JAVA Programming Assignment 02

Name : Kedarnath Prasad Chavan

PRN : 21610047

Batch : S-4

1. Explain difference between method overloading and method overriding.

Method overloading and method overriding are two important concepts in object-oriented programming languages such as Java. Although they share some similarities, they have distinct differences in terms of their purpose, implementation, and behavior.

Method Overloading:
Method overloading is a feature that allows multiple methods to have the same name in a class. However, the parameters of the methods must be different. The main purpose of method overloading is to provide different implementations of the same method for different argument types or numbers. When a method is called, the Java Virtual Machine (JVM) decides which version of the method to execute based on the type and number of arguments passed. Method overloading is also called compile-time polymorphism or static polymorphism because the method to be executed is determined at compile-time.

Method Overriding:
Method overriding is a feature that allows a subclass to provide its own implementation of a method that is already defined in its superclass. In other words, a subclass can replace the behavior of an inherited method with its own implementation. To override a method, the method signature (name, return type, and parameter types) of the subclass method must match exactly with that of the superclass method. The main purpose of method overriding is to achieve runtime polymorphism or dynamic polymorphism, where the actual method to be executed is determined at runtime based on the object type.

Differences:

Method overloading occurs within the same class, whereas method overriding occurs between a superclass and a subclass.
In method overloading, the method name is the same, but the parameters are different. In method overriding, the method name, return type, and parameter types must be the same.
Method overloading is resolved at compile-time, whereas method overriding is resolved at runtime.

Method overloading is used to provide different implementations of the same method for different argument types or numbers, whereas method overriding is used to provide a new implementation of a method that is already defined in the superclass.

2. Implement all string functions in java.

In Java, the String class provides many built-in functions to manipulate strings. Here are the implementations of some of the most commonly used string functions:

length(): Returns the length of a string.

```
public static int stringLength(String str) {
    return str.length();
}
```

charAt(): Returns the character at a specified index in a string.

```
public static char charAt(String str, int index) {
    return str.charAt(index);
}
```

concat(): Concatenates two strings.

```
public static String concat(String str1, String str2) {
    return str1.concat(str2);
}
```

indexOf(): Returns the index of a specified character or substring within a string. If the character or substring is not found, it returns -1.

```
public static int indexOf(String str, String substr) {
    return str.indexOf(substr);
}
```

substring(): Returns a substring of a string starting at a specified index and ending at another specified index or at the end of the string.

```
public static String substring(String str, int startIndex, int endIndex) {
    return str.substring(startIndex, endIndex);
}
```

toLowerCase(): Converts a string to lowercase.

```
public static String toLowerCase(String str) {
    return str.toLowerCase();
}
```

toUpperCase(): Converts a string to uppercase.public static String toUpperCase(String str) {
    return str.toUpperCase();
}

trim(): Removes whitespace from both ends of a string.
public static String trim(String str) {
    return str.trim();
}

