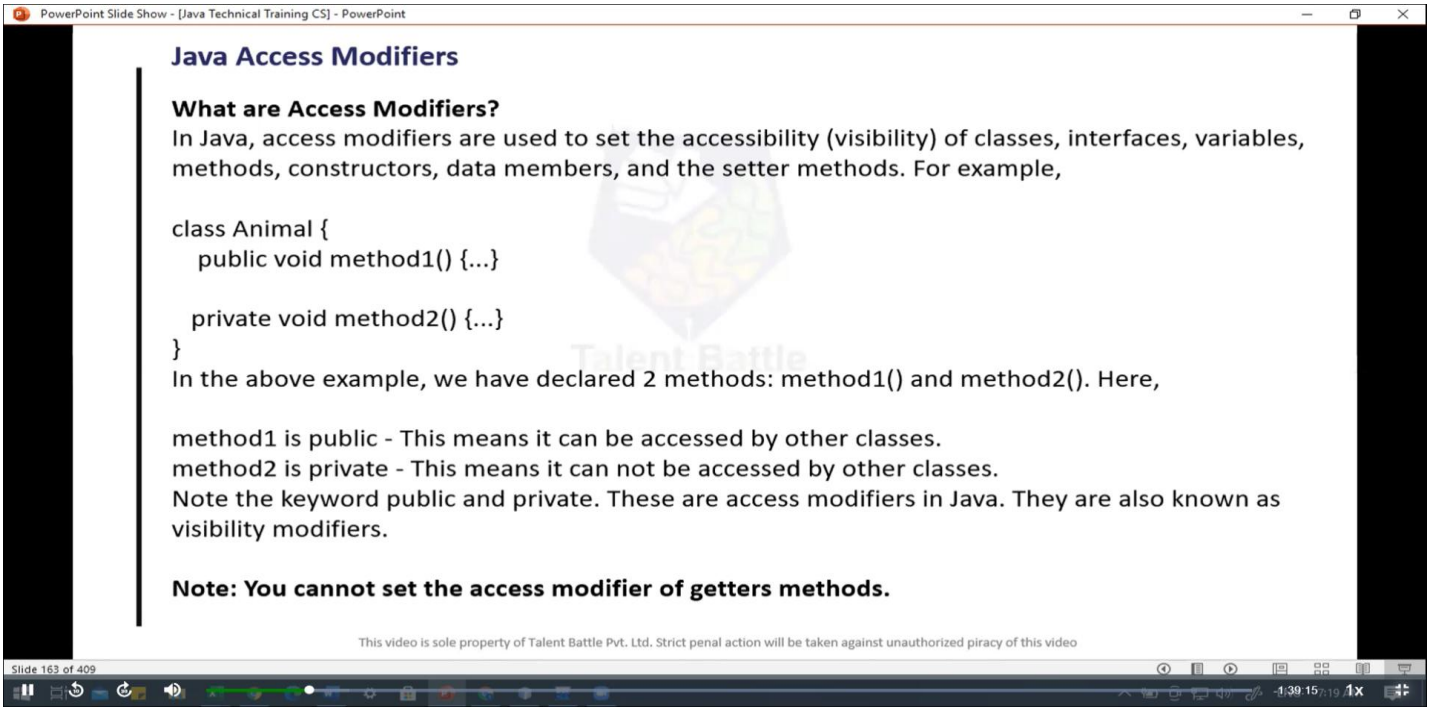
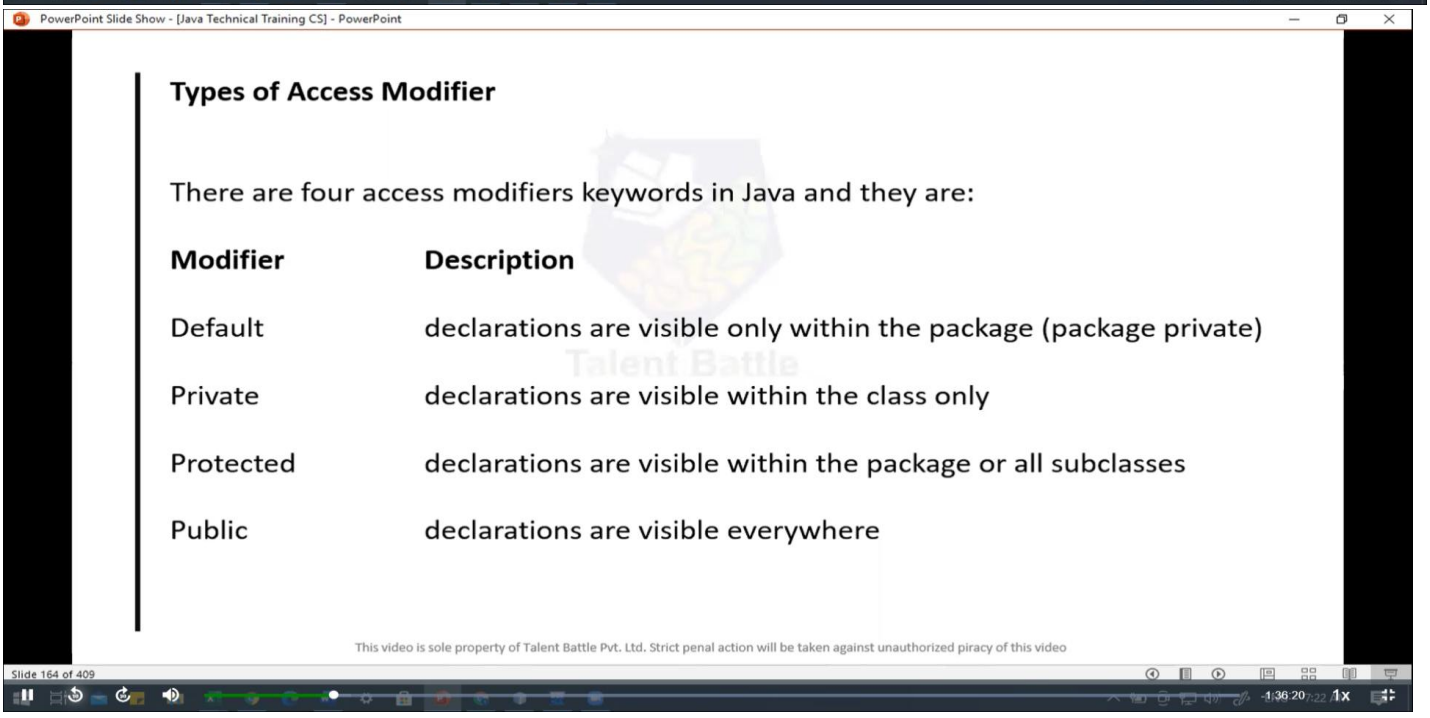
Day 6

