public static void main(String[] args) {

        //print HelloWorld to the terminal
        System.out.println("Hello, World");
    }

}
```

- Similarities with the C version of HelloWorld include
  - The presence of a main method (starting point for the flow of execution) - Vocabulary note: method not function in this case
  - A pre-defined instruction to print "Hello World" to the screen
  - Some kind of input to the main method (String[] args)
  - A familiar return type (void)
  - the use of a semicolon (;) to mark the end of an instruction

- Differences from the C version of HelloWorld include
  - The keyword class
  - The keyword public (formally, an "Access modifier" - we will return to this later)
  - The keyword static (again, an element to which we will return later)
  - Slightly different syntax for comments - //
  - Different syntax for the print command - System.out.println()

  We will return to each of these differences later

- To some extent, you will also see similarities between
  - the ways in which you compile and run a Java program and
  - the ways in which you compile and run a C program
- These similarities can be misleading, however
  - On closer examination, the steps taken to prepare a Java program for execution
  - Are different from those taken when preparing a C program

- Java programs are both compiled and interpreted.
- Instead of translating Java programs into a machine language, the Java compiler generates Java byte code for its Virtual Machine
  - Byte code is easy (and fast) to interpret, like machine language,
  - but it is also portable, like a high-level language.
- Thus, it is possible to compile a Java program on one machine, transfer the byte code to another machine over a network, and then interpret the byte code on the other machine.
- This ability is one of the advantages of Java over many other high-level languages.

- We can use compilation/interpetation of the HelloWorld program on an earlier slide as an example:
- Before you start, the filename in which you save your main method should match the name of the class with the extension .java. In this case, HelloWorld.java
- Note: Java is case sensitive, just like C.

- To run the code:
- we need to compile it: `javac HelloPrinter.java`
- This will generate a file `HelloPrinter.class`, containing the virtual machine byte code
- We can now run the code by typing the name of the class file
- NB When running a Java file, you omit the .class suffix:

```
$javac HelloWorld.java
$java HelloWorld
Hello, World
```

- If you write a program involving multiple classes,
- you would normally save each class in a different file
- Note: Each file must have a name of the form [classname].java
- where [classname] is the name of the class contained in the file

A Shortcut:

- If you have multiple java files, you can compile tham all with one command.
- i.e. You can use `javac *.java` to compile all .java files in the current directory in one go.
- To run the program, you then need to use the interpreter command (`java`) on the class that contains the `main` method.
- We will discuss multi-class programs shortly

```
$javac *.java
$java HelloWorld
Hello, World
```

- Just as you did with C,
- We would strongly suggest that you adopt an iterative approach to developping Java programs i.e.
  - start with a program that works
  - (such as the Java version of HelloWorld introduced earlier)
  - and extend/debug it in small increments
  - until you have a program that does what you want

- In order to go beyond our HelloWorld program, however
  - You will need to understand that Java is an Object Oriented (OO) language
  - (Sometimes described as an Object Oriented Programming (OOP) language)

- This begs two obvious questions:
  - What does it mean to be "Object Oriented" when writing code?
  - And, in a programming context, what is an Object anyway?

- Object Oriented Programming: A type of programming in which programmers define not only the data type of a data structure, but also the types of operations (methods) that can be applied to the data structure. In this way, the data structure becomes an object that includes both data and functions.
- http://www.webopedia.com/TERM/O/object_oriented_programming_OOP.html

- In other words, an Object Oriented Programming approach maintains this interest in associating data with the operations that can be performed on it:

How does Java "support" an OO approach to programming?

- C allowed us to use
  - **data structures** and
  - **functions** which allow us to manipulate those structures
- Java forces us to use
  - **combinations of data structures and functions** (Objects)
    - ★ some of which are pre-defined for us
    - ★ and some of which we can define ourselves

- For example, if we write a program which stores information about and allows us to print out a Train Ticket
  - i.e. code that allows us to store an initial cost for the ticket
  - ... update the cost of that ticket
  - and print out the current cost

- Java forces us to
  - **define** a combination of data and operations once (a Class)
  - and then **create** one or more examples (instances) of that class (Objects)

- More specifically, each time we want to use a new combination of data and operations, Java forces us to
  - start by defining a Class which combines that data and those operations
  - before creating one or more examples of that class (one or more Objects)
- You can think of this process as
  - defining a template (the Class)
  - and using it to create one or more Objects using that template

