JAVA ASSIGNMENT

Name :- Manish kumar Singh

Roll No:- DS7B-2208

Department: Big Data Analytics

Course: M. Tech

1) Write a program to display complete data of an object without calling explicid method or any dot operator.

Ans :-

```
public class Student {
        int roll;
        String name;
        int age;
        Student(){
            roll = -1;
            name = "unknown";
            age = -1;
        }
        Student(int r, String n, int a) {
            roll = r;
            name = n;
            age = a;
        }
        public String toString() {
            return "Roll:" + roll + ", Name: " + name + ", Age: " + age;
        }
        public static void main (String args[]){
            Student st = new Student();
            Student st1 = new Student (101, "Smith", 32);
            System.out.println(st);
            System.out.println(st1);
        }
}
```

2) Write a program to display working of break and continue with labeled break and labeled continue.

Ans :-

```
public class BreakAndContinueExample {
    public static void main(String[] args) {
        for(int i=0; i<5; i++) {
            if(i==3) {
                break;
            }
            System.out.println("i= " +i);
        }
        System.out.println();

        for(int i=0; i<5; i++) {
            if(i==3) {
                continue;
            }
            System.out.println("i = " +i);
        }
        System.out.println();

        outer:
        for(int i=0; i<3; i++) {
            for(int j=0; j<3; j++) {
                if(i==1 && j==1) {
                    break outer;
                }
                System.out.println("i = " +i+ ", j" + j);
            }
        }
        System.out.println();

        outer:
        for(int i=0; i<3; i++) {
            for(int j=0; j<3; j++) {
                if(i==1 && j==1) {
                    continue outer;
                }
                System.out.println("i= " + i + ",j= " +j);
            }
        }
    }
}
```

3) Write a program to show static data members accessing in non-static context

Ans:-

```
public class StaticDataExample {
    private static int count = 0;
    public StaticDataExample(){
        count++;
    }
    public void displayCount(){
        // Accessing static data member in non static context
        System.out.println("Number of objects created: " +
            staticDataExample.count);
    }
    public static void main(String[] args) {
        StaticDataExample obj1 = new StaticDataExample();
        StaticDataExample obj2 = new StaticDataExample();
        obj1.displayCount();
    }
}
```

In this program, we have a static data member 'count' that keeps track of the number of objects of the 'StaticDataExample' class that have been created. In the constructor, we increment 'count' each time a new object is created.

We also have a non-static method 'displayCount' that accesses the static data member 'count' and displays the total number of objects that have been created.

In the 'main' method, we create two objects of the 'StaticDataExample' class. Then, we call the non-static method 'displayCount' on 'obj1' to display the total number of objects that have been created.

4) Write a program to show the final keywords with its uses?

Ans:-
The 'final' keyword in java is used to create constants, indicate that the variable or method should not be overridden, or not to prevent a class from being subclassed.

```
public class FinalExample {
    public static final double PI = 3.14159;

    public final void display() {
        System.out.println("This method cannot be overriden.");
    }
    public static void main(String[] args) {
        final int NUM=10;
        System.out.println("Value of PI: " + FinalExample.PI);
        FinalExample obj = new FinalExample();
        obj.display();
    }
}
```

In this program, we have a 'final' constant 'PI' that represent the value of pi. We also have a 'final' method 'display' that cannot be overriden by any sub classes.

In the 'main' method, we have a "final" variable 'NUM' that is initialized to 10 and cannot be changed later. We then display the value of 'PI' and call the 'display' method on an instance of the 'FinalExample' class.

If we attempt to change the value of 'NUM', it will result in a compilation error because 'NUM' is a final variable and cannot be modified.

5) Write a program to show the syntax of inheritance in java?

Ans:-

In Java there are four types of inheritance : Single inheritance, multilevel inheritance, hierarchical inheritance, and multible inheritance through interfaces.

```
class Animal {
    public void eat(){
        System.out.println("Animal is eating...");
    }
}
//Single inheritance
class Dog extends Animal {
    public void bark(){
        System.out.println("Dog is barking...")
    }
}
//Multilevel inheritance
class BabyDog extends Dog {
    public void weep(){
        System.out.println("Baby dog is weeping...");
    }
}

//Hierarchical inheritance
class Cat extends Animal {
    public void meow(){
        System.out.println("Cat is meowing...");
    }
}
//Multible inheritance through interfaces
interface Mammal {
    public void run();
}
```

```
interface Reptile {
    public void crawl();
}
class Crocodile implements Mammal, Reptile {
    public void run() {
        System.out.println("Crocodile is running...");
    }
    public void crawl() {
        System.out.println("Crocodile is crawling...");
    }
}
class InheritanceExample {
    public static void main(String[] args) {
        // Single inheritance
        Dog d = new Dog();
        d.eat();
        d.bark();
        // Multilevel inheritance
        BabyDog bd = new BabyDog();
        bd.eat();
        bd.bark();
        bd.weep();
        // Hierarchical inheritance
        Cat c = new Cat();
        c.eat();
        c.meow();
        // Multiple inheritance through interfaces
        Crocodile croc = new Crocodile();
        croc.run();
        croc.crawl();
    }
}
```

