NAME – VINAY SANDIP DHAKE

AF CODE – AF04955282

Java Basics and OOPs Assignment

1. What is Java? Explain its features.

Java is a high-level, object-oriented programming language developed by Sun Microsystems (now owned by Oracle). It is platform-independent, secure, and widely used for building web and mobile applications.

KEY FEATURES:

1. Object-Oriented : Everything in Java is treated as an object, allowing for modular programs and reusable code.

2. Platform-Independent : Java code is compiled into bytecode, which can run on any platform with a JVM.

3. Simple and Familiar : Java's syntax is clean and easy to understand, especially for those familiar with C or C++.

4. Secure : Java has built-in security features like bytecode verification, sandboxing, and runtime security checks.

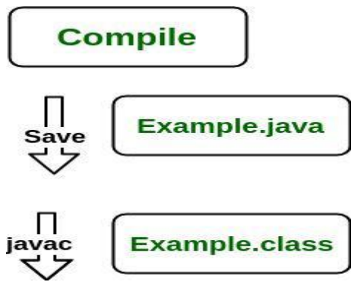
5. Robust : It handles errors gracefully with strong memory management and exception handling.

6. Multithreaded : Java allows the development of programs that can perform multiple tasks at once.

2. Explain the Java program execution process.

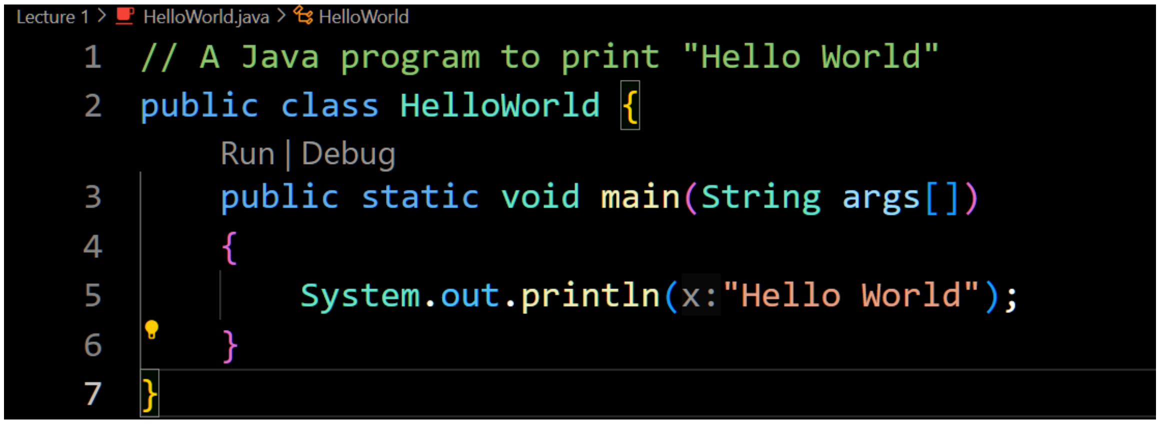
The JDK enables the development and execution of Java programs. Consider the following process:

- **Java Source File (e.g., Example.java)**: You write the Java program in a source file.
- **Compilation:** The source file is compiled by the Java Compiler (part of JDK) into bytecode, which is stored in a .class file (e.g., Example.class).
- **Execution:** The bytecode is executed by the JVM (Java Virtual Machine), which interprets the bytecode and runs the Java program.

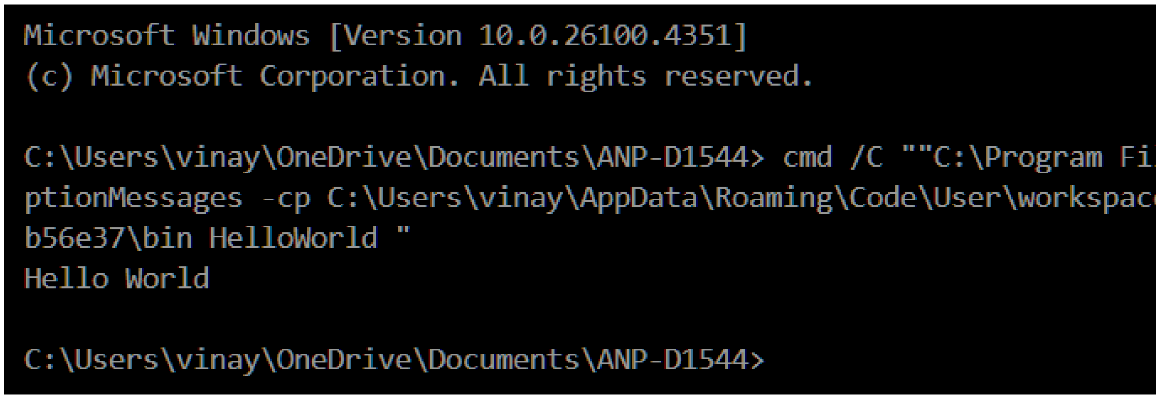


3. Write a simple Java program to display 'Hello World'.

```
public class HelloWorld {
   public static void main(String[] args) {
      System.out.println("Hello World");
   }
}
```



```
Lecture 1 > HelloWorld.java > HelloWorld
// A Java program to print "Hello World"
public class HelloWorld {
       Run | Debug
       public static void main(String args[])
       {
           System.out.println(x:"Hello World");
       }
}
```

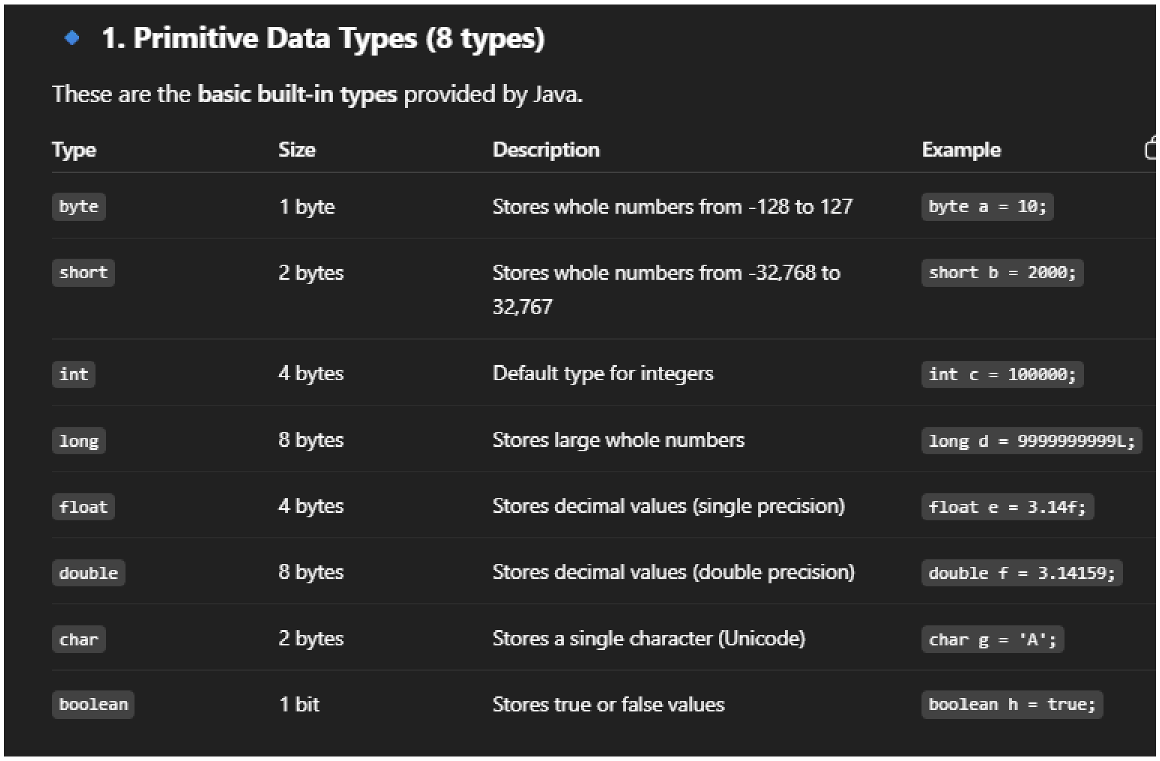


```
Microsoft Windows [Version 10.0.26100.4351]
(c) Microsoft Corporation. All rights reserved.

C:\Users\vinay\OneDrive\Documents\ANP-D1544> cmd /C ""C:\Program Fi
ptionMessages -cp C:\Users\vinay\AppData\Roaming\Code\User\workspac
b56e37\bin HelloWorld "
Hello World

C:\Users\vinay\OneDrive\Documents\ANP-D1544>
```

4. What are data types in Java? List and explain them.

In Java, **data types define the type of data** a variable can hold. They are **essential** because Java is a **strongly typed language**, meaning every variable must be declared with a data type.

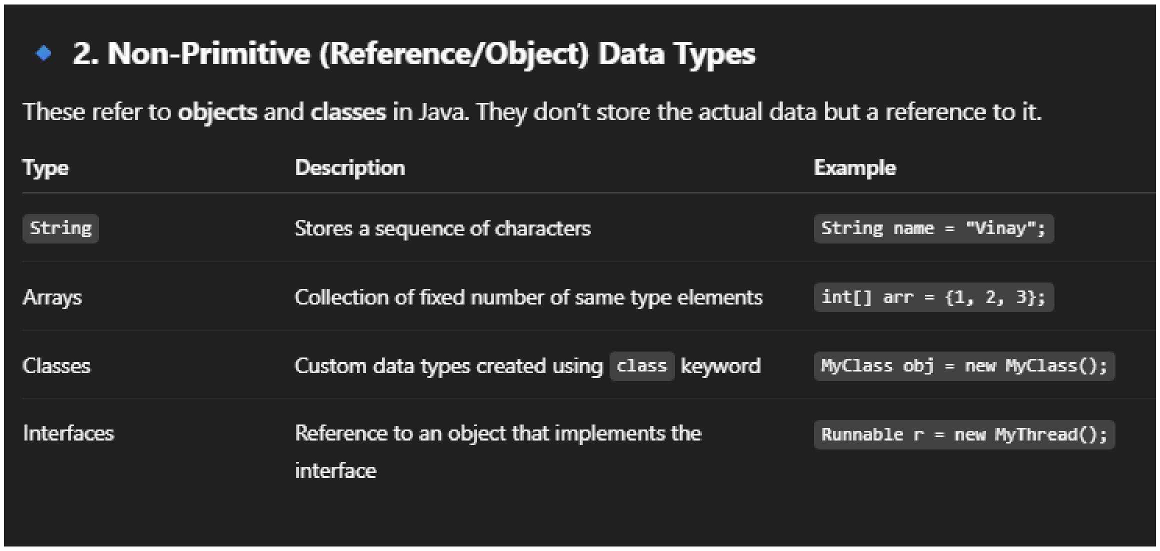
Java data types are broadly divided into **two categories**:



🔹 1. Primitive Data Types (8 types)

These are the **basic built-in types** provided by Java.