Java Access, Modifiers. Recursion Inheritance





PowerPoint Slide Show - [Java Technical Training CS] - PowerPoint

Default Access Modifier

If we do not explicitly specify any access modifier for classes, methods, variables, etc, then by default the default access modifier is considered. For example,

```
package defaultPackage;
class Logger {
   void message(){
      System.out.println("This is a message");
   }
}
```

Here, the Logger class has the default access modifier. And the class is visible to all the classes that belong to the defaultPackage package. However, if we try to use the Logger class in another class outside of defaultPackage, we will get a compilation error.

-1:35:38 1x

PowerPoint Slide Show - [Java Technical Training CS] - PowerPoint

Private Access Modifier

When variables and methods are declared private, they cannot be accessed outside of the class. For example,

```
class Data {
   // private variable
   private String name;
}
public class Main {
   public static void main(String[] main){
      // create an object of Data
      Data d = new Data();
      // access private variable and field from another class
      d.name = "Programming";
   }
}
```

In the above example, we have declared a private variable named name and a private method named display().
The error is generated because we are trying to access the private variable and the private method of the Data class from the Main class.

-1:34:45 1x

PowerPoint Slide Show - [Java Technical Training CS] - PowerPoint

```
class Data {
    private String name;
    // getter method
    public String getName() {
        return this.name;
    }
    // setter method
    public void setName(String name) {
        this.name= name;
    }
}
public class Main {
    public static void main(String[] main){
        Data d = new Data();
        // access the private variable using the getter and setter
        d.setName("Programming");
        System.out.println(d.getName());
    }
}
```

Note: We cannot declare classes and interfaces private in Java.

-1:30:40 1x

PowerPoint Slide Show - [Java Technical Training CS] - PowerPoint

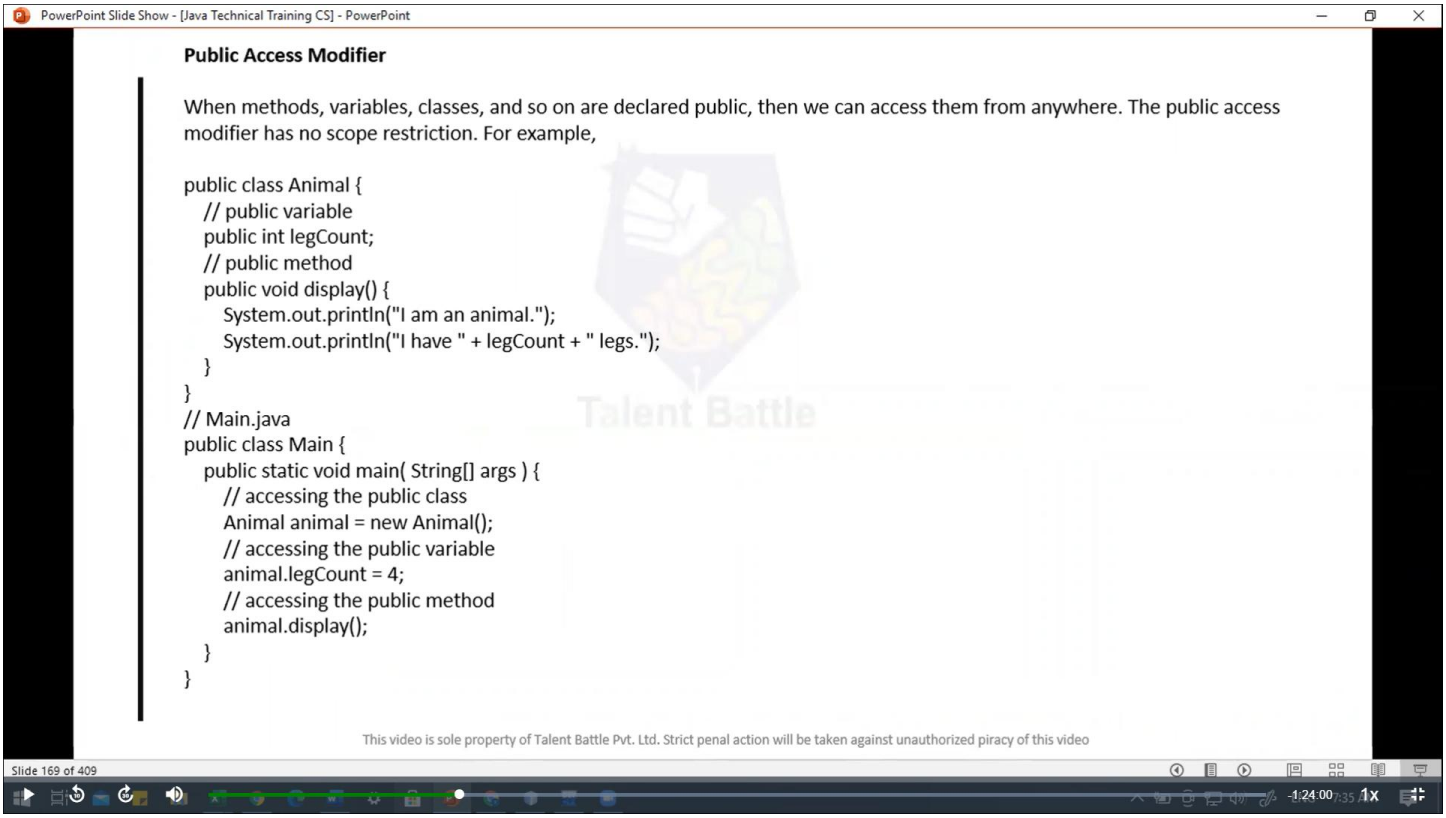
Protected Access Modifier

When methods and data members are declared protected, we can access them within the same package as well as from subclasses. For example,