In this program, we have a base class 'Animal' that has a 'eat' method. We then have a 'Dog' class that it inherits from 'Animal' using single inheritance and has a 'bark' method. The 'BabyDog' class inherits from 'Dog' using multilevel inheritance and has a 'weep' method. The 'Cat' class also inherit from 'Animal' using hierarchical inheritance and has a 'meow' method.

We then have two two interfaces for inheritance 'Mammal' and 'Reptile' that have a 'run' and 'crawl' method, respectively. The 'Crocodile' class implements both interfaces using multiple inheritance through interfaces.

In 'main' method, we create object of each class and call their respective methods. This demonstrate how each type of inheritance works in java.

6) Write a program to show the execution of constructor in multilevel inheritance.

Ans:- In multilevel inheritance, a subclass inherits from a super class, which in turn can also inherit from another superclass. In this case, the constructors of all the classes involved are executed in a specific order.

```
class Animal {
    public Animal(){
        System.out.println("Animal constructor called");
    }
}
class Mammal extends Animal {
    public Mammal(){
        System.out.println("Mammal constructor called");
    }
}
class Dog extends Mammal {
    public Dog(){
        System.out.println("Dog constructor called");
    }
}
public class InheritanceExample {
    public static void main(String[] args){
        Dog dog = new Dog();
    }
}
```

Here we have Dog class inherits from Mammal, which in turn inherits from 'Animal'. Each class has its own constructor which prints a message when called.

When we create a 'Dog' object in the 'main' method, the constructors are executed in the following order:

1. The 'Animal' constructor is executed first, since it is the superclass of 'Mammal'.
2. The 'Mammal' constructor is executed next, since it is the superclass of 'Dog'
3. Finally, the 'Dog' constructor is executed.

Q. What is the use of interface. How it is helpful for multiple inheritance.

Ans:-
In Java, interfaces are used to implement multiple inheritance. Multiple inheritance refers to the ability to inherit from multiple classes, which is not allowed in Java. However, interfaces provide a way to achieve similar functionality.

An interface in java is a collection of ~~object~~ abstract methods that can be implemented by a class. When a class implements an interface, it provides an implementation for all methods defined in the interface. This allows a class to define its behaviour for a specific set of methods, while also being able to inherit behaviour from other classes.

```
interface A{
        void methodA();
        }
interface B{
        void methodB();
        }
class MyClass implements A,B{
        public void methodA(){
            System.out.println("Method A implementation");
        }
        public void methodB(){
            System.out.println("Method B implementation");
            }
        }
public class InterfaceExample {
    public static void main(String[], args) {
                MyClass obj = new MyClass();
                obj.methodA();
                obj.methodB();
            }
        }
```

Here, we have two interfaces 'A' and 'B' each with single method. We also have a class 'MyClass' that implements both ~~method~~ interfaces and provide the implementation of each methods. In the 'main' method, we create an object of 'MyClass' and called both methods in it.

8) How can you create a package, write steps, Also write a program to access public, protected, private and default data members in various areas.

Ans:-
In Java, a package is a grouping of related classes, interfaces, and other resources. It provides a way to organize code and make it more manageable. A package can contain classes with different access modifiers, such as public, protected, private and default (no modifier).

To create a package, we need to follow these steps:

1. Create a directory with the name of your package (for example, 'package com.example.mypackage')
2. Save your Java files in this directory with the package declaration at the top.
3. Compile your java files using the '-d' option to specify the directory where the compiled class files should be saved.

Let's say we have a Java class 'MyClass' with different access modifier

```
package com.example.mypackage;
public class MyClass{
        public int publicVar = 1;
        protected int protectedVar = 2;
        private int privateVar = 3;
        int defaultVar = 4;
    }
```

Here, we have a public variable 'publicVar', a protected variable 'protectedVar', a private variable 'privateVar', and a default variable 'defaultVar'. These variables have different access modifiers, which control where they can be accessed from.

To access these variables in different areas, we can create another Java class in the same package or a different package.

