

Contents

- Method Overloading
- Static members
- Inheritance

Method Overloading

- Methods that have different parameter lists and different definitions.
- It is used when objects are required to perform similar tasks but using different input parameters.
- When we call a method in an object, Java matches up the method name first and then the number and type of parameters to decide which one of the definitions to execute.
- This process is known as “**Compile time polymorphism**” or “**static polymorphism.**”

Different Number of parameters in argument list

```
class DisplayOverloading
{
    public void disp(char c)
    {
        System.out.println(c);
    }
    public void disp(char c, int num)
    {
        System.out.println(c + " "+num);
    }
}
```

```
class Sample
{
    public static void main(String args[])
    {
        DisplayOverloading obj = new DisplayOverloading();
        obj.disp('a');
        obj.disp('a',10);
    }
}
```

Difference in data type of parameters

```
class DisplayOverloading2
{
    public void disp(char c)
    {
        System.out.println(c);
    }
    public void disp(int c)
    {
        System.out.println(c );
    }
}
```

```
class Sample2
{
    public static void main(String args[])
    {
        DisplayOverloading2 obj = new DisplayOverloading2();
        obj.disp('a');
        obj.disp(5);
    }
}
```

Sequence of data type of arguments

```
class DisplayOverloading3
{
public void disp(char c, int num)
{
System.out.println("I m the first definition of method disp"+c+num);
}
public void disp(int num, char c)
{
System.out.println("I m the 2nd definition of method disp"+num+c );
}
}
```

```
class Sample3
{
public static void main(String args[])
{
DisplayOverloading3 obj = new DisplayOverloading3();
obj.disp('x', 51 );
obj.disp(52, 'y');
}}
```

Method return type doesn't matter in case of overloading.

```
class Demo2
{
    public double myMethod(int num1, int num2)
    {
        System.out.println("First myMethod of class Demo");
        return num1+num2;
    }
    public int myMethod(int var1, int var2)
    {
        System.out.println("Second myMethod of class Demo");
        return var1-var2;
    }
}

class Sample5
{
    public static void main(String args[])
    {
        Demo2 obj2= new Demo2();
        obj2.myMethod(10,10);
        obj2.myMethod(20,12);
    }
} //compile time error
```

Static Members(Variables/methods)

- Every time the class is instantiated , a new copy of instance variables and instance methods is created, they are accessed using the objects.
- We want to define a member that is common to all objects and accessed without using a particular object.
- Such members can be defined as follows:

static int count;

static int max(int x,int y);

Static Members(Variables/methods)

- Since these members are associated with the class ,they are referred to as **class variables** and **class methods**.
- Static variables and static methods can be called without using the objects.
- They are also available for use by other classes.
- Java class libraries contain a large number of class methods.
- Math class for math operations:min(), max(), avg(), sin(), cos(), tan(), round(),pow()

e.g. `double x=Math.sqrt(25.0);`

Inheritance

Inheritance

Inheritance

- Reusability
- Deriving a new class from an old one
- Inheritance allows subclasses to inherit all the variables and methods of their parent classes.
- Forms:
 - Single inheritance
 - Multiple inheritance
 - Hierarchical inheritance
 - Multilevel inheritance

Defining a subclass

```
class subclassname extends superclassname  
{  
    variable declaration;  
    methods declaration;  
}
```

- The keyword `extends` signifies that the properties of the superclassname are extended to subclassname.
- The subclass will now contain its own variables and methods as well those of the superclass.

Subclass Constructor

- A subclass constructor is used to construct the instance variables of both the subclass and the superclass .
- The subclass constructor uses the keyword `super` to invoke the constructor method of the superclass.
- The keyword **`super`** is used provided:
 - Super may only be used within a subclass constructor method.
 - The call to superclass constructor must appear as the first statement within the subclass constructor.
 - The parameters in the super class must match the order and type of the instance variable declared in the superclass.

Multilevel Inheritance

Class A

```
{  
  -----  
  -----  
}
```

Class B extends A

```
{  
  -----  
  -----  
}
```

Class C extends B

```
{  
  -----  
  -----  
}
```

Overriding Methods

- A method defined in a super class is inherited by its subclass and is used by the objects created by the subclass.
- If we want an object to respond to the same method but have different behaviour when that method is called. We override the method defined in the superclass.
- This is possible by defining a method in the subclass that has the same name, same arguments and same return type as a method in the superclass.
- When that method is called, the method defined in the subclass is invoked and executed instead of the one in the superclass.
- This is known as **method overriding**.

Final variables and methods

- All methods and variables can be overridden by default in subclasses.
- If we wish to prevent the subclasses from overriding the members of the superclass, we can declare them as final using the keyword **final** as a modifier.

```
final int SIZE=100;  
final void showstatus();
```

Final classes

- Sometimes we may like to prevent a class being further subclasses for security reasons.
- A class that cannot be subclassed is called a final class.

final class abc{.....}

final class pqr extends someclass{.....}

- Any attempt to inherit these classes will cause an error and the compiler will not allow it.

Abstract methods and classes

- To indicate that a method must always be redefined in a subclass, thus making overriding compulsory.
- This is done using the modifier keyword **abstract** in the method definition.

```
abstract class shape
{
.....
    abstract void draw();
.....
}
```

- When a class contains one or more abstract methods ,it should be declared **abstract**

Dynamic Method Dispatch

- Method overriding is one of the ways in which Java supports Runtime Polymorphism.
- Dynamic method dispatch is the mechanism by which a call to an overridden method is resolved at run time, rather than compile time.
- When an overridden method is called through a superclass reference, Java determines which version(superclass/subclasses) of that method is to be executed based upon the type of the object being referred to at the time the call occurs. Thus, this determination is made at run time.

Dynamic Method Dispatch

- At run-time, it depends on the type of the object being referred to (not the type of the reference variable) that determines which version of an overridden method will be executed
- A superclass reference variable can refer to a subclass object. This is also known as upcasting. Java uses this fact to resolve calls to overridden methods at run time.

Dynamic Method Dispatch

- Therefore, if a superclass contains a method that is overridden by a subclass, then when different types of objects are referred to through a superclass reference variable, different versions of the method are executed.

Upcasting

SuperClass obj = new SubClass

