23/2/2024

There are two main categories of data types: primitive data types and non-primitive (or reference) data types.

Primitive Data Types:

Primitive data types are typically of fixed size and directly represent simple values.

Examples of primitive data types include integers, floating-point numbers, characters, booleans, and pointer

Non-primitive (Reference) Data Types:

data types that are derived from primitive data types.

created by the programmer

because variables of these types store references (memory addresses) to the actual data rather than the data itself.

examples of non-primitive data types include arrays, strings, objects, classes, interfaces

Differences between Primitive and Non-primitive Data Types:

Nature:

Storage:

Primitive data types directly contain the actual value they represent.

Non-primitive data types contain references to the location in memory where the actual data is stored.

Size:

Primitive data types usually have fixed sizes defined by the programming language.

Non-primitive data types can have variable sizes depending on the data they contain.

Operations:

Primitive data types typically support basic operations directly provided by the programming language.

Non-primitive data types often require custom operations or methods defined by the programmer

The ASCII standard defines codes for 128 characters

Uppercase letters (A-Z): ASCII values 65-90

Lowercase letters (a-z): ASCII values 97-122

Digits (0-9): ASCII values 48-57

Space character: ASCII value 32

null represents the **absence of a reference.**

void represents the absence of a return value in function or method declarations.

"x |= 3" is a shorthand way of performing a bitwise OR operation between the variable x and the number 3.

public class Main {

public static void main(String[] args) {

int x = 5;

x |= 3;

System.out.println(x);

}

}

Ans: 7

if we apply ">>=" to 5 and 3, it would mean that 5 is shifted to the right by 3 positions. So, 5 would become 0 because it shifted all the way to the right.

public class Main {

public static void main(String[] args) {

int x = 5;

