Object Oriented Programming

OOP Vocabulary Review

Class

The class definition provides a *template* or *blueprint, which* describes the *data (instance variables)* contained within, and the *behavior (methods)* common to all objects of a certain kinds.

Data

The data is contained in variables defined within the class (often called instance *variables, data members*, *properties* or *attributes*). The instance variables may be primitives, such as int, float etc or they may be references to other objects. These can only be accessible after instantiating an object.

Behavior

The behavior is controlled by methods defined within the class (often called member *methods*).

Object

Object is the implementation of class. It is a software bundle of variables and methods. Objects of same class will have its own and equal number of instance variables. Member methods are invoked using objects.

In OOP terminology, objects are also called *instance*.

Classes vs. Objects

Remember, class is just a blueprint or an abstract concept or a definition. For example the map of a building can be considered as a class. Objects on the other hand are the implementation of class. The building itself can be considered as an object.

When you create a class you are describing how objects of that class look and how they will behave. You don't actually get anything until you create an object of that class and at that point data storage is created and methods become available.

Constructor

Constructors are used to create objects from classes.

A constructor has the same name as the class with no return type. It is called when an object of the class is going to be created. The main purpose of writing constructor is initialization of instance variables

Unlike C++, Java does not provide any initialization list with constructor. You can initialize your variables inside the constructor. Example given at the end.

If you don't provide constructor for class, the JVM will provide a default (zero argument) constructor and initialize the instance variables to default values as shown in the tables and object references to null.

Default values for primitive types

Primitives	**Default Valuea**
Boolean	false
Char	'\uoooo' (null)
Byte	(byte)0
Short	(short)0
Int	0
Long	0
Float	0.0F
Double	0.0d

Remember the same rule is not applied to local variables (variables declared inside any method), they must be initialized manually before they are used or otherwise compiler will indicate an error.

```
public void someMethod( ) {
      int x;

       x++;          //   x is a local variable and must be explicitly initialized
                    //   so this statement produces compile time error
}
```

Classes can have multiple constructors distinguished by different arguments (overloading) to meet requirements. The constructor that accepts parameters are often called parameterized constructor.

Note: JVM will provide a default constructor only if a class doesn't define any constructor.

Destructors

Destructors are used to destroy objects and free memory. Since memory management in java, is not the job of programmer therefore there are no destructors in Java. You will not find any ~Employee() type method in java.

To know more about memory management, see handout on Memory Management

Defining a Class

```
class  Point  {

       private int xCord;
       private int yCord;

       public Point (……)  {……}

       public void display (……)
       {
          ……….
        }

} //end of class
```

inastance variables and symbolic constants

constructor – how to create and initialize objects

methods – how to manipulate those objects (may or may not include its own “driver”, i.e., main( ))

NAMING CONVENTIONS

While writing code in java, please use the following naming conventions. Remember it is not mandatory to follow these conventions but they are widely used across the world and its mandatory for you to use these conventions inside your course

- MyClass
- myMethod
- myVariable
- MY_CONSTANT

Comparison with C++

The complete comparison between java and C++ made and discussed in detail in handout “Similarities and differences between C++ and Java”.

Some important points to consider when defining a class in java as you probably noticed from the above given skeleton are

- There are no global variables or functions. Everything resides inside a class. Remember we wrote our main method inside a class. (For example, in HelloWorldApp program)

- Specify access modifiers (public, private or protected) for each member method or data members at every line.

- There is no semicolon (;) at the end of class

– All methods (functions) are written inline. There are no separate header and implementation files.

Access Modifiers

An access level determines whether other classes can use/access a particular instance variable or call a particular method. The Java programming language supports four access specifiers for instance variables and methods: `private`, `protected`, `public`, and, if left unspecified, package. The following table shows the access permitted by each specifier.

Access Levels

Specifier	**Class**	**Package**	**Subclass**	**World**
Private	Y	N	N	N
Package	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

- **Private**
  Only accessible within this class

- **Package**
  Default access if no access modifier is provided. A method of variable with package access is accessible to all classes of the same package. We will discuss packages in more detail in the later handouts. For now you can think of a package as a directory.

- **Protected**
  Generally, used for inheritance. Accessible only to the class itself and to its subclasses

- **public**
  Accessible anywhere by anyone

Getters / Setters

As the access modifier used for instance variable is mostly private or protected, It is usual for a class to provide getter (accessors) & setters (mutators) methods for these instance variable with public access modifier.

If x & y are the instance variables than for setters, word "set" is used before the instance variable name like setX, setY. For getters, word "get" is used before the instance variable name getX, getY.

this keyword

Suppose you are inside a method and you need a reference to the current object. **this** reference would solve the problem. The **this** keyword that can only be used inside a method produces the reference to the object the method has been called for.

