Method Overriding in Java

Defining method again with same name,with same parameters but in different classes is called method Overriding.

If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding**.

In other words, If subclass provides the specific implementation of the method that has been provided by one of its parent class, it is known as Method ~~Overriding.Also known as Runtime Polymorphism/Dynamic polymorphism/Late binding.~~

~~Overriding method resolution is also known as "Dynamic Method Dispatch"~~

ADVANTAGE OF JAVA METHOD OVERRIDING

- Method Overriding is used to provide specific implementation of a method that is already provided by its super class.
- Method Overriding is used for Runtime Polymorphism.

RULES FOR METHOD OVERRIDING

1. method must have same name as in the parent class
2. method must have same parameter as in the parent class.
3. ~~must be IS-A relationship (inheritance~~).

UNDERSTANDING THE PROBLEM WITHOUT METHOD OVERRIDING

Let's understand the problem that we may face in the program if we don't use method overriding.

```
class Vehicle{
void run(){System.out.println("Vehicle is running");}
}
class Bike extends Vehicle{

public static void main(String args[]){
Bike obj = new Bike();
obj.run();
```

```
  }
}
```

```
Output: Vehicle is running
```

Problem is that I have to provide a specific implementation of run() method in subclass that is why we use method overriding.

In this example, we have defined the run method in the subclass as defined in the parent class but it has some specific implementation. The name and parameter of the method is same and there is ~~IS-A relationship~~ between the classes, so there is method overriding.

```
class Vehicle{
void run(){System.out.println("Vehicle is running");}
}
class Bike extends Vehicle{
void run(){System.out.println("Bike is running safely");}

public static void main(String args[]){
Bike obj = new Bike();
obj.run();
}
```

```
Output: Bike is running safely
```

REAL EXAMPLE OF JAVA METHOD OVERRIDING

Consider a scenario, Bank is a class that provides functionality to get rate of interest. But, rate of interest varies according to banks. For example, SBI, ICICI and AXIS banks could provide 8%, 7% and 9% rate of interest.

```
class Bank{
int getRateOfInterest(){return 0;}
}

class SBI extends Bank{
int getRateOfInterest(){return 8;}
}

class ICICI extends Bank{
int getRateOfInterest(){return 7;}
}
class AXIS extends Bank{
int getRateOfInterest(){return 9;}
}

class Test{
public static void main(String args[]){
```

