Encapsulation

- Encapsulation in java is a process of wrapping code and data together into a single unit, for example capsule i.e. mixed of several medicines.
- A java class is the example of encapsulation
- Wrapping of data and functions together as a single unit is known as encapsulation.
- By default data is not accessible to outside world and they are only accessible through the functions which are wrapped in a class.
- prevention of data direct access by the program is called data hiding or information hiding

CamScanner

Cont...

Main.java

Create a class named "Main" with a variable x:

```
public class Main {
  int x = 5;
}
```

Remember from the Java Syntax chapter that a class should always start with an uppercase first letter, and that the name of the java file should match the class name

CamScanner

JAVA CLASSES/OBJECTS

- Java is an object-oriented programming language.
- Everything in Java is associated with classes and objects, along with its attributes and methods.
- For example: in real life, a car is an object.
- The car has attributes, such as weight and color, and methods, such as drive and brake
- A Class is like an object constructor, or a "blueprint" for creating objects

Creating a Class

To create a class, use the keyword **class:**

CamScanner

Creating an Object

In Java, an object is created from a class. We have already created the class named Main, so now we can use this to create objects.

To create an object of Main, specify the class name, followed by the object name, and use the keyword new:

Syntax:

```
ClassName  reference_variable = new  Classname( );
```

Example

Create an object called "myObj" and print the value of x:

```
public class Main {
  int x = 5;
 public static void main(String[] args) {
    Main myObj = new Main();
    System.out.println(myObj.x);
  }
}
```

CamScanner

METHODS

- A Java method is a collection of statements that are grouped together to perform an operation.
- A portion of code that performs a specific Operation
- Creating Method
- Considering the following example to explain the syntax of a method –

Syntax

```
public static int methodName(int a, int b) {
   // body }
```

CamScanner

Cont...

- ✓ Here,
- ✓ **public static** – modifier
- ✓ **int** – return type
- ✓ **methodName** – name of the method
- ✓ **a, b** – formal parameters
- ✓ **int a, int b** – list of parameters

CamScanner

Cont..

- Method definition consists of a method header and a method body. The same is shown in the following syntax –
- **Syntax**
- **modifier returnType nameOfMethod (Parameter List) { // method body }**
- **The syntax shown above includes –**

modifier – It defines the access type of the method and it is optional to use.

returnType – Method may return a value.

nameOfMethod – This is the method name. The method signature consists of the method name and the parameter list.

Parameter List – The list of parameters, it is the type, order, and number of parameters of a method. These are optional, method may contain zero parameters.

method body – The method body defines what the method does with the stat

CamScanner

Method Calling

- For using a method, it should be called. There are two ways in which a method is called i.e., method returns a value or returning nothing (no return value).
- The process of method calling is simple.
- When a program invokes a method, the program control gets transferred to the called method.
- This called method then returns control to the caller in two conditions, when

The return statement is executed.

It reaches the method ending closing brace.

CamScanner

JAVA CONTROL STRUCTURES:

- A control structure is a syntactic form in a language to express flow of control.
- A sequence of statements is executed depending on whether or not the condition it true or false .
- This means the program chooses between two or more alternative paths.
- Hence it is the basic decision-making process in computer programming; flow control determines how a computer will respond when given certain conditions and parameters.

CamScanner

Cont...

- This condition is called sequence accomplishment .
- This scenario is known as transfer of program control
- Decision making is an important part of programming.
- It is used to specify the order in which statements are executed.

CamScanner

Java if-statement

- In Java, an if statement is a conditional statement that runs a different set of statements depending on whether an expression is true or false

Syntax:

