Polymorphism-

One entity that behaves differently in different cases called as polymorphism. Or Same name with different forms is the concept of polymorphism

Example- Light button, we are using that button to on or off the lights. Calculation of Income Tax as per income, etc

We can use same abs() method to calculate the absolute value for int type, long type, float type etc.

Example:

1. abs(int)

2. abs(long)

3. abs(float)

How to achieve polymorphism in java?

We can achieve polymorphism by using two ways.

1. Method overloading-
2. Method overriding-

1. 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.

   Having overloading concept in java reduces complexity of the programming.

   Rules-

   1) Method name must be same.
   2) Parameter or argument must be different.(sequence of argument, number of argument or datatype should be different)
   3) Return type can be anything
   4) Access specifier can be anything
   5) Exception thrown can be 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

addStudent (“ram”,”pawar”,”Pune”);

//End User 2

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?

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 existing user will calls the method I mean they need to add one more extra attribute, in future again, you got requirement to add one more attributes 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 attribute 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;

class X { }

class Y extends X { }

class Z extends Y { }

public class Overloading {

	void test1(X x) {
		System.out.println("test1- X"); }

	void test1(Y y) {
		System.out.println("test1- Y"); }

	void test1(Z z) {
		System.out.println("test1- Z"); }
} //package name remove
public class OverloadTestMain {

	public static void main(String[] args) {

		Overloading overloading= new Overloading();
		X x= new X();
		overloading.test1(x);
		Y y= new Y();
		overloading.test1(y);
		Z z= new Z();
		overloading.test1(z);

		Y y1= new Z();
		overloading.test1(y1);
```

```
        } }
```

```
test1- X
test1- Y
test1- Z
test1- Y
```

Example- 3

```
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("Jeevan");
                a.test(new X());
                a.test(new String());
        }
}
```

Output :

```
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 run time 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.

2. **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.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-

1) Method name must be same. That is method signature must be same. Method signature means method name with argument (return type and access specifier are not part of method signature).
2) Until 1.4 version the return types must be same but from 1.5 version onwards covariant return types are allowed.
3) Method parameters must be same.

Note-

1) We can extend the method scope in overriding but not reduce the visibility of it.
2) 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.
   But there are no restrictions for un-checked exceptions.

Why?
Flexibility
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

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

Note:

1) Until 1.4 version the return types must be same but from 1.5 version onwards covariant return types are allowed.
   According to this Child class method return type need not be same as Parent class method return type its Child types also allowed.

```
class Parent {
        public Object methodOne() {
                return null;
        }
}

class Child extends Parent {
        public String methodOne() {
                return null;
        }
}
```

2) Parent class final methods we can't override in the Child class.
3) Parent class non final methods we can override as final in child class. We can override native methods in the child classes.
4) Private methods are not visible in the Child classes hence overriding concept is not applicable for private methods.
5) We should override Parent class abstract methods in Child classes to provide implementation.
6) While overriding we can't reduce the scope of access modifier.
   Example:

```
class Parent {
        public void methodOne() { }
 }
```

```
class Child extends Parent {
        protected void methodOne( ) { }
}
```

Output:
Compile time error

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 Account{

        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 RBI {
	void getSimpleIntereset(float simpleRate) {

//logic here

	}
}

class Axis extends RBI {

	void getSimpleIntereset(float simpleRate) {

//logic here

	}
}

class HDFC extends RBI {

	void getSimpleIntereset(float simpleRate) {

//logic here

	}
}
```

Live Example-2

```
class Policy {

	void getPremium(Customer customer) {
		//logic here
	}
}
```

```
class TermPolicy extends Policy {

	void getPremium(Customer customer) {
		//logic here

	}
}

class RiderProtection extends TermPolicy {

	void getPremium(Customer customer) {
		//logic here

	}
}
```