Overriding
++++++++++
Whatever the parent has by default avaialbe to the child class through inheritance,if the child is not satisfied with the parent class method implementation then child class is allowed to redifine that parent class method in the Child class in its own way this process is called as "Overriding".

In java Polymorphism is one of the pillar of oops.

We have 2 types of polymorphism
 a. Compiletimebinding/Earlybinding/staticbinding
      eg: Overloading, method-hiding

 b. Runtime/dynamic/latebinding
      eg: Overriding

In case of Overriding JVM will play a vital role of binding the method call to method body, so we say overriding as "RuntimePolymorphism".
In case of Overriding compiler will just check the reference type and see whether the method signature is present in the class or not.

eg#1.

```
class Parent
{
      public void property()
      {
            System.out.println("Cash+Land+Gold...");
      }

      public void marry()
      {
            System.out.println("Relative Girl.....");
      }

}
class Child extends Parent
{
      @Override
      public void marry()
      {
            System.out.println("Some XYZ girl...");
      }
}
public class Test
{
      public static void main(String[] args)
      {
            Parent p = new Parent();
            p.marry();//Method Resolution(JVM) :: Parent Object

            System.out.println();

            Child c = new Child();
            c.marry();//Method Resolution(JVM) :: Child Object

            System.out.println();

            Parent p1 = new Child();
            p1.marry();//Method Resolution(JVM) ::Child Object
```

```
	}
}

Output
Relative Girl.....

Some XYZ girl.....

Some XYZ girl.....

Difference b/w Method Overloading and Overriding
++++++++++++++++++++++++++++++++++++++++++++++
Overloading :: lines of code would be more and compiler will play a major role of
binding the method call based on the reference type.

eg#1.
class Animal
{
	public void eat()
	{
		System.out.println("Animal is eating...");
	}
	public void sleep()
	{
		System.out.println("Animal is sleeping...");
	}
	public void breathe()
	{
		System.out.println("Animal is breathing...");
	}
}
class Tiger extends Animal
{
	//informing compiler about overidden method
	@Override
	public void eat()
	{
		System.out.println("Tiger hunts and eat...");
	}
}
class Deer extends Animal
{
	//informing compiler about overidden method
	@Override
	public void eat()
	{
		System.out.println("Deer will graze and eat...");
	}
}
class Monkey extends Animal
{
	//informing compiler about overidden method
	@Override
	public void eat()
	{
		System.out.println("Monkey steal and eat...");
	}
}
```

```
//Helper class
class Forest
{
	//Method Overloading
	public void allowAnimal(Tiger t)
	{
		t.eat();
		t.sleep();
		t.breathe();
	}
	public void allowAnimal(Deer d)
	{
		d.eat();
		d.sleep();
		d.breathe();
	}
	public void allowAnimal(Monkey m)
	{
		m.eat();
		m.sleep();
		m.breathe();
	}
}
public class Test
{
	public static void main(String[] args)
	{
			Tiger t  = new Tiger();
			Deer d   = new Deer();
			Monkey m = new Monkey();

			Forest f = new Forest();
			f.allowAnimal(t);
			System.out.println();
			f.allowAnimal(d);
			System.out.println();
			f.allowAnimal(m);

	}
}

Output
Tiger hunts and eat...
Animal is sleeping...
Animal is breathing...

Deer will graze and eat...
Animal is sleeping...
Animal is breathing...

Monkey steal and eat...
Animal is sleeping...
Animal is breathing...



Overriding
++++++++++
```

In case of Overriding lines of code would be less, but because of JVM playing a role the actions will be peformed based on the runtime object.

```
class Animal
{
	public void eat()
	{
		System.out.println("Animal is eating...");
	}
	public void sleep()
	{
		System.out.println("Animal is sleeping...");
	}
	public void breathe()
	{
		System.out.println("Animal is breathing...");
	}
}
class Tiger extends Animal
{
	//informing compiler about overidden method
	@Override
	public void eat()
	{
		System.out.println("Tiger hunts and eat...");
	}
}
class Deer extends Animal
{
	//informing compiler about overidden method
	@Override
	public void eat()
	{
		System.out.println("Deer will graze and eat...");
	}
}
class Monkey extends Animal
{
	//informing compiler about overidden method
	@Override
	public void eat()
	{
		System.out.println("Monkey steal and eat...");
	}
}

//Helper class
class Forest
{
	/*
		RunTime Polymphism[1:M]

				= new Tiger();
		      Animal ref = new Deer();
				= new Monkey();
	*/
	public void allowAnimal(Animal ref)
	{
		ref.eat();
```

```
            ref.sleep();
            ref.breathe();

            System.out.println();
      }
}
public class Test
{
      public static void main(String[] args)
      {
                  Tiger t  = new Tiger();
                  Deer d   = new Deer();
                  Monkey m = new Monkey();

                  Forest f = new Forest();

                  f.allowAnimal(t);
                  f.allowAnimal(d);
                  f.allowAnimal(m);

      }
}
```

```
Output
Tiger hunts and eat...
Animal is sleeping...
Animal is breathing...

Deer will graze and eat...
Animal is sleeping...
Animal is breathing...

Monkey steal and eat...
Animal is sleeping...
Animal is breathing...
```

