10234 1231551 4CSE-F

Assignment - 1

1 Java is platform independent but JVM is platform dependent. Justify.

→ Java's platform independence is achieved through bytecode compilation, where java source code is compiled into bytecode, which is a platform-independent format. To execute this bytecode, a Java Virtual Machine (JVM) is required. JVM is platform dependent because of following reasons

1. Interpretation

→ JVM interprets bytecode and translates it into native machine code for execution. Different OS have different machine architectures and instruction sets, requiring platform-specific JVM implementation.

2. Native libraries and System Calls

→ JVM interacts with the underlying OS through native libraries and system calls for tasks like file I/O and memory management. These interactions are platform-dependent.

3 Platform - specific Features

→ Despite Java's abstraction of platform-specific details, certain functionalities like GUI programming are inherently platform-dependent due to differences in rendering and event handling.

4 Optimizations and Performance

→ JVM implementations are optimized for specific hardware and OS architectures, requiring platform-specific tuning for improved performance

2 What is the use of wrapper class? Explain an important method of wrapper class.

→ Wrapper classes in java are used to provide an object representation of primitive data types. They allow primitive data types to be treated as objects, enabling them to be used in situations where objects are required, such as collections, generics, and method overloading.

→ An important method of wrapper classes is 'valueOf()'. This method is used to create an instance of the wrapper class from given primitive value or a string representing the primitive value. It provides a convenient way to convert primitive data types to their corresponding wrapper objects.

```
public class WrapperExample {
    public static void main (String[] args) {

        Integer intObj1 = Integer.valueOf(10);
        Integer intObj2 = Integer.valueOf("20");

        Double doubleObj1 = Double.valueOf(3.14);
        Double doubleObj2 = Double.valueOf(2.71);

        System.out.println("Integer Objects: " + intObj1
            + ", " + intObj2);
        System.out.println("Double Objects: " + doubleObj1
            + ", " + doubleObj2);
    }
}
```

3 What is Anonymous inner class? WAP to override the method using Anonymous inner class.

→ An anonymous inner class in java is a class that doesn't have a name and is defined and instantiated at the same time. It's often used for implementing interfaces or extending classes in a single expression.

```
interface Greeting {
    void greet();
}

public class AnonymousInnerClassExample {
    public static void main(String[] args) {
        Greeting greeting = new Greeting() {
            @Override
            public void greet() {
                System.out.println("Hello, from anonymous
                    inner class!");
            }
        };

        greeting.greet();
    }
}
```

4 Explain the following keyword with a proper example 1] static 2] super

1 static

→ The static keyword in java is used to declare members that belong to the class rather than to any instance of the class. It means the static member is shared among all instances of the class.

2 super

→ The super keyword in java is used to refer to the immediate parent class object. It is used to call the parent class constructor, parent class methods, or parent class variables.

```
class Animal {
    static int count = 0;
    string sound = "Animal Sound";
    Animal() { count++; }

    void makeSound() {
        System.out.println(sound);
    }
}
class Dog extends Animal {
    String sound = "Bark";

    Dog() { super(); }

    void makeSound() {
        super.makeSound();
        System.out.println(sound);
    }

    static int getCount() {
        return count;
    }
}
public class staticSuperExample {
 public static void main(String[] args) {
    Dog dog1 = new Dog();
    Dog dog2 = new Dog();
    Dog dog3 = new Dog();
```

```
dog1.makeSound();
System.out.println("Total Dogs: " +
    Dog.getCount());

    }
}
```

5 What is dynamic method dispatch? WAP to implement of dynamic method dispatch using interface.

⇒ Dynamic method dispatch is a mechanism in Java where the method to be invoked is determined at runtime based on the actual type of the object being referred to, rather than the reference type. This allows a subclass to override a method of its superclass and the method called is based on the type of objects being referred to by the reference variable.

```
interface Animal{ void makeSound(); }
class Dog implements Animal {
  @Override
  public void makeSound(){
    System.out.println("Dog barks");
  }
}
class Cat implements Animal {
  @Override
  public void makeSound() {
    System.out.println("Cat meows");
  }
}
```

```
public class DynamicMethodDispatch {
  public static void main(String[] args) {
    Animal dog = new Dog();
    Animal cat = new Cat();

    dog.makeSound();
    cat.makeSound();

    dog = new Cat();
    dog.makeSound();
  }
}
```