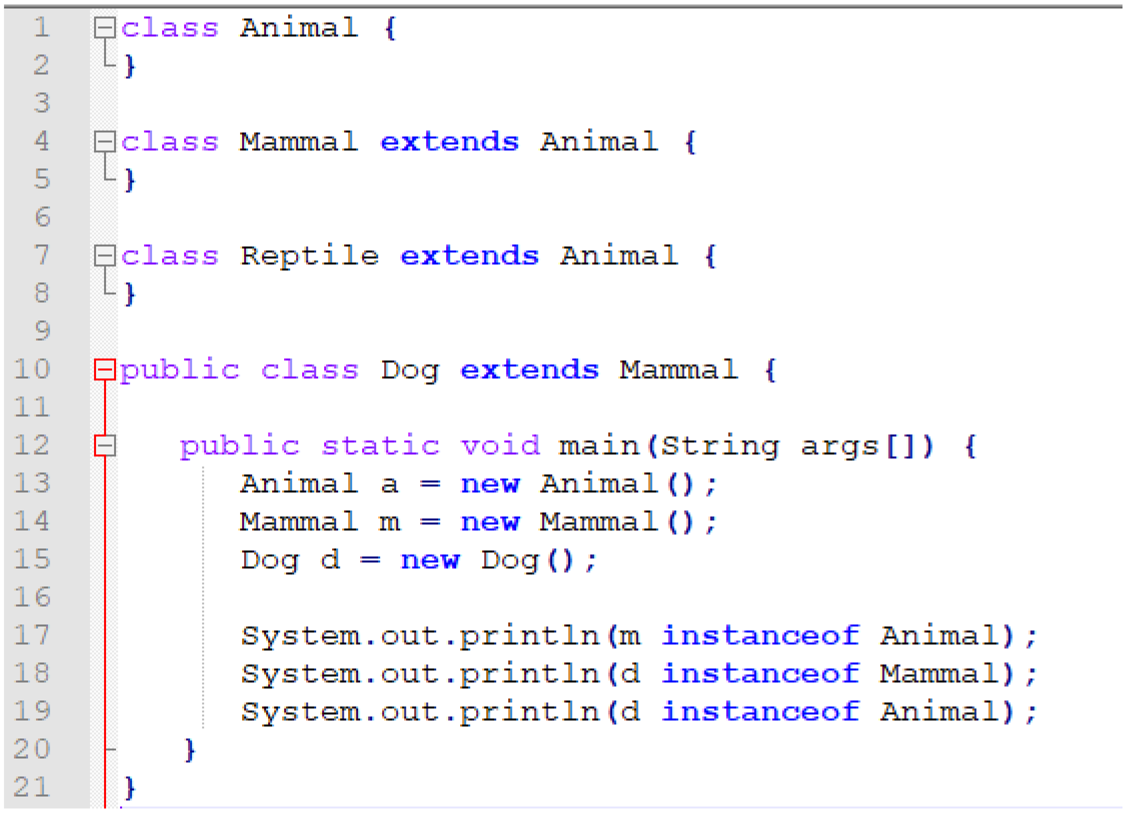
```
SBI s=new SBI();
ICICI i=new ICICI();
AXIS a=new AXIS();
System.out.println("SBI Rate of Interest: "+s.getRateOfInterest());
System.out.println("ICICI Rate of Interest: "+i.getRateOfInterest());
System.out.println("AXIS Rate of Interest: "+a.getRateOfInterest());
}
}
```

```
Output:
SBI Rate of Interest: 8
ICICI Rate of Interest: 7
AXIS Rate of Interest: 9
```

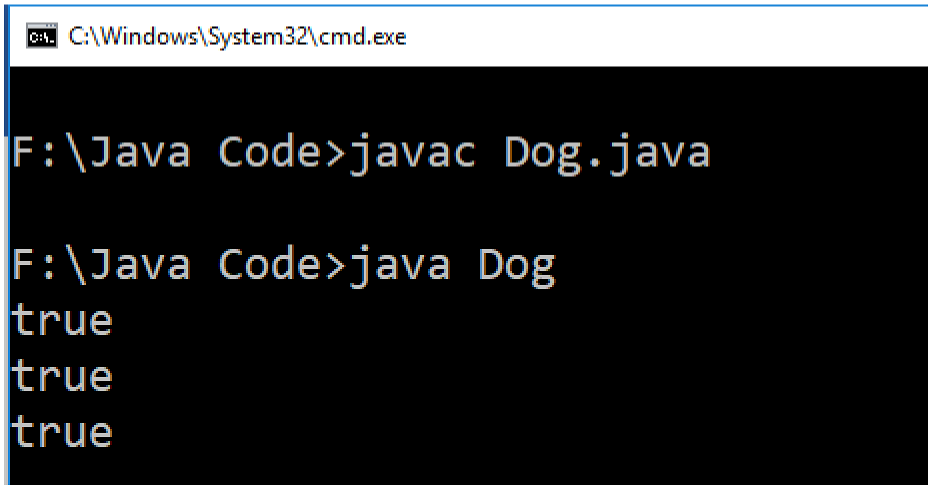
instanceof operator

The **java instanceof operator** is used to test whether the object is an instance of the specified type (class or subclass or interface).

The instanceof in java is also known as type ~~*comparison operator*~~ because it compares the instance with type. ~~It returns either true or false.~~ If we apply the instanceof operator with any variable that has null value, it returns false.



```
class Animal {
}

class Mammal extends Animal {
}

class Reptile extends Animal {
}

public class Dog extends Mammal {

    public static void main(String args[]) {
        Animal a = new Animal();
        Mammal m = new Mammal();
        Dog d = new Dog();

        System.out.println(m instanceof Animal);
        System.out.println(d instanceof Mammal);
        System.out.println(d instanceof Animal);
    }
}
```



C:\Windows\System32\cmd.exe

```
F:\Java Code>javac Dog.java

F:\Java Code>java Dog
true
true
true
```

SUPER KEYWORD

The **super** ~~is a reference variable~~ that is used to refer immediate parent class object. Whenever you create the instance of subclass, an instance of parent class is created implicitly i.e. referred by super reference variable.