3. Implement all stringbuffer functions in java.

```
public class MyStringBuffer {
    private char[] value;
    private int count;

    public MyStringBuffer() {
        value = new char[16];
        count = 0;
    }

    public MyStringBuffer(int capacity) {
        value = new char[capacity];
        count = 0;
    }

    public MyStringBuffer(String str) {
        value = new char[str.length() + 16];
        count = str.length();
        System.arraycopy(str.toCharArray(), 0, value, 0, count);
    }

    public int capacity() {
        return value.length;
    }

    public int length() {
        return count;
    }

    public synchronized MyStringBuffer append(Object obj) {
        return append(String.valueOf(obj));
    }

    public synchronized MyStringBuffer append(String str) {
        int len = str.length();
        if (len == 0) {
```

```
            return this;
        }
        int newCount = count + len;
        if (newCount > value.length) {
            expandCapacity(newCount);
        }
        str.getChars(0, len, value, count);
        count = newCount;
        return this;
    }

    public synchronized MyStringBuffer append(char[] str, int offset, int len) {
        if (offset < 0 || offset + len > str.length) {
            throw new IndexOutOfBoundsException();
        }
        if (len == 0) {
            return this;
        }
        int newCount = count + len;
        if (newCount > value.length) {
            expandCapacity(newCount);
        }
        System.arraycopy(str, offset, value, count, len);
        count = newCount;
        return this;
    }

    public synchronized MyStringBuffer append(CharSequence s) {
        if (s == null) {
            s = "null";
        }
        return append(s.toString());
    }

    public synchronized MyStringBuffer append(CharSequence s, int start, int end)
{
        if (s == null) {
```

```
        s = "null";
    }
    return append(s.subSequence(start, end).toString());
}

public synchronized MyStringBuffer append(char c) {
    int newCount = count + 1;
    if (newCount > value.length) {
        expandCapacity(newCount);
    }
    value[count++] = c;
    return this;
}

public synchronized MyStringBuffer delete(int start, int end) {
    if (start < 0 || end > count || start > end) {
        throw new StringIndexOutOfBoundsException();
    }
    int len = end - start;
    if (len > 0) {
        System.arraycopy(value, end, value, start, count - end);
        count -= len;
    }
    return this;
}

public synchronized MyStringBuffer deleteCharAt(int index) {
    if (index < 0 || index >= count) {
        throw new StringIndexOutOfBoundsException();
    }
    System.arraycopy(value, index + 1, value, index, count - index - 1);
    count--;
    return this;
}

public synchronized MyStringBuffer replace(int start, int end, String str) {
    if (start < 0 || end > count || start > end) {
```

```
    throw new StringIndexOutOfBoundsException();
}
int len = str.length();
int newCount = count - (end - start) + len;
if (newCount > value.length) {
    expandCapacity(newCount);
}
System.arraycopy(value, end, value, start + len, count - end);
str.getChars
```

4. Explain with example declaration of string using string literal and new keyword.

In Java, there are two ways to declare a String object: using a string literal or using the new keyword.

String Literal:
A string literal is a sequence of characters enclosed in double quotes. When a string literal is assigned to a String variable, Java automatically creates a String object that contains the same sequence of characters. For example:

```
String str1 = "Hello";  // Using String Literal
```

In this example, the string literal "Hello" is assigned to the variable str1. Java automatically creates a new String object that contains the characters "Hello".

new Keyword:
The new keyword is used to create a new instance of a class. To create a new String object using the new keyword, we must use the String class constructor. For example:

```
String str2 = new String("Hello");  // Using new keyword
```

In this example, the String constructor is called with the argument "Hello". This creates a new String object that contains the characters "Hello", which is assigned to the variable str2.

Note that when a String object is created using the new keyword, it always creates a new object in memory, even if another String object with the same contents already exists in memory. However, when a String object is created using a string literal, Java checks if an object with the same contents already exists in memory. If it does, the same object is reused. This is called string interning.

5. Create a class named “Shape” with a method to print “this is shape.” Then create two other classed named ‘Rectangle’, ‘Circle’ inheiting the shape class, both having a method to print “this is rectangular shape” and this is circular shape” respectively. Create a subclass “square” of ‘rectangle’ having a method to print “square is a rectangle’. Now call the method of ‘shape’ and ‘rectangle’ class by the object of ‘square class.

Here's the implementation of the classes as described in the problem statement:

```
class Shape {
    public void printShape() {
        System.out.println("This is shape");
    }
}

class Rectangle extends Shape {
    public void printShape() {
        System.out.println("This is rectangular shape");
    }
}

class Circle extends Shape {
    public void printShape() {
        System.out.println("This is circular shape");
    }
}

class Square extends Rectangle {
    public void printSquare() {
        System.out.println("Square is a rectangle");
    }
}

public class Main {
    public static void main(String[] args) {
        Square square = new Square();
```

```
        square.printShape();    // This will call the printShape() method of Rectangle
class
        square.printSquare();   // This will call the printSquare() method of Square
class
    }
}
```

