Core Java

1)

Given:

```
public class TaxUtil {
  double rate = 0.15;

  public double calculateTax(double amount) {
     return amount * rate;
  }
}
```

Would you consider the method calculateTax() a 'pure function'? Why or why not?

If you claim the method is NOT a pure function, please suggest a way to make it pure.

1)

A function is pure if:

- It always returns the same output for the same input.
- It has no side effects (doesn't read or modify any external state).

Given Code:

```
public class TaxUtil {

        double rate = 0.15;

        public double calculateTax(double amount) {

        return amount * rate;

        }

}
```

calculateTax() is not a pure function because:

- It relies on an instance variable rate, which can be changed externally.
- Therefore, the output of calculateTax(amount) may vary even if amount stays the same.

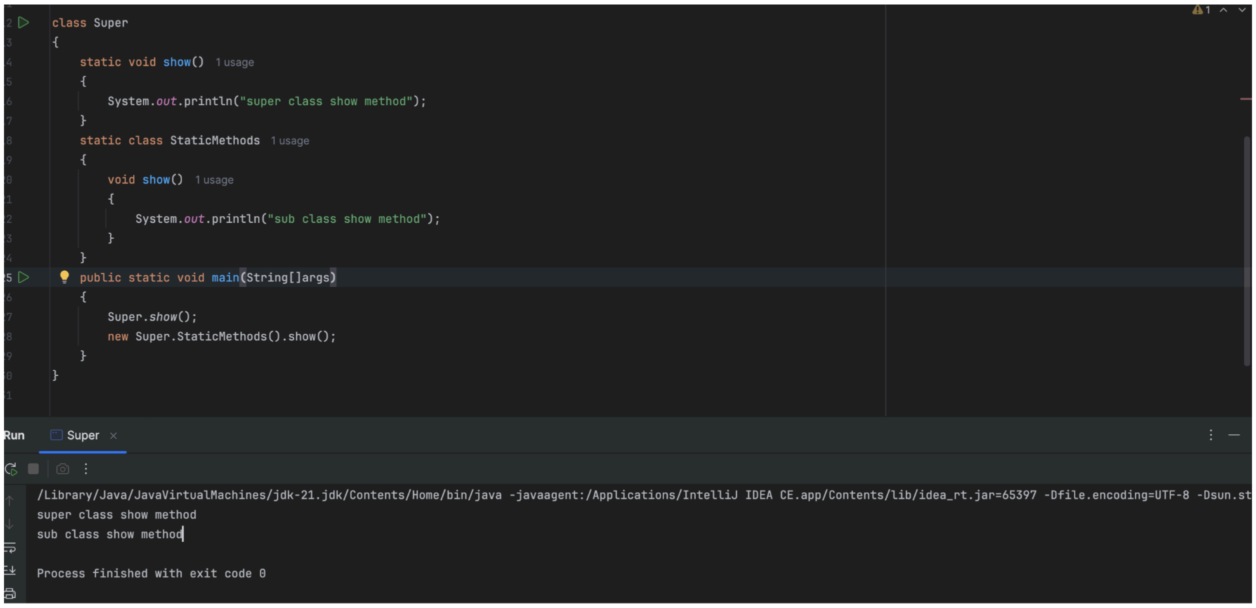
To make it pure make the method independent of external state by passing rate as a parameter:

```
public class TaxUtil {



        public double calculateTax(double amount, double rate) {

            return amount * rate;

          }

        }
```

Now, calculateTax(100.0, 0.15) will always return the same result.

2)
What will be the output for the following code?

```
class Super
{
   static void show()
   {
      System.out.println("super class show method");
   }
   static class StaticMethods
   {
      void show()
      {
         System.out.println("sub class show method");
      }
   }
   public static void main(String[]args)
   {
      Super.show();
      new Super.StaticMethods().show();
   }
}
```



```
class Super
{
    static void show()  1 usage
    {
        System.out.println("super class show method");
    }
    static class StaticMethods  1 usage
    {
        void show()  1 usage
        {
            System.out.println("sub class show method");
        }
    }
    public static void main(String[]args)
    {
        Super.show();
        new Super.StaticMethods().show();
    }
}
```

```
/Library/Java/JavaVirtualMachines/jdk-21.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib/idea_rt.jar=65397 -Dfile.encoding=UTF-8 -Dsun.st
super class show method
sub class show method

Process finished with exit code 0
```

Final Output:
super class show method
sub class show method

3) What will be the output for the following code?

```
class Super
{
   int num=20;
   public void display()
   {
      System.out.println("super class method");
   }
}
public class ThisUse extends Super
{
   int num;
   public ThisUse(int num)
   {
      this.num=num;
   }
   public void display()
   {
      System.out.println("display method");
   }
   public void Show()
   {
        this.display();
        display();
        System.out.println(this.num);
        System.out.println(num);
    }
    public static void main(String[]args)
    {
        ThisUse o=new ThisUse(10);
        o.show();
    }
}
```

