**🔹 1. What is OOP? (Object-Oriented Programming)**

**English:** OOP means writing programs using "objects" that have data and actions. It's like creating mini-real-life things inside your code.

**Urdu:**  
OOP ka matlab hai program banana "objects" ke zariye, jisme data aur actions dono hotay hain. Ye real-life cheezon jaise kaam karta hai.

**🧠 Real-life Example:**

A **Car** is an object. It has:

* **Data**: color, model, speed
* **Actions**: start(), stop()

**🔹 2. Encapsulation**

**English:** Encapsulation means *hiding data* inside a class using **private variables** and accessing them using **getter-setter methods**.

**Urdu:**  
Encapsulation ka matlab hai ke data ko class ke andar **private** bana kar chhupa diya jaye, aur use access karne ke liye **methods** diye jayein.

**💻 .. Example:**

public class Student {

private String name; // private = hidden

// Getter method (read data)

public String getName() {

return name;

}

// Setter method (write data)

public void setName(String n) {

name = n;

}

}

**🔹 3. Inheritance**

**English:** Inheritance allows one class to **use** the features of another class. Like a child inheriting traits from parents.

**Urdu:**  
Inheritance ka matlab hai ek class doosri class ke **features** le leti hai. Jaise bacha parents se cheezein inherit karta hai.

**💻 .. Example:**

// Parent Class

class Animal {

void sound() {

System.out.println("Animals make sounds");

}

}

// Child Class

class Dog extends Animal {

void bark() {

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

}

}

// Main Class

public class Main {

public static void main(String[] args) {

Dog d = new Dog();

d.sound(); // from parent

d.bark(); // own method

}

}

**🔹 4. Polymorphism**

**English:** Polymorphism means **one name, many forms**. Same method can behave differently depending on context.

**Urdu:**  
Polymorphism ka matlab hai **ek naam, kai kaam**. Method aik hi hoti hai lekin uska kaam alag hota hai situation ke mutabiq.

**💻 .. Example:**

class Shape {

void draw() {

System.out.println("Drawing a shape");

}

}

class Circle extends Shape {

void draw() {

System.out.println("Drawing a circle");

}

}

class Square extends Shape {

void draw() {

System.out.println("Drawing a square");

}

}

public class Test {

public static void main(String[] args) {

Shape s1 = new Circle(); // Runtime polymorphism

Shape s2 = new Square();

s1.draw(); // Output: Drawing a circle

s2.draw(); // Output: Drawing a square

}

}

**🔹 5. Abstraction & Abstract Classes**

**English:** Abstraction means **showing only important details**, hiding the rest. Use **abstract classes** to achieve this.

**Urdu:**  
Abstraction ka matlab hai **sirf zaroori cheezein dikhana**, aur baaqi detail chhupa lena. Iske liye hum **abstract class** use karte hain.

**💻 .. Example:**

abstract class Animal {

abstract void makeSound(); // abstract method

void sleep() {

System.out.println("Animal sleeps");

}

}

class Cat extends Animal {

void makeSound() {

System.out.println("Meow");

}

}

public class Main {

public static void main(String[] args) {

Animal obj = new Cat();

obj.makeSound(); // Meow

obj.sleep(); // Animal sleeps

}

}

**🔹 1. Simple Array in ..**

**🔸 English:**

An **array** stores multiple values of the **same type** in one variable. Its size is **fixed**.

**🔸 Urdu:**

**Array** aik aisa container hai jisme aap **same type ke multiple values** store kar saktay ho. Iska size fix hota hai.

**💻 Example:**

public class TestArray {

public static void main(String[] args) {

int[] numbers = {10, 20, 30, 40}; // array of 4 integers

// Accessing values

System.out.println(numbers[0]); // Output: 10

System.out.println(numbers[2]); // Output: 30

}

}

**🔹 2. ArrayList in ..**

**🔸 English:**

**ArrayList** is like an array but its **size is flexible** — you can add/remove elements anytime.

**🔸 Urdu:**

**ArrayList** array ki tarah hoti hai lekin iska size **change hota rehta hai** — aap chahein to element add/remove kar saktay hain.