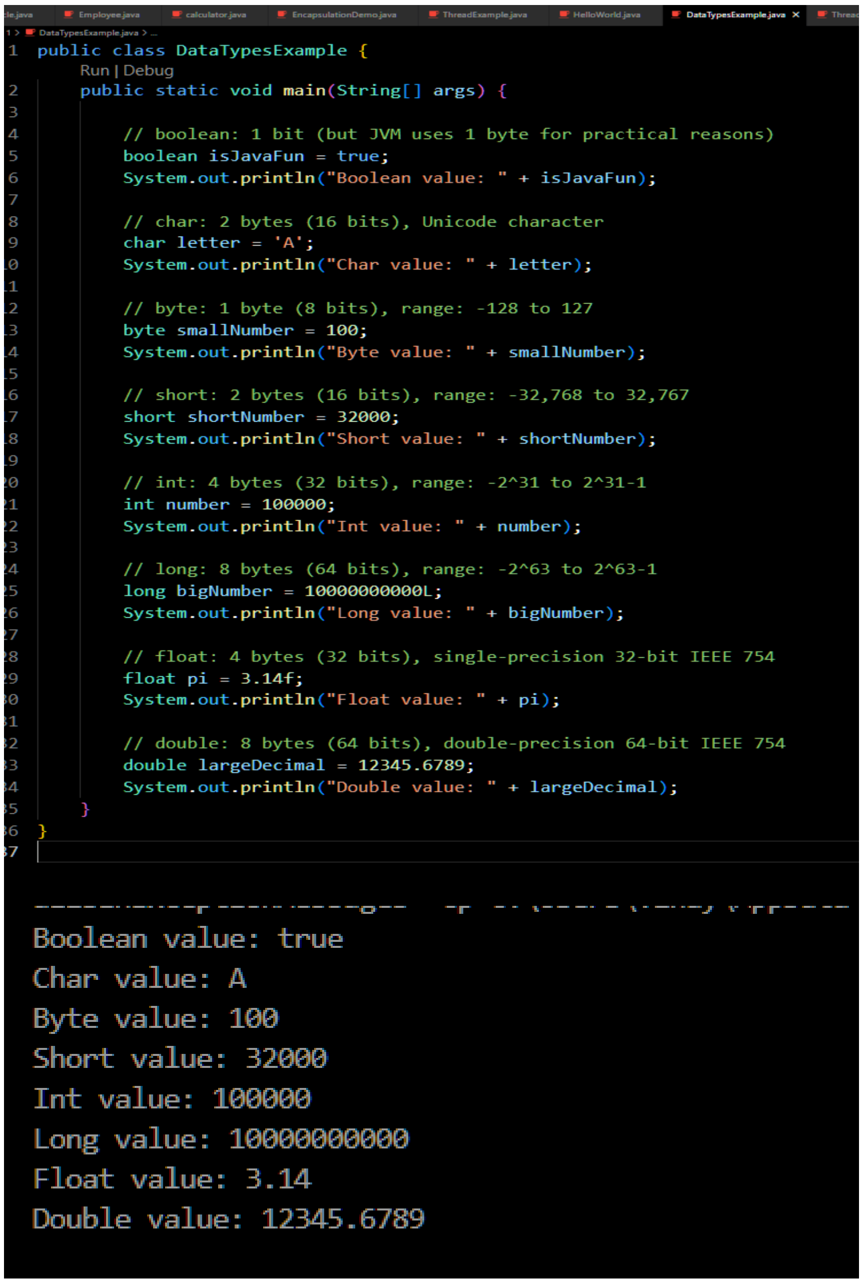
Type	Size	Description	Example
`byte`	1 byte	Stores whole numbers from -128 to 127	`byte a = 10;`
`short`	2 bytes	Stores whole numbers from -32,768 to 32,767	`short b = 2000;`
`int`	4 bytes	Default type for integers	`int c = 100000;`
`long`	8 bytes	Stores large whole numbers	`long d = 9999999999L;`
`float`	4 bytes	Stores decimal values (single precision)	`float e = 3.14f;`
`double`	8 bytes	Stores decimal values (double precision)	`double f = 3.14159;`
`char`	2 bytes	Stores a single character (Unicode)	`char g = 'A';`
`boolean`	1 bit	Stores true or false values	`boolean h = true;`



🔹 2. Non-Primitive (Reference/Object) Data Types

These refer to **objects** and **classes** in Java. They don't store the actual data but a reference to it.

Type	Description	Example
`String`	Stores a sequence of characters	`String name = "Vinay";`
Arrays	Collection of fixed number of same type elements	`int[] arr = {1, 2, 3};`
Classes	Custom data types created using `class` keyword	`MyClass obj = new MyClass();`
Interfaces	Reference to an object that implements the interface	`Runnable r = new MyThread();`



```java
public class DataTypesExample {
    Run | Debug
    public static void main(String[] args) {

        // boolean: 1 bit (but JVM uses 1 byte for practical reasons)
        boolean isJavaFun = true;
        System.out.println("Boolean value: " + isJavaFun);

        // char: 2 bytes (16 bits), Unicode character
        char letter = 'A';
        System.out.println("Char value: " + letter);

        // byte: 1 byte (8 bits), range: -128 to 127
        byte smallNumber = 100;
        System.out.println("Byte value: " + smallNumber);

        // short: 2 bytes (16 bits), range: -32,768 to 32,767
        short shortNumber = 32000;
        System.out.println("Short value: " + shortNumber);

        // int: 4 bytes (32 bits), range: -2^31 to 2^31-1
        int number = 100000;
        System.out.println("Int value: " + number);

        // long: 8 bytes (64 bits), range: -2^63 to 2^63-1
        long bigNumber = 10000000000L;
        System.out.println("Long value: " + bigNumber);

        // float: 4 bytes (32 bits), single-precision 32-bit IEEE 754
        float pi = 3.14f;
        System.out.println("Float value: " + pi);

        // double: 8 bytes (64 bits), double-precision 64-bit IEEE 754
        double largeDecimal = 12345.6789;
        System.out.println("Double value: " + largeDecimal);
    }
}
```

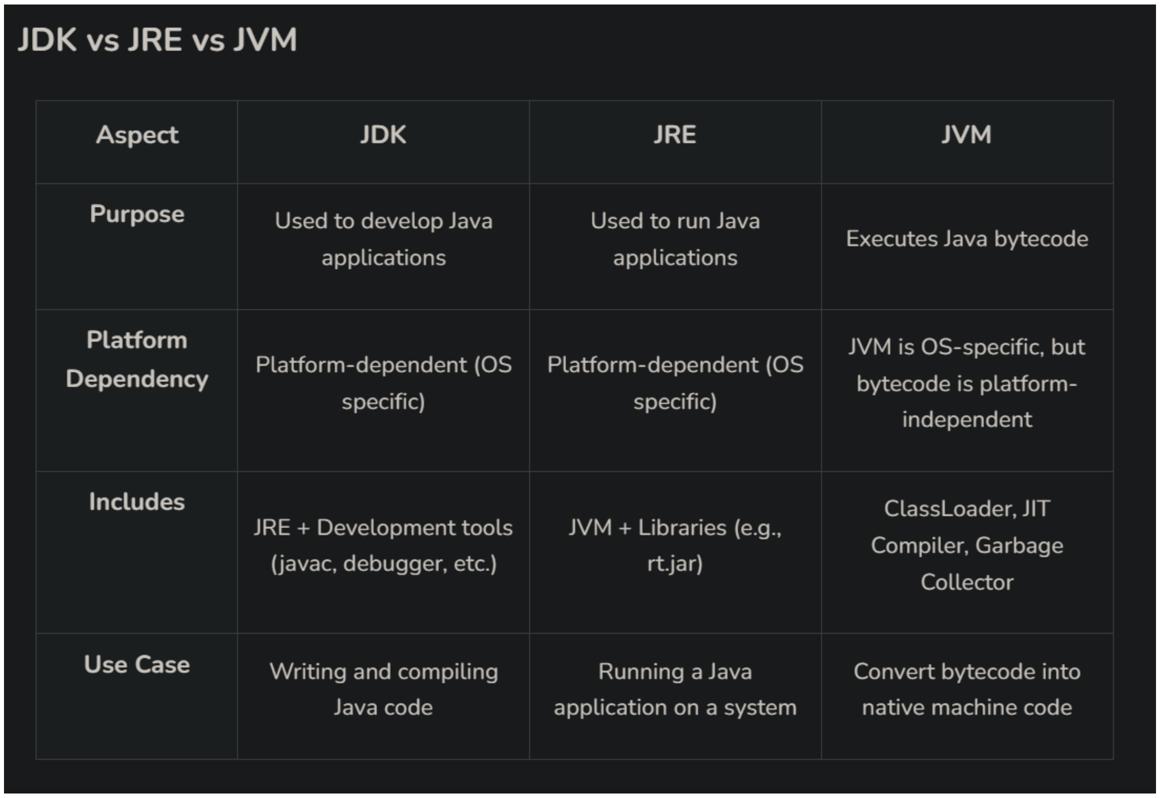
```
Boolean value: true
Char value: A
Byte value: 100
Short value: 32000
Int value: 100000
Long value: 10000000000
Float value: 3.14
Double value: 12345.6789
```

5. What is the difference between JDK, JRE, and JVM?

JDK: Java Development Kit is a software development environment used for developing Java applications and applets.

JRE: JRE stands for Java Runtime Environment, and it provides an environment to run only the Java program onto the system.

JVM: JVM stands for Java Virtual Machine and is responsible for executing the Java program.