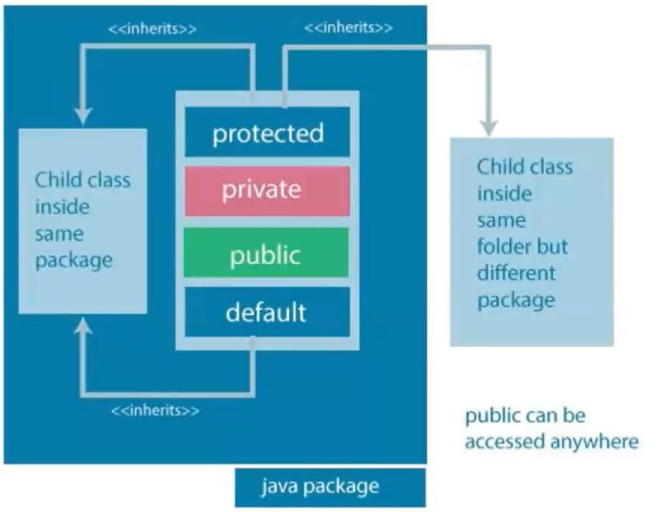
```
class Animal {
    // protected method
    protected void display() {
        System.out.println("I am an animal");
    }
}
class Dog extends Animal {
    public static void main(String[] args) {
        // create an object of Dog class
        Dog dog = new Dog();
         // access protected method
        dog.display();
    }
}
```

Note: We cannot declare classes or interfaces protected in Java.

-1:26:57 1x


PowerPoint Slide Show - [Java Technical Training CS] - PowerPoint
Public Access Modifier
When methods, variables, classes, and so on are declared public, then we can access them from anywhere. The public access modifier has no scope restriction. For example,
public class Animal {
// public variable
public int legCount;
// public method
public void display() {
System.out.println("I am an animal.");
System.out.println("I have " + legCount + " legs.");
}
}
// Main.java
public class Main {
public static void main(String[] args) {
// accessing the public class
Animal animal = new Animal();
// accessing the public variable
animal.legCount = 4;
// accessing the public method
animal.display();
}
}
Talent Battle
This video is sole property of Talent Battle Pvt. Ltd. Strict penal action will be taken against unauthorized piracy of this video
Slide 169 of 409
-1:24:00
1x

PowerPoint Slide Show - [Java Technical Training CS] - PowerPoint



public can be accessed anywhere

source folder

Access modifiers are mainly used for encapsulation. It can help us to control what part of a program can access the members of a class. So that misuse of data can be prevented.

-1:22:28 1x

PowerPoint Slide Show - [Java Technical Training CS] - PowerPoint

Java this Keyword

this Keyword

In Java, this keyword is used to refer to the current object inside a method or a constructor. For example,

```
class Main {
   int instVar;
   Main(int instVar){
      this.instVar = instVar;
      System.out.println("this reference = " + this);
   }
   public static void main(String[] args) {
      Main obj = new Main(8);
      System.out.println("object reference = " + obj);
   }
}
```

this is nothing but the reference to the current object.

ENG 7:38 AM

Using this for Ambiguity Variable Names

In Java, it is not allowed to declare two or more variables having the same name inside a scope (class scope or method scope). However, instance variables and parameters may have the same name. For example,

```
class MyClass {
   // instance variable
   int age;

   // parameter
   MyClass(int age){
      age = age;
   }
}
```

In the above program, the instance variable and the parameter have the same name: age. Here, the Java compiler is confused due to name ambiguity.

PowerPoint Slide Show - [Java Technical Training CS] - PowerPoint

Using this for Ambiguity Variable Names

In Java, it is not allowed to declare two or more variables having the same name inside a scope (class scope or method scope). However, instance variables and parameters may have the same name. For example,

```
class MyClass {
   // instance variable
   int age;

   // parameter
   MyClass(int age){
      age = age;
   }
}
```

In the above program, the instance variable and the parameter have the same name: age. Here, the Java compiler is confused due to name ambiguity.

This video is sole property of Talent Battle Pvt. Ltd. Strict penal action will be taken against unauthorized piracy of this video

Slide 172 of 409

First, let's see an example without using this keyword:

```
class Main {

  int age;
  Main(int age){
    age = age;
  }

  public static void main(String[] args) {
    Main obj = new Main(8);
    System.out.println("obj.age = " + obj.age);
  }
}
```

Output:

```
mc.age = 0
```


Now, let's rewrite the code using **this** keyword.

```
class Main {

  int age;
  Main(int age){
    this.age = age;
  }

  public static void main(String[] args) {
    Main obj = new Main(8);
    System.out.println("obj.age = " + obj.age);
  }
}
```

Output:

```
obj.age = 8
```

Now, we are getting the expected output. It is because when the constructor is called, this inside the constructor is replaced by the object obj that has called the constructor. Hence the age variable is assigned value 8.

this with Getters and Setters

```
class Main {
  String name;

  // setter method
  void setName( String name ) {
     this.name = name;
  }
  // getter method
  String getName(){
     return this.name;
  }
  public static void main( String[] args ) {
     Main obj = new Main();
     // calling the setter and the getter method
     obj.setName("Toshiba");
     System.out.println("obj.name: "+obj.getName());
  }
}
```

Here, we have used this keyword:

- to assign value inside the setter method
- to access value inside the getter method

Using this in Constructor Overloading

While working with constructor overloading, we might have to invoke one constructor from another constructor.

In such a case, we cannot call the constructor explicitly. Instead, we have to use this keyword.

Here, we use a different form of this keyword. That is, **this()**.

PowerPoint Slide Show - [Java Technical Training CS] - PowerPoint

Java final keyword

In Java, the final keyword is used to denote constants. It can be used with variables, methods, and classes.

