Java Core Concepts and Examples

Practical 1. Java Basics and IDE Setup

a. Simple Java Program to Print "Hello, Java!"
b. Compile and Run Using Command-Line Tools

To compile the Java program:
> javac HelloJava.java

To run the compiled Java program:
> java HelloJava

Practical 2. Working with Classes and Objects

a. Student Class with Data Members and Methods

```
public class Student
{
   String name;
   int age;

   // Method to display student details
   void display()
        {
            System.out.println("Name: " + name);
            System.out.println("Age: " + age);
         }

           public static void main(String[] args)
        {
            Student s1 = new Student();
            s1.name = "Alice";
            s1.age = 20;
            s1.display();
          }
        }
```

b. Method Overloading using Calculator Class

```
public class Calculator {
    int add(int a, int b) {
        return a + b;
    }
    double add(double a, double b) {
        return a + b;
    }
    public static void main(String[] args) {
        Calculator calc = new Calculator();
        System.out.println("Sum (int): " + calc.add(10, 20));
        System.out.println("Sum (double): " + calc.add(10.5, 20.5));
    }
}
```

c. Using this Keyword and Constructors

```
public class StudentLifecycle {
    String name;
    int age;

    // Constructor using this keyword
    StudentLifecycle(String name, int age) {
        this.name = name;
        this.age = age;
    }

    void display() {
        System.out.println("Student Name: " + this.name);
        System.out.println("Student Age: " + this.age);
    }

    public static void main(String[] args) {
        StudentLifecycle s = new StudentLifecycle("Bob", 21);
        s.display();
    }
}
```

Practical 3. Constructors and Access Control

a. Default, Parameterized, and Copy Constructors

```
public class Student {
    String name;
    int age;

    // Default constructor
    Student() {
        name = "Default";
        age = 0;
    }

    // Parameterized constructor
    Student(String name, int age) {
        this.name = name;
        this.age = age;
    }

    // Copy constructor
    Student(Student s) {
        this.name = s.name;
        this.age = s.age;
    }

    void display() {
        System.out.println("Name: " + name + ", Age: " + age);
    }

    public static void main(String[] args) {
        Student s1 = new Student();
        Student s2 = new Student("Alice", 22);
        Student s3 = new Student(s2);
        s1.display();
        s2.display();
        s3.display();
    }
}
```

b. Constructor Overloading using Book Class

```
public class Book {
    String title;
    int pages;

    Book() {
        title = "Unknown";
        pages = 0;
    }

    Book(String title) {
        this.title = title;
        this.pages = 100;
    }

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

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

    public static void main(String[] args) {
        Book b1 = new Book();
        Book b2 = new Book("Java Basics");
        Book b3 = new Book("Advanced Java", 500);
        b1.display();
        b2.display();
        b3.display();
    }
}
```

c. Access Modifiers Example

```
public class AccessControl {
    public int publicVar = 1;
    private int privateVar = 2;
    protected int protectedVar = 3;
    int defaultVar = 4;

    public void display() {
        System.out.println("Public: " + publicVar);
        System.out.println("Private: " + privateVar);
        System.out.println("Protected: " + protectedVar);
        System.out.println("Default: " + defaultVar);
    }
}
```

d. Final, Static Members and Garbage Collection

```
public class GCExample {
   final int finalValue = 10;
   static int count = 0;

   GCExample() {
      count++;
   }

   protected void finalize() {
      System.out.println("Object is garbage collected");
   }

   public static void main(String[] args) {
      GCExample obj1 = new GCExample();
      GCExample obj2 = new GCExample();
      System.out.println("Count: " + GCExample.count);
      obj1 = null;
      obj2 = null;
      System.gc();
   }
}
```

