

**Topic 1: Using toString() and getClass()**

**Problem Statement 1:**

Create a class Employee with fields id, name, and salary. Override the toString() method to print employee details in a readable format. In the main method, create multiple Employee objects and print their class name using getClass().getName().

**Hints:**

● Override toString() to provide a meaningful string representation. ● Use getClass() to obtain runtime class information.

● Display both the object details and its class name.

| class Employee {  int id;  String name;  double salary;   Employee(int id, String name, double salary) {  this.id = id;  this.name = name;  this.salary = salary;  }   public String toString() {  return "Employee ID: " + id + ", Name: " + name + ", Salary: " + salary;  } }  public class EmployeeDemo {  public static void main(String[] args) {  Employee e1 = new Employee(101, "John", 50000);  Employee e2 = new Employee(102, "Alice", 60000);  Employee e3 = new Employee(103, "Bob", 55000);   System.out.println(e1);  System.out.println("Class: " + e1.getClass().getName());  System.out.println(e2);  System.out.println("Class: " + e2.getClass().getName());  System.out.println(e3);  System.out.println("Class: " + e3.getClass().getName());  } } |
| --- |

**Topic 2: equals() vs ==**

**Problem Statement 2:**

Create a class Product with productId and productName fields. Compare two Product objects using both == and .equals() to demonstrate the difference between reference and content comparison. Override the equals() method to compare objects by productId.

**Hints:**

● == checks reference equality, .equals() checks logical equality.

● Override equals() properly using the @Override annotation.

● Print results of both comparisons for clarity.

| class Product {  int productId;  String productName;   Product(int productId, String productName) {  this.productId = productId;  this.productName = productName;  }   @Override  public boolean equals(Object obj) {  if (this == obj)  return true;  if (obj == null || getClass() != obj.getClass())  return false;  Product p = (Product) obj;  return productId == p.productId;  } }  public class ProductComparison {  public static void main(String[] args) {  Product p1 = new Product(101, "Laptop");  Product p2 = new Product(101, "Laptop");  Product p3 = p1;   System.out.println(p1 == p2);  System.out.println(p1.equals(p2));  System.out.println(p1 == p3);  System.out.println(p1.equals(p3));  } } |
| --- |

1



**Topic 3: hashCode() and equals() Contract**

**Problem Statement 3:**

Create a Student class with rollNo and name fields. Override both equals() and hashCode() so that two students with the same roll number are considered equal. Demonstrate how these methods affect object storage in a HashSet.

**Hints:**

● Use Objects.hash() to generate hash codes.

● Ensure equals() and hashCode() produce consistent results.

● Add duplicate objects to a HashSet and observe the output.

| import java.util.\*;  class Student {  int rollNo;  String name;   Student(int rollNo, String name) {  this.rollNo = rollNo;  this.name = name;  }   @Override  public boolean equals(Object obj) {  if (this == obj)  return true;  if (obj == null || getClass() != obj.getClass())  return false;  Student s = (Student) obj;  return rollNo == s.rollNo;  }   @Override  public int hashCode() {  return Objects.hash(rollNo);  }   public String toString() {  return "Student[RollNo=" + rollNo + ", Name=" + name + "]";  } }  public class StudentHashSetDemo {  public static void main(String[] args) {  HashSet<Student> set = new HashSet<>();  Student s1 = new Student(1, "John");  Student s2 = new Student(2, "Alice");  Student s3 = new Student(1, "John");   set.add(s1);  set.add(s2);  set.add(s3);   for (Student s : set)  System.out.println(s);  } } |
| --- |

**Topic 4: Deep vs Shallow Cloning of Objects**

**Problem Statement 4:**

Create a class Library containing a list of Book objects. Implement cloning such that shallow cloning only copies object references while deep cloning copies the entire list with individual book data. Modify one book in the cloned object and observe its effect on the original. **Hints:**

● Use Cloneable interface and override clone().

● For deep cloning, clone each Book object inside the list manually.

● Use loops or streams to copy nested objects.

| import java.util.\*;  class Book implements Cloneable {  String title;   Book(String title) {  this.title = title;  }   @Override  protected Object clone() throws CloneNotSupportedException {  return super.clone();  }   public String toString() {  return title;  } }  class Library implements Cloneable {  List<Book> books;   Library(List<Book> books) {  this.books = books;  }   @Override  protected Object clone() throws CloneNotSupportedException {  return super.clone();  }   protected Library deepClone() throws CloneNotSupportedException {  List<Book> clonedBooks = new ArrayList<>();  for (Book b : books)  clonedBooks.add((Book) b.clone());  return new Library(clonedBooks);  }   public String toString() {  return books.toString();  } }  public class LibraryCloneDemo {  public static void main(String[] args) throws CloneNotSupportedException {  List<Book> bookList = new ArrayList<>();  bookList.add(new Book("Java"));  bookList.add(new Book("Python"));  Library original = new Library(bookList);   Library shallow = (Library) original.clone();  Library deep = original.deepClone();   shallow.books.get(0).title = "C++";  System.out.println("After modifying shallow clone:");  System.out.println("Original: " + original);  System.out.println("Shallow: " + shallow);   deep.books.get(1).title = "Data Structures";  System.out.println("After modifying deep clone:");  System.out.println("Original: " + original);  System.out.println("Deep: " + deep);  } } |
| --- |

**Topic 5: Member and Static Inner Classes**

**Problem Statement 5:**

Create an University class with a non-static inner class Department and a static nested class ExamCell. The Department class should access outer class data, while the ExamCell performs general exam operations. Demonstrate access of both inner types from the main method.

**Hints:**

● Use Outer.Inner syntax to create a member inner class object.

● Access outer class fields directly from member inner class.

● Use class name to access static nested class methods.

| class University {  String universityName;   University(String universityName) {  this.universityName = universityName;  }   class Department {  String deptName;   Department(String deptName) {  this.deptName = deptName;  }   void showDetails() {  System.out.println("University: " + universityName + ", Department: " + deptName);  }  }   static class ExamCell {  static void conductExam() {  System.out.println("ExamCell: Conducting exams for all departments.");  }  } }  public class UniversityInnerDemo {  public static void main(String[] args) {  University uni = new University("ABC University");  University.Department dept = uni.new Department("Computer Science");  dept.showDetails();  University.ExamCell.conductExam();  } } |
| --- |

2



**Topic 6: Local and Anonymous Inner Classes**

**Problem Statement 6:**

Create a Payment class with a method processTransaction(). Inside it, define a **local inner class** Validator that checks if payment amount is valid. Also, create an **anonymous inner class** implementing an interface Discount to apply discount dynamically. **Hints:**

● Define local inner class inside a method body.

● Use anonymous inner class for one-time interface implementation.

● Call methods of both classes inside processTransaction().

| interface Discount {  double apply(double amount); }  class Payment {  void processTransaction(double amount) {  class Validator {  boolean isValid(double amt) {  return amt > 0;  }  }   Validator validator = new Validator();  if (validator.isValid(amount)) {  Discount discount = new Discount() {  public double apply(double amt) {  return amt \* 0.9;  }  };  double finalAmount = discount.apply(amount);  System.out.println("Original Amount: " + amount);  System.out.println("Final Amount after Discount: " + finalAmount);  } else {  System.out.println("Invalid payment amount.");  }  } }  public class PaymentInnerDemo {  public static void main(String[] args) {  Payment p = new Payment();  p.processTransaction(1000);  p.processTransaction(-500);  } } |
| --- |

3