Question 1

a) Define Object Oriented Programming. (5 marks)

Object-Oriented Programming (OOP) is a programming paradigm based on the concept of objects. Objects are instances of classes, which can encapsulate data and behavior (methods) together. OOP allows for modularity, code reuse, and abstraction by organizing software design around objects rather than functions and logic. Key principles of OOP include encapsulation, inheritance, polymorphism, and abstraction.

b) Name any five primitive data types (apart from char) in Java, and give an appropriate example of an attribute where it may be used, for example, char can be used for gender: M or F. (5x2 marks)

1. int:
    - Example: `int age;` // Used for storing a person's age.
2. double:
    - Example: `double salary;` // Used for storing an employee's salary.
3. boolean:
    - Example: `boolean isMarried;` // Used to store the marital status of a person.
4. byte:
    - Example: `byte grade;` // Used to store a student's grade (assuming it's on a scale from 0 to 100).
5. float:
    - Example: `float temperature;` // Used for storing the temperature in degrees Celsius.

c) Write a Java method to calculate the VAT amount for a product. A product may be zero-rated or taxable at 15%. A product may also be entitled to a discount percentage. Then, create a main method to test the above method. (10 marks)

```
public class VATCalculator {

    public static double calculateVAT(double price, boolean isTaxable, double discountPerc
        double discountedPrice = price - (price * discountPercentage / 100);
        double vat = 0;
        if (isTaxable) {
            vat = discountedPrice * 0.15;
        }
        return vat;
    }

    public static void main(String[] args) {
        double price1 = 100.00;
        boolean isTaxable1 = true;
        double discount1 = 10.0; // 10% discount

        double vat1 = calculateVAT(price1, isTaxable1, discount1);
        System.out.println("VAT for product 1: " + vat1);

        double price2 = 200.00;
        boolean isTaxable2 = false;
        double discount2 = 5.0; // 5% discount

        double vat2 = calculateVAT(price2, isTaxable2, discount2);
        System.out.println("VAT for product 2: " + vat2);
    }
}
```

Explanation:

- The `calculateVAT` method calculates the VAT amount for a product. It takes three parameters: `price` (the original price of the product), `isTaxable` (a boolean indicating if the product is taxable or not), and `discountPercentage` (the discount percentage to be applied to the product).
- The method first calculates the discounted price by applying the discount to the original price.
- It then checks if the product is taxable. If it is, it calculates the VAT as 15% of the discounted price.

- The `main` method demonstrates the use of the `calculateVAT` method by calculating the VAT for two different products and printing the results.

Question 2

a) Write a Java program that creates an array of 2 book objects with the following information:

```java
class Book {
    String ISBN;
    String title;
    String author;
    int pages;

    Book(String ISBN, String title, String author, int pages) {
        this.ISBN = ISBN;
        this.title = title;
        this.author = author;
        this.pages = pages;
    }

    void display() {
        System.out.println("ISBN: " + ISBN);
        System.out.println("Title: " + title);
        System.out.println("Author: " + author);
        System.out.println("Pages: " + pages);
    }
}

public class Main {
    public static void main(String[] args) {
        Book[] books = new Book[2];

        books[0] = new Book("111-222", "Mystery Island", "John Smith", 100);
        books[1] = new Book("222-231", "The Crime", "Louis Paul", 125);

        for (Book book : books) {
            book.display();
            System.out.println();
        }
    }
}
```

b) Arrays, ArrayList and LinkedList form part of Java Data Structures. For each data structure:

i. Mention in which circumstances each is more appropriate to use. (3 marks)

1. **Arrays:**

    - Use when the size of the collection is fixed and known in advance.
    - Provides fast access to elements using index.
    - Suitable for simple data structures and performance-critical applications where memory overhead needs to be minimized.
2. ArrayList:
    - Use when the size of the collection can vary dynamically.
    - Provides fast random access to elements using index.
    - Suitable for applications where frequent read operations are required, and occasional additions or deletions are made.
3. LinkedList:
    - Use when frequent insertions and deletions are required.
    - Provides sequential access to elements.
    - Suitable for applications where elements are frequently added or removed from the beginning or end of the list.

ii. Give the Java code to declare an instance of each. (3 marks)

1. Array:
    ```java
    int[] array = new int[10];
    ```
2. ArrayList:
    ```java
    ArrayList<String> arrayList = new ArrayList<>();
    ```
3. LinkedList:
    ```java
    LinkedList<String> linkedList = new LinkedList<>();
    ```

iii. Compare the features and advantages of each. (9 marks)

1. Arrays:
    - Features:
        - Fixed size.
        - Direct access to elements via index.
        - Stores elements of the same type.
    - Advantages:
        - Low memory overhead.

        - Fast access to elements using index.
        - Suitable for performance-critical applications.