USAGE OF SUPER KEYWORD

1. super is used to refer immediate parent class instance variable.
2. super() is used to invoke immediate parent class constructor.
3. super is used to invoke immediate parent class method.

1) SUPER IS USED TO REFER IMMEDIATE PARENT CLASS INSTANCE VARIABLE.

Problem without super keyword

```java
class Vehicle{
  int speed=50;
}

class Bike extends Vehicle{
  int speed=100;

  void display(){
   System.out.println(speed);//will print speed of Bike
  }
  public static void main(String args[]){
   Bike b=new Bike();
   b.display();

}
}
```

```
Output:100
```

In the above example Vehicle and Bike both class have a common property speed. Instance variable of current class is refered by instance by default, but I have to refer parent class instance variable that is why we use super keyword to distinguish between parent class instance variable and current class instance variable.

Solution by super keyword

```java
//example of super keyword

class Vehicle{
  int speed=50;
}

class Bike extends Vehicle{
  int speed=100;

  void display(){
   System.out.println(super.speed);//will print speed of Vehicle now
  }
  public static void main(String args[]){
   Bike b=new Bike();
   b.display();

}
}
```

```
Output:50
```

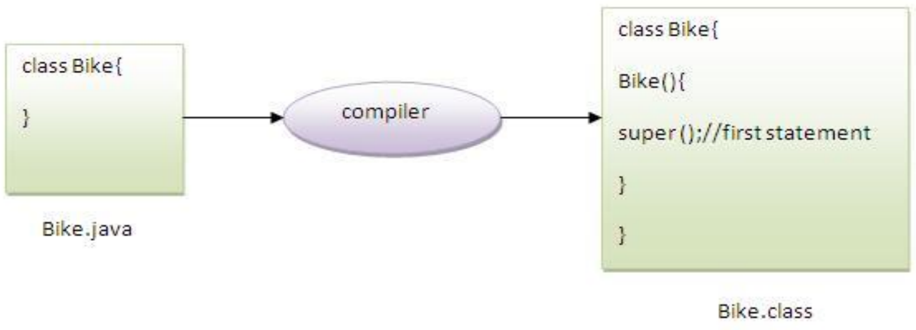
2) SUPER IS USED TO INVOKE PARENT CLASS CONSTRUCTOR.

The super keyword can also be used to invoke the parent class constructor as given below:

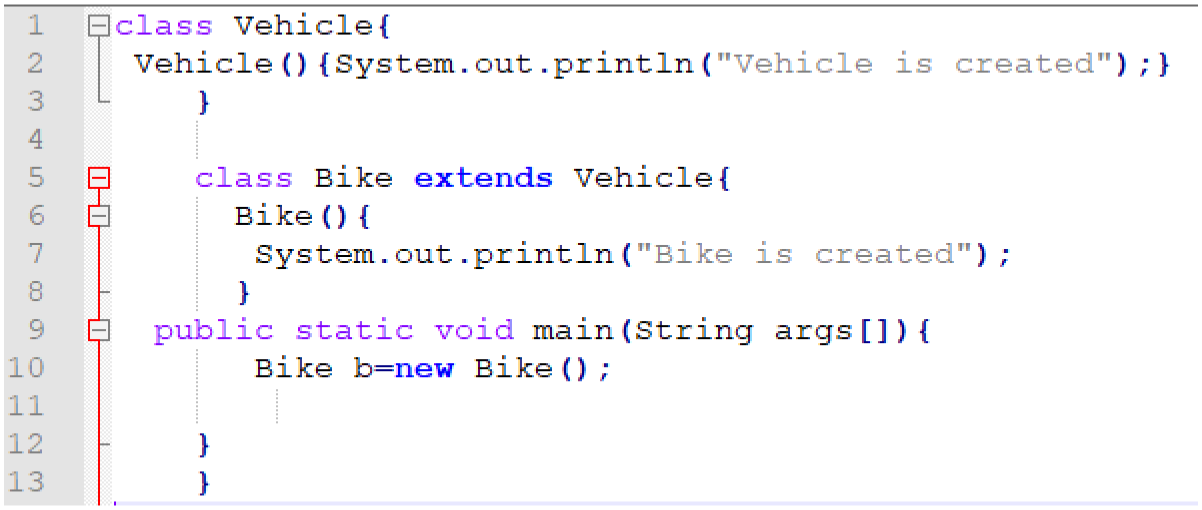
```java
class Vehicle{
  Vehicle(){System.out.println("Vehicle is created");}
}

class Bike extends Vehicle{
  Bike(){
   super();//will invoke parent class constructor
   System.out.println("Bike is created");
  }
  public static void main(String args[]){
   Bike b=new Bike();

}
}
```

```
Output:Vehicle is created
 Bike is created
```