6. Create game characters using the concept of inheritance. Suppose, in your
game, you want three characters - a maths teacher, a footballer and a businessman. Since, all of the characters are persons, they can walk and talk. However, they also have some special skills. A maths teacher can teach maths, a footballer can play football and a businessman can run a business. You can individually create three classes who can walk, talk and perform their special skill. In each of the classes, you would be copying the same code for walk and talk for each character. If you want to add a new feature - eat, you need to
implement the same code for each character. This can easily become error prone (when copying) and duplicate codes. It'd be a lot easier if we had a
Person class with basic features like talk, walk, eat, sleep, and add special skills to those features as per our characters. This is done using inheritance. Using inheritance, now you don't implement the same code for walk and
talk for each class. You just need to inherit them. So, for Maths teacher (derived class), you inherit all features of a Person (base class) and add a new feature TeachMaths. Likewise, for a footballer, you inherit all the features of a Person and add a new feature PlayFootball and so on.

Here's an example implementation of the game characters using inheritance:

```
// The base class for all persons
class Person {
    public void walk() {
        System.out.println("I am walking");
    }

    public void talk() {
        System.out.println("I am talking");
    }

    public void eat() {
        System.out.println("I am eating");
```

```
    }

    public void sleep() {
        System.out.println("I am sleeping");
    }
}

// Derived class for a Maths teacher
class MathsTeacher extends Person {
    public void teachMaths() {
        System.out.println("I am teaching maths");
    }
}

// Derived class for a Footballer
class Footballer extends Person {
    public void playFootball() {
        System.out.println("I am playing football");
    }
}

// Derived class for a Businessman
class Businessman extends Person {
    public void runBusiness() {
        System.out.println("I am running a business");
    }
}
```

In the above implementation, we have a base class Person which has common features like walk, talk, eat, and sleep. Then, we have three derived classes MathsTeacher, Footballer, and Businessman which inherit the common features from Person class and add their own unique features. MathsTeacher has a teachMaths method, Footballer has a playFootball method, and Businessman has a runBusiness method.

Now, to create an object of any of these game characters, we simply create an object of the respective class and call its methods:

```
public class Main {
    public static void main(String[] args) {
        MathsTeacher mt = new MathsTeacher();
        mt.walk();
        mt.talk();
        mt.teachMaths();

        Footballer f = new Footballer();
        f.walk();
        f.talk();
        f.playFootball();

        Businessman b = new Businessman();
        b.walk();
        b.talk();
        b.runBusiness();
    }
}
```

7. WAP to manage the employee allowance from a specific department by creating class structure as follow,

InheritanceEx2
|
|
InheritanceEx2Main.java
|
|- dept | Deparment.java
|
|
|
|- emp | Employee.java extends Department
|
|
|
|- allowance | Allowance.java extends Employee
|
| [Multilevel Inheritance]

Here's an example implementation of the given class structure:

```
// Department class
class Department {
    protected String name;

    public Department(String name) {
        this.name = name;
    }

    public void display() {
        System.out.println("Department Name: " + name);
    }
}

// Employee class extending Department
class Employee extends Department {
    protected String empId;
```

```
    protected String name;

    public Employee(String deptName, String empId, String name) {
        super(deptName);
        this.empId = empId;
        this.name = name;
    }

    public void display() {
        super.display();
        System.out.println("Employee Id: " + empId);
        System.out.println("Employee Name: " + name);
    }
}

// Allowance class extending Employee
class Allowance extends Employee {
    private double amount;

    public Allowance(String deptName, String empId, String name, double
amount) {
        super(deptName, empId, name);
        this.amount = amount;
    }

    public void display() {
        super.display();
        System.out.println("Allowance Amount: " + amount);
    }
}

// Main class
public class InheritanceEx2Main {
    public static void main(String[] args) {
        // Creating an Allowance object
        Allowance allowance = new Allowance("IT", "E001", "John", 5000.0);
```

```
        // Displaying the Allowance details
        allowance.display();
    }
}
```

In the above implementation, we have three classes Department, Employee, and Allowance which use multilevel inheritance. Employee extends Department and Allowance extends Employee.

Department has a constructor to initialize the department name and a display method to display the department name.

Employee has a constructor to initialize the department name, employee id, and employee name. It also overrides the display method of Department to display the employee id and employee name along with the department name.

Allowance has a constructor to initialize the department name, employee id, employee name, and allowance amount. It overrides the display method of Employee to display the allowance amount along with the employee details.

