Equals and HashCode Contract - Java Interview Notes

Equals and HashCode Contract - Java Interview Notes

1. Purpose

- Used for object comparison and storing objects in hash-based collections like:
- HashMap, HashSet, Hashtable, LinkedHashMap, etc.

2. Core Contract

If two objects are equal according to equals(), then:

a.equals(b) -> true -> a.hashCode() == b.hashCode()

The reverse is not always true:

a.hashCode() == b.hashCode() does not imply a.equals(b)

(Collisions are allowed.)

3. Method Signatures

public boolean equals(Object obj)

public int hashCode()

4. When to Override

- Override both in:
- Custom key classes used in maps/sets.
- Value comparison scenarios (e.g., user-defined equality).

5. Best Practices

- Use @Override annotation.

- Use Objects.equals() and Objects.hash() for clean code.
- Always override hashCode() if equals() is overridden.

6. Example

```
class Employee {
String id;
String name;

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

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

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

7. Common Mistakes

- Overriding equals() but not hashCode().
- Using mutable fields in equals() and hashCode().
- Failing to maintain symmetry, transitivity, and consistency.

8. Important Rules for equals()

- Reflexive: x.equals(x) must be true.
- Symmetric: x.equals(y) <-> y.equals(x)
- Transitive: x.equals(y) & y.equals(z) -> x.equals(z)
- Consistent: Multiple invocations return the same result.
- Non-nullity: x.equals(null) must return false.