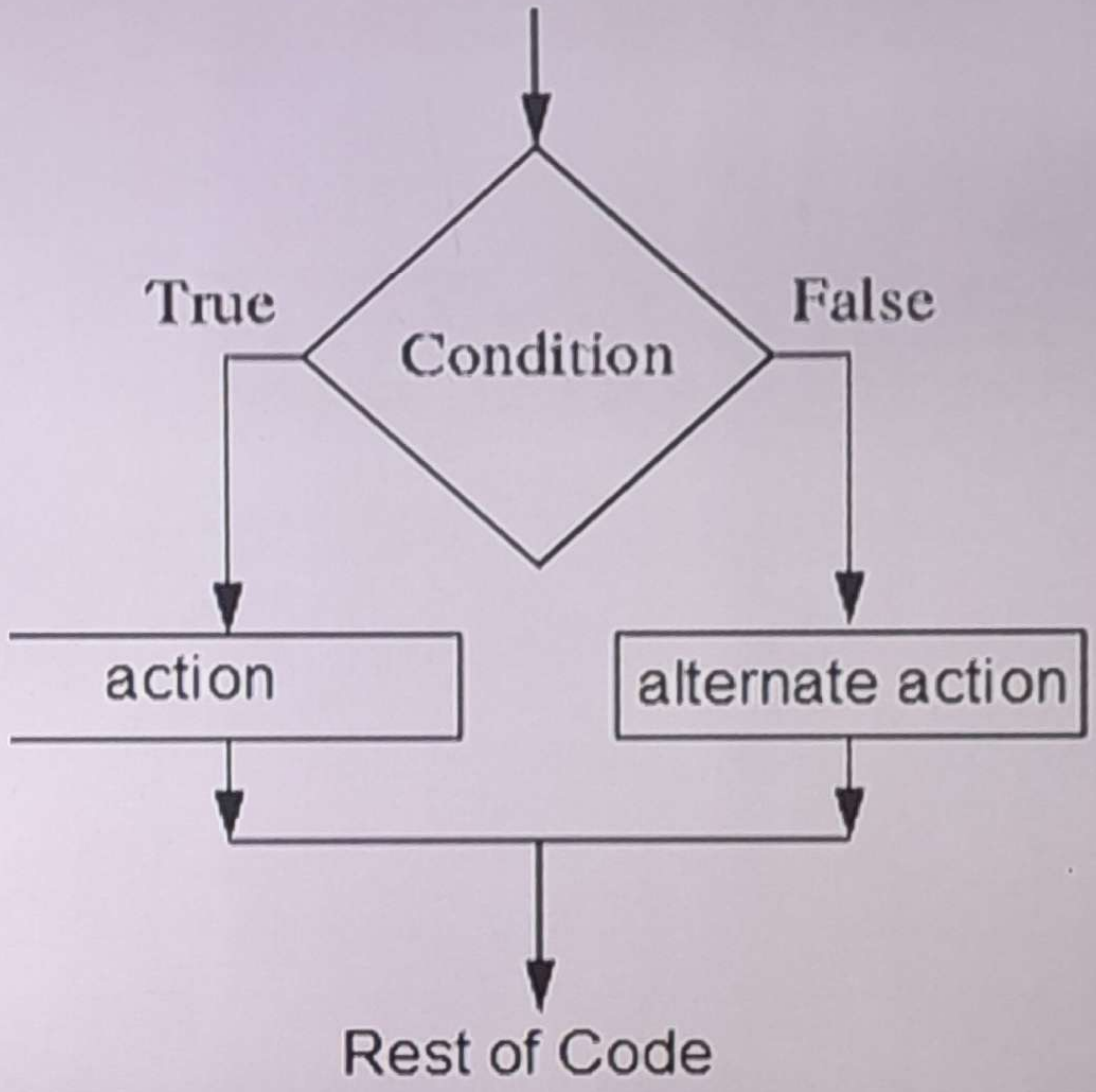
```
If (condition){
  //code to be executed if the condition is true
}
```

CamScanner

Cont...

- In the above syntax, the if statement evaluates the test expression inside parenthesis.
- If test expression is evaluated to true (nonzero) , statements inside the body of if is executed.
- If test expression is evaluated to false (0) , statements inside the body of if is skipped

CamScanner



CamScanner

Example:

```
public class TestClass {

  public static void main(String[] args) {

    int totalMarks=55;

    if(totalMarks>50){

      System.out.print("You have passed the exam !!");

    }

  }

}
```

CamScanner

Java if...else Statement

- The else statement is to specify a block of code to be executed, if the condition in the if statement is false

Syntax:

```
if(condition){

  //code to be executed if the condition is true

}else{

  //code to be executed if the condition is false

}
```

The else clause of an if...else statement is associated with the closest previous if statement in tl same scope that does not have a corresponding else statement.

CamScanner

Example:

```
public class TestClass {
  public static void main(String[] args) {
    int totalMarks=48;
    if(totalMarks>50){
      System.out.print("You have passed the exam !!");
    }
    else {
      System.out.print("You have failed the exam !!");
          }
        }
      }
```

In the above code, the if stat evaluate the expression is true or false. In this case the totalMarks>50 is false, then the control goes to the body of else block , that is the program will execute the code inside else block.

CamScanner

Java if...else if Statement (if else ladder)

- If you want to evaluate more than one conditions at the same time , you can use else if statement in Java.
- Multi selection enables the developer to determine the actions that have to be accomplished in certain conditions by imposing a requisite.
- You can combine an else and an if to make an else if and test a whole range of mutually exclusive possibilities

CamScanner

Syntax:

```
if(condition1){
  //code to be executed if the condition1 is true
}else if(condition2){
  //code to be executed if condition2 is true
}
else if(condition3){
  //code to be executed if condition3 is true
  }
   else{
     //code to be executed if all the above conditions are false
   }
```