In the InheritanceEx2Main class, we create an object of Allowance class with some sample values and call its display method to display the allowance details.

8. Write a Java Program to demonstrate StringBuilder class methods.

Sure, here is an example Java program that demonstrates some of the methods of the StringBuilder class:

```
public class StringBuilderDemo {

  public static void main(String[] args) {
    StringBuilder sb = new StringBuilder("Hello, World!");

    // Append methods
    sb.append(" How are you?");
    System.out.println(sb.toString());

    // Insert methods
    sb.insert(13, "doing ");
    System.out.println(sb.toString());

    // Replace method
    sb.replace(0, 5, "Hi");
    System.out.println(sb.toString());

    // Delete method
    sb.delete(0, 3);
    System.out.println(sb.toString());

    // Reverse method
    sb.reverse();
    System.out.println(sb.toString());
  }
}
```

```
class Result {
    public void result() {
        System.out.println("This is the result of the exam.");
    }
}

class UGResult extends Result {
    @Override
    public void result() {
        System.out.println("This is the result of the Undergraduate exam.");
    }
}

class PGResult extends Result {
    @Override
    public void result() {
        System.out.println("This is the result of the Postgraduate exam.");
    }
}

public class MethodOverrideDemo {
    public static void main(String[] args) {
        Result r = new Result();
        r.result();  // prints "This is the result of the exam."

        UGResult ug = new UGResult();
        ug.result();  // prints "This is the result of the Undergraduate exam."

        PGResult pg = new PGResult();
        pg.result();  // prints "This is the result of the Postgraduate exam."
    }
}
```

9. Write a Java Program to demonstrate Method overriding.(create class Result with method result(). Override method result() in UGResult and PGResult class)

```
class Result {
    void result() {
        System.out.println("The result is: Pass");
    }
}

class UGResult extends Result {
    @Override
    void result() {
        System.out.println("The result of UG exam is: Distinction");
    }
}

class PGResult extends Result {
    @Override
    void result() {
        System.out.println("The result of PG exam is: First Class");
    }
}

public class MethodOverrideDemo {
    public static void main(String[] args) {
        Result res1 = new Result();
        res1.result();

        UGResult res2 = new UGResult();
        res2.result();

        PGResult res3 = new PGResult();
        res3.result();
    }
}
```

10. Write a java program to create a class called STUDENT with data members PRN, Name and age. Using inheritance, create a classes called UGSTUDENT and PGSTUDENT having fields as semester, fees and stipend. Enter the data for at least 5 students. Find the semester wise average age for all UG and PG students separately.

```
import java.util.Scanner;

class Student {
    protected String prn;
    protected String name;
    protected int age;

    public Student(String prn, String name, int age) {
        this.prn = prn;
        this.name = name;
        this.age = age;
    }
}

class UGStudent extends Student {
    private int semester;
    private double fees;
    private double stipend;

    public UGStudent(String prn, String name, int age, int semester, double fees,
double stipend) {
        super(prn, name, age);
        this.semester = semester;
        this.fees = fees;
        this.stipend = stipend;
    }

    public int getSemester() {
        return semester;
    }
```

```
    public double getFees() {
        return fees;
    }

    public double getStipend() {
        return stipend;
    }
}

class PGStudent extends Student {
    private int semester;
    private double fees;
    private double stipend;

    public PGStudent(String prn, String name, int age, int semester, double fees,
double stipend) {
        super(prn, name, age);
        this.semester = semester;
        this.fees = fees;
        this.stipend = stipend;
    }

    public int getSemester() {
        return semester;
    }

    public double getFees() {
        return fees;
    }

    public double getStipend() {
        return stipend;
    }
}

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

```
        Scanner sc = new Scanner(System.in);