1. Access public variable from another class in the same package

```
package com.example.mypackage;
public class AnotherClass{
    public static void main(String[] args){
        MyClass obj = new MyClass();
        System.out.println(obj.publicVar);
    }
}
```

→ Here, we create a new class "AnotherClass" in the same package 'com.example.mypackage' and create an object of 'MyClass'. We can access public variable 'publicVar' directly.

2. Access protected variable from a subclass in the same package

```
package com.example.mypackage;
public class SubClass extends MyClass{
    public static void main(String[] args){
        SubClass obj = new SubClass();
        System.out.println(obj.protectedVar);
    }
}
```

Here, we create a 'subclass' of 'MyClass' in the same package 'com.example.mypackage'. We can access the protected variable 'protectedVar' from the subclass.

3. Access private variable within the same class:

```
package com.exemple.mypackage;
public class MyClass {
    private int privateVar = 3;
    public void myMethod() {
        System.out.println(privateVar);
    }
}
```

Here we create a new method 'myMethod' in 'MyClass' and access the private variable 'privateVar' directly within the same class.

4. Access default variable from another class in a different package.

```
package com.example.mypackage;
class AnotherClass {
    public static void main(String[] args) {
        MyClass obj = new MyClass();
        System.out.println(obj.defaultVar);
    }
}
```

Here we create a new class 'AnotherClass' in a differen package 'com.example.anotherpackage' and create an object of 'MyClass'. We can access the default variable 'defaultVar' directly because two classes are in the same package.

Category	private	public	protected	default
Same Class	✓	✓	✓	✓
Same Package Non-Subclass	X	✓	✓	✓
Same Package Sub-class	X	✓	✓	✓
Other Package Non-Subclass	X	✓	X	X
Other Package Sub-class	X	✓	✓	X

9) Write a user defined string function encrypt and decrypt for encryption and decryption according to passed value

Ans:-

```
import java.util.Scanner;
public class Encryption {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.println("Enter a message to encrypt: ");
        String message = input.nextLine();
        System.out.println("Enter an encryption key: ");
        int key = input.nextInt();
        String encryptedMessage = encrypt(message, key);
        System.out.println("Encrypted message: " + encryptedMessage);
        String decryptedMessage = decrypt(encryptedMessage, key);
        System.out.println("Decrypted message: " + decryptedMessage);
    }
public static String encrypt(String message, int key) {
        StringBuilder result = new StringBuilder();
        for(int i=0; i<message.length(); i++) {
            char c = (char)(message.charAt(i) + key);
            result.append(c);
        }
        return result.toString();
    }
public static String decrypt(String message, int key) {
        StringBuilder result = new StringBuilder();
        for(int i=0; i<message.length(); i++) {
            char c = (char)(message.charAt(i) - key);
            result.append(c);
        }
        return result.toString();
    }
}
```

10) Write a user defined function to count total Number of occurance of given character of the string. Improve the function for string also.

Ans :-

```
public class CountOccurrence {
    public static void main(String[] args) {
        String str = "The quick brown fox jumps over the lazy dog";
        char ch = 'o';
        String subStr = "the";
        int count = countOccurrences(str, ch);
        System.out.println("Occurrence of character 'o': " + count);
        count = countOccurrences(str, subStr);
        System.out.println("Occurrence of string 'the': " + count);
    }
    public static int countOccurrences(String str, char ch) {
        int count = 0;
        for(int i=0; i< str.length(); i++) {
            if(str.charAt(i) == ch) {
                count++;
            }
        }
        return count;
    }
    public static int countOccurrences(String str, String subStr){
        int count = 0;
        int index = 0;
        while(index != -1){
            index = str.indexOf(subStr, index);
            if(index != -1){
                count++;
                index += subStr.length();
            }
        }
        return count;
    }
}
```

11) Create a database for employees of an organization and perform CRUD operation.