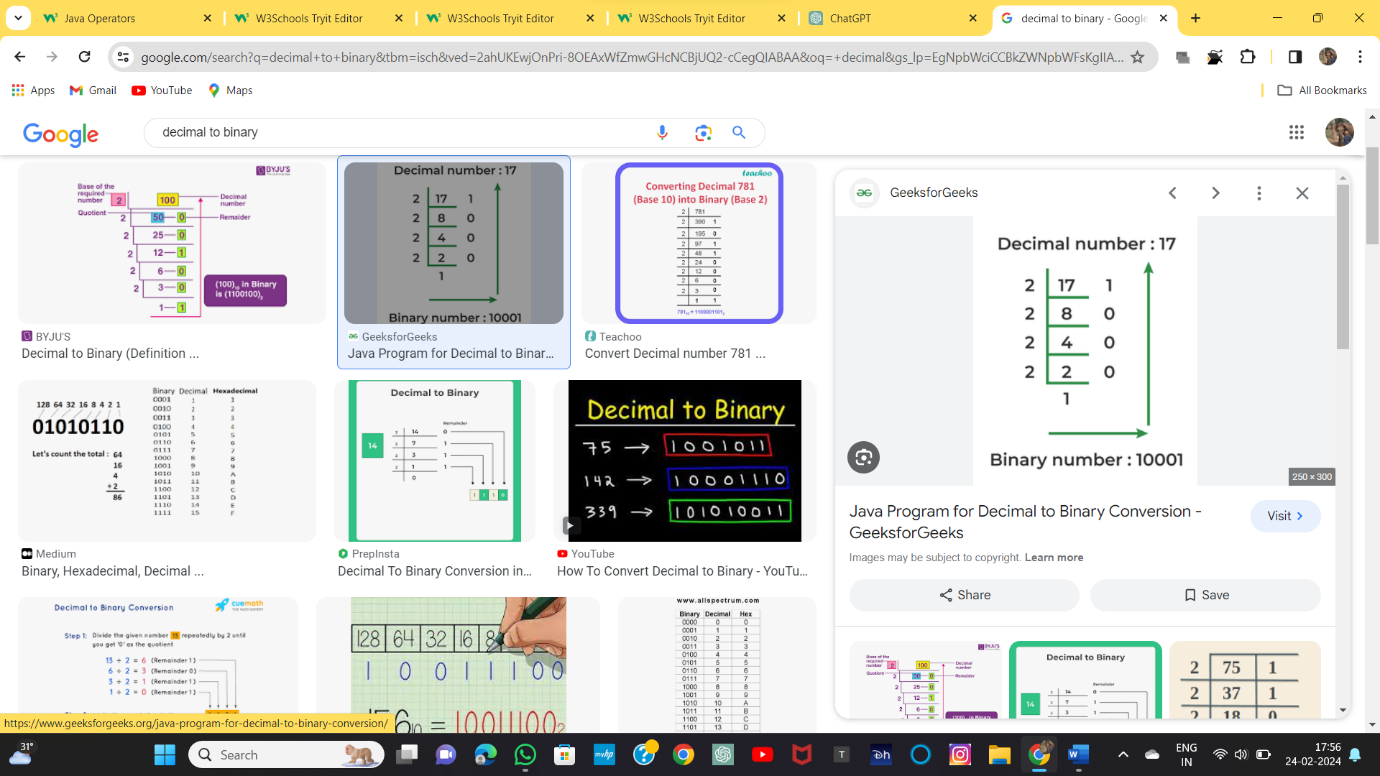
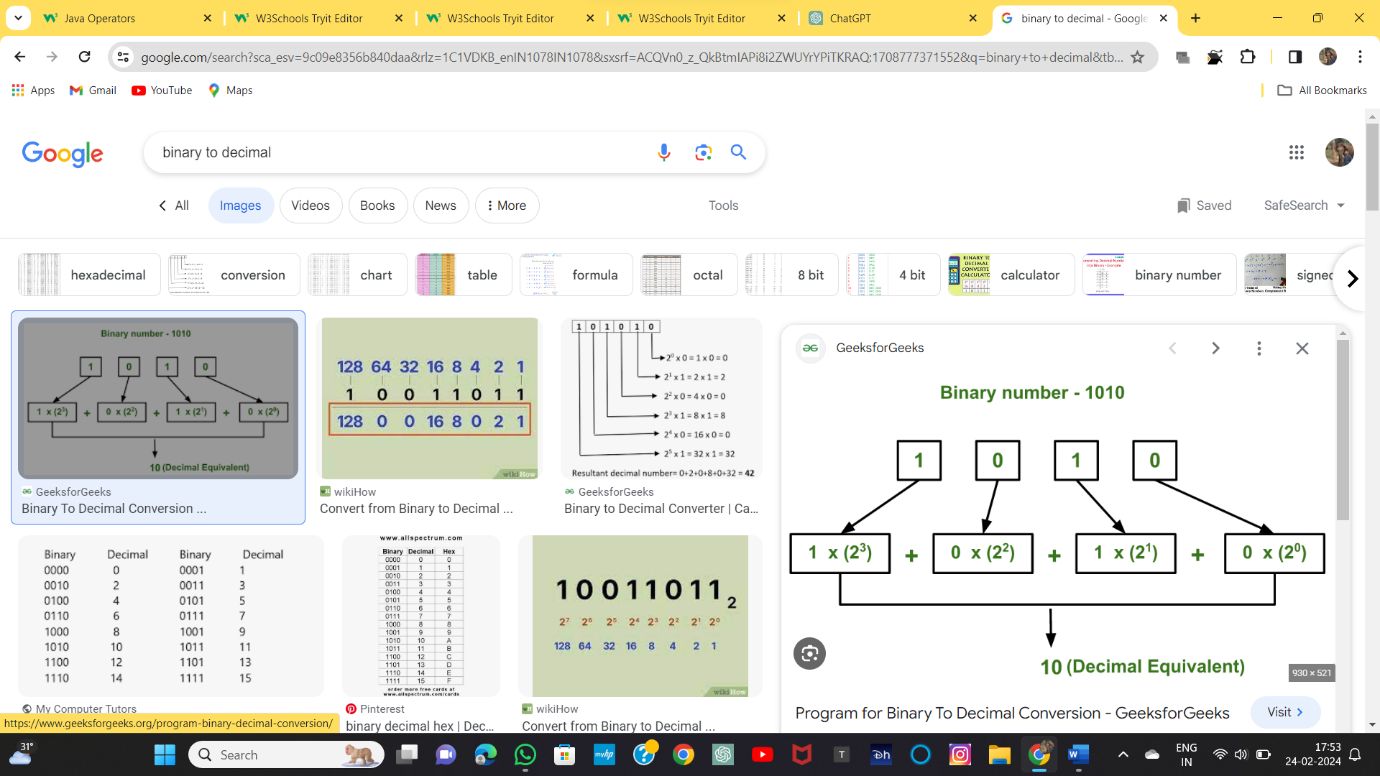
x >>= 3;