CamScanner

Explanation:

```
1:  if(totalMarks>=80){
2:    System.out.print("Got Higher First Class");
3:  }
4:  else if (totalMarks>=60 & &  totalMarks < 80){
5:    System.out.print("Got First Class");
6:  }
7:  else if (totalMarks>=40 & & totalMarks < 60){
8:    System.out.print("Just pass only");
9:  }
10: else {
11:   System.out.print("You have failed the exam !!");
12: }
```

CamScanner

Explanation:

```
1:  if(totalMarks>=80){
2:    System.out.print("Got Higher First Class");
3:  }
4:  else if (totalMarks>=60 & &  totalMarks < 80){
5:    System.out.print("Got First Class");
6:  }
7:  else if (totalMarks>=40 & & totalMarks < 60){
8:    System.out.print("Just pass only");
9:  }
10: else {
11:   System.out.print("You have failed the exam !!");
12: }
```

CamScanner

Example:

```
public class TestClass {
  public static void main(String[] args) {
    int totalMarks=64;
    if(totalMarks>=80){
      System.out.print("Got Higher First Class");
    }
    else if (totalMarks>=60  & & totalMarks < 80 ){
      System.out.print("Got First Class");
    }
    else if (totalMarks>=40 & & totalMarks < 60){
      System.out.print("Just pass only");
    }
    else {
      System.out.print("You have failed the exam !!");
       }
       }
      }
```

CamScanner

Create a constructor

```
// Create a Main class
public class Main {
  int x;  // Create a class attribute
  // Create a class constructor for the Main class
  public Main() {
    x = 5;  // Set the initial value for the class attribute x
  }
  public static void main(String[] args) {
 Main myObj = new Main(); // Create an object of class Main (This will call the constructor)
   System.out.println(myObj.x); // Print the value of x
  }
}
// Outputs 5
```

CamScanner

Parameters Vs Arguments

- A **parameter** is a variable named in the function or method definition.
- It acts as a placeholder for the data the function will use.
- **An argument** is the actual value that is passed to the function or method when it is called
- The values that are declared within a method when the method is called are known as an **argument**.
- Whereas, the variables that are defined when the method is declared are known as a **parameter**

CamScanner

Example:

```
1.  public class Addition
2.  {
3.  public static void main(String[] args)
4.  {
5.  int a = 19;
6.  int b = 5;
7.  //method calling
8.  int c = add(a, b);   //a and b are arguments
9.  System.out.println("The sum of a and b is= " + c);
10. }
11. //user defined method
12. public static int add(int n1, int n2)   //n1 and n2 are formal parameters
13. {
14. int s;
15. s=n1+n2;
16. return s; //returning the sum
17. }
18. }
```

CamScanner

METHOD OVERRIDING and OVERLOADING

- When two or more methods in the same class have the same name but different parameters, it's called **Overloading.**
- **Method overloading** –Two or more methods in the same class with same method name but different parameters
- When the method signature (name and parameters) are the same in the superclass and the child class, it's called **Overriding**.
- **Method overriding-**Two methods , one in the superclass and the other in subclass with the same name and same parameters

