CM 10227: Lecture 10

Dr Rachid Hourizi and Dr. Michael Wright

December 6, 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
  - Java by Dissection (Free pdf online)
  - The Java Tutorial (https://docs.oracle.com/javase/tutorial/)

Resources

- The places that you can get additional support if you are finding the pace of the course a little fast now include
  - A labs (Continued from week 1)
  - B labs
  - ... Wednesday 11:15-13:05 EB0.7
  - ... Fridays 17:15 to 19:15 in CB 5.13)
  - The Drop in Session
  - ... booked 20 min appointments
  - ... Friday 11.15-13.05 1E 3.9
  - PAL sessions (Mondays 14:15 to 15:05 1E 3.9)

Last week

- Errors
- Exceptions
- Style : Writing Better Code

Last week

- Java API Libraries

- As we have seen throughout the course, a large part of learning to program is learning how to re-use code e,g.
  - Classes
  - Methods
  - Inheritance
  - Abstract Classes
  - Interfaces
- As well as reusing our own code, we have also discussed opportunities to use those provided by the developers of the Java language

- Java comes delivered with a library of useful classes, which are referred to as the Java API
- There really is no point in reinventing the wheel...

- Examples of Java library class include

- Random - a library class used to generate random numbers
- Math - a library class for performing basic numeric operations
- ArrayList - a library class for storing and manipulating a resizable array
- StringBuilder - a library class for manipulating a mutable sequence of characters

- The Java class library contains,
  - Thousands of classes
  - Tens of thousands of methods
  - Many useful classes that make life much easier
- These classes are organised in packages that follow a directory structure
- A competent Java programmer must be able to work with the libraries.
- (Just as competent programmers using other languages must be able to work with the libraries provided by those languages)

- We have looked at the specification of individual clases
- Comprehensive documentation of the Java libraries is however available in HTML format; Readable in a web browser
- Class API: Application Programmers Interface:
- Interface description for all library classes
- `http://docs.oracle.com/javase/7/docs/api/`

- As you go forward with your Java programming, you should...

- ... know some important classes by name;
- ... know how to find out about other classes.

- Remember! We only need to know the interface, not the implementation.

- You will need to know about a library class because you will use objects of that class to provide common functionality within your programs
- e.g. you may use objects of the Random class to generate random numbers

- In this case, we use the class library in just the same way that we might use classes that we have created ourselves:
- We create a new instance of the class (a new Object) using the new keyword

```
Random randomGenerator = new Random();
```

- And can then use that Object's methods to provide the functionality that we need within our own program

```
int index2 = randomGenerator.nextInt(100);
```

- Other useful constructor and methods in the Random class include

```
Random(long seed)
// seed is the initial value of the internal state of
   the pseudorandom number generator

public int nextInt()

public long nextLong()

public boolean nextBoolean()
```

- Another useful class is ArrayList

- Resizable-array
- Each ArrayList instance has a capacity and as elements are added its capacity grows automatically

```
ArrayList<String> myArrayList
    = new ArrayList<String>();

myArrayList.add("Hello");
myArrayList.add("World");
```

- Other useful methods in the ArrayList class include

```
public boolean contains(Object o)

public E get(int index)

public int size()

public Object[] toArray()
```

- Another really useful method in ArrayList is iterator()...
- ... which returns an Iterator for the ArrayList

```
public Iterator<E> iterator()
```

- Iterator is an interface...
- ... which defines a common set of methods to iterate over a collection

- Why is this useful?
- Lets think about generalisation and encapsulation of functionality
- If we have a collection of elements we want to iterate over (i.e. loop through)...
- ... do we really want to care about how find the size of the collection or get its next element?

```
for(int i=0; i<myArrayList.size(); i++){
    System.out.println(myArrayList.get(i));
}
```

- Although most collections implement the AbstractList interface so have the size and get methods...
- ... if these names were to change
- ... or we want to ensure we use the fastest possible method of iteration over a collection
- ... an Iterator object is preferable

```
ArrayList<String> myArrayList
    = new ArrayList<String>();

myArrayList.add("Hello");
myArrayList.add("World");

Iterator i = myArrayList.iterator();

while(i.hasNext()){
    System.out.print(i.next());
}
```

```
$ Hello World
```

```
ArrayList<String> myArrayList
    = new ArrayList<String>();

myArrayList.add("Hello");
myArrayList.add("World");

Iterator<String> i = myArrayList.iterator();
for( ; i.hasNext(); ){
    System.out.print(i.next());
}
```

```
$ Hello World
```

Aside: Generics

- Generics allows you to specify types (classes and interfaces) to be parameters when defining classes, interfaces and methods

- Advantages are...
- ... stronger type checks at compile time
- ... elimination of casts
- ... enabling programmers to implement generic algorithms

Aside: Generics

- Stronger type checks at compile time

- Java compiler applies strong type checking to generic code
- Issues errors if the code violates type safety
- Fixing compile-time errors is easier than fixing runtime errors

Aside: Generics

- Elimination of casts

```
List list = new ArrayList();
list.add("hello");
String s = (String) list.get(0);
```

```
List<String> list = new ArrayList<String>();
list.add("hello");
String s = list.get(0); // no cast
```