Ans:-

```
import java.sql.*;
import java.util.ArrayList;
import java.util.List;
public class EmployeeDao{
    private Connection connection;
    public EmployeeDao{
        try{
            Class.forName("com.mysql.jdbc.Driver");
            String url = "jdbc:mysql://localhost:3306/mydatabase";
            String username = "root";
            String password = "password";
            connection = DriverManager.getConnection(url, username,
                password);
            }catch(ClassNotFoundException | SQLException e){
                e.printStackTrace();
            }
        }
    public void createEmployee(Employee employee){
        try{
            String sql = "INSERT INTO EMPLemployee(name, address,
            phone_number, email) VALUES (?,?,?,?)";
            PreparedStatement statement = connection.prepareS-
            -tatement(sql);
            statement.setString(1, employee.getName());
            statement.setString(2, employee.getAddress());
            statement.setString(3, employee.getPhoneNumber());
            statement.setString(4, employee.getEmail());
            statement.executeUpdate();
        }catch(SQLException e){
            e.printStackTrace();
        }
    }
```

```
public Employee getEmployeeById(int id){
    Employee employee = null;
    try{
      String sql = "SELECT * FROM employee WHERE id = ?";
     PreparedStatement statement = connection.prepare
      Statement (sql);
      statement.setInt(1,id);
     ResultSet resultSet = statement.executeQuery();
     if(resultSet.next()){
         employee = new Employee();
         employee.setId(resultSet.getInt("id"));
         employee.setName(resultSet.getString("name"));
         employee.setAddress(resultSet.getString("address"));
         employee.setPhoneNumer(resultSet.getString("phone_
                 number"));
          employee.setEmail(resultSet.getString("email"));
        }
     } catch(SQLException e){
          e.printStackTrace();
        }
        return employee;
    }
public List<Employee> getAllEmployees(){
    List<Employee> employees = new ArrayList<>();
    try{
      String sql = "SELECT * FROM employee";
     PreparedStatement statement = connection.prepare
      statement(sql);
     ResultSet resultSet = new Employee();
     employee.setId(resultSet.getInt("id"));
     employee.setName(resultSet.getString("name"));
     employee.setAddress(resultSet.getString("address"));
     employee.setPhoneNumber(resultSet.getString("p
         hone_number"));
```

```
                employee.setEmail(resultSet.getString("email"));
                employees.add(employee);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return employees;
    }

public void updateEmployee(Employee employee){
    try {
        String sql = "UPDATE employee SET name = ?, phone_number = ?,
            email = ?, WHERE id = ?";
        PreparedStatement statement = connection.prepareStatement(sql);
        statement.setString(1, employee.getName());
        statement.setString(2, employee.getAddress());
        statement.setString(3, employee.getPhoneNumber());
        statement.setString(4, employee.getEmail());
        statement.setString(5, employee.getId());
        statement.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

public void deleteEmployee(int id){
    try{
        String sql = "DELETE FROM employee WHERE id = ?";
        preparedStatement statement = connection.prepareStatement(sql);
        statement.executeUpdate();
    } catch (SQLException e){
        e.printStackTrace();
    }
}
```

```
public void close() {
    try {
        connection.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

public class Employee {
    private int id;
    private String name;
    private String address;
    private String phoneNumber;
    private String email;

    public Employee() {
    }

    public Employee(String name, String address, String
        phoneNumber, String email) {
        this.name = name;
        this.address = address;
        this.phoneNumber = phoneNumber;
        this.email = email;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }
```

```
    public void setName(String name){
        this.name = name;
    }
public String getAddress(){
    return address;
}
public void setAddress(String address){
    this.address = address;
}
public String getPhoneNumber(){
    return phoneNumber;
}
public void setPhoneNumber(String phoneNumber){
    this.phoneNumber = phoneNumber;
}
public String getEmail(){
    return email;
}
public void setEmail(String email){
    this.email= email;
}
}
public class Main{
    public static void main(String[] args){
        EmployeeDao dao = new EmployeeDao();
        Employee employee1 = new Employee("JohnDoe","123 Main St",
                "555-555-5555","johndoe@example.com");
        dao.createEmployee(employee1);
        Employee 2 = dao.getEmployeeById(1);
        System.out.println(employee2.getName());

        List<Employee> employees= dao.getAllEmployees();
        for(Employee employee: employees){
            System.out.println(employee.getName());
        }
```

```
employee2.setName("Jane Doe");
dao.updateEmployee(employee2);
dao.deleteEmployee(1);
dao.close()
}
```

Note: This code assumes the existence of a MySQL database named "mydatabase" with a table named "employee" with columns "id" (INT), "name" (VARCHAR), "address" (VARCHAR), "phone-number" (VARCHAR), and "email" (VARCHAR).

19) Write HTML tag and its uses.

Ans:-

- <html> : This tag indicates the start and end of an HTML document. All other HTML tags should be placed within this tag.
- <head> : This tag contains information about the web page that is not displayed on the page itself, such as the title of the page, links to stylesheets, and metadata.
- <title> : This tag is used to define the title of the page, which is displayed in the browser's title bar and search engine results.
- <body> : This tag contain visible content of the page, which includes text, images, and other HTML tags.
- <h1> to <h6> : This tags are used to create headings of the different sizes, with <h1> being the largest <h6> being the smallest.
- <b> : This text is used to create paragraph of text.
- <a> : This tag is used to create hyperlinks to other web pages or files.
- <img> : This tag is used to insert images into the webpages.