```
public class Ticket
{
    int cost = 10;

    //constructor
    public Ticket()
    {
        System.out.println("New Ticket Created!");
    }
    //mutator
    public void changePrice(int newPrice)
    {
        cost=newPrice;
    }
    //accessor
    public int getPrice()
    {
        return cost;
    }
}
```

```
public class TicketMachine
{
    public static void main(String[] args)
    {
        \\create new Ticket
        Ticket tm = new Ticket();

        \\change the price of that ticket
        tm.changePrice(20);

        \\retrieve and print the price of that ticket
        System.out.println("" + tm.getPrice());
    }
}
```

```
New Ticket Created!
20
```

Exercise

It is useful at this stage to consider some examples of classes (templates) and objects (concrete examples created using those templates):

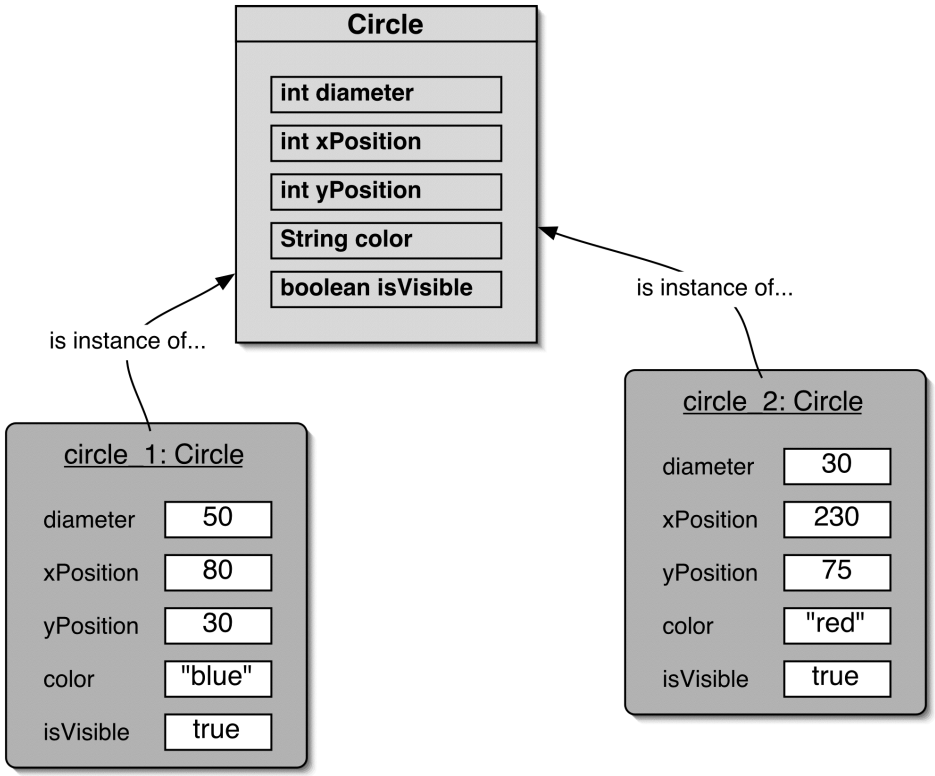
TelephoneNumber	lord-of-the-rings
BankAccount	Diary
harry-potter-and-the-Philosopher-Stone	myDiary
01225-38-5053	michaelWright
Book	Lecturer
rachidHourizi	myAccount

Attributes and Fields

- Note from the Ticket example:
  - Just as C Structures contained values stored in member variables
  - an Object has attributes: i.e. values stored in fields. (The data you encapsulate)
  - the class defines what fields an Object has, but each Object stores its own set of values. The set of values stored in a single Object is called the state of the object.

State

- We could also imagine the definition of a circle class (i.e. a template)
  - Which stores information about the diameter, (x and y) position of the centre, colour and visibility of that circle
- And use it to create two different circles



Methods and Parameters

- We also saw previously that objects/classes also have operations which can be invoked. They are called methods
- If, with our Circle example, we wanted to be able to
  - draw examples of the circle class (Circle Objects) on screen and then
  - move those circles a specified distance to the right
- we could include
  - a drawCircle() method in our definition of the Circle class and
  - a moveRight() method in our definition of the Circle class

