Java Basics and OOPs Assignment

NAME – Krushna Donge **AF CODE – AF04953342**

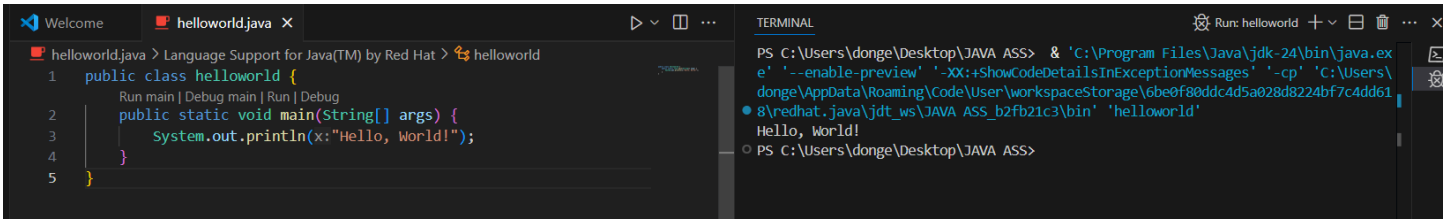
1. What is Java? Explain its features.
Java is a high-level, object-oriented programming language developed by Sun Microsystems (now owned by Oracle). It is platform-independent, secure, and widely used for building web and mobile applications.

Features:
- Platform Independent
- Object-Oriented
- Simple and Secure
- Robust
- Multithreaded
- Portable
- Distributed

2. Explain the Java program execution process.
1. Write code in a .java file.
2. Compile it using javac to create a .class file (bytecode).
3. JVM executes the bytecode on any platform.

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

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



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

Java has:
1. Primitive types: int, float, double, char, boolean, byte, short, long
2. Non-primitive types: String, Array, Class, Object

Example:

java

CopyEdit

int age =19;

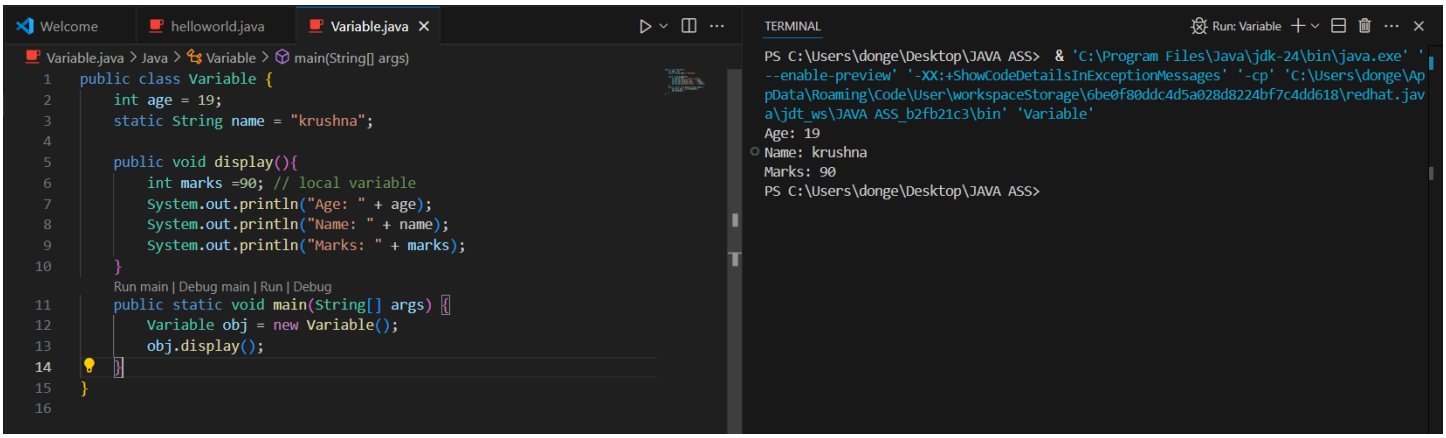
String name = "Krushna";

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

JDK: Java Development Kit (tools for development)
JRE: Java Runtime Environment (runs Java apps)
JVM: Java Virtual Machine (executes bytecode)