Once any entity (variable, method or class) is declared final, it can be assigned only once. That is,

- the final variable cannot be reinitialized with another value
- the final method cannot be overridden
- the final class cannot be extended

PowerPoint Slide Show - [Java Technical Training CS] - PowerPoint

In Java, we cannot change the value of a final variable. For example,

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

    // create a final variable
    final int AGE = 32;

    // try to change the final variable
    AGE = 45;
    System.out.println("Age: " + AGE);
  }
}
```

Note: It is recommended to use uppercase to declare final variables in Java.

Java final Method

In Java, the final method cannot be overridden by the child class. For example,

```
class FinalDemo {
  // create a final method
  public final void display() {
    System.out.println("This is a final method.");
  }
}
class Main extends FinalDemo {
 // try to override final method
 public final void display() {
  System.out.println("The final method is overridden.");
 }
 public static void main(String[] args) {
  Main obj = new Main();
  obj.display();
 }
}
```


Java final Class

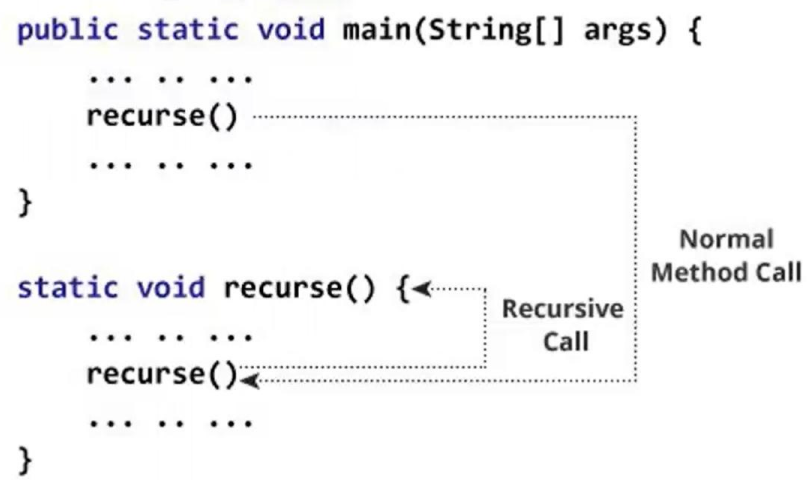
In Java, the final class cannot be inherited by another class. For example,

```
// create a final class
final class FinalClass {
  public void display() {
    System.out.println("This is a final method.");
  }
}
// try to extend the final class
class Main extends FinalClass {
  public  void display() {
    System.out.println("The final method is overridden.");
  }
  public static void main(String[] args) {
    Main obj = new Main();
    obj.display();
  }
}
```

PowerPoint Slide Show - [Java Technical Training CS] - PowerPoint

Java Recursion

In Java, a method that calls itself is known as a recursive method. And, this process is known as recursion.



48:58 1x

PowerPoint Slide Show - [Java Technical Training CS] - PowerPoint

Example: Factorial of a Number Using Recursion

```
class Factorial {

   static int factorial( int n ) {
      if (n != 0)  // termination condition
         return n * factorial(n-1); // recursive call
      else
         return 1;
   }

   public static void main(String[] args) {
      int number = 4, result;
      result = factorial(number);
      System.out.println(number + " factorial = " + result);
   }
}
```

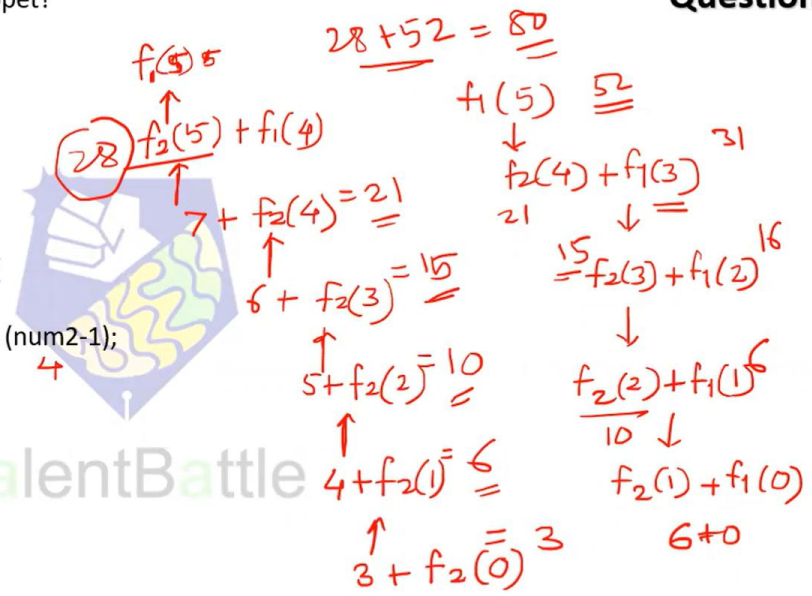

46:16 1x

Question 1

What is the output of the following code snippet?

```
class Demo{
public static int specialAdd(int num1) {
if (num1!=0)
return (num1+2)+specialAdd(num1-1);
else
return 3;
}
public static int extraordinaryAdd(int num2) {
if (num2!=0)
return specialAdd (num2) +extraordinaryAdd (num2-1);
else
return 0;
}
public static void main(String[] args) {
System.out.println( (extraordinaryAdd(5)));
}
}
```

~~A)~~ 80
B) 52
C) 70
D) 25

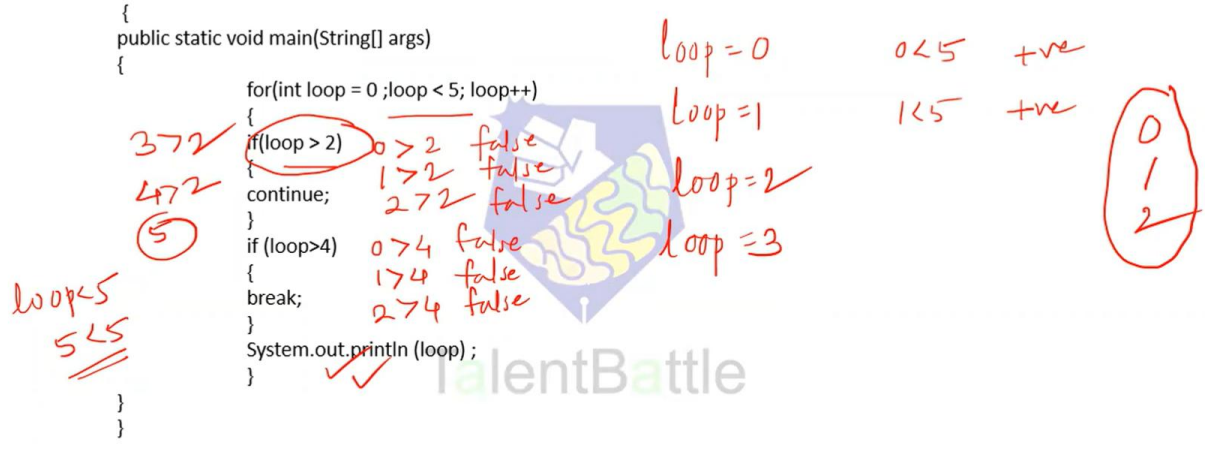


PowerPoint Slide Show - [Sample Paper - Java Part 1] - PowerPoint

Question 1

What is the output of the following code snippet?