Practical 4. Inheritance and Polymorphism
a. Single and Multilevel Inheritance

```
class Person {
   String name;
   void showPerson() {
      System.out.println("Person Name: " + name);
   }
}

class Employee extends Person {
   int empId;
   void showEmployee() {
      System.out.println("Employee ID: " + empId);
   }
}

class Manager extends Employee {
   String department;
   void showManager() {
      System.out.println("Department: " + department);
   }

   public static void main(String[] args) {
      Manager m = new Manager();
      m.name = "Alice";
      m.empId = 101;
      m.department = "HR";
      m.showPerson();
      m.showEmployee();
      m.showManager();
   }
}
```

b. Method Overriding and Dynamic Dispatch

```
class Animal {
    void sound() {
        System.out.println("Animal makes sound");
    }
}
class Dog extends Animal {
    void sound() {
        System.out.println("Dog barks");
    }
}
public class TestPolymorphism {
    public static void main(String[] args) {
        Animal a = new Dog(); // dynamic dispatch
        a.sound();
    }
}
```

c. super Keyword and instanceof

```
class Vehicle {
   int speed = 50;
}

class Car extends Vehicle {
   int speed = 100;

   void display() {
      System.out.println("Speed: " + super.speed);
   }

   public static void main(String[] args) {
      Vehicle v = new Car();
      if (v instanceof Car) {
         Car c = (Car) v;
         c.display();
      }
   }
}
```

Practical 5. Abstract Classes and Interfaces

a. Abstract Class Shape with Subclasses

```
abstract class Shape {
   abstract void draw();
}

class Circle extends Shape {
   void draw() {
      System.out.println("Drawing Circle");
   }
}

class Rectangle extends Shape {
   void draw() {
      System.out.println("Drawing Rectangle");
   }

   public static void main(String[] args) {
      Shape s1 = new Circle();
      Shape s2 = new Rectangle();
      s1.draw();
      s2.draw();
   }
}
```

b. Printable Interface with Document and Image Classes

```
interface Printable {
   void print();
}

class Document implements Printable {
   public void print() {
      System.out.println("Printing Document");
   }
}

class Image implements Printable {
   public void print() {
      System.out.println("Printing Image");
   }

   public static void main(String[] args) {
      Printable p1 = new Document();
      Printable p2 = new Image();
      p1.print();
      p2.print();
   }
}
```

c. Functional Interface and Default Methods

```
@FunctionalInterface
interface Calculator {
    int operation(int a, int b);
    default void show() {
        System.out.println("Functional Interface with default method");
    }
}

public class FunctionalDemo {
    public static void main(String[] args) {
        Calculator add = (a, b) -> a + b;
        System.out.println("Sum: " + add.operation(10, 20));
        add.show();
    }
}
```

d. Comparable and Comparator

```
import java.util.*;
class Student implements Comparable<Student> {
   int id;
   String name;
   Student(int id, String name) {
      this.id = id;
      this.name = name;
   }

   public int compareTo(Student s) {
      return this.id - s.id;
   }

   public String toString() {
      return id + " " + name;
   }
}

class NameComparator implements Comparator<Student> {
   public int compare(Student s1, Student s2) {
      return s1.name.compareTo(s2.name);
   }
}

public class TestSort {
   public static void main(String[] args) {
      ArrayList<Student> list = new ArrayList<>();
      list.add(new Student(3, "Zara"));
      list.add(new Student(1, "Bob"));
      list.add(new Student(2, "Alice"));

      Collections.sort(list);
      System.out.println("Sorted by ID: " + list);

      Collections.sort(list, new NameComparator());
      System.out.println("Sorted by Name: " + list);
   }
}
```

Practical 6. Exception Handling
a. Checked and Unchecked Exceptions

```
import java.io.*;

public class ExceptionDemo {
    public static void main(String[] args) {
        // Unchecked Exception
        int a = 10, b = 0;
        try {
            int c = a / b;
        } catch (ArithmeticException e) {
            System.out.println("Division by zero: " + e);
        }
        // Checked Exception
        try {
            FileReader fr = new FileReader("file.txt");
        } catch (FileNotFoundException e) {
            System.out.println("File not found: " + e);
        }
    }
}
```

b. try, catch, finally, throw, and throws

```
public class TryCatchFinally {
   static void divide(int a, int b) throws ArithmeticException {
      if (b == 0)
         throw new ArithmeticException("Cannot divide by zero");
      else
         System.out.println("Result: " + (a / b));
   }
   public static void main(String[] args) {
      try {
         divide(10, 0);
      } catch (ArithmeticException e) {
         System.out.println("Exception caught: " + e);
      } finally {
         System.out.println("Finally block executed");
      }
   }
}
```

c. User-defined Exception

```
class InvalidAgeException extends Exception {
   InvalidAgeException(String s) {
      super(s);
   }
}
public class CustomExceptionDemo {
   static void validate(int age) throws InvalidAgeException {
      if (age < 18)
         throw new InvalidAgeException("Age is less than 18");
      else
         System.out.println("Valid age");
   }

   public static void main(String[] args) {
      try {
         validate(16);
      } catch (InvalidAgeException e) {
         System.out.println("Caught: " + e.getMessage());
      }
   }
}
```

