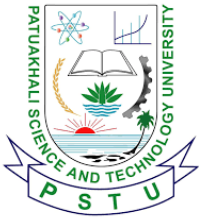
Assignment No.-01

Final Examination Question Solve



Submitted T0:

Dr.Md. Sumsuzzaman Sobuz Sir

Department of Computer & Communication Engineering

Faculty of Computer Science And Engineering

Submitted By

Name : *Mohammed Sakib Hasan*

Id : *2102052*

Faculty Of Computer Science And Engineering

Session *: 2021-2022*

Submite Date : 06/09/2023

1 [A] Fill in the blanks in each of the following statements:

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

ii) The public methods of a class are also known as the class's interface or API (Application Programming Interface).

iii) Lists and tables of values can be stored in arrays and databases .

iv) The number used to refer to a particular array element is called the elements 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 each 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 methods static methods**.**

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

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

Answer

```
int sum = 0;
int count = 0;
for (int i = 1; i <= 99; i++)
 {
   if (i % 2 != 0)
{
      sum += i;
      count
   }
}
```

```
System.out.println("Sum of odd integers: " + sum);

System.out.println("Count of odd integers: " + count);
```

b) 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.

Answer

```
int i = 1;

while (i <= 20)

 {

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

   i++;

   // Check if we need to start a new line after printing five integers

   if (i % 6 == 1) {

      System.out.println();

}
```

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

[C]

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

Answer:

Java achieves its "write once, run anywhere" nature through the use of the Java Virtual Machine (JVM). Java source code is compiled into bytecode, which is platform-independent and can be executed on any device or operating system that has a compatible JVM. This bytecode is not tied to the specifics of the underlying hardware or operating system, allowing Java programs to be portable across different platforms

ii) What happens at runtime during Java compilation?

Answer:

Java compilation actually occurs before runtime. During Java compilation, the Java source code (.java files) is translated into bytecode (.class files) by the Java compiler. The bytecode contains instructions that the JVM can execute. Compilation involves syntax checking, type checking, and generating the bytecode. If there are any compilation errors, the code won't be successfully compiled into bytecode.

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

Answer:

Yes, you can save a Java source file with a name different from the class name, but it's a convention to have the public class name match the filename (excluding the ".java" extension).

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

Answer:

Yes, you can have multiple classes in a single Java source file, but only one of them should be declared as `public`, and the public class must match the filename (excluding the ".java" extension). Non-public classes within the same file can have different names.

[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.

Answer:

```
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 && i != k && j != 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.

Answer :

In Java, there are four main access specifiers (or access modifiers) that control the visibility and accessibility of class members (fields, methods, and constructors) within a class. These access specifiers are:

1. public: Members marked as public are accessible from any class or package. They have the widest visibility and can be accessed from anywhere in the program.

Example:

```
public class MyClass
{
    public int publicField;
    public MyClass()
{
    }
    public void publicMethod()
{
    }
}
```

Members with the public access modifier are widely accessible and can be used by any part of the program.

[B]

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

Answer :

Rules of Constructors in Java:

1. Constructor Name: The constructor name must be the same as the class name.
2. No Return Type: Constructors do not have a return type, not even void.
3. Overloading: You can have multiple constructors in a class with different parameter lists (constructor overloading).
4. Implicit Default Constructor: If you don't provide any constructors in a class, Java provides a default constructor with no arguments.
5. Explicit Default Constructor: If you provide any constructors (with or without parameters), and you want to keep the default constructor as well, you must explicitly define it in your class

Purpose of a Default Constructor:

The purpose of a default constructor in Java is to initialize an object with default values or perform any necessary setup when an object is created without providing specific constructor arguments.

Example of Default Constructor:

```
public class Person {
   String name;
   int age;
   public Person() {
      name = "Unknown";
      age = 0;
   }
```

```
    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
    public void displayInfo() {
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
    }

    public static void main(String[] args) {
        Person person1 = new Person();
        Person person2 = new Person("Alice", 25);
        System.out.println("Person 1:");
        person1.displayInfo();
        System.out.println("Person 2:");
        person2.displayInfo();
    }
}
```

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

```
public class Test
{
Test(int a, int b)
{
System.out.println(&quot;a = &quot;+a+&quot; b = &quot;+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);

} }

Answer :

The output of the given Java program will be an error, and the program will not compile successfully. The reason is that there is no constructor in the Test class that accepts two byte arguments (a and b), so the line Test test = new Test(a, b); will result in a compilation error.

ii)

class Test

{

int i; }

public class Main

{

public static void main (String args[])