```
void moveRight(int distance){
        ...some code...
}
```

- The first piece of code used to implement a method e.g.
  - void moveRight(int distance)
- provides a short description of what the method does
- that short description is called the method's signature
- The collection of methods in a class is referred to as the interface of that class

- Methods are called or invoked
- At the point of being called, they may need to recieve information as input if they are to perform the desired operation
  - for example, our moveRight() method may need to be given a distance in order to move a Circle Object as intended
- As in C, that information can be passed in the form of one or more parameters
- in the moveRight(int distance) example, int distance is an example a parameter

Data Types

- Both Fields and Parameters have types. A type defines what kinds of values a parameter can take.
- In Java (as in C) you have to specify the type.
- e.g. we know that void moveRight(int distance) takes an integer as input because the distance parameter is defined as being of type int.

- In Java, everything has a type.
- Examples of types: int, String, Circle, ...
- Java is staticly typed language
- i.e. once you have given a field or parameter a type, that type cannot change
- Note: Each time you create a new class, you have defined a new type

Source Code

- Each class has source code associated with it that defines its details (fields and methods).
- The code defining each class determines the structure and the behavior of each of its instances (i.e. the structure and behaviour of each Object created using the class template).
- This source code is compiled and interpreted by Java.

Return Values

- Methods may return a result via a return value.
- Example: `String getName()` This method returns a String.
- Example: `void changeName()` Void indicates that this method does not return anything

Developing Java Programs

- To learn to develop Java programs, one needs to learn how to write class definitions, including fields and methods, and how to put these classes together
- We will look at these issues in more detail during the remainder of this course

Looking at our ticket example in more detail ....

```
public class Ticket
{
    int cost = 10;

    //constructor
    public Ticket()
    {
        System.out.println("New Ticket Created!");
    }
    //mutator
    public void changePrice(int newPrice)
    {
        cost=newPrice;
    }
    //accessor
    public int getPrice()
    {
        return cost;
    }
}
```

```
public class TicketMachine
{
    public static void main(String[] args)
    {
        \\create new Ticket
        Ticket tm = new Ticket();

        \\change the price of that ticket
        tm.changePrice(20);

        \\retrieve and print the price of that ticket
        System.out.println("" + tm.getPrice());
    }
}
```

```
New Ticket Created!
20
```

Basic class structure

```
public class TicketMachine
{
    Inner part of
    the class omitted.
}
```

Basic class structure

```
public class ClassName
{
    Fields
    Constructors
    Mutator Methods
    Accessor Methods
}
```

Fields

Recap:

- Fields store values for an object.
- They are also known as instance variables.
- Fields define the state of an object.
- Fields have an associated type.

```
public class TicketMachine
{
    private int price;
    private int balance;
    private int total;

    Constructor and methods omitted.
}
```

Constructors

- Constructors create and initialize an object.
- Then assign the necessary memory to the created object
- They have the same name as their class.
- They store initial values into the fields.
- They often receive external parameter values for this.

