Object Oriented Programming

Bob Singh

Introduction to OOP

- Object-Oriented Programming is an approach that provides a way of modularizing programs by creating partitioned memory area for both data and functions that can be used as template for creating such modules on demand.
- Some of the languages that support OOP features are:
  - Smalltalk
  - C++
  - Ada
  - Java

Features of OOP languages

- OOP allows to decompose a problem into a number of entities called Objects and build data and methods around these entities.
- OOP ties the data closely to the methods that operate on it and protects it from unintentional modification by other methods.
- The data of an object can be accessed only by the methods of that object. However, methods of one object can access the methods of other objects.

Features of OOP languages

- In OOP, new data and methods can be easily added whenever necessary.
- OOP follows bottom-up approach in program design.

Basic Concepts of OOP

- Objects
- Classes

Objects

- Basic runtime entities in an object-oriented system. May represent a person, a place, a bank account, a table of data or any item that the program may handle.
- Any programming problem is analyzed in terms of objects and the nature of communication between them.
- Objects are chosen such that they match closely with the real-world objects.

Objects

- An object takes up space in the memory and has an associated address.
- When a program is executed, the objects interact by sending messages to one another.
  - For example, in a banking program, 'customer' and 'account' are two classes. The customer object may send a request to the account object requesting for the balance.
    - **Joe** is an object of **customer** class
    - **Joe's saving account** is an object of **account** class
  - Each object contains data and code to manipulate that data.
  - The basic approach of object-oriented programming is that objects can interact with each other without knowing the details of each other's data or code.

Classes

- A class may be thought of as a data type and an object as a variable of that data type.
- A class is a description of making an object, which contains fields and methods. It provides a sort of template for an object.
- Classes are user-defined data types and behave like the built-in types such as **`int`**.
- Thus, a class is a collection of objects of similar type. For example, mango, apple and orange are members of the class **`fruit`**.
-

Benefits of OOP

- In OOP, through inheritance redundant code can be eliminated to extend the use of existing classes.
- Data hiding helps the programmer to build secure programs.
- Easy to partition the work in a project, based on objects.
- Object-oriented systems can be easily upgraded from small to large systems.
- Multiple objects can co-exist without any interference.

Applications of OOP

- Real-time systems
- Simulation and modeling
- Object-oriented databases
- Hypertext, hypermedia
- Decision support and office automation systems
- CAD systems

Instance Variables and Methods

- A class contains two sections:
  - Variable declaration
  - Method declaration
- These variables and methods are called as instance variables and instance methods.
- Every time a class is instantiated, a new copy of each of them is created. They are accessed by using dot operators.

Static Variables and Methods

- A member that is common to all the objects and accessed without using any particular objects is called as a static member.
- For example,

```
/* static member variable */
    static int count;
 /* static member method */
    Static int max(int x, int y);
```

Static Variables and Methods

- Static variables and static methods are also called as class variables and class methods respectively.
- Static variables are used when we want to have a variable common to all instances of a class.
  - For example, a variable that could keep a count of how many objects of a class have been created.

Static Variables and Methods

- Java creates only one copy for a static variable, which can be used even if the class is never instantiated.
- A static variable can be accessed using the dot notation just like instance variables.
- But a static variable can also be accessed with the class itself.
- Static methods are also called without using objects.

Restrictions of Static Methods

- They can only call other static methods.
- They can only access static data of the class

Static Variables and Methods

```
class FamilyMember {
   static String  Lastname = “Smith";
   String Firstname;
   int age;
   .........
}
System.out.println("Family's Last Name
is + FamilyMember.Lastname);
```

Static Variables and Methods

- In the above example, each instance of the class **`FamilyMember`** will have their own values for **`Firstname`** and **`age`**.
- But the static variable **`Lastname`** has only one value for all the family members.
- If the **`Lastname`** is changed all the instances of the class **`FamilyMember`** will have changed **`LastName`**.

When are methods declared as static methods?

- Methods which are of general use, but directly affect an instance of the class are declared as static methods.
- Java class libraries contain a large number of static methods. For example, the **`Math`** class of Java library defines many static methods to perform math operations that can be used in a program. For example,

```
float a =  Math.sqrt(25.0);
int largest = Math.max(x,y);
```

Example of Static Variables, Methods

```
class demo {
 static float mul(float x, float y)
 {
     return x * y;
 }
 static float divide(float x, float y)
 {
     return x / y;
 }
} // class demo
```

Example of Static Variables, Methods

```
class demoStatic {
   public static void main(String args[]) {
   demo a1 = new demo( );
   float a = a1.mul(32, 4);
   float b = demo.divide(4, 2);
   System.out.println(" value of a is = "+ a);
   System.out.println("value of b is = " + b);
   }
}  //class demoStatic
```

Objects and Classes

- Java is a true object-oriented programming language, with programs consisting of classes.
- Anything we wish to represent in a Java program must be encapsulated in a class, which defines the state and behavior of the basic program components known as objects.
- Classes have data items and functions which work on these data items.
- In Java, data items are called as fields and the functions are called as methods.

Defining a Class

- A class is a user-defined data type,which packs together a group of logically related data items and methods that work on these data items.
- Once a class has been defined, variables of that class can be declared.
- In Java these variables are termed as instances of classes i.e. objects.

