Object Class Methods, Inner Classes Practice Problem
Object Modeling & toString()

Problem: "Vehicle Rental System"

```
class Vehicle {
    private String registrationNo;
    private String type;
    private double ratePerDay;

    // Constructor initializing all fields
    public Vehicle(String registrationNo, String type, double ratePerDay) {
        this.registrationNo = registrationNo;
        this.type = type;
        this.ratePerDay = ratePerDay;
    }

    // Getters
    public String getRegistrationNo() {
        return registrationNo;
    }

    public String getType() {
        return type;
    }

    public double getRatePerDay() {
        return ratePerDay;
    }

    // Overriding toString() method
    @Override
    public String toString() {
        return "Vehicle: " + registrationNo + ", Type: " + type + ", Rate:
$" + ratePerDay + "/day";
    }
}

public class VehicleRental {
    public static void main(String[] args) {
        // 1. Create Vehicle objects
        Vehicle v1 = new Vehicle("MH12AB1234", "Sedan", 1500);
        Vehicle v2 = new Vehicle("MH14XY5678", "SUV", 2000);
```

```
        // 2. Print the Vehicle objects
        System.out.println(v1);
        System.out.println(v2);

        // 3. Compare (example: by registration number)
        if (v1.getRegistrationNo().equals(v2.getRegistrationNo())) {
            System.out.println("Both vehicles are the same.");
        } else {
            System.out.println("Vehicles are different.");
        }
    }
}
```

Vehicle: MH12AB1234, Type: Sedan, Rate: $1500.0/day
Vehicle: MH14XY5678, Type: SUV, Rate: $2000.0/day
Vehicles are different.

---

equals(),
==
, and hashCode()
Problem: "Employee Authentication System"

```
import java.util.HashSet;

class Employee {
    private String empCode;
    private String name;

    public Employee(String empCode, String name) {
        this.empCode = empCode;
        this.name = name;
    }

    // Override equals() - same empCode means same employee
    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true;                // same object
        if (obj == null || getClass() != obj.getClass()) return false;
        Employee other = (Employee) obj;
```

```
        return empCode.equals(other.empCode);         // same empCode →
same employee
    }

    // Override hashCode() based on empCode
    @Override
    public int hashCode() {
        return empCode.hashCode();
    }

    // toString() showing both fields
    @Override
    public String toString() {
        return "Employee Code: " + empCode + ", Name: " + name;
    }
}

public class EmployeeAuth {
    public static void main(String[] args) {
        // 1. Create Employee objects
        Employee e1 = new Employee("BL001", "Ritika");
        Employee e2 = new Employee("BL001", "Ritika S.");

        // 2. Compare using == and equals()
        System.out.println("e1 == e2 ? " + (e1 == e2));
        System.out.println("e1.equals(e2) ? " + e1.equals(e2));

        // 3. Print both employees
        System.out.println(e1);
        System.out.println(e2);

        // 4. Test using HashSet<Employee>
        HashSet<Employee> set = new HashSet<>();
        set.add(e1);
        set.add(e2);

        System.out.println("HashSet contents: " + set);
        System.out.println("HashSet size: " + set.size());
    }
}
```

e1 == e2 ? false

e1.equals(e2) ? true
Employee Code: BL001, Name: Ritika
Employee Code: BL001, Name: Ritika S.
HashSet contents: [Employee Code: BL001, Name: Ritika]
HashSet size: 1

---

getClass()
Problem: "Payment Gateway"

```
class Payment {
    public void pay() {
        System.out.println("Generic payment");
    }
}

class CreditCardPayment extends Payment {
    @Override
    public void pay() {
        System.out.println("Payment done using Credit Card");
    }
}

class WalletPayment extends Payment {
    @Override
    public void pay() {
        System.out.println("Payment done using Wallet");
    }
}

public class PaymentGateway {
    public static void main(String[] args) {
        // 1. Create array of Payment references
        Payment[] payments = {
            new CreditCardPayment(),
            new WalletPayment(),
            new Payment()
        };

        // 2. Loop, display class name, and call pay()
```

```
        for (Payment p : payments) {
            System.out.println("Processing: " +
p.getClass().getSimpleName());
            p.pay();
            System.out.println("----------------------");
        }
    }
}
```

Processing: CreditCardPayment
Payment done using Credit Card

Processing: WalletPayment
Payment done using Wallet

Processing: Payment
Generic payment

---

clone(), Shallow vs Deep Copy
Problem: "Course Registration System"

```
class ContactInfo implements Cloneable {
    private String email;
    private String phone;

    public ContactInfo(String email, String phone) {
        this.email = email;
        this.phone = phone;
    }

    public String getEmail() { return email; }
    public String getPhone() { return phone; }
    public void setEmail(String email) { this.email = email; }
    public void setPhone(String phone) { this.phone = phone; }

    @Override
    public String toString() {
        return "Email: " + email + ", Phone: " + phone;
```

```
    }

    @Override
    protected Object clone() throws CloneNotSupportedException {
        return super.clone(); // default shallow clone
    }
}

class Student implements Cloneable {
    private String id;
    private String name;
    private ContactInfo contact;

    public Student(String id, String name, ContactInfo contact) {
        this.id = id;
        this.name = name;
        this.contact = contact;
    }

    public ContactInfo getContact() { return contact; }

    // Shallow copy (copies reference)
    protected Student shallowCopy() throws CloneNotSupportedException {
        return (Student) super.clone();
    }

    // Deep copy (creates new ContactInfo clone)
    protected Student deepCopy() throws CloneNotSupportedException {
        Student cloned = (Student) super.clone();
        cloned.contact = (ContactInfo) contact.clone(); // separate object
        return cloned;
    }

    @Override
    public String toString() {
        return "ID: " + id + ", Name: " + name + ", " + contact;
    }
}

public class Registration {
    public static void main(String[] args) throws
CloneNotSupportedException {
        ContactInfo contact = new ContactInfo("ritika@example.com",
```

```
"9876543210");
        Student s1 = new Student("S101", "Ritika", contact);

        // Shallow clone
        Student shallow = s1.shallowCopy();

        // Deep clone
        Student deep = s1.deepCopy();

        System.out.println("Before modification:");
        System.out.println("Original: " + s1);
        System.out.println("Shallow:  " + shallow);
        System.out.println("Deep:     " + deep);

        // Modify contact info
        s1.getContact().setEmail("updated@mail.com");
        s1.getContact().setPhone("9999999999");

        System.out.println("\nAfter modifying original contact:");
        System.out.println("Original: " + s1);
        System.out.println("Shallow:  " + shallow);
        System.out.println("Deep:     " + deep);
    }
}
```