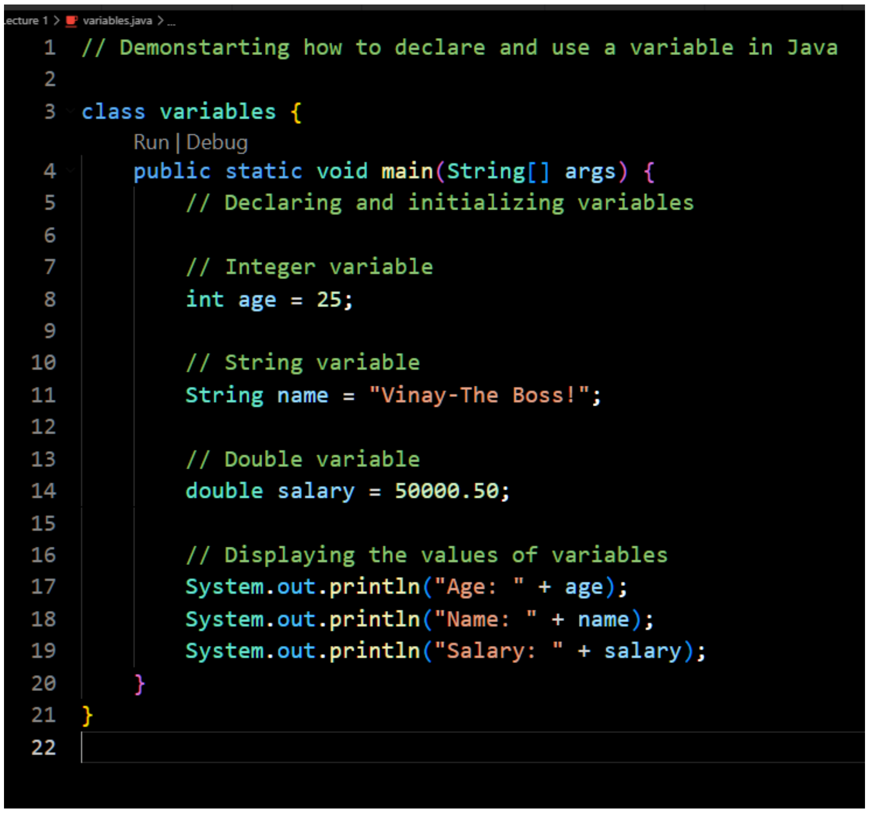


JDK vs JRE vs JVM

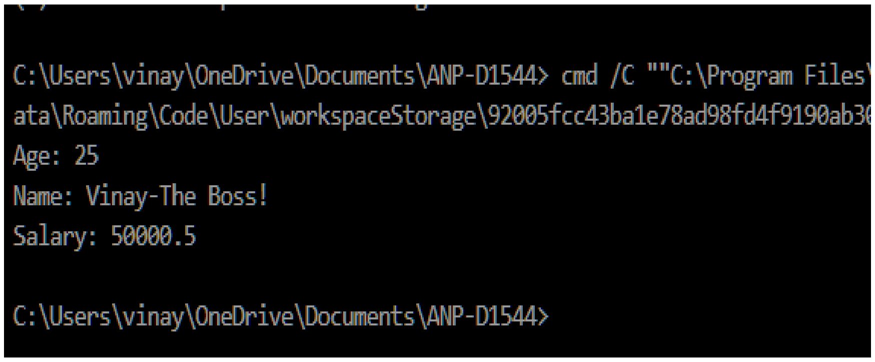
Aspect	JDK	JRE	JVM
Purpose	Used to develop Java applications	Used to run Java applications	Executes Java bytecode
Platform Dependency	Platform-dependent (OS specific)	Platform-dependent (OS specific)	JVM is OS-specific, but bytecode is platform-independent
Includes	JRE + Development tools (javac, debugger, etc.)	JVM + Libraries (e.g., rt.jar)	ClassLoader, JIT Compiler, Garbage Collector
Use Case	Writing and compiling Java code	Running a Java application on a system	Convert bytecode into native machine code

6. What are variables in Java? Explain with examples.

In Java, variables are containers that store data in memory, it defines how data is stored, accessed, and manipulated.



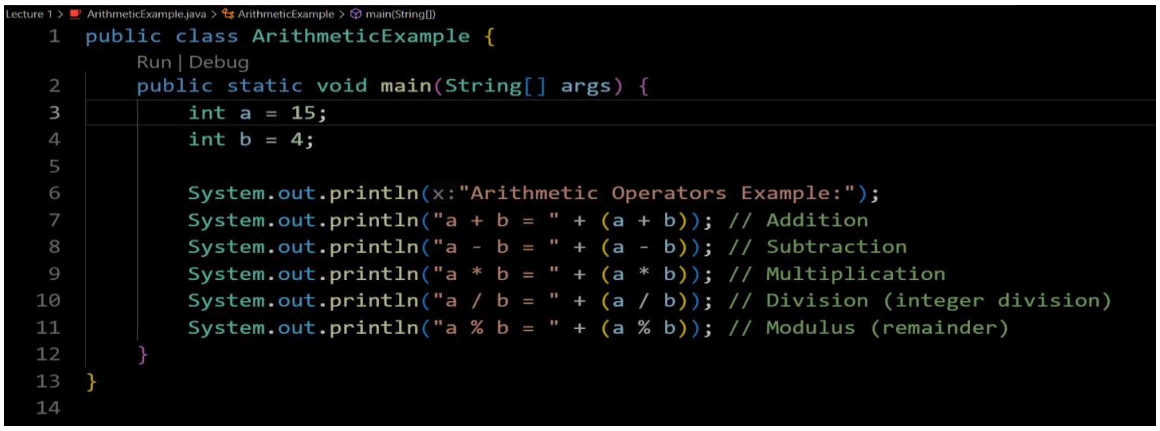
```java
// Demonstarting how to declare and use a variable in Java

class variables {
    public static void main(String[] args) {
        // Declaring and initializing variables

        // Integer variable
        int age = 25;

        // String variable
        String name = "Vinay-The Boss!";

        // Double variable
        double salary = 50000.50;

        // Displaying the values of variables
        System.out.println("Age: " + age);
        System.out.println("Name: " + name);
        System.out.println("Salary: " + salary);
    }
}

```



```
C:\Users\vinay\OneDrive\Documents\ANP-D1544> cmd /C ""C:\Program Files
ata\Roaming\Code\User\workspaceStorage\92005fcc43ba1e78ad98fd4f9190ab3
Age: 25
Name: Vinay-The Boss!
Salary: 50000.5

C:\Users\vinay\OneDrive\Documents\ANP-D1544>
```

7. What are the different types of operators in Java?

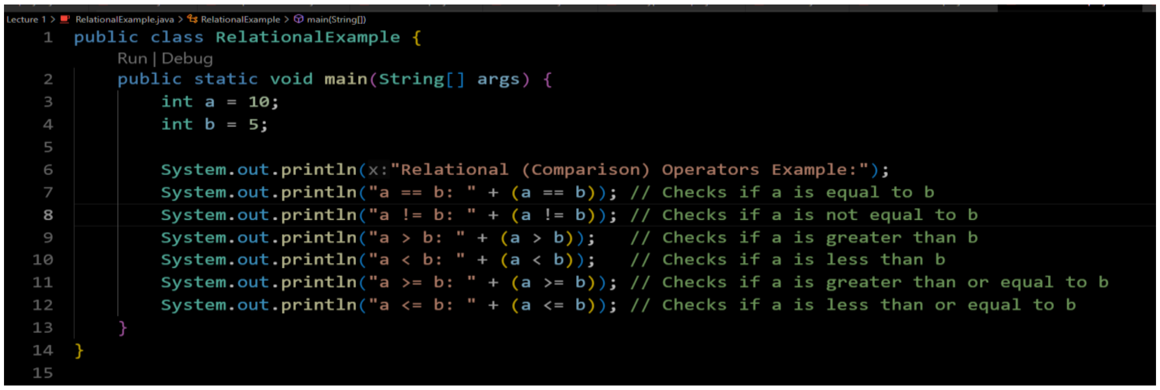
ARITHMETIC OPERATORS:



Lecture 1 > ArithmeticExample.java > ArithmeticExample > main(String[])

```java
public class ArithmeticExample {
    Run | Debug
    public static void main(String[] args) {
        int a = 15;
        int b = 4;

        System.out.println(x:"Arithmetic Operators Example:");
        System.out.println("a + b = " + (a + b)); // Addition
        System.out.println("a - b = " + (a - b)); // Subtraction
        System.out.println("a * b = " + (a * b)); // Multiplication
        System.out.println("a / b = " + (a / b)); // Division (integer division)
        System.out.println("a % b = " + (a % b)); // Modulus (remainder)
    }
}
```

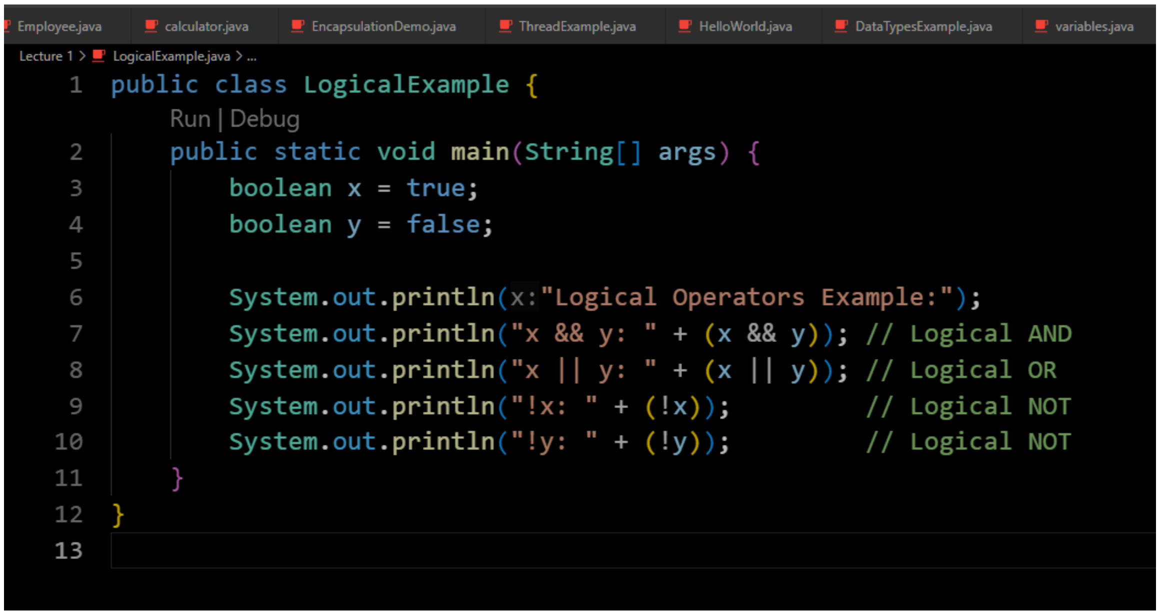
RELATIONAL OPERATORS:



Lecture 1 > RelationalExample.java > RelationalExample > main(String[])

```java
public class RelationalExample {
    Run | Debug
    public static void main(String[] args) {
        int a = 10;
        int b = 5;

        System.out.println(x:"Relational (Comparison) Operators Example:");
        System.out.println("a == b: " + (a == b)); // Checks if a is equal to b
        System.out.println("a != b: " + (a != b)); // Checks if a is not equal to b
        System.out.println("a > b: " + (a > b));   // Checks if a is greater than b
        System.out.println("a < b: " + (a < b));   // Checks if a is less than b
        System.out.println("a >= b: " + (a >= b)); // Checks if a is greater than or equal to b
        System.out.println("a <= b: " + (a <= b)); // Checks if a is less than or equal to b
    }
}
```

