Java & JEE Training

Day 14 – Object Class

MindsMapped Consulting

Quick recap: Topics covered till now

- Data Types – Primitive (byte, short, int, long, double, float, char, boolean)
- Data Types – Wrapper (Byte, Short, Integer …)
- Data Types – Classes for String (String - immutable, StringBuilder – mutable, StringBuffer – mutable+threadsafe); Concept of String pooling / interning
- Arrays
- Conditional statements, Loops
- OOP with Java –
  - Polymorphism – Method overloading (Compile-time), Method Overriding (Runtime)
  - Inheritance – extending classes, implementing interfaces; multiple inheritance
  - Abstraction – abstract classes vs interfaces
  - Encapsulation – POJO classes / Beans
- Packaging, Class Modifiers (public, protected, default, private)
- Static members vs Instance members
- Exception handling

Object Class
as
“The basic unit of any collection”

MindsMapped Consulting

Object class

- Superclass for all Java classes
- Any class without explicit *extends* clause is a direct subclass of Object
- Methods of Object include:
  - String toString()
  - boolean equals (Object other)
  - int hashCode()

Method toString()

- Returns String representation of object; describes state of object
- Automatically called when:
  - Object is concatenated with a String
  - Object is printed using print() or println()
  - Object reference is passed to assert statement of the form:

    assert condition : object

Example

```
Rectangle r = new Rectangle (0,0,20,40);
System.out.println(r);
```

Prints out:

```
java.awt.Rectangle[x=0,y=0,width=20,height=40]
```

More on toString()

- **Default toString() method just prints (full) class name & hash code of object**
- **Not all API classes override toString()**
- **Good idea to implement for debugging purposes:**
  - **Should return String containing values of important fields along with their names**
  - **Should also return result of getClass().getName() rather than hard-coded class name**

Overriding toString(): example

```
public class Employee
{
        public String toString()
        {
                return getClass().getName()
                + "[name=" + name
                + ",salary=" + salary
                + "]";
        }
        …
}
```

Typical String produced: Employee[name=John Doe,salary=40000]

Overriding toString in a subclass

- **Format superclass first**
- **Add unique subclass details**
- **Example:**

```
public class Manager extends Employee
{
  public String toString()
  {
          return super.toString()
                  + "[department=" + department + "]";
  }
...
}
```

Example continued

- **Typical String produced:**
  Manager[name=Mary Lamb,salary=75000][department=Admin]

- **Note that superclass reports actual class name**

Equality testing

- **Method equals() tests whether two objects have same contents**
- **By contrast, == operator test 2 references to see if they point to the same object (or test primitive values for equality)**
- **Need to define for each class what "equality" means:**
  - Compare all fields
  - Compare 1 or 2 key fields

Equality testing

- **Object.equals tests for identity:**

```
public class Object {

        public boolean equals(Object obj) {
                return this == obj;
        }
        ...
}
```

- **Override equals if you don't want to inherit that behavior**

Overriding equals()

- **Good practice to override, since many API methods assume objects have well-defined notion of equality**
- **When overriding equals() in a subclass, can call superclass version by using super.equals()**

Requirements for equals() method

- **If x is not null, then x.equals(null) must be false**

Reflexive	***a=a***
Symmetric	***if a=b, then b=a***
Transitive	***if a=b, and b=c, then a=c***

The perfect equals() method

```
public boolean equals(Object otherObject)
{
        if (this == otherObject) return true;
        if (otherObject == null) return false;
        if (getClass() != otherObject.getClass()) return false;
        ...
}
```

Hashing

- **Technique used to find elements in a data structure quickly, without doing full linear search**
- **Important concepts:**
  - **Hash code: integer value used to find array index for data storage/retrieval**
  - **Hash table: array of elements arranged according to hash code**
  - **Hash function: computes hash code for element; uses algorithm likely to produce different hash codes for different objects to minimize collisions**

Hashing in Java

- **Java library contains HashSet and HashMap classes**
  - **Use hash tables for data storage**
  - **Since Object has a hashCode method (hash function), any type of object can be stored in a hash table**

Default hashCode()

- **Hashes memory address of object; consistent with default equals() method**
- **If you override equals(), you should also redefine hashCode()**
- **For class you are defining, use product of hash of each field and a prime number, then add these together – result is hash code**

Example

```
public class Employee
{
        public int hashCode()
        {
                return 11 * name.hashCode()
                + 13 * new Double(salary).hashCode();
        }
        …
}
```