Java is case-sensitive, and the code has:

```
o.show();
```

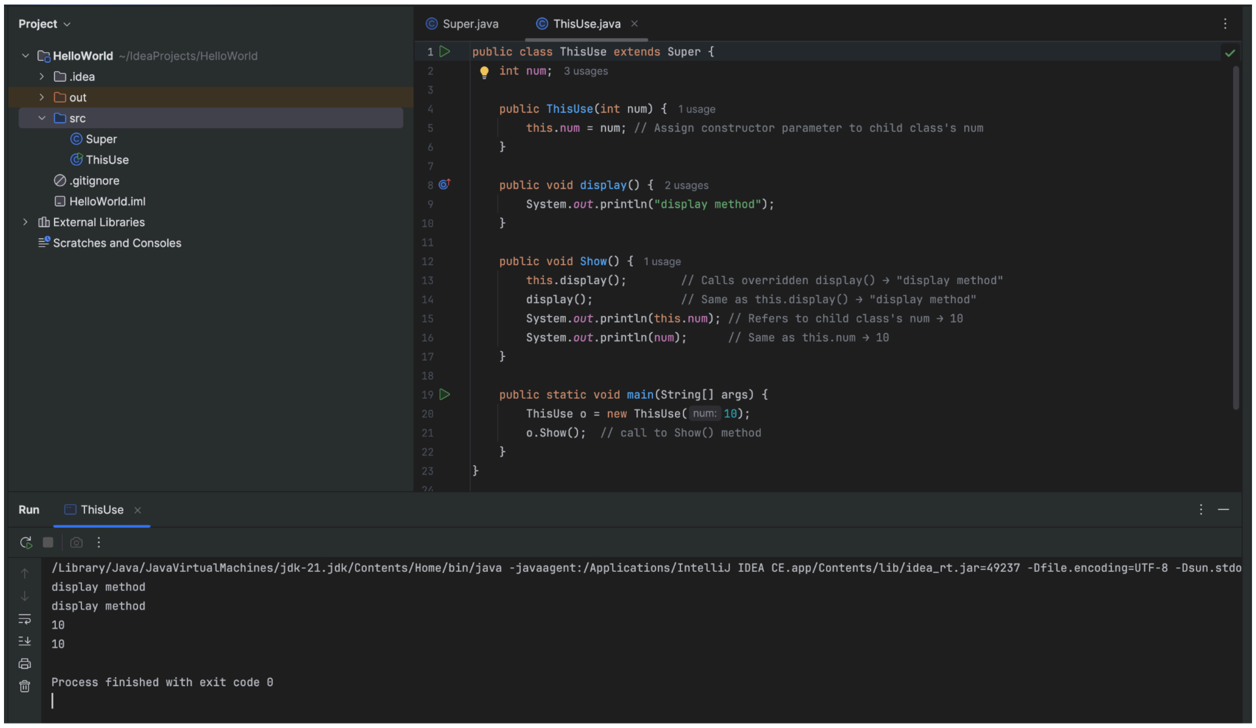
But the method is defined as:

```
public void Show()
```

So `o.show();` will cause a compile-time error

On correcting the method call in `main()` to:

`o.Show();`



```java
public class ThisUse extends Super {
    int num;  3 usages

    public ThisUse(int num) {  1 usage
        this.num = num; // Assign constructor parameter to child class's num
    }

    public void display() {  2 usages
        System.out.println("display method");
    }

    public void Show() {  1 usage
        this.display();        // Calls overridden display() → "display method"
        display();             // Same as this.display() → "display method"
        System.out.println(this.num); // Refers to child class's num → 10
        System.out.println(num);      // Same as this.num → 10
    }

    public static void main(String[] args) {
        ThisUse o = new ThisUse( num: 10);
        o.Show();  // call to Show() method
    }
}
```

```
/Library/Java/JavaVirtualMachines/jdk-21.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib/idea_rt.jar=49237 -Dfile.encoding=UTF-8 -Dsun.stdo
display method
display method
10
10

Process finished with exit code 0
```

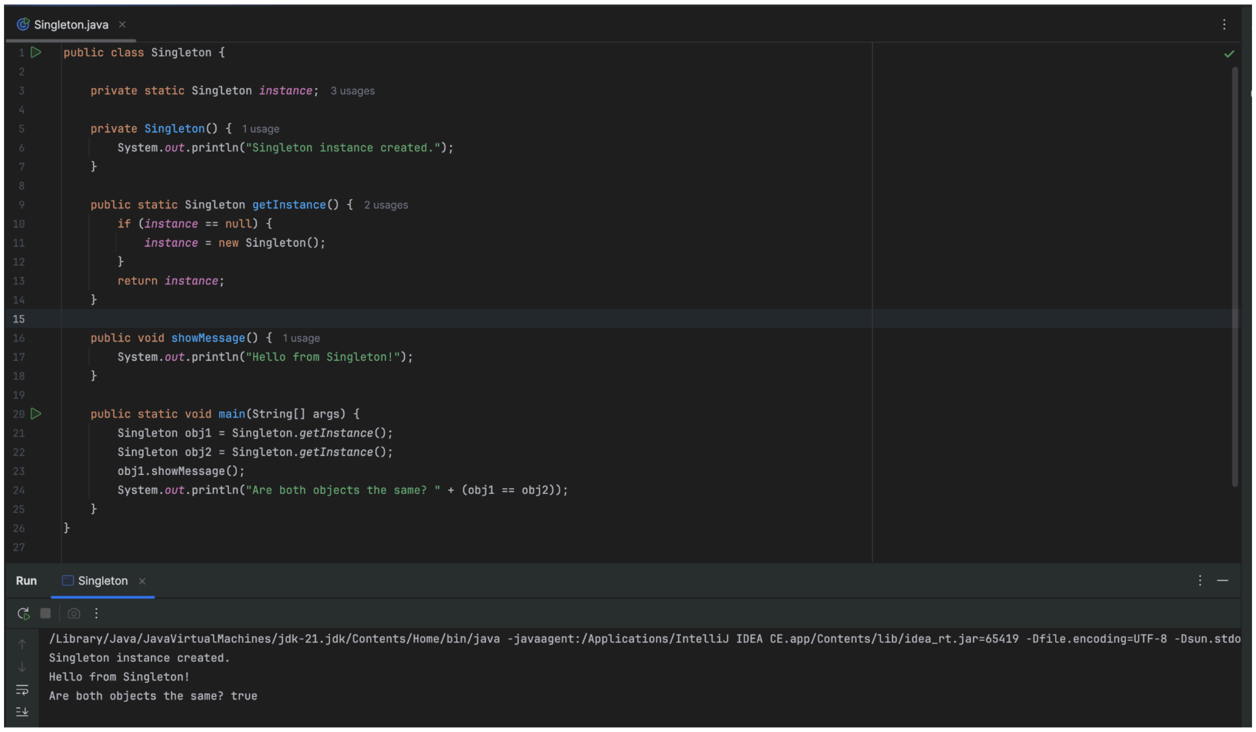
4) What is the singleton design pattern? Explain with a coding example.

The Singleton Design Pattern ensures that a class has only one instance throughout the program and provides a global access point to that instance.

It's commonly used for managing shared resources such as configurations, logging, database connections, etc.