🔸 Import is needed: import ...util.ArrayList;

**💻 Example:**

import ...util.ArrayList;

public class ExampleList {

public static void main(String[] args) {

ArrayList<String> fruits = new ArrayList<>();

// Adding elements

fruits.add("Apple");

fruits.add("Mango");

fruits.add("Banana");

// Accessing elements

System.out.println(fruits.get(1)); // Output: Mango

// Size of list

System.out.println(fruits.size()); // Output: 3

}

}

**🔹 3. for loop**

**🔸 English:**

The **for loop** is used to **repeat** a block of code multiple times.

**🔸 Urdu:**

**for loop** kisi code ko **bar chalane** ke liye use hoti hai.

**💻 Example with Array:**

public class LoopExample {

public static void main(String[] args) {

int[] marks = {60, 70, 80, 90};

for(int i = 0; i < marks.length; i++) {

System.out.println("Mark: " + marks[i]);

}

}

}

**🔹 4. if-else condition**

**🔸 English:**

Used for **decision making** — if the condition is true, do something; else, do something else.

**🔸 Urdu:**

**if-else** condition decision lene ke liye use hoti hai. Agar condition true ho to ek kaam karo, warna doosra.

**💻 Example:**

public class IfElseExample {

public static void main(String[] args) {

int number = 15;

if (number > 10) {

System.out.println("Number is greater than 10");

} else {

System.out.println("Number is 10 or less");

}

}

}

**🔁 Using all together: Array + for loop + if-else**

..

..

public class ResultCheck {

public static void main(String[] args) {

int[] marks = {45, 78, 90, 33, 67};

for (int i = 0; i < marks.length; i++) {

if (marks[i] >= 50) {

System.out.println("Student " + (i+1) + ": Pass");

} else {

System.out.println("Student " + (i+1) + ": Fail");

}

}

}

}

**Output:**

Student 1: Fail

Student 2: Pass

Student 3: Pass

Student 4: Fail

Student 5: Pass

**✅ Summary (Today's Concepts):**

| **Topic** | **Urdu Summary** |
| --- | --- |
| Array | Fixed size, same type values |
| ArrayList | Flexible size list |
| For Loop | Repeat code |
| If-Else | Decision making |

**🔹 1. File Handling in ..**

**🔸 English:**

File Handling allows you to **read from** and **write to** a file (like .txt file) using .. code.

**🔸 Urdu:**

File Handling ka matlab hai ke aap .. code se kisi file (.txt) mein **likh sakte** ho aur **parh bhi sakte** ho.

**✅ Common Classes:**

* FileWriter: To **write** to a file
* FileReader or Scanner: To **read** a file

**💻 Example 1: Write to a File**

import ...io.FileWriter;

import ...io.IOException;

public class WriteFileExample {

public static void main(String[] args) {

try {

FileWriter writer = new FileWriter("output.txt");

writer.write("Hello Alishbah!\nThis is a file writing example.");

writer.close();

System.out.println("File written successfully.");

} catch (IOException e) {

System.out.println("An error occurred.");

e.printStackTrace();

}

}

}

**Explanation (Urdu):**  
Yahan hum FileWriter se output.txt file banakar usme text likh rahe hain. try-catch use kiya gaya hai agar koi error aaye to handle ho jaye.

**💻 Example 2: Read from a File**

import ...io.File;

import ...io.FileNotFoundException;

import ...util.Scanner;

public class ReadFileExample {

public static void main(String[] args) {

try {

File = new File("output.txt");

Scanner reader = new Scanner(file);

while (reader.hasNextLine()) {

String data = reader.nextLine();

System.out.println(data);

}

reader.close();

} catch (FileNotFoundException e) {

System.out.println("File not found.");

e.printStackTrace();

}

}

}

**🔹 2. Exception Handling in ..**

**🔸 English:**

An **exception** is an error that happens while running a program. .. lets you handle exceptions using **try-catch**.

**🔸 Urdu:**

**Exception** aik error hoti hai jo program chaltay waqt aati hai. .. mein hum isko **try-catch** block se handle karte hain.