Practical 7. Multithreading and Synchronization

a. Thread using Thread and Runnable

```
class MyThread extends Thread {
   public void run() {
      System.out.println("Thread using Thread class");
   }
}

class MyRunnable implements Runnable {
   public void run() {
      System.out.println("Thread using Runnable interface");
   }
}

public class ThreadDemo {
   public static void main(String[] args) {
      MyThread t1 = new MyThread();
      t1.start();
      Thread t2 = new Thread(new MyRunnable());
      t2.start();
   }
}
```

b. Thread Priority and Sleep

```
public class PrioritySleepDemo extends Thread {
   public void run() {
      System.out.println("Running thread: " + Thread.currentThread().getName());
   }

   public static void main(String[] args) throws InterruptedException {
      PrioritySleepDemo t1 = new PrioritySleepDemo();
      PrioritySleepDemo t2 = new PrioritySleepDemo();

      t1.setPriority(Thread.MIN_PRIORITY);
      t2.setPriority(Thread.MAX_PRIORITY);

      t1.start();
      Thread.sleep(500);
      t2.start();
   }
}
```

c. Synchronization

```
class Table {
   synchronized void printTable(int n) {
      for (int i = 1; i <= 5; i++) {
         System.out.println(n * i);
      }
   }
}
class MyThread1 extends Thread {
   Table t;
   MyThread1(Table t) { this.t = t; }
   public void run() { t.printTable(5); }
}
class MyThread2 extends Thread {

   Table t;

   MyThread2(Table t) { this.t = t; }

   public void run() { t.printTable(100); }

}

public class SyncExample {

   public static void main(String[] args) {

      Table obj = new Table();

      new MyThread1(obj).start();

      new MyThread2(obj).start();

   }

}
```

d. Inter-thread Communication (wait() / notify())

```
class SharedResource {
   boolean flag = false;
   synchronized void produce() throws InterruptedException {
      if (flag) wait();
      System.out.println("Produced");
      flag = true;
      notify();
   }
   synchronized void consume() throws InterruptedException {
      if (!flag) wait();
      System.out.println("Consumed");
      flag = false;
      notify();
   }
}
public class WaitNotifyDemo {
   public static void main(String[] args) {
      SharedResource s = new SharedResource();
      new Thread(() -> {
         try {
            while (true) s.produce();
         } catch (Exception e) {}
      }).start();

      new Thread(() -> {
         try {
            while (true) s.consume();
         } catch (Exception e) {}
      }).start();
   }
}
```

Practical 8. Swing Basics and Layouts

a. Simple Login Form using JFrame

```
import javax.swing.*;
public class LoginForm {
   public static void main(String[] args) {
      JFrame frame = new JFrame("Login Form");
      JLabel userLabel = new JLabel("Username:");
      JTextField userText = new JTextField();
      JLabel passLabel = new JLabel("Password:");
      JPasswordField passText = new JPasswordField();
      JButton loginButton = new JButton("Login");

      frame.setSize(300, 200);
      frame.setLayout(null);

      userLabel.setBounds(20, 20, 80, 25);
      userText.setBounds(100, 20, 165, 25);
      passLabel.setBounds(20, 60, 80, 25);
      passText.setBounds(100, 60, 165, 25);
      loginButton.setBounds(100, 100, 80, 25);

      frame.add(userLabel);
      frame.add(userText);
      frame.add(passLabel);
      frame.add(passText);
      frame.add(loginButton);

      frame.setVisible(true);
   }
}
```