LOGICAL OPERATORS

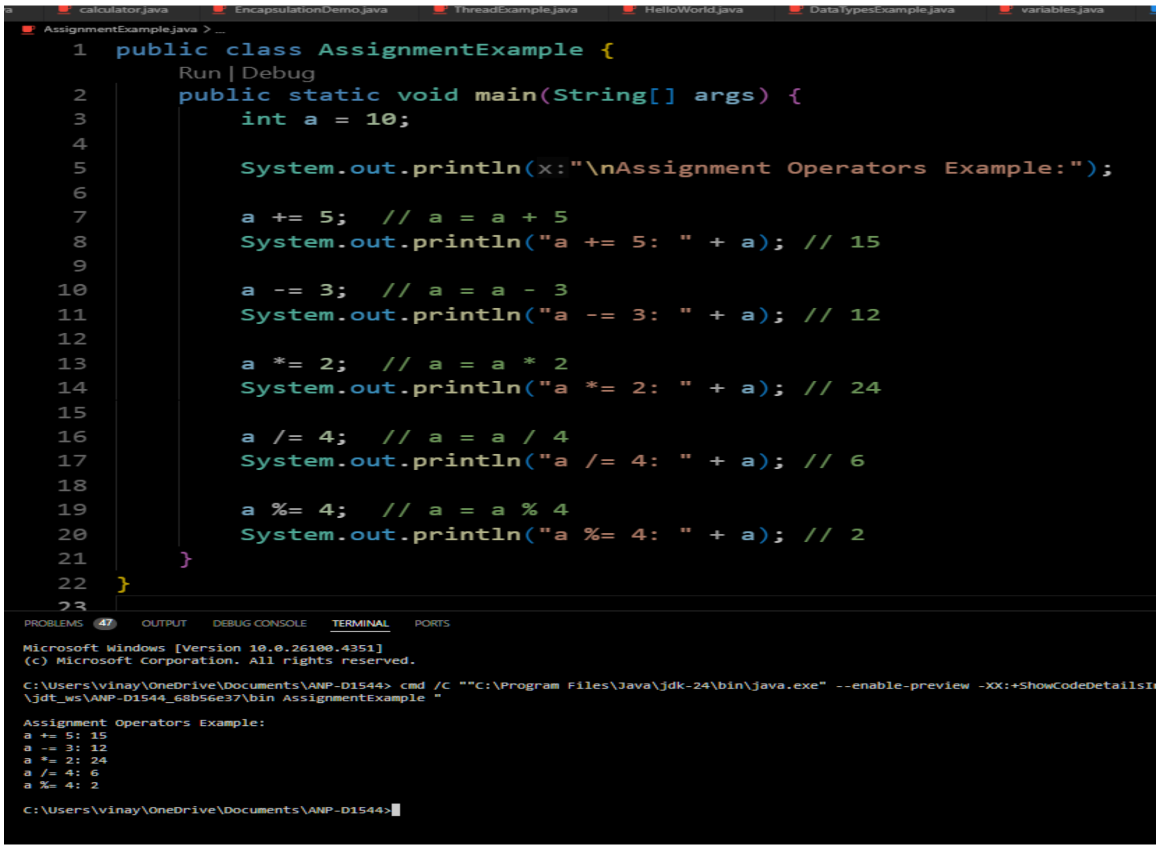


Employee.java | calculator.java | EncapsulationDemo.java | ThreadExample.java | HelloWorld.java | DataTypesExample.java | variables.java

Lecture 1 > LogicalExample.java > ...

```java
public class LogicalExample {
    Run | Debug
    public static void main(String[] args) {
        boolean x = true;
        boolean y = false;

        System.out.println(x:"Logical Operators Example:");
        System.out.println("x && y: " + (x && y)); // Logical AND
        System.out.println("x || y: " + (x || y)); // Logical OR
        System.out.println("!x: " + (!x));         // Logical NOT
        System.out.println("!y: " + (!y));         // Logical NOT
    }
}
```

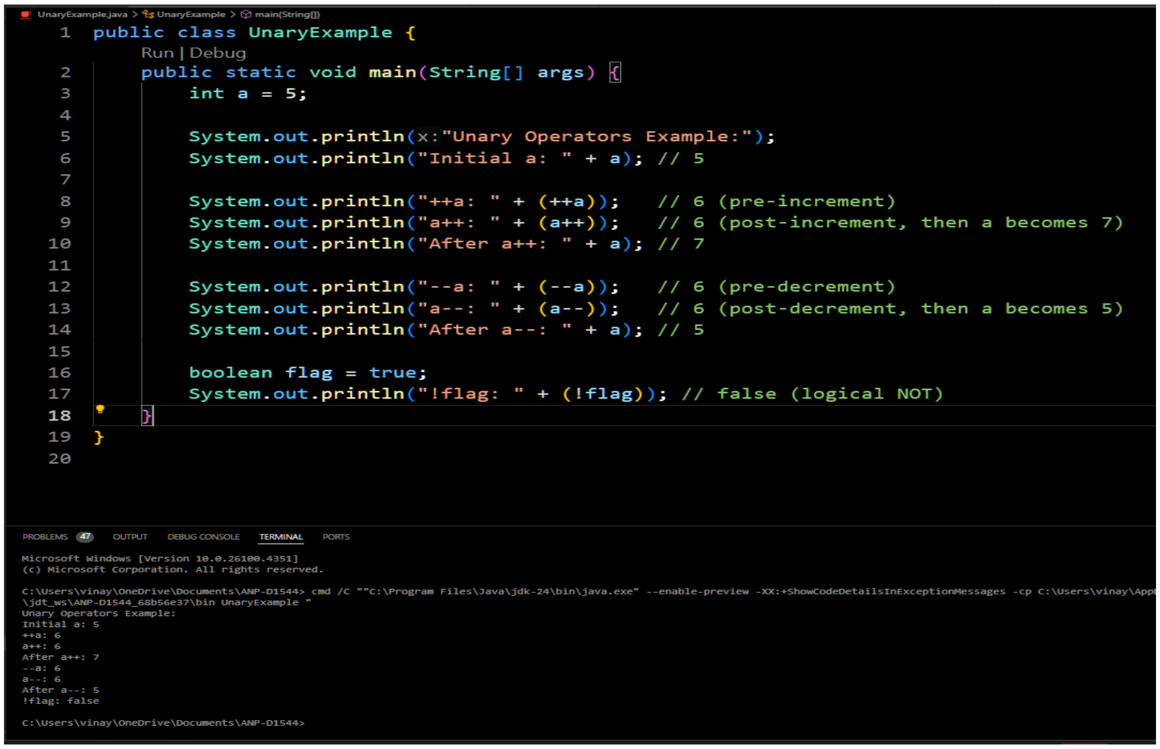
ASSIGNMENT OPERATORS:



```java
public class AssignmentExample {
    public static void main(String[] args) {
        int a = 10;

        System.out.println(x:"\nAssignment Operators Example:");

        a += 5;  // a = a + 5
        System.out.println("a += 5: " + a); // 15

        a -= 3;  // a = a - 3
        System.out.println("a -= 3: " + a); // 12

        a *= 2;  // a = a * 2
        System.out.println("a *= 2: " + a); // 24

        a /= 4;  // a = a / 4
        System.out.println("a /= 4: " + a); // 6

        a %= 4;  // a = a % 4
        System.out.println("a %= 4: " + a); // 2
    }
}
```

```
Microsoft Windows [Version 10.0.26100.4351]
(c) Microsoft Corporation. All rights reserved.

C:\Users\vinay\OneDrive\Documents\ANP-D1544> cmd /C ""C:\Program Files\Java\jdk-24\bin\java.exe" --enable-preview -XX:+ShowCodeDetailsI
\jdt_ws\ANP-D1544_68b56e37\bin AssignmentExample "

Assignment Operators Example:
a += 5: 15
a -= 3: 12
a *= 2: 24
a /= 4: 6
a %= 4: 2

C:\Users\vinay\OneDrive\Documents\ANP-D1544>
```

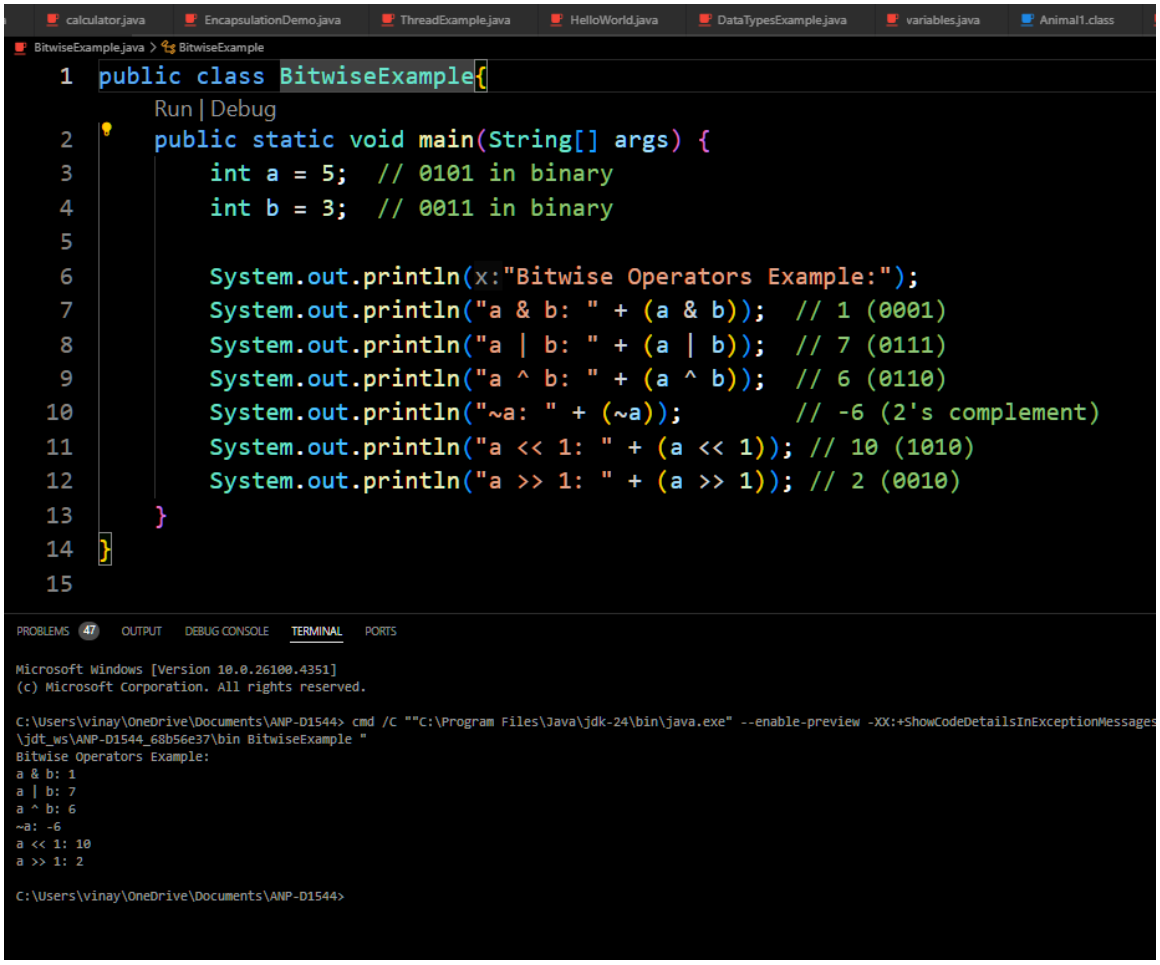
UNARY OPERATORS:



```java
public class UnaryExample {
    public static void main(String[] args) {
        int a = 5;

        System.out.println(x:"Unary Operators Example:");
        System.out.println("Initial a: " + a); // 5

        System.out.println("++a: " + (++a));    // 6 (pre-increment)
        System.out.println("a++: " + (a++));    // 6 (post-increment, then a becomes 7)
        System.out.println("After a++: " + a); // 7

        System.out.println("--a: " + (--a));    // 6 (pre-decrement)
        System.out.println("a--: " + (a--));    // 6 (post-decrement, then a becomes 5)
        System.out.println("After a--: " + a); // 5

        boolean flag = true;
        System.out.println("!flag: " + (!flag)); // false (logical NOT)
    }
}
```

```
Microsoft Windows [Version 10.0.26100.4351]
(c) Microsoft Corporation. All rights reserved.

C:\Users\vinay\OneDrive\Documents\ANP-D1544> cmd /C ""C:\Program Files\Java\jdk-24\bin\java.exe" --enable-preview -XX:+ShowCodeDetailsInExceptionMessages -cp C:\Users\vinay\App
\jdt_ws\ANP-D1544_68b56e37\bin UnaryExample "
Unary Operators Example:
Initial a: 5
++a: 6
a++: 6
After a++: 7
--a: 6
a--: 6
After a--: 5
!flag: false

C:\Users\vinay\OneDrive\Documents\ANP-D1544>
```

