```java
package com.ravi.dynamic_polymorphism;

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

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

class Sparrow extends Bird
{
   @Override
   public void roam()
   {
      System.out.println("Sparrow Bird is Roaming");
   }

   //Overloading
   public int fly(double height)
   {
      System.out.println("Sparrow is flying with "+height+" height");
      return 0;
   }
}
public class OverloadingAndOverriding
{
   public static void main(String[] args)
   {
       Sparrow sparrow = new Sparrow();
       sparrow.roam();
       sparrow.fly(15.6);
   }

}
```

Note : From the above program, It is clear that Overloading is possible in super and sub class.

Variable hiding by using upcasting :

----

* In Java, We have method overriding concept that means we can override non static methods, In java we don't have variable overriding concept.

* In upcasting **variables are always executed from the current reference** regardless of object.

* In java only obejct behavior will change but not the properties.

```java
package com.ravi.dynamic_polymorphism;

class RBI
{
   protected String ifscCode = "RBIHYD000012";

   public String loan()
   {
      return "Bank should provide loan";
   }
}

class HDFC extends RBI
{
   protected String ifscCode = "HDFCAMEERPET01009";  //Variable Hiding

   @Override
   public String loan()
   {
      return "HDFC provides loan @ 9.2% ROI";
   }
}
public class VariableHiding
{
   public static void main(String[] args)
   {
      RBI rbi = new HDFC();
      System.out.println(rbi.ifscCode +"  : "+rbi.loan());

   }

}
```

Output : RBIHYD000012   HDFC provides loan @ 9.2% ROI

Can we override private methods ?

No, We can't override private method of super class because private methods are not visible (not available) to the sub class hence we can't override.

We can't use @Override annotation on private method of sub class because it is not overridden method, actually it is re-declared by sub class developer.

```java
package com.ravi.dynamic_polymorphism;

class Alpha
{
   private void show()
   {
      System.out.println("Private show method of Alpha class");
   }
}

class Beta extends Alpha
{
   public void show() //Re-declaration
   {
      System.out.println(" show method of Beta class");
   }
}
public class OverridingPrivateMethod
{
   public static void main(String[] args)
   {
      new Beta().show();
   }
}
```

Role of Access Modifier while Overriding a Method :

----

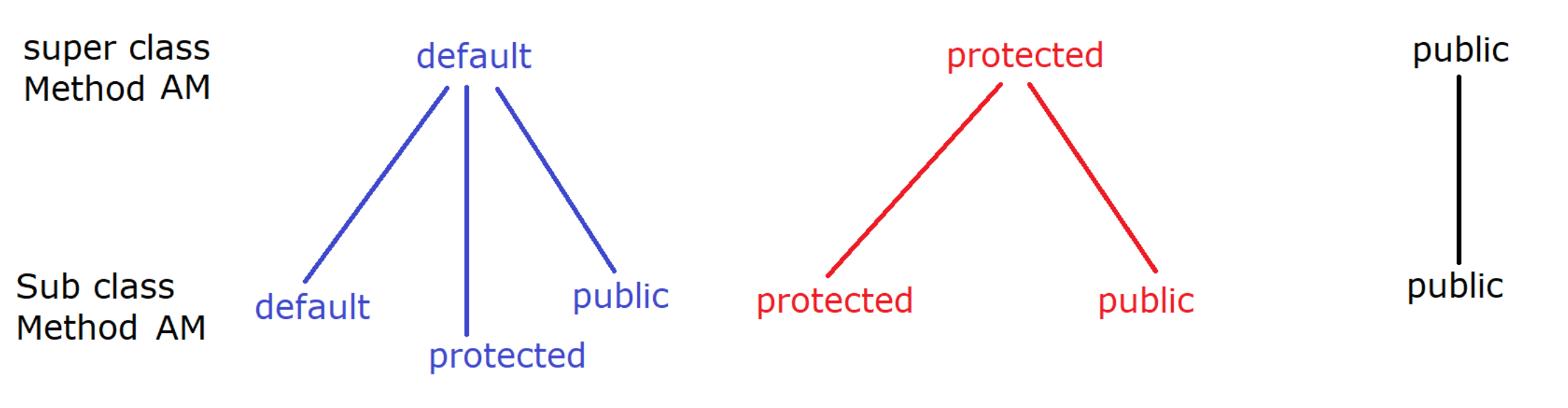
* We have 4 access modifiers in java i.e

   1) private   2) default   3) protected 4) public

* When we override a method from super class then at the time of overriding **we cannot reduce the visibility of the overridden method** otherwise we will get compilation error.

* In terms of **accessibility** public > protected > default > private that means public has highest accessibility than other access modifier.

* At the time of overriding, sub class method access modifier must be **greater OR equal in comparison to access modifier of super class method.**




```java
package com.ravi.dynamic_polymorphism;

class Super
{
   public void show()
   {
      System.out.println("Show method of Super class");
   }
}
class Sub extends Super
{
   protected void show() //error [cannot reduce the visibility]
   {
      System.out.println("Show method of Sub class");
   }
}

public class VisibilityDemo
{
   public static void main(String[] args)
   {
      Super s1 = new Sub();
      s1.show();

   }

}
```

```java
package com.ravi.dynamic_polymorphism;

class Employee
{
   public void restaurant()
   {
   }
}
class Manager extends Employee
{
   private void restaurant()
   {
   }
}

public class VisibilityDemo1
{
   public static void main(String[] args)
    {


   }

}
```

Polymorphic behavior of object by using loose coupling concept :

```java
package com.ravi.loose_coupling;

class Animal
{
   public void roam()
   {
      System.out.println("Generic Animal is roaming");
   }
}
class Dog extends Animal
{
   public void roam()
   {
      System.out.println("Dog Animal is roaming");
   }
}
class Lion extends Animal
{
   public void roam()
   {
      System.out.println("Lion Animal is roaming");
   }
}

public class PlymorphicBehaviorDemo1
{
   public static void main(String[] args)
   {
      Animal a1 = new Dog();
      checkAnimal(a1);

      a1 = new Lion();
      checkAnimal(a1);
   }

   public static void checkAnimal(Animal animal)
   {
      animal.roam();
   }
}
```

How to call specific method of Lion and Dog class