b. Layout Managers: FlowLayout & BorderLayout

```
import javax.swing.*;
import java.awt.*;
public class LayoutExample {
   public static void main(String[] args) {
      JFrame frame = new JFrame("FlowLayout Example");
      frame.setLayout(new FlowLayout());
      frame.add(new JButton("Button 1"));
      frame.add(new JButton("Button 2"));
      frame.setSize(300, 100);
      frame.setVisible(true);
      JFrame borderFrame = new JFrame("BorderLayout Example");
      borderFrame.setLayout(new BorderLayout());
      borderFrame.add(new JButton("North"), BorderLayout.NORTH);
      borderFrame.add(new JButton("South"), BorderLayout.SOUTH);
      borderFrame.add(new JButton("East"), BorderLayout.EAST);
      borderFrame.add(new JButton("West"), BorderLayout.WEST);
      borderFrame.add(new JButton("Center"), BorderLayout.CENTER);
      borderFrame.setSize(300, 200);
      borderFrame.setVisible(true);
   }
}
```

c. Input Validation with JOptionPane

```
import javax.swing.*;
import java.awt.event.*;

public class ValidationExample {
   public static void main(String[] args) {
      JFrame frame = new JFrame("Validation");
      JTextField tf = new JTextField();
      JButton btn = new JButton("Validate");

      tf.setBounds(50, 50, 150, 20);
      btn.setBounds(50, 100, 100, 30);

      btn.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            String input = tf.getText();
            if (input.isEmpty()) {
               JOptionPane.showMessageDialog(frame, "Field cannot be empty");
            } else {
               JOptionPane.showMessageDialog(frame, "Valid input: " + input);
            }
         }
      });

      frame.add(tf);
      frame.add(btn);
      frame.setSize(300, 200);
      frame.setLayout(null);
      frame.setVisible(true);
   }
}
```

9. Event Handling in GUI

a. ActionListener for Button Click

```
import javax.swing.*;
import java.awt.event.*;
public class ButtonClickExample {
   public static void main(String[] args) {
      JFrame frame = new JFrame("Action Listener Example");
      JButton btn = new JButton("Click Me");
      btn.setBounds(100, 100, 120, 30);
      btn.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            JOptionPane.showMessageDialog(frame, "Button Clicked!");
         }
      });
      frame.add(btn);
      frame.setSize(300, 200);
      frame.setLayout(null);
      frame.setVisible(true);
   }
}
```

b. ItemListener for Checkboxes and Radio Buttons

```
import javax.swing.*;
import java.awt.event.*;

public class ItemListenerExample {
   public static void main(String[] args) {
      JFrame frame = new JFrame("ItemListener Example");

      JCheckBox cb = new JCheckBox("Java");
      JRadioButton rb1 = new JRadioButton("Male");
      JRadioButton rb2 = new JRadioButton("Female");

      cb.setBounds(50, 50, 100, 30);
      rb1.setBounds(50, 100, 100, 30);
      rb2.setBounds(150, 100, 100, 30);

      ButtonGroup group = new ButtonGroup();
      group.add(rb1);
      group.add(rb2);

      cb.addItemListener(e -> JOptionPane.showMessageDialog(frame,
            cb.isSelected() ? "Java Selected" : "Java Unselected"));

      rb1.addItemListener(e -> {
         if (rb1.isSelected()) JOptionPane.showMessageDialog(frame, "Male Selected");
      });

      rb2.addItemListener(e -> {
         if (rb2.isSelected()) JOptionPane.showMessageDialog(frame, "Female
Selected");
      });

      frame.add(cb);
      frame.add(rb1);
      frame.add(rb2);
      frame.setSize(300, 200);
      frame.setLayout(null);
      frame.setVisible(true);
   }
}
```

c. MouseListener Example

```
import javax.swing.*;
import java.awt.event.*;

public class MouseListenerExample {
   public static void main(String[] args) {
      JFrame frame = new JFrame("Mouse Listener Example");
      JLabel label = new JLabel("Click anywhere");

      label.setBounds(50, 50, 200, 30);
      frame.addMouseListener(new MouseAdapter() {
         public void mouseClicked(MouseEvent e) {
            label.setText("Mouse Clicked at X: " + e.getX() + ", Y: " + e.getY());
         }
      });

      frame.add(label);
      frame.setSize(300, 200);
      frame.setLayout(null);
      frame.setVisible(true);
   }
}
```

d. Inner & Anonymous Classes for Events

```
import javax.swing.*;
import java.awt.event.*;

public class InnerClassEvent {
   public static void main(String[] args) {
      JFrame frame = new JFrame("Anonymous Class Example");
      JButton btn = new JButton("Click");

      btn.setBounds(100, 80, 100, 30);

      btn.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            JOptionPane.showMessageDialog(frame, "Handled by anonymous class");
         }
      });

      frame.add(btn);
      frame.setSize(300, 200);
      frame.setLayout(null);
      frame.setVisible(true);
   }
}
```