6 Write down differences between interface and abstract class with proper syntax and example.

Interface	Abstract Class
→ Interface is a reference type containg only constants, method signature default methods, static methods, nested types, and static constants.	Abstract class declared with abstract keyword thus can't be instantiated directly. It may contain abstract method, concrete methods, variables, constructors, and other types of members
→ Supports multiple inheritance.	Does not support multiple inheritance
→ Can't have constructor	can have constructors
→ Methods are declared but not implemeted. implementing classes must provide an implimentation for all declared methods.	Methods can be both abstract or concrete. subclasses may choose to override or inherit concrete method
→ Methods are default public and abstract. variable are public, static and final	Methods and variables can have various access modifiers like public, protected, default or private.

```
interface Animal {
    void makeSound();
}

abstract class Shape {
    abstract void draw();
    void display() {
        System.out.println("Displaying shape");
    }
}
public class Main {
  public static void main(String[] args) {
    Animal dog = new Animal() {
        public void makeSound() {
            System.out.println("Dog barks");
        }
    };
    dog.makeSound();

    Shape rectangle = new Shape() {
        void draw() {
            System.out.println("Drawing rectangle");
        }
    };
    rectangle.draw();
    rectangle.display();
  }
}
```

11 What is an Exception? Explain checked exception in detail.

→ An exception in java is an event that disrupts the normal flow of program's execution. It occurs when something unexpected happens during runtime

→ Such as an error condition or an exceptional circumstance that can't be handled by the program.

→ Exceptions provide a way to handle errors gracefully and prevent the program from crashing.

• Checked Exceptions

→ Checked exceptions are the exceptions that are checked at compile-time by the Java compiler.

→ These exceptions are subclasses of Exception but not subclasses of RuntimeException

→ Checked exceptions are generally used to represent error conditions that a program can anticipate and recover from, such as file I/O errors, network connection issues, etc.

```
import java.io.FileReader;
import java.io.IOException;

public class Main {
  public static void main(String[] args) {
    try {
        FileReader reader = new
        FileReader("temp.txt");
    } catch(IOException ex) {
        ex.printStackTrace();
    }
  }
}
```

12 What is the package? Explain different types of access specifier in java programming.

→ In Java programming, a package is a way to organize related classes, interface, and sub-package into a single namespace.

→ Packages help in grouping classes that are part of the same module or functionality, making code organization and management easier.

→ They also facilitate access control and prevent naming conflicts by providing a unique namespace for each package.

• Types of Access specifier

1. Public :- Classes, methods, and fields marked as public are accessible from anywhere, both within the same package and from other packages. This means that public members can be used by any other class in any package.

2 Protected

→ Members with the protected access specifier are accessible within the same package and by subclasses, even if they are in different package. protected members are not accessible by unrelated classes outside the package hierarchy.

3 Default (package-private)

→ When no access specifier is specified the default access specifier is applied. Classes, methods, and fields with default access are accessible only within the same package. They are not visible to classes in other packages.

4 Private

→ Members marked as private are accessible only within the same class. They can't be accessed from outside the class, not even by subclasses. private members are used to encapsulate implementation details and hide them from other classes.

7 It is required to compute SPI of n students of your college for their registered subjects in a semester. Declare a class called student having following data members: id_no, no_of_subjects_registered, subject_code, subject_credits, grade_obtained and spi

