Polymorphism in Java

The word polymorphism means having many forms.Polymorphism is considered one of the important features of Object-Oriented Programming. Polymorphism allows us to perform a single action in different ways. In other words, polymorphism allows you to define one interface and have multiple implementations. The word “poly” means many and “morphs” means forms, So it means many forms.

Types of Java polymorphism

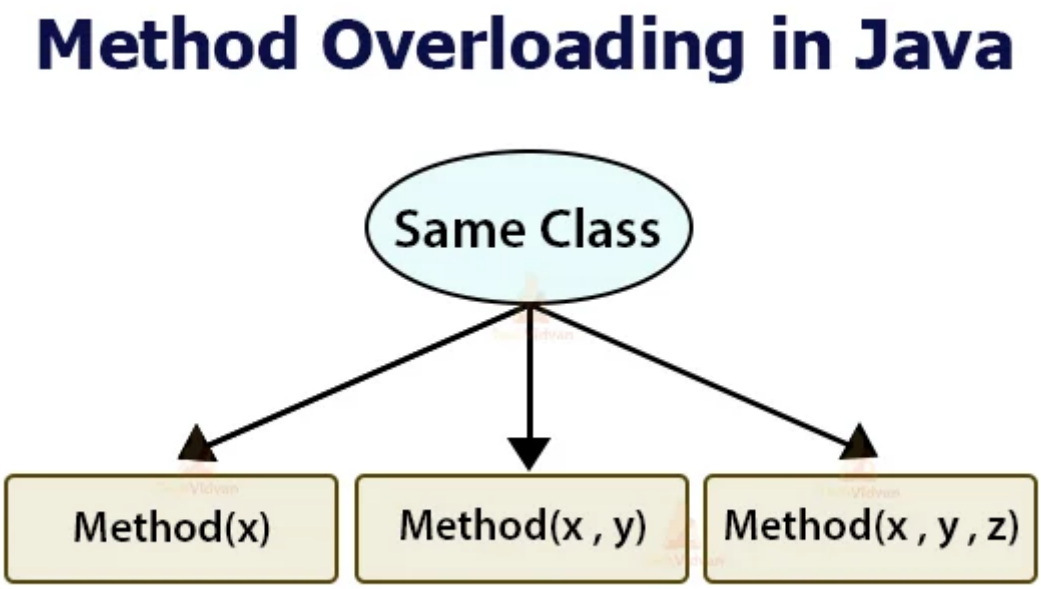
In Java polymorphism is mainly divided into two types:

- Compile-time Polymorphism
- Runtime Polymorphism

Compile-Time Polymorphism

It is also known as static polymorphism. This type of polymorphism is achieved by Method overloading or operator overloading(Not Supported in java).

If a class has multiple methods having same name but different in parameters, it is known as **Method Overloading**.



Advantage of method overloading

Method overloading *increases the readability of the program*.

Different ways to overload the method

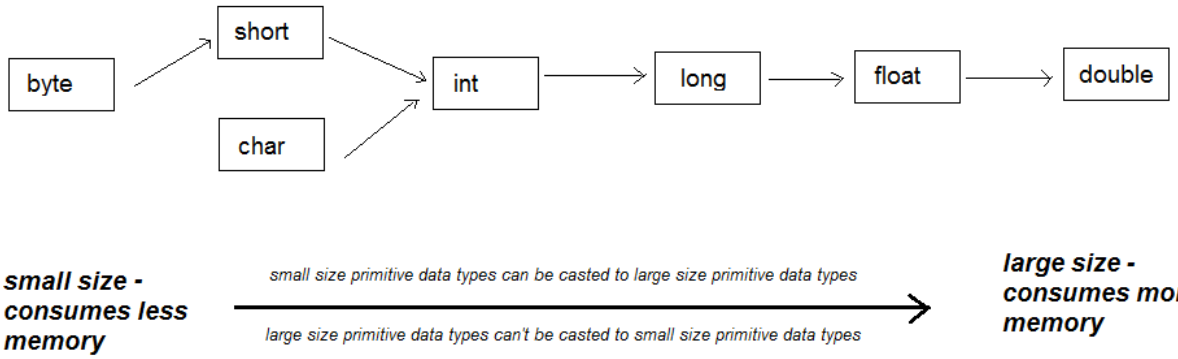
There are two ways to overload the method in java

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

```
package Polymorphism;

public class Overloading {

  public void Display() {
    System.out.println("Inside First Display method");
  }



  public void Display(String val) {
    System.out.println("Inside Second Display method, value is: " + val);
  }

  public void Display(String val1, String val2) {
    System.out.println(
      "Inside Third Display method, values are : " + val1 + "," + val2
    );
  }
```

```
  public void Display(int val1, int val2) {
    System.out.println(
      "Inside Forth Display method, values are : " + val1 + "," + val2
    );
  }

  public void Display(double val1, double val2) {
    System.out.println(
      "Inside Fifth Display method, values are : " + val1 + "," + val2
    );
  }

  public static void main(String args[]) {
    Overloading oo = new Overloading();
    oo.Display(); //Calls the first Display method
    oo.Display("Dev"); //Calls the second Display method
    oo.Display("Debasish", "Sahoo"); //Calls the third Display method
    oo.Display(12, 14); //Calls the Fourth Display method
    oo.Display(1.6, 1.9); //Calls the Fifth Display method
  }
}
```