Key Features:

- Private constructor to prevent instantiation from outside.
- Static variable to hold the single instance.
- Public static method (often `getInstance()`) to return the instance.



```java
public class Singleton {

    private static Singleton instance;  3 usages

    private Singleton() {  1 usage
        System.out.println("Singleton instance created.");
    }

    public static Singleton getInstance() {  2 usages
        if (instance == null) {
            instance = new Singleton();
        }
        return instance;
    }

    public void showMessage() {  1 usage
        System.out.println("Hello from Singleton!");
    }

    public static void main(String[] args) {
        Singleton obj1 = Singleton.getInstance();
        Singleton obj2 = Singleton.getInstance();
        obj1.showMessage();
        System.out.println("Are both objects the same? " + (obj1 == obj2));
    }
}
```

```
/Library/Java/JavaVirtualMachines/jdk-21.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib/idea_rt.jar=65419 -Dfile.encoding=UTF-8 -Dsun.stdo
Singleton instance created.
Hello from Singleton!
Are both objects the same? true
```

5) How do we make sure a class is encapsulated? Explain with a coding example.

Encapsulation is an object-oriented programming principle where:

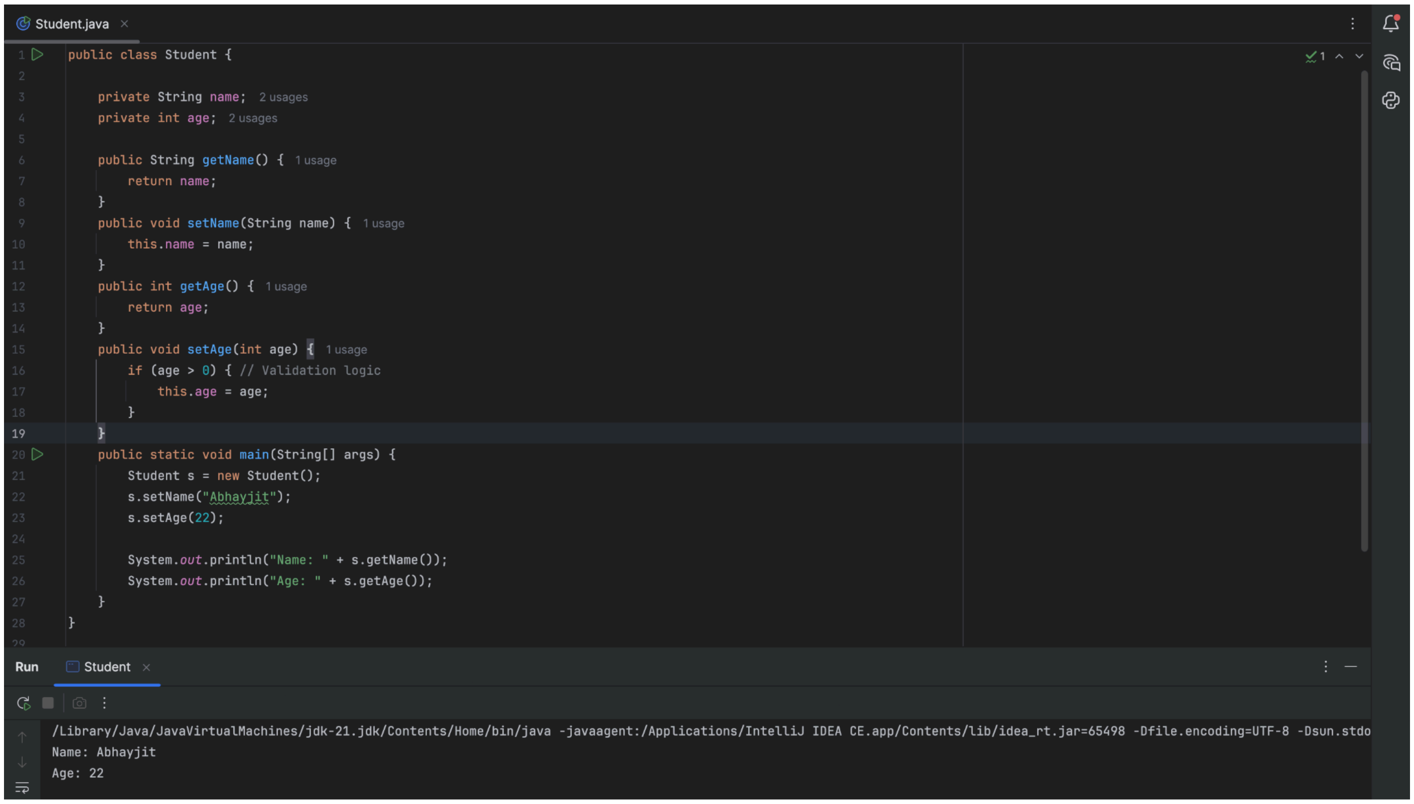
- The internal state (data) of a class is hidden from the outside.
- Access to that data is provided only through public methods (getters and setters).

It helps:

- Protect data from unauthorized access or modification.
- Maintain control over how values are set or retrieved.
- Achieve data hiding and modularity.

To ensure a Class is Encapsulated:

- Declare all fields as private.
- Provide public getter and setter methods to access and modify the fields.



```java
public class Student {

    private String name;  2 usages
    private int age;  2 usages

    public String getName() {  1 usage
        return name;
    }
    public void setName(String name) {  1 usage
        this.name = name;
    }
    public int getAge() {  1 usage
        return age;
    }
    public void setAge(int age) {  1 usage
        if (age > 0) { // Validation logic
            this.age = age;
        }
    }
    public static void main(String[] args) {
        Student s = new Student();
        s.setName("Abhayjit");
        s.setAge(22);

        System.out.println("Name: " + s.getName());
        System.out.println("Age: " + s.getAge());
    }
}
```

```
/Library/Java/JavaVirtualMachines/jdk-21.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib/idea_rt.jar=65498 -Dfile.encoding=UTF-8 -Dsun.stdo
Name: Abhayjit
Age: 22
```

6) Perform CRUD operation using ArrayList collection in an EmployeeCRUD class for the below Employee

```
class Employee{
        private int id;
        private String name;
        private String department;
    }
```