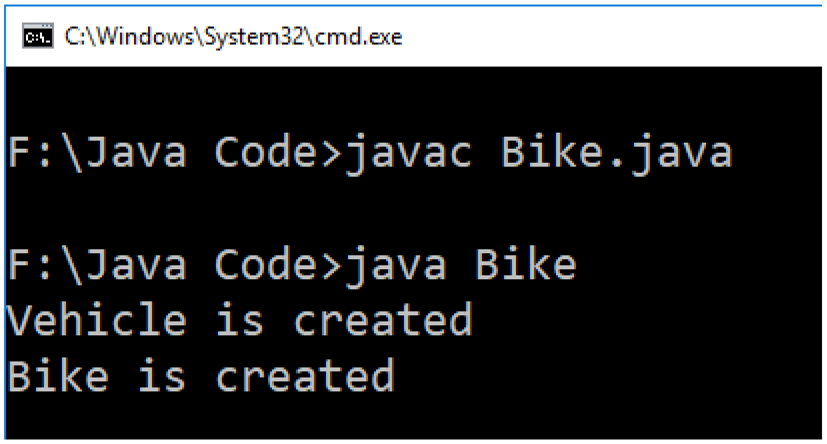
As we know well that default constructor is provided by compiler automatically but it also adds super() for the first statement. If you are creating your own constructor and you don't have either this() or super() as the first statement, compiler will provide super() as the first statement of the constructor.



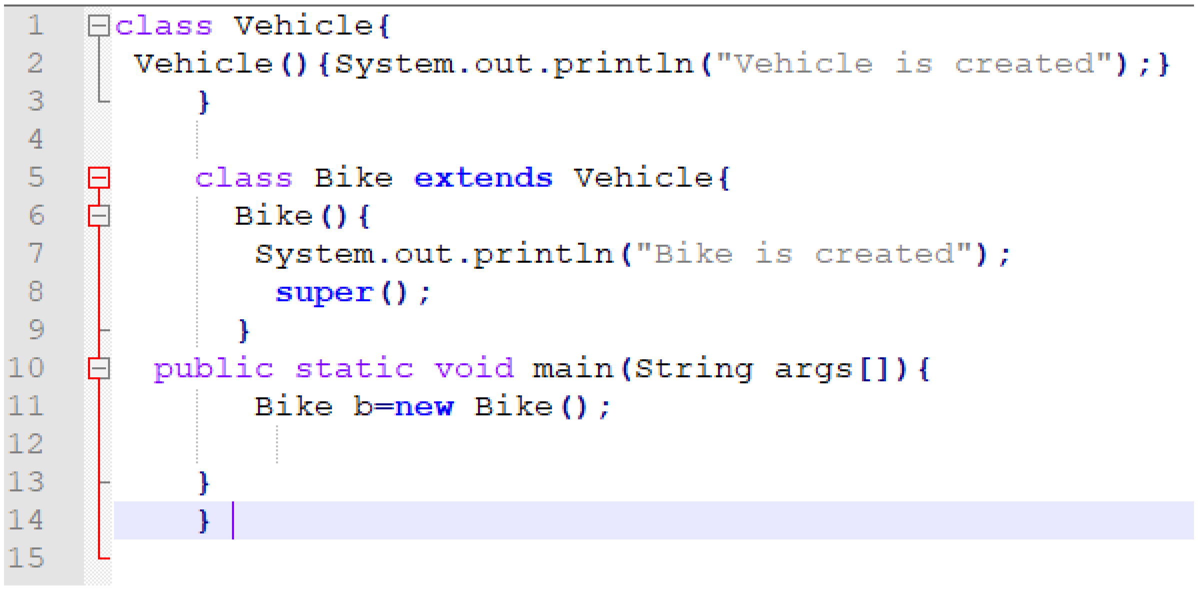
EXAMPLE OF SUPER KEYWORD WHERE SUPER() IS PROVIDED BY THE COMPILER IMPLICITLY.