**💻 Example:**

..

..

public class ExceptionExample {

public static void main(String[] args) {

try {

int num = 5 / 0; // error: divide by zero

System.out.println("Result: " + num);

} catch (ArithmeticException e) {

System.out.println("Cannot divide by zero!");

}

}

}

**Output:**

csharp

..

Cannot divide by zero!

**✅ Common Types of Exceptions:**

| **Exception Type** | **Urdu Meaning** |
| --- | --- |
| ArithmeticException | Maths error (e.g., divide by 0) |
| ArrayIndexOutOfBounds | Array ke bahar access |
| FileNotFoundException | File nahi mili |
| NullPointerException | Null value par kaam karna |

**💻 Another Example: Handle multiple exceptions**

..

..

public class MultiCatchExample {

public static void main(String[] args) {

try {

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

System.out.println(arr[5]); // Invalid index

} catch (ArrayIndexOutOfBoundsException e) {

System.out.println("Array index is out of bounds!");

} catch (Exception e) {

System.out.println("Some other error occurred.");

}

}

}

**🔚 Summary:**

| **Topic** | **English Summary** | **Urdu Summary** |
| --- | --- | --- |
| File Handling | Reading/writing to file | File se likhna/parhna |
| Exception Handling | Handling errors using try-catch | Program ke errors ko sambhalna |

**🔹 1. Abstract Class in ..**

**🔸 English:**

An **abstract class** is a class that **cannot be directly used to create objects**. It may contain **abstract methods** (without body) and **regular methods** (with body). Other classes **inherit** it and provide **complete implementation**.

**🔸 Urdu:**

**Abstract class** aik aisi class hoti hai jiska **object directly create nahi kiya ja sakta**. Isme kuch methods **body ke baghair (abstract)** hotay hain aur kuch ke sath hotay hain. Dusri classes isay **inherit karke** methods ka complete code deti hain.

**💻 Example: Abstract Class**

..

..

abstract class Animal {

abstract void sound(); // abstract method

void eat() { // normal method

System.out.println("Animals eat food");

}

}

class Dog extends Animal {

void sound() {

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

}

}

public class TestAbstract {

public static void main(String[] args) {

Dog d = new Dog();

d.sound(); // Output: Dog barks

d.eat(); // Output: Animals eat food

}

}

✅ **Key Points:**

* abstract keyword is used.
* abstract method has **no body**.
* You must **extend** abstract class to use it.

**🔹 2. Interfaces in ..**

**🔸 English:**

An **interface** is like a contract. It only contains **method declarations** (no body). A class that **implements** the interface must provide code for **all methods**.

**🔸 Urdu:**

**Interface** aik contract ki tarah hoti hai. Isme sirf method ka **naam hota hai**, koi body nahi. Jo class is interface ko **implement** karti hai, usay **saray methods ka code likhna zaroori hota hai**.

**💻 Example: Interface**

..

..

interface Animal {

void sound(); // no body

void eat(); // no body

}

class Cat implements Animal {

public void sound() {

System.out.println("Cat meows");

}

public void eat() {

System.out.println("Cat eats fish");

}

}

public class TestInterface {

public static void main(String[] args) {

Cat c = new Cat();

c.sound(); // Output: Cat meows

c.eat(); // Output: Cat eats fish

}

}

✅ **Key Points:**

* interface keyword is used.
* Methods are **public and abstract by default**.
* Class must use implements and write **all method bodies**.

**🔄 Abstract Class vs Interface — Comparison Table**

| **Feature** | **Abstract Class** | **Interface** |
| --- | --- | --- |
| Object Creation | ❌ Not allowed directly | ❌ Not allowed directly |
| Method Body | ✅ Can have some methods with body | ❌ Only method names (before .. 8) |
| Inheritance | extends keyword | implements keyword |
| Multiple Inheritance | ❌ Not supported | ✅ Allowed via multiple interfaces |
| Variable Types | Normal variables allowed | Only constants (public static final) |
| Use Case | Partial abstraction | Total abstraction / multiple behavior |

**🔚 Summary**

| **Topic** | **Urdu Summary** |
| --- | --- |
| Abstract Class | Aadhi tayar class — method body aur baghair dono |
| Interface | Sirf method naam — class ko poora implement karna hota hai |

**🔶 Remaining/Extra Important Topics (Check These Too)**

**1. Constructors**

**English**: Special methods used to initialize objects automatically.  
**Urdu**: Jab object banayen to constructor **automatic run hota hai**.