Java Code:

Employee Class

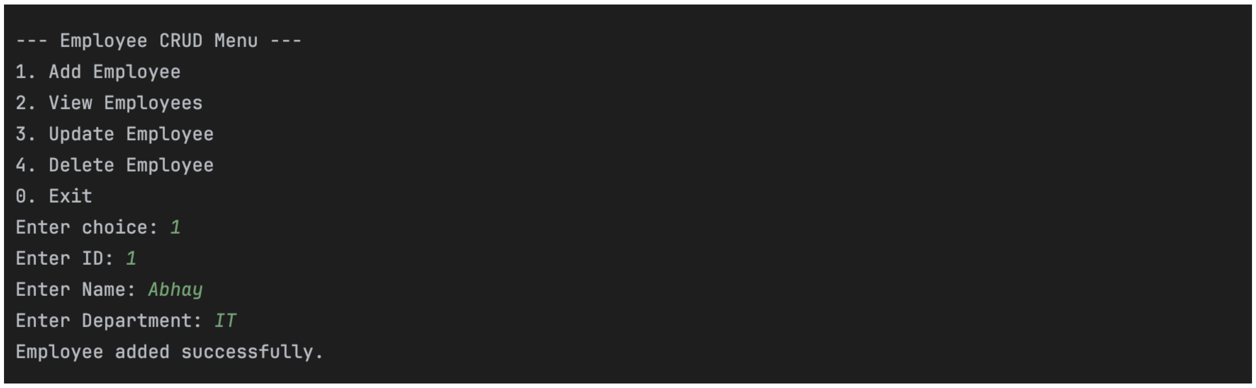
```
public class Employee {
    private int id;
    private String name;
    private String department;

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

    public int getId() {
        return id;
    }

    public String getName() {
        return name;
    }

    public String getDepartment() {
        return department;
    }

    public void setName(String name) {
        this.name = name;
    }

    public void setDepartment(String department) {
        this.department = department;
    }

    public String toString() {
        return "Employee [ID=" + id + ", Name=" + name + ", Department=" + department + "]";
    }
}
```

EmployeeCRUD Class

```
import java.util.ArrayList;
import java.util.Scanner;

public class EmployeeCRUD {
    private ArrayList<Employee> employees = new ArrayList<>();
    private Scanner scanner = new Scanner(System.in);

    // Create
    public void addEmployee(Employee emp) {
        employees.add(emp);
        System.out.println("Employee added successfully.");
    }

    // Read
    public void viewEmployees() {
        if (employees.isEmpty()) {
            System.out.println("No employees found.");
        } else {
            for (Employee emp : employees) {
                System.out.println(emp);
            }
        }
    }

    // Update
    public void updateEmployee(int id) {
        for (Employee emp : employees) {
            if (emp.getId() == id) {
                System.out.print("Enter new name: ");
                String name = scanner.nextLine();
                System.out.print("Enter new department: ");
                String dept = scanner.nextLine();
                emp.setName(name);
                emp.setDepartment(dept);
                System.out.println("Employee updated successfully.");
                return;
            }
        }
        System.out.println("Employee with ID " + id + " not found.");
    }

    // Delete
```

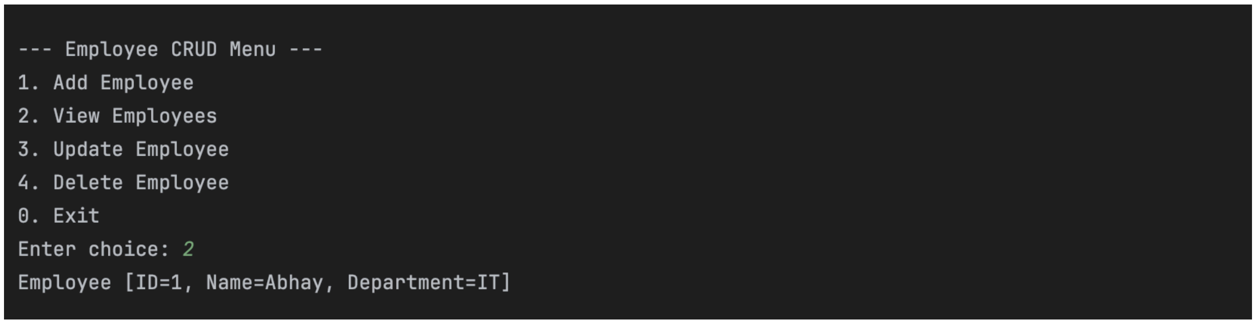
```
public void deleteEmployee(int id) {
    for (Employee emp : employees) {
        if (emp.getId() == id) {
            employees.remove(emp);
            System.out.println("Employee deleted successfully.");
            return;
        }
    }
    System.out.println("Employee with ID " + id + " not found.");
}

public static void main(String[] args) {
    EmployeeCRUD crud = new EmployeeCRUD();
    Scanner sc = new Scanner(System.in);
    int choice;

    do {
        System.out.println("\n--- Employee CRUD Menu ---");
        System.out.println("1. Add Employee");
        System.out.println("2. View Employees");
        System.out.println("3. Update Employee");
        System.out.println("4. Delete Employee");
        System.out.println("0. Exit");
        System.out.print("Enter choice: ");
        choice = sc.nextInt();
        sc.nextLine(); // consume newline

        switch (choice) {
            case 1:
                System.out.print("Enter ID: ");
                int id = sc.nextInt();
                sc.nextLine(); // consume newline
                System.out.print("Enter Name: ");
                String name = sc.nextLine();
                System.out.print("Enter Department: ");
                String dept = sc.nextLine();
                crud.addEmployee(new Employee(id, name, dept));
                break;
            case 2:
                crud.viewEmployees();
                break;
            case 3:
                System.out.print("Enter ID to update: ");
                int updateId = sc.nextInt();
                sc.nextLine(); // consume newline
                crud.updateEmployee(updateId);
                break;
            case 4:
                System.out.print("Enter ID to delete: ");
                int deleteId = sc.nextInt();
                sc.nextLine(); // consume newline
                crud.deleteEmployee(deleteId);
                break;
            case 0:
                System.out.println("Exiting program.");
                break;
            default:
                System.out.println("Invalid choice!");
          }
        } while (choice != 0);

        sc.close();
    }
}
```