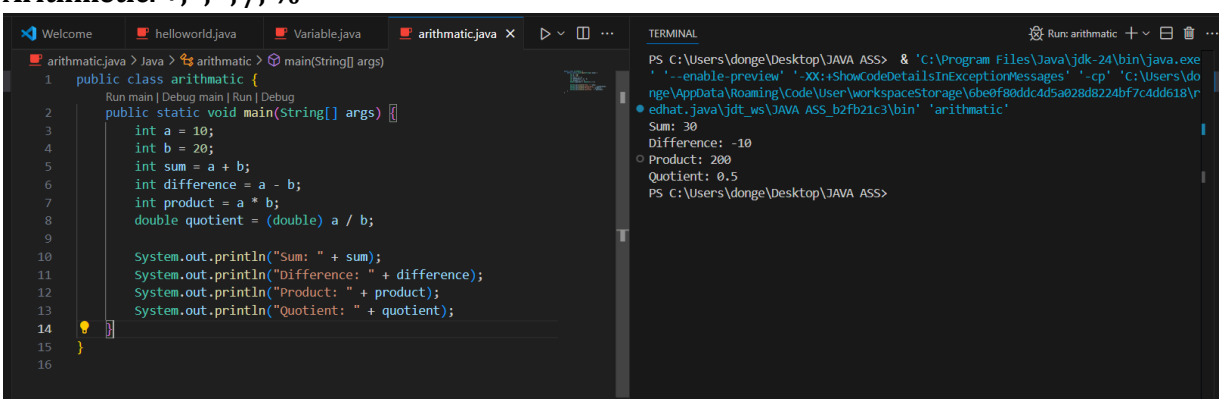
6. What are variables in Java? Explain with examples.

Variables store data. Example:
int age = 19;
String name = "Krushna";

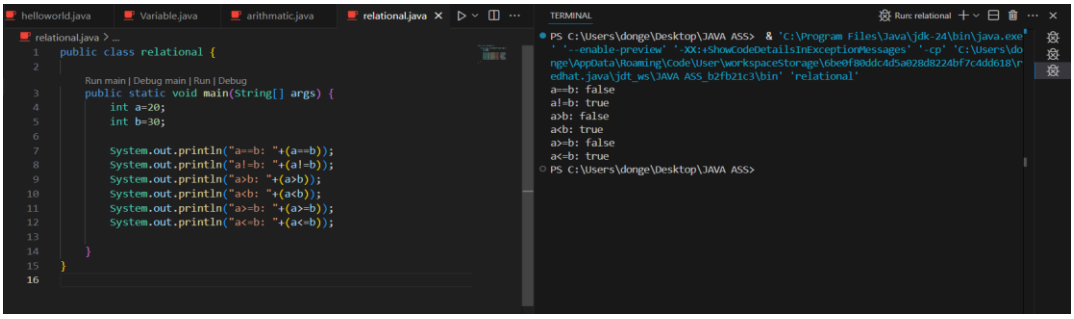


7. What are the different types of operators in Java?

Arithmetic: +, -, *, /, %



Relational: ==, !=, >, <, >=, <=



Logical: &&, ||, !

```java
public class logical {

    Run main | Debug main | Run | Debug
    public static void main(String[] args) {
        int a = 10;
        int b = 20;

        //logical AND(&&)
        System.out.println((a<b)&& (b>14));
        //logical OR(||)
        System.out.println((a<b) || (b<14));
        //logical NOT(!)
        System.out.println(!(a<b));
        //logical XOR(^)
        System.out.println((a<b) ^ (b>14));
        //logical NAND
        System.out.println(!((a<b) && (b>14)));
        //logical NOR
        System.out.println(!((a<b) || (b<14)));

    }
}
```

```
PS C:\Users\donge\Desktop\JAVA ASS>  & 'C:\Program Files\Java\jdk-24\bin\java.exe' '--enable-preview' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\donge\AppData\Roaming\Code\User\workspaceStorage\6be0f80ddc4d5a028d8224bf7c4dd618\redhat.java\jdt_ws\JAVA ASS_b2fb21c3\bin' 'logical'
true
true
false
false
false
false
PS C:\Users\donge\Desktop\JAVA ASS>
```

Assignment: =, +=, -=, etc.