System.out.println(x);

}

}

Ans: 0



f you add a number and a string, the result will be a string concatenation:

Example

String x = "10";

int y = 20;

String z = x + y; // z will be 1020 (a String)

\n New Line

\r Carriage Return

\t Tab

\b Backspace

\f Form Feed

The Math.abs(x) method returns the absolute (positive) value of x:

public class Main {

public static void main(String[] args) {

System.out.println(Math.abs(-4.7));

}

}

ANS : 4.7

To get more control over the random number, for example, if you only want a random number between 0 and 100, you can use the following formula:

Example

int randomNum = (int)(Math.random() \* 101); // 0 to 100

Short Hand If...Else

int time = 20;

String result = (time < 18) ? "Good day." : "Good evening.";

System.out.println(result);

Java Switch Statements

int day = 4;

switch (day) {

case 6:

System.out.println("Today is Saturday");

break;

case 7:

System.out.println("Today is Sunday");

break;

default:

System.out.println("Looking forward to the Weekend");

}

For-Each Loop

There is also a "for-each" loop, which is used exclusively to loop through elements in an array:

String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

for (String i : cars) {

System.out.println(i);

}

Volvo

BMW

Ford

Mazda

The break statement can also be used to jump out of a loop.

Java Continue

If a specified condition occurs, and continues with the next iteration in the loop.

for (int i = 0; i < 10; i++) {

if (i == 4) {

continue;

}

System.out.println(i);

}

example skips the value of 4

public class Main {

public static void main(String[] args) {

int i = 0;

while (i < 10) {

if (i == 4) {

continue;

}

System.out.println(i);

i++;

}

}

}

0

1

2

3

public class Main {

public static void main(String[] args) {

int i = 0;

while (i < 10) {

if (i == 4) {

**i++;**

continue;

}

System.out.println(i);

i++;

}

}

}

0

1

2

3

5

6

7

8

9

26/2/2024

Procedural Programming:

In procedural programming, the focus is on writing procedures or functions that perform operations on data.

Data and functions are separate entities.

function calculateArea(length, width)

return length \* width

length = 5

width = 3

area = calculateArea(length, width)

print area

Object-Oriented Programming (OOP):

In object-oriented programming, the focus is on creating objects that contain both data and methods (functions) that operate on the data.

class Rectangle {

private int length;

private int width;

public Rectangle(int length, int width) {

this.length = length;

this.width = width;

}

public int calculateArea() {

return length \* width;

}

}

Rectangle rectangle = new Rectangle(5, 3);

int area = rectangle.calculateArea();

System.out.println(area);

5. Object-oriented programming has several advantages over procedural programming:

OOP makes it possible to create full reusable applications with less code and shorter development time

OOP helps to keep the Java code DRY "Don't Repeat Yourself",

6. why new keyword is used " Main myObj = new Main();" while creating a object of a class

Ans: Memory Allocation:

When you use new, you're requesting the system to allocate memory on the heap for a new object instance.