📌 Types:

* Default Constructor
* Parameterized Constructor

..

..

class Student {

String name;

Student(String n) { // Constructor

name = n;

}

void show() {

System.out.println("Name: " + name);

}

public static void main(String[] args) {

Student s1 = new Student("Alishbah");

s1.show();

}

}

**2. Method Overloading (Compile-time Polymorphism)**

**English**: Same method name, different parameters.  
**Urdu**: Aik hi method ka naam, lekin alag arguments.

..

..

class MathOp {

int add(int a, int b) {

return a + b;

}

double add(double a, double b) {

return a + b;

}

}

**3. Method Overriding (Run-time Polymorphism)**

**English**: Subclass changes method of parent class.  
**Urdu**: Child class parent ki method ko overwrite karti hai.

..

..

class Animal {

void sound() {

System.out.println("Animal sound");

}

}

class Dog extends Animal {

void sound() {

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

}

}

**4. Static Keyword**

**English**: Belongs to class, not object. Used for **shared** variables/methods.  
**Urdu**: static cheez class ki hoti hai, har object ka alag nahi hota.

..

..

class Counter {

static int count = 0;

Counter() {

count++;

System.out.println(count);

}

public static void main(String[] args) {

new Counter(); // 1

new Counter(); // 2

}

}

**5. ‘this’ Keyword**

**English**: Refers to **current object**.  
**Urdu**: this keyword current object ko refer karta hai.

..

..

class Person {

String name;

Person(String name) {

this.name = name; // this = current object

}

}

**6. Access Modifiers**

**English**: Control who can access what.  
**Urdu**: Kis class ya object ko kis cheez ka access milega.

| **Modifier** | **Access Scope** |
| --- | --- |
| public | Har jagah access |
| private | Sirf usi class mein |
| protected | Same package ya subclass |
| (default) | Same package only |

**🟩 1. EASY LEVEL PRACTICE PROGRAMS (Must-do for Paper)**

**✅ Program 1: Sum of Array Elements**

..

..

public class SumArray {

public static void main(String[] args) {

int[] arr = {10, 20, 30, 40};

int sum = 0;

for (int num : arr) {

sum += num;

}

System.out.println("Total: " + sum);

}

}

**✅ Program 2: Check Even or Odd (if-else)**

..

..

public class EvenOdd {

public static void main(String[] args) {

int num = 7;

if (num % 2 == 0) {

System.out.println("Even Number");

} else {

System.out.println("Odd Number");

}

}

}

**✅ Program 3: Factorial Using for Loop**

..

..

public class Factorial {

public static void main(String[] args) {

int n = 5, fact = 1;

for (int i = 1; i <= n; i++) {

fact \*= i;

}

System.out.println("Factorial: " + fact);

}

}

**✅ Program 4: Use of Constructor and this keyword**

..

..

class Student {

String name;

Student(String name) {

this.name = name;

}

void display() {

System.out.println("Name: " + name);

}

public static void main(String[] args) {

Student s = new Student("Alishbah");

s.display();

}

}

**✅ Program 5: Print Table of a Number**

..

..

public class Table {

public static void main(String[] args) {

int n = 4;

for (int i = 1; i <= 10; i++) {

System.out.println(n + " x " + i + " = " + (n \* i));

}

}

}

**🟨 2. TRICKY / THORA LOGIC WALA PROGRAMS**

**🔶 Program 6: Reverse a String**

..

..

public class ReverseString {

public static void main(String[] args) {

String str = "OOP..";

String rev = "";

for (int i = str.length() - 1; i >= 0; i--) {

rev += str.charAt(i);

}

System.out.println("Reverse: " + rev);

}

}

**🔶 Program 7: Palindrome Number (e.g. 121, 353)**

..