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

        int a=20;
        int b = 30;
        a+= b; // a = a + b
        System.out.println("a+=b: " + a); // 50

        int c = 40;
        c%=2;
        System.out.println("c%=2: " + c); // 0
    }
}
```

```
PS C:\Users\donge\Desktop\JAVA ASS>  & 'C:\Program Files\Java\jdk-24\bin\java.exe' '--enable-preview' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\donge\AppData\Roaming\Code\User\workspaceStorage\6be0f80ddc4d5a028d8224bf7c4dd618\redhat.java\jdt_ws\JAVA ASS_b2fb21c3\bin' 'Assignment'
a+=b: 50
c%=2: 0
PS C:\Users\donge\Desktop\JAVA ASS>
```

Bitwise: &, |, ^

```java
public class Bitwise {
    Run main | Debug main | Run | Debug
    public static void main(String[] args) {
        int a =5;
        int b = 3;
        // Bitwise AND (&)
        System.out.println("a & b: " + (a & b)); // 1
        // Bitwise OR (|)
        System.out.println("a | b: " + (a | b)); // 7
        // Bitwise XOR (^)
        System.out.println("a ^ b: " + (a ^ b)); // 6
        // Bitwise NOT (~)
        System.out.println("~a: " + (~a)); // -6
        // Bitwise left shift (<<)
        System.out.println("a << 1: " + (a << 1)); // 10
        // Bitwise right shift (>>)
        System.out.println("a >> 1: " + (a >> 1)); // 2
        // Bitwise unsigned right shift (>>>)
        System.out.println("a >>> 1: " + (a >>> 1)); // 2
        // Bitwise AND with assignment (&=)
    }
}
```