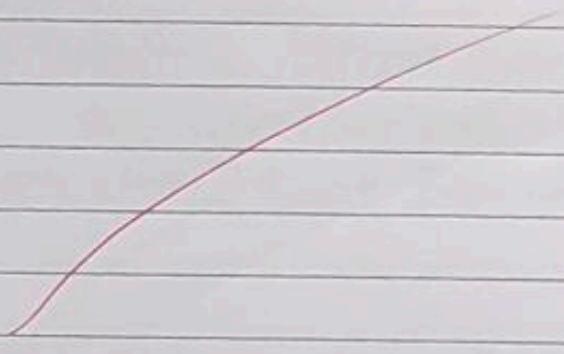
- Define constructor and calculate_spi methods
- Define main to instantiate an array of objects of class student to process data of n students to be given as cmd arguments.

```
import java.util.Scanner;

public class students {
    private int noOfSubjectsRegistered;
    private int[] subjectCode;
    private int[] subjectCredits;
    private int[] gradeObtained;
    private double[] spi;

    Students(int noOfSubjectsRegistered) {
        this.noOfSubjectsRegistered = noOfSubjectsRegistered;
        this.subjectCode = new int[noOfSubjectsRegistered];
        this.subjectCredits = new int[noOfSubjectsRegistered];
        this.gradeObtained = new int[noOfSubjectsRegistered];
        this.spi = new double[noOfSubjectsRegistered];
    }
```

```
void setDetails(int index, int subjectCode, int
        subjectCredits, int gradeObtained){
    this.subjectCode[index] = subjectCode;
    this.subjectCredits[index] = subjectCredits;
    this.gradeObtained[index] = gradeObtained;
}

void calculateSpi(int index) {
    spi[index] = (gradeObtained[index] *
                  subjectCredits[index]) / 100.0;
}

void display(int index){
    for(int i=0; i<noOfSubjectsRegistered; i++) {
        calculateSpi(i);
        System.out.println("Subject Code: " +
                           subjectCode[i]);
        System.out.println("Subject Credits: " +
                           subjectCredits[i]);
        System.out.println("Grade Obtained: " +
                           gradeObtained[i]);
        System.out.println("SPI: " + spi[i]);
    }
    double totalSpi += spi[i]; =0;
    for(int i =0 ; i < noOfSubjectsRegistered; i++){
        totalSpi+= spi[i];
    }
    double overallSpi = totalSpi / noOfSubjectRegistered;
    System.out.println("Overall SPI: " + overallSpi);
}
```

```
public static void main(String[] args) {
  Scanner sc = new Scanner(System.in);

  System.out.println("Enter how many students: ");
  int noOfStudents = sc.nextInt();

  Students[] st = new Students[noOfStudents];

  int noOfSubjectsRegistered = 0, subjectCode,
  subjectCredits, gradeObtained;

  for(int i=0; i<noOfStudents; i++) {
    System.out.print("Enter no of subjects
      for student: " + (i+1) + ": ");
    noOfSubjectsRegistered = sc.nextInt();
    st[i] = new Students(noOfSubjectsRegistered);

    for(int j=0; j<noOfSubjectsRegistered; j++){
      System.out.print("Enter subject code for
        student" + i+1 + " for subject: " + j+1 + ": ");
      subjectCode = sc.nextInt();

      System.out.print("Enter subject credits for
        student" + i+1 + " for student" + j+1 + ": ");
      subjectCredits = sc.nextInt();

      System.out.print("Enter Grade Obtained for
        student" + j+1 + " (0-100): ");
      gradeObtained = sc.nextInt();
```

```
            st[i].setDetails(j, subjectCode, subjectCredits,
                    gradeObtained);
        }
    }
    for(int i=0; i<noOfStudents; i++){
        st[i].display(i);
    }

    sc.close();
  }
}
```



8 Declare a class called a book having author_name as private data members. Extend book class to have two sub classes called book_publication & paper_publication. Each of these classes have private member called title. WAP to show usage of dynamic method dispath to display book or ppuper publication of a given author

```
import java.util.Scanner;

class BookPublication extends Book {
 private String title;

 BookPublication (String authorName, String title) {
   super(authorName);
   this.title = title;
 }
 void display() {
   System.out.println("Author Name: " +
                      super.getAuthorName());
   System.out.println("Book Publication Title: " + title);
 }

class PaperPublication extends Book {
 private string title;

 PaperPublication (String authorName, String title) {
   super(authorName);
   this.title = title;
 }
```

```
    void display() {
        System.out.println("Author Name: " +
                    super.getAuthorName());
        System.out.println("Paper Publication Title: "
            + title);
    }
}
public abstract class Book {
  private String authorName;

  Book(String authorName){ this.authorName =
                              authorName; }
  abstract void display();
  String getAuthorName() { return authorName; }

  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter Name of the Author: ");
    String authorName = sc.nextLine();
    System.out.print("Enter title ");
    String title = sc.nextLine();

    Book b = new PaperPublication(authorName, title);
    b.display();
    System.out.print("Enter Title: ");
    title = sc.nextLine();
    b = new BookPublication(authorName, title);
    b.display();
    sc.close();
  }
}
```

