Programming with Java

Object-Oriented Programming

In object-oriented programs data is represented by objects. Objects have two sections, fields (instance variables) and methods. **Fields** tell you what an object is. **Methods** (the instructions of java that are used in writing a program) tell you what an object does. These fields and methods are closely tied to the object's real characteristics and behavior. When a program is run, messages are passed back and forth between objects. When an object receives a message, it responds accordingly as defined by its methods.

Object-oriented programming is alleged to have several advantages including:

- Simpler, easier-to-read programs
- More efficient reuse of code
- Faster time to market
- More robust, error-free code

In practice, object-oriented programs have been just as slow, expensive, and buggy as traditional non-object-oriented programs. In large part, this is because the most popular object-oriented language is C++. C++ is a complex, difficult language that shares all the complication of C while sharing none of C's efficiencies. However, it is possible in practice to write clean, easy-to-read Java code. In C++ this is almost unheard of outside of programming textbooks.

Classes in Java:

A class is a blueprint from which individual objects are created. A sample of a class is given below:

```
class Student{
	//defining fields
	int id;//field or data member or instance variable
	String name;
	//creating main method inside the Student class
	public static void main(String args[]){
	//Creating an object or instance
	Student s1=new Student();//creating an object of Student
	//Printing values of the object
	System.out.println(s1.id);//accessing member through reference
	variable
	System.out.println(s1.name);  }}
```

A class can contain any of the following variable types.

• *Local variables*: Variables defined inside methods, constructors or blocks are called local variables. The variable will be declared and initialized within the method and the variable will be destroyed when the method has completed.

```
public class Test{
      public void ageDifference(int a, int b){
            int difference= a-b;
            System.out.println("The age difference is : " + difference);
      }
      public static void main(String args[]){
            Test test = new Test();
            test.ageDifference(45, 20);
      }
}
```

• *Instance variables*: Instance variables are variables within a class but outside any method. These variables are initialized when the class is instantiated. Instance variables can be accessed from inside any method, constructor or blocks of that particular class.

```
public class Employee{
      // this instance variable is visible for any child class.
      public String name;
      // salary variable is visible in Employee class only.
      private double salary;
      // The name variable is assigned in the constructor.
      public Employee (String empName){
            name = empName;
      }
      // The salary variable is assigned a value.
      public void setSalary(double empSal){
            salary = empSal;
      }
      // This method prints the employee details.
      public void printEmp(){
            System.out.println("name : " + name );
            System.out.println("salary :" + salary);
      }
      public static void main(String args[]){
            Employee empOne = new Employee("Ransika");
            empOne.setSalary(1000);
```

```
        empOne.printEmp();
    }
}
```

• ***Class variables***: Class variables are variables declared within a class, outside any method, with the static keyword.

```
public class Employee{
    // salary variable is a private static variable
    private static double salary;
    // DEPARTMENT is a constant
    public static final String DEPARTMENT = "Development ";
    public static void main(String args[]){
        salary = 1000;
        System.out.println(DEPARTMENT + "average salary:" + salary);
    }
}
```

Static Variable Example

```
class Student {
    int rollno;
    String name;
    static String college = "ITS";
    Student(int r,String n){
        rollno = r;
        name = n;
    }
    void display() {
        System.out.println(rollno + " " + name + " " + college);
    }
    public static void main(String args[]) {
        Student s1 = new Student(111, "Karan");
        Student s2 = new Student(222, "Aryan");
        s1.display();
        s2.display();
    }
}
```

Multiple Classes

We can have multiple classes in different Java files or single Java file. If you define multiple classes in a single Java source file, it is a good idea to save the file name with the class name which has main() method. Only one class should have a main method and it is also the only class that can be declared public.

Example:

```
//Creating Student class.
class Student{
	int id;
	String name;
}
//Creating another class TestStudent1 which contains the main method
class TestStudent1{
	public static void main(String args[]){
		Student s1=new Student();
		System.out.println(s1.id);
		System.out.println(s1.name);
	}
}
```

***Note**: it is not recommended to have multiple classes in one file. Do that only if your program requires it.

Constructors:

Every class has a constructor. If we do not explicitly write a constructor for a class, the Java compiler builds a default constructor for that class.

Each time a new object is created, at least one constructor will be invoked. The main rule of constructors is that they should have the same name as the class. A class can have more than one constructor.