Method Overloading and Type Promotion



TypePromotion

```
class Add{
  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[]){
  Add obj=new Add();
  obj.sum(20,20);//now second int literal will be promoted to long
```

```
  obj.sum(20,20,20);

  }
}
```

Type Promotion (if matching found)

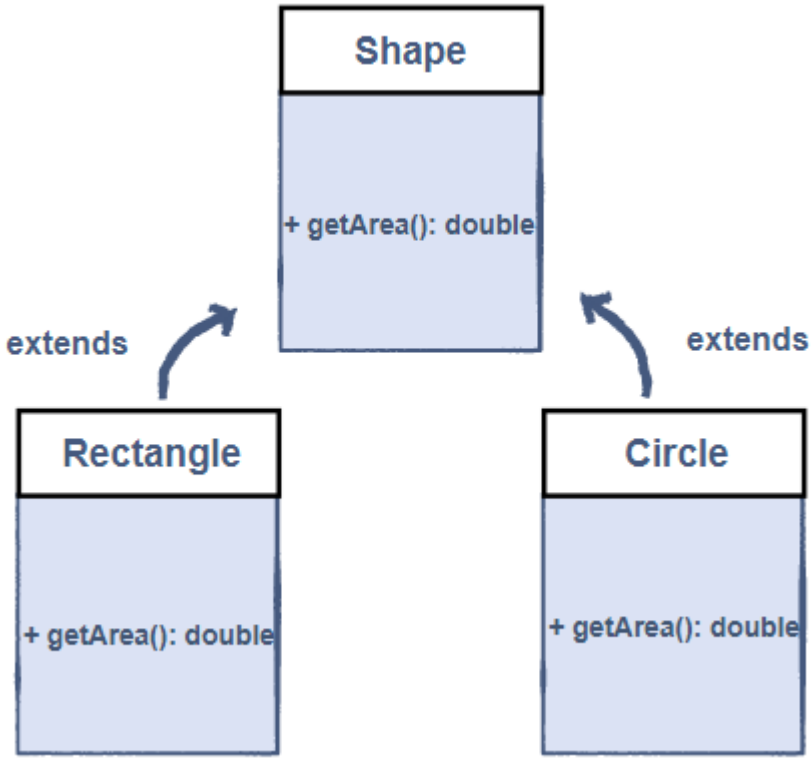
```
class Add{
  void sum(int a,int b){System.out.println("int arg method invoked");}
  void sum(long a,long b){System.out.println("long arg method invoked");}

  public static void main(String args[]){
  Add obj=new Add();
  obj.sum(20,20);//now int arg sum() method gets invoked
  }
}
```

Type Promotion (ambiguity)

```
class Add{
  void sum(int a,long b){System.out.println("a method invoked");}
  void sum(long a,int b){System.out.println("b method invoked");}

  public static void main(String args[]){
  Add obj=new Add();
  obj.sum(20,20);//now ambiguity
  }
}
```

Method Overriding

In Java, Overriding is a feature that allows a subclass or child class to provide a specific implementation of a method that is already provided by one of its super-classes or parent classes. When a method in a subclass has the same name, the same parameters or signature, and the same return type(or sub-type) as a method in its super-class, then the method in the subclass is said to *override* the method in the super-class.



Usage of Java Method Overriding

- Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
- Method overriding is used for runtime polymorphism

Rules for Java Method Overriding

1. The method must have the same name as in the parent class
2. The method must have the same parameter as in the parent class.
3. There must be an IS-A relationship (inheritance).

```
package polymorphism;

class Base {
  void Msg() {
    System.out.println("This is Base class");
  }
}

class Overriding extends Base {
  void Msg() {
    System.out.println("This is Overriding class");
  }
```

```
  public static void main(String args[]) {
    Overriding obj = new Overriding();
    obj.Msg();

    Base obj2 = new Overriding();
    obj2.Msg();
  }
}
```

Rules for method overriding

1. Overriding and Access Modifiers

The **access modifier** for an overriding method can allow more, but not less, access than the overridden method. For example, a protected instance method in the superclass can be made public, but not private, in the subclass. Doing so will generate a compile-time error.

2. Final methods can not be overridden

If we don't want a method to be overridden, we declare it as **final**. Please see **Using Final with Inheritance**.

3. Static methods can not be overridden

When you define a static method with the same signature as a static method in the base class, it is known as **method hiding**.

	Superclass Instance Method	**Superclass Static Method**
Subclass Instance Method	Overrides	Generates a compile-time error
Subclass Static Method	Generates a compile-time error	Hides

4. Private methods can not be overridden

Private methods cannot be overridden as they are bonded during compile time. Therefore we can't even override private methods in a subclass.

5. The overriding method must have the same return type (or subtype)

From Java 5.0 onwards it is possible to have different return types for an overriding method in the child class, but the child's return type should be a sub-type of the parent's return type. This phenomenon is known as the **covariant return type**.

6. Invoking overridden method from sub-class

We can call the parent class method in the overriding method using the **super keyword**.