9 WAP to find area of different shapes using function overloading like rectangle, triangle, sphere passing different arguments to overloaded method Area().

```
import java.util.Scanner;

public class Shape {
  double area(double radius) {
     return Math.PI * radius * radius;
  }
  double area(double length, double height) {
     return ((length * height) / 2);
  }
  double area(float length, float width) {
     return length * width;
  }
  double area(float radius) {
     return 4 * Math.PI * radius * radius;
  }

public static void main(String[] args) {
  Scanner sc = new Scanner(System.in);
  Shape sp = new Shape();

  System.out.print("Enter radius: ");
  double radius = sc.nextDouble();
  System.out.print("Enter length: ");
  double length = sc.nextDouble();
```

```
System.out.print("Enter width: ");
double width = sc.nextDouble();

System.out.println("Area of Circle is: " +
  String.format("%.4f", sp.area(radius)));
System.out.println("Area of Triangle is: " +
  String.format("%.4f", sp.area(length, width)));
System.out.println("Area of Rectangle is: " +
  sp.area((float) length, (float) width));
System.out.println("Area of sphere is: " +
  String.format("%.4f", sp.area((float) radius)));

  sc.close();
  }
}
```

10 WAP to find the average of marks for student object and print the appropriate massage according to the average Marks

Avg Marks	Grade
80 to 100	Honours
60 to 79	First Division
50 to 59	second Division
40 to 49	Third Division
0 to 39	Fail

```
import java.util.Scanner;

public class StudentsAvg {
 private int[] noOfSubjects;
 private int totalMarks;

 StudentsAvg(int noOfSubjects) {
  this.noOfSubjects = new int[noOfSubjects];
  totalMarks = 0;
 }
 void setDefails(int index, int marks) {
  this.noOfSubjects[index] = marks;
  totalMarks = (totalMarks + marks);
 }
 void display(int index) {
  for(int i=0; i<noOfSubjects.length; i++) {
   System.out.println(i+1 + ": Marks: " +
    noOfSubjects[i]);
  }
 }
```

```
String determineRank() {
    totalMarks /= noOfSubjects.length;
    if (totalMarks >= 80 && totalMarks <= 100)
        return "Honours";
    } else if (totalMarks >= 60 && totalMarks <= 79){
        return "First Division";
    } else if (totalMarks >= 50 && totalMarks <= 59){
        return "second Division";
    } else if (totalMarks >= 40 && totalMarks <= 39){
        return "Fail";
    }
    return "";
}
public static void main (String[] args){
    Scanner sc = new Scanner (System.in);

    System.out.println("Enter no of students: ");
    int noOfStudents = sc.nextInt();

    StudentsAvg [] st = new StudentsAvg[noof
                                        students];

    for(int i=0; i<noOfStudents; i++) {
        System.out.print("Enter no of subjects: ");
        int noOfSubjects = sc.nextInt();

        st[i] = new StudentsAvg (noOfSubjects);
```

```
        for(int j=0; j<noOfSubjects; j++){
            System.out.print("Enter marks: ");
            int marks = sc.nextInt();

            st[i].setDetails(j, marks);
        }
    }
    for(int i=0; i<noOfStudents; i++){
        System.out.println("\n" + i+1 + "
                    Student's Marks: ");
        st[i].display(i);
        System.out.println("The grant is: " +
            st[i].determineGrant());
    }

    sc.close();
}
}
```