

2AIT515 - OBJECT ORIENTED SOFTWARE DEVELOPMENT

Lecture 1: Object Oriented Programming and Java Fundamentals

Dr Dimitris C. Dracopoulos

email: d.dracopoulos@westminster.ac.uk

1 Why Java?

The following are some characteristics of Java which make it particularly attractive to use:

- “Simple”. Although Java it was written so that it will be simple to use, no programming language is really simple. Some new features of Java 1.5 have made it more “complex” than it used to be.
- Secure. The security mechanisms applied by the Java Virtual Machine, make safety one of the greatest strengths of Java.
- Platform Independent (“write once, run everywhere”).
- Many libraries (in the form of packages). New libraries are introduced constantly.
- Designed for distributed systems.

2 Java vs C++

Java and C++ have many commonalities. However they are different programming languages. Some of the most important differences are:

- C++ supports operator overloading. Operators like +, -, *, etc. can be redefined to work with user defined types. For example, + can be overloaded for a class `MyString`, so that the result of `a+b` (where `a` and `b` are objects of class `MyString`), is the concatenation of the characters in `a` and `b`.
- Java has automatic memory management (garbage collector). There is no need to deallocate types allocated in the heap (in C++ this has to be done explicitly using the `delete` operator).

- All Java objects are allocated dynamically (in the heap).
- Java does not support multiple inheritance.

Some people claim that Java programs are slower than the C++ equivalent, This is not true, especially in recent versions of Java.

3 The Simplest Java Program

```
public class Hello {
    public static void main(String[] args) {
        // Display a message in the screen
        System.out.println("Hello World\n");
    }
}
```

4 Compiling a Java Program

The above program is contained in a file called `Hello.java`.

```
javac Hello.java
```

creates a file `Hello.class` which contains the *bytecode* for the Java program.

- Bytecode is not native code, i.e. it cannot be run directly in the machine in which the source code was compiled.
- Bytecode can be run by the *Java Virtual Machine* (in short JVM), in which all Java programs run.
- This is why Java programs are platform independent. A Java program which was compiled in a specific machine (producing the bytecode `.class` file), can be run in the JVM of another machine, without the need of recompilation.

5 Running a Java program

```
java Hello
```

The above `java` command will:

1. Start the Java Virtual Machine.
2. Load the bytecode from the file `Hello.class` into the JVM.
3. The Java Virtual Machine will execute the loaded bytecode.

The compilation and run process of a Java program can be seen in Figure 1.

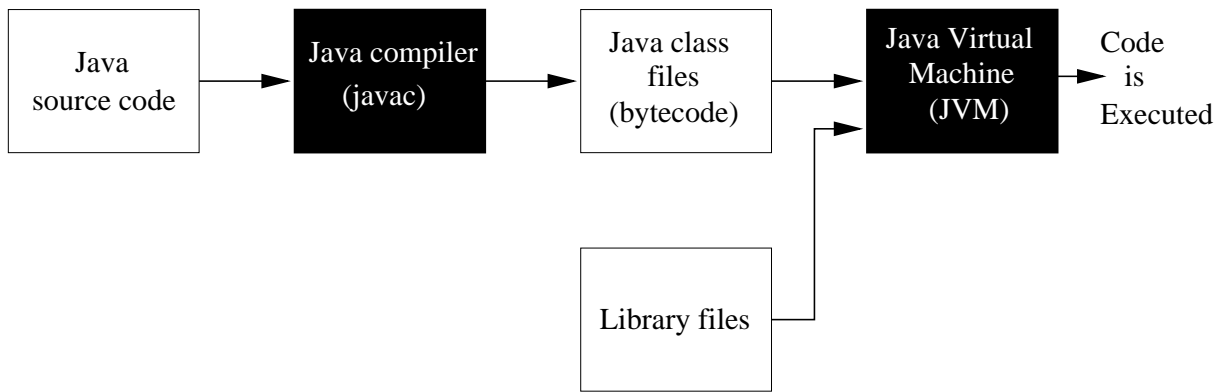


Figure 1: The steps necessary to compile and run a Java program.