```
PS C:\Users\donge\Desktop\JAVA ASS>  & 'C:\Program Files\Java\jdk-24\bin\java.exe' '--enable-preview' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\donge\AppData\Roaming\Code\User\workspaceStorage\6be0f80ddc4d5a028d8224bf7c4dd618\redhat.java\jdt_ws\JAVA ASS_b2fb21c3\bin' 'Bitwise'
a & b: 1
a | b: 7
a ^ b: 6
~a: -6
a << 1: 10
a >> 1: 2
a >>> 1: 2
PS C:\Users\donge\Desktop\JAVA ASS>
```

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

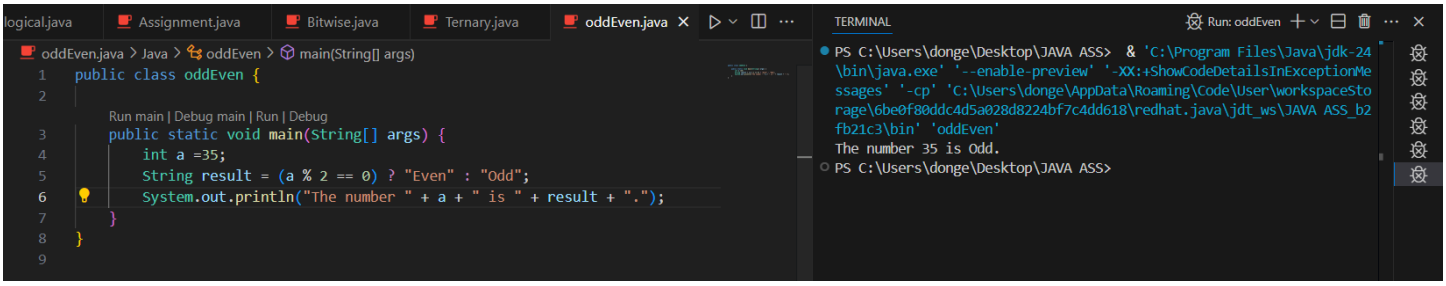
Executes one block of code if the condition is true, otherwise executes another block.

3. switch Statement

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

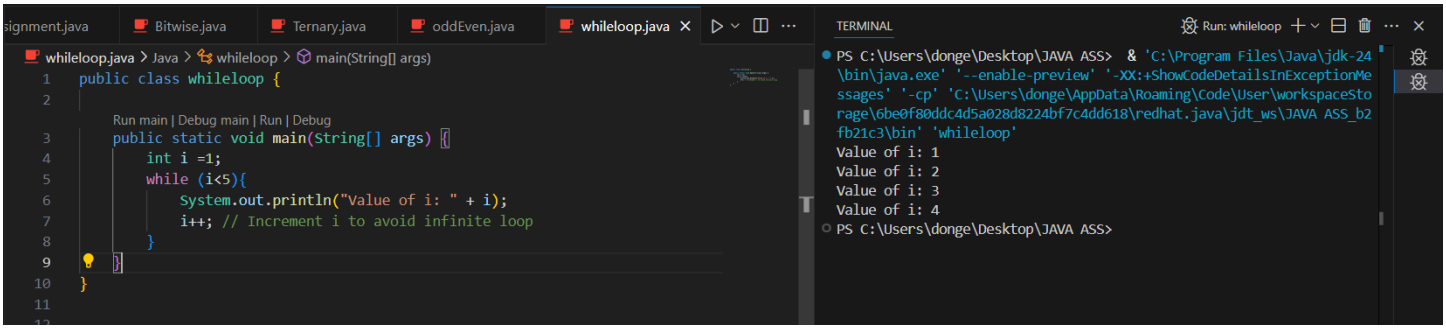
9. Write a Java program to find whether a number is even or odd.

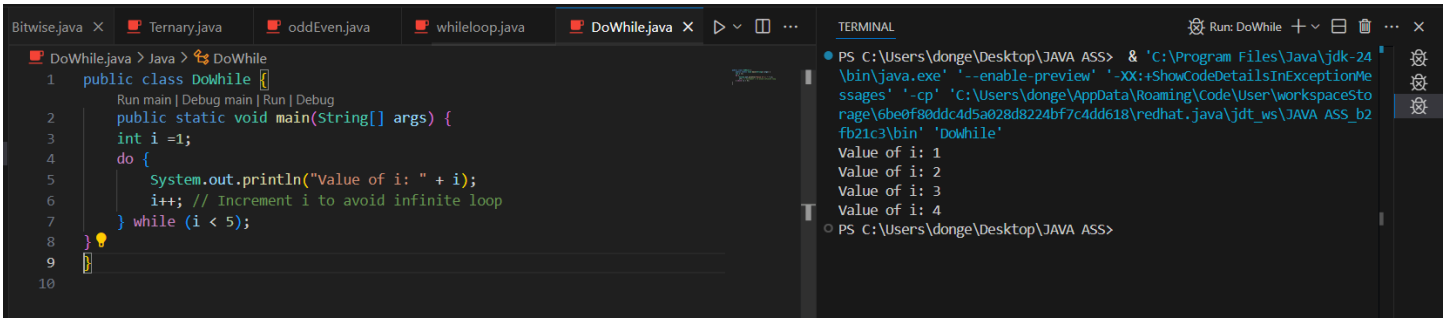
```
import java.util.Scanner;
public class EvenOdd {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int num = sc.nextInt();
    if (num % 2 == 0)
      System.out.println("Even");
    else
      System.out.println("Odd");
  }
}
```



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





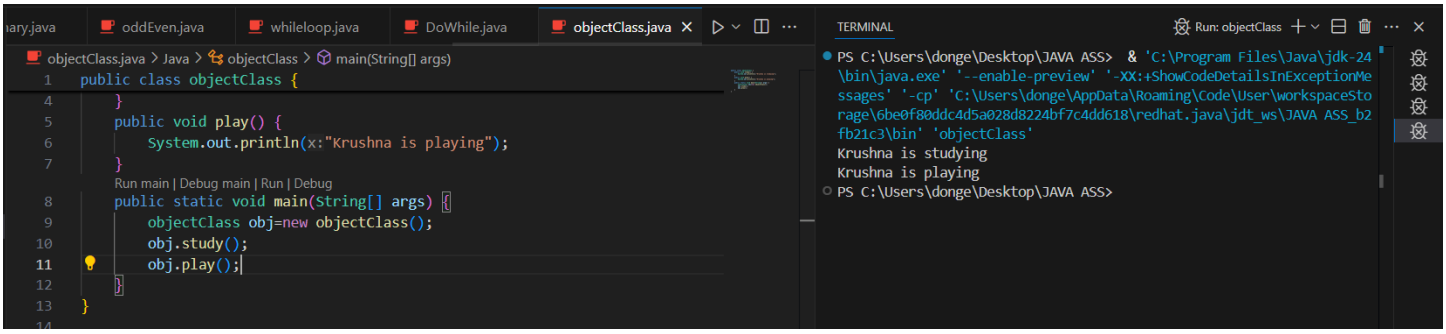
Object-Oriented Programming (OOPs)

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() {} }



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()); }}
```

```java
class Rectangle {
    int length, breadth;

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

public class Main2 {
    Run main | Debug main | Run | Debug
    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
public class inheritance {
    Run main | Debug main | Run | Debug
    public static void main(String[] args) {
        Car myCar = new Car();
        myCar.move(); // Inherited from Vehicle
        myCar.airConditioning(); // Specific to Car
    }
}

// Parent class
class Vehicle {
    void move() {
        System.out.println(x:"Vehicle is moving");
    }
}

// Child class
class Car extends Vehicle {
    void airConditioning() {
        System.out.println(x:"Car has air conditioning");
    }
}
```

```
PS C:\Users\donge\Desktop\JAVA ASS>  & 'C:\Program Files\Java\jdk-24\bin\java.exe' '--enable-preview' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\donge\AppData\Roaming\Code\User\workspaceStorage\6be0f80ddc4d5a028d8224bf7c4dd618\redhat.java\jdt_ws\JAVA ASS_b2fb21c3\bin' 'inheritance'
Vehicle is moving
Car has air conditioning
PS C:\Users\donge\Desktop\JAVA ASS>
```