```
//constructor
public Ticket()
{
    System.out.println("New␣Ticket␣Created!");
}
```

Parameters

- Parameter names inside a constructor or method are referred to as Formal Parameters
- Parameter values provided from the outside are referred to as Actual Parameters.
- In the constructor `TicketMachine(int ticketCost)`
- ticketCost is a formal parameter.
- When the constructor is called, using `TicketMachine(500)`,
- 500 is an actual parameter.

Scope and Lifetime

- The scope of a variable/parameter defines the section of the code from which it can be accessed.
- For instance variables (fields) this is the entire class.
- For parameters, this is the constructor or method that declares it.
- Trick: find the enclosing $\{\}$, this is the scope.
- The lifetime of a variable/parameter describes how long the variable continues to exist before it is destroyed.

Assignment

- Similar to C:
- Values are stored into fields (and other variables) via assignment statements:
  - `variable = expression;`
  - e.g. `price = ticketCost;`
- Both sides of the assignment should have the same type, e.g. int, double, String, TicketMachine, ...
- A variable stores a single value, so (as in C) any previous value is lost after an assignment.

Accessor Methods I

- All methods implement object behaviour.
- And all methods have a structure consisting of a header and a body.
- The header defines the methods signature e.g.

```
public int getPrice()    {

        ... method body ...

}
```

- The body encloses the methods statements e.g.

```
... header{

        return price;

}
```

- We can, however, be more precise about the kind of behaviour implemented by particular methods
- Accessors, for example are methods which provide information about an object.
- i.e. they allow users, other programs or other parts of your program to gain information about an object's state

```
public int getPrice()
{
    return price;
}
```

Mutator Methods

- Mutators have a similar method structure: header and body.
- Used to mutate (i.e., change) an objects state.
- Achieved through changing the value of one or more fields.
  - Typically receive parameters.
  - Typically contain assignment statements.

Mutator methods

```
//mutator
public void changePrice(int newPrice)
{
    cost=newPrice;
}
```

Comments/Documentation

- As with C, Comments make Java source code easier to read for humans. No effect on the functionality.
- Three sorts:
  - `// comment`: single-line comments
  - `/* comments */`: multiple-lines more detail
  - `/** */`: similar to previous, but used when documentation software is used.

Data Types

- As we have seen in the code examples above, Java (like C) provides us with common data types
- e.g. integers:
  - int distance =7;
- you won't be surprised to learn that Java also provides us with chars
  - char myChar ='c';
- booleans
  - boolean a = false;
- and floats
  - flaot f = 4.0;

Data Types

- Java also provides us with ways to implement Strings and Arrays
- We will return to these data types
- ..but for now be a little careful with them
- ..Java and C treat them in rather different ways

- Each time that we define a class, however, we define a new data type
- Can be used as parameter, field and return types
- The internal is hidden from the user
  - No direct access to fields (unless special reason)
  - Access to state via accessor and mutator methods
- User does not need to know how the class is implemented to use/instantiate it
- The usage of a class is defined by its methods

Making choices

```
if(perform some test) {
    Do the statements here if the test gave a true
        result
}
else {
    Do the statements here if the test gave a false
        result
}
```

Coding Convention

- If statement
  - Always use { , even if there is only one statement
  - In case there is an else statement, start on a new line and use {
- Indentation
  - Always indent your code, even if your text editor does not do it automatically
- Document your code, the sooner the better.

Boolean Tests

- == : equality
- > : greater than
- < : less than
- <= : less or equal than
- >= : greater or equal than
- ! = : not equal

Local variables

- Fields are one sort of variable.
  - They store values through the life of an object.
  - They are accessible throughout the class.
  - A bit like global variables in C
- Methods can include shorter-lived variables.
  - They exist only as long as the method is being executed.
  - They are only accessible from within the method.
  - Like function variables in C

Local variables

```
public int refundBalance()
{
    int amountToRefund;
    amountToRefund = balance;
    balance = 0;
    return amountToRefund;
}
```

Review

- Class bodies contain fields, constructors and methods.
- Fields store values that determine an objects state.
- Constructors initialize objects.
- Methods implement the behaviour of objects.
- Mutators (mutator methods) change the state of a object
- Accessors (accessor methods) provide information about the state of an object

Review

- Fields, parameters and local variables are all variables.
- Fields persist for the lifetime of an object.
- Parameters are used to receive values into a constructor or method.
- Local variables are used for short-lived temporary storage.
- Objects can make decisions via conditional (if) statements.
- A true or false test allows one of two alternative courses of actions to be taken.