```
public class Tester
{
public static void main(String[] args)
{
        for(int loop = 0 ;loop < 5; loop++)
        {
        if(loop > 2)
        {
        continue;
        }
        if (loop>4)
        {
        break;
        }
        System.out.println (loop) ;
        }
}
}
```



A) 0 1 2
B) 0
C) 0 1 3 4
D) 0 1

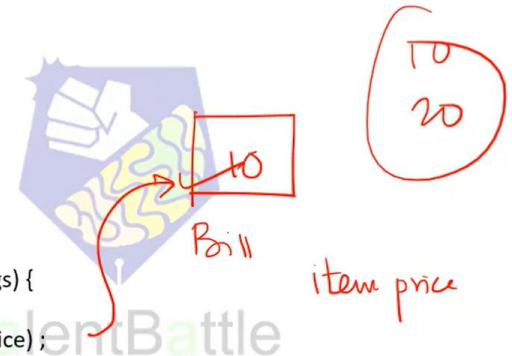
Slide 2 of 8

PowerPoint Slide Show - [Sample Paper - Java Part 1] - PowerPoint

Question 2

Whit is the output of following code snippet?

```
class Bill{
        int itemPrice;
        public Bill(int  itemPrice) {
        this.itemPrice = itemPrice;
    }
        void display () {
        int  itemPrice = 20;
        System.out.println(itemPrice) ;
        }
}
class Demo {
        public static void main(String[] args) {
        Bill billObj = new Bill(10);
        System.out.println(billObj .itemPrice) ;
        billObj.display();
        }
}
```



A) 10
0

B) 10
20

C) 10
10

D) Error in the class as there is no default constructor defined

TalentBattle

Slide 3 of 8

-28:42 1x

PowerPoint Slide Show - [Java Technical Training CS] - PowerPoint

Java instanceof Operator

The **instanceof** operator in Java is used to check whether an object is an instance of a particular class or not.

Its syntax is

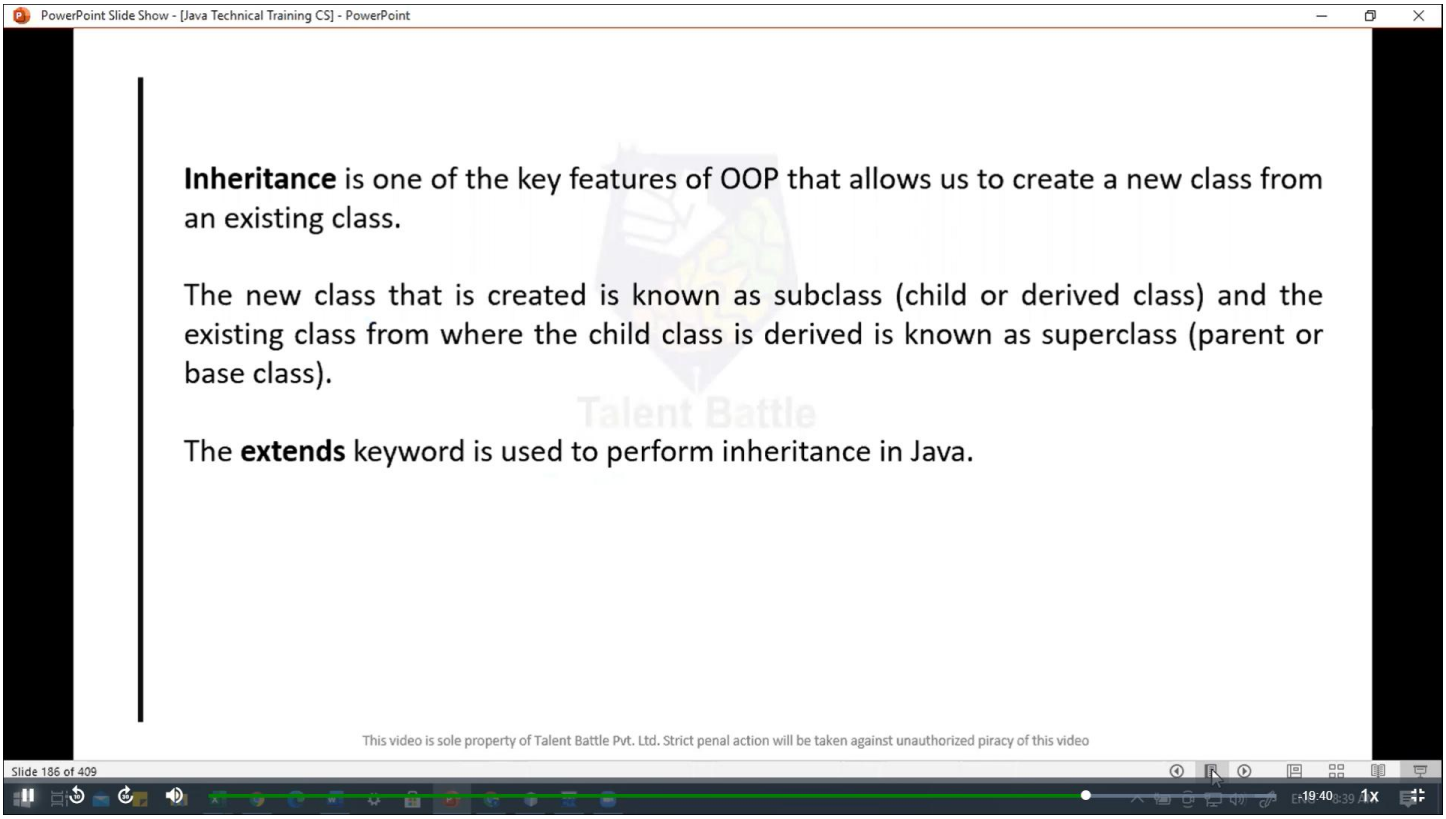
objectName instanceOf className;

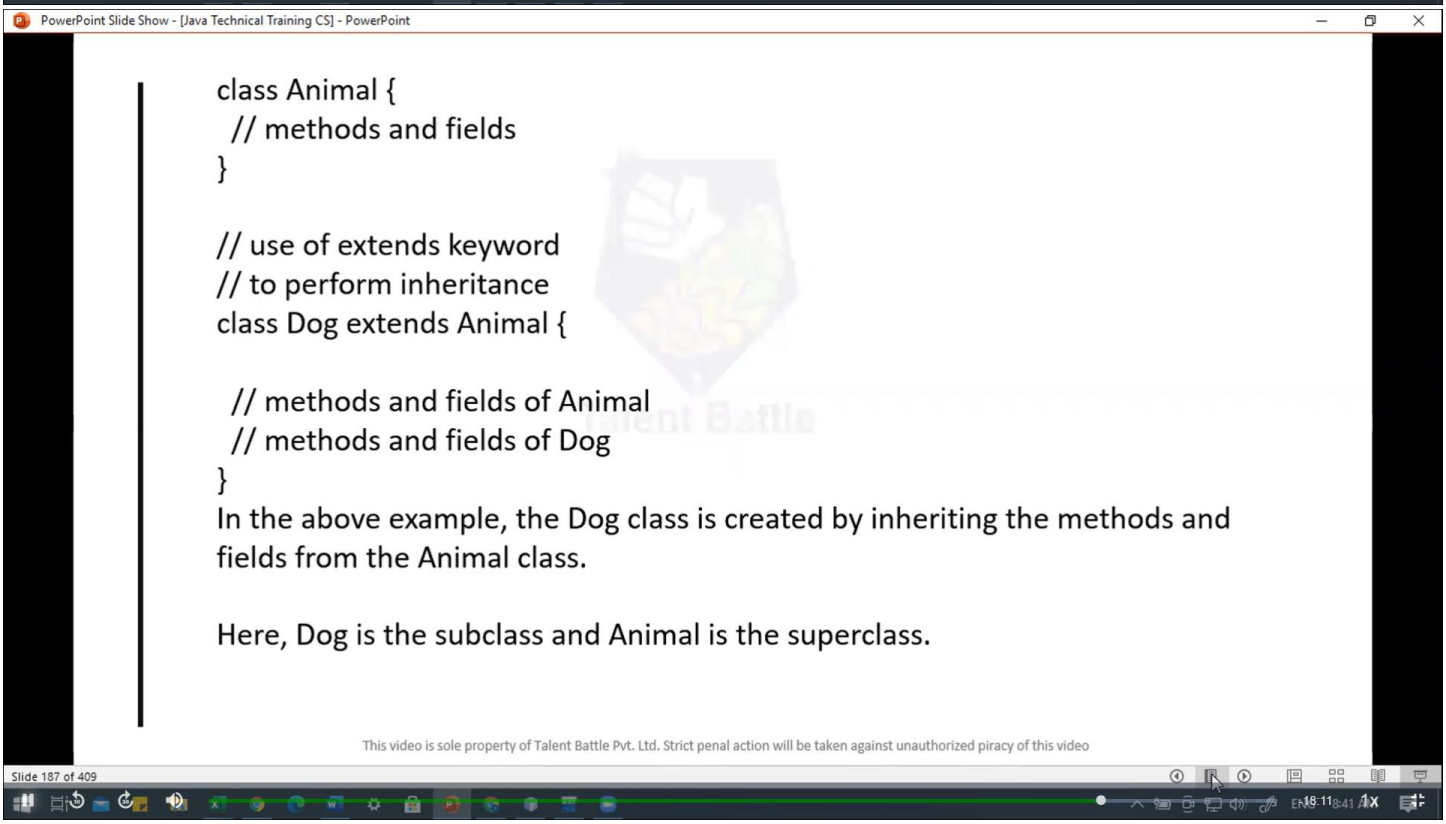
Here, if objectName is an instance of className, the operator returns true. Otherwise, it returns false.