Create employee



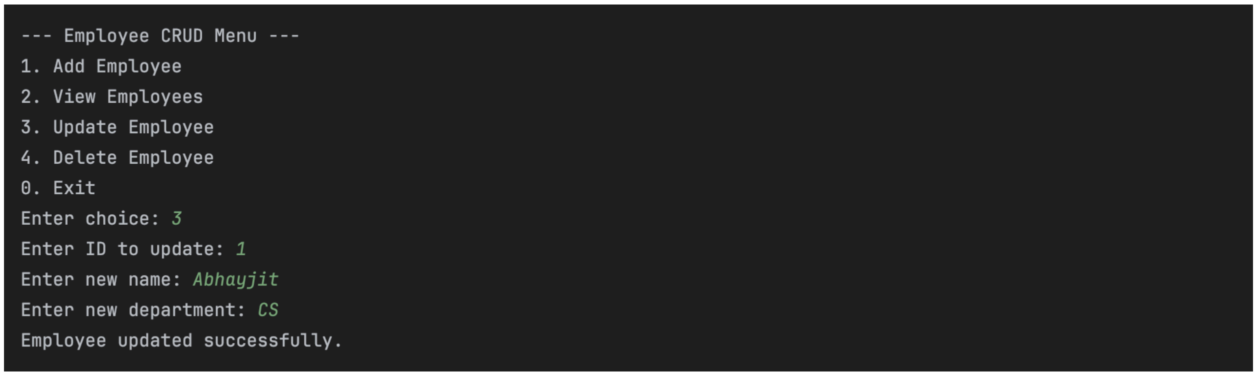
```
--- Employee CRUD Menu ---
1. Add Employee
2. View Employees
3. Update Employee
4. Delete Employee
0. Exit
Enter choice: 1
Enter ID: 1
Enter Name: Abhay
Enter Department: IT
Employee added successfully.
```

Read Employee



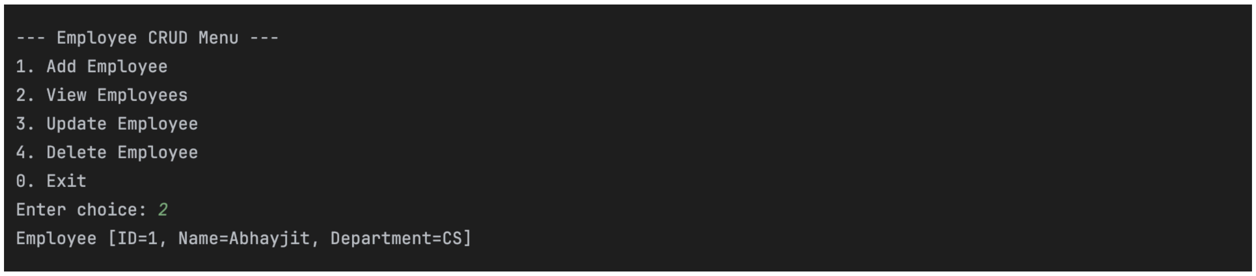
```
--- Employee CRUD Menu ---
1. Add Employee
2. View Employees
3. Update Employee
4. Delete Employee
0. Exit
Enter choice: 2
Employee [ID=1, Name=Abhay, Department=IT]
```

Update Employee



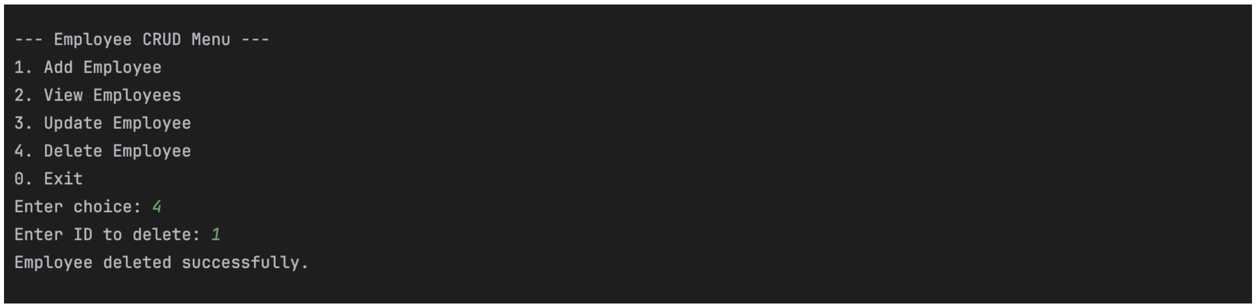
```
--- Employee CRUD Menu ---
1. Add Employee
2. View Employees
3. Update Employee
4. Delete Employee
0. Exit
Enter choice: 3
Enter ID to update: 1
Enter new name: Abhayjit
Enter new department: CS
Employee updated successfully.
```

Read Employee



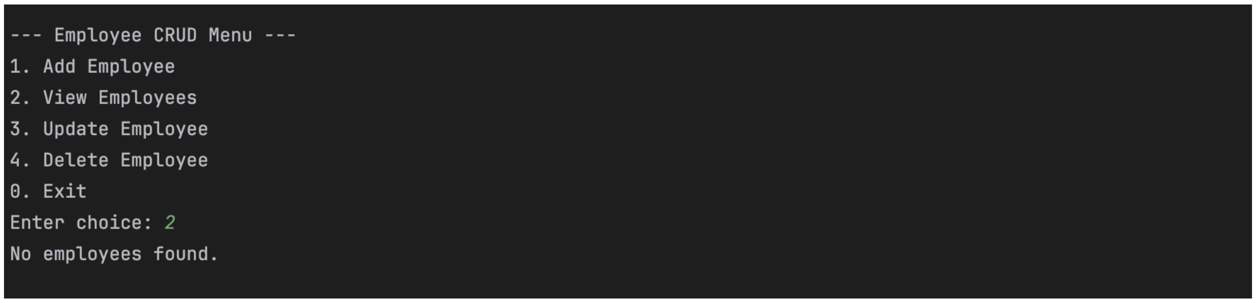
```
--- Employee CRUD Menu ---
1. Add Employee
2. View Employees
3. Update Employee
4. Delete Employee
0. Exit
Enter choice: 2
Employee [ID=1, Name=Abhayjit, Department=CS]
```

