**Final Exam-2021**

**[A] Fill in the blanks in cach of the following statements:**

**i) If a class declares constructors, the compiler will not create a default constructor .**

**ii) The public methods of a class are also known as the class's interference or API.**

**iii) Lists and tables of values can be stored in array and applications**

**iv) The number used to refer to a particular array clement is called the index.**

**v) A variable known only within the method in which it's declared is called a local variable**

**vi) It's possible to have several methods with the same name that cach operate on different types or numbers of arguments. This feature is called method overloading.**

**vii) Typically,for statements are used for counter-controlled repetition and while statements for sentinel-controlled repetition.**

**viii) Methods that perform common tasks and do not require objects are called static methods**

**[ B] Write a Java statement or a set of Java statements to accomplish each of the following tasks:**

1. **Sum the odd integers between 1 and 99, using a for statement. Assume that the integer variables sum and Count have been declared.**

int sum = 0;

int count = 0;

for (int i = 1; i <= 99; i += 2) {

sum += i;

count++;

}

1. **Print the integers from 1 to 20, using a while loop and the counter variable i. Assume that the variable i has been declared, but not initialized. Print only five integers per line.**

int i = 1;

while (i <= 20) {

System.out.print(i + " ");

if (i % 5 == 0) {

System.out.println();

}

i++;

}

**c) Repeat part (c), using a for statement.**

for (int i = 1; i <= 20; i++) {

System.out.print(i + " ");

if (i % 5 == 0) {

System.out.println();

}

}

**C] i) What gives Java its 'write once and run anywhere' nature?**

Java's 'write once and run anywhere' nature is achieved through its platform independence, which is enabled by the Java Virtual Machine (JVM). Java source code is compiled into bytecode, which can run on any device with a JVM, regardless of the underlying hardware and operating system.

**ii) What happens at runtime during Java compilation?**

During Java compilation, the Java source code is translated into bytecode by the Java compiler. This bytecode is then executed by the JVM at runtime.

**iii) Can you save a Java source file by another name than the class name?**

Yes, you can save a Java source file with a name different from the class name. However, the file name should match the public class name within the file.

**iv) Can you have multiple classes in a java source file?**

Yes, you can have multiple classes in a Java source file, but only one of them should be declared as public, and the file name should match the name of that public class.

**[D] Write a Java program to create and display unique three-digit number using 1, 2, 3, 4. Also count how many three-digit numbers are there.**

public class UniqueThreeDigitNumbers {

public static void main(String[] args) {

int count = 0;

for (int i = 1; i <= 4; i++) {

for (int j = 1; j <= 4; j++) {

for (int k = 1; k <= 4; k++) {

if (i != j && j != k && i != k) {

int number = i \* 100 + j \* 10 + k;

System.out.println(number);

count++;

}

}

}

}

System.out.println("Total unique three-digit numbers: " + count);

}

}

**2)**

**[A] What are the various access specifiers in Java? Write an example of public access modifier.**

The various access specifiers in Java are:

Public: The member can be accessed from any other class.

Protected: The member can only be accessed within its own package and by subclasses.

Default (no specifier): The member is accessible only within its own package.

Private: The member is accessible only within its own class.

Example of the public access modifier:

public class MyClass {

public int publicVariable;

public void publicMethod() {

// Code for the public method

}

}

**B)Write the rules of Constructor. What is the purpose of a default constructor? Explain with example.**

Constructor Rules:

* Constructor name must be the same as the class name.
* Constructors do not have a return type, not even void.
* A class can have multiple constructors with different parameters (overloading).
* Constructors can call other constructors using this().

