

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.

Features:

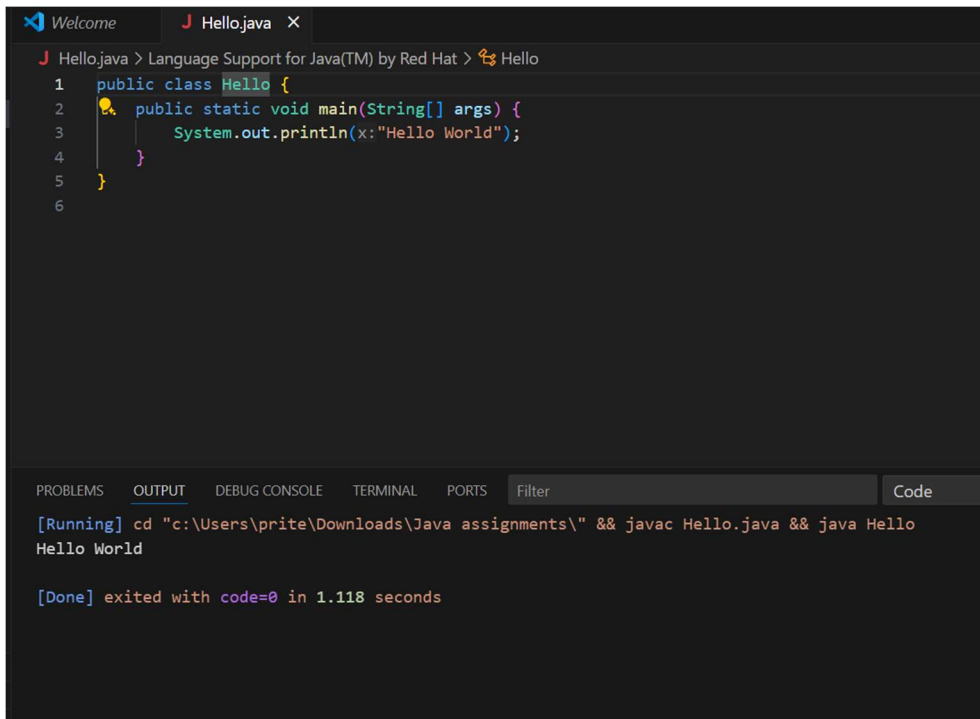
- Platform Independent
- Object-Oriented
- Simple and Secure
- Robust
- Multithreaded
- Portable
- Distributed

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");  
    }  
}
```

A screenshot of an IDE window titled 'Hello.java'. The editor shows a Java program:

```
1 public class Hello {  
2     public static void main(String[] args) {  
3         System.out.println(x:"Hello World");  
4     }  
5 }  
6
```

 Below the editor is a terminal panel with tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS. The terminal shows the command `cd "c:\Users\prite\Downloads\Java assignments\" && javac Hello.java && java Hello` and the output `Hello World`. A status bar at the bottom indicates `[Done] exited with code=0 in 1.118 seconds`.

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 = 25;
```

```
String name = "Abhijeet";
```