        UGStudent[] ugStudents = new UGStudent[2];
        PGStudent[] pgStudents = new PGStudent[3];

        for (int i = 0; i < 2; i++) {
            System.out.println("Enter details for UG student " + (i + 1));
            System.out.print("PRN: ");
            String ugPrn = sc.nextLine();
            System.out.print("Name: ");
            String ugName = sc.nextLine();
            System.out.print("Age: ");
            int ugAge = sc.nextInt();
            System.out.print("Semester: ");
            int ugSemester = sc.nextInt();
            System.out.print("Fees: ");
            double ugFees = sc.nextDouble();
            System.out.print("Stipend: ");
            double ugStipend = sc.nextDouble();

            ugStudents[i] = new UGStudent(ugPrn, ugName, ugAge, ugSemester,
ugFees, ugStipend);

            sc.nextLine(); // consume the newline character left by nextDouble()
        }

        for (int i = 0; i < 3; i++) {
            System.out.println("Enter details for PG student " + (i + 1));
            System.out.print("PRN: ");
            String pgPrn = sc.nextLine();
            System.out.print("Name: ");
            String pgName = sc.nextLine();
            System.out.print("Age: ");
            int pgAge = sc.nextInt();
            System.out.print("Semester: ");
            int pgSemester = sc.nextInt();
            System.out.print("Fees: ");
```

```
            double pgFees = sc.nextDouble();
            System.out.print("Stipend: ");
            double pgStipend = sc.nextDouble();

            pgStudents[i] = new PGStudent(pgPrn, pgName, pgAge, pgSemester,
pgFees, pgStipend);

            sc.nextLine(); // consume the newline character left by nextDouble()
        }
```

**11. Implement hybrid inheritance using all access specifiers (public, private,
protected).**

```
public class HybridInheritanceDemo {
    public static void main(String[] args) {
        DerivedClass1 obj1 = new DerivedClass1();
        obj1.display();
        obj1.print();

        DerivedClass2 obj2 = new DerivedClass2();
        obj2.display();
        obj2.print();

        DerivedClass3 obj3 = new DerivedClass3();
        obj3.display();
        obj3.print();
        obj3.show();
    }
}

class BaseClass {
    private int a = 10;
    protected int b = 20;
    public int c = 30;

    public void display() {
        System.out.println("a = " + a);
        System.out.println("b = " + b);
        System.out.println("c = " + c);
    }
}

class DerivedClass1 extends BaseClass {
    private int d = 40;

    public void print() {
```

```
        System.out.println("d = " + d);
        System.out.println("b = " + b);
        System.out.println("c = " + c);
    }
}

class DerivedClass2 extends BaseClass {
    private int e = 50;

    public void print() {
        System.out.println("e = " + e);
        System.out.println("b = " + b);
        System.out.println("c = " + c);
    }
}

class DerivedClass3 extends DerivedClass1 {
    private int f = 60;

    public void show() {
        System.out.println("f = " + f);
        System.out.println("b = " + b);
        System.out.println("c = " + c);
    }
}
```

12. Write a program to implement a class Teacher contains two fields Name and Qualification. Extend the class to Department, it contains Dept. No and Dept. Name. An Interface named as College it contains one field Name of the College. Using the above classes and Interface get the appropriate information and display it.

```
interface College {
    String COLLEGE_NAME = "ABC College";
}

class Teacher {
    String name;
    String qualification;

    Teacher(String name, String qualification) {
        this.name = name;
        this.qualification = qualification;
    }
}

class Department extends Teacher {
    int deptNo;
    String deptName;

    Department(String name, String qualification, int deptNo, String deptName) {
        super(name, qualification);
        this.deptNo = deptNo;
        this.deptName = deptName;
    }

    void display() {
        System.out.println("Name of the teacher: " + name);
        System.out.println("Qualification of the teacher: " + qualification);
        System.out.println("Department Number: " + deptNo);
        System.out.println("Department Name: " + deptName);
        System.out.println("College Name: " + College.COLLEGE_NAME);
    }
```