Purpose of Default Constructor:

A default constructor is automatically generated by the compiler if no constructor is defined in the class. Its purpose is to initialize the object with default values. For example:

public class MyClass {

// Default constructor

public MyClass() {

// Initialization code

}

}

**[C] i) What is the output of the following Java program?**

**public class Test**

**{**

**Test(int a, int b)**

**{**

**System.out.println("a= "+a+" b="+b); }**

**Test(int a, float b)**

**{ System.out.println("a= "+a+" b = "+b);**

**}**

**public static void main (String args[])**

**{**

**byte a = 10;**

**byte b = 15;**

**Test test = new Test(a,b);**

**}}**

Output:

a= 10 b=15

**ii)**

**class Test**

**{**

**int i; }**

**public class Main**

**{**

**public static void main (String args[])**

**{**

**Test test = new Test();**

**System.out.println(test.i);**

**}**

**}**

Output: 0

**iii)**

**class Test**

**{**

**public static void main (String args[]) {**

**for(int i=0; 0; i++) {**

**System.out.println("Hello PSTU CSE");**

**}**

**} }**

Output: the program will not compile successfully, and there won't be any output.

Top of Form

**[D] Write a Java program to print a pyramid using star pattern. Number of rows input from keyboard.**

import java.util.Scanner;

public class PyramidPattern {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of rows for the pyramid: ");

int rows = scanner.nextInt();

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

for (int j = 1; j <= rows - i; j++)

System.out.print(" ");

for (int k = 1; k <= 2 \* i - 1; k++)

System.out.print("\*");

System.out.println();

}

}

}

**3)**

**[A] What is the static variable? Explain a java program with and without static variable.**

Static variable:

* A static variable is a variable that belongs to the class rather than an instance of the class.
* It is shared among all instances (objects) of the class.
* Declared using the static keyword.

Example with static variable:

public class ExampleStaticVariable {

static int count = 0;

public ExampleStaticVariable() {

count++;

}

public static void main(String[] args) {

ExampleStaticVariable obj1 = new ExampleStaticVariable();

ExampleStaticVariable obj2 = new ExampleStaticVariable();

System.out.println("Number of objects created: " + count);

}

}

Example without static variable:

public class ExampleWithoutStaticVariable {

int count = 0;

public ExampleWithoutStaticVariable() {

count++;

}

public static void main(String[] args) {

ExampleWithoutStaticVariable obj1 = new ExampleWithoutStaticVariable();

ExampleWithoutStaticVariable obj2 = new ExampleWithoutStaticVariable();

// Each object has its own count variable

System.out.println("Object 1 count: " + obj1.count);

System.out.println("Object 2 count: " + obj2.count);

}

}

**[B] i) What is the difference between static (class) method and instance method?**

Difference between static (class) method and instance method:

* Static (class) method is associated with the class itself, not with instances of the class. It is declared using the static keyword.
* Instance method is associated with an instance (object) of the class and can access instance variables.

**ii) What are the main uses of this keyword?**

Main uses of the this keyword:

* It is used to refer to the current instance of the class.
* It is used to distinguish instance variables from local variables when they have the same name.
* It can be used to invoke the current object's method.

**C) Define Object and Class. Write Object and Class Example: main outside the class and main within the class.**

**Object:**

* An object is an instance of a class in Java.
* It is a runtime entity and represents a real-world entity that has a state and behavior.
* Objects are created from classes and have characteristics (attributes) and behaviors (methods).

**Class:**

* A class is a blueprint or template for creating objects.
* It defines a data structure along with methods to work on the data.
* Classes encapsulate data and behavior into a single unit.

Object and Class Example:

// Class definition

class Car {

// Instance variables

String brand;

String model;

int year;

// Method to display information

void displayInfo() {

System.out.println("Brand: " + brand + ", Model: " + model + ", Year: " + year);

}

}

public class Main {

public static void main(String[] args) {

// Creating objects

Car car1 = new Car();

Car car2 = new Car();

// Setting object properties

car1.brand = "Toyota";

car1.model = "Camry";

car1.year = 2022;

car2.brand = "Honda";

car2.model = "Accord";

car2.year = 2021;

// Calling object methods

car1.displayInfo();

car2.displayInfo();

}

}

Main Outside the Class:

public class Main {

public static void main(String[] args) {

// Code outside the class

}

}

Main Within the Class:

public class MyClass {

public static void main(String[] args) {

// Code within the class

}

}

**[D] Write a Java program to sort an array of given integers using the Bubble sorting Algorithm Original Array:[7, 5, 3, 2, 1, 0, 45]**