- and <li>: These tag is used to create ~~hyperlinks to other webpages or files~~. bulleted list <ul> creates an unordered list while <li> create each list item.

- and <li>: These tags are used to create numbered lists. creates an ordered list, while <li> create each list item.

- : These tag is used to group related elements together and apply styling or scripting to ~~small~~ them as a unit.

- : These tag is used to apply styling or scripting to small portions of text or other HTML elements.

- <table>, <tr>, <th>, <td> : These tags are used to create tables. <table>: creates the table; <tr> create each row <th> creates a header cell and <td> creates a data cell.

- <form>, <input>, <button> : These tags are used to create ~~tables~~. forms of user input. <form> creates the form <input> creates various types of form input fields, and <button> creates buttons that trigger form action.

- <select>, <option> ; These tags are used to create drop-down menus for form input. <select> : creates the menu, while <option> creates each selectable option in the menu.

- <label> : This tag is used to create text label for a form input field.

- <textarea>; This tag is used to create a larger text input field for user input.

-
; This tag is used to create a ~~text label~~ line break, moving the next content to a new line.

- <hr>; This tag creates a horizontal line across the page, often used to visually separate content.

- <style>: This tag is used to define CSS styles for the pages or specific HTML elements.

- <script>: This tag is used to insert JavaScript code into the page

- <meta>: This tag contains metadata about the pages, such as keywords and description, which is often used by search engines

- <nav>, <header>, <footer>, <section>, <article>, <aside>: These tags are used to create structural elements for the page, making it easier to understand the organization of the content. <nav> creates a navigation bar. <header> creates a page header <footer> creates a pagefooter, <section> creates a section of related content, <article> create an article or story, and <aside> create a related content section.

13) What do you mean by stored procedure in databases.

Ans:-

A stored procedure in a database is a precompiled set of SQL statements and procedural logic that is stored in the database and can be executed repeatedly. It is similar to a function in a programming language that can be called with parameters and can return results.

→ In Java, stored procedures can be used to execute complex database operations that require multiple SQL statements, such as data validation, updates and queries. Java provides several ways to call stored procedures, including the JDBC (Java Database Connectivity) API and various ORM (Object-Relational Mapping) framework such as Hibernate and JPA (Java Persistence API).

→ To call stored procedure in Java, we need to first define it in the database and then write java code to execute it.

→ The JDBC API provides methods to prepare and execute stored procedure, pass parameters to them and retrive results.

Calling a stored procedure that return a result set.

```
String sql = "{call get.employees(?,?)}";
CallableStatement stmt = conn.prepareCall(sql);
stmt.setInt(1,1);
stmt.setInt(2,10);
ResultSet rs = stmt.executeQuery();
while(rs.next()) {
        int empId = rs.getInt("employee.id");
```

```
String empName = rs.getString("employee.name");
double empSalary = rs.getDouble("employee.salary");
System.out.println(empId + " " + empName + " " + empSalary);
}
```

In this example, the "get_employees" stored procedure is called with two parameter (the start and end endices of the result set to retrive). The ResultSet object is used to iterate over the rows of the result set and extract the values of each column.

14) How many types of driver are available in java explain it.

Ans:-

In Java, there are four types of JDBC (Java Database Connectivity) drivers available.

1) JDBC-ODBC Bridge Driver:- This type of driver provides a JDBC front-end to ODBC (Open Database Connectivity) drivers, allowing Java programs to access ODBC-accessible databases. The JDBC-ODBC Bridge driver is provided by Sun MicroSystems and is included in the Java Development Kit (JDK).

2) Native API Driver:- This type of driver uses database-specific native code (written in languages such as C or C++) to access the database directly. This driver is specific to a particular database and operating system, and requires the appropriate native libraries to be installed. Native API drivers are usually provided by the database vendors.

3) Network Protocol Driver:- This type of driver uses a middle-tier server to communicate with the database server over a network protocol such as TCP/IP. The java program connects to the middle-tier server using JDBC, and the middle

tier server communicates with the database server using the database's native protocol. Network protocol drivers are also known as client-server drivers or two-tier drivers.

4) Thin Driver :- This type of driver is a pure Java implementation of the JDBC API and communicates directly with the database server over a network protocol such as TCP/IP. The thin driver doesnot require any native code or middleware, and is provided by the database vendors as a Java library. Thin drivers are also known as type 4 drivers, and are most commonly used type of JDBC driver today.