Why requires memory: In oops the focus is on creating objects that contain both data and methods (functions) that operate on the data.

So to hold data we need memory.

Constructor Invocation: The new keyword also triggers the execution of a special method called the constructor

7. Constructor in java?

Ans: A constructor in Java is a special type of method that serves the essential purpose of initializing objects when they're created.

Key Points:

**Name:** The constructor's name must match the class name it belongs to.

**No Return Type**: Unlike regular methods, constructors don't have a return type, not even void. They implicitly return the newly created object itself.

**Automatic Invocation:** Constructors are invoked automatically whenever you use the **new keyword** to create an object of the class.

**Initialization**: The primary function of a constructor is to initialize the object's state. This typically involves assigning initial values to the object's fields (member variables) to establish its starting condition.

Optional: While you can define constructors for a class, **it's not mandatory**. If you don't explicitly define one, the **compiler will** provide a default constructor with an empty body.

**Parameters:** Constructors can have parameters to accept values during object creation, allowing for customization of the initial state. This is known as constructor overloading, where you can have multiple constructors with different parameter lists to cater to various initialization scenarios

8. Final keyword

Variables

When used with a variable, final makes it read-only, meaning its value can only be assigned once. used to create constants,

final double PI = 3.14159;

PI = 3.0; // This will cause a compilation error because PI is final

Methods:

Declaring a method as final prevents it from being **overridden by subclasses** in an inheritance hierarchy.

class Parent {

public final void display() {

System.out.println("Parent class display method");

}

}

class Child extends Parent {

// This attempt to override display() will cause a compilation error

// because display() is final in the Parent class

public void display() {

System.out.println("Child class display method"); // Not allowed

}

}

Classes:

Making a class final prohibits it from being extended by other classes.

final class MathUtils {

public static int add(int a, int b) {

return a + b;

}

}

// This attempt to create a subclass of MathUtils will cause a compilation error

// because MathUtils is final

class ExtendedMathUtils **extends** MathUtils {

// ...

}

9. Overridding vs OverLoading?

Overloading:

Occurs within a **single class**.

Involves multiple methods sharing the same name but having different parameter lists.

The compiler determines which overloaded method to call based on the number and types of arguments provided at the call site.

class Calculator {

public int add(int a, int b) {

return a + b;

}

public double add(double a, double b) {

return a + b;

}

public static void main(String[] args) {

Calculator calc = new Calculator();

System.out.println(calc.add(5, 3)); // Calls the first add() with int arguments

System.out.println(calc.add(2.5, 1.8)); // Calls the second add() with double arguments

}

}

Overriding:

Occurs in inheritance hierarchies.

Involves a subclass redefining a method inherited from its superclass.

The method signature (name and parameter list) must be the same in both the subclass and superclass.

// Java program to demonstrate

// method overriding in java

// Base Class

class Parent {

void show() { System.out.println("Parent's show()"); }

}

// Inherited class

class Child extends Parent {

// This method overrides show() of Parent

@Override void show()

{

System.out.println("Child's show()");

}

}

// Driver class

class Main {

public static void main(String[] args)

{

// If a Parent type reference refers

// to a Parent object, then Parent's

// show is called

Parent obj1 = new Parent();

obj1.show();

// If a Parent type reference refers

// to a Child object Child's show()