BITWISE OPERATOR:



```java
public class BitwiseExample{
    public static void main(String[] args) {
        int a = 5;  // 0101 in binary
        int b = 3;  // 0011 in binary

        System.out.println(x:"Bitwise Operators Example:");
        System.out.println("a & b: " + (a & b));  // 1 (0001)
        System.out.println("a | b: " + (a | b));  // 7 (0111)
        System.out.println("a ^ b: " + (a ^ b));  // 6 (0110)
        System.out.println("~a: " + (~a));         // -6 (2's complement)
        System.out.println("a << 1: " + (a << 1)); // 10 (1010)
        System.out.println("a >> 1: " + (a >> 1)); // 2 (0010)
    }
}
```

```
Microsoft Windows [Version 10.0.26100.4351]
(c) Microsoft Corporation. All rights reserved.

C:\Users\vinay\OneDrive\Documents\ANP-D1544> cmd /C ""C:\Program Files\Java\jdk-24\bin\java.exe" --enable-preview -XX:+ShowCodeDetailsInExceptionMessages
\jdt_ws\ANP-D1544_68b56e37\bin BitwiseExample "
Bitwise Operators Example:
a & b: 1
a | b: 7
a ^ b: 6
~a: -6
a << 1: 10
a >> 1: 2

C:\Users\vinay\OneDrive\Documents\ANP-D1544>
```

8. Explain control statements in Java (if, if-else, switch).

1. if Statement

Executes a block of code only if a specified condition is true.

2. if-else Statement

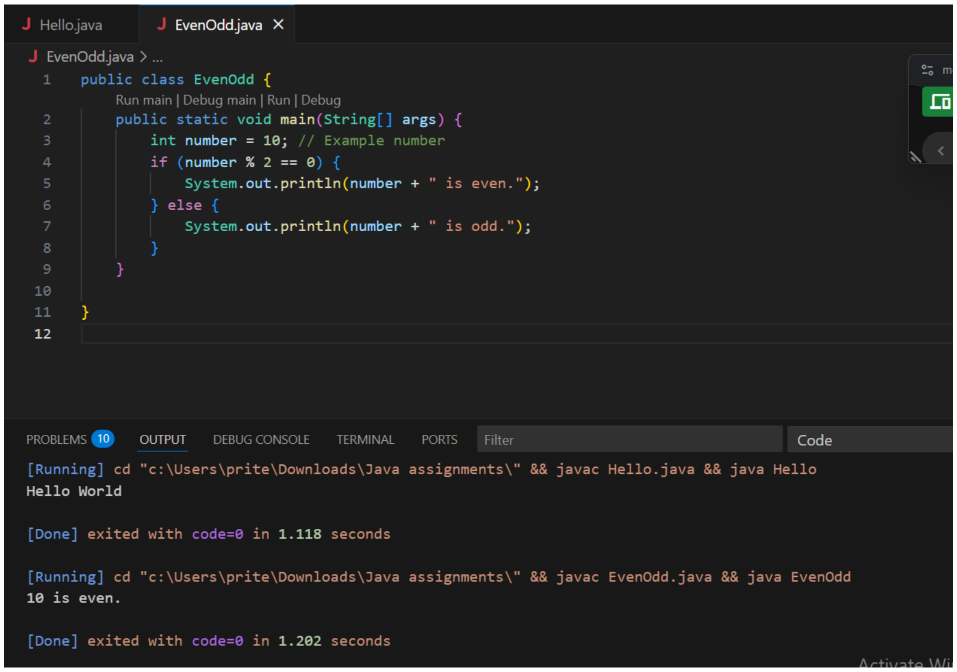
Executes one block of code if the condition is true, otherwise executes another block.

3. switch Statement

Used to select one option from multiple choices based on the value of a variable. Each option is called a "case".

9. Write a Java program to find whether a number is even or odd.

```java
import java.util.Scanner;
public class EvenOdd {
   public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int num = sc.nextInt();
        if (num % 2 == 0)
            System.out.println("Even");
        else
            System.out.println("Odd");
    }
}
```



```java
public class EvenOdd {
    public static void main(String[] args) {
        int number = 10; // Example number
        if (number % 2 == 0) {
            System.out.println(number + " is even.");
        } else {
            System.out.println(number + " is odd.");
        }
    }

}

```

```
[Running] cd "c:\Users\prite\Downloads\Java assignments\" && javac Hello.java && java Hello
Hello World

[Done] exited with code=0 in 1.118 seconds

[Running] cd "c:\Users\prite\Downloads\Java assignments\" && javac EvenOdd.java && java EvenOdd
10 is even.

[Done] exited with code=0 in 1.202 seconds
```

10. What is the difference between while and do-while loop?

while: checks condition before executing, May never execute
do-while: executes at least once, Executes at least once

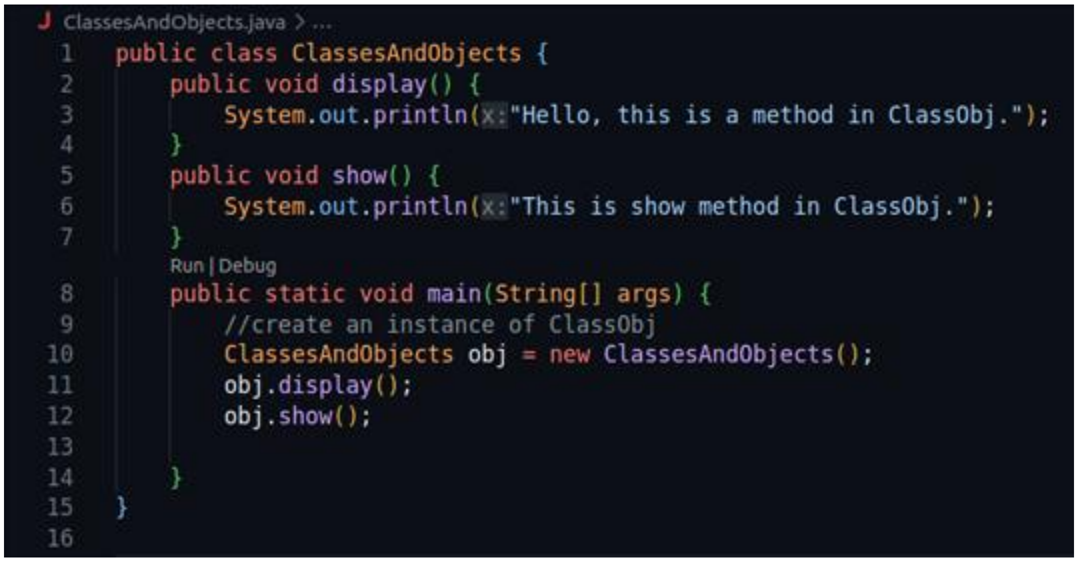
Object-Oriented Programming (OOPs)

1. What are the main principles of OOPs in Java?

- **Encapsulation**: Data hiding using classes
- **Abstraction**: Hiding implementation details
- **Inheritance**: Code reuse through subclasses
- **Polymorphism**: Many forms of methods/objects

2. What is a class and an object in Java? Give examples.

Class: Blueprint of object
Object: Instance of class
Example:
class Car { String color; void drive() {} }



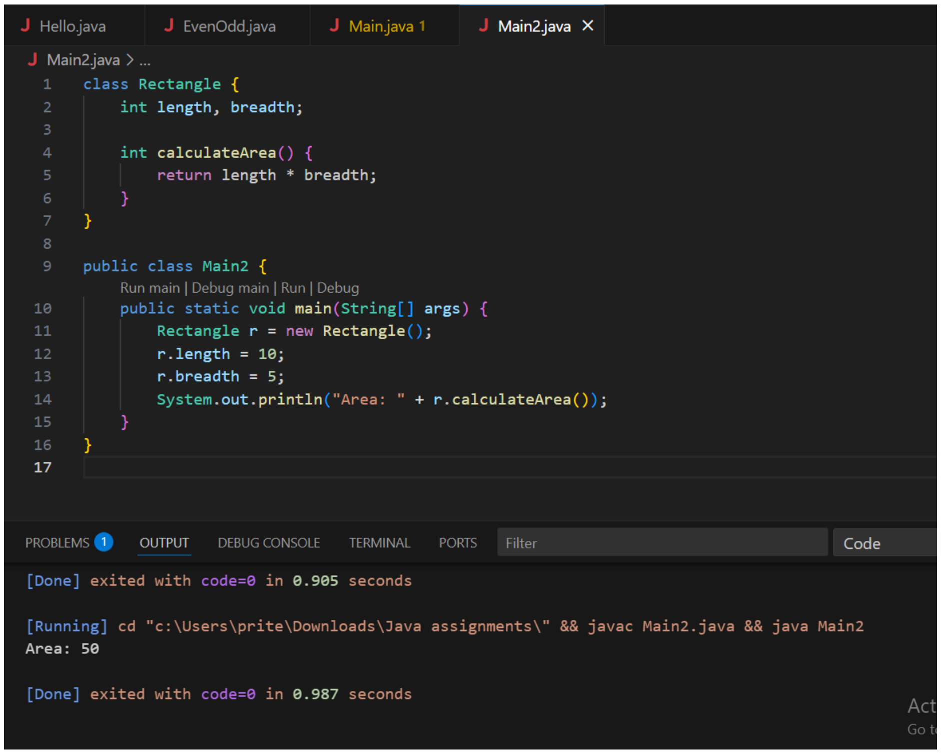
ClassesAndObjects.java > ...

```java
public class ClassesAndObjects {
    public void display() {
        System.out.println(x:"Hello, this is a method in ClassObj.");
    }
    public void show() {
        System.out.println(x:"This is show method in ClassObj.");
    }
    Run | Debug
    public static void main(String[] args) {
        //create an instance of ClassObj
        ClassesAndObjects obj = new ClassesAndObjects();
        obj.display();
        obj.show();

    }
}
```

3. Write a program using class and object to calculate area of a rectangle.

```
class Rectangle {

  int length, breadth;



  int calculateArea() {

    return length * breadth;

  }

}
```

```
public class Main {
  public static void main(String[] args) {
    Rectangle r = new Rectangle();
    r.length = 10;
    r.breadth = 5;
    System.out.println("Area: " + r.calculateArea());
  }
}
```