{

Test test = new Test();

System.out.println(test.i);

} }

Answer:

The output of the second Java program will be 0. In this program, an instance of the Test class is created, and the default value of an integer variable (i) is printed. Since the i variable is an instance variable and not explicitly initialized, it gets the default value for its type, which is 0 for integers.

iii)

class Test

{

public static void main (String args[])

{

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

{

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

} } }

Answer :

The third Java program will not compile successfully due to a syntax error. The **for** loop condition is written as **for(int i=0; 0; i++)**, which should be in the format **for(initialization; condition; iteration)**. The condition **0** is not a valid condition, and this will result in a compilation error

[D]

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

Answer:

```
import java.util.Scanner;

public class PyramidPattern {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        // Input the number of rows from the keyboard

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

        int numRows = scanner.nextInt();

        // Print the pyramid pattern

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

            // Print spaces before stars

            for (int j = 1; j <= numRows - i; j++) {
```

```
            System.out.print(" ");
        }
        // Print stars
        for (int k = 1; k <= 2 * i - 1; k++) {
            System.out.print("*");
        }
        // Move to the next line
        System.out.println();
    }
    scanner.close();
  }
}
```

3 [A]

What is the static variable? Explain a java program with and without static variable.

Answer :

Static Variable (Class Variable):

- A static variable, also known as a class variable, is a variable that belongs to the class itself rather than any particular instance of the class.
- It is shared among all instances of the class, meaning that if one instance changes the value of a static variable, it affects all other instances.
- Static variables are declared using the static keyword and are typically used to store data that is common to all instances of the class

With Static Variable:

```
public class WithStaticVariable {
  // Static variable
  static int staticVar = 0;
  public static void main(String[] args) {
```

```
        WithStaticVariable obj1 = new WithStaticVariable();
        WithStaticVariable obj2 = new WithStaticVariable();
        // Both instances share the same static variable
        obj1.staticVar = 1;
        obj2.staticVar = 2;
        // Print the static variable from both instances
        System.out.println("obj1.staticVar: " + obj1.staticVar); // Output: obj1.staticVar: 2
        System.out.println("obj2.staticVar: " + obj2.staticVar); // Output: obj2.staticVar: 2
    }
}
```

Without Static Variable:

```
public class WithoutStaticVariable {
    int nonStaticVar = 0;
    public static void main(String[] args) {
        WithoutStaticVariable obj1 = new WithoutStaticVariable();
        WithoutStaticVariable obj2 = new WithoutStaticVariable();
        // Each instance has its own non-static variable
        obj1.nonStaticVar = 1;
        obj2.nonStaticVar = 2;
        // Print the non-static variables from both instances
        System.out.println("obj1.nonStaticVar: " + obj1.nonStaticVar); // Output: obj1.nonStaticVar: 1
        System.out.println("obj2.nonStaticVar: " + obj2.nonStaticVar); // Output: obj2.nonStaticVar: 2
    }
}
```

[B]

i) What is the difference between static (class) method and instance method?

Answer:

static methods are associated with the class itself and cannot access instance-specific data, while instance methods are associated with individual objects and can access both static and instance-specific data.

ii) What are the main uses of this keyword?

Answer:

The main uses of the "this" keyword in Java are:

1. **Reference to Current Object**: It refers to the current instance of the class and is often used to distinguish between instance variables and method parameters with the same name**.**
2. **Constructor Chaining:** It can be used to call one constructor from another constructor in the same class, allowing for constructor chaining.
3. **Passing the Current Object:** It can be used to pass the current object as an argument to another method or constructor.
4. **Returning the Current Object:** It can be used to return the current object from a method, enabling method chaining (fluent interface).
5. **Accessing Instance Variables:** It is used to access instance variables when there is a shadowing effect (local variable hides an instance variable with the same name)**.**

[C]

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

Answer:

Object is an instance of a class that represents a real-world entity, possessing both state and behavior**.**

Class is a blueprint or template for creating objects, defining their attributes (state) and methods (behavior**).**

Example with Main Outside the Class:

- Objects are created in the main method outside the class definition.
- Main method serves as an entry point to the program**.**

Example with Main Within the Class:

- Objects are created and operated on directly within the class.
- Main method is part of the class and can be used for testing or as an entry point for the program in simpler applications.