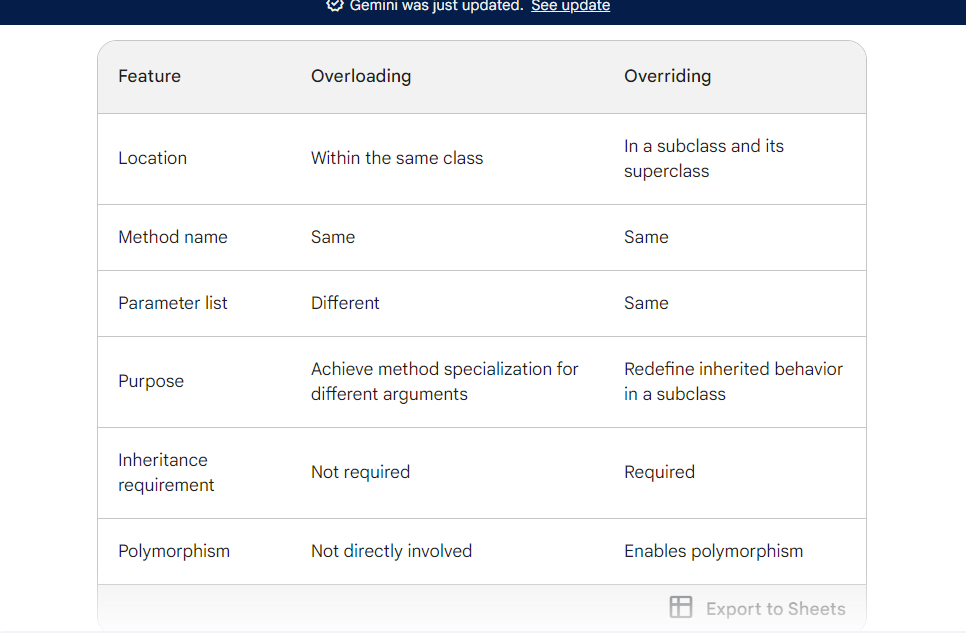
// is called. This is called RUN TIME

// POLYMORPHISM.

Parent obj2 = new Child();

obj2.show();

}

}

10. Static vs Public

In the example above, we created a static method, which means that it can be accessed without creating an object of the class, unlike public, which can only be accessed by objects:



11. Using Multiple Classes:

In this example, we have created two files in the same directory:

Main.java

Second.java

Compile both file:

C:\Users\Your Name>javac Main.java

C:\Users\Your Name>javac Second.java

Run the Second.java file:

C:\Users\Your Name>java Second

Q. Imagine I want to develop app like inshorts or I want to implement Google history where latest watched history will appear on top what data structure will be ideal to choose to implement it and why

Ans : Stack

27-02-2024

Encapsulation in Java

In the realm of Java, encapsulation is a fundamental principle of object-oriented programming (OOP) that combines data (attributes or variables) and the methods (functions) that operate on that data into a single unit called a class

Benefits of Encapsulation

Data Protection: Encapsulation safeguards data (members) by declaring them as private within the class

Controlled Access: You grant controlled access to the data through public or protected methods, typically known as getters and setters.

Retrieve data:

Update data:

Data Integrity: With control over how data is accessed and modified, you can implement validation and business logic within your getters and setters

Achieving Encapsulation:

Declare class members (data) as private:

Provide getters and setters:

public class Person {

private String name;

private int age;

public Person(String name, int age) {

this.name = name;

this.age = age;

}

public String getName() {

return name;

}

public void setName(String name) {

// Add validation logic if necessary

this.name = name;

}

public int getAge() {

return age;

}

public void setAge(int age) {

// Add validation logic if necessary

this.age = age;

}

}

2.Abstraction??

Abstraction is a fundamental concept in object-oriented programming (OOP) that focuses on providing essential details (what) and hiding implementation details (how). In

. Abstract Classes:

An abstract class is a class **that cannot be instantiated** directly. It acts as a blueprint for defining the common behavior and interface that its subclasses must adhere to.

Abstract classes can contain both abstract methods (methods without a body) and concrete methods (methods with a defined implementation).

Subclasses inherit from an abstract class and must implement the abstract methods before they can be instantiated.

Abstract method?

In Java, an abstract method is a specific type of method declared within an abstract class.

The abstract method does not have a body or curly braces {}. This signifies that the method's implementation is not provided in the abstract class.

3.Interfaces in Java?

An interface in Java is a fundamental blueprint that outlines the behavior a class must adhere to. It serves as a contract, specifying the methods a class must implement without providing any implementation details

Interfaces define method signatures (what the method does) but not the implementation (how it does it).