Example of a constructor is given below:

```
public class Student{
	public Student(){
	}
	public Student(String name){
	// This constructor has one parameter, name.
	}
}
```

Example

```
class Counter {
	static int count = 0;
	Counter() {
		count++;
		System.out.println(count);
	}
	public static void main(String args[]) {
	Counter c1 = new Counter();
	Counter c2 = new Counter();
	Counter c3 = new Counter();
	}
}
```

Creating an Object:

As mentioned previously, a class provides the blueprints for objects. So basically, an object is created from a class. In Java, the new key word is used to create new objects.

There are three steps when creating an object from a class:

• **Declaration:** A variable declaration with a variable name with an object type.

• **Instantiation:** The 'new' key word is used to create the object.

• **Initialization:** The 'new' keyword is followed by a call to a constructor. This call initializes the new object.

Example of creating an object is given below:

```
public class Puppy{
	public Puppy(String name){
		// This constructor has one parameter, name.
		System.out.println("Passed Name is :" + name );
	}
	public static void main(String []args){
		// Following statement would create an object myPuppy
		Puppy myPuppy = new Puppy( "tommy" );
	}
}
```

Accessing Instance Variables and Methods:

Instance variables and methods are accessed via created objects. To access an instance variable the fully qualified path should be as follows:

```
/* First create an object */
ObjectReference = new Constructor();
/* Now call a variable as follows */
ObjectReference.variableName;
/* Now you can call a class method as follows */
ObjectReference.MethodName();
```

Example:

This example explains how to access instance variables and methods of a class:

```
public class Puppy{
	int puppyAge;
	public Puppy(String name){
		// This constructor has one parameter, name.
		System.out.println("Name chosen is :" + name );
	}
	public void setAge( int age ){
		puppyAge = age;
	}
	public int getAge( ){
		System.out.println("Puppy's age is :" + puppyAge );
		return puppyAge;
	}
	public static void main(String []args){
		/* Object creation */
		Puppy myPuppy = new Puppy( "tommy" );
		/* Call class method to set puppy's age */
		myPuppy.setAge( 2 );
		/* Call another class method to get puppy's age */
		myPuppy.getAge( );
		/* You can access instance variable as follows as well */
		System.out.println("Variable Value :" + myPuppy.puppyAge );
	}
}
```

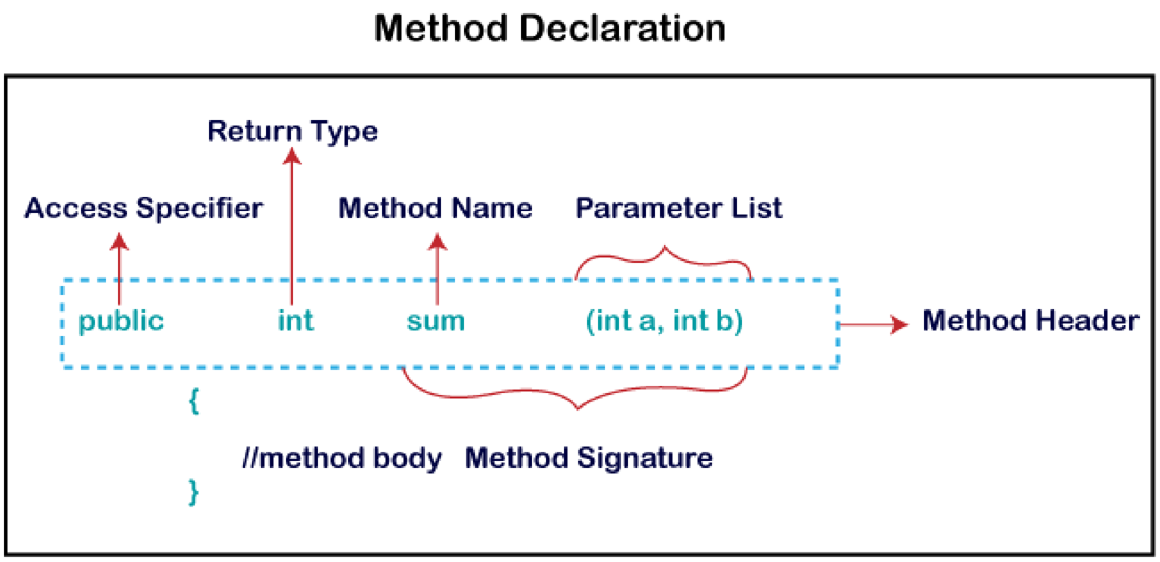
Java Methods

A **method** is a block of code or collection of statements, or a set of code grouped together to perform a certain task or operation. It is used to achieve the **reusability** of code. We write a method once and use it many times. We do not require to write code again and again. It also provides the **easy modification** and **readability** of code, just by adding or removing a chunk of code. The method is executed only when we call or invoke it.