eg#2.
Overloading

```
class Plane
{
      String engine;
      float fuel;
      int wheel;

      public void takeOff()
      {
            System.out.println("Plane tookoff...");
      }
      public void fly()
      {
            System.out.println("Plane is flying...");
      }
      public void land()
      {
            System.out.println("Plane is landing...");
      }
      public void carry()
      {
```

```
            System.out.println("Plane is carrying...");
      }
}

class Passenger extends Plane
{
      @Override
      public void carry()
      {
            System.out.println("Carrying Passengers...");
      }
}
class Cargo extends Plane
{
      @Override
      public void carry()
      {
            System.out.println("Carrying Cargo...");
      }
}

class Fighter extends Plane
{
      @Override
      public void carry()
      {
            System.out.println("Carrying Weapons...");
      }
}

//Helper class
class Airport
{
      //MethodOverloading :FalsePolymrophsim[1:1] => Virtually [1:M]
      public void allowPlane(Cargo c)
      {
            c.takeOff();
            c.carry();
            c.fly();
            c.land();
            System.out.println();
      }

      public void allowPlane(Passenger p)
      {
            p.takeOff();
            p.carry();
            p.fly();
            p.land();
            System.out.println();
      }

      public void allowPlane(Fighter f)
      {
            f.takeOff();
            f.carry();
            f.fly();
            f.land();
            System.out.println();
```

```
	}

}
public class Test
{
	public static void main(String[] args)
	{
		//Creating 3 objects of Plane Type
		Cargo c = new Cargo();
		Passenger p =new Passenger();
		Fighter f = new Fighter();

		//Taking the actions for all the 3 planes
		Airport a = new Airport();
		a.allowPlane(c);
		a.allowPlane(p);
		a.allowPlane(f);
	}
}

Output
Plane tookoff...
Carrying Cargo...
Plane is flying...
Plane is landing...

Plane tookoff...
Carrying Passengers...
Plane is flying...
Plane is landing...

Plane tookoff...
Carrying Weapons...
Plane is flying...
Plane is landing...

Overriding
++++++++++
eg#1.
class Plane
{
	String engine;
	float fuel;
	int wheel;

	public void takeOff()
	{
		System.out.println("Plane tookoff...");
	}
	public void fly()
	{
		System.out.println("Plane is flying...");
	}
	public void land()
	{
		System.out.println("Plane is landing...");
	}
	public void carry()
	{
```

```
            System.out.println("Plane is carrying...");
      }
}

class Passenger extends Plane
{
      @Override
      public void carry()
      {
            System.out.println("Carrying Passengers...");
      }
}
class Cargo extends Plane
{
      @Override
      public void carry()
      {
            System.out.println("Carrying Cargo...");
      }
}

class Fighter extends Plane
{
      @Override
      public void carry()
      {
            System.out.println("Carrying Weapons...");
      }
}

//Helper class
class Airport
{
      /*MethodOverriding :TruePolymrophsim[1:M]
      Runtime polymorphism
                              = new Cargo();
                  Plane ref   = new Passenger();
                              = new Fighter();
      */
      public void allowPlane(Plane ref)
      {
            System.out.println("Object name is :: "+ref.getClass().getName());
            ref.takeOff();
            ref.carry();
            ref.fly();
            ref.land();
            System.out.println();
      }
}
public class Test
{
      public static void main(String[] args)
      {
            //Creating 3 objects of Plane Type
            Cargo c = new Cargo();
            Passenger p =new Passenger();
            Fighter f = new Fighter();

            //Taking the actions for all the 3 planes
```

```
            Airport a = new Airport();
            a.allowPlane(c);
            a.allowPlane(p);
            a.allowPlane(f);
      }
}
Output

Object name is :: Cargo
Plane tookoff...
Carrying Cargo...
Plane is flying...
Plane is landing...

Object name is :: Passenger
Plane tookoff...
Carrying Passengers...
Plane is flying...
Plane is landing...

Object name is :: Fighter
Plane tookoff...
Carrying Weapons...
Plane is flying...
Plane is landing...

Rules of Overriding
++++++++++++++++++
What is method signature in java?
 public void add(int a, int b){

 }
MethodSignature : methodName(paramList...)

Rule1:
 In case of MethodOverriding, method signature should be same in child class while
overriding.
 It is possible to change the return type also if it is of reference
type[Relationship should be "IS-A"(inheritance)].
 If the return type is of primitive type, then we can't change the returntype, if
we try to change it would result in "CE"

eg#1.
class Parent
{
      public Object methodOne(){
            return null;
      }
}
class Child extends Parent
{
      @Override
      public String methodOne(){
            return "sachin";
      }
}
public class Test
{
      public static void main(String[] args)
```

```
	{
		Parent p = new Child();
		System.out.println(p.methodOne());//sachin
	}
}

eg#2.
class Parent
{
	public Number methodOne(){
		return null;
	}
}
class Child extends Parent
{
	@Override
	public Integer methodOne(){
		return 10;
	}
}
public class Test
{
	public static void main(String[] args)
	{
		Parent p = new Child();
		System.out.println(p.methodOne());//10
	}
}

eg#3.
class Parent
{
	public String methodOne(){
		return null;
	}
}
class Child extends Parent
{
	@Override
	public Object methodOne(){ //CE
		return 10;
	}
}
public class Test
{
	public static void main(String[] args)
	{
		Parent p = new Child();
		System.out.println(p.methodOne());
	}
}

eg#4.
class Parent
{
	public int methodOne(){
		return 10;
	}
}
```

```
class Child extends Parent
{
	@Override
	public void methodOne(){
		System.out.println("sachin");
	}
}
public class Test
{
	public static void main(String[] args)
	{
		Parent p = new Child();
		p.methodOne();
	}
}
```

