How to call specific method of Dog and Lion class :
----

* If we want to call specific method of Dog and Lion class then we need the following two things :

  1) In order to satisfy java compiler we need Dog class OR Lion class Reference
  2) In order to call the appropriate class method we need Lion and Dog object

* So, the conclusion is, We need downcasting

```java
package com.ravi.loose_coupling;

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


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

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

   public void roar()
   {
      System.out.println("Lion is roaring");
   }
}

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

   public static void checkAnimal(Animal animal)
   {
      Dog d1 = (Dog) animal;  //down-casting
      d1.roam();
      d1.bark();
   }
}
```

Note : In the above program we have applied downcasting to call Dog class specific method (bark()) but here we have one limitation we cannot pass Lion or any sub class object of Animal otherwise we will get java.lang.ClassCastException.

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

         a1 = new Lion();               Here If we pass Lion Object then Lion
         checkAnimal(a1);               object cannot be converted to Dog object
      }                                        so ClassCastException.

      public static void checkAnimal(Animal animal)
      {
         Dog d1 = (Dog) animal;  //down-casting
         d1.roam();
         d1.bark();
      }
```

---

instanceof operator :
----
* It is a relational operator so it will return true OR false.

* It is also a keyword in java.

* It is used to verify whether a reference variable is pointing to a particular type of object or not, The type can be a class OR interface.

* In between reference variable and class OR interface type, We must have IS-A relation otherwise we will get compilation error.

```java
package com.ravi.instanceof_demo;

class Alpha
{
}
class Beta extends Alpha
{
}
class Gamma extends Beta
{
}

public class InstanceofDemo1
{
    public static void main(String[] args)
    {
        Gamma g = new Gamma();

        if(g instanceof Gamma)
        {
          System.out.println("g is pointing to Gamma Object");
        }

        if(g instanceof Beta)
        {
          System.out.println("g is pointing to Beta Object");
        }

        if(g instanceof Alpha)
        {
          System.out.println("g is pointing to Alpha Object");
        }

        if(g instanceof Object)
        {
          System.out.println("g is pointing to Object class Object");
        }

    }

}
```

```java
package com.ravi.instanceof_demo;

public class InstanceDemo2
{
    public static void main(String[] args)
    {
        Integer i = 90;

        if(i instanceof Number)
        {
            System.out.println("i is pointing to Integer object");
        }

    }

}
```

```java
package com.ravi.instanceof_demo;

class Bird
{
   public void fly()
   {

   }
}

class Fish
{
   public void swim()
   {

   }
}


public class InstanceDemo3
{
   public static void main(String[] args)
   {
      Fish fish = new Fish();

      if(fish instanceof Bird)   //error becoz there is no IS-A relation between
      {                                                            Bird and Fish
      }
   }

}
```

```java
package com.ravi.instanceof_demo;


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

class BMW extends Vehicle
{

}

class Audi extends Vehicle
{

}



public class InstanceDemo4
{

   public static void main(String[] args)
   {

      //BMW b = new BMW();
      //acceptCarType(b);

      Audi a = new Audi();
      acceptCarType(a);
   }

   public static void acceptCarType(Vehicle v)
   {

      if(v instanceof BMW)
      {
         System.out.println("BMW Car");
      }
      else
      {
         System.out.println("AUDI Car");
      }

   }

}
```

```java
package com.ravi.loose_coupling;

class Payment
{
   public void makePayment(double payment)
   {
      System.out.println("Generic Payment");
   }
}
class UPI extends Payment
{
   @Override
   public void makePayment(double payment)
   {
      System.out.println("Making a payment of :"+payment+ " from UPI");
   }

   public void offer()
   {
      System.out.println("Make a payment through UPI and get 100 RS 
cashback");
   }

}

class CreditCard extends Payment
{
   @Override
   public void makePayment(double payment)
   {
      System.out.println("Making a payment of :"+payment+ " from CreditCard");

   }

   public void offer()
   {
      System.out.println("Make a payment through credit card and get 2 days 
Holdiday in GOA");
   }
}


class DebitCard extends Payment
{
   @Override
   public void makePayment(double payment)
   {
      System.out.println("Making a payment of :"+payment+ " from DebitCard");

   }

   public void offer()
   {
      System.out.println("Make a payment through debit card and convert your 
EMI's into NO cost EMI");
   }
}



public class PaymentProcessing
{
   public static void main(String[] args)
   {
      Payment p1 = new UPI();
      paymentProcessing(p1);

      p1 = new CreditCard();
      paymentProcessing(p1);

      p1 = new DebitCard();
      paymentProcessing(p1);

   }


   public static void paymentProcessing(Payment payment) //payment = CC obj
   {
      if(payment instanceof UPI)
      {
         UPI upi = (UPI) payment;
         upi.makePayment(15000);
         upi.offer();
      }

      else if(payment instanceof CreditCard)
      {
         CreditCard cc = (CreditCard) payment;
         cc.makePayment(20000);
         cc.offer();
      }

      else if(payment instanceof DebitCard card)
      {
         card.makePayment(25000);
         card.offer();
      }

   }

}
```

Same program we want to re-write using var args :

```java
package com.ravi.loose_coupling;

class Payment
{
   public void makePayment(double payment)
   {
      System.out.println("Generic Payment");
   }
}
class UPI extends Payment
{
   @Override
   public void makePayment(double payment)
   {
      System.out.println("Making a payment of :"+payment+ " from UPI");
   }

   public void offer()
   {
      System.out.println("Make a payment through UPI and get 100 RS cashback");
   }

}

class CreditCard extends Payment
{
   @Override
   public void makePayment(double payment)
   {
      System.out.println("Making a payment of :"+payment+ " from CreditCard");
   }

   public void offer()
   {
      System.out.println("Make a payment through credit card and get 2 days Holdiday 
in GOA");
   }
}


class DebitCard extends Payment
{
   @Override
   public void makePayment(double payment)
   {
      System.out.println("Making a payment of :"+payment+ " from DebitCard");
   }

   public void offer()
   {
      System.out.println("Make a payment through debit card and convert your EMI's 
into NO cost EMI");
   }
}



public class PaymentProcessing
{
   public static void main(String[] args)
   {
      UPI upi = new UPI();
      CreditCard creditcard = new CreditCard();
      DebitCard debitcard = new DebitCard();

      paymentProcessing(upi, creditcard, debitcard);
   }

   public static void paymentProcessing(Payment ...payment)
   {

       for(Payment p : payment)
       {
           if(p instanceof UPI u)
           {
              u.makePayment(15000);
              u.offer();
           }

           else if(p instanceof CreditCard cc)
           {
               cc.makePayment(20000);
              cc.offer();
           }

           else if(p instanceof DebitCard dc)
           {
               dc.makePayment(20000);
              dc.offer();
           }
       }


   }

}
```

*** What is Method Hiding ?
            OR
     Can we override static method ?
           OR
     Can we override main method ?

* We **cannot override static method because static method belongs to class** but not as a part of object.

* When we try to override static method by writing same signature and compaitable return type then It **is not overriding**, **Actually It is method Hiding**.

* In order to learn method hiding we have different cases :

Case 1:

The following program explains, Method Hiding is possible with static method in super and sub class.

```java
class Super
{
    public static void show()
    {
    }
}
class Sub extends Super
{
    public static int show()  //Method Hiding
    {
        return 0;
    }
}
public class Demo
{
    public static void main(String[] args)
    {
        Super s = new Sub();
        s.show();
    }
}
```

Error Message : show() in sub cannot **hide** show() in super because return type **int** is not compaitable with **void**