CamScanner

Java Method Overloading

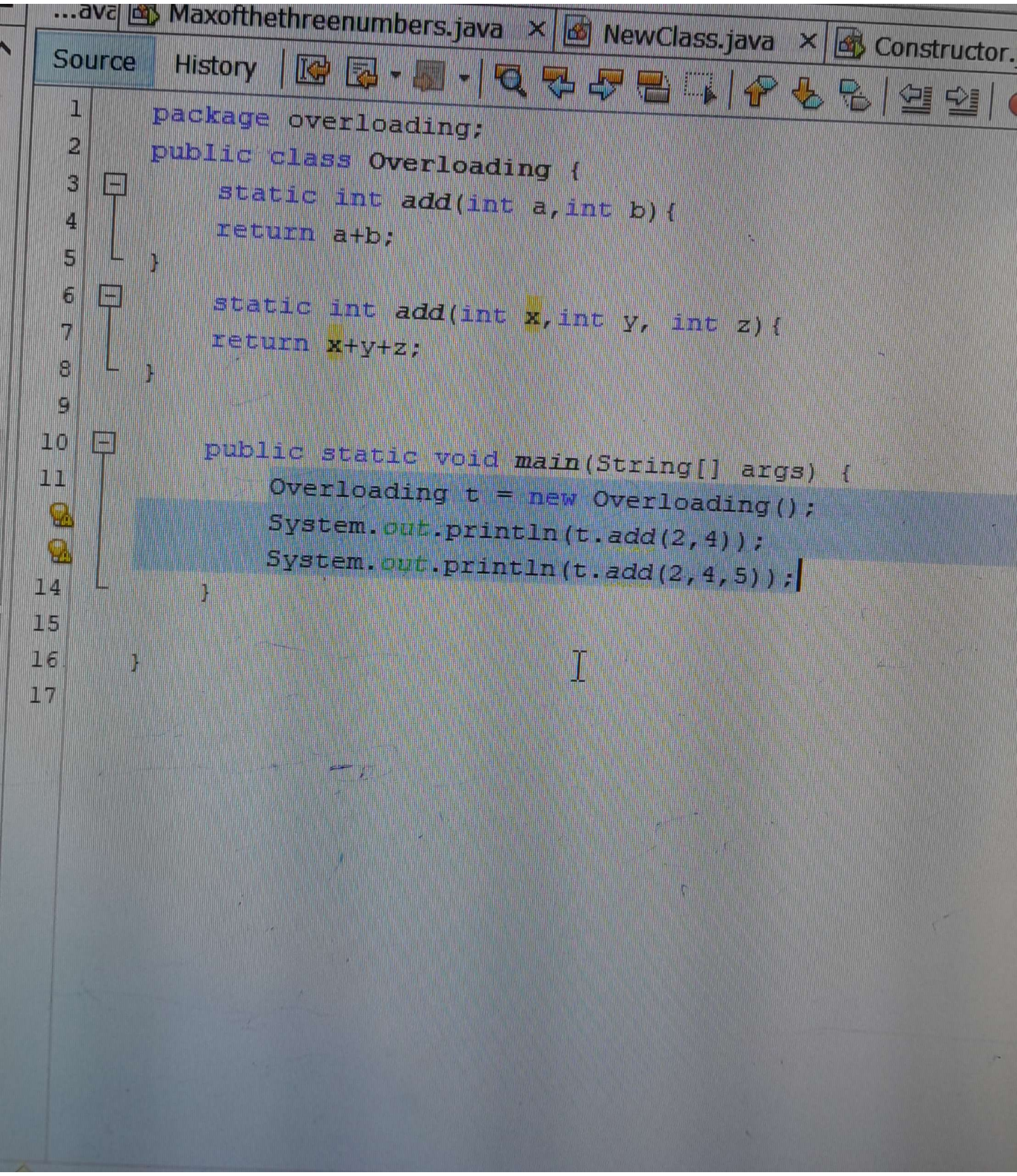
```
class OverloadingExample
{
static int add(int a,int b)
{
return a+b;
}
static int add(int a,int b,int c)
{
return a+b+c;
}
}
```

CamScanner

Java Method Overriding

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

CamScanner


...ava
Maxofthethreenumbers.java
NewClass.java
Constructor.
Source
History
1 package overloading;
2 public class Overloading {
3 static int add(int a,int b){
4 return a+b;
5 }
6 static int add(int x,int y, int z){
7 return x+y+z;
8 }
9
10 public static void main(String[] args) {
11 Overloading t = new Overloading();
System.out.println(t.add(2,4));
System.out.println(t.add(2,4,5));
14 }
15
16 }
17

CamScanner

INHERITANCE

Inheritance can be defined as the procedure or mechanism of acquiring all the properties and behaviors of one class to another, i.e., acquiring the properties and behavior of child class from the parent class.

When one object acquires all the properties and behaviours of another object, it is known as inheritance. Inheritance represents the **IS-A relationship**, also known as *parent-child* relationship.