Slide 183 of 409

-23:33 1x

Java Inheritence





PowerPoint Slide Show - [Java Technical Training CS] - PowerPoint

```
class Animal {
  // field and method of the parent class
  String name;
  public void eat() {
    System.out.println("I can eat");
  }
}
class Dog extends Animal {
    public void display() {
    System.out.println("My name is " + name);
  }
}
class Main {
  public static void main(String[] args) {
        Dog labrador = new Dog();
        labrador.name = “Tommy";
    labrador.display();
        labrador.eat();
  }
}
```

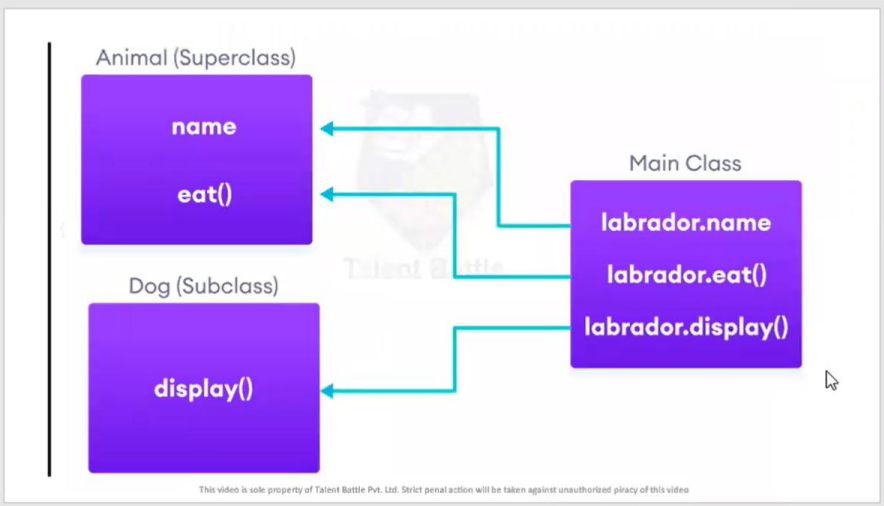

Slide 188 of 409

Java Technical Training CS - PowerPoint

Rohit Bag

File Home Insert Design Transitions Animations Slide Show Review View Help Foxit Reader PDF Tell me what you want to do Share

Paste Layout Reset Section New Slide Clipboard Slides Font Text Direction Align Text Convert to SmartArt Paragraph Arrange Quick Styles Shape Fill Shape Outline Shape Effects Drawing Find Replace Select Editing



Click to add notes

Slide 189 of 409 English (India) Notes Comments 87%

is-a relationship

In Java, inheritance is an **is-a** relationship. That is, we use inheritance only if there exists an is-a relationship between two classes. For example,

- **Car** is a **Vehicle**
- **Orange** is a **Fruit**
- **Surgeon** is a **Doctor**
- **Dog** is an **Animal**

Here, **Car** can inherit from **Vehicle**, **Orange** can inherit from **Fruit**, and so on.

Method Overriding in Java Inheritance