Delete Employee



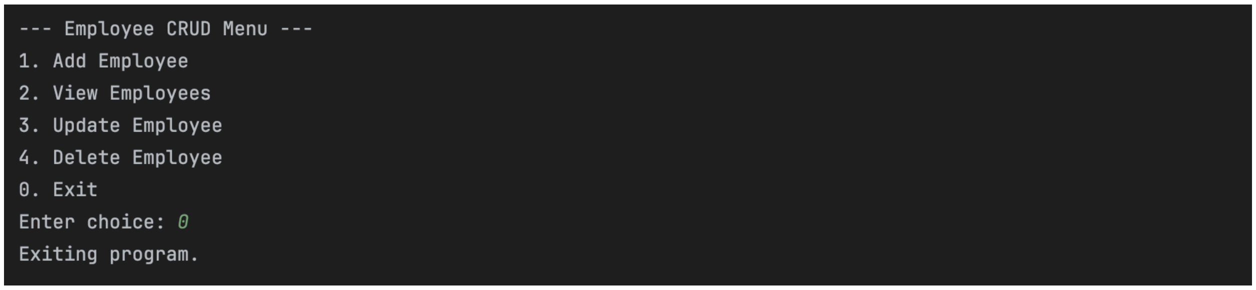
```
--- Employee CRUD Menu ---
1. Add Employee
2. View Employees
3. Update Employee
4. Delete Employee
0. Exit
Enter choice: 4
Enter ID to delete: 1
Employee deleted successfully.
```

Read Employee



```
--- Employee CRUD Menu ---
1. Add Employee
2. View Employees
3. Update Employee
4. Delete Employee
0. Exit
Enter choice: 2
No employees found.
```

Exit



```
--- Employee CRUD Menu ---
1. Add Employee
2. View Employees
3. Update Employee
4. Delete Employee
0. Exit
Enter choice: 0
Exiting program.
```

7) Perform CRUD operation using JDBC in an EmployeeJDBC class for the below Employee

```
class Employee{
        private int id;
        private String name;
        private String department;
}
```

Java Code

Employee Class

```
public class Employee {
   private int id;
   private String name;
   private String department;

   // Constructor
   public Employee(int id, String name, String department) {
      this.id = id;
      this.name = name;
      this.department = department;
   }

   // Getters
   public int getId() {
      return id;
```

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

    // Setters
    public void setName(String name) {
        this.name = name;
    }
    public void setDepartment(String department) {
        this.department = department;
    }

    // toString
    @Override
    public String toString() {
        return "Employee [ID=" + id + ", Name=" + name + ", Department=" + department + "]";
    }
}
```

EmployeeJDBC

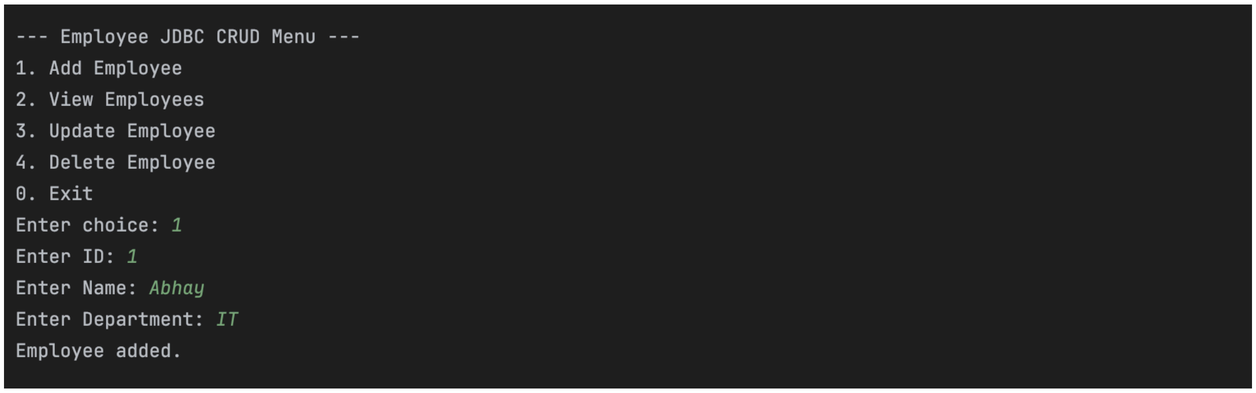
```
import java.sql.*;
import java.util.Scanner;

public class EmployeeJDBC {
    private static final String URL = "jdbc:mysql://localhost:3306/employee";
    private static final String USER = "root";
    private static final String PASSWORD = "mysql123";

    private Connection connect() throws SQLException {
        return DriverManager.getConnection(URL, USER, PASSWORD);
    }

    // CREATE
    public void addEmployee(Employee emp) {
        String query = "INSERT INTO employee (id, name, department) VALUES (?, ?, ?)";
        try (Connection conn = connect(); PreparedStatement ps = conn.prepareStatement(query))
{
            ps.setInt(1, emp.getId());
            ps.setString(2, emp.getName());
            ps.setString(3, emp.getDepartment());
            ps.executeUpdate();
            System.out.println("Employee added.");
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    // READ
    public void viewEmployees() {
        String query = "SELECT * FROM employee";
        try (Connection conn = connect(); Statement stmt = conn.createStatement(); ResultSet rs =
stmt.executeQuery(query)) {
            while (rs.next()) {
                Employee emp = new Employee(rs.getInt("id"), rs.getString("name"),
rs.getString("department"));
                System.out.println(emp);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    // UPDATE
    public void updateEmployee(int id, String newName, String newDept) {
        String query = "UPDATE employee SET name = ?, department = ? WHERE id = ?";
        try (Connection conn = connect(); PreparedStatement ps = conn.prepareStatement(query))
{
            ps.setString(1, newName);
            ps.setString(2, newDept);
            ps.setInt(3, id);
            int rows = ps.executeUpdate();
            if (rows > 0) {
                System.out.println("Employee updated.");
            } else {
                System.out.println("Employee not found.");
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    // DELETE
    public void deleteEmployee(int id) {
        String query = "DELETE FROM employee WHERE id = ?";
        try (Connection conn = connect(); PreparedStatement ps = conn.prepareStatement(query))
    {
            ps.setInt(1, id);
            int rows = ps.executeUpdate();
            if (rows > 0) {
                System.out.println("Employee deleted.");
            } else {
                System.out.println("Employee not found.");
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    // MAIN method for testing
    public static void main(String[] args) {
        EmployeeJDBC crud = new EmployeeJDBC();
        Scanner sc = new Scanner(System.in);
        int choice;

