Q.NO.1

a. Discuss TWO (2) advantages and TWO (2) disadvantages of Java.

Ans:- Advantages and Disadvantages of Java:

Advantages:

1. Platform Independence: Java is known for its "Write Once, Run Anywhere" (WORA) capability. Java code is compiled into an intermediate form called bytecode, which can run on any platform with a Java Virtual Machine (JVM). This platform independence makes it easier to develop software that can run on various operating systems without modification.

2. Strong Community and Ecosystem: Java has a vast and active developer community, which means there is a wealth of libraries, frameworks, and tools available for Java developers. This ecosystem accelerates development and simplifies tasks, from building web applications with Spring to creating mobile apps with Android.

Disadvantages:

1. Slower Execution Speed: Java applications are generally slower in execution compared to natively compiled languages like C or C++. This is because Java code is executed by the JVM, which adds an additional layer of abstraction and can lead to some performance overhead.

2. Verbosity: Java code can be quite verbose, requiring a lot of boilerplate code for simple tasks. This verbosity can make development more time-consuming and may lead to more code to maintain and debug.

b. List any FIVE (5) of Java reserved words.

Ans:- Five Java Reserved Words:

1. public: Used to declare a class, method, or variable as accessible from any other class.

2. static: Indicates that a method or variable belongs to the class rather than an instance of the class.

3. void: Specifies that a method does not return a value.

4. if: Used to create conditional statements.

5. class: Declares a class in Java.

c. Given the declaration int x = 9, what is the value of the following expressions?

i. x / 3.0 / 6 * 2

ii. 14 % (x / 5 + 1)

iii. --x + ++x

iv. --x == x++

v. x++ == --x

Ans:-

i. x / 3.0 / 6 * 2 = 1.0

ii. 14 % (x / 5 + 1) = 0

iii. --x + ++x = 17

iv. --x == x++ = true

v. x++ == --x = true

d. List FIVE (5) examples of primitive data types in Java.

Ans:- Here are five examples of primitive data types in Java:

1. int: Represents integer values.

2. double: Represents floating-point numbers with double precision.

3. char: Represents a single character.

4. boolean: Represents a true or false value.

5. byte: Represents a small integer value.

e. Classify each of the following as either a valid or invalid Java identifier:

i) _malaysiakita

ii) hIghValue

iii) ^specialValue

iv) ooxTrick

v) Special$value

vii) %test

Ans:-

i) _malaysiakita - Valid

ii) hIghValue - Valid

iii) ^specialValue - Invalid (Identifiers cannot contain special characters like '^'.)

iv) ooxTrick - Valid

v) Special$value - Invalid (Identifiers cannot contain the '$' symbol.)

vii) %test - Invalid (Identifiers cannot contain the '%' symbol.)

Q.NO.2

Below are the three Java files, which are printed Shape.java, Square.java, and Rectangle.java. Each is in a separate package.

Shape.java

```
package shapes;
public interface Shape
{
 public int area();
}
```

Square.java

```
package square;
public abstract class Square implements Shape
{
 private int width, height;
 public Square (int width, int height)
 {
 this.width = width;
 this.height = height;
 }
 public int area()
 {
 return width * height;
 }
 public toString()
 {
 return "square";
 }
}
```

Rectangle.java

```
package rectangle;
public class Rectangle extends Square implements Shape
{
public String toString()
 {
 return "rectangle";
 }
}
```

a. Add import statements to all Java files.

Ans:-

1. Shape.java (No import statements are needed in this file because it only contains the `Shape` interface):

```java
package shapes;

public interface Shape {
    public int area();
}
```

1. Square.java (Import the `shapes.Shape` interface):

```java
package square;
import shapes.Shape; // Import the Shape interface

public abstract class Square implements Shape {
    private int width, height;

    public Square(int width, int height) {
        this.width = width;
        this.height = height;
    }

    public int area() {
        return width * height;
    }

    public String toString() {
        return "square";
    }
}
```

1. Rectangle.java (Import the `shapes.Shape` interface and the `square.Square` class):

java Copy code

```java
package rectangle;
import shapes.Shape;      // Import the Shape interface
import square.Square;     // Import the Square class

public class Rectangle extends Square implements Shape {
    public String toString() {
        return "rectangle";
    }
}
```

b. Add a field to the Rectangle class to count the number of Rectangle created in the program.

Ans:-

```java
package rectangle;

import square.Square;
import shapes.Shape;

public class Rectangle extends Square implements Shape {
    private static int rectangleCount = 0;

    public Rectangle(int width, int height) {
        super(width, height);
        rectangleCount++;
    }

    public String toString() {
        return "rectangle";
    }

    public static int getRectangleCount() {
        return rectangleCount;
    }
}
```

c. Implement the rectangle constructor in the Rectangle.java.