6 Major Characteristics of Object Oriented Programming

Four of the major characteristics of the Object Oriented Paradigm are:

1. Abstraction
2. Polymorphism
3. Inheritance
4. Encapsulation

Peter Van der Linden suggests the mnemonic *APIE* for remembering these (Figure 2).

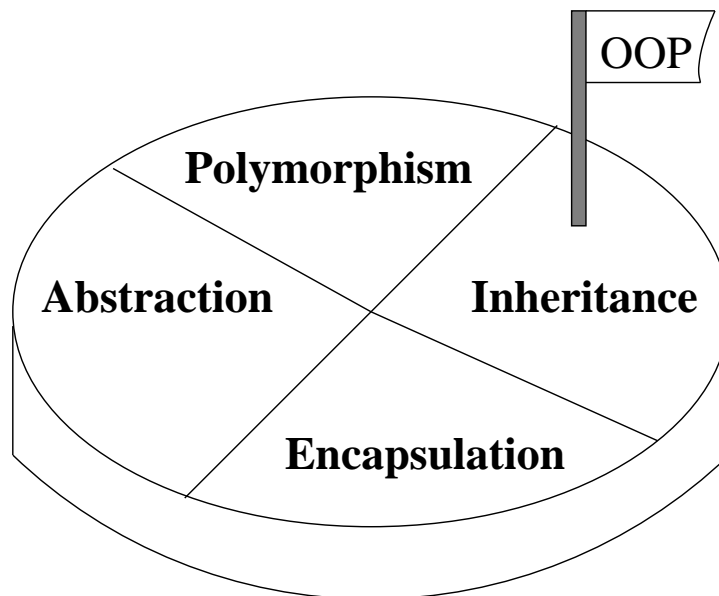


Figure 2: The major characteristics of the object oriented paradigm.

6.1 Abstraction

In the process of solving a problem:

- A particular representation of the solution must be chosen.
- The representation (*object*) of such a component contains the important characteristics (data) of the component, and the allowed operations on them which are necessary for the particular situation.

Example:

To model a car in a particular problem, only the size of the car and its colour are important. Therefore, to represent a car in this specific problem, a class holding information about the size of the car and its colour is needed only.

Such a class is created and it is an abstraction of a real car, because all the other characteristics of a real-world car are not needed and therefore not modelled.

6.2 Inheritance

Inheritance organises classes in hierarchies.

- It is based on how similar the functionality of classes is.

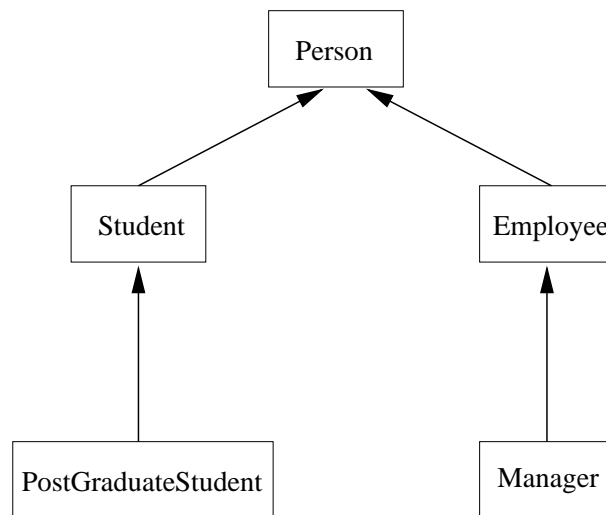


Figure 3: Inheritance of classes in UML notation. Classes **Student** and **Employee** are derived from class **Person**. Class **PostGraduateStudent** derives from **Student**, while **Manager** derives (inherits) from class **Employee**.

- A class **A** which inherits from a class **B** is called the child of **B**. **B** is called the parent class of **A**. In other words, **A** is derived from **B**.

