

**LAB PROBLEM 1: Comparing Objects Using equals() and == (Any Four)**

**Topic:** Object Class Methods – equals() vs ==

**Problem Statement:**

Create a Book class with title and author fields. Override the equals() method to compare two books based on their title and author. Demonstrate the difference between == and .equals() using two Book objects.

**Hints:**

● Use .equals() for content comparison and == for reference comparison ● Override equals() properly to avoid reference equality

● Use @Override annotation when redefining equals()

| class Book {  private String title;  private String author;   public Book(String title, String author) {  this.title = title;  this.author = author;  }   @Override  public boolean equals(Object obj) {  if (this == obj)  return true;  if (obj == null || getClass() != obj.getClass())  return false;  Book book = (Book) obj;  return title.equals(book.title) && author.equals(book.author);  } }  public class BookComparison {  public static void main(String[] args) {  Book b1 = new Book("1984", "George Orwell");  Book b2 = new Book("1984", "George Orwell");  Book b3 = b1;   System.out.println("b1 == b2: " + (b1 == b2));  System.out.println("b1.equals(b2): " + b1.equals(b2));  System.out.println("b1 == b3: " + (b1 == b3));  System.out.println("b1.equals(b3): " + b1.equals(b3));  } } |
| --- |

**LAB PROBLEM 2: toString() and getClass() Usage**

**Topic:** Object Class Methods – toString(), getClass()

**Problem Statement:**

Create a Car class with brand, model, and price fields. Override the toString() method to display object details. In the main method, print the class name of an object using getClass().getName().

**Hints:**

● Use toString() for readable object representation

● Use getClass() to get runtime class information

● Print both the object (to invoke toString()) and its class name

| class Car {  private String brand;  private String model;  private double price;   public Car(String brand, String model, double price) {  this.brand = brand;  this.model = model;  this.price = price;  }   @Override  public String toString() {  return "Brand: " + brand + ", Model: " + model + ", Price: " + price;  } }  public class CarDetails {  public static void main(String[] args) {  Car c1 = new Car("Toyota", "Camry", 30000);  System.out.println(c1);  System.out.println("Class Name: " + c1.getClass().getName());  } } |
| --- |

1



**LAB PROBLEM 3: hashCode() and equals() Contract**

**Topic:** Object Class Methods – hashCode() and equals() Relationship

**Problem Statement:**

Create a Student class with id and name fields. Override both equals() and hashCode() methods to ensure two students with the same id are treated as equal. Demonstrate storing Student objects in a HashSet and show how duplicates are handled. **Hints:**

● Use Objects.hash(id) for hashCode()

● Ensure consistent results for equals() and hashCode()

● Print HashSet elements to verify duplicates are avoided

| import java.util.\*;  class Student {  private int id;  private String name;   public Student(int id, String name) {  this.id = id;  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 id == s.id;  }   @Override  public int hashCode() {  return Objects.hash(id);  }   @Override  public String toString() {  return "Student{id=" + id + ", name='" + name + "'}";  } }  public class StudentHashSetDemo {  public static void main(String[] args) {  HashSet<Student> students = new HashSet<>();  students.add(new Student(1, "Alice"));  students.add(new Student(2, "Bob"));  students.add(new Student(1, "Alice"));  for (Student s : students)  System.out.println(s);  } } |
| --- |

**LAB PROBLEM 4: Cloning an Object**

**Topic:** Object Cloning – Shallow Copy vs Deep Copy

**Problem Statement:**

Create a Person class with name and Address object as fields. Implement Cloneable and demonstrate both **shallow** and **deep** cloning.

**Hints:**

● Use implements Cloneable and super.clone()

● For deep copy, create a new Address object manually in clone()

● Print original and cloned objects to compare changes

| class Address {  String city;  String state;   public Address(String city, String state) {  this.city = city;  this.state = state;  }   @Override  public String toString() {  return city + ", " + state;  } }  class Person implements Cloneable {  String name;  Address address;   public Person(String name, Address address) {  this.name = name;  this.address = address;  }   @Override  protected Object clone() throws CloneNotSupportedException {  return super.clone();  }   protected Person deepClone() throws CloneNotSupportedException {  Person cloned = (Person) super.clone();  cloned.address = new Address(address.city, address.state);  return cloned;  }   @Override  public String toString() {  return name + " - " + address;  } }  public class CloningDemo {  public static void main(String[] args) throws CloneNotSupportedException {  Address addr = new Address("Delhi", "Delhi");  Person p1 = new Person("John", addr);   Person shallowCopy = (Person) p1.clone();  Person deepCopy = p1.deepClone();   p1.address.city = "Mumbai";   System.out.println("Original: " + p1);  System.out.println("Shallow Copy: " + shallowCopy);  System.out.println("Deep Copy: " + deepCopy);  } } |
| --- |

2



**LAB PROBLEM 5: Member Inner Class**

**Topic:** Inner Class – Member Inner Class

**Problem Statement:**

Create an Outer class with a private field message and a non-static inner class Inner that displays the message. Create an object of Inner class using an instance of Outer and call its display method.

**Hints:**

● Define inner class inside outer class without static keyword

● Use syntax: Outer.Inner innerObj = outerObj.new Inner(); ● Access outer class members directly from inner class

**LAB PROBLEM 6: Static, Local, and Anonymous Inner Classes**

**Topic:** Inner Class Variants

**Problem Statement:**

Create a Calculator class that includes:

1. A **static nested class** Operation to perform addition.

2. A **local inner class** inside a method to perform subtraction.

3. An **anonymous inner class** implementing an interface MathOperation for multiplication.

Demonstrate all three operations.

**Hints:**

● Define Operation as a static nested class

● Define local inner class inside a method

● Use anonymous inner class for single-use implementation

3