Rule2:
Private methods are not visible in child class, so Overriding them in the child class is not possible.

eg#1.

```
class Parent
{
	private void methodOne(){
		System.out.println("From Parent...");
	}
}
class Child extends Parent
{
	@Override
	private void methodOne(){
		System.out.println("From Child...");
	}
}
```

Output :CE

eg2.

```
class Parent
{
	private void methodOne(){
		System.out.println("From Parent...");
	}
}

class Child extends Parent
{
	private void methodOne(){
		System.out.println("From Child...");
	}
}
```

Even though the above code would run, still the method present in Child class is not overriden method it is a specialized private method under child class.

Note:
final access modifier can be applied to
	a. class	:: These classes won't participate in inheritance.
	b. method	:: These methods implementation can't be changed,but it will be inherited to child class.
	c. variable	:: it would be treated as compile time constants whose value

should not be changed during the execution.

eg#1.

```
class Parent
{
	public final void methodOne(){
		System.out.println("From Parent...");
	}
}
class Child extends Parent
{

}
public class Test
{
	public static void main(String[] args)
	{
		Parent p = new Child();
		p.methodOne();
	}
}
```

Output

```
From Parent...
```

Rule3:

1. Parent class final methods can't be changed to non-final in child class during overriding.
2. Parent class non-final methods can be made as final in child class during overriding.

eg#1.

```
class Parent
{
	public final void methodOne(){
		System.out.println("From Parent...");
	}
}
class Child extends Parent
{
	public void methodOne(){
		System.out.println("From Child...");
	}
}
public class Test
{
	public static void main(String[] args)
	{
		Parent p = new Child();
		p.methodOne();
	}
}
```

eg#2.

```
class Parent
{
	public void methodOne(){
		System.out.println("From Parent...");
```

```
	}
}
class Child extends Parent
{
	@Override
	public final void methodOne(){
		System.out.println("From Child...");
	}
}
public class Test
{
	public static void main(String[] args)
	{
		Parent p = new Child();
		p.methodOne();
	}
}
```

Rule4: In case of Overriding, we can increase the privilege of the access modifier,if we try to decrease it would result in "CompileTimeError".