        do {
            System.out.println("\n--- Employee JDBC CRUD Menu ---");
            System.out.println("1. Add Employee");
            System.out.println("2. View Employees");
            System.out.println("3. Update Employee");
            System.out.println("4. Delete Employee");
            System.out.println("0. Exit");
            System.out.print("Enter choice: ");
            choice = sc.nextInt();
            sc.nextLine(); // consume newline

            switch (choice) {
                case 1:
                    System.out.print("Enter ID: ");
                    int id = sc.nextInt();
                    sc.nextLine();
                    System.out.print("Enter Name: ");
                    String name = sc.nextLine();
                    System.out.print("Enter Department: ");
                    String dept = sc.nextLine();
                    crud.addEmployee(new Employee(id, name, dept));
                    break;
                case 2:
```

```
                crud.viewEmployees();
                break;
            case 3:
                System.out.print("Enter ID to update: ");
                int uid = sc.nextInt();
                sc.nextLine();
                System.out.print("Enter new Name: ");
                String newName = sc.nextLine();
                System.out.print("Enter new Department: ");
                String newDept = sc.nextLine();
                crud.updateEmployee(uid, newName, newDept);
                break;
            case 4:
                System.out.print("Enter ID to delete: ");
                int did = sc.nextInt();
                sc.nextLine();
                crud.deleteEmployee(did);
                break;
            case 0:
                System.out.println("Exiting...");
                break;
            default:
                System.out.println("Invalid choice!");
        }
    } while (choice != 0);