One common use of this reference is resolving name conflicts between instance variable and the argument name passed to a method. Now, to refer instance variable, use **this.instanceVariableName.** Example of using this has been illustrated in the following program

Task – Defining a Student class

The following example will illustrate how to write a class. We want to write a “Student” class that

- should be able to store the following characteristics of student
  - Roll No
  - Name

- Provide default, parameterized and copy constructors
  - Provide standard getters/setters for instance variables
    - Make sure, roll no has never assigned a negative value i.e. ensuring the correct state of the object
    - Provide print method capable of printing student object on console

Student Class Code

```
// File Student.java

/* Demonstrates the most basic features of a class.  A student
   is defined by their name and rollNo.  There are standard
   getter/setters for name and rollNo.

   NOTE A well documented class should include an introductory
   comment like this. Don't get into all the details – just
   introduce the landscape.
*/

public class Student {

     private String name;
     private int rollNo;

    // Standard Setters
     public void setName (String name) {
            this.name = name;
      }

      // Note the masking of class level variable rollNo
      public void setRollNo (int rollNo) {
            if (rollNo > 0) {
                 this.rollNo = rollNo;
            }else {
                 this.rollNo = 100;
            }
      }

    // Standard Getters
     public String getName ( ) {
            return name;
      }

     public int getRollNo ( ) {
            return rollNo;
      }
        // Default Constructor
        public Student() {
             name = “not set”;
             rollNo = 100;
        }

        // parameterized Constructor for a new student
        public Student(String name, int rollNo) {
             setName(name);      //call to setter of name
             setRollNo(rollNo);  //call to setter of rollNo
        }

        // Copy Constructor for a new student
        public Student(Student s) {
            name = s.name;
            rollNo = s.rollNo;
        }

       // method used to display method on console

       public void print () {
           System.out.print("Student name: " +name);
           System.out.println(", roll no: " +rollNo);
       }

} // end of class
```

Using a Class

In Java objects of a class are always created on heap using the new operator (except String class) followed by constructor. The new invokes the specified constructor.

Point p = new Point(); //call to default constructor

Remember that, there is no → operator in java, so instance variables and methods are accessed using dot (.) operator.

p.display( ); // call to display method of Point class

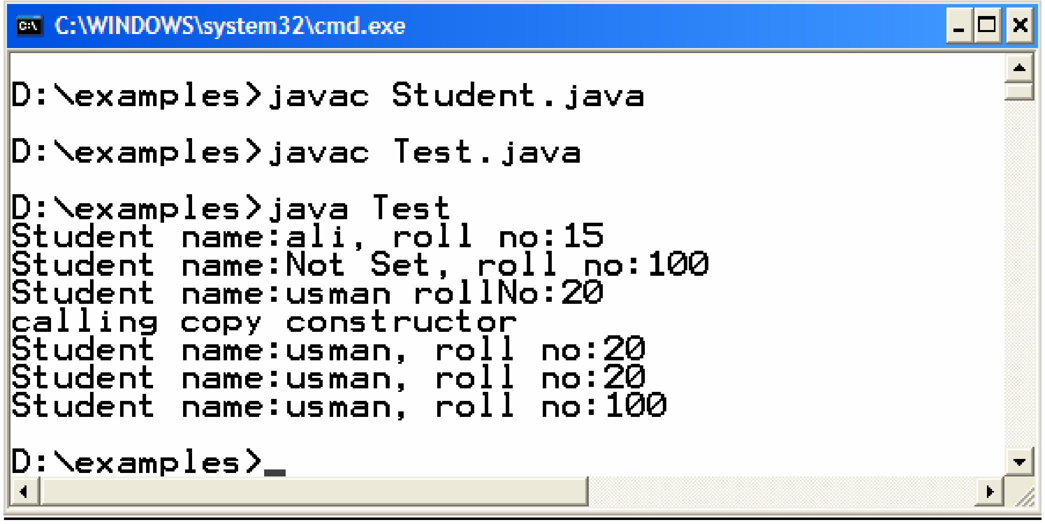
Note: Objects are always passed by reference and primitives are always passed by value in java.

Student Client Code

```
// File Test.java

/* This class create Student class objects and demonstrates
   how to call various methods on objects
*/

public class Test{

   public static void main (String args[]){

    // Make two students
    Student s1 = new Student("ali", 15);
    Student s2 = new Student(); //call to default costructor

    s1.print();
    s2.print();

    s2.setName("usman");
    s2.setRollNo(20);

    System.out.print("Student name:" + s2.getName());
    System.out.println(" rollNo:" + s2.getRollNo());

    System.out.println("calling copy constructor");
    Student s3 = new Student(s2); //call to copy constructor

    s2.print();
    s3.print();

    s3.setRollNo(-10); //Roll No would be set to 100

    s3.print();
    /*NOTE: public vs. private
      A statement like "b.rollNo = 10;" will not compile in a
      client of the Student class when rollNo is declared
      protected or private
    */

  } //end of main
} //end of class
```

Compile & Execute

Compile both classes using **javac** commad. Run Test class using **java** command.



```
D:\examples>javac Student.java

D:\examples>javac Test.java

D:\examples>java Test
Student name:ali, roll no:15
Student name:Not Set, roll no:100
Student name:usman rollNo:20
calling copy constructor
Student name:usman, roll no:20
Student name:usman, roll no:20
Student name:usman, roll no:100

D:\examples>_
```

Points to Remember Revisited

- Recompile the class after making any changes
- Save your program before compilation
- Only run a class using java command that contains the main method because program executions always starts form main

Exercise

Write a class in Java that represents a Bank Account. A bank account contains the following information

- Name of Account (String)
- ID of Account (String)
- Balance (double)
- Type of Account , Saving or Current (String)

Provide appropriate constructors and getter / setter methods for each of the attributes. Also define a test class. Create objects of the Account class inside the test class and use it

Reference:

- Sun java tutorial: http://java.sun.com/docs/books/tutorial/java
- Thinking in java by Bruce Eckle
- Beginning Java2 by Ivor Hortan