**Polymorphism**-

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

Example-

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

2. a person acts as an employee in the office, a customer in the shopping mall, a passenger in bus/train, a student in school, and a son at home.

3. Smartphone is entity that behaves different such as text message, calling, send mail, video call etc.

**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 called as early binding, compile time polymorphism or static binding.

Rules-

Method name must be same.

Parameter or argument must be different.

Return type is anything

Access specifier is anything

**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 addStudent (String firstname, String lastname, String city) {

}

End user is calling the class as below

//End User 1

Employee employee=new Employee ();

employee. addStudent (“ram”,” pawar”,” Pune”);

//End User 2

employee. addStudent (“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 in this case?

1. Modified field/Attributes/Variable into the existing method.

2. Design new method and add new parameter into it.

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

Second way, design 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.

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.

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 called as late binding, run time polymorphism or dynamic binding. Dynamic

Method dispatch etc.

Rules-

Method name must be same.

Return type must be same or different.

Access specifier is anything.

Parameters must be same.

Note- we can extend the method scope in overriding but not reduce the visibility of it.

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(variable or method) 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.

Note-Why we need to write the sub class

* 1. To add the new features
  2. 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

}

}