- A child class inherits all the fields and methods of its parent class.
- A child class should always be consistent with the “is a” relationship with the parent class. For example, a `Student` “is a” `Person`.

7 Classes and Objects (Structure of a Java Program)

- An object oriented program has multiple instances (objects) of various classes.
- The objects interact with each other by sending signals to each other (i.e. calling methods on other objects).

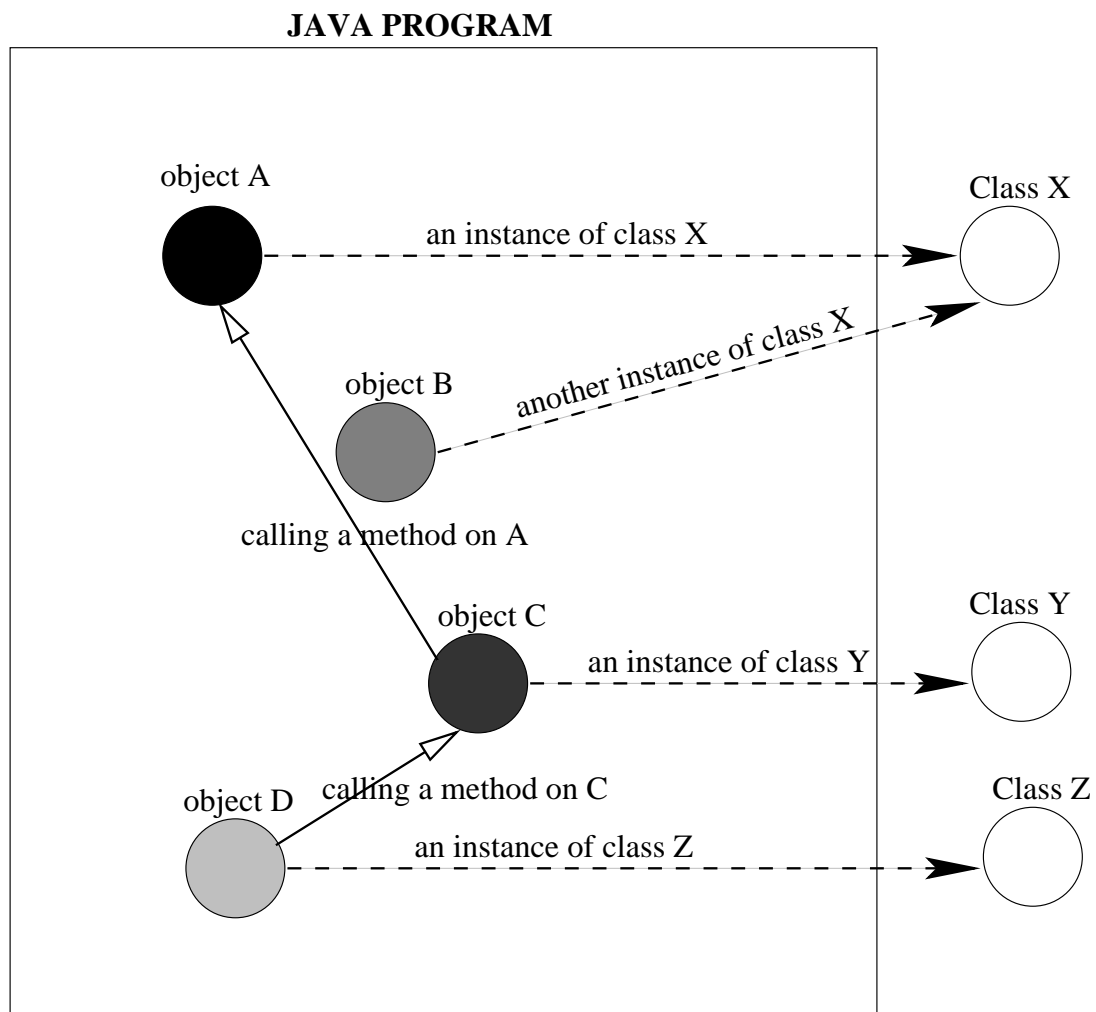


Figure 4: A typical example of the composition and basic operation of an object oriented program.