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

DemoOverrideSuper.java > Java > TestOverride

```java
public class DemoOverrideSuper {
    public void display() {
        System.out.println(x:"Display method in DemoOverrideSuper class");
    }
}

class ChildClass extends DemoOverrideSuper {
    @Override
    public void display() {
        System.out.println(x:"Display method in ChildClass");
    }
}

class TestOverride {
    Run main | Debug main | Run | Debug
    public static void main(String[] args) {
        DemoOverrideSuper obj1 = new DemoOverrideSuper();
        obj1.display(); // Calls parent class method

        DemoOverrideSuper obj2 = new ChildClass();
        obj2.display(); // Calls overridden method in child class
    }
}
```

```
PS C:\Users\donge\Desktop\JAVA ASS>  & 'C:\Program Files\Java\jdk-24
\bin\java.exe' '--enable-preview' '-XX:+ShowCodeDetailsInExceptionMe
ssages' '-cp' 'C:\Users\donge\AppData\Roaming\Code\User\workspaceSto
rage\6be0f80ddc4d5a028d8224bf7c4dd618\redhat.java\jdt_ws\JAVA ASS_b2
fb21c3\bin' 'TestOverride'
Display method in DemoOverrideSuper class
Display method in ChildClass
PS C:\Users\donge\Desktop\JAVA ASS>
```

DemoOverrideSuper.java > Java > TestOverride

```java
public class DemoOverrideSuper {
    public void display() {
        System.out.println(x:"Display method in DemoOverrideSuper class");
    }
}

class ChildClass extends DemoOverrideSuper {
    @Override
    public void display() {
        System.out.println(x:"Display method in ChildClass");
    }
}

class TestOverride {
    Run main | Debug main | Run | Debug
    public static void main(String[] args) {
        DemoOverrideSuper obj1 = new DemoOverrideSuper();
        obj1.display(); // Calls parent class method

        DemoOverrideSuper obj2 = new ChildClass();
        obj2.display(); // Calls overridden method in child class
    }
}
```

```
PS C:\Users\donge\Desktop\JAVA ASS>  & 'C:\Program Files\Java\jdk-24
\bin\java.exe' '--enable-preview' '-XX:+ShowCodeDetailsInExceptionMe
ssages' '-cp' 'C:\Users\donge\AppData\Roaming\Code\User\workspaceSto
rage\6be0f80ddc4d5a028d8224bf7c4dd618\redhat.java\jdt_ws\JAVA ASS_b2
fb21c3\bin' 'TestOverride'
Display method in DemoOverrideSuper class
Display method in ChildClass
PS C:\Users\donge\Desktop\JAVA ASS>
```

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

Encapsulation.java

Encapsulation > Encapsulation.java > Language Support for Java(TM) by Red Hat > BankAccount

```java
class BankAccount{
    private  String accountHolder;
    private double balance;

    public void setAccountHolder(String name) {
        this.accountHolder = name;
    }

    public String getAccountHolder() {
        return accountHolder;
    }

    public void deposit(double amount) {
        if (amount > 0) {
            balance += amount;
        } else {
            System.out.println(x:"Deposit amount must be positive.");
        }
    }

    public void withdraw(double amount) {
        if (amount > 0 && amount <= balance) {
            balance -= amount;
        } else {
            System.out.println(x:"Invalid withdrawal amount.");
        }
    }

    public double getBalance() {
        return balance;
    }

}

// Add this wrapper class to run the program
public class Encapsulation {
    Run main | Debug main | Run | Debug
    public static void main(String[] args) {
        BankAccount account = new BankAccount();
        account.setAccountHolder(name:"John Doe");
        account.deposit(amount:1000);
        account.withdraw(amount:500);
        System.out.println("Account Holder: " + account.getAccountHolder());
        System.out.println("Balance: " + account.getBalance());
    }
```

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

Abstraction hides implementation details.
Achieved via abstract classes and interfaces