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 = "Abhijeet";
```

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

- Arithmetic: +, -, *, /, %
- Relational: ==, !=, >, <, >=, <=
- Logical: &&, ||, !
- Assignment: =, +=, -=, etc.
- Unary: ++, --
- Bitwise: &, |, ^

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

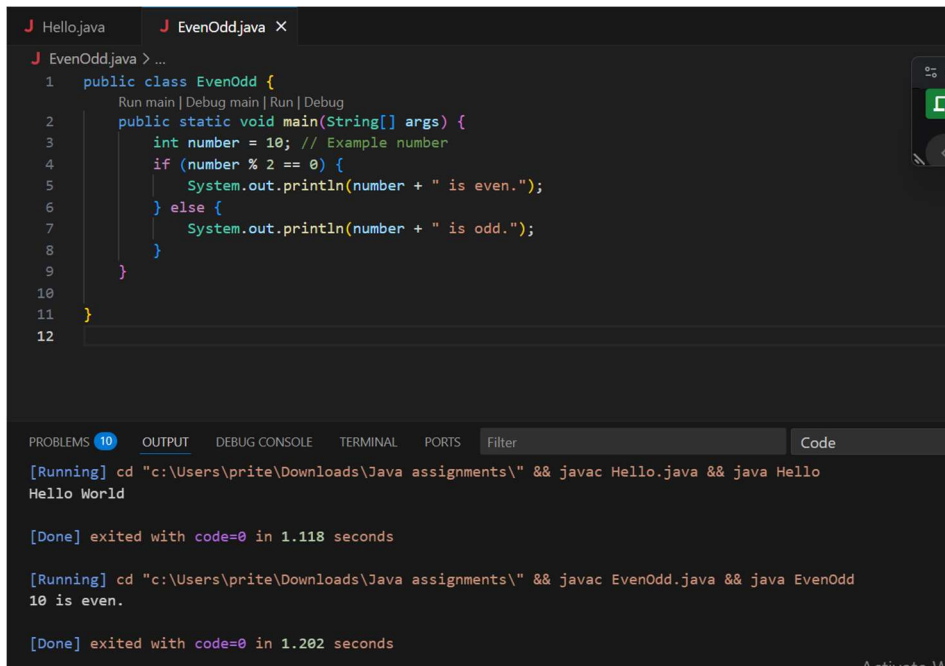
Executes one block of code if the condition is true, otherwise executes another block.

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");
    }
}
```



The screenshot shows an IDE with two tabs: 'Hello.java' and 'EvenOdd.java'. The 'EvenOdd.java' tab is active, displaying the following code:

```
1 public class EvenOdd {  
2     public static void main(String[] args) {  
3         int number = 10; // Example number  
4         if (number % 2 == 0) {  
5             System.out.println(number + " is even.");  
6         } else {  
7             System.out.println(number + " is odd.");  
8         }  
9     }  
10 }  
11  
12
```

Below the code editor, the 'OUTPUT' tab is selected, showing the execution results:

```
[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() {} }
```

```

J ClassesAndObjects.java > ...
1  public class ClassesAndObjects {
2      public void display() {
3          System.out.println(x:"Hello, this is a method in ClassObj.");
4      }
5      public void show() {
6          System.out.println(x:"This is show method in ClassObj.");
7      }
8      Run | Debug
9      public static void main(String[] args) {
10         //create an instance of ClassObj
11         ClassesAndObjects obj = new ClassesAndObjects();
12         obj.display();
13         obj.show();
14     }
15 }
16

```

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();
```

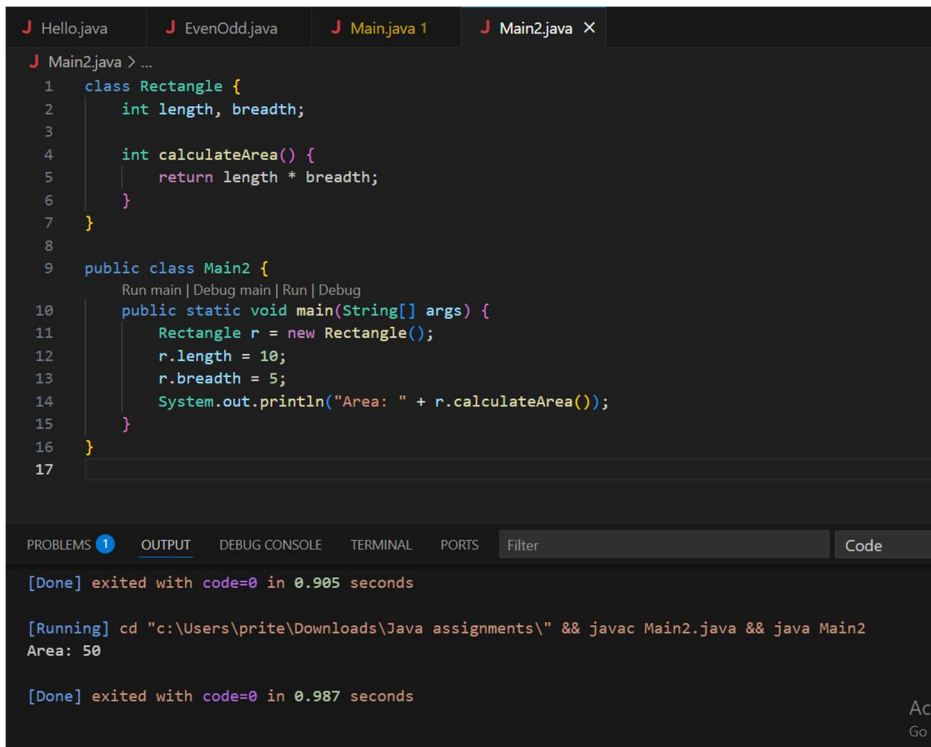
```
        r.length = 10;
```