[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, 45]

Answer:

```
public class BubbleSort {
    public static void main(String[] args) {
        int[] arr = {7, -5, 3, 2, 1, 0, 45};
        System.out.println("Original Array:");
        printArray(arr);
        bubbleSort(arr);
        System.out.println("\nSorted Array:");
        printArray(arr);
    }
    public static void bubbleSort(int[] arr) {
        int n = arr.length;
        boolean swapped;

        for (int i = 0; i < n - 1; i++) {
            swapped = false;
            for (int j = 0; j < n - 1 - i; j++) {
                if (arr[j] > arr[j + 1]) {
                    int temp = arr[j];
```

```
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
                swapped = true;
            }
        }
        if (!swapped) {
            break;
        }
    }
  }
  public static void printArray(int[] arr) {
    for (int num : arr) {
        System.out.print(num + " ");
    }
    System.out.println();
  }
}
```

4 [A]

Differentiate between the throw and throws keyword.

Answer:

throw is used to raise an exception explicitly, while throws is used to declare that a method might throw exceptions, leaving the responsibility of handling those exceptions to the caller**.**

[B]

“Aggregation represents HAS-A relationship.”-explain with example.

Answer:

Aggregation represents a "HAS-A" relationship in object-oriented programming, where one class contains or is composed of another class as part of its attributes. For example, a Library class may have a collection of Book objects as a part of its attributes to represent the books it contains

```
import java.util.ArrayList;

import java.util.List;

class Book {

    private String title;

    private String author;

    private String isbn;

    public Book(String title, String author, String isbn) {

        this.title = title;

        this.author = author;

        this.isbn = isbn;

    }

}

class Library {

    private String name;

    private String address;

    private List<Book> books;

    public Library(String name, String address) {

        this.name = name;

        this.address = address;

        this.books = new ArrayList<>();

    }

    public void addBook(Book book) {

        books.add(book);

    }
```

```
}

public class Main {

   public static void main(String[] args) {

      Book book1 = new Book("The Catcher in the Rye", "J.D. Salinger", "978-0-316-76948-0");

      Book book2 = new Book("To Kill a Mockingbird", "Harper Lee", "978-0-06-112008-4");

      Library library = new Library("Central Library", "123 Main St");

      library.addBook(book1);

      library.addBook(book2);

      for (Book book : library.getBooks()) {

         System.out.println("Title: " + book.getTitle() + ", Author: " + book.getAuthor() + ", ISBN: " +
book.getIsbn());

      }

   }

}
```

[C]

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

Answer:

In Java, you can make a class read-only or write-only by controlling access to its fields using access modifiers**:**

1. **Read-Only Class**: To make a class read-only, declare its fields as private and provide only getter methods (methods that retrieve the field values) for those fields. This way, external code can only read the values but cannot modify them.
2. **Write-Only Class:** To make a class write-only, declare its fields as private and provide only setter methods (methods that set the field values) for those fields. This way, external code can only modify the values but cannot directly read them

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

Answer:

Instance initializer blocks are used in Java to initialize instance data members with complex or shared initialization logic. While simple assignments in constructors work for straightforward cases, instance initializer blocks help maintain clean and organized code, handle multiple constructors consistently, ensure proper initialization order, and separate concerns when initialization logic becomes more intricate.

[E]

How can you achieve abstraction in java?

Answer:

Abstraction in Java can be achieved through:

1. **Abstract Classes:** Use the `abstract` keyword to create classes with abstract methods that must be implemented by subclasses. Abstract classes can also have concrete methods.
2. **Interfaces:** Define interfaces using the `interface` keyword, specifying a set of abstract methods that implementing classes must provide implementations for. Multiple interfaces can be implemented by a class.

These mechanisms allow you to define a blueprint for classes while hiding implementation details, promoting code reusability, and ensuring that specific methods are implemented by subclasses or implementing classes.

5 [A] Write a java program for demonstrating several thread states.

Answer:

```
public class ThreadStateDemo {

    public static void main(String[] args) throws InterruptedException {

        Thread newThread = new Thread(() -> {

            System.out.println("Thread is in the NEW state.");

        });

        Thread runnableThread = new Thread(() -> {

            System.out.println("Thread is in the RUNNABLE state.");

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

                System.out.println("Running: " + i);

            }

        });
```

```
Object lock = new Object();
Thread blockedThread1 = new Thread(() -> {
    synchronized (lock) {
        System.out.println("Thread 1 acquired the lock and is in the BLOCKED state.");
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
});
Thread blockedThread2 = new Thread(() -> {
    synchronized (lock) {
        System.out.println("Thread 2 is trying to acquire the lock and is BLOCKED.");
    }
});
Thread waitingThread = new Thread(() -> {
    synchronized (lock) {
        try {
            System.out.println("Thread is in the WAITING state.");
            lock.wait(); // Wait indefinitely
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
});
```

```
        newThread.start();
        runnableThread.start();
        blockedThread1.start();
        blockedThread2.start();
        waitingThread.start();
        Thread.sleep(2000);
        System.out.println("Thread is in the TERMINATED state.");
    }
}
```

[B] “Java doesn't allow the return type-based overloading, but JVM always allows return type-based overloading.”- justify the statement with example.