Classes can implement multiple interfaces.

Interface implementation enforces a contract, ensuring a class provides the necessary behavior.

Interfaces promote abstraction, polymorphism, loose coupling, and code reusability.

No, you cannot directly create an object of an interface class in Java. This is a fundamental principle of interfaces: they exist to define a contract of behavior, not to provide concrete implementations.

Anonymous Inner Class in Java:

In Java, an anonymous inner class is a special type of inner class that does not have a separate name. It is declared and instantiated within a single expression, usually used where a one-time implementation of an interface or a subclass is needed

What is Constructor Chaining?

Constructor chaining is a mechanism in Java that allows a constructor to call another constructor within the same class.

28/02/24

1. What is a destructor in C++?

In C++, a destructor is a special member function that is automatically called when an object goes out of scope or is explicitly destroyed using the delete or delete[] operator.

// Destructor to deallocate memory

~Resource()

{

delete[] data; // Deallocate memory

std::cout << "Resource object destroyed." << std::endl;

}

1. what is a destructor ? in java

, Java doesn't have destructor

a process called garbage collection to manage memory allocation and deallocation.

Destructors:

Special methods in certain languages that are automatically called when an object goes out of scope or is explicitly destroyed.

Their primary purpose is to release resources associated with the object, like memory.

Key Points about Garbage Collection:

Non-deterministic: can vary based on memory usage patterns and JVM settings.

Understand that finalize() is not a replacement for destructors.

1. Overriding is a mechanism used to achieve polymorphism, allowing for specialized behavior within inheritance hierarchies
2. can we overload the constructor and destructor?

In Java, you can overload constructors but not destructors. As mentioned earlier, Java does not have destructors in the same sense as languages like C++. However, you can achieve similar functionality by implementing the `AutoCloseable` interface and using try-with-resources or by manually releasing resources in a `finally` block.

