Polymorphism:

*The word **Polymorphism** means having **Many Forms.***
*The most common use of Polymorphism in OOP occurs when a **Parent Class reference** is used to refer to a **Child Class object**.*

*In Java **Polymorphism** is mainly divided into two types:*
***Compile time Polymorphism** (or Static polymorphism)*
***Runtime Polymorphism** (or Dynamic polymorphism)*

Compile time Polymorphism
Complile time Polymorphism** is achieved by **Method Overloading** or **Operator Overloading.

Runtime polymorphism:
Runtime time Polymorphism** is achieved by **Method Overloading

Method Overloading: →

*If we are having **Two Methods** with **Same Name**, but there should be different **Arguments** or different **Data types**.*
*It is possible to Overload any number of **Methods**.*

***Types of Overloading**:*

1.Method Overloading
2.Constructor Overloading
3.Operator Overloading

Method Overloading

```
public void getFulelCost(int petrolCost, int dieselCost) {

System.out.println("Petrol Cost: " + petrolCost);
System.out.println("Diesel Cost: " + dieselCost);
}

public void getFulelCost(float petrolCost, float dieselCost) {

System.out.println("Petrol Cost: " + petrolCost);
System.out.println("Diesel Cost: " + dieselCost);
}

public static void main(String[] args) {

Product product = new Product();
product.getFulelCost(85, 75);
product.getFulelCost(85f, 75f);
}
```

Method Overloading

```
// Same Method But arguments must be different
public void userName(int sId, String fName, String lName, String roles, long contact) {
System.out.println(sId);
System.out.println(fName);
System.out.println(lName);
System.out.println(roles);
System.out.println(contact);
}
// Same Method But arguments must be different
public void userName(int sId, String fName, String lName, String roles) {
System.out.println(sId);
System.out.println(fName);
System.out.println(lName);
System.out.println(roles);
}

public static void main(String[] args) {
User user = new User();
user.userName(101, "Sai", "Kiran", "Singer", 9876543210L);
user.userName(101, "Sai", "Kiran", "Dancer");
}
```

Operator Overloading:

One operator can act as more than **One form** is called **Operator Loading**.

The only **operator** overloaded in java language is ‘+’
we can make the operator (‘+’) for **string class to concatenate two strings.**
+ operator to **add integers.**

```
int a = 10;
int b = 20;
System.out.println(a + b); // 30

String s1 = "10";
String s2 = "20";
System.out.println(s1+s2); //1020
```

Overloaded Constructors: →

*A **class** contain more than **one constructor**, and all these **constructors** having **same name** but **different arguments.***
*Hence these Constructors are called **Overloaded Constructors.***

Important Notes:
→Parent class constructor by default available to child.
→Hence Overriding concept is not applicable for constructors
→Constructors can be Overloaded
→We can take constructor in any class including abstract class. But, we cant take inside the Interface.
→One Constructor is used to call only one Constructor at a time.
→Methods can call multiple methods at a time.
*→The applicable modifiers for the constructor are **public, private, protected and default.***

Constructor

```
public Eg1() {
System.out.println("Default Constructor");
}

public Eg1(int a, int b) {
System.out.println(a+b);
}

public Eg1(int a, int b, int c) {
System.out.println(a + b + c);
}

public Eg1(float a, float b) {
System.out.println(a + b);
}

new Eg1();
new Eg1(10, 20);
new Eg1(10f, 20f);
new Eg1(10, 20, 30);
```

Method Overriding: →

*If the **Child Class** not satisfy the **Parent Class** **Method Implementation.***
*Then it is possible to override that method in the **child class** based on the **child class requirement.***

If we want to work on Method Overriding we need two classes.

*If we are overriding the **Child Class** and **Parent Class** **Method Signatures** must be same otherwise we are getting compilation error.*

Method Overiding

```
public class Parent {
public void m1() {
System.out.println("M1 Method Parent");
}
}
```

```
public class Child extends Parent {

public void m1() {
System.out.println("M1 Method Child");
}

public static void main(String[] args) {

//Parent p = new Parent();
//p.m1(); // calling parent class method

//Child child = new Child();
//child.m1(); // overriding happened, but we need super class ref, not sub class ref

Parent p = new Child();
p.m1(); //overriding happened, right way to maintain overriding
}
}
```

Method Overiding

```java
public class Vechiles {
public void buyVechile(String vechileName, String vechileColor ) {
System.out.println("Vechile Name: " + vechileName);
System.out.println("Vechile Color: " + vechileColor);
}
}
```

```java
public class TestDrive extends Vechiles {

public void buyVechile(String vechileName, String vechileColor) {
System.out.println("Vechile Name: " + vechileName);
System.out.println("Vechile Color: " + vechileColor);
}

public static void main(String[] args) {
Vechiles p = new TestDrive();
p.buyVechile("BMW", "Black");
}
}
```

Difference between Overriding and Overloading?

Overloading	***Overriding***
In Overloading Method names must be same	*In Overriding Method names must be same*
In Overloading Arguments must be different	*In Overriding Arguments must be same including order.*
In Overloading Method signature must be different, Method Overloading is performed with in class.	*In Overriding Method signature must be same, Method overriding occurs in two classes that have IS-A(inheritance) relationship*
*Method Overloading is an example of **compile time polymorphism** or **static** or **early binding.***	*Method Overriding is an example of **run time polymorphism** or **dynamic polymorphism** or **late binding**.*
Private, static, final methods can be Overloaded.	*Private, static, final methods cannot be Overridden.*