1. What is the contract between `equals()` and `hashCode()`?

Answer:

The contract is defined by the Java specification:

- If two objects are **equal** according to `equals()`, they **must** return the **same hash code**.
- If two objects return the same hash code, they **may or may not** be equal (hash collision).
- If two objects are **not equal**, it's **not required** but better if they return different hash codes to avoid collisions.

---

2. Why must we override `hashCode()` when overriding `equals()`?

Answer:

Collections like `HashMap`, `HashSet`, and `Hashtable` use `hashCode()` to group objects into buckets. If `equals()` is overridden but `hashCode()` isn't, equal objects might go into different buckets, violating the contract and causing bugs (like duplicates in a `HashSet`).

---

3. Can two unequal objects have the same hash code?

Answer:

Yes, this is called a **hash collision**. It's legal and expected behavior. The `hashCode()` function only spreads objects across a finite number of buckets, so collisions are inevitable.

---

4. What is the effect of a bad `hashCode()` implementation?

Answer:

- Poor distribution (e.g., always returning the same hash code) degrades `HashMap` or `HashSet` performance from **O(1)** to **O(n)**.
- Increases the number of collisions.
- Causes uneven load in hash buckets.

---

5. Should `equals()` be symmetric in subclasses?

Answer:

Yes. `equals()` should be **symmetric**: if `a.equals(b)` is true, then `b.equals(a)` must also be true. Using `instanceof` for type checks in inheritance chains often breaks symmetry.

---

6. What's the difference between `==` and `equals()`?

Answer:

- `==` compares **reference equality** (i.e., whether two references point to the same object).
- `.equals()` compares **logical equality** (i.e., whether the object contents are the same).

---

7. Can mutable objects be used as keys in a `HashMap`?

Answer:

Yes, but it's **not recommended**. If the object's fields used in `equals()`/`hashCode()` change after insertion, the map may not be able to find the object again, causing data corruption.

---

8. What is the role of `Objects.equals()` in Java 7+?

Answer:

`Objects.equals(a, b)` safely handles nulls: it returns true if both are null, or if `a.equals(b)` is true. It avoids `NullPointerException`.

CODING INTERVIEW QUESTIONS WITH ANSWERS

---

1. Create a `Person` class and override `equals()` and `hashCode()` properly.

```
java
CopyEdit
class Person {
    private String name;
    private int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    @Override
```

```
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Person person = (Person) o;
        return age == person.age && name.equals(person.name);
    }

    @Override
    public int hashCode() {
        return Objects.hash(name, age);
    }
}
```

---

2. What happens if you override `equals()` but not `hashCode()`? Demonstrate.

```
java
CopyEdit
class BadPerson {
    private String name;

    @Override
    public boolean equals(Object o) {
        return o instanceof BadPerson && ((BadPerson)
o).name.equals(this.name);
    }
}

// Test
Set<BadPerson> set = new HashSet<>();
BadPerson p1 = new BadPerson("John");
BadPerson p2 = new BadPerson("John");
set.add(p1);
set.add(p2); // Won't detect duplicate without correct hashCode()
```

---

3. Write a class with a composite key for use in `HashMap`.

```
java
CopyEdit
class DepartmentKey {
    private String dept;
    private int location;

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (!(o instanceof DepartmentKey)) return false;
        DepartmentKey that = (DepartmentKey) o;
        return location == that.location && dept.equals(that.dept);
    }

    @Override
    public int hashCode() {
        return Objects.hash(dept, location);
    }
}
```

---

4. How to violate `equals()` symmetry with inheritance?

```
java
CopyEdit
class Animal {
    public boolean equals(Object o) {
        return o instanceof Animal;
    }
}

class Dog extends Animal {
    public boolean equals(Object o) {
        return o instanceof Dog;
    }
}
```

This breaks symmetry:

```
java
CopyEdit
Animal a = new Animal();
Dog d = new Dog();
a.equals(d); // true
d.equals(a); // false
```

---

5. Create a cached `hashCode()` for an immutable class.

```
java
CopyEdit
class ImmutableKey {
    private final String value;
    private int hash;

    public ImmutableKey(String value) {
        this.value = value;
    }

    @Override
    public int hashCode() {
        if (hash == 0) {
            hash = value.hashCode();
        }
        return hash;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (!(o instanceof ImmutableKey)) return false;
        ImmutableKey that = (ImmutableKey) o;
        return value.equals(that.value);
    }
}
```

---

6. Write a JUnit test for `equals()` and `hashCode()`.

```
java
CopyEdit
@Test
public void testEqualsHashCode() {
    Person p1 = new Person("Alice", 30);
    Person p2 = new Person("Alice", 30);

    assertEquals(p1, p2);
    assertEquals(p1.hashCode(), p2.hashCode());
}
```

---

7. Using `@Data` in Lombok and excluding fields.

```
java
CopyEdit
@Data
class Product {
    private String name;
    private double price;

    @EqualsAndHashCode.Exclude
    private Date createdAt;
}
```

---

8. Refactor legacy `equals()` using `Objects.equals()`.

Before:

```
java
CopyEdit
public boolean equals(Object o) {
    Employee e = (Employee) o;
    return this.id == e.id && this.name.equals(e.name); // Risky
}
```

After:

```
java
CopyEdit
@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (!(o instanceof Employee)) return false;
    Employee e = (Employee) o;
    return id == e.id && Objects.equals(name, e.name);
}

@Override
public int hashCode() {
    return Objects.hash(id, name);
}
```