2. ArrayList:
    - Features:
        - Dynamic size, can grow and shrink as needed.
        - Allows random access to elements using index.
        - Implements the `List` interface.
    - Advantages:
        - Provides better performance than `LinkedList` for accessing elements by index.
        - Dynamic resizing allows flexibility in managing the size of the collection.
        - Supports various convenient methods like `add`, `remove`, and `contains`.
3. LinkedList:
    - Features:
        - Dynamic size.
        - Elements are linked using pointers, each element (node) contains a reference to the next and previous elements.
        - Implements the `List` and `Deque` interfaces.
    - Advantages:
        - Efficient insertions and deletions, especially at the beginning and end of the list.
        - No need to specify initial size.
        - Can be used as a stack, queue, or deque due to its `Deque` implementation.

Summary:

- Use **Arrays** when you need fixed-size collections with fast random access and minimal memory overhead.
- Use **ArrayList** when you need dynamic-size collections with fast random access and flexible size management.
- Use **LinkedList** when you need dynamic-size collections with efficient insertions and deletions, particularly at the ends of the list.

Question 3

a) Using Java, write down the parent class `Total` to contain the two data members: `quantity` and `unit_price`, constructor, set and get methods. (7 marks)

```java
class Total {
    private int quantity;
    private double unit_price;

    public Total(int quantity, double unit_price) {
        this.quantity = quantity;
        this.unit_price = unit_price;
    }

    public int getQuantity() {
        return quantity;
    }

    public void setQuantity(int quantity) {
        this.quantity = quantity;
    }

    public double getUnit_price() {
        return unit_price;
    }

    public void setUnit_price(double unit_price) {
        this.unit_price = unit_price;
    }
}
```

b) Write down the child class `Item` from the `Total` class. Add a method `Calculate` that multiplies `quantity` by `unit_price`, and display the result. (7 marks)

```
class Item extends Total {

    public Item(int quantity, double unit_price) {
        super(quantity, unit_price);
    }

    public void Calculate() {
        double total = getQuantity() * getUnit_price();
        System.out.println("Total: " + total);
    }
}
```

c) Write down a main method that creates an object of type `Item`, and calls the method `Calculate` to display the result of the product of `quantity` and `unit_price`. (7 marks)

```
public class Main {
    public static void main(String[] args) {
        Item item = new Item(10, 5.5);
        item.Calculate();
    }
}
```

d) Mention all the Object Oriented concepts you have used in the program and at which lines. (4 marks)

1. **Encapsulation:**
    - Lines 2-20: The `Total` class encapsulates the data members `quantity` and `unit_price` with private access and provides public getter and setter methods to access and modify these fields.
2. **Inheritance:**
    - Line 22: The `Item` class extends the `Total` class, inheriting its data members and methods.
3. **Polymorphism:**
    - Line 28: The `Item` class overrides the `Calculate` method to provide its own implementation, demonstrating polymorphism by extending the functionality of the `Total` class.
4. **Abstraction:**

- Throughout the program: The use of classes and methods to abstract the implementation details and provide a clear interface for interacting with objects.

```java
class Total { // Encapsulation (Lines 2-20)
    private int quantity;
    private double unit_price;

    public Total(int quantity, double unit_price) {
        this.quantity = quantity;
        this.unit_price = unit_price;
    }

    public int getQuantity() {
        return quantity;
    }

    public void setQuantity(int quantity) {
        this.quantity = quantity;
    }

    public double getUnit_price() {
        return unit_price;
    }

    public void setUnit_price(double unit_price) {
        this.unit_price = unit_price;
    }
}

class Item extends Total { // Inheritance (Line 22)

    public Item(int quantity, double unit_price) {
        super(quantity, unit_price);
    }

    public void Calculate() { // Polymorphism (Line 28)
        double total = getQuantity() * getUnit_price();
        System.out.println("Total: " + total);
    }
}

public class Main {
    public static void main(String[] args) {
        Item item = new Item(10, 5.5);
        item.Calculate();
    }
}
```

In this program:

- Encapsulation is used to protect the data members `quantity` and `unit_price` by making them private and providing public getter and setter methods.
- Inheritance is demonstrated by the `Item` class extending the `Total` class, inheriting its fields and methods.
- Polymorphism is shown by the `Calculate` method in the `Item` class, which overrides the functionality of the parent class.
- Abstraction is achieved by using classes and methods to hide the implementation details and expose a clear interface for object interaction.

Question 4

a) Give the six (6) steps to create a database connection and to perform a Select SQL statement in Java. (6 marks)

1. Load the JDBC Driver:

   ```
   Class.forName("com.mysql.cj.jdbc.Driver");
   ```

2. Establish the Connection:

   ```
   Connection connection = DriverManager.getConnection("jdbc:mysql://localhost:330
   ```

3. Create a Statement Object:

   ```
   Statement statement = connection.createStatement();
   ```

4. Execute the Query:

```
ResultSet resultSet = statement.executeQuery("SELECT * FROM table_name");
```

5. Process the Result Set:

```
while (resultSet.next()) {
    // Process each row
    System.out.println("Column1: " + resultSet.getString("column1"));
    // Add more columns as needed
}
```

6. Close the Connection:

```
resultSet.close();
statement.close();
connection.close();
```

b) Explain whether a database connection requires a runtime block and why. (4 marks)

Yes, a database connection typically requires a runtime block to handle potential exceptions that may occur during the connection process. For example, the `Class.forName`, `getConnection`, and `executeQuery` methods can throw `ClassNotFoundException` and `SQLException`. Using a try-catch block ensures that these exceptions are properly handled, preventing the program from crashing and allowing for appropriate error messages or actions to be taken.