```
        r.breadth = 5;
```

```
        System.out.println("Area: " + r.calculateArea());
```

```
    }
```

```
}
```



```
J Hello.java J EvenOdd.java J Main.java 1 J Main2.java X
J Main2.java > ...
1 class Rectangle {
2     int length, breadth;
3
4     int calculateArea() {
5         return length * breadth;
6     }
7 }
8
9 public class Main2 {
10     Run main | Debug main | Run | Debug
11     public static void main(String[] args) {
12         Rectangle r = new Rectangle();
13         r.length = 10;
14         r.breadth = 5;
15         System.out.println("Area: " + r.calculateArea());
16     }
17 }

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS Filter Code

[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.



```
SingleInheritance.java
1 // Here Animal is Parent class where as Cat is child class.
2 // Child class can use the properties of parent class but parent class can't use the methods and functions and properties of child class
3 // This is a example of Single inheritance as there is a single parent and single child.
4
5
6 class Animal{
7     void eat(){
8         System.out.println("Animal is eating");
9     }
10 }
11
12 class Cat extends Animal{
13     void meow(){
14         System.out.println("cat is meowing");
15     }
16 }
17
18 public class SingleInheritance {
19     Run | Debug
20     public static void main(String[] args) {
21         Cat c = new Cat();
22         c.meow();
23         c.eat();
24
25         Animal a= new Animal();
26         a.eat();
27     }
28 }
29
```

```

1 // here the class Animal is Grand parent class.
2 // Cat is Parent class and the child class of Animal, can access the properties of its parent class Animal only.
3 // Lastly Kitten is Child class of Cat and this class can access properties and functions of both the parent classes.
4 class Animal{
5     void ani(){
6         System.out.println("HI there, I am an Animal!!");
7     }
8 }
9 class Cat extends Animal{
10     void cat(){
11         System.out.println("HI there, I am an Cat!!");
12     }
13 }
14 class Kitten extends Cat{
15     void kit(){
16         System.out.println("HI there, I am an Kitten!!");
17     }
18 }
19 public class MultilevelInheritance {
20     public static void main(String[] args) {
21         // Animal class can perform its own functions and methods only.
22         Animal a = new Animal();
23         a.ani();
24         // Cat class can perform its own functions and methods as well as the functions and methods of Animal class.
25         Cat c = new Cat();
26         c.ani();
27         c.cat();
28         // Kitten class can perform its own functions and methods as well as the functions and methods of Animal class and Cat class.
29         Kitten k = new Kitten();
30         k.ani();
31         k.cat();
32         k.kit();
33     }
34 }