The most important method in Java is the **main()** method.

Method Declaration

The method declaration provides information about method attributes, such as visibility, return-type, name, and arguments. It has six components that are known as **method header**, as shown in the following figure.



Method Signature: Every method has a method signature. It is a part of the method declaration. It includes the **method name** and **parameter list**.

Access Specifier: Access specifier or modifier is the access type of the method. It specifies the visibility of the method. Java provides **four** types of access specifier:

- **Public:** The method is accessible by all classes when we use public specifier in our application.
- **Private:** When we use a private access specifier, the method is accessible only in the classes in which it is defined.
- **Protected:** When we use protected access specifier, the method is accessible within the same package or subclasses in a different package.
- **Default:** When we do not use any access specifier in the method declaration, Java uses default access specifier by default. It is visible only from the same package only.

Return Type: Return type is a data type that the method returns. It may have a primitive data type, object, collection, void, etc. If the method does not return anything, we use void keyword.

Method Name: It is a unique name that is used to define the name of a method. It must be corresponding to the functionality of the method. Suppose if we are creating a method for subtraction of two numbers, the method name must be **subtraction().** A method is invoked by its name.

It is also possible that a method has the same name as another method name in the same class, it is known as **method overloading**.

Parameter List: It is the list of parameters separated by a comma and enclosed in the pair of parentheses. It contains the data type and variable name. If the method has no parameter, let the parentheses blank.

Method Body: It is a part of the method declaration. It contains all the actions to be performed. It is enclosed within the pair of curly brackets.

Types of Methods

There are two types of methods in Java: Predefined Method and User-defined Method

- Predefined Method

In Java, predefined methods are the method that is already defined in the Java class libraries is known as predefined methods. It is also known as the **standard library method** or **built-in method**. We can directly use these methods just by calling them in the program at any point. Some pre-defined methods are **length(), equals(), compareTo(), sqrt(),** etc. When we call any of the predefined methods in our program, a series of codes related to the corresponding method runs in the background that is already stored in the library.

Each and every predefined method is defined inside a class. Such as **max()** method is defined in the **Java.lang.Math** class. It finds the largest number between two.

Example:

```
public class PredefinedMethodDemo{
public static void main(String[] args){
// using the max() method of Math class
System.out.print("The maximum number is: " + Math.max(9,7));
}
}
```

- **User-defined Method**

The method written by the user or programmer is known as **a user-defined** method. These methods are modified according to the requirement.

How to Create a User-defined Method

Let's create a user defined method that checks the number is even or odd. First, we will define the method.

```
import java.util.Scanner;
public class EvenOdd  {
      public static void main (String args[]){
            Scanner scan=new Scanner(System.in);
            System.out.print("Enter the number: ");
            int num=scan.nextInt();
            //method calling
            findEvenOdd(num);
      }
      //user defined method
      public static void findEvenOdd(int num) {
            if(num%2==0)
            System.out.println(num+" is even");
            else
            System.out.println(num+" is odd");
      }
}
```

Let's see another program that return a value to the calling method.

```
public class Addition {
      public static void main(String[] args){
            int a = 19, b = 5;
            //method calling
            int c = add(a, b);
      System.out.println("The sum of a and b is= " + c);
      }
      //user defined method
      public static int add(int n1, int n2){
            int s;
            s=n1+n2;
            return s; //returning the sum  }}
```

Static Method

A method that has static keyword is known as static method. In other words, a method that belongs to a class rather than an instance of a class is known as a static method. We can also create a static method by using the keyword **static** before the method name.

