Overriding equals() & hashcode() methods

P. Shamoi

Intro

- ***Equals*** and ***hashCode*** in Java are two fundamental methods which are declared in Object class and part of core Java library.
- ***equals***() method is used to compare Objects for equality while ***hashCode()*** is used to generate an integer code corresponding to that object.

Default implementation

- Default implementation of equals() method provided by java.lang.Object compares memory location and only returns true if two references are pointing to the same memory location i.e. essentially they are the same objects.
- e.g. String overrides equals, whose implementation of equals() method returns true if the content of two String objects is exactly the same.

Equals() and hashcode() contract

- If two objects are equal by equals() method then their hashcodes must be the same.

- If two objects are not equal by equals() method then their hashcodes could be the same or different.

Standard approach to override equals

- Do ***this*** check -- if yes then return true
- Do ***null*** check -- if yes then return false.
- Use getClass() method to check that classes are the same

```
if (obj == this) {
    return true;
}
```

```
if((obj == null) || (obj.getClass() != this.getClass())) {
        return false;
}
```

- Type cast the object
- Compare individual attribute starting with numeric attribute. (WHY ?)
  - If the first field does not match, don't try to match the rest of attributes and return false.

Example

```
public class User {
    private String name;
    private int age;
    private String passport;

    //getters and setters, constructor

    @Override
    public boolean equals(Object o) {

        if (o == this) return true;
        if (!(o instanceof User)) {
            return false;
        }

        User user = (User) o;

        return user.name.equals(name) &&
                user.age == age &&
                user.passport.equals(passport);
    }
```

THE MORAL IS – IF YOU OVERRIDE EQUALS(), OVERRIDE HASHCODE

hashcode()

- When inserting an object into a *hastable* you use a key. The hash code of this key is calculated, and used to determine where to **store** the object internally.
- When you need to lookup an object in a hashtable you also use a key. The hash code of this key is calculated and used to determine where to search for the object.

- The hash code only points to a certain "area" (or list, bucket etc) internally.
- Since different key objects could potentially have the same hash code, the hash code itself is no guarantee that the right key is found. **The hashtable then iterates this area (all keys with the same hash code) and uses the key's equals() method to find the right key.**

- So, as you can see, a combination of the **hashCode**() and **equals**() methods are used when storing and when looking up objects in a hashtable.

If equal, then same hash codes too.

Same hash codes no guarantee of being equal.

Examples

```
public int hashCode(){
  return (int) employeeId *
               firstName.hashCode() *
               lastName.hashCode();
}
```

```
public int hashCode() {
    int result = 17;
    result = 31 * result + name.hashCode();
    result = 31 * result + age;
    result = 31 * result + passport.hashCode();
    return result;
}
```

In JDK 7

```
@Override
public boolean equals(Object o) {

    if (o == this) return true;
    if (!(o instanceof User)) {
        return false;
    }
    User user = (User) o;
    return age == user.age &&
            Objects.equals(name, user.name) &&
            Objects.equals(passport, user.passport);
}

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

For JDK 7 and above, you can use the new Objects class to generate the equals and hash code values.

Common mistake

- Instead of overriding equals() method programmer overloads it.
- Syntax of **equals** method defined in **Object** class is `public boolean equals(Object obj)` but many people unintentionally overload **equals** method in Java by writing `public boolean equals(Person obj)`, instead of using Object as an argument they use their class name.
- WHY it is a mistake?

Usage

- inserting and retrieving Object in HashMap
- avoiding duplicates in HashSet and other Set implementations
- to determine if a collection contains a given element

And…

- every other place where you need to compare Objects.