eg#1.

```
class Parent
{
	void methodOne(){
		System.out.println("From Parent...");
	}
}
class Child extends Parent
{
	@Override
	public void methodOne(){
		System.out.println("From Child...");
	}
}
public class Test
{
	public static void main(String[] args)
	{
		Parent p = new Child();
		p.methodOne();
	}
}
```

Output

```
From Child...
```

eg#2.

```
class Parent
{
	public void methodOne(){
		System.out.println("From Parent...");
	}
}
class Child extends Parent
{
	@Override
	void methodOne(){
		System.out.println("From Child...");
```

```
	}
}
public class Test
{
	public static void main(String[] args)
	{
		Parent p = new Child();
		p.methodOne();
	}
}
```

Output: CE: attempting to assign weaker access privileges; was public

Overriding w.r.t static methods
++++++++++++++++++++++++++++++
1. We can't override static method as non-static

eg#1.

```
class Parent
{
	public static void methodOne(){
		System.out.println("From Parent...");
	}
}
class Child extends Parent
{
	public void methodOne(){
		System.out.println("From Child...");
	}
}
public class Test
{
	public static void main(String[] args)
	{
		Parent p = new Child();
		p.methodOne();
	}
}
```

Output:: CE

Rule2:Non-static method can't be made static in Overriding

eg#1

```
class Parent
{
	public  void methodOne(){
		System.out.println("From Parent...");
	}
}
class Child extends Parent
{
	@Override
	public static void methodOne(){
		System.out.println("From Child...");
	}
}
public class Test
{
```

```
	public static void main(String[] args)
	{
		Parent p = new Child();
		p.methodOne();
	}
}
Output: CE:
```

Rule3: static methods can't be Overriden

```
eg#1.
class Parent
{
	public static void methodOne(){
		System.out.println("From Parent...");
	}
}
class Child extends Parent
{
	@Override
	public static void methodOne(){
		System.out.println("From Child...");
	}
}
public class Test
{
	public static void main(String[] args)
	{
		Parent p = new Child();
		p.methodOne();
	}
}
Output: CE
```

MethodHiding
++++++++++++
In case of static methods, we assume the child class method is overriden,but reality is it is not overriden where as it would "MethodHidden", where compiler will bind the method calls based on reference type.

```
eg#1.
class Parent
{
	public static void methodOne(){
		System.out.println("From Parent...");
	}
}
class Child extends Parent
{
	public static void methodOne(){
		System.out.println("From Child...");
	}
}
public class Test
{
	public static void main(String[] args)
	{
		Parent p = new Child();
```

```
            p.methodOne();//From Parent...
      }
}
```

Difference b/w methodhiding and methodoverloading
++
MethodHiding
      a.both child class and parent class methods should be static.
      b.Method resoultion will be taken care by compiler based on the reference type.
      c.Method hiding is considered as "static binding/early binding".

MethodOverrding
      a.both child class and parent class methods should be non-static.
      b.Method resoultion will be taken care by JVM based on the runtime object.
      c.Method Overriding is considered as "runtime binding/late binding".

eg#1.

```
class Parent
{
      public static void methodOne(){
            System.out.println("From Parent...");
      }
}
class Child extends Parent
{
      public static void methodOne(){
            System.out.println("From Child...");
      }
}
public class Test
{
      public static void main(String[] args)
      {
            Parent p  = new Parent();
            p.methodOne();//From Parent...

            Child c = new Child();
            c.methodOne();//From Child...

            Parent p1 = new Child();
            p1.methodOne();//From Parent...
      }
}
```

Overriding w.r.t var-args method
++++++++++++++++++++++++++++++++
A var-arg method should be overriden as "var-arg" method only. if we try to override with normal method then it would become overloading but not overriding.