Practical 10. JDBC and Advanced Swing Integration

⚠️ *Make sure you have MySQL or SQLite JDBC driver added to your classpath.*

a. JDBC Connection Example (MySQL)

```
import java.sql.*;
public class DBConnectionExample {
   public static void main(String[] args) {
      try {
         Class.forName("com.mysql.cj.jdbc.Driver"); // MySQL Driver
         Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/test", "root", "password");
         System.out.println("Connected to Database");
         con.close();
      } catch (Exception e) {
         e.printStackTrace();
      }
   }
}
```

b. CRUD Operations using PreparedStatement

```
import java.sql.*;
public class CRUDExample {
   public static void main(String[] args) {
      try {
         Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/test", "root", "password");

         // INSERT
         PreparedStatement ps = con.prepareStatement("INSERT INTO student (name,
age) VALUES (?, ?)");
         ps.setString(1, "John");
         ps.setInt(2, 20);
         ps.executeUpdate();

         // SELECT
         ResultSet rs = con.createStatement().executeQuery("SELECT * FROM
student");
         while (rs.next()) {
            System.out.println(rs.getInt(1) + " " + rs.getString(2));
         }

         con.close();
      } catch (Exception e) {
         e.printStackTrace();
      }
   }
}
```

c. Display Records in JTable

```
import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.sql.*;

public class TableDisplay {
   public static void main(String[] args) {
      JFrame frame = new JFrame("Student Records");
      DefaultTableModel model = new DefaultTableModel();
      JTable table = new JTable(model);
      model.addColumn("ID");
      model.addColumn("Name");
      model.addColumn("Age");

      try {
         Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/test", "root", "password");
         Statement stmt = con.createStatement();
         ResultSet rs = stmt.executeQuery("SELECT * FROM student");

         while (rs.next()) {
            model.addRow(new Object[]{rs.getInt("id"), rs.getString("name"),
rs.getInt("age")});
         }

         con.close();
      } catch (Exception e) {
         e.printStackTrace();
      }

      JScrollPane sp = new JScrollPane(table);
      frame.add(sp);
      frame.setSize(400, 300);
      frame.setVisible(true);
   }
}
```

d. GUI + JDBC Integration (Student Registration)

```
import javax.swing.*;
import java.awt.event.*;
import java.sql.*;

public class StudentForm {
   public static void main(String[] args) {
      JFrame frame = new JFrame("Student Registration");

      JLabel nameLabel = new JLabel("Name:");
      JTextField nameField = new JTextField();
      JLabel ageLabel = new JLabel("Age:");
      JTextField ageField = new JTextField();
      JButton submitBtn = new JButton("Submit");

      nameLabel.setBounds(20, 20, 100, 30);
      nameField.setBounds(100, 20, 150, 30);
      ageLabel.setBounds(20, 60, 100, 30);
      ageField.setBounds(100, 60, 150, 30);
      submitBtn.setBounds(100, 100, 100, 30);

      submitBtn.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            try {
               String name = nameField.getText();
               int age = Integer.parseInt(ageField.getText());

               Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/test", "root", "password");
               PreparedStatement ps = con.prepareStatement("INSERT INTO student
(name, age) VALUES (?, ?)");
               ps.setString(1, name);
               ps.setInt(2, age);
               ps.executeUpdate();

               JOptionPane.showMessageDialog(frame, "Student Registered!");
               con.close();
            } catch (Exception ex) {
               ex.printStackTrace();
            }
```

```
            }
        });

        frame.setLayout(null);
        frame.setSize(300, 200);
        frame.add(nameLabel);
        frame.add(nameField);
        frame.add(ageLabel);
        frame.add(ageField);
        frame.add(submitBtn);
        frame.setVisible(true);
    }
}
```