TestATM.java 2

TestATM.java > Java > TestATM

```java
abstract class ATM{
    abstract void withdraw(double amount);
    abstract void checkBalance();
}

class MyAt extends ATM{
    double balance = 1000.0;

    @Override
    void withdraw(double amount) {
        if (amount > 0 && amount <= balance) {
            balance -= amount;
            System.out.println("Withdrawal successful. New balance: " + balance);
        } else {
            System.out.println(x:"Invalid withdrawal amount.");
        }
    }

    @Override
    void checkBalance() {
        System.out.println("Current balance: " + balance);
    }
}

public class TestATM {
    Run main | Debug main | Run | Debug
    public static void main(String[] args) {
        MyAt atm = new MyAt();
        atm.checkBalance();
        atm.withdraw(amount:200);
        atm.checkBalance();
        atm.withdraw(amount:1000); // Should show invalid withdrawal
    }
}
```

CHAT

TERMINAL

```
PS C:\Users\donge\Desktop\JAVA ASS>  & 'C:\Program Files\Java\jdk-24\bin\java.exe' '-agentlib:jdwp=transport=dt_socket,server=n,suspend=y,address=localhost:62587' '--enable-preview' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\donge\AppData\Roaming\Code\User\workspaceStorage\6be0f80ddc4d5a028d8224bf7c4dd618\redhat.java\jdt_ws\JAVA ASS_b2fb21c3\bin' 'TestATM'
Current balance: 1000.0
Withdrawal successful. New balance: 800.0
Current balance: 800.0
Invalid withdrawal amount.
PS C:\Users\donge\Desktop\JAVA ASS>
```

powershell
Run: TestATM
Debug: Test...

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

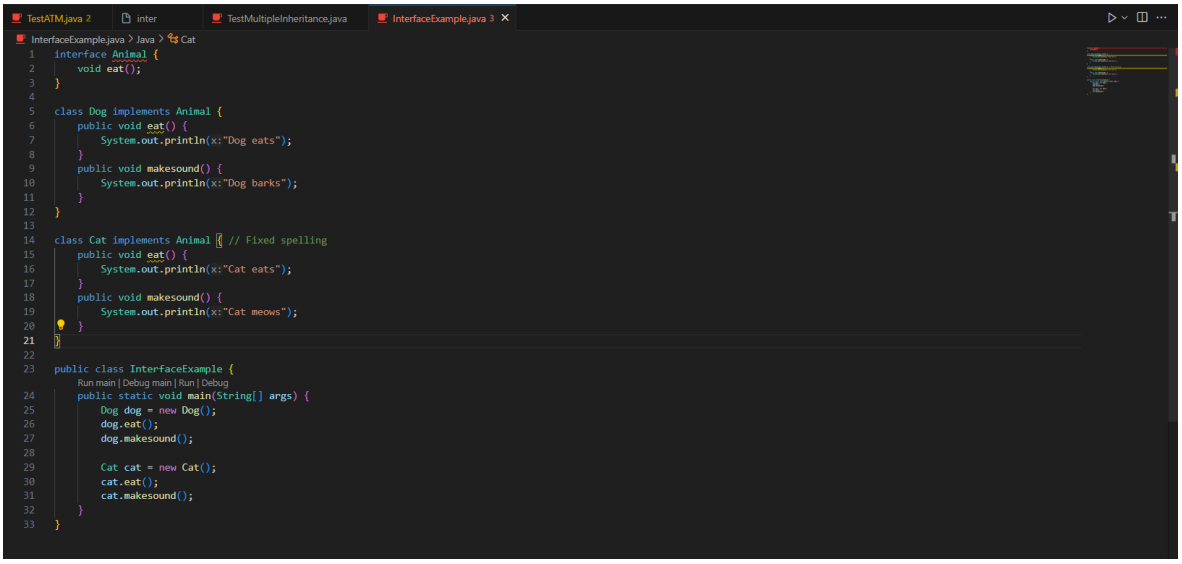
Interface in Java:

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.



```java
interface Animal {
    void eat();
}

class Dog implements Animal {
    public void eat() {
        System.out.println("Dog eats");
    }
    public void makesound() {
        System.out.println("Dog barks");
    }
}

class Cat implements Animal { // Fixed spelling
    public void eat() {
        System.out.println("Cat eats");
    }
    public void makesound() {
        System.out.println("Cat meows");
    }
}

public class InterfaceExample {
    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.eat();
        dog.makesound();

        Cat cat = new Cat();
        cat.eat();
        cat.makesound();
    }
}
```