eg#1.

```
class Parent
{
      //var-arg method
```

```
	public void methodOne(int... i){
		System.out.println("From Parent...");
	}
}
class Child extends Parent //overloading not overriding
{
	//normal method
	public void methodOne(int i){
		System.out.println("From Child...");
	}
}

public class Test
{
	public static void main(String[] args)
	{
		Parent p = new Parent();
		p.methodOne(10);//From Parent...

		Child c = new Child();
		c.methodOne(10);//From Child...

		Parent p1 = new Child();
		p1.methodOne(10);//From Parent...

	}
}


eg#2.
class Parent
{
	//var-arg method
	public void methodOne(int... i){
		System.out.println("From Parent...");
	}
}
class Child extends Parent // Overriding
{
	@Override
	public void methodOne(int... i){
		System.out.println("From Child...");
	}
}

public class Test
{
	public static void main(String[] args)
	{
		Parent p = new Parent();
		p.methodOne(10);//From Parent...

		Child c = new Child();
		c.methodOne(10);//From Child...

		Parent p1 = new Child();
		p1.methodOne(10);//From Child...

	}
```

```
}

Overriding w.r.t variables
++++++++++++++++++++++++++
=> Overriding is not applicable for variables.
=> Variable resolution is always taken care by compiler based on reference type.

eg#1.
class Parent
{
      int x= 888;
}
class Child extends Parent
{
      int x= 999;
}
public class Test
{
      public static void main(String[] args)
      {
            Parent p = new Parent();
            System.out.println(p.x);//888

            Child c = new Child();
            System.out.println(c.x);//999


            Parent p1 = new Child();
            System.out.println(p1.x);//888

      }
}

Question
class Foo {
      public int a = 3;
      public void addFive() { a += 5; System.out.print("f "); }
}
class Bar extends Foo {
      public int a = 8; //Bar :: a = 13
      public void addFive() { this.a += 5; System.out.print("b " ); }
}
Invoked with:
Foo f = new Bar();
f.addFive(); // b
System.out.println(f.a);//3

What is the result?
A. b 3
B. b 8
C. b 13
D. f 3
E. f 8
F. f 13
G. Compilation fails.
H. An exception is thrown at runtime.


Q>
```

```
class Thingy { Meter m = new Meter(); }
class Component { void go() { System.out.print("c"); } }
class Meter extends Component { void go() { System.out.print("m"); } }
class DeluxeThingy extends Thingy {
      public static void main(String[] args) {
                  DeluxeThingy dt = new DeluxeThingy();
                  dt.m.go();
                  Thingy t = new DeluxeThingy();
                  t.m.go();
      }
}
```

Which two are true? (Choose two.)

A. The output is mm.
B. The output is mc.
C. Component is-a Meter.
D. Component has-a Meter.
E. DeluxeThingy is-a Component.
F. DeluxeThingy has-a Component.

Q>

```
class Foo {
      private int x;
      public Foo( int x ){ this.x = x;}
      public void setX( int x ) { this.x = x; }
      public int getX(){ return x;}
}
public class Gamma {
      static Foo fooBar(Foo foo) {
            foo = new Foo(100);
            return foo;
      }
public static void main(String[] args) {
       Foo foo = new Foo( 300 );
      System.out.println( foo.getX() + "-");

      Foo fooFoo = fooBar(foo);
      System.out.println(foo.getX() + "-");
      System.out.println(fooFoo.getX() + "-");

      foo = fooBar( fooFoo);
      System.out.println( foo.getX() + "-");
      System.out.println(fooFoo.getX());
 }
}
```

What is the output of the program shown in the exhibit?

A. 300-100-100-100-100
B. 300-300-100-100-100
C. 300-300-300-100-100
D. 300-300-300-300-100

instance control flow in parent to child relationship
++

Whenever we are creating an object of child class the following sequence of events will take place

a. Identification of instance variable from Parent to Child.

 b. Execution of instance variable assignments and instance block only in Parent class.
 c. Execution of parent class constructor.

++
 d. Execution of instance variable assignments and instance block only in child class.
 e. Execution of child class constructor.

eg#1.

```
class Parent
{
      int x= 10;
      {
            methodOne();
            System.out.println("Parent fist instance block...");
      }

      Parent()
      {
            System.out.println("Parent class constructor...");
      }
      public static void main(String... args)
      {
            Parent p = new Parent();
            System.out.println("Parent class main method...");
      }
      public void methodOne()
      {
            System.out.println(y);
      }
      int y =20;
}

class Child extends Parent
{
      int i= 100;

      {
            methodTwo();
            System.out.println("Child fist instance block...");
      }

      Child()
      {
            System.out.println("Child class constructor...");
      }
      public static void main(String... args)
      {
            Child  c = new Child();
            System.out.println("Child class main method...");
      }
      public void methodOne()
      {
            System.out.println(j);
      }
      int j =200;
}
```

```
public class Test
{
	public static void main(String[] args)
	{
		Parent p = new Parent();
		System.out.println(p.x);//888

		Child c = new Child();
		System.out.println(c.x);//999


		Parent p1 = new Child();
		System.out.println(p1.x);//888

	}
}
```