Ans:-

```java
package rectangle;

import square.Square;

public class Rectangle extends Square implements Shape {

    public Rectangle(int width, int height) {
        super(width, height);
    }

    public String toString() {
        return "rectangle";
    }
}
```

d. Referencing your completed classes, write the output of the following code segments:

i. Rectangle r = new Rectangle(3);

System.out.println(r.toString());

Ans:- rectangle

ii. Square s= new Rectangle(3);

System.out.println(s.toString());

Ans:- rectangle

iii. Shape p = new square(3);

System.out.println(p.toString());

Ans:- square

iv. Rectangle = new Rectangle(3);

System.out.println(r.area());

Ans:- 12

v. Square s= new square(4,3);

System.out.println(s.area());

Ans:- 12

e. Based on the code above, do you find any logical error? State your answer.

Ans:- Yes, there is a logical error in the provided code. The error is in the `Square.java` file, specifically in the `toString()` method implementation. The `toString()` method is missing its return type declaration, which should be `String`.

Q.NO. 3

a. Fix the errors in the following class definitions:

```
class Staff
{
 private String name;
 private int id;
 public Staff (String name, int id)
{
 this.name = name;
 this.id = id;
}
 public String getName ( ) {
 return name;
}
 public int getId ( ) {
 return id;
}
}
class Manager extends Staff
{
 private String department;
 public Manager (String name, int id, String department)
{
 this.name = name;
 this.id = id;
 this.department = department;
```

}

public void getDepartment ( )

{

return department;

}

}

Ans:-

```java
class Staff {
    private String name;
    private int id;

    public Staff(String name, int id) {
        this.name = name;
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public int getId() {
        return id;
    }
}

class Manager extends Staff {
    private String department;

    public Manager(String name, int id, String department) {
        super(name, id);
        this.department = department;
    }

    public String getDepartment() {
        return department;
    }
}
```

b. Correct the following code:

```
public interface Shapes
{
 public void printMessage (String s)
 {
 System.out.println ("This Shape is " + s );
 }
 public double area ( ) ; // to calculate the area of the shape
}
```

Ans:-

```
public interface Shapes
{
   void printMessage(String s);

   double area();
}
```

c. Design a class Circle that implements the corrected interface of Question (b).

Ans:-

```java
public class Circle implements Shapes
{
    private double radius;

    public Circle(double radius)
    {
        this.radius = radius;
    }

    @Override
    public void printMessage(String s)
    {
        System.out.println("This Shape is " + s);
    }

    @Override
    public double area()
    {
        return Math.PI * radius * radius;
    }

    public static void main(String[] args)
    {
        Circle circle = new Circle(5.0);
        circle.printMessage("Circle");
        System.out.println("Area of the circle: " + circle.area());
    }
}
```

d. Why do Java languages which support inheritance also support Polymorphism? Explain your answer.

Ans:-

Java and many other object-oriented programming languages that support inheritance also support polymorphism because polymorphism is a fundamental concept in object-oriented programming that complements and enhances the capabilities of inheritance. Let's explore why these two concepts are closely related:

1. Abstraction and Encapsulation: Object-oriented programming (OOP) encourages the use of abstraction and encapsulation to model real-world entities and their behaviors. Inheritance allows you to create a new class by inheriting the properties and behaviors of an existing class (base or parent class). Polymorphism extends this concept by enabling you to work with objects of different classes in a uniform way, as long as they share a common interface or base class.

2. Code Reusability: Inheritance promotes code reusability by allowing you to reuse and extend existing classes. When you inherit from a base class, you inherit its methods and properties, reducing the need to rewrite code. Polymorphism allows you to use these inherited methods in a flexible and adaptable manner.

3. Flexibility and Extensibility: Polymorphism allows you to write code that works with objects at a more abstract level. You can write functions or methods that accept parameters of a base class type, and they can work with objects of derived classes. This flexibility makes your code more extensible because you can add new classes without having to modify existing code.

4. Method Overriding: Polymorphism in Java often involves method overriding, where a subclass provides its implementation of a method that is already defined in the superclass. This allows each subclass to customize the behavior of inherited methods while still adhering to a common interface defined by the superclass.

5. Dynamic Binding: Java uses dynamic binding to determine the method to execute at runtime based on the actual type of the object, not just its declared type. This dynamic dispatch is a crucial aspect of polymorphism, allowing you to call the appropriate method for a specific object without knowing its exact class at compile time.

Q.NO.4

a. Explain each of the following:

- Abstraction
- The state of an object
- The method signature
- Modularization

Ans:-

1. Abstraction: Abstraction is the process of simplifying complex systems by focusing on essential features while hiding unnecessary details. It involves creating a conceptual model that represents the key aspects of an object or system without getting into the specifics.