```
class Animal {
  public void eat() {
    System.out.println("I can eat");
  }
}
class Dog extends Animal {
  // overriding the eat() method
  @Override
  public void eat() {
    System.out.println("I eat dog food");
  }
  // new method in subclass
  public void bark() {
    System.out.println("I can bark");
  }
}
class Main {
  public static void main(String[] args) {
    // create an object of the subclass
    Dog labrador = new Dog();
    // call the eat() method
    labrador.eat();
    labrador.bark();
  }
}
```

super Keyword in Java Inheritance

Previously we saw that the same method in the subclass overrides the method in superclass.

In such a situation, the super keyword is used to call the method of the parent class from the method of the child class.

```java
package Day6;

/*
// Note: We cannot declare classes and interfaces private in java
// Private Access Modifier
class Data{
  // private variable
  private String name;

  // getter method
  public String getName(){
    return this.name;
  }

  // setter method
  public  void setName(String name){
    this.name = name;
  }
}

public class Main {
  public static void main(String[] args) {
    // create an object of Data
    Data d = new Data();

    // access the private variable using the getter and setter
    d.setName("Programming");
    System.out.println(d.getName());
  }
}

// ============= output ================
// PS C:\Users\hp\Desktop\TCS IT\Java> java Day6.Main
// Programming

 */



 //-------------------------------------------------------

 /*
 // Public Access Modifier
 // When methods, variables, classes, and so on are declared public, then we cna access them
from anywhere. The public access modifier has no scope restriction.

 class Animal {
  // public variable
  public int legCount;
```

```
 // public method
 public void display(){
   System.out.println("I am an animal.");
   System.out.println("I have " + legCount + " legs.");
 }
}

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

   // accessing the public class
   Animal animal = new Animal();

   // accessing the public variable
   animal.legCount = 4;

   // accessing the public method
   animal.display();
 }

}

// ==================== output ====================
//  PS C:\Users\hp\Desktop\TCS IT\Java>  c:; cd 'c:\Users\hp\Desktop\TCS IT\Java'; &
'C:\Program Files\Java\jdk-21\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp'
'C:\Users\hp\AppData\Roaming\Code\User\workspaceStorage\3f5645188fb5b1977966aea27e202e46\redh
at.java\jdt_ws\Java_a379e1ef\bin' 'Day6.Main'

// I am an animal.
// I have 4 legs.
// PS C:\Users\hp\Desktop\TCS IT\Java>



// =============== different method to get output ==================
// PS C:\Users\hp\Desktop\TCS IT\Java> java Day6.Main
// I am an animal.
// I have 4 legs.
// PS C:\Users\hp\Desktop\TCS IT\Java>


 */




// ------------------------------------------------------

/*
// Java this keyword
```

```
// In java, this keyword is used to refer to the current object inside a method or a
constructor.

// this  is nothing but the reference to the current  object.

class Main{
  int instVar;

  Main(int instVar){
    this.instVar = instVar;
    System.out.println("this reerence = " + this);
  }

  public static void main(String[] args) {
    // creating an object of the class
    Main obj = new Main(10);
    System.out.println("Object reference = " + obj);
  }
}


// ============= output ======================
// PS C:\Users\hp\Desktop\TCS IT\Java\Day6> java .\Main.java
// this reerence = Day6.Main@473b46c3
// Object reference = Day6.Main@473b46c3
 */



 //-------------------------------------------------------

 /*
 // First,let's see an example without using this keyword:

 class Main{
  int age;
  Main(int age){
    age = age;
  }

  public static void main(String[] args) {
    // creating an object of the class
    Main obj = new Main(10);
    System.out.println("Object reference = " + obj.age);
  }
 }

 // =============== output ==================
//  PS C:\Users\hp\Desktop\TCS IT\Java\Day6> java
.\Main.java
```

```
// Object reference = 0

 */


 // ------------------------------------------------------------

 /*
 // Now, let's rewrite the code using this keyword.
 // Now, we are getting this expected output. It is because when the constructor is called,
this inside the constructor is replaced by the object obj that has called the constructor.
Hence the age variable is assigned value 8.

 class Main{

  int age;
  Main(int age){
    this.age = age;
  }

  public static void main(String[] args) {
    // creating an object of the class
    Main obj = new Main(8);
    System.out.println("obj.age = " + obj.age);
  }
 }

 // ================== output
//  PS C:\Users\hp\Desktop\TCS IT\Java\Day6> java
.\Main.java
// obj.age = 8

 */


 // ------------------------------------------

 /*
 // this with Getters and Setters
 // Here, we have used this keyword:
  // - to assign  value inside the setter method
  // - to access value inside the getter method

class Main {
  String name;  // instance

  // setter method
  void setName(String name)  // String name is parameter
  {
    this.name = name;
```

```
  }

  // getter method
  String getName(){
    return this.name;
  }

  public static void main(String[] args) {
    // creating an object of the class
    Main obj = new Main();

