**Polymorphism**-

One entity that behaves differently in different cases called as polymorphism.

Example- Light button, we are using that button to on or off the lights.

**How to achieve polymorphism in java?**

We can achieve polymorphism by using two ways.

1. Method overloading-
2. Method overriding-
3. Method overloading-

It is the same method name with different argument called as Method overloading. There is no need of super and sub class relationship.

It is also called as early binding, compile time polymorphism or static binding.

Rules-

* Method name must be same.
* Parameter or argument must be different.(sequence ofargument,numberof
* argumentordatatypeshouldbedifferent)
* Return type is anything
* Access specifier is anything
* Exceptionthrowncanbeanything

**Example-1**

**package** com.tests;

**public** **class** TestMain {

**void** add(**int** a, **int** b) {

System.***out***.println(a + b);

}

**void** add(**double** a, **double** b) {

System.***out***.println(a + b);

}

**void** add(**double** a) {

System.***out***.println(a);

}

**void** add(**int** a, **int** b, **int** c) {

System.***out***.println(a + b + c);

}

}

**package** com.tests;

**public** **class** ExampleMain {

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

TestMain testmain = **new** TestMain();

testmain.add(10.5);

testmain.add(10.5, 11.5);

testmain.add(2, 4);

testmain.add(5, 10, 15);

}

}

Output is

10.5

22.0

6

30

**Why?**

Suppose we got the business requirement from the client in last year

Class Employee {

Void addEmployee (String firstname, string lastname, string city) { }

End user is calling the class as below

//End User 1

addEmployee (“ram”,”pawar”,”Pune”);

//End User 2

addEmployee (“ram”,”deshmukh”,”Mumbai”);

After that I got the new requirement from the client in current year, to update the pan card details.

What options we have?

1. Modified into the existing method.

2. Create the new method with new parameter.

First way modifying into existing method is not good approach, it will increase the unit testing of it. If we are make the changes into existing method, then how user calls the method I mean they need to add one more extra field, in future again, you got requirement to add one more field so every time user need to change at their side, this is not the good thing.

Second way, create the same method in that class and add the new field into it. If client second want pan card details so he can call that method otherwise calls the first method if pan card is not required.

**Example- 2**

**package** com.poly;

**public** **class** A{

**void** test(Object object) {

System.***out***.println("test- Object");

}

**void** test(String string) {

System.***out***.println("test- String");

}

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

A a = **new** A();

a.test(**new** Object());

a.test("ram");

a.test(**new** A());

a.test(**new** String());

}

}

test- Object

test- String

test- Object

test- String

Why it is called as compile time polymorphism?

Because it is decided at compile time which one method should get called that’s why it is called as compile time polymorphism.

In overloading compiler is responsible to perform method resolution (decision) based on the reference type (but not based on runtime object). Hence overloading is also considered as compile time polymorphism (or) static polymorphism (or) early biding.

In overloading method resolution is always based on reference type and runtime object won't play any role in overloading.

1. **Method overriding-**
   * It is the same method name with same argument called as method overriding.
   * There is need of super and sub relationship.
   * It is also known as late binding, run time polymorphism or dynamic binding. etc.

Whatever the Parent has by default available to the Child through inheritance, if the Child is not satisfied with Parent class method implementation then Child is allow to redefine that Parent class method in Child class in its own way this process is called overriding.

In overriding method resolution is always takes care by JVM based on runtime object hence overriding is also considered as runtime polymorphism or dynamic polymorphism or late binding.

Rules-

* Method name must be same.
* Method argument also be same
* Return type must be same or different.(Till 1.4 it must be same later covariant return types are allowed.)
* Access specifier is anything.

Note-

* we can extend the method scope in overriding but not reduce the visibility of it.
* While overriding if the child class method throws any checked exception compulsory the parent class method should throw the same checked exception or its parent otherwise we will get compile time error.

Why?

* Maintainability
* Readability of code.

Example-

package com.override.demo;

public class A {

void m1() {

System.out.println("class - A- m1 () method");

}

}

package com.override.demo;

public class B extends A {

@Override

void m1() {

System.out.println("class - B- m1 () method");

}

void m7() {

System.out.println("class- B- m7() method");

}

}

package com.override.demo;

public class TestMain {

public static void main(String[] args) {

B b= new B();

b.m1();

b.m7();

}

}

Output-

class - B- m1 () method

class- B- m7() method

Program Explaination-

* In the above program, B is implementing the method m1 () with the same signature as super class A i.e m1 () of class B is overriding m1() of class A.
* If you want to add new features to existing class, then you should not disturb the existing class. You should always write the subclass of that class that is the best practice.

Why we write the sub class

* To add the new features
* To inherit the existing functionality.

Subclass method's access modifier must be the same or higher than the superclass method access modifier

|  |  |
| --- | --- |
| Superclass | In subclass, we can have access specifier |
| public | public |
| protected | protected, public |
| Default | default, protected, public |
| Private | We cannot override the private |

**Method Overloading**- **Live Example-1**

Class MobilePattern{

void getMobilePattern(Thumb thumb){

//logic here

}

void getMobilePattern(int number){

//logic here

}

void getMobilePattern(int x1, int y1, int x2, int y2){

//logic here

}

}

**Live Example-2**

Class Banking{

void getBanking(CreditCard creditCard){

//logic here

}

void getBanking(Netbanking netBanking){

//logic here

}

void getBanking(DebitCard debitCard){

//logic here

}

void getBanking(UPI upi){

//logic here

}

}

**Method Overriding**- **Live Example-1**

Class SBI {

void getSimpleIntereset(float simpleRate){

//logic here

}

}

Class Axis extends SBI{

void getSimpleIntereset(float simpleRate){

//logic here

}

}

Class HDFC extends Axis {

void getSimpleIntereset(float simpleRate){

//logic here

}

}

**Live Example-2**

Class FirstTier {

void getSeatAvailability(int seat){

//logic here

}

}

Class SecondTier extends FirstTier{

void getSeatAvailability(int seat){

//logic here

}

}

Class ThirdTier extends SecondTier {

void getSeatAvailability(int seat){

//logic here

}

}