Uses of inheritance in java

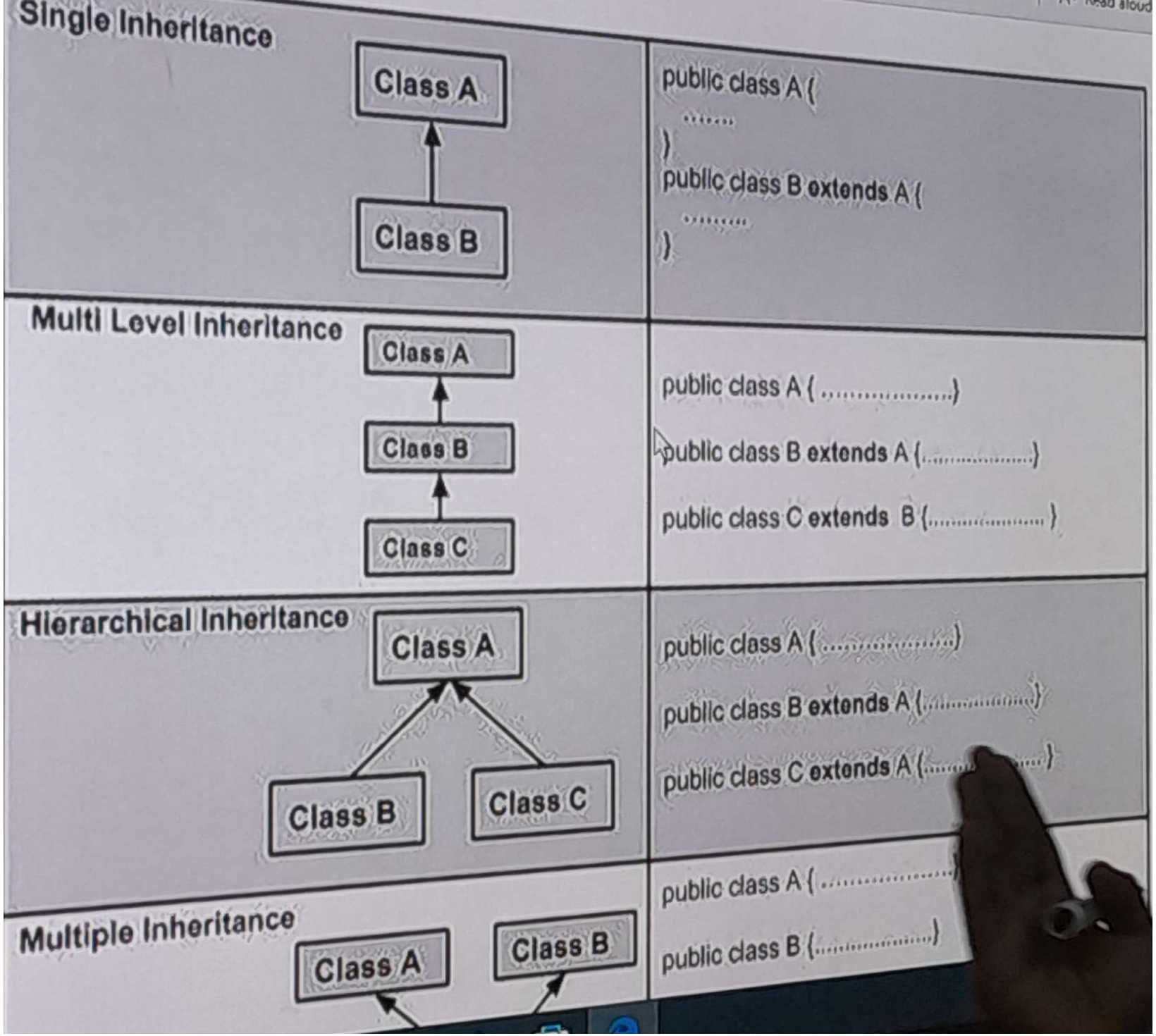
- For Method Overriding (so runtime polymorphism can be achieved).
- For Code Reusability.

Types of inheritance in java: single, multilevel and hierarchical inheritance. Multiple and hybrid inheritance is supported through interface only.

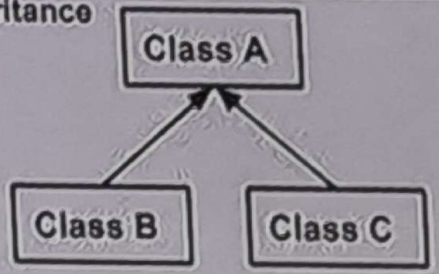
Syntax:

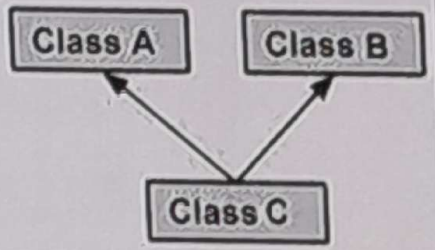
```
class subClass extends superClass
{
  //methods and fields
```

Type here to search
1:05 PM
7/9/2025
CamScanner


Single Inheritance
Class A
Class B
public class A {
.....
}
public class B extends A {
.....
}
Multi Level Inheritance
Class A
Class B
Class C
public class A { .....}
public class B extends A {.....}
public class C extends B {.....}
Hierarchical Inheritance
Class A
Class B
Class C
public class A { .....}
public class B extends A {.....}
public class C extends A {.....}
Multiple Inheritance
Class A
Class B
public class A { .....}
public class B {.....}

oops new.pdf
file:///C:/Users/Timetabler/Desktop/lec1/OOPS/oops%20new.pdf
of 174 Contents Fit to page Page view Read aloud Add notes

Hierarchical Inheritance 	public class A {.....} public class B **extends** A {.....} public class C **extends** A {.....}

Multiple Inheritance
 
	public class A { .....}
	public class B {.....}
	public class C **extends** A,B {
	.....
	} // Java does not support mutiple Inheritance