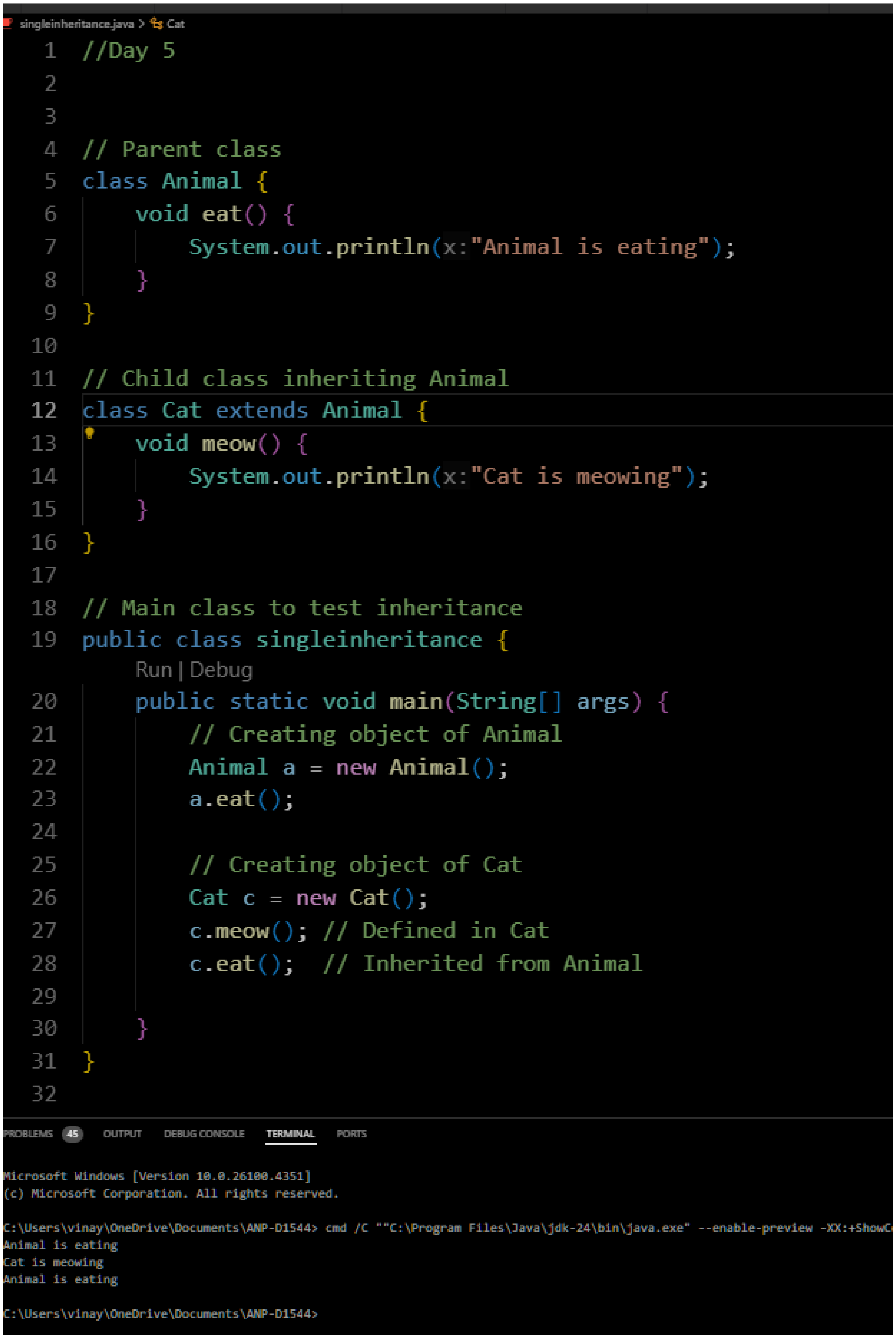


```
class Rectangle {
    int length, breadth;

    int calculateArea() {
        return length * breadth;
    }
}

public class Main2 {
    public static void main(String[] args) {
        Rectangle r = new Rectangle();
        r.length = 10;
        r.breadth = 5;
        System.out.println("Area: " + r.calculateArea());
    }
}
```

```
[Done] exited with code=0 in 0.905 seconds

[Running] cd "c:\Users\prite\Downloads\Java assignments\" && javac Main2.java && java Main2
Area: 50

[Done] exited with code=0 in 0.987 seconds
```

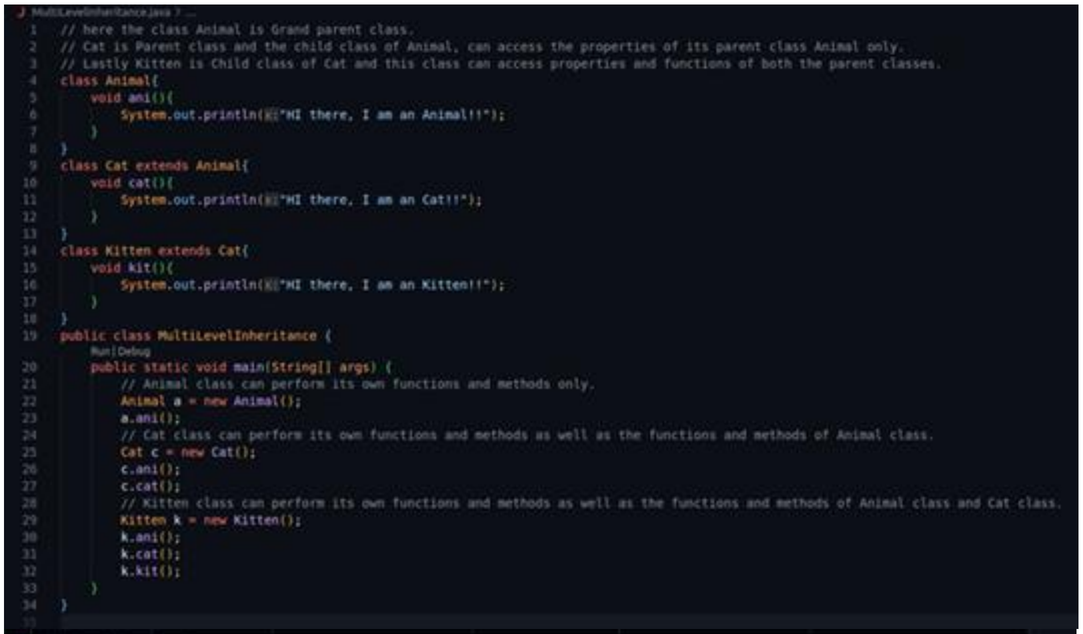
4. Explain inheritance with real-life example and Java code.

Inheritance is an OOP principle where one class (child) inherits the properties and behaviors of another class (parent). It promotes code reusability and supports method overriding.



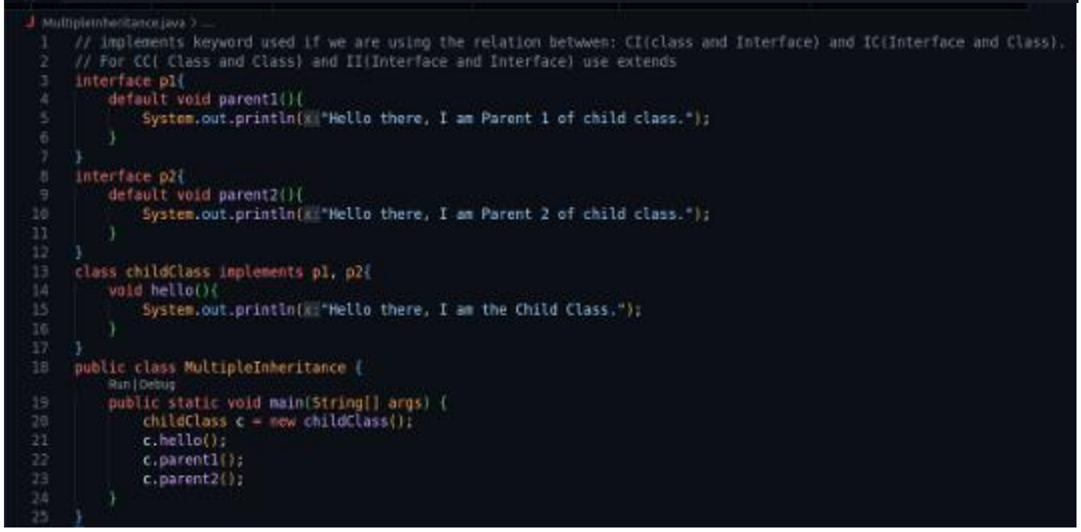
```java
//Day 5


// Parent class
class Animal {
    void eat() {
        System.out.println(x:"Animal is eating");
    }
}

// Child class inheriting Animal
class Cat extends Animal {
    void meow() {
        System.out.println(x:"Cat is meowing");
    }
}

// Main class to test inheritance
public class singleinheritance {
    Run | Debug
    public static void main(String[] args) {
        // Creating object of Animal
        Animal a = new Animal();
        a.eat();

        // Creating object of Cat
        Cat c = new Cat();
        c.meow(); // Defined in Cat
        c.eat();  // Inherited from Animal

    }
}

```

```
Microsoft Windows [Version 10.0.26100.4351]
(c) Microsoft Corporation. All rights reserved.

C:\Users\vinay\OneDrive\Documents\ANP-D1544> cmd /C ""C:\Program Files\Java\jdk-24\bin\java.exe" --enable-preview -XX:+ShowC
Animal is eating
Cat is meowing
Animal is eating

C:\Users\vinay\OneDrive\Documents\ANP-D1544>
```



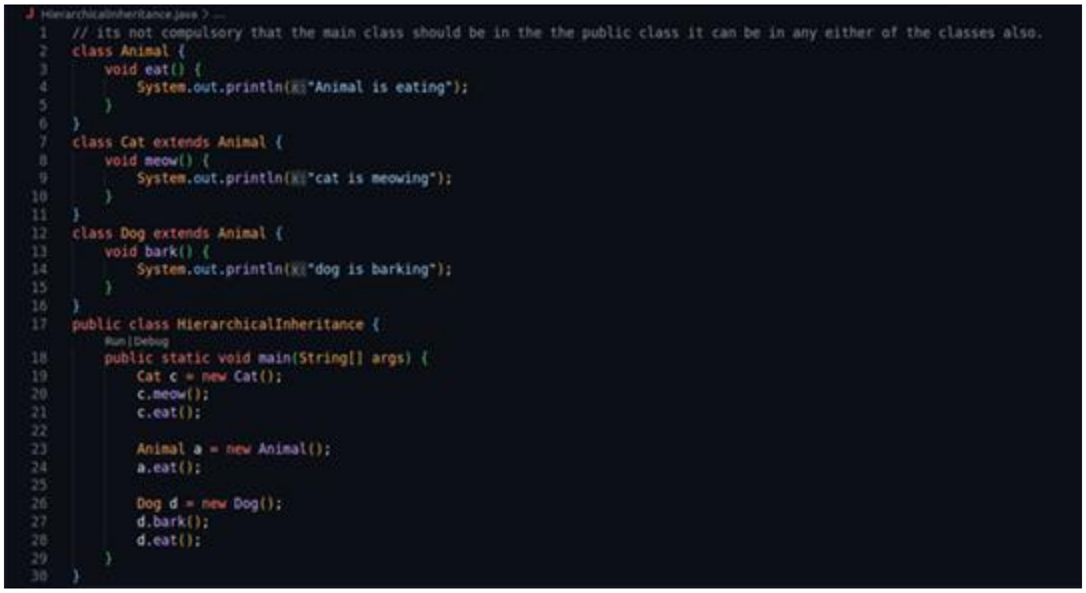
MultiLevelInheritance.java > ...

```java
// here the class Animal is Grand parent class.
// Cat is Parent class and the child class of Animal, can access the properties of its parent class Animal only.
// Lastly Kitten is Child class of Cat and this class can access properties and functions of both the parent classes.
class Animal{
    void ani(){
        System.out.println("HI there, I am an Animal!!");
    }
}
class Cat extends Animal{
    void cat(){
        System.out.println("HI there, I am an Cat!!");
    }
}
class Kitten extends Cat{
    void kit(){
        System.out.println("HI there, I am an Kitten!!");
    }
}
public class MultiLevelInheritance {
    public static void main(String[] args) {
        // Animal class can perform its own functions and methods only.
        Animal a = new Animal();
        a.ani();
        // Cat class can perform its own functions and methods as well as the functions and methods of Animal class.
        Cat c = new Cat();
        c.ani();
        c.cat();
        // Kitten class can perform its own functions and methods as well as the functions and methods of Animal class and Cat class.
        Kitten k = new Kitten();
        k.ani();
        k.cat();
        k.kit();
    }
}
```



