by Kunal Sir

Assignment : Method Overriding (Run-Time Polymorphism)

Part A: Multiple Choice Questions (MCQs)

- Method overriding is also known as:
  a) Compile time polymorphism
  b) Static binding
  c) Run time polymorphism
  d) Method overloading

  Ans : c) Run time polymorphism

- Method overriding occurs between:
  a) Two unrelated classes
  b) Same class
  c) Parent and child class
  d) Interfaces only

  Ans : c) Parent and child class

- Which of the following is mandatory for method overriding?
  a) Same method name only
  b) Same method name and parameters
  c) Same return type only
  d) Same access modifier only

  Ans : b) Same method name and parameters

- Which method **cannot** be overridden in Java?
  a) public method
  b) protected method
  c) final method
  d) default method

  Ans : c) final method

- Why static methods cannot be overridden?
  a) They are slow
  b) They use dynamic binding
  c) They use static binding
  d) They are private

  Ans : c) They use static binding

- Which annotation is used to ensure a method is overridden correctly?
  a) @FunctionalInterface
  b) @Override
  c) @Inherited
  d) @SuppressWarnings

  Ans : b) @Override

- Access modifier of an overridden method should be:
  a) Weaker than parent
  b) Same or stronger than parent
  c) Always public
  d) Always private

  Ans : b) Same or stronger than parent

- Which return type rule is correct in method overriding?
  a) Must always be void
  b) Must be different
  c) Must be same or subclass type
  d) Can be anything

  Ans : c) Must be same or subclass type

- Private methods cannot be overridden because:
  a) They are slow
  b) They are final
  c) They are not visible to child class
  d) They use static binding.

  Ans : c) They are not visible to child class

- Method overriding provides:
  a) Code duplication
  b) Code hiding
  c) Additional meaning to existing behavior
  d) Multiple inheritance

  Ans : c) Additional meaning to existing behavior

- Which binding is used in method overriding?
  a) Static binding
  b) Early binding
  c) Dynamic binding
  d) Manual binding

  Ans : c) Dynamic binding

- What will be called at runtime?
  a) Parent class method
  b) Child class method
  c) Both methods
  d) Depends on reference variable

  Ans : b) Child class method

- Which keyword prevents method overriding?
  a) static
  b) private
  c) final
  d) abstract

  Ans : c) final

- Method overriding supports which OOP concept?
  a) Encapsulation
  b) Inheritance
  c) Polymorphism
  d) Abstraction

  Ans : c) Polymorphism

- Which version of Java introduced @Override annotation?
  a) Java 1.2
  b) Java 1.4
  c) Java 1.5
  d) Java 1.8

  Ans : c) Java 1.5

Part B: Problem Statements

Problem 1: Bank Interest Calculation

Description: A bank provides interest on deposited money using a general interest calculation method. However, a **Savings Bank account** offers a higher interest rate compared to a normal bank account. Since the savings bank is not satisfied with the parent bank's interest logic, it provides its own implementation.

Task: Override the `calculateInterest()` method in the `SavingsBank` class.

Sample Input: Amount = 10000

Sample Output: Savings Bank Interest: 1200


```
class Bank {
    void calculateInterest() {
        System.out.println("Bank Interest: 800");
    }
}

class SavingsBank extends Bank {
    @Override
    void calculateInterest() {
        System.out.println("Savings Bank Interest: 1200");
    }
}
```

Problem 2: Vehicle Speed

Description: A general vehicle class displays an average speed. A **Bike** is a specific type of vehicle that runs faster than the default speed defined in the vehicle class. Therefore, the bike class overrides the speed display method.

Task: Override the `showSpeed()` method in the `Bike` class.

Sample Input: No input required

Sample Output: Bike speed is 80 km/h

```
class Vehicle {
    void showSpeed() {
        System.out.println("Vehicle speed is 50 km/h");
    }
}

class Bike extends Vehicle {
    @Override
    void showSpeed() {
        System.out.println("Bike speed is 80 km/h");
    }
}
```

Problem 3: Employee Salary Calculation

Description: In a company, an employee receives a basic salary. A **Manager** is also an employee but gets additional benefits and incentives. Hence, the salary calculation for a manager is different from a normal employee.

Task: Override the `calculateSalary()` method in the `Manager` class.