Before modification:
Original: ID: S101, Name: Ritika, Email: ritika@example.com, Phone: 9876543210
Shallow: ID: S101, Name: Ritika, Email: ritika@example.com, Phone: 9876543210
Deep: ID: S101, Name: Ritika, Email: ritika@example.com, Phone: 9876543210

After modifying original contact:
Original: ID: S101, Name: Ritika, Email: updated@mail.com, Phone: 9999999999
Shallow: ID: S101, Name: Ritika, Email: updated@mail.com, Phone: 9999999999
Deep: ID: S101, Name: Ritika, Email: ritika@example.com, Phone: 9876543210

Member Inner Class
Problem: "Hospital Management"

```
class Hospital {
    private String name;
```

```
    public Hospital(String name) {
        this.name = name;
    }

    // Inner class Department
    public class Department {
        private String deptName;

        public Department(String deptName) {
            this.deptName = deptName;
        }

        // Display department info along with hospital name
        public void showInfo() {
            System.out.println("Hospital: " + name + ", Department: " +
deptName);
        }
    }

    // Method to create Department object
    public Department createDepartment(String deptName) {
        return new Department(deptName);
    }
}

public class HospitalManagement {
    public static void main(String[] args) {
        // 1. Create Hospital with 2 Departments
        Hospital hospital = new Hospital("CityCare Hospital");

        Hospital.Department dept1 =
hospital.createDepartment("Cardiology");
        Hospital.Department dept2 = hospital.createDepartment("Neurology");

        // Display department info
        dept1.showInfo();
        dept2.showInfo();
    }
}
```

Hospital: CityCare Hospital, Department: Cardiology
Hospital: CityCare Hospital, Department: Neurology

Static Nested Class
Problem: "App Configuration"

```
class AppConfig {
    private String appName;

    public AppConfig(String appName) {
        this.appName = appName;
    }

    // Static nested class
    public static class NetworkConfig {
        private String host;
        private int port;

        public NetworkConfig(String host, int port) {
            this.host = host;
            this.port = port;
        }

        // Display network config
        public void showConfig() {
            System.out.println("Network Configuration:");
            System.out.println("Host: " + host);
            System.out.println("Port: " + port);
        }
    }
}

public class AppConfigurator {
    public static void main(String[] args) {
        // Create instance of static nested class without outer class
instance
        AppConfig.NetworkConfig netConfig =
            new AppConfig.NetworkConfig("192.168.1.10", 8080);

        // Print details
        netConfig.showConfig();
    }
}
```

Network Configuration:
Host: 192.168.1.10
Port: 8080

Local Inner Class
Problem: "Voting System"
// File: VotingSystem.

```
public class VotingSystem {

    public void processVote(String voterId, String candidate) {

        // Local Inner Class defined inside a method
        class VoteValidator {
            public boolean validate(String id) {
                // Example rule: voterId must start with "V" and be 5
characters long
                return id != null && id.startsWith("V") && id.length() ==
5;
            }
        }

        // Create instance of the local inner class
        VoteValidator validator = new VoteValidator();

        // Validate and print result
        if (validator.validate(voterId)) {
            System.out.println("Vote accepted for voter ID " + voterId +
                               " -> Candidate: " + candidate);
        } else {
            System.out.println("Invalid voter ID: " + voterId + ". Vote
rejected.");
        }
    }

    public static void main(String[] args) {
        VotingSystem system = new VotingSystem();

        // Test various voter IDs
        system.processVote("V1234", "Alice");
```

```
        system.processVote("12345", "Bob");
        system.processVote("V5678", "Charlie");
        system.processVote(null, "David");
    }
}
```

Vote accepted for voter ID V1234 -> Candidate: Alice
Invalid voter ID: 12345. Vote rejected.
Vote accepted for voter ID V5678 -> Candidate: Charlie
Invalid voter ID: null. Vote rejected.

Anonymous Inner Class
Problem: "Notification Service"

```
interface Notifier {
    void send(String message);
}

public class Service {
    public void triggerAlert() {
        // Anonymous inner class implementing Notifier
        Notifier notifier = new Notifier() {
            @Override
            public void send(String message) {
                System.out.println("🔔 Alert: " + message);
            }
        };

        // Use the anonymous inner class to send an alert
        notifier.send("System overload detected! Please check
immediately.");
    }

    public static void main(String[] args) {
        new Service().triggerAlert();
    }
}
```

🔔 Alert: System overload detected! Please check immediately.