5COSC001W - OBJECT ORIENTED PROGRAMMING

Lecture 1: Object Oriented Programming and Some Java Fundamentals

Dr Dimitris C. Dracopoulos

email: d.dracopoulos@westminster.ac.uk

1 Programming Paradigms

The following is a classification of programming languages:

- Procedural programming
- Functional programming
- Logic Programming
- Object-oriented programming

2 Why Object Oriented Programming (OOP)?

Some reasons to study OOP are:

- Relatively Easy
- Powerful
- Many languages with similar syntax (Java, C++, C#)
- Popular (Job Market)

3 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.

4 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 1).

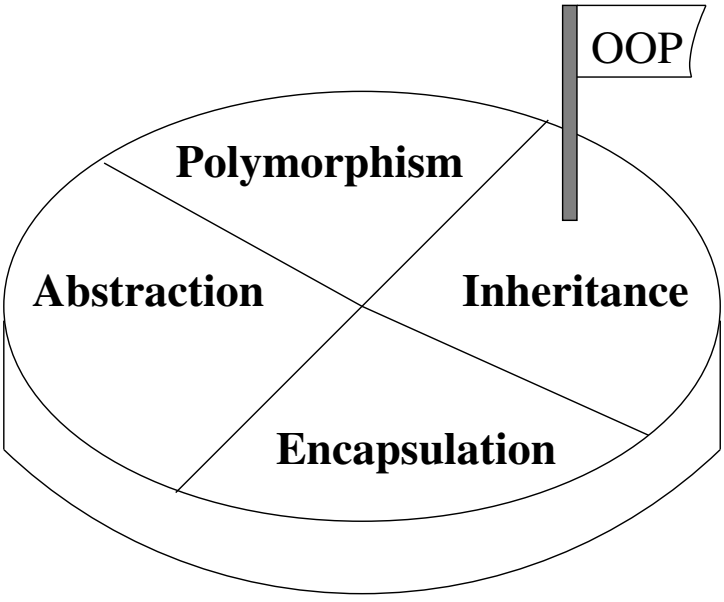


Figure 1: The major characteristics of the object oriented paradigm.

4.1 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.

4.2 Inheritance

Inheritance organises classes in hierarchies.

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

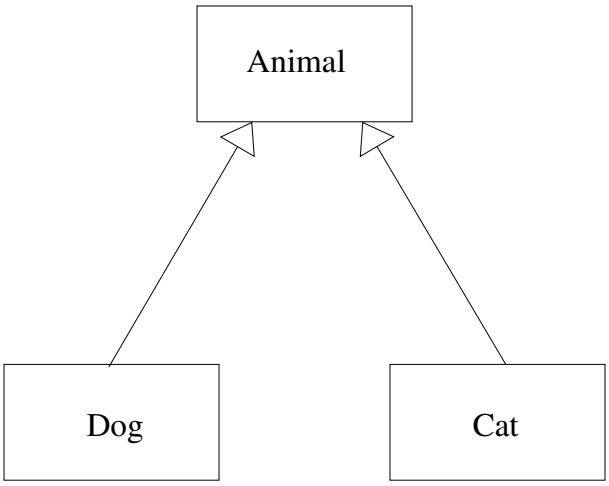


Figure 2: Inheritance of classes in UML notation. Classes `Student` and `Employee` are derived from class `Person`. Class `PostGraduateStudent` derives from `Student`, while `Manager` derives (inherits) from class `Employee`.

- 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`.