**Sorted Array: [-5, 0, 1, 2, 3, 7, 451]**

public class BubbleSortExample {

public static void main(String[] args) {

int[] arr = {7, 5, 3, 2, 1, 0, 45};

System.out.println("Original Array: " + Arrays.toString(arr));

// Sorting using Bubble Sort

for (int i = 0; i < arr.length - 1; i++) {

for (int j = 0; j < arr.length - i - 1; j++) {

if (arr[j] > arr[j + 1]) {

// Swap if the element found is greater than the next element

int temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

System.out.println("Sorted Array: " + Arrays.toString(arr));

}

}

**4)**

**[A] Differentiate between the throw and throws keyword.**

* **throw** is used to explicitly throw an exception in a program.
* **throws** is used in the method signature to declare the exceptions that a method might throw.

**[B] "Aggregation represents HAS-A relationship."-explain with example.**

* Aggregation is a type of association between two classes, representing a "HAS-A" relationship.
* In aggregation, one class contains an object of another class.
* It implies a more permanent relationship where one class is a part of another class.

**Example :**

class Address {

String street;

String city;

String state;

Address(String street, String city, String state) {

this.street = street;

this.city = city;

this.state = state;

}

}

class Employee {

int empId;

String empName;

Address empAddress; // Aggregation

Employee(int empId, String empName, Address empAddress) {

this.empId = empId;

this.empName = empName;

this.empAddress = empAddress;

}

void displayInfo() {

System.out.println("Employee ID: " + empId + ", Employee Name: " + empName);

System.out.println("Address: " + empAddress.street + ", " + empAddress.city + ", " + empAddress.state);

}

}

**[C] Is it possible to make any class read-only or write-only in java? How?**

* In Java, it's not directly possible to make a class read-only or write-only.
* However, you can control access to the class's properties by providing only getter methods (read-only) or only setter methods (write-only).

**D) What is the use of instance initializer block while we can directly assign a value in instance data member?**

* Instance initializer block is used to initialize instance variables.
* It is executed each time an instance of the class is created.
* It is useful when you want to perform complex initialization that cannot be done at the time of variable declaration.

**Example:**

class Example {

int x;

// Instance initializer block

{

x = 10;

System.out.println("Instance initializer block executed");

}

// Constructor

Example() {

System.out.println("Constructor executed");

}

}

**[E] How can you achieve abstraction in java?**

* Abstraction in Java is achieved through abstract classes and interfaces.
* Abstract classes can have abstract methods (methods without a body) that must be implemented by the concrete subclasses.
* Interfaces can declare abstract methods that must be implemented by classes that implement the interface.

**5)**

**A) Write a java program for demonstrating several thread states.**

public class ThreadStateExample {

public static void main(String[] args) {

Thread thread = new Thread(() -> {

// Thread logic

for (int i = 0; i < 5; i++) {

try {

Thread.sleep(500);

System.out.println(Thread.currentThread().getName() + " is running");

} catch (InterruptedException e) {

e.printStackTrace();

}

}

});

System.out.println("Thread State: " + thread.getState()); // NEW

thread.start();

System.out.println("Thread State after start(): " + thread.getState()); // RUNNABLE

try {

thread.join();

} catch (InterruptedException e) {

e.printStackTrace();

}

System.out.println("Thread State after join(): " + thread.getState()); // TERMINATED

}

}

**B) "Java doesn't allow the return type-based overloading, but JVM always allows return type-based overloading."- justify the statement with example.**

Java does not allow return type-based overloading. It considers the return type of a method as part of its signature, and methods with the same name and parameter types but different return types are considered the same. This is because the JVM cannot distinguish between them based on their return types alone.

Example

public class ReturnTypeOverloadingExample {

// Compile-time error: Method with a different return type

// public int add(int a, int b) { ... }

public double add(int a, int b) {

return a + b;

}

public static void main(String[] args) {

ReturnTypeOverloadingExample example = new ReturnTypeOverloadingExample();

System.out.println(example.add(5, 10));

}

}

**C)Multiple inheritances is not supported through class in java, but it is possible by an interface, why?**

Java supports multiple inheritances through interfaces to avoid the Diamond Problem. The Diamond Problem occurs when a class inherits from two classes that have a common ancestor. If both parent classes have the same method, the compiler doesn't know which method to call.