```
}

public class TeacherDemo {
    public static void main(String[] args) {
        Department dept = new Department("John Doe", "M.Sc.", 101, "Computer
Science");
        dept.display();
    }
}
```

Online Registration For MHT-CET-2023 **Application Form**

GOVERNMENT OF MAHARASHTRA
STATE COMMON ENTRANCE TEST CELL, MAHARASHTRA STATE
8th Floor, New Excelsior Building, A.K.Nayak Marg, Fort, Mumbai-400001. (M.S.)

Application Form for Online Registration For MHT-CET-2023

Application No. : **231191997** Version No : **1**

Personal Details

Candidate's Full Name	**SALUNKE SHITAL SATISH**
Father's Name	**SATISH**
Mother's Name	**RAJASHREE**
Gender	**Female**
Date of Birth	**27/07/2005**
Religion	**Hindu**
Region	**Rural**
Mother Tongue	**Marathi**
Annual Family Income	**50,001 - 1,00,000**
Nationality	**Indian**
MHT-CET-2023 Application Fee Paid (₹)	**1200/-**

Permanent Address

Address Line 1	**At-palwan, Post-sapatane(T)**		
Address Line 2	**Tal-Madha**		
Address Line 3	**Dist-Solapur**		
State	**Maharashtra**	District	**Solapur**
Taluka	**Madha**	Village	**Palwan**
PIN Code	**413210**		

Address for Correspondence

Address Line 1	**At-palwan, Post-sapatane(T)**		
Address Line 2	**Tal-Madha**		
Address Line 3	**Dist-Solapur**		
State	**Maharashtra**	District	**Solapur**
Taluka	**Madha**	Village	**Palwan**
PIN Code	**413210**		
Telephone No	**-**		

Domicile and Category Details

Are you Domiciled in the State of Maharashtra?	**Yes**
Category	**Open**
Wish to Apply for EWS (Economically Weaker Section) Seats ?	**Yes**
EWS Certificate Status	**Available**
Are you Person With Disability ?	**No**

Qualification Details

SSC / Equivalent Details

Have you passed SSC or Equivalent Exam from India?	**Yes**
SSC/Equivalent Board	**Maharashtra State Board of Secondary and Higher Secondary Education, Pune**
SSC/Equivalent Passing Year	**2021**
SSC/Equivalent Percentage	**94.20**
State From Which you Passed SSC/Equivalent	**Maharashtra**
District From Which you Passed SSC/Equivalent	**Solapur**
Taluka From Which you Passed SSC/Equivalent	**Madha**
School Name of SSC/Equivalent	**NEW ENGLISH SCHOOL, GHOTI**

HSC / Equivalent Details

Are you Appearing /Appeared 12th (HSC) exam in 2023	**Yes**
HSC/Equivalent Board	**Maharashtra State Board of Secondary and Higher Secondary Education, Pune**
HSC Passing Year	**NA**
State Where your 12th(HSC)College is Situated	**Maharashtra**
District Where your 12th(HSC)College is Situated	**Solapur**
Taluka Where your 12th(HSC)College is Situated	**Madha**
HSC School/College Name	**NEW ENGLISH SCHOOL AND JR. COLLEGE, GHOTI**

MHT-CET-2023 Examination Details

Subject Group for MHT-CET-2023	**Both(PCM and PCB) (Physics, Chemistry, Mathematics and Biology)**
Language for the Question Paper	**English**
State for MHT-CET-2023 Examination Center	**Maharashtra**
Exam Center at Preference Number 1	**Solapur**
Exam Center at Preference Number 2	**Pune**
Exam Center at Preference Number 3	**Ahmednagar**
Exam Center at Preference Number 4	**Satara**

Document Uploaded

Sr. No.	Document Name
1.	Aadhaar Card

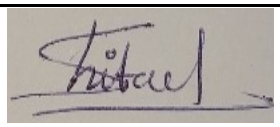
Note :

• You are required to Upload Eligibility Certificate for Economically Weaker Section at the time of Centralised Admission Process(CAP 2023).

Declaration

I agree to the following conditions

I have gone through the information brochure and understand the eligibility and qualifying criteria.
The information filled by me is true to the best of my knowledge.
I understand I will be liable for penal action for submitting incorrect information.

Date : **16/03/2023**	
Last Modified By : **231191997, 152.57.118.102:55126**	Signature of Applicant
Last Modified On : **16/03/2023 3:10:07 PM**	(**SALUNKE SHITAL SATISH**)
Printed On : **16/03/2023 3:10:28 PM**	