Example:

```
try {
    Class.forName("com.mysql.cj.jdbc.Driver");
    Connection connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/datab
    Statement statement = connection.createStatement();
    ResultSet resultSet = statement.executeQuery("SELECT * FROM table_name");

    while (resultSet.next()) {
        System.out.println("Column1: " + resultSet.getString("column1"));
    }

    resultSet.close();
    statement.close();
    connection.close();
} catch (ClassNotFoundException | SQLException e) {
    e.printStackTrace();
}
```

c) Below is a sample Java program:

```java
private JButton btn = new JButton("Click Me!"); // Line 1
private int num = 0; // Line 2

public TestInteractiveButton(String s) // Line 3
{
    super(s); // Line 4
    setSize(400, 300); // Line 5
    setLocation(100, 200); // Line 6
    setLayout(new FlowLayout(FlowLayout.CENTER)); // Line 7
    btn.addActionListener(this); // Line 8
    add(btn); // Line 9
    setVisible(true); // Line 10
}

public void actionPerformed(ActionEvent e) // Line 11
{
    if(e.getSource().equals(btn)) // Line 12
    {
        num++; // Line 13
        System.out.println("You clicked me " + num + " times."); // Line 14
    }
}
```

i. Explain the lines 1, 3, 5, 6, 7, 8, 9, 10, 11, 13 and 15. (11 marks)

1. **Line 1:** Creates a new `JButton` with the label "Click Me!".
2. **Line 3:** Constructor for the `TestInteractiveButton` class, taking a string `s` as an argument.
3. **Line 5:** Sets the size of the window to 400 pixels wide by 300 pixels high.
4. **Line 6:** Sets the location of the window to 100 pixels from the left edge of the screen and 200 pixels from the top edge.
5. **Line 7:** Sets the layout manager for the window to `FlowLayout`, centered.
6. **Line 8:** Adds an action listener to the button `btn`, making the current instance (`this`) listen for button clicks.
7. **Line 9:** Adds the button `btn` to the window.
8. **Line 10:** Makes the window visible on the screen.
9. **Line 11:** Defines the `actionPerformed` method, which is called when an action event (like a button click) occurs.

10. **Line 13:** Increments the counter variable `num` by 1 each time the button is clicked.
11. **Line 15:** Prints the message "You clicked me `num` times." to the console, where `num` is the current count of button clicks.

ii. Give the missing lines or libraries required at the top of the program and explain their purpose. (4 marks)

Missing imports:

```java
import javax.swing.JButton;
import javax.swing.JFrame;
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
```

Explanation:

1. **`import javax.swing.JButton;`** - Imports the `JButton` class used to create button components.
2. **`import javax.swing.JFrame;`** - Imports the `JFrame` class used to create a window.
3. **`import java.awt.FlowLayout;`** - Imports the `FlowLayout` class used to set the layout manager of the window.
4. **`import java.awt.event.ActionEvent;`** - Imports the `ActionEvent` class used to handle action events.
5. **`import java.awt.event.ActionListener;`** - Imports the `ActionListener` interface, which is implemented to handle action events from the button.

Complete Program:

```
import javax.swing.JButton;
import javax.swing.JFrame;
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class TestInteractiveButton extends JFrame implements ActionListener {
    private JButton btn = new JButton("Click Me!"); // Line 1
    private int num = 0; // Line 2

    public TestInteractiveButton(String s) { // Line 3
        super(s); // Line 4
        setSize(400, 300); // Line 5
        setLocation(100, 200); // Line 6
        setLayout(new FlowLayout(FlowLayout.CENTER)); // Line 7
        btn.addActionListener(this); // Line 8
        add(btn); // Line 9
        setVisible(true); // Line 10
    }

    public void actionPerformed(ActionEvent e) { // Line 11
        if (e.getSource().equals(btn)) { // Line 12
            num++; // Line 13
            System.out.println("You clicked me " + num + " times."); // Line 14
        }
    }

    public static void main(String[] args) {
        new TestInteractiveButton("Interactive Button Test");
    }
}
```

In this complete program:

- The necessary imports are included at the top.
- The `TestInteractiveButton` class extends `JFrame` and implements `ActionListener`.
- The constructor initializes the window and adds the button with an action listener.
- The `actionPerformed` method handles button click events, increments a counter,

and prints the result to the console.

- The `main` method creates an instance of `TestInteractiveButton`, starting the application.