2. The state of an object: The state of an object refers to the collection of attributes or properties that define its current condition or characteristics at a particular point in time. It encapsulates the data and values associated with the object.

3. The method signature: The method signature is a unique identifier for a function or method in a programming language. It includes the method's name and its parameter list (the number, order, and types of parameters), but it does not include the method's body or implementation details.

4. Modularization: Modularization is the process of breaking down a complex system or program into smaller, manageable modules or components. Each module typically has a specific function or responsibility, making it easier to develop, maintain, and understand the overall system.

b. Consider the following class definition that has comments. Add method definitions in the blank areas denoted by …….. for each method as the comment that precedes it indicates.

```
/**
* The Student class represents a student in a student
administration system.
* It holds the student details.
*/
public class Student
{
private String name; // the student's full name
 private String id; // the student ID
 private int credits; // the amount of credits for study taken so far
// Create a new student with a given name and ID number.
public Student (String fullName, String studentID)
{
 name = fullName;
 id = studentID;
 credits = 0;
}
// Return the full name of this student.
i)……..
// Set a new name for this student.
ii)……..
// Return the student ID of this student.
iii)……..
// Add some credit points to the student's accumulated credits.
v)……..
/* Check the number of credit points this student has
accumulated.
* If it is less than 132, then print “Not Graduated”, otherwise print
```

“Graduated”.
*/

vi)……..

// Print the student's name and ID number to the output terminal.

vii)……..

Ans:-

```java
/**
 * The Student class represents a student in a student administration system.
 * It holds the student details.
 */
public class Student {
    private String name;    // the student's full name
    private String id;      // the student ID
    private int credits;    // the amount of credits for study taken so far

    // Create a new student with a given name and ID number.
    public Student(String fullName, String studentID) {
        name = fullName;
        id = studentID;
        credits = 0;
    }

    // Return the full name of this student.
    public String getName() {
        return name;
    }

    // Set a new name for this student.
    public void setName(String newName) {
        name = newName;
    }
```

```java
    // Return the student ID of this student.
    public String getStudentID() {
        return id;
    }

    // Add some credit points to the student's accumulated credits.
    public void addCredits(int additionalCredits) {
        credits += additionalCredits;
    }

    /* Check the number of credit points this student has accumulated.
     * If it is less than 132, then print "Not Graduated", otherwise print "Graduated". */
    public void checkGraduationStatus() {
        if (credits < 132) {
            System.out.println("Not Graduated");
        } else {
            System.out.println("Graduated");
        }
    }

    // Print the student's name and ID number to the output terminal.
    public void printStudentInfo() {
        System.out.println("Student Name: " + name);
        System.out.println("Student ID: " + id);
    }
}
```

c. Design a class called Meeting to represent meetings in a diary.

The Meeting class has the following fields:

- time of the meeting represented as string in hours and minutes,

- location of the meeting (such as “room 1234”),

-subject to represent the meeting’s subject (such as “Management’s meeting”).

Time, location and subject are stored as strings.

The class should include a constructor and the following methods:

- setTime: to set the time.

- setLocation: to set the location.

- setSubject: to set the subject.

- getSubject: to return the subject of the meeting.

- printDetails: to print all information of a meeting in the following form:

Meeting in room 1234 at 12:50pm; Subject: Management’s Meeting

Ans:-

```
public class Meeting {
    // Fields
    private String time;
    private String location;
    private String subject;

    // Constructor
    public Meeting(String time, String location, String subject) {
        this.time = time;
        this.location = location;
        this.subject = subject;
    }

    // Setter methods
    public void setTime(String time) {
        this.time = time;
    }

    public void setLocation(String location) {
        this.location = location;
    }

    public void setSubject(String subject) {
        this.subject = subject;
    }
```

```java
    // Getter method
    public String getSubject() {
        return subject;
    }

    // Method to print meeting details
    public void printDetails() {
        System.out.println("Meeting in " + location + " at " + formatTime(time) + "; Subject: " + subject);
    }

    // Helper method to format time (assuming input is in HH:mm format)
    private String formatTime(String time) {
        String[] parts = time.split(":");
        int hours = Integer.parseInt(parts[0]);
        int minutes = Integer.parseInt(parts[1]);

        String period = (hours >= 12) ? "pm" : "am";
        if (hours > 12) {
            hours -= 12;
        } else if (hours == 0) {
            hours = 12;
        }

        return String.format("%d:%02d%s", hours, minutes, period);
    }

    public static void main(String[] args) {
        Meeting meeting = new Meeting("12:50", "room 1234", "Management's meeting");
        meeting.printDetails();
    }
}
```