Interfaces provide a way to achieve multiple inheritances because a class can implement multiple interfaces without any conflict.

**Example:**

interface A {

void methodA();

}

interface B {

void methodB();

}

// Multiple inheritances through interfaces

class MyClass implements A, B {

public void methodA() {

System.out.println("Method A");

}

public void methodB() {

System.out.println("Method B");

}

}

**D] Can we initialize blank final variable? How?**

A blank final variable is a final variable that is not initialized at the time of declaration. It must be initialized in the constructor of the class. Once initialized, its value cannot be changed.

Example :

public class BlankFinalVariableExample {

final int blankFinalVar;

// Constructor to initialize the blank final variable

public BlankFinalVariableExample(int value) {

blankFinalVar = value;

}

public static void main(String[] args) {

BlankFinalVariableExample example = new BlankFinalVariableExample(10);

System.out.println("Blank Final Variable: " + example.blankFinalVar);

}

}

**6)**

**i) class Dog{**

**public static void main(String args[]){**

**Dog d=null;**

**System.out.println(d instanceof Dog);**

**}}**

**Output:** false

**ii)public class JavaExceptionExample {**

**public static void main(String args[]) {**

**try{**

**int data=100/0;**

**}catch(ArithmeticException e){Syste**

**m.out.println(c);}**

**System.out.println("rest of the code...**

**");**

**}**

**}**

**Output:**

java.lang.ArithmeticException: / by zero

rest of the code...

(**B)How to access package from another package?**

To access a class or interface from one package in another package, you need to follow these steps:

1. **Import the Package:** Use the **import** statement in the class where you want to use the classes or interfaces from another package. You can import a specific class/interface or the entire package using the **\*** wildcard.

// Import a specific class

import com.example.otherpackage.OtherClass;

// Import the entire package

import com.example.otherpackage.\*;

1. **Create an Object:** After importing, you can create an object of the class you want to use.

OtherClass obj = new OtherClass();

1. **Access Members:** Now you can access the methods and fields of the object as needed.

obj.someMethod();

**(C) What is the purpose of join method?**

In Java, the join() method is used to wait for a thread to finish its execution. The join() method allows one thread to wait until another thread completes its execution. When a thread calls the join() method on another thread, the calling thread will be blocked until the specified thread completes.

Example:

Thread thread1 = new Thread(() -> {

// Some task

});

Thread thread2 = new Thread(() -> {

// Some task

});

thread1.start();

thread2.start();

try {

thread1.join(); // Wait for thread1 to finish

thread2.join(); // Wait for thread2 to finish

} catch (InterruptedException e) {

e.printStackTrace();

}

// Code here will be executed after both threads complete thei

**[D] How to perform two tasks by two threads?**

To perform two tasks concurrently using two threads, you can create two separate threads and assign different tasks to each thread.

Example:

class Task1 implements Runnable {

public void run() {

// Task 1

}

}

class Task2 implements Runnable {

public void run() {

// Task 2

}

}

public class TwoTaskExample {

public static void main(String[] args) {

Thread thread1 = new Thread(new Task1());

Thread thread2 = new Thread(new Task2());

thread1.start();

thread2.start();

}

}

**(E) What is the Thread Scheduler and what is the difference between preemptive scheduling and time slicing?**

* **Thread Scheduler:** The thread scheduler is responsible for managing the execution of threads in a multithreaded environment. It determines the order in which threads run on the CPU.
* **Preemptive Scheduling:** In preemptive scheduling, the operating system can interrupt a currently running thread to start or resume another thread. This interruption is typically based on priority levels assigned to threads. The scheduler decides when to interrupt a running thread and switch to another.
* **Time Slicing:** Time slicing is a form of preemptive scheduling where each thread is allocated a fixed time slice or time quantum. When a thread's time slice expires, the scheduler interrupts that thread and gives the CPU to the next thread in the queue. This ensures that all threads get a fair share of CPU time.