Sample Input: Basic Salary = 30000

Sample Output: Manager Salary: 45000

```
class Employee {
    void calculateSalary() {
        System.out.println("Employee Salary: 30000");
    }
}

class Manager extends Employee {
    @Override
    void calculateSalary() {
        System.out.println("Manager Salary: 45000");
    }
}
```

Problem 4: Online Payment System

Description: An online payment system processes payments in a standard way. When the payment is done using a **Credit Card**, extra service charges are applied. The credit card payment method modifies the original payment behavior.

Task: Override the `processPayment()` method in the `CreditCardPayment` class.

Sample Input: Amount = 2000

Sample Output: Credit Card Payment Processed: 2100

```
class Payment {
    void processPayment() {
        System.out.println("Payment Processed: 2000");
    }
}

class CreditCardPayment extends Payment {
    @Override
    void processPayment() {
        System.out.println("Credit Card Payment Processed: 2100");
    }
}
```

Problem 5: Shape Area Calculation

Description: A shape class contains a general method to calculate area. A **Rectangle** is a specific shape and has its own formula to calculate area using length and breadth.

Task: Override the `calculateArea()` method in the `Rectangle` class.

Sample Input: Length = 5, Breadth = 4

Sample Output: Area of Rectangle: 20

```
class Shape {
    void calculateArea() {
        System.out.println("Area not defined");
    }
}

class Rectangle extends Shape {
    @Override
    void calculateArea() {
        System.out.println("Area of Rectangle: 20");
    }
}
```

Problem 6: Mobile Notification System

Description: A mobile phone receives notifications in a general way. A **WhatsApp application** customizes how notifications are shown to the user. Therefore, it changes the default notification behavior.

Task: Override the `sendNotification()` method in the `WhatsApp` class.

Sample Input: No input required

Sample Output: WhatsApp Notification Received

```
class Mobile {
    void sendNotification() {
        System.out.println("General Notification");
    }
}

class WhatsApp extends Mobile {
    @Override
    void sendNotification() {
        System.out.println("WhatsApp Notification Received");
    }
}
```

Problem 7: Food Ordering System

Description: A restaurant prepares food for customers who visit the restaurant directly. When food is ordered online, the preparation process includes packing and delivery instructions. Hence, the online order process differs from normal food preparation.

Task: Override the `prepareFood()` method in the `OnlineOrder` class.

Sample Input: No input required

Sample Output: Food prepared for online delivery

```
class Restaurant {
    void prepareFood() {
        System.out.println("Food prepared for dine-in");
    }
}

class OnlineOrder extends Restaurant {
    @Override
    void prepareFood() {
        System.out.println("Food prepared for online delivery");
    }
}
```

Problem 8: Examination Result Display

Description: An examination system displays results in a standard format. A **Practical Exam** displays results differently by including practical marks and remarks.

Task: Override the `displayResult()` method in the `PracticalExam` class.

Sample Input: No input required

Sample Output: Practical Exam Result Displayed

```
class Exam {
    void displayResult() {
        System.out.println("Exam Result Displayed");
    }
}

class PracticalExam extends Exam {
    @Override
    void displayResult() {
        System.out.println("Practical Exam Result Displayed");
    }
}
```

Problem 9: Login System

Description: A system allows users to log in using a username and password. An **Admin** user requires additional security, such as OTP verification, during login.

Task: Override the `login()` method in the `AdminLogin` class.

Sample Input: No input required

Sample Output: Admin Login with OTP verification

```
class UserLogin {
    void login() {
        System.out.println("User Login Successful");
    }
}

class AdminLogin extends UserLogin {
    @Override
    void login() {
        System.out.println("Admin Login with OTP verification");
    }
}
```

Problem 10: Transport Fare Calculation

Description: A transport service calculates fare based on distance. A **Cab service** adds extra convenience charges such as booking fees and comfort charges, so its fare calculation is different.

Task: Override the `calculateFare()` method in the `Cab` class.

Sample Input: Distance = 10 km

Sample Output: Cab Fare: 250

```
class Transport {
    void calculateFare() {
        System.out.println("Fare: 200");
    }
}

class Cab extends Transport {
    @Override
    void calculateFare() {
        System.out.println("Cab Fare: 250");
    }
}
```