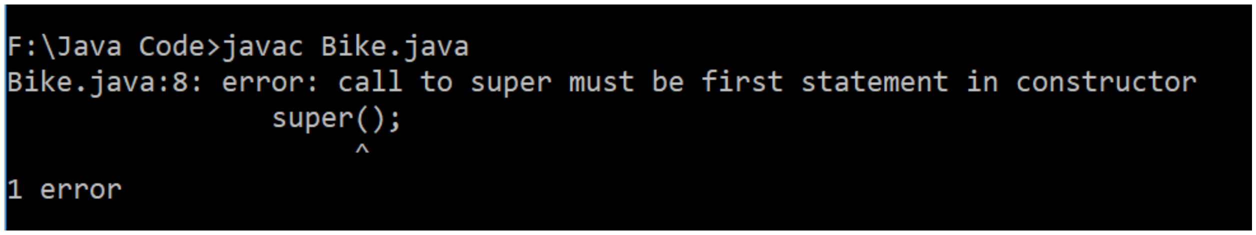
```java
class Vehicle{
 Vehicle(){System.out.println("Vehicle is created");}
    }

    class Bike extends Vehicle{
      Bike(){
       System.out.println("Bike is created");
      }
  public static void main(String args[]){
       Bike b=new Bike();

    }
    }
```



```
C:\Windows\System32\cmd.exe

F:\Java Code>javac Bike.java

F:\Java Code>java Bike
Vehicle is created
Bike is created
```



```java
class Vehicle{
 Vehicle(){System.out.println("Vehicle is created");}
    }

    class Bike extends Vehicle{
      Bike(){
       System.out.println("Bike is created");
        super();
      }
  public static void main(String args[]){
       Bike b=new Bike();

    }
    }
```



```
F:\Java Code>javac Bike.java
Bike.java:8: error: call to super must be first statement in constructor
                super();
                     ^
1 error
```

3) SUPER CAN BE USED TO INVOKE PARENT CLASS METHOD.

The super keyword can also be used to invoke parent class method. It should be used in case subclass contains the same method as parent class as in the example given below:

```java
class Person
{
void message()
{
System.out.println("welcome");
}
}

class Student extends Person
{
void message()//override
{
System.out.println("welcome to java");
}
 void display()
{
message();//will invoke current class message() method
super.message();//will invoke parent class message() method
}

public static void main(String args[])
{
Student s=new Student();
s.display();
}
}
```

```
Output:welcome to java
```

```
welcome
```

In the above example Student and Person both classes have message() method if we call message() method from Student class, it will call the message() method of Student class not of Person class because priority is given to local.

In case there is no method in subclass as parent, there is no need to use super. In the example given below message() method is invoked from Student class but Student class does not have message() method, so you can directly call message() method.

Program in case super is not required

```java
class Person{
void message(){System.out.println("welcome");}
}

class Student extends Person{

void display(){
message();//will invoke parent class message() method
}

public static void main(String args[]){
Student s=new Student();
s.display();
}
}
```

```
Output:welcome
```