Answer:

Java does not permit method overloading based solely on differences in return types during compile-time resolution. However, the JVM can differentiate between methods with different return types based on actual argument types during runtime execution, allowing return type-based overloading at the JVM level.

[C]

Multiple inheritance is not supported through class in java, but it is possible by an interface, why?
Answer:

Java supports multiple inheritance through interfaces because it avoids the issues related to the diamond problem:

1. **Interface Implementation:** In Java, a class can implement multiple interfaces. When a class implements multiple interfaces, it is essentially promising to provide implementations for all the methods declared in those interfaces.
2. **No Implementation Conflict:** Interfaces only declare method signatures, and they do not provide method implementations. This means there is no conflict when a class implements multiple interfaces, as long as the class provides concrete implementations for all the methods declared in those interfaces.

Here's a simple example:

```
interface A {
```

```
    void methodA();
}
interface B {
    void methodB();
}

class MyClass implements A, B {
    @Override
    public void methodA() {
        System.out.println("Implementation of methodA");
    }
    @Override
    public void methodB() {
        System.out.println("Implementation of methodB");
    }
}
```

6 [A]

What will be the output of the following Java programs?

i)

```
class Dog{
 public static void main(String args[]){
Dog d=null;
  System.out.println(d instanceof Dog);
 } }
```

Answer: False

1. ii) class A{

```
protected void msg(){System.out.println(&quot;Hello java&quot;);}}

public class Simple extends A{

void msg(){System.out.println(&quot;Hello java&quot;);}

 public static void main(String args[]){

   Simple obj=new Simple();

   obj.msg();  }}
```

Answer: Compiler error! But it'll print "Hello java" if protected is removed from the method msg of class A.

1.

iii)

```
public class JavaExceptionExample

{

 public static void main(String args[]){

    try{

  int data=100/0;

  }catch(ArithmeticException e){Syste

m.out.println(e);}

  System.out.println(&quot;rest of the code...

&quot;);

 } }
```

Answer: : java.lang.ArithmeticException: / by zero rest of the code

iv)

```
class Animal{

 void eat(){System.out.println(&quot;eating...&quot;);}  }

 class Dog extends Animal{

void bark(){System.out.println(&quot;barking...&quot;);}  }

 class Cat extends Animal{
```

void meow(){System.out.println("meowing...");} }

class TestInheritance3{

public static void main(String args[]){

Cat c=new Cat();

c.meow();

c.eat();

c.bark();}}

Answer: Compiler error (bark method not available).

Without c.bark() it will print meowing… and eating…

[B]

How to access package from another package?

Answer: : In java to access package from another one, we use the import command. For example,

import packagename.classname;

[C]

What is the purpose of join method?

Answer: join method is used when one thread waits for an another thread to finish it's job. For example, if we want to finish the process of ThreadB and then continue, we can use,

ThreadB.join();

[D]

How to perform two tasks by two threads?

Answer: : To perform two task, we can two separate threads and assign each of them two different task and start them. In this way those two will start working simultaneously.

[E]

What is the Thread Scheduler and what is the difference between preemptive scheduling and time slicing?

Answer: A component of Java that decides which thread to run or execute and which thread to wait is called a **thread scheduler in Java**. In Java, a thread is only chosen by a thread scheduler if it is in the runnable state. However, if there is more than one thread in the runnable state, it is up to the thread scheduler to pick one of the threads and ignore the other ones.

- **Preemptive scheduling** allows the scheduler to take a running thread away from the CPU and give it to another thread, even if the first thread has not finished its execution. This is done based on the priority of the threads. A thread with a higher priority will preempt a thread with a lower priority.
- **Time slicing** is a type of preemptive scheduling where each thread is given a certain amount of time to run on the CPU. After the time slice expires, the thread is preempted and another thread is given a chance to run. This process continues until all of the threads have had a chance to run.

The main difference between preemptive scheduling and time slicing is that in preemptive scheduling, the scheduler can preempt a running thread at any time, while in time slicing, the thread is only preempted after its time slice expires.