MultipleInheritance.java > ...

```java
// implements keyword used if we are using the relation betwwen: CI(class and Interface) and IC(Interface and Class).
// For CC( Class and Class) and II(Interface and Interface) use extends
interface p1{
    default void parent1(){
        System.out.println("Hello there, I am Parent 1 of child class.");
    }
}
interface p2{
    default void parent2(){
        System.out.println("Hello there, I am Parent 2 of child class.");
    }
}
class childClass implements p1, p2{
    void hello(){
        System.out.println("Hello there, I am the Child Class.");
    }
}
public class MultipleInheritance {
    public static void main(String[] args) {
        childClass c = new childClass();
        c.hello();
        c.parent1();
        c.parent2();
    }
}
```



```java
// its not compulsory that the main class should be in the the public class it can be in any either of the classes also.
class Animal {
    void eat() {
        System.out.println("Animal is eating");
    }
}
class Cat extends Animal {
    void meow() {
        System.out.println("cat is meowing");
    }
}
class Dog extends Animal {
    void bark() {
        System.out.println("dog is barking");
    }
}
public class HierarchicalInheritance {
    public static void main(String[] args) {
        Cat c = new Cat();
        c.meow();
        c.eat();

        Animal a = new Animal();
        a.eat();

        Dog d = new Dog();
        d.bark();
        d.eat();
    }
}
```

5. What is polymorphism? Explain with compile-time and runtime examples.

Polymorphism: Same method behaves differently.
Compile-time: Method Overloading
Runtime: Method Overriding

Polymorphism means "many forms". In Java, it allows one interface, method, or object to behave in different ways. It is a key concept of Object-Oriented Programming (OOP).

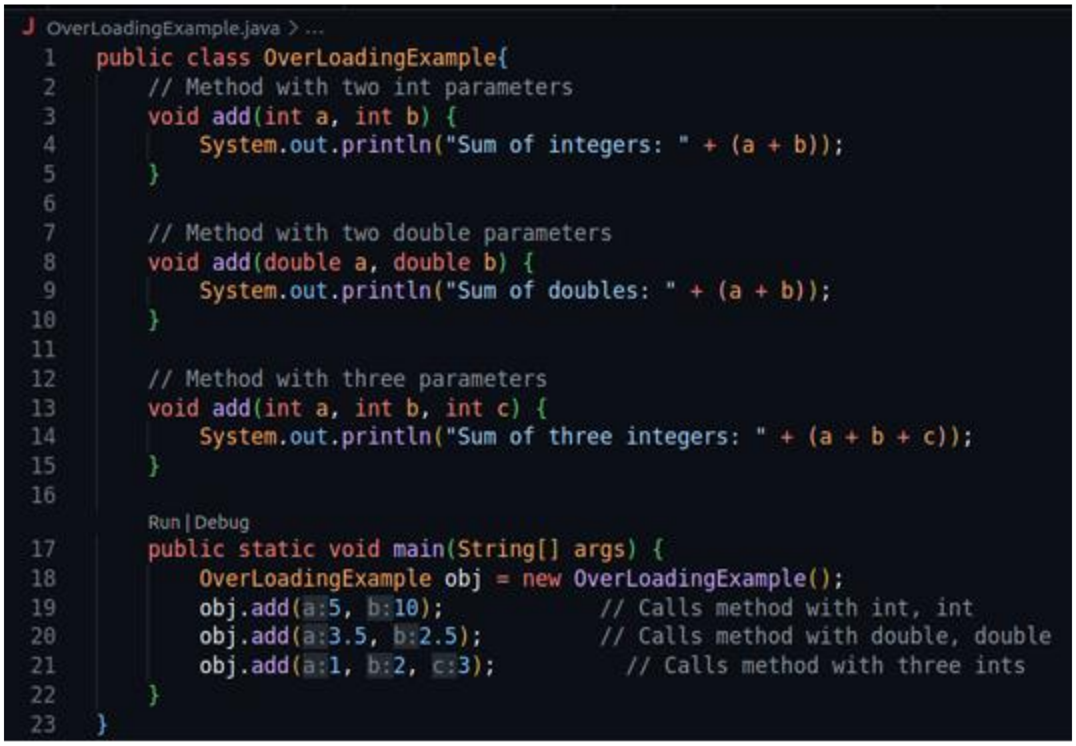
There are two types of polymorphism in Java:

- Compile-time Polymorphism (Method Overloading)

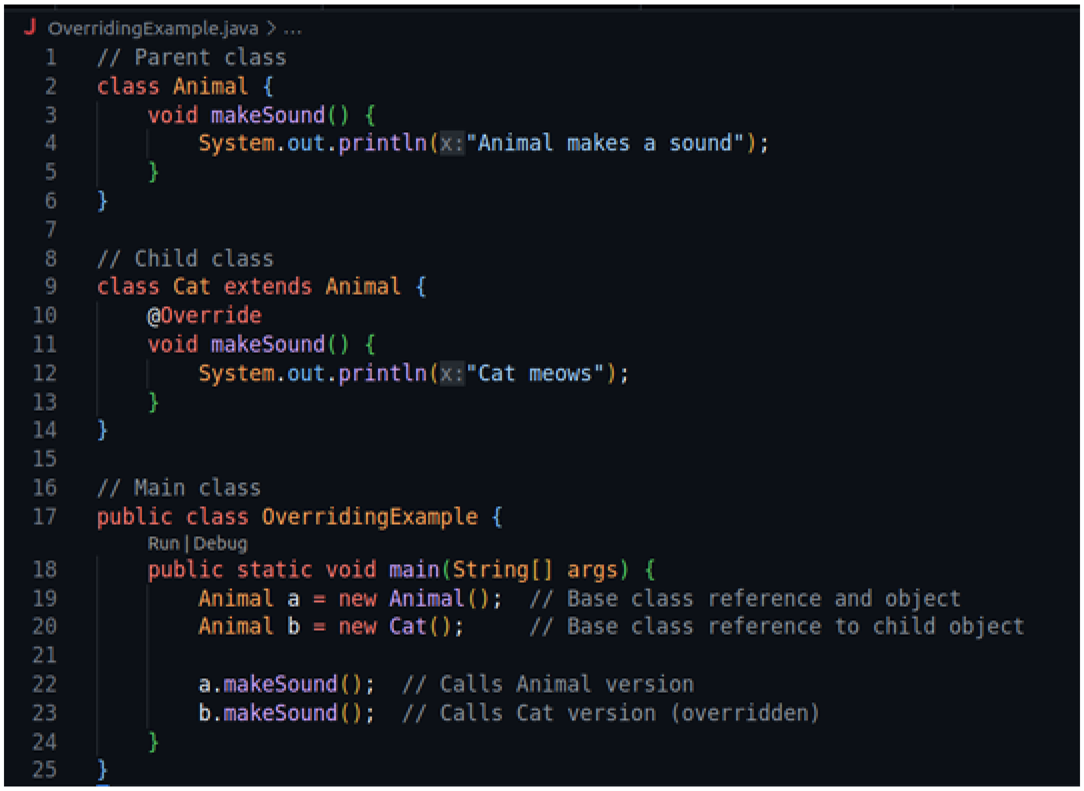
1. It occurs when multiple methods in the same class have the same name but different parameters.
2. The method to be called is decided at compile time

6. What is method overloading and method overriding? Show with examples.

Overloading: Same method, different params
Overriding: Subclass modifies superclass method


OverLoadingExample.java > ...

```java
public class OverLoadingExample{
    // Method with two int parameters
    void add(int a, int b) {
        System.out.println("Sum of integers: " + (a + b));
    }

    // Method with two double parameters
    void add(double a, double b) {
        System.out.println("Sum of doubles: " + (a + b));
    }

    // Method with three parameters
    void add(int a, int b, int c) {
        System.out.println("Sum of three integers: " + (a + b + c));
    }

    Run | Debug
    public static void main(String[] args) {
        OverLoadingExample obj = new OverLoadingExample();
        obj.add(a:5, b:10);            // Calls method with int, int
        obj.add(a:3.5, b:2.5);         // Calls method with double, double
        obj.add(a:1, b:2, c:3);          // Calls method with three ints
    }
}
```


OverridingExample.java > ...

```java
// Parent class
class Animal {
    void makeSound() {
        System.out.println(x:"Animal makes a sound");
    }
}

// Child class
class Cat extends Animal {
    @Override
    void makeSound() {
        System.out.println(x:"Cat meows");
    }
}

// Main class
public class OverridingExample {
    Run | Debug
    public static void main(String[] args) {
        Animal a = new Animal();  // Base class reference and object
        Animal b = new Cat();     // Base class reference to child object

        a.makeSound();  // Calls Animal version
        b.makeSound();  // Calls Cat version (overridden)
    }
}
```

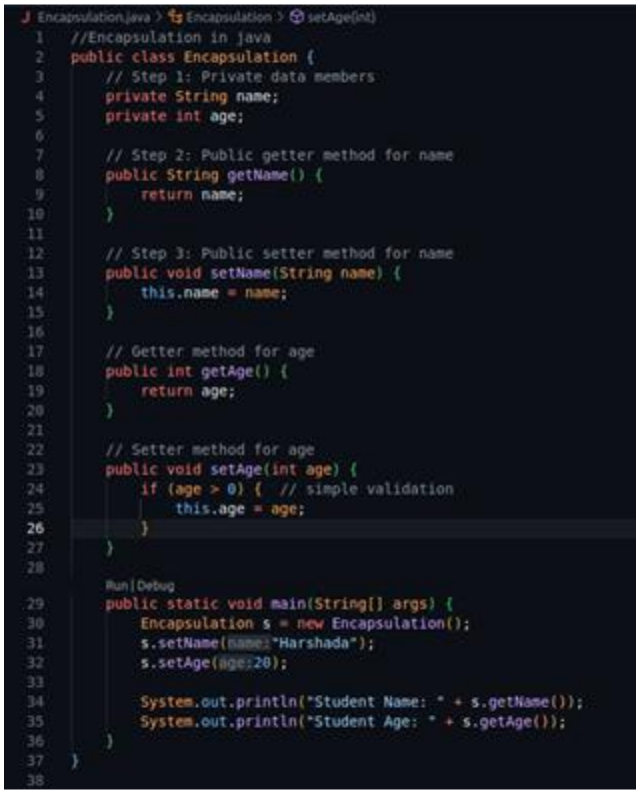
7. What is encapsulation? Write a program demonstrating encapsulation.