Aside: Generics

- Enabling programmers to implement generic algorithms

```
public class Box<T> {
    // T stands for "Type"
    private T t;

    public void set(T t) { this.t = t; }
    public T get() { return t; }
}
```

```
Box<Integer> box1 = new Box<Integer>();
Box<String> box1 = new Box<String>();
```

Aside: Generics

- The most commonly used type parameter names are...

```
E - Element (used extensively by the Java Collections
    Framework)
K - Key
N - Number
T - Type
V - Value
```

Back to Java API

- Interface vs. Implementation
- For each class provided by the Java API, the Java developers provide documentation
- The documentation includes:
  - the name of the class;
  - a general description of the class;
  - a list of constructors and methods
  - return values and parameters for constructors and methods
  - a description of the purpose of each constructor and method
- The interface of the class

- The documentation does not include
- Private fields
  - (most fields are private)
- Private methods
- The bodies (implementation code) for each method
- Detailed implementation of the class

- Most classes from the library must be imported using an import statement
- (Those from java.lang, however, do not)
- They can then be used like classes from the current project.

- Classes are organised in packages.
- Single classes may be imported:
- Whole packages can be imported:

```
import java.util.ArrayList;

import java.util.*;
```

Another Example of the Java API

- StringBuilder
- Lets consider string concatenation

```
String name = "Michael";
String lastName = "Wright";

System.out.println(name + "␣" + lastName);
```

- Remember that String is immutable
- So when we concatenate a new String is created
- For this example it is not a problem
- i.e. the "performance" of the program is not likely to suffer

```
String name = "Michael";
String lastName = "Wright";

System.out.println(name + "␣" + lastName);
```

- But what about this...

```
String s = "";
for(int i=0; i<50000; i++){
    s = s + "␣" + i;
}
```

- For large amounts of concatenation, especially in loops...
- ... we might consider using a StringBuilder

- A mutable sequence of characters
- Includes method to append, insert and reverse
- toString() is also a useful method

- StringBuilder example

```
StringBuilder s = new StringBuilder();

for(int i=0; i<50000; i++){
    s.append("␣");
    s.append(i);
}
```

- Different ways to concatenate a String

```
// +
String fullName = "Michael" + "␣" + "Wright";

// concat method in String
String fullName = "Michael".concat("␣Wright");

// StringBuilder
StringBuilder fullName = new StringBuilder();
fullName.append("Michael");
fullName.append("␣");
fullName.append("Wright");
```

Class Variable and Methods

- Class libraries can also define constants
- A constant is an identifier (a name) with an associated value which cannot be altered by the program during normal execution – the value is constant.
- This is contrasted with a variable, which is an identifier (a name) associated with a value that can be changed during normal execution – the value is variable.

- Math.PI (the number pi) is and example of a constant
- We only need to store pi once
- And can then use it over and over again in our programs
- If we look at the documentation for Math.PI in the Java API
- (i.e. the documentation for the field PI within the Math class)
- We can see it is a static final i.e. it is constant and stored at the class level
- `http://docs.oracle.com/javase/7/docs/api/java/lang/Math.html`

- Static methods

- Note that it is also possible to create class methods
- i.e. methods that are declared as static
- These methods can be called directly from the class rather than from an Object that instantiates that class

- The Math class, for example, provides a method that raises the first argument (a) to the power of the second argument (b):

```
public static double pow(double a, double b);
```

```
public class MyClass{

   public void printVolume(double radius){
      double volume = Math.pow(Math.PI*radius, 2);
      System.out.println("volume␣is:␣" +volume);
   }

}
```

- The Double class provides a method that parses a String into an double:

```
public static double parseDouble(String s);
```

```
public class MyClass{

  public void printVolume(String radiusString){
    double radius = Double.parseDouble(radiusString);

    double volume = Math.pow(Math.PI*radius, 2);
    System.out.println("volume␣is:␣" +volume);
  }

}
```

- Note that we can create our own
  - constants
  - class variables
  - and class constants
- We can also create our own class methods

```
final int myConstant = 1;
static int myClassVariable = 2;
static final int myClassConstant = 3;

public static void myClassMethod(){
    ...
}
```

- **Instance methods ...**
- ... are associated with an object and
- ... use the instance variables of that object.
- This is the default.

```
public class MyClass{

    private int myVariable;

    public int getMyVariable(){
        return myVariable;
    }

}
```

- **Static methods ...**
- ... use no instance variables of any object of the class they are defined in
- ... (if you define a method to be static, you will be given a rude message by the compiler if you try to access any instance variables)
- ... you can access static variables, but except for constants, this is unusual.
- ... typically take all they data from parameters and compute something from those parameters, with no reference to variables.
- ... this is typical of methods which do some kind of generic calculation

```
public class MyUtilClass{

    public static double mean(int[] p) {
        int sum = 0;  // sum of all the elements
        for (int i=0; i<p.length; i++) {
            sum += p[i];
        }
        return ((double)sum) / p.length;
    }

}
```

- Other examples of static methods in the Java API

```
java.awt.Color

// static variables
Color.BLACK
Color.RED

// static methods
public static Color getColor(String nm)

public static int HSBtoRGB(float hue,
           float saturation,
           float brightness)
```

Summary

- Java comes delivered with a library of useful classes, which are referred to as the Java API
- There really is no point in reinventing the wheel...

- As you go forward with your Java programming, you should...

- ... know some important classes by name
- ... know how to find out about other classes