Static and non static method Execution with different cases :

Different Cases :	Non static Method	Static Method
Animal a = new Animal(); a.eat();	Compiler will search eat() method in the Animal class and JVM will also execute from Animal class	Compiler will search eat() method in the Animal class and JVM will also execute from Animal class
Dog d = new Dog(); d.eat();	Compiler will search eat() method in the Dog class and JVM will execute from Dog class.	Compiler will search eat() method in the Dog class and JVM will execute from Dog class.
Animal a1 = new Dog(); a1.eat();	Compiler will search eat() method in the Animal class whereas JVM will start executing from Dog class.	Compiler will seatch eat method in the Animal class and JVM will also **execute fom Animal class.** static methods are not overridden

//Programs :

```
package com.ravi.method_overriding;

class Animal
{
   public void eat()
   {
      System.out.println("Generic Animal is eating");
   }
}
class Lion extends Animal
{
   public void eat()
   {
      System.out.println("Lion Animal is eating");
   }
}

class Dog extends Animal
{
   public void eat()
   {
      System.out.println("Dog Animal is eating");
   }
}

public class OverridingDemo1
{
   public static void main(String[] args)
   {
      Animal a1 = null;
      a1 = new Lion();  a1.eat();  //Dynamic Method Dispatch
      a1 = new Dog();   a1.eat();  //Dynamic Method Dispatch
   }

}
```

```
package com.ravi.method_overriding;

class Vehicle
{
   public void run()
   {
      System.out.println("Generic Vehicle is running");
   }
}
class Car extends Vehicle
{
   public void run()
   {
      System.out.println("Car is running");
   }
}

class Audi extends Car
{
   public void run()
   {
      System.out.println("Audi Car is running");
   }
}

public class OverridingDemo2
{
   public static void main(String[] args)
   {
      Vehicle v = new Audi();
      v.run();
   }

}
```

IQ :

```
package com.ravi.method_overriding;

class Person
{
   public int age()
   {
      return 35;
   }

   public void printAge()
   {
      System.out.println(this.age());
   }
}
class Student extends Person
{
   public int age()
   {
      return 22;
   }
}
public class OverridingDemo3
{
   public static void main(String[] args)
   {
      Person p = new Student();
      p.printAge(); // Output is 22
   }
}
```

```
package com.ravi.method_overriding;

class Person
{
   public int age()
   {
      return 35;
   }

   public void printAge()
   {
      System.out.println(this.age());
   }
}
class Student extends Person
{
   public int age()
   {
      return 22;
   }

   public void printAge()
   {
      System.out.println(super.age());
   }
}
public class OverridingDemo3
{
   public static void main(String[] args)
   {
      Person p = new Student();
      p.printAge();
   }
}
```

@Override Annotation :

* Annotation concept is introduced by java from JDK 1.5V.
* All the annotations must be start with @ symbol.
* An annotation defines metadata (data about data) that means It provides the information to Compiler and Developer both that the method is Overridden method.
* If we apply @Override annotation on the sub class Overridden method but If the method is not overridden then we will get Compilation error
* In order to get the appropriate behavior, It is recommended to write @**O**verride annotation before an Overridden method.

```
 package com.ravi.method_overriding;

 class Bird
 {
      public void fly()
      {
         System.out.println("Generic Bird is flying");
      }
 }

 class Parrot extends Bird
 {
      @Override
      public void fly()
      {
         System.out.println("Parrot Bird is flying");
      }
 }
 class Peacock extends Bird
 {
      @Override
      public void fly()
      {
         System.out.println("Peacock Bird is flying");
      }
 }
 public class OverridingDemo4
 {
    public static void main(String[] args)
    {
       Bird b = null;
       b = new Parrot();b.fly();
       b = new Peacock();b.fly();
    }

 }
```