    // calling the setter and the getter method
    obj.setName("Toshiba");
    System.out.println("obj.name: " + obj.getName());
  }
}

// =============== output ======================
// PS C:\Users\hp\Desktop\TCS IT\Java\Day6> java .\Main.java
// obj.name: Toshiba

*/


//-------------------------------------------------------

/*
// In java, we cannot  change the value of a final variable.
// Note: It is recommended to use uppercase to declare final variables in java

class Main{
  public static void main(String[] args){
    // create a final variable
    final int  AGE = 32;

    // try to change the final variable
    // AGE = 45; //  will gives error

    // Exception in thread "main" java.lang.Error: Unresolved compilation problem:
    // The final local variable AGE cannot be assigned. It must be blank and not using a
compound assignment

    System.out.println("Age: " + AGE);
  }
}

// =================== output ====================
// PS C:\Users\hp\Desktop\TCS IT\Java\Day6> java Main.java
// Age: 32
```

```
 */


// ------------------------------------------------------

/*
// Java final Method
// In java, the final method cannot be overriden by the child class.

class FinalDemo{
  // create a final method
  public final void display(){
    System.out.println("This is a final method.");
  }
}

class Main extends FinalDemo {
  // try to override the final method
  public final void display(){
    System.out.println("The final method is overriden.");
  }
  // Exception in thread "main" java.lang.Error: Unresolved compilation problem:
  // The final method display() cannot be overridden.


  public static void main(String[] args) {
    Main obj = new Main();
    obj.display();
  }
}

//=============== output ========================
// PS C:\Users\hp\Desktop\TCS IT\Java\Day6> java Main.java
// Main.java:258: error: display() in Main cannot override display() in FinalDemo
//   public final void display(){
//                     ^
//   overridden method is final
// 1 error
// error: compilation failed

 */



// ------------------------------------------------------------

/*
// Example: Factorial of a Number Using Recursion

class Factorial{
  static int factorial(int n){
```

```
    if(n != 0) // termination condition
      return n * factorial(n-1); // recursive call
    else{
      return 1; // base case
    }
  }

  public static void main(String[] args){
    int number = 4, result;
    result = factorial(number);
    System.out.println("Factorial of " + number + " = " + result);
  }
}

// ++++++++++++ output ++++++++++++++++++
// PS C:\Users\hp\Desktop\TCS IT\Java> javac Day6/Main.java
// PS C:\Users\hp\Desktop\TCS IT\Java> java Day6.Factorial
// Factorial of 4 = 24

 */



 //-------------------------------------------------------
/*
 class Main{
  public static void main(String[] args) {
    // create a variable of string type
    String name = "Programming";

    // checks if name is instance of String
    boolean result1 = name instanceof String;
    System.out.println("name is an instance of String: " + result1);

    // create an object of Main
    Main obj = new Main();

    // checks if obj is an instance of Main
    boolean result2 = obj instanceof Main;
    System.out.println("obj is an instance of Main: " + result2);
  }
 }

//  +++++++++++++++ output ++++++++++++++++++++
// @Dheeraj2002kumar ➜ /workspaces/Foundation-Language/Java (main) $ javac Day6/Main.java
// @Dheeraj2002kumar ➜ /workspaces/Foundation-Language/Java (main) $ java Day6.Main
// name is an instance of String: true
// obj is an instance of Main: true

 */
```

```java
//-----------------------------------------------------------

/*
// Inheritance

class Animal{
  // field and method of the parent class
  String name;

  public void eat(){
    System.out.println("I can eat");
  }
}

class Dog extends Animal{
  public void display(){
    System.out.println("My name is " + name);
  }
}

class Main{
  public static void main(String[] args){
    Dog labrador = new Dog();
    labrador.name = "Tommy";
    labrador.display();
    labrador.eat();
  }
}

//+++++++++++++++ output +++++++++++++++++
// @Dheeraj2002kumar ➜ /workspaces/Foundation-Language/Java (main) $ javac Day6/Main.java
// @Dheeraj2002kumar ➜ /workspaces/Foundation-Language/Java (main) $ java Day6.Main
// My name is Tommy
// I can eat

 */




 //-------------------------------------------------------

 /*
 // Method Overriding in Java Inheritance

 class Animal{
  public void eat(){
    System.out.println("I can eat");
```

```
 }
}

class Dog extends Animal{
 // overriding the eat() method
 @Override
 public void eat(){
   System.out.println("I eat dog food..");
 }

 // new method in subclass
 public void bark(){
   System.out.println("I can bark");
 }
}

class Main{
 public static void main(String[] args) {
   // create an object of the subclass
   Dog labrador = new Dog();

   // call the eat() method
   labrador.eat();
   labrador.bark();
 }
}



// ++++++++++++++ output ++++++++++++++++++
//  @Dheeraj2002kumar ➜ /workspaces/Foundation-Language/Java (main) $ javac Day6/M
// ain.java
// @Dheeraj2002kumar ➜ /workspaces/Foundation-Language/Java (main) $ java  Day6/M
// ain.java
// error: can't find main(String[]) method in class: Day6.Animal
// @Dheeraj2002kumar ➜ /workspaces/Foundation-Language/Java (main) $ java Day6.Main
// I eat dog food..
// I can bark

*/




// ================================================

// Super keyword in java inheritance

class Animal{
  public void eat(){
```

```java
    System.out.println("I can eat");
  }
}

class Dog extends Animal{
  // overriding the eat() method
  @Override
  public void eat(){
    // call method of superclass
    super.eat();
    System.out.println("I eat dog food..");
  }

  // new method in subclass
  public void bark(){
    System.out.println("I can bark....");
  }
}

public class Main {

  public static void main(String[] args) {
    Dog labrador = new Dog();

    // call the eat() method
    labrador.eat();
    labrador.bark();
  }
}


// ++++++++++++++++++ output +++++++++++++++++++++
// @Dheeraj2002kumar ➜ /workspaces/Foundation-Language/Java (main) $ javac Day6/Main.java
// @Dheeraj2002kumar ➜ /workspaces/Foundation-Language/Java (main) $ java Day6.Main
// I can eat
// I eat dog food..
// I can bark....
```