..

public class Palindrome {

public static void main(String[] args) {

int num = 121, rev = 0, temp = num;

while (temp != 0) {

rev = rev \* 10 + temp % 10;

temp /= 10;

}

if (num == rev)

System.out.println("Palindrome");

else

System.out.println("Not Palindrome");

}

}

**🔶 Program 8: Find Largest of 3 Numbers**

..

..

public class Largest {

public static void main(String[] args) {

int a = 10, b = 30, c = 25;

if (a > b && a > c)

System.out.println("Largest: " + a);

else if (b > c)

System.out.println("Largest: " + b);

else

System.out.println("Largest: " + c);

}

}

**🔶 Program 9: Use of try-catch (Exception Handling)**

..

..

public class TryCatchExample {

public static void main(String[] args) {

try {

int x = 5 / 0; // Error: divide by 0

} catch (ArithmeticException e) {

System.out.println("Can't divide by zero!");

}

}

}

**🔶 Program 10: File Writing (File Handling)**

..

..

import ...io.\*;

public class WriteFile {

public static void main(String[] args) {

try {

FileWriter fw = new FileWriter("output.txt");

fw.write("OOP .. is easy!");

fw.close();

System.out.println("File written successfully.");

} catch (IOException e) {

System.out.println("File error: " + e.getMessage());

}

}

}

**🎯 Final Practice Program – 13 Marks (Based on Abstract Class + Interface)**

**📚 Program Topic: *Employee Salary Management System***

**✅ Program Explanation (in simple words):**

* Ek abstract class hogi Person, jisme name, age aur abstract method displayInfo() hoga.
* Ek interface hoga Payable jisme calculateSalary() method hoga.
* Employee class Person ko extend karegi aur Payable ko implement karegi.
* Har employee ka base salary input lenge, us par bonus add karenge.
* Exception use karenge agar salary negative ho.
* File mein result write karenge.
* Loop se multiple employees input karenge.

**💻 FULL CODE:**

..

..

import ...io.\*;

import ...util.\*;

// Interface

interface Payable {

double calculateSalary(); // abstract method

}

// Abstract class

abstract class Person {

String name;

int age;

Person(String name, int age) {

this.name = name;

this.age = age;

}

abstract void displayInfo(); // abstract method

}

// Subclass

class Employee extends Person implements Payable {

private double baseSalary;

private double bonus;

Employee(String name, int age, double baseSalary) {

super(name, age);

this.baseSalary = baseSalary;

this.bonus = 0.10 \* baseSalary; // 10% bonus

}

// Method overriding

@Override

void displayInfo() {

System.out.println("\nName: " + name);

System.out.println("Age: " + age);

System.out.println("Total Salary: " + calculateSalary());

}

// Interface method

public double calculateSalary() {

return baseSalary + bonus;

}

}

public class SalarySystem {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

try {

System.out.print("Enter number of employees: ");

int n = sc.nextInt();

Employee[] employees = new Employee[n];

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

System.out.println("\n--- Employee " + (i + 1) + " ---");

System.out.print("Enter name: ");

String name = sc.next();

System.out.print("Enter age: ");

int age = sc.nextInt();

System.out.print("Enter base salary: ");

double salary = sc.nextDouble();

if (salary < 0) {

throw new Exception("Salary cannot be negative.");

}

employees[i] = new Employee(name, age, salary);

}

// Write to file

FileWriter fw = new FileWriter("employees.txt");

for (Employee e : employees) {

e.displayInfo();

fw.write(e.name + " | Age: " + e.age + " | Salary: " + e.calculateSalary() + "\n");

}

fw.close();

System.out.println("\nData saved in 'employees.txt'");

} catch (Exception e) {

System.out.println("Error: " + e.getMessage());