Type here to search
CamScanner

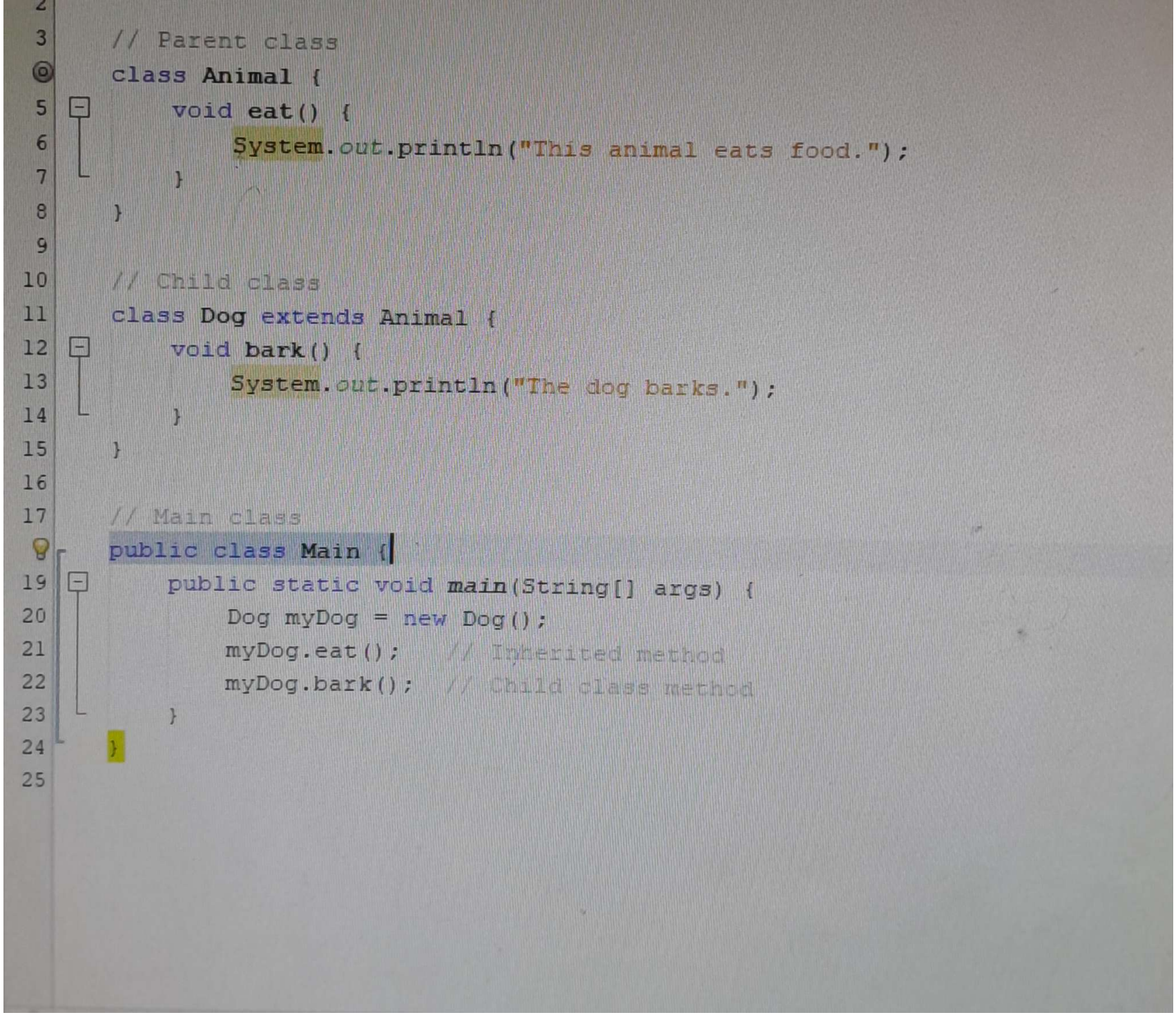
Lecture 2 - PowerPoint
Sign in
Insert Draw Design Transitions Animations Slide Show Record Review View Help Tell me what you want to do
Share

Group Assignment practical

1. Discuss the **exception Handling In Java** with syntax and program example
2. Explain **Try-catch Block** in java with program example
3. Desribe the **Java Packages** with syntax and program example
4. Discuss the **Java. Awt Package** with program example
5. Explain the **Java Swing Package** with syntax and program example

Click to add notes
of 46
Accessibility: Investigate
Notes
Comments

CamScanner



```java
// Parent class
class Animal {
    void eat() {
        System.out.println("This animal eats food.");
    }
}

// Child class
class Dog extends Animal {
    void bark() {
        System.out.println("The dog barks.");
    }
}

// Main class
public class Main {
    public static void main(String[] args) {
        Dog myDog = new Dog();
        myDog.eat();  // Inherited method
        myDog.bark();  // Child class method
    }
}
```