Encapsulation is the process of binding data (variables) and code (methods) into a single unit, typically a class, and restricting direct access to some of the object's components. It is achieved by:

- Making variables private
- Providing public getter and setter methods to access and modify those variables

Benefits of Encapsulation:

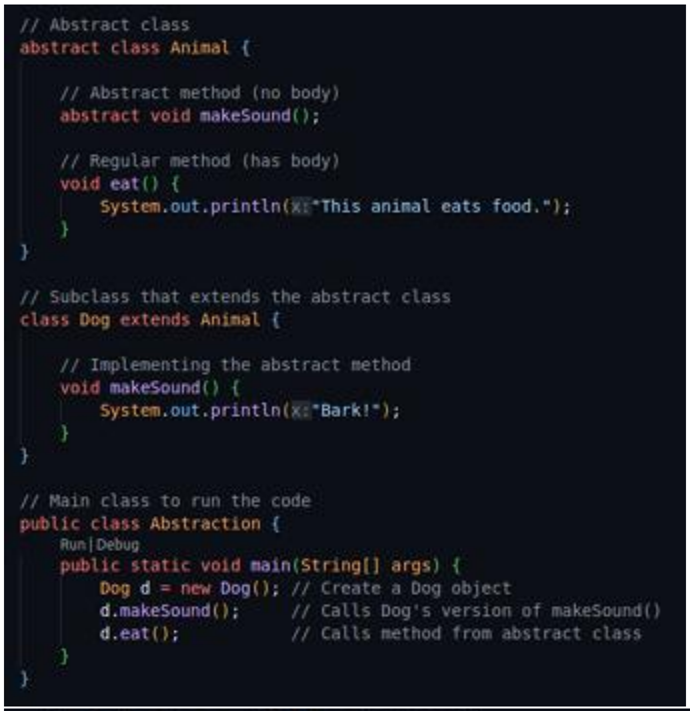
- Protects data from unauthorized access
- Increases code maintainability and flexibility
- Makes the class easier to use and modify



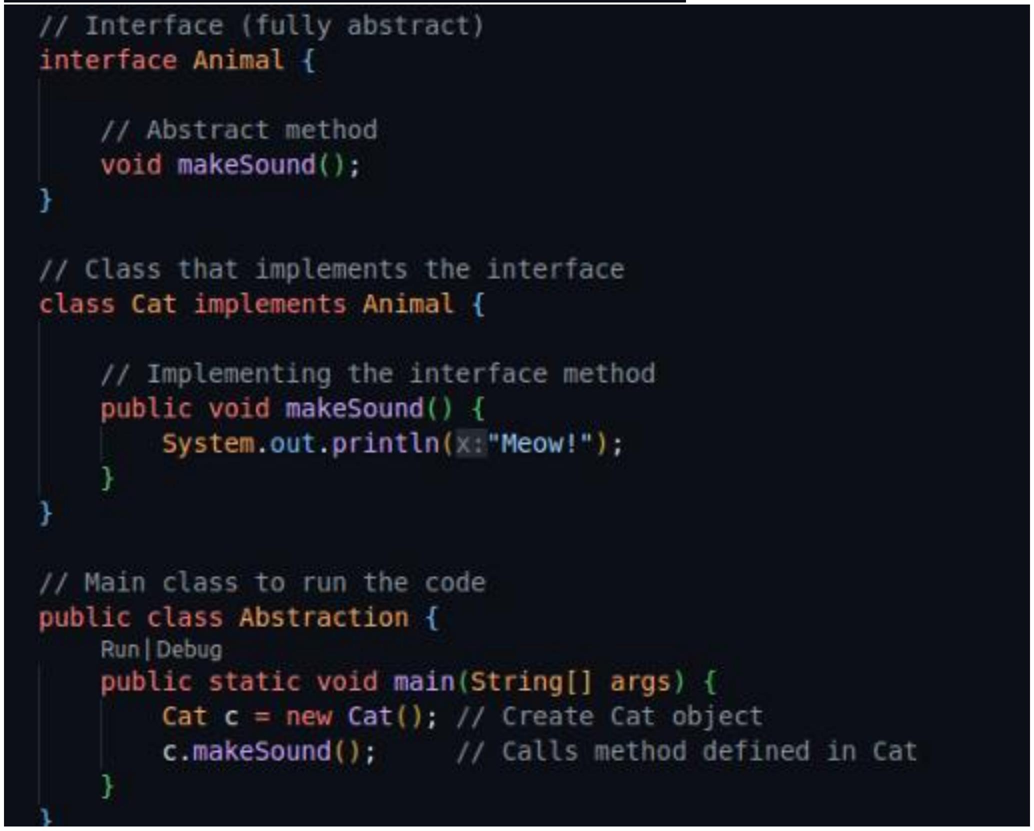
```java
//Encapsulation in java
public class Encapsulation {
    // Step 1: Private data members
    private String name;
    private int age;

    // Step 2: Public getter method for name
    public String getName() {
        return name;
    }

    // Step 3: Public setter method for name
    public void setName(String name) {
        this.name = name;
    }

    // Getter method for age
    public int getAge() {
        return age;
    }

    // Setter method for age
    public void setAge(int age) {
        if (age > 0) {  // simple validation
            this.age = age;
        }
    }

    public static void main(String[] args) {
        Encapsulation s = new Encapsulation();
        s.setName(name:"Harshada");
        s.setAge(age:20);

        System.out.println("Student Name: " + s.getName());
        System.out.println("Student Age: " + s.getAge());
    }
}
```

8. What is abstraction in Java? How is it achieved?

Abstraction hides implementation details.
Achieved via abstract classes and interfaces



```java
// Abstract class
abstract class Animal {

    // Abstract method (no body)
    abstract void makeSound();

    // Regular method (has body)
    void eat() {
        System.out.println(x:"This animal eats food.");
    }
}

// Subclass that extends the abstract class
class Dog extends Animal {

    // Implementing the abstract method
    void makeSound() {
        System.out.println(x:"Bark!");
    }
}

// Main class to run the code
public class Abstraction {
    Run|Debug
    public static void main(String[] args) {
        Dog d = new Dog(); // Create a Dog object
        d.makeSound();     // Calls Dog's version of makeSound()
        d.eat();           // Calls method from abstract class
    }
}
```



```java
// Interface (fully abstract)
interface Animal {

    // Abstract method
    void makeSound();
}

// Class that implements the interface
class Cat implements Animal {

    // Implementing the interface method
    public void makeSound() {
        System.out.println(x:"Meow!");
    }
}

// Main class to run the code
public class Abstraction {
    Run|Debug
    public static void main(String[] args) {
        Cat c = new Cat(); // Create Cat object
        c.makeSound();     // Calls method defined in Cat
    }
}
```

9. Explain the difference between abstract class and interface.

Abstract Class in Java:

An abstract class is a class that is declared with the abstract keyword.

It can have:

- Abstract methods (without a body)
- Concrete methods (with a body)

It is used when you want to provide a base class with some shared code and some methods that must be implemented by child classes.

- Supports partial abstraction
- Can have constructors
- Can have instance variables
- Can be inherited using extends
- A class can extend only one abstract class

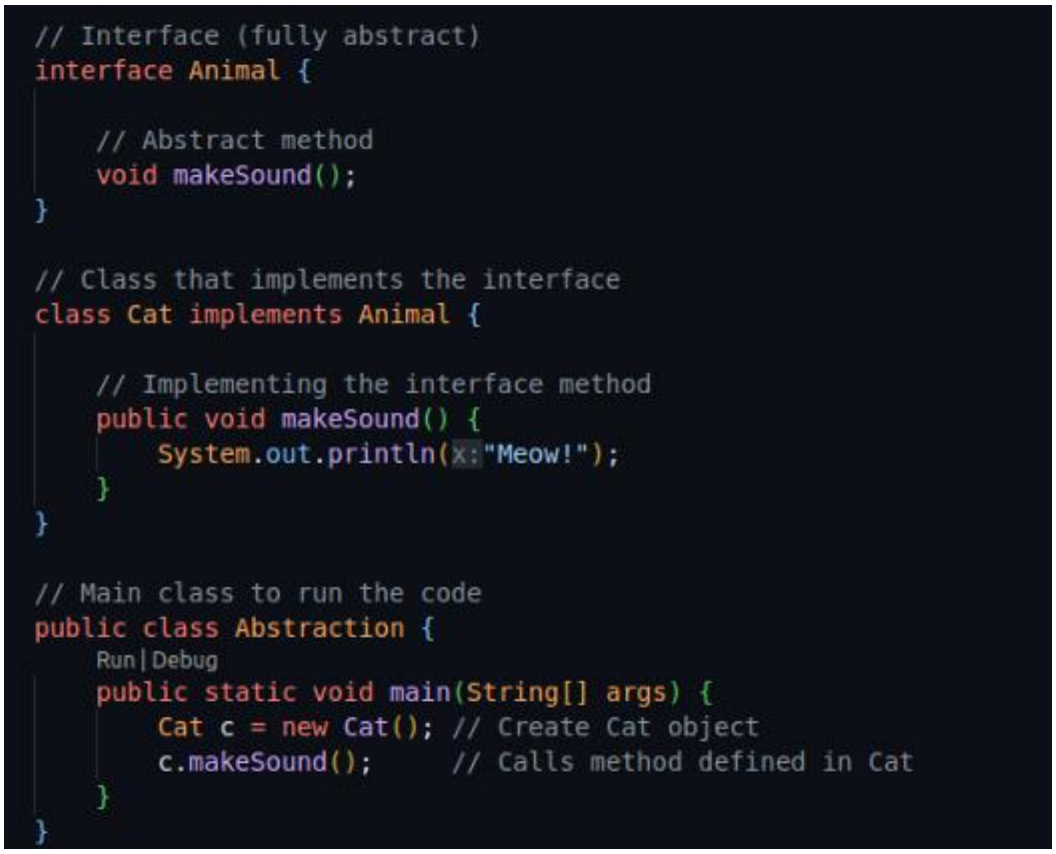
Interface in Java:

An interface is a blueprint of a class that contains only abstract methods (by default) and constants.

It is used to define a set of rules or behaviors that multiple unrelated classes can follow.

- Supports full abstraction (100%)
- All methods are abstract and public by default
- Variables are public, static, and final
- No constructors allowed
- A class can implement multiple interfaces

10. Create a Java program to demonstrate the use of interface.



```
// Interface (fully abstract)
interface Animal {

    // Abstract method
    void makeSound();
}

// Class that implements the interface
class Cat implements Animal {

    // Implementing the interface method
    public void makeSound() {
        System.out.println(x:"Meow!");
    }
}

// Main class to run the code
public class Abstraction {
    Run | Debug
    public static void main(String[] args) {
        Cat c = new Cat(); // Create Cat object
        c.makeSound();     // Calls method defined in Cat
    }
}
```