5 Classes and Objects (Structure of an Object Oriented 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).

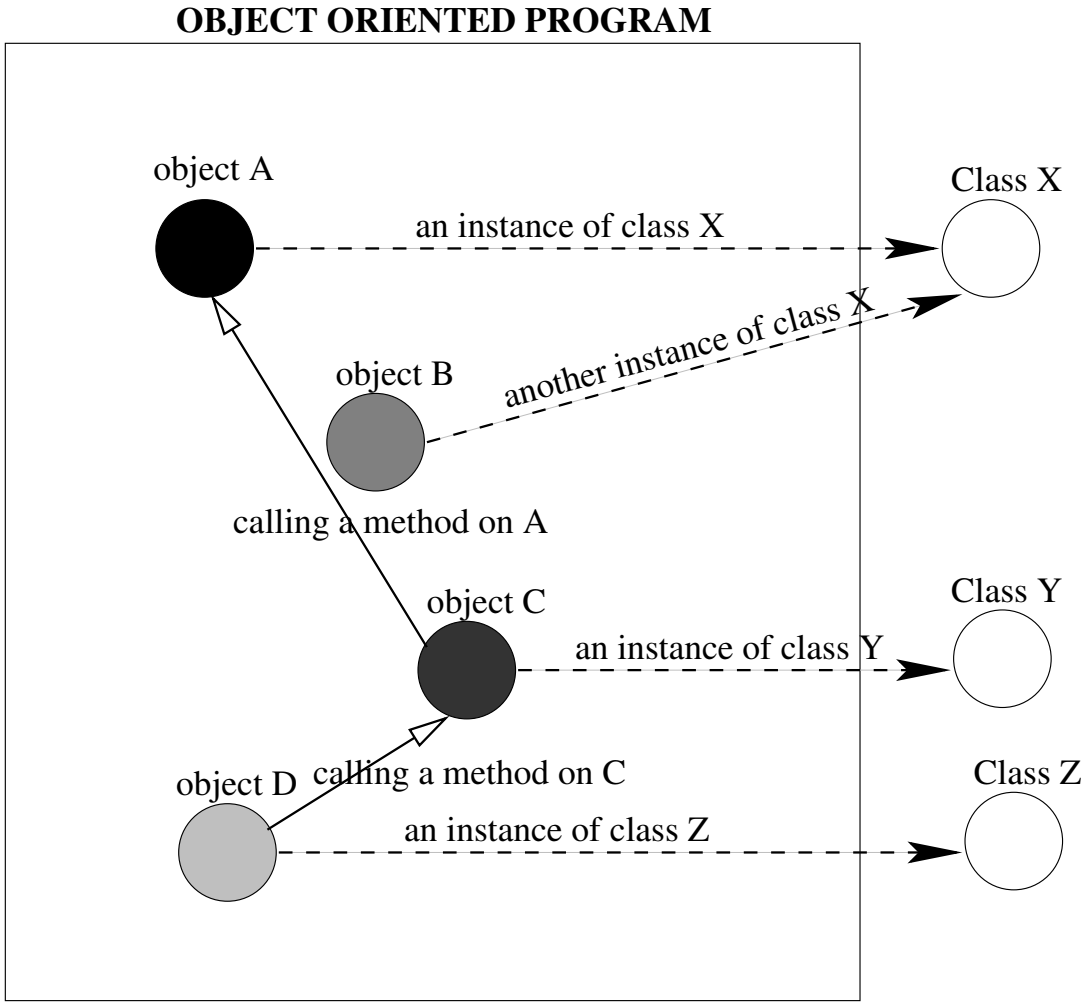


Figure 3: A typical example of the composition and basic operation of an object oriented program.

6 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");
  ```

7 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:

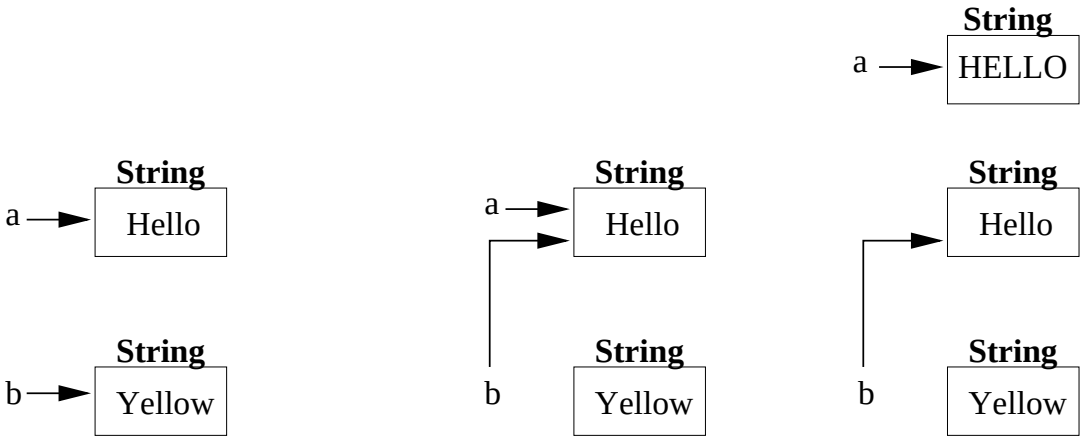
```
n=11
n=22
```

8 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 4.
- 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 4), 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 4: What happens during assignment of one object reference to another. Objects are not duplicated.