}

}

}

**🧠 What to Revise for This Program in Paper:**

| **Concept** | **How It’s Used** |
| --- | --- |
| **Abstract Class** | Person class with abstract method displayInfo() |
| **Interface** | Payable with method calculateSalary() |
| **Inheritance** | Employee extends Person |
| **Polymorphism** | displayInfo() overridden |
| **Encapsulation** | private baseSalary, bonus used internally |
| **Constructor** | Employee(String, int, double) |
| **If-Else** | Salary check using if (salary < 0) |
| **Exception Handling** | try-catch with throw |
| **Loop** | for loop for multiple employees |
| **File Handling** | FileWriter saves output |

**🎁 BONUS TIP:**

Mam agar kehti hain:

"Program mein abstract class aur interface dono use karna chahiye"

Toh **yehi wala logic likh do**, bas variable names change kar lena (e.g., Employee → Student, Salary → Marks etc.) and **tum full marks le jaogi**! 🏆

**📘 🔹 Section A: Most Expected MCQs (with correct answer starred ⭐)**

**1. Which of the following is not an OOP concept?**  
A) Inheritance  
B) Encapsulation  
⭐ C) Compilation  
D) Polymorphism

**2. Which keyword is used to create an abstract class?**  
A) interface  
⭐ B) abstract  
C) class  
D) static

**3. Which method must be defined inside an interface?**  
A) static  
⭐ B) abstract  
C) final  
D) constructor

**4. FileWriter class belongs to which package?**  
A) ...util  
⭐ B) ...io  
C) ...file  
D) ...text

**5. Which loop is best when number of iterations is known?**  
⭐ A) for loop  
B) while loop  
C) do-while  
D) switch

**6. What happens if exception is not handled?**  
A) Program continues  
⭐ B) Program crashes  
C) Error ignored  
D) Variable reset

**7. Which is the correct syntax for inheritance in ..?**  
⭐ A) class B extends A  
B) class A extends B  
C) class A inherit B  
D) inherit A : B

**8. What is the return type of a constructor?**  
A) int  
B) void  
C) static  
⭐ D) No return type

**9. Which access modifier allows full access from outside the class?**  
⭐ A) public  
B) private  
C) protected  
D) default

**10. To prevent division by zero, which concept is used?**  
A) Overriding  
⭐ B) Exception Handling  
C) Inheritance  
D) File Handling

1. **Which keyword is used to inherit a class?**  
   A) implements  
   B) inherits  
   C) extends  
   D) uses

**Answer:** C) extends

1. **What will happen if an exception is not handled?**  
   A) Program runs normally  
   B) Program terminates abruptly  
   C) Exception is ignored  
   D) Program restarts

**Answer:** B) Program terminates abruptly

1. **Which of these is not a .. access modifier?**  
   A) public  
   B) private  
   C) protected  
   D) external

**Answer:** D) external

1. **Which of these cannot be instantiated?**  
   A) Class  
   B) Interface  
   C) Object  
   D) Method

**Answer:** B) Interface

1. **What is method overriding?**  
   A) Two methods with the same name but different parameters  
   B) Subclass provides a specific implementation of a method from superclass  
   C) Writing two methods in one class  
   D) None of the above

**Answer:** B) Subclass provides a specific implementation of a method from superclass

1. **Which of the following is used to handle exceptions in ..?**  
   A) try-catch  
   B) if-else  
   C) switch  
   D) for loop

**Answer:** A) try-catch

1. **Which class is used to write data to a file?**  
   A) FileReader  
   B) FileWriter  
   C) Scanner  
   D) BufferedReader

**Answer:** B) FileWriter

1. **What does the final keyword mean in ..?**  
   A) Variable or method cannot be changed or overridden  
   B) Variable is global  
   C) Class is abstract  
   D) Method is static

**Answer:** A) Variable or method cannot be changed or overridden

1. **Which loop executes the body at least once?**  
   A) for  
   B) while  
   C) do-while  
   D) switch

**Answer:** C) do-while

1. **What is encapsulation?**  
   A) Hiding data and providing access through methods  
   B) Extending a class  
   C) Multiple forms of methods  
   D) Abstract class usage

**Answer:** A) Hiding data and providing access through methods

**🔥 Output-Based Questions Practice (3 marks each)**

**Q1. What is the output of the following code?**

..