The main advantage of a static method is that we can call it without creating an object. It can access static data members and change the value of it. It is used to create an instance method. It is invoked by using the class name. The best example of a static method is the **main()** method.

```java
public class Display {
	public static void main(String[] args) {
		show();
	}
	static void show() {
		System.out.println("It is an example of static method.");
	}
}
```

If the method is not declared as static, then we must create an object of its class before calling it.

```java
public class InstanceMethodExample  {
	public static void main(String [] args) {
		//Creating an object of the class
		InstanceMethodExample obj = new InstanceMethodExample();
		//invoking instance method
		System.out.println("The sum is: "+obj.add(12, 13));
	}
	//user-defined method because we have not used static keyword
	public int add(int a, int b) {
		//returning the sum
		return a+b;
	}
}
```

Method Overloading

If a class has multiple methods with the same name but different in parameters, it is known as ***Method Overloading***. If we need to perform only one operation, having same name of the methods increases the readability of the program.

Suppose your program have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as *a(int,int)* for two parameters, and *b(int,int,int)* for three parameters, then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs.

So, we perform method overloading to figure out the program quickly.

There are two ways to overload the method in java

1. By changing number of arguments
2. By changing the data type

Example 1: Changing no. of arguments

In this example, we have created two methods, first add() method performs addition of two numbers and second add method performs addition of three numbers.

```
class TestOverloading1{
	static int add(int a,int b){
		return a+b;
	}
	static int add(int a,int b,int c){
		return a+b+c;}
	}
	public static void main(String[] args){
		System.out.println(add(11,11));
		System.out.println(add(11,11,11));
	}
}
```

Example 2: Changing data type of arguments

In this example, we have created two methods that differs in data type. The first add method receives two integer arguments and second add method receives two double arguments.

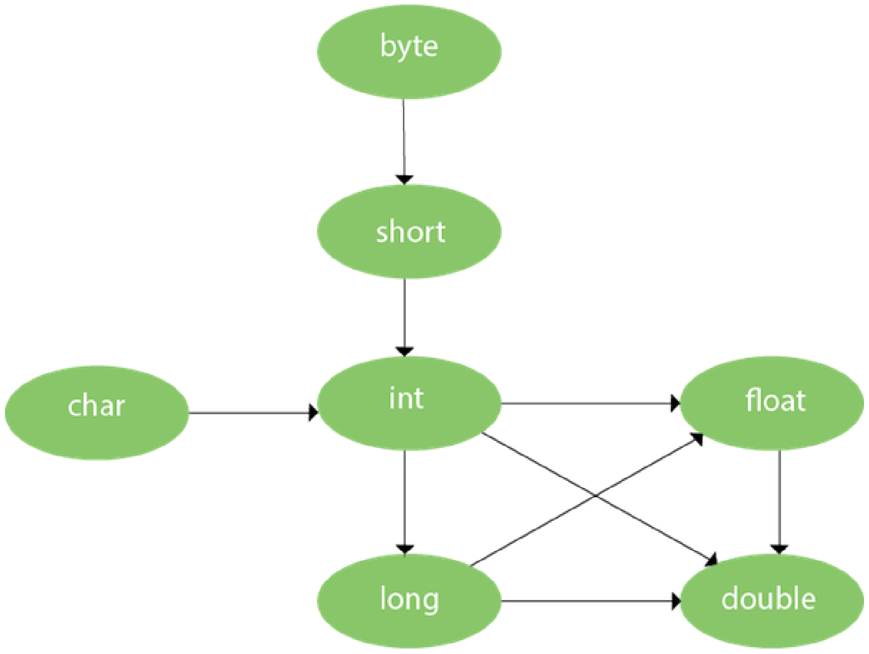
```
class TestOverloading2{
	static int add(int a, int b){
		return a+b;
		}
	static double add(double a, double b){
		return a+b;
		}

	public static void main(String[] args){
		System.out.println(add(11,11));
```

```
                System.out.println(add(12.3,12.6));
        }
}
```

**Note*: You can have any number of main methods in a class by method overloading. But JVM calls main() method that receives string array as arguments only.

Note: When calling methods, one data type is promoted to another implicitly if no matching datatype is found. Let's understand the concept by the figure given below:



As displayed in the above diagram, byte can be promoted to short, int, long, float or double. The short datatype can be promoted to int, long, float or double. The char datatype can be promoted to int, long, float or double and so on.

Example of Method Overloading with Type Promotion

```
class OverloadingCalculation1{
        void sum(int a,long b){
                System.out.println(a+b);
                }
        void sum(int a,int b,int c){
                System.out.println(a+b+c);
                }

        public static void main(String args[]){
```

```
        OverloadingCalculation1 obj=new OverloadingCalculation1();
        obj.sum(20,20);//second int literal will be promoted to long
        obj.sum(20,20,20);
    }
}
```