Defining a Class

- The basic form of class definition is:

```
class classname  [extends superclassname] {
  [variable declaration; ]
  [methods declaration; ] }
```

- Classname and superclassname are any valid Java identifiers.
- **`extends`** is a keyword, which indicates that the properties of superclassname class are extended to the classname class. This concept is called as inheritance.

Defining a Class

```
class empty {
  /* this is a valid class definition
}
```

- In this class **`empty`**, the body is empty.
- This means class **`empty`** does not contain any properties and therefore cannot do anything. But we can compile it and even create objects using it.

Adding Variables to a Class

- Data is encapsulated in a class by placing data fields inside the body of class definition.
- Can declare the instance variables exactly the same way local variables are declared.

Adding Variables to a Class

```
class Rectangle {
   int length; //instance variable
   int width;
}
```

- The class **`Rectangle`** contains two integer type instance variables. Instance variables of the same type can be declared on the same line.
- These instance variables are declared, but they have not been allocated any storage space in the memory yet.

Adding Methods to a Class

- Methods are necessary for manipulating the data contained in the class.
- Methods are declared inside the body of the class but immediately after the declaration of instance variables.
- The general form of a method declaration is:

```
type methodname (parameter list) {
  method body;
}
```

Adding Methods to a Class

- Method declaration has four basic parts
  - The name of the method (methodname) **`methodname`** is a valid identifier.
  - The **`type`** of the value the method returns. This could be a simple data type such as an **`int`** or a user-defined type i.e. a class. It could be of type **`void`**, if the method does not return any value.

Adding Methods to a Class

- Method declaration parts (contd.)
  - Parameter List - always enclosed in parentheses. This list contains variable names and input types. The variables in the parameter list are separated by commas. In case, there are no parameters that have to be passed to the method, the declaration must retain the empty parentheses.
  - Method body - describes the operations to be performed on the data.

Adding Methods to a Class

- Examples of Valid Method Declaration

```
int Area(int length, int width);
int Average(int num1, int num2);
void Display();
void getData(int x, int  y);
```

Adding Methods to a Class

```
class Rectangle {
    int length, width;
    void getData(int x, int y) {
            length = x;
            width = y; }
    int rectArea() {
            int area = length * width;
            return area;
    }
} //class Rectangle
```

Creating Objects

- An object in Java is a block of memory that contains space to store the instance variables.
- Creating an object is also referred to as instantiating an object.
- Objects in Java are created using the **`new`** operator.
- The **`new`** operator creates an object of the specified class and returns a reference to that object.

Creating Objects

// Instantiating objects using **`new`** operator

```
Rectangle rect1;
rect1 = new Rectangle( );
rect2 = new Rectangle( );
```

- The statement **`Rectangle rect1`**; declares a variable **`rect1`** of type **`Rectangle`** to hold the object reference and the statement **`rect1 = new Rectangle( );`** assigns the object reference to **`rect1`** . **`rect1`** is now an object of class **`Rectangle`**.

Creating Objects

- Each object has its own copy of the instance variables of the class. This means any changes to the instance variables of one object will have no effect on the instance variables of another object.
- It is also possible to create two or more references to the same object as given below:

```
Rectangle r1 = new Rectangle();
Rectangle r2 = r1; // assigning one object reference variable to another
```

- Now both **`r1`** and **`r2`** refer to the same object.

Accessing Class Members

- When the object has been created, each object contains its own set of instance variables.
- In order to use these variables in the program, we have to assign values to these variables.
- Cannot access the instance variables and methods directly outside the class. For this, we have to use the object of that class and the dot operator.

Accessing Class Members

- The syntax of how an object is used to access the instance variable or a method is given below:

```
objectname.variable name;
objectname.methodname(parameter list);
```

  - objectname is the name of the object
  - variablename is the name of the instance variable inside the object
  - methodname is the method that is called
  - parameter list is a comma separated list of actual values or expressions that must match in type, number and order with the parameter list of the methodname declared in the class.

Advanced Method Concepts

- A method of a class can be called by:
  - an object of that class, using the dot operator.
  - another method of the same class by using the name of that method. This is called as nesting of methods.

```
class Nesting {
  int m, n;
  Nesting (int x, int y) {
      m = x; n = y; } //Contd.
```

Advanced Method Concepts

```
int largest( ); // to find out if m, n is
large { if(m > n)
    return m;
else
    return n; }
void display( ) {
    int large = largest( );
    System.out.println("Largest number    =
    “     + large);  }
} // end of the definition of class Nesting
```

Advanced Method Concepts

```
class NestingTest {
   public static void main(String args[]) {
       Nesting example = new Nesting(50, 40);
       example.display();
   }
} //NestingTest
```

- The class **`Nesting`** defines one constructor and two methods. These two methods are: **`largest()`** and **`display()`**. The method **`display()`** calls the method **`largest()`** to determine the largest of the two numbers and then displays the result.

Method Call in Java (Program1.java)

- To call a particular method, object of the class in which that method is stored has to be used to call that method.
- The syntax for calling a method in Java is:

  ```
  objectName.methodName(arg1,arg2,arg3);
  ```

  - objectName is the name of the object of the class in which the method is defined.
  - methodName is the name of the method that has to be called.

Misc. Things in OOP

- Comparing Objects
  - Regular Objects – (Program2.java)
  - Strings – (Program3.java)
- Usage of “this” operators (Program4.java)
- Command Line arguments – (Program5.java)
- Object Containment – (Program6.java)