Here's an example demonstrating constructor overloading in Java:

```java

public class MyClass {

private int value;

// Constructor with no parameters

public MyClass() {

this.value = 0;

}

// Constructor with one parameter

public MyClass(int value) {

this.value = value;

}

// Getter method

public int getValue() {

return value;

}

public static void main(String[] args) {

MyClass obj1 = new MyClass(); // Calls the first constructor

MyClass obj2 = new MyClass(10); // Calls the second constructor

System.out.println("Value of obj1: " + obj1.getValue()); // Output: Value of obj1: 0

System.out.println("Value of obj2: " + obj2.getValue()); // Output: Value of obj2: 10

}

}

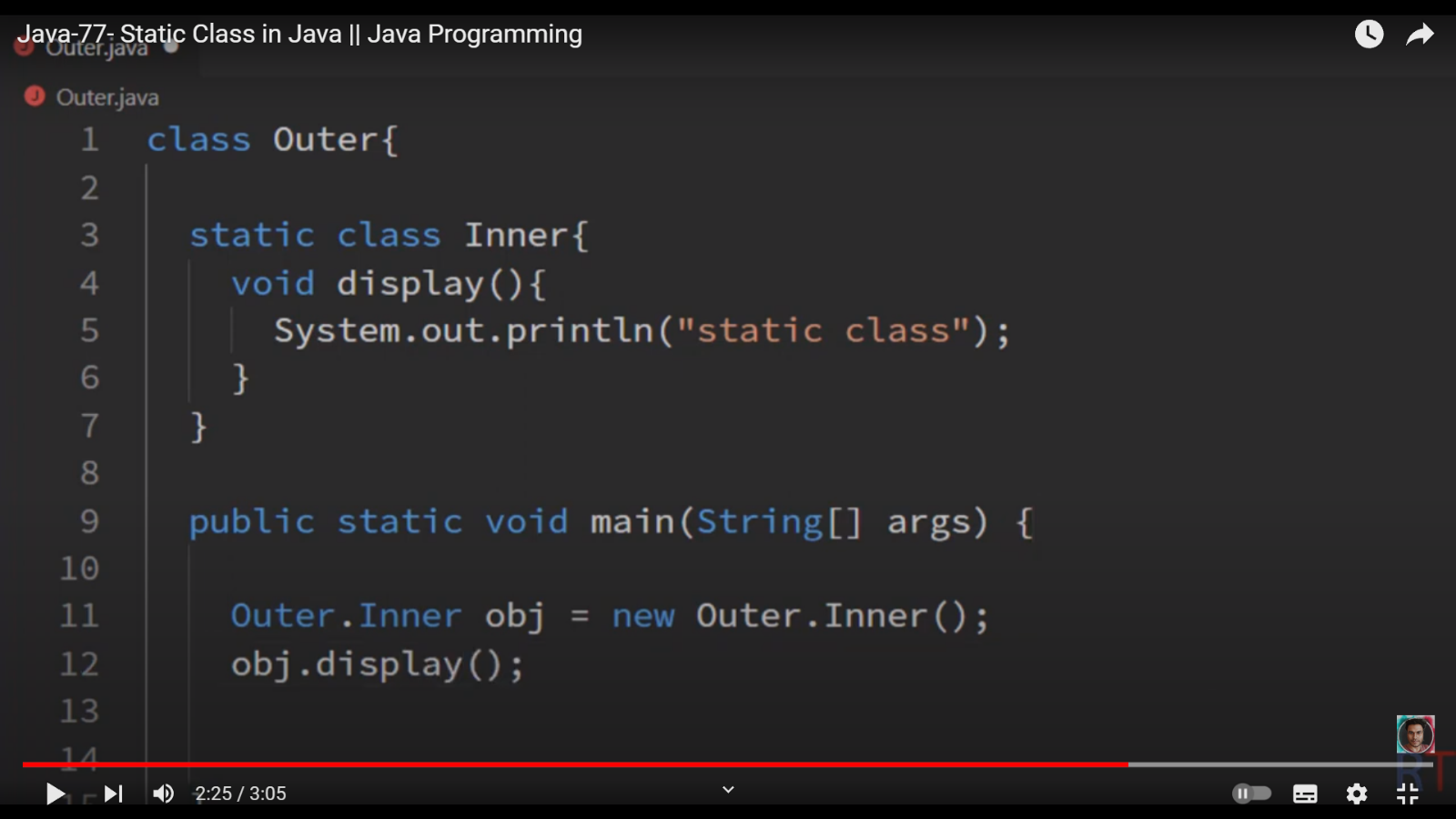
```

In this example, `MyClass` has two constructors: one with no parameters and one with a single `int` parameter. Depending on how the object is instantiated, the appropriate constructor will be called.

As for destructors, Java does not allow explicit destructors. Instead, Java relies on garbage collection to reclaim memory when objects are no longer referenced. If you need to perform cleanup actions before an object is garbage collected, you can implement the `finalize()` method, but its usage is discouraged due to its unpredictable nature.

1. What is the static class?

Ans: Cannot be instantiated: Unlike regular classes, you cannot create objects of a static **class directly**. This means you cannot use the new keyword to instantiate them.



6. Singleton Class in Java

A singleton class in Java adheres to the Singleton design pattern, ensuring that only one instance of the class exists throughout the application. This instance acts as a central access point for its functionality, offering several benefits.

29-02-2024

1.Types of Error ?

Syntax Errors:

public class SyntaxErrorExample {

public static void main(String[] args) {

int x = 10

System.out.println("The value of x is: " + x);

}

}

**Runtime Errors:**

public class RuntimeErrorExample {

public static void main(String[] args) {

int[] numbers = {1, 2, 3};

System.out.println("Element at index 3: " + numbers[3]);

}

**}**

Logical Errors:

public class LogicalErrorExample {

public static void main(String[] args) {

int x = 5;

int y = 2;

int result = x / y; // Incorrect calculation

System.out.println("Result of division: " + result);

}

}