```

```

1 // implements keyword used if we are using the relation between: CI(class and Interface) and IC(Interface and Class).
2 // For CC( Class and Class) and II(Interface and Interface) use extends
3 interface p1{
4     default void parent1(){
5         System.out.println("Hello there, I am Parent 1 of child class.");
6     }
7 }
8 interface p2{
9     default void parent2(){
10         System.out.println("Hello there, I am Parent 2 of child class.");
11     }
12 }
13 class childClass implements p1, p2{
14     void hello(){
15         System.out.println("Hello there, I am the Child Class.");
16     }
17 }
18 public class MultipleInheritance {
19     public static void main(String[] args) {
20         childClass c = new childClass();
21         c.hello();
22         c.parent1();
23         c.parent2();
24     }
25 }

```

```

1 // its not compulsory that the main class should be in the the public class it can be in any either of the classes also.
2 class Animal {
3     void eat() {
4         System.out.println("Animal is eating");
5     }
6 }
7 class Cat extends Animal {
8     void meow() {
9         System.out.println("cat is meowing");
10    }
11 }
12 class Dog extends Animal {
13     void bark() {
14         System.out.println("dog is barking");
15    }
16 }
17 public class HierarchicalInheritance {
18     public static void main(String[] args) {
19         Cat c = new Cat();
20         c.meow();
21         c.eat();
22
23         Animal a = new Animal();
24         a.eat();
25
26         Dog d = new Dog();
27         d.bark();
28         d.eat();
29     }
30 }

```

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


```

J OverLoadingExample.java > ...
1  public class OverLoadingExample{
2      // Method with two int parameters
3      void add(int a, int b) {
4          System.out.println("Sum of integers: " + (a + b));
5      }
6
7      // Method with two double parameters
8      void add(double a, double b) {
9          System.out.println("Sum of doubles: " + (a + b));
10     }
11
12     // Method with three parameters
13     void add(int a, int b, int c) {
14         System.out.println("Sum of three integers: " + (a + b + c));
15     }
16
17     Run | Debug
18     public static void main(String[] args) {
19         OverLoadingExample obj = new OverLoadingExample();
20         obj.add(a:5, b:10);           // Calls method with int, int
21         obj.add(a:3.5, b:2.5);       // Calls method with double, double
22         obj.add(a:1, b:2, c:3);      // Calls method with three ints
23     }

```

```

J OverridingExample.java > ...
1  // Parent class
2  class Animal {
3      void makeSound() {
4          System.out.println(x:"Animal makes a sound");
5      }
6  }
7
8  // Child class
9  class Cat extends Animal {
10     @Override
11     void makeSound() {
12         System.out.println(x:"Cat meows");
13     }
14 }
15
16 // Main class
17 public class OverridingExample {
18     Run | Debug
19     public static void main(String[] args) {
20         Animal a = new Animal(); // Base class reference and object
21         Animal b = new Cat();    // Base class reference to child object
22
23         a.makeSound(); // Calls Animal version
24         b.makeSound(); // Calls Cat version (overridden)
25     }

```

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

```
J Encapsulation.java > Encapsulation > setAge(int)
1 //Encapsulation in java
2 public class Encapsulation {
3     // Step 1: Private data members
4     private String name;
5     private int age;
6
7     // Step 2: Public getter method for name
8     public String getName() {
9         return name;
10    }
11
12    // Step 3: Public setter method for name
13    public void setName(String name) {
14        this.name = name;
15    }
16
17    // Getter method for age
18    public int getAge() {
19        return age;
20    }
21
22    // Setter method for age
23    public void setAge(int age) {
24        if (age > 0) { // simple validation
25            this.age = age;
26        }
27    }
28
29    Run | Debug
30    public static void main(String[] args) {
31        Encapsulation s = new Encapsulation();
32        s.setName(name:"Marshada");
33        s.setAge(age:20);
34
35        System.out.println("Student Name: " + s.getName());
36        System.out.println("Student Age: " + s.getAge());
37    }
38 }
```

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

Abstraction hides implementation details.

Achieved via abstract classes and interfaces

```
// 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
    }
}
```

```
// 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
    }
}
```