..

class Test {

public static void main(String[] args) {

int[] arr = {5, 10, 15, 20};

int sum = 0;

for(int i = 0; i < arr.length; i++) {

if(arr[i] % 10 == 0) {

sum += arr[i];

}

}

System.out.println(sum);

}

}

**Q2. What is the output of this code?**

..

..

interface A {

void show();

}

class B implements A {

public void show() {

System.out.println("Hello from B");

}

}

public class Main {

public static void main(String[] args) {

A obj = new B();

obj.show();

}

}

**🐞 Error Identification Question (3 marks)**

Find and fix the error in the following code:

..

..

abstract class Animal {

abstract void sound();

void sleep() {

System.out.println("Sleeping.");

}

}

class Dog extends Animal {

void sound() {

System.out.println("Bark");

}

}

public class Main {

public static void main(String[] args) {

Animal a = new Animal();

a.sound();

}

}

**Output-Based Questions Answers**

**Q1. Output?**

..

..

int[] arr = {5, 10, 15, 20};

int sum = 0;

for(int i = 0; i < arr.length; i++) {

if(arr[i] % 10 == 0) {

sum += arr[i];

}

}

System.out.println(sum);

**Explanation:**  
Only elements divisible by 10 are added: 10 and 20  
Sum = 10 + 20 = 30

**Output:**  
30

**Q2. Output?**

..

..

interface A {

void show();

}

class B implements A {

public void show() {

System.out.println("Hello from B");

}

}

public class Main {

public static void main(String[] args) {

A obj = new B();

obj.show();

}

}

**Explanation:**  
obj is an interface reference to class B object. Calling show() prints the overridden method.

**Output:**  
Hello from B

**🐞 Error Identification Question**

..

..

abstract class Animal {

abstract void sound();

void sleep() {

System.out.println("Sleeping.");

}

}

class Dog extends Animal {

void sound() {

System.out.println("Bark");

}

}

public class Main {

public static void main(String[] args) {

Animal a = new Animal(); // ERROR here

a.sound();

}

}

**Error:**  
You **cannot create an object of abstract class** Animal.

**Fix:**

Create object of subclass Dog instead:

Animal a = new Dog();

a.sound();

**🔧 Short Snippet Code (Example Answer)**

public class EvenNumbers {

public static void main(String[] args) {

for(int i = 1; i <= 10; i++) {

if(i % 2 == 0) {

System.out.println(i);

}

}

}

}

**🔧 More Short Snippet Codes (3 marks each)**

**Snippet 1: Check if a number is prime or not**

..

..

public class PrimeCheck {

public static void main(String[] args) {

int num = 7;

boolean isPrime = true;

for(int i = 2; i <= num/2; i++) {

if(num % i == 0) {

isPrime = false;

break;

}

}

if(isPrime) {

System.out.println(num + " is prime");

} else {

System.out.println(num + " is not prime");

}

}

}

**Snippet 2: Use ArrayList to add and print elements**

..

..

import ...util.ArrayList;

public class ArrayListExample {

public static void main(String[] args) {

ArrayList<String> list = new ArrayList<String>();

list.add("Apple");

list.add("Banana");

list.add("Mango");

for(String fruit : list) {

System.out.println(fruit);

}

}

}

**Snippet 3: Demonstrate method overloading**

..

..

public class Overloading {

void show(int a) {

System.out.println("Integer: " + a);

}

void show(String b) {

System.out.println("String: " + b);

}

public static void main(String[] args) {

Overloading obj = new Overloading();

obj.show(5);

obj.show("Hello");

}

}

**Snippet 4: Simple try-catch for exception handling**

..

..

public class ExceptionExample {

public static void main(String[] args) {

try {

int data = 50 / 0;

} catch(ArithmeticException e) {

System.out.println("Cannot divide by zero");

}

}

}

**Snippet 5: Interface example with two methods**

..

..

interface Drawable {

void draw();

void resize();

}

class Circle implements Drawable {

public void draw() {

System.out.println("Drawing circle");

}

public void resize() {

System.out.println("Resizing circle");

}

}

public class TestInterface {

public static void main(String[] args) {

Circle c = new Circle();

c.draw();

c.resize();

}

}