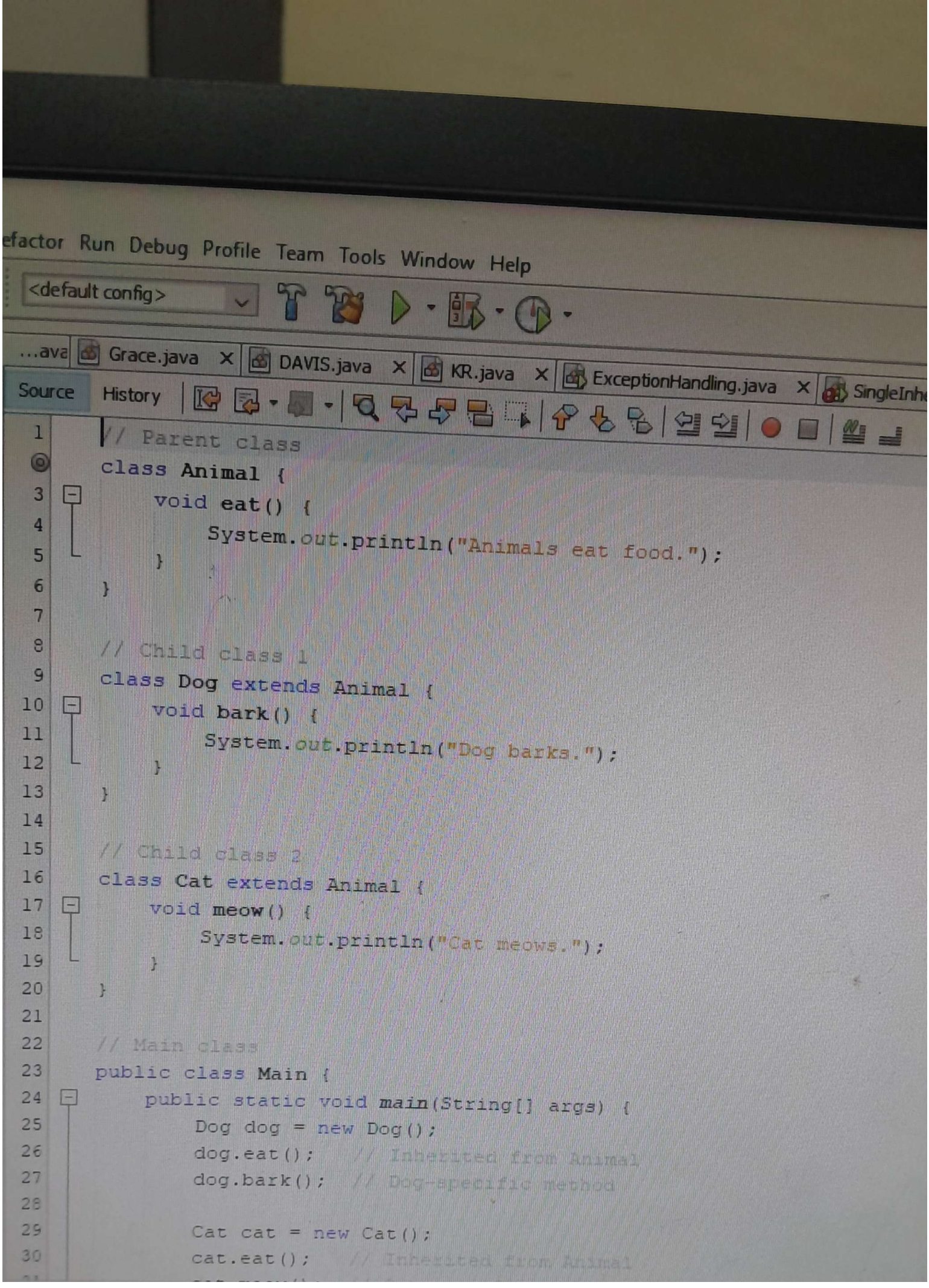
CamScanner

```java
// Intermediate class
class Dog extends Animal {
    void bark() {
        System.out.println("The dog barks.");
    }
}

// Derived class
class Puppy extends Dog {
    void weep() {
        System.out.println("The puppy weeps.");
    }
}

// Main class
public class Main {
    public static void main(String[] args) {
        Puppy myPuppy = new Puppy();

        myPuppy.eat();   // Method from Animal
        myPuppy.bark();  // Method from Dog
        myPuppy.weep();  // Method from Puppy
    }
}
```