    sc.close();
  }
}
```

MYSQL TABLE CREATION

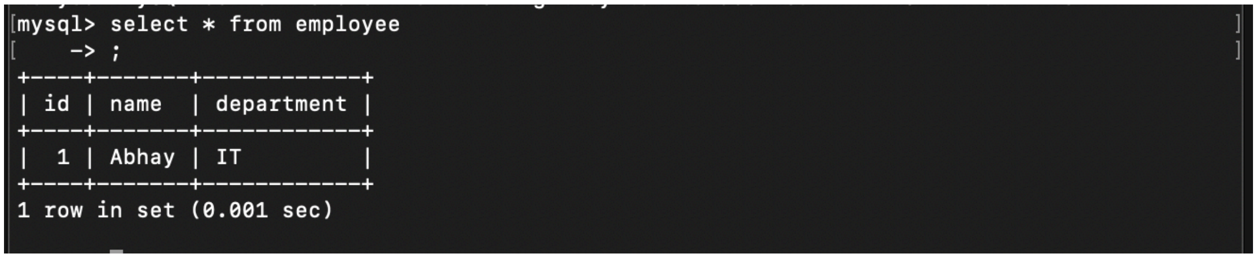
```
CREATE TABLE employee (
        id INT PRIMARY KEY,
        name VARCHAR(100),
        department VARCHAR(100)
);
```

Create Employee



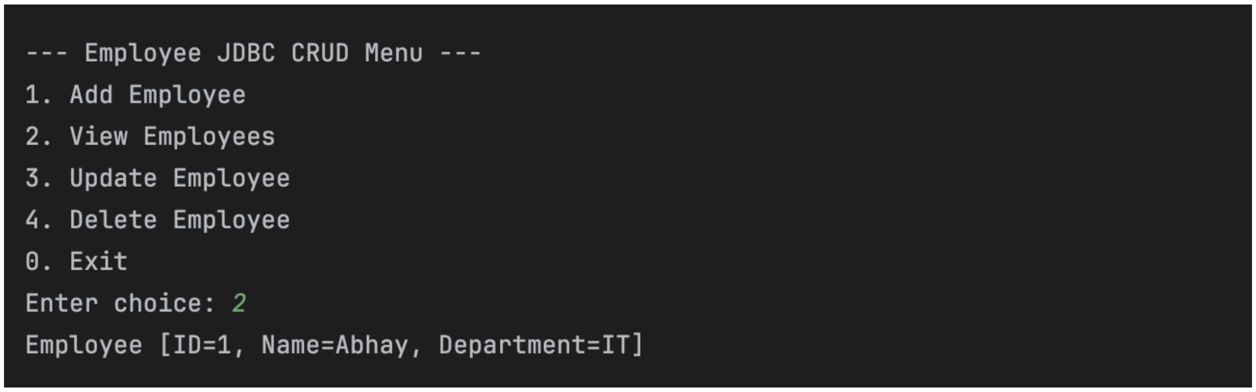
```
--- Employee JDBC CRUD Menu ---
1. Add Employee
2. View Employees
3. Update Employee
4. Delete Employee
0. Exit
Enter choice: 1
Enter ID: 1
Enter Name: Abhay
Enter Department: IT
Employee added.
```

Database



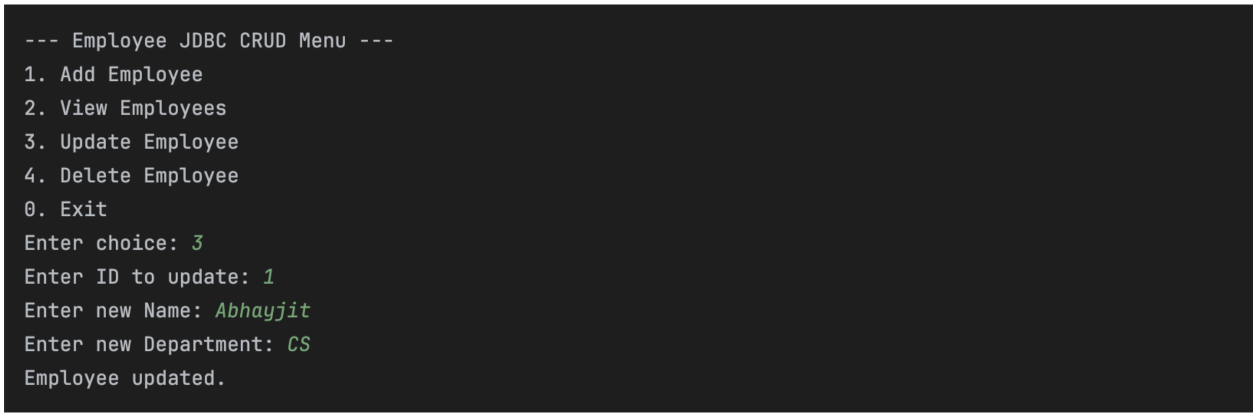
```
mysql> select * from employee
    -> ;
+----+-------+------------+
| id | name  | department |
+----+-------+------------+
|  1 | Abhay | IT         |
+----+-------+------------+
1 row in set (0.001 sec)
```

Read



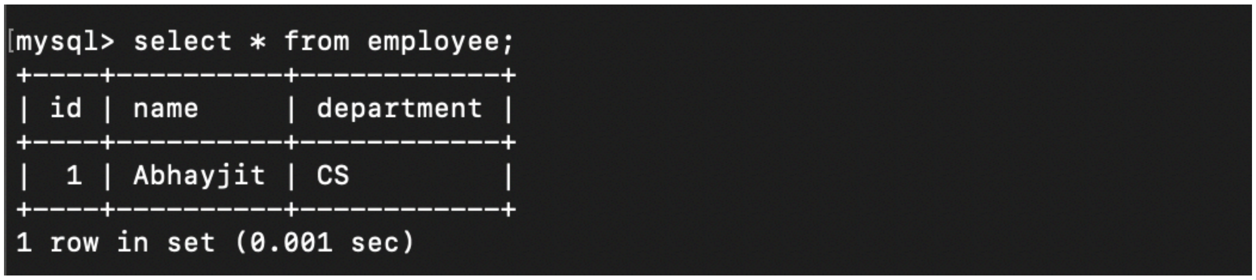
```
--- Employee JDBC CRUD Menu ---
1. Add Employee
2. View Employees
3. Update Employee
4. Delete Employee
0. Exit
Enter choice: 2
Employee [ID=1, Name=Abhay, Department=IT]
```

Update



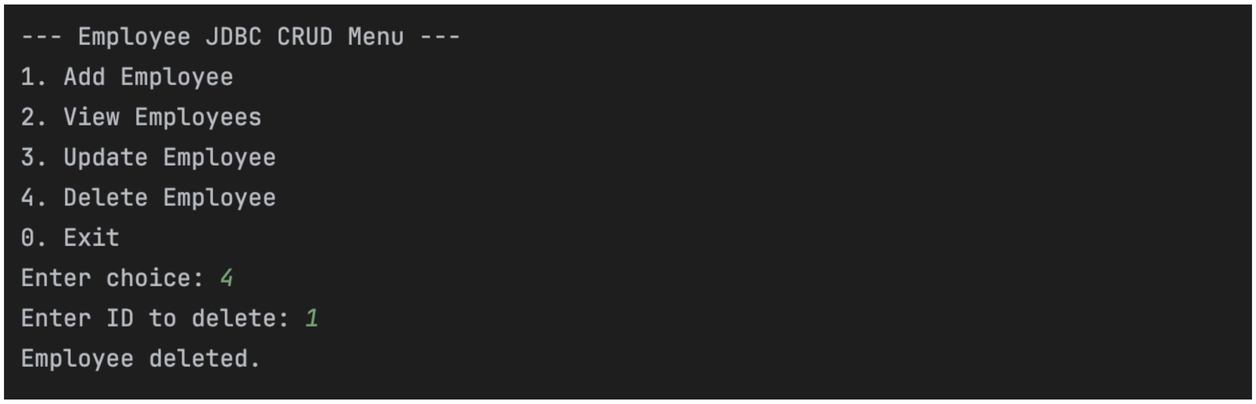
```
--- Employee JDBC CRUD Menu ---
1. Add Employee
2. View Employees
3. Update Employee
4. Delete Employee
0. Exit
Enter choice: 3
Enter ID to update: 1
Enter new Name: Abhayjit
Enter new Department: CS
Employee updated.
```

Database



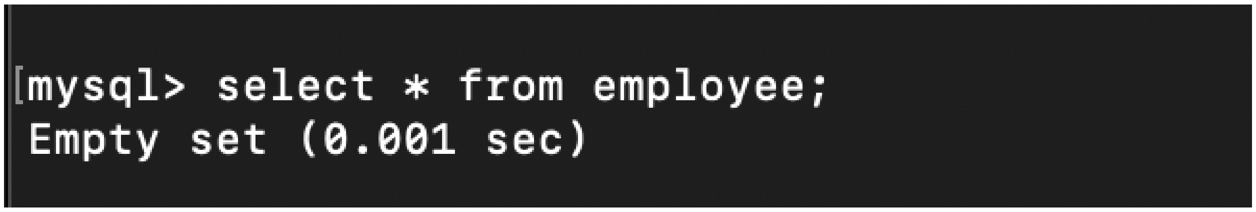
```
mysql> select * from employee;
+----+----------+------------+
| id | name     | department |
+----+----------+------------+
|  1 | Abhayjit | CS         |
+----+----------+------------+
1 row in set (0.001 sec)
```

Delete



```
--- Employee JDBC CRUD Menu ---
1. Add Employee
2. View Employees
3. Update Employee
4. Delete Employee
0. Exit
Enter choice: 4
Enter ID to delete: 1
Employee deleted.
```

Database



```
[mysql> select * from employee;
Empty set (0.001 sec)
```