8 Primitive types and Objects

A Java program has primitive types and user defined types (classes). Instances of classes are called objects.

In the following code, "Hello World" is an object, i.e. an instance of class `String` (a class which is defined in the standard Java library). It is assigned to variable `greeting`. Similarly, 15 is a primitive type (`int`), assigned to variable `i`.

```
String greeting = "Hello World"; // object assigned to a variable reference
int i = 15; // a primitive type (int) assigned to a variable
```

- Objects are created using the `new` operator. However, `String` objects are a special case, as they can also be created by simply enclosing a number of characters in double quotes, as it is seen in the code segment above.

```
String message = new String("First week of lectures");
```

9 Calling Methods on Objects

A class defines methods which can be called on objects of the class.

For example, method `length` is defined in the library class `String`, and it can be called for any object of the class. The method returns the number of characters in a `String` object:

```
String greeting = "Hello World";
int n = greeting.length();
System.out.println("n=" + n);
```

```
String message = new String("First week of lectures");
n = message.length();
System.out.println("n=" + n);
```

The above segment of code displays:

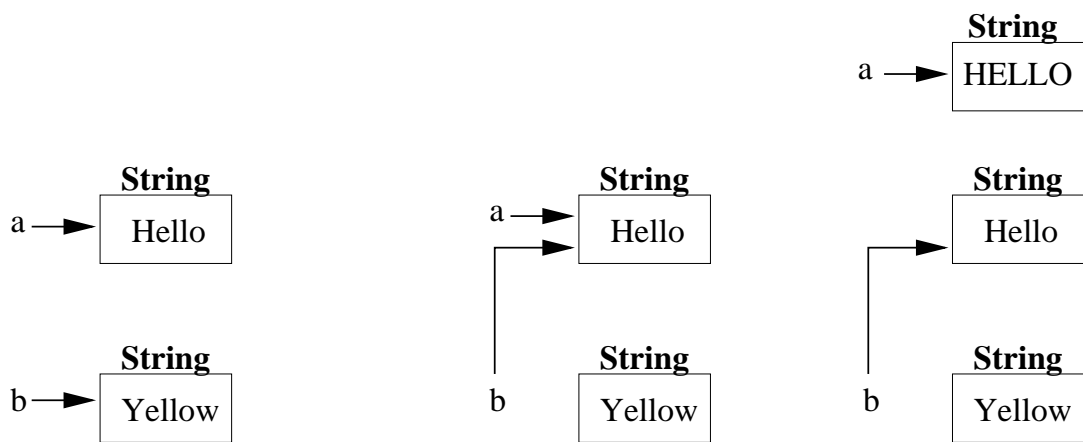
```
n=11
n=22
```

10 Assigning an object reference variable to another variable

- Variables which are assigned objects, actually store the address of the object.
- This means that assigning such a variable to another, will not duplicate the object but will create an additional reference to the initial object. This is seen in Figure 5.
- Note that the assignment operator is the single equals '=' sign. This is different than the double equals sign '==' used to test for equality. *It is a common programming mistake, to use the single equals sign when testing for equality.*

Note that calling method `toUpperCase()` on the object stored by `a` (Figure 5), will create a new `String` object. The new object's address is stored in `a` (because of the assignment `a = a.toUpperCase()`), or otherwise `a` will point to the newly created object. Thus, object references are very similar to C++ pointers, but no dereferencing is needed using the `*` operator.

Java object references are very similar to C++ reference variables but no ampersand is needed when they are declared.



1) After: `String a = "Hello";`
`String b = "Yellow";`

2) After: `b = a;`

3) After:
`a = a.toUpperCase();`

Figure 5: What happens during assignment of one object reference to another. Objects are not duplicated.