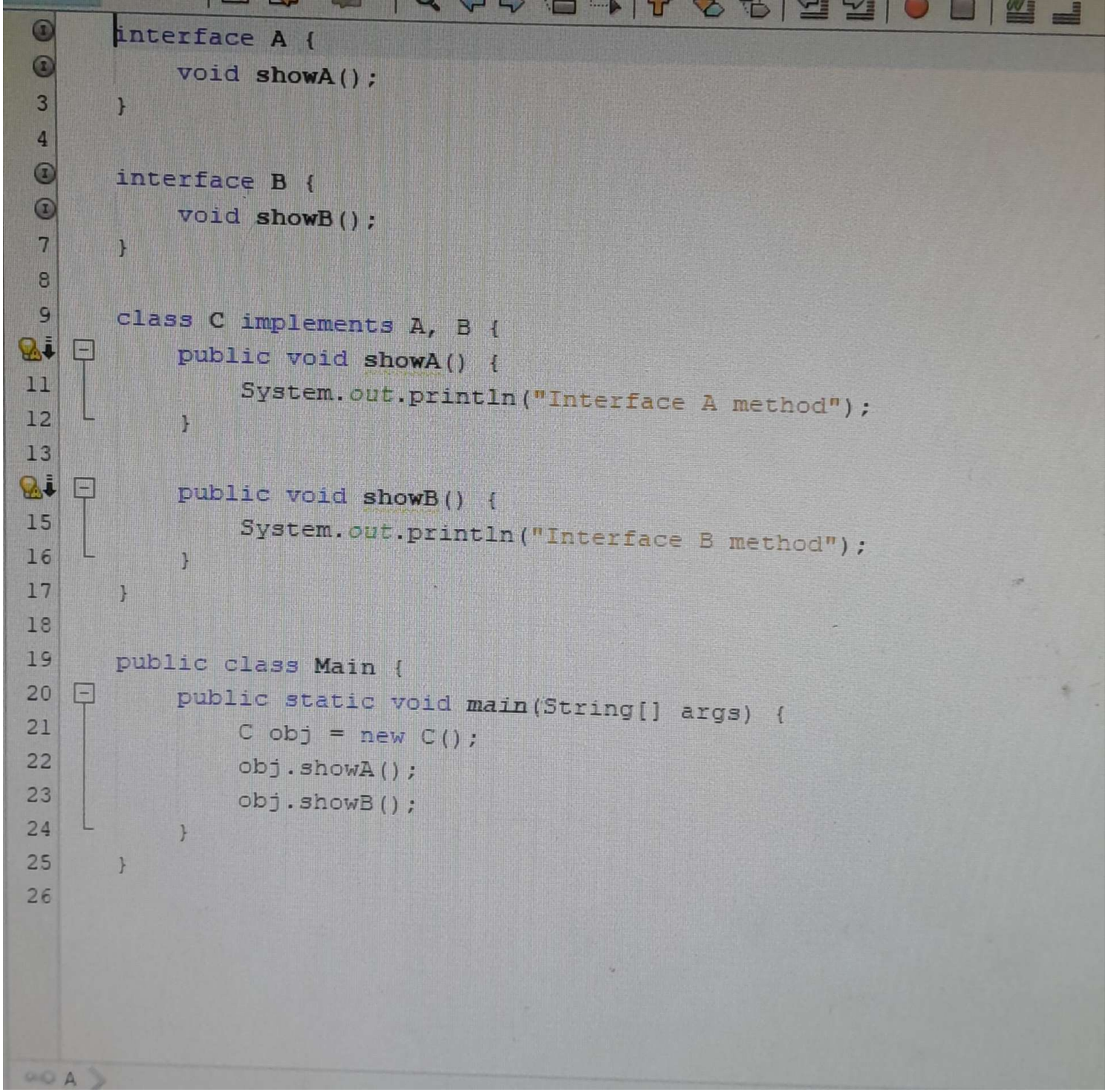
CamScanner

```
        System.out.println("Dog barks.");
    }
}

// Child class 2
class Cat extends Animal {
    void meow() {
        System.out.println("Cat meows.");
    }
}

// Main class
public class Main {
    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.eat();  // Inherited from Animal
        dog.bark();  // Dog-specific method

        Cat cat = new Cat();
        cat.eat();  // Inherited from Animal
        cat.meow();  // Cat-specific method
    }
}
```

CamScanner


efactor Run Debug Profile Team Tools Window Help
<default config>
...ava
Grace.java
DAVIS.java
KR.java
ExceptionHandling.java
SingleInhe
Source
History
// Parent class
class Animal {
void eat() {
System.out.println("Animals eat food.");
}
}
// Child class 1
class Dog extends Animal {
void bark() {
System.out.println("Dog barks.");
}
}
// Child class 2
class Cat extends Animal {
void meow() {
System.out.println("Cat meows.");
}
}
// Main class
public class Main {
public static void main(String[] args) {
Dog dog = new Dog();
dog.eat(); // Inherited from Animal
dog.bark(); // Dog-specific method
Cat cat = new Cat();
cat.eat(); // Inherited from Animal

CamScanner



```java
interface A {
    void showA();
}

interface B {
    void showB();
}

class C implements A, B {
    public void showA() {
        System.out.println("Interface A method");
    }

    public void showB() {
        System.out.println("Interface B method");
    }
}

public class Main {
    public static void main(String[] args) {
        C obj = new C();
        obj.showA();
        obj.showB();
    }
}
```

CamScanner