Inheritance

Part 4 – Object Class

Chapter 5, Core Java Volume I

Contents

- Object Class
- euqals Method
- hasCode method
- toString method
- Example
- Design Hints for Inheritance

Object : The Cosmic Superclass

- The "Object" class is a superclass of all Java classes.

  ```
  java.lang.Object
  ```

- Every class has the Object class as a superclass directly or indirectly by default.

  ```
  public class Employee { }                    // Object is a super class implicitly
  public class Employee extends Object { }     // Object is a super class explicitly
  public class Manger extends Employee { }     // Object is a super class indirectly
  ```

- All array types are class types that extend the Object class.
- Any object or array reference can be stored in a variable of type Object: (*is-a relationship*)

  ```
  Object obj1 = new Employee("Harry Hacker", 35000);
  Object obj2 = new int[10];
  ```

- **Caution:** Since a variable of type Object is used as a generic holder, to do specific operation , we need to casting

  ```
  Employee e = (Employee) obj1 ;
  ```

- Object class has useful methods: equals(), hashCode(), toString(), etc.

The equals Method

- equals() tests whether the object references are identical.
- Override to test when two objects should be equal in terms of their states.
- Example: Consider two Employee objects equal if their fields are the same.

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

   Employee other = (Employee) otherObject;
   return name.equals(other.name)
      && salary == other.salary
      && hireDay.equals(other.hireDay);
}
```

The *equals* Method

- If name or hireDay are null, How to compare them?
- **Solution**: invoke the static method "Objects.equals(a,b)"-(*null safe*).
  - This method returns true if both arguments a and b are null.
  - It returns false if only one is null.
  - Otherwise, calls a.equals(b).

```
return Objects.equals(name, other.name)
        && salary == other.salary
       && Object.equals(hireDay, other.hireDay);
```

The *equals* Method in a Subclass

- How to override **equals()** method in a subclass
  - First, invoke **equals()** method on super class: **super.equals()**
  - If it returns true, then compare instance fields of a subclass.

```
public class Manager extends Employee
{
  …
  @Override
  public boolean equals(Object otherObject)
  {   // super.equals checks  that "this" and other belong to the same class
     if ( ! super.equals(otherObject) )
          return false;
     Manager other = (Manager) otherObject;
     return this.bonus == other.bonus; // compare fields
  }
}
```

The hashCode Method

- Hash code=integer derived from an object.
- Hash codes should be scrambled: If x and y are not equal, then x.hashCode() and y.hashCode() should be different.
  - Hash code computation in the String class:

```
int hash = 0;
for (int i = 0; i < length(); i++)
   hash = 31 * hash + charAt(i);
```

- "Hello".hashCode() is 69609650, "Harry".hashCode() is 69496448.

The hashCode Method

- Hash codes must be consistent: If x and y are equal, then their hash codes must be equal.
  - hashCode() in Object class is derived from memory location.
  - Override hashCode() whenever you override equals()!
  - Combine the hash codes of the fields that the equals() method compares:

```
public class Employee
{
   . . .
   public int hashCode()
   {
      return Objects.hash(name, salary, hireDay);
   }
}
```

The toString Method

- public String toString();
- toString() method returns a string representation of an object.
- When we concatenate a string and an object, the toString method is invoked on the object:

  "Center: " + p ; // compiler calls p.toString() automatically
- Note: The "Object" class defines the toString() to print the class name and the hash code of the object.
  - For example, the call System.out.println(System.out), display the following:
  - java.io.PrintStream@2f6684
- Note: we must Override toString() to get meaningful meaning for our own class .

The toString Method

- Example: java.awt.Point class

```
java.awt.Point[x=10,y=20]
```

- Override toString() method

```
public class Point
{
    …
   @Override
    public String toString()
    {
       return  “ java.awt.Point[x =" + x + ", y= " + y + "] ";
    }
 }
```

Inheritance and the toString Method

- In *Employee* class:

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

- In *Manager* subclass:

```
public String toString()
{
   return super.toString() + "[bonus=" + bonus + "]";
}
```

- Result format:

```
Manager[name=...,salary=...,hireDay=...][bonus=...]
```

Example: EqualsTest

```
package equals;
public class EqualsTest
{
 public static void main(String[] args)
 {
   Employee alice1 = new Employee("Alice Adams", 75000, 1987, 12, 15);
   Employee alice2 = alice1;
   Employee alice3 = new Employee("Alice Adams", 75000, 1987, 12, 15);
   Employee bob = new Employee("Bob Brandson", 50000, 1989, 10, 1);
  System.out.println("alice1 == alice2:" + (alice1 == alice2));
  System.out.println("alice1 == alice3:" + (alice1 == alice3));
  System.out.println("alice1.equals(alice3):" + alice1.equals(alice3));
  System.out.println("alice1.equals(bob):" + alice1.equals(bob));
  System.out.println("bob.toString(): " + bob);
```

```
  Manager carl = new Manager("Carl Cracker", 80000, 1987, 12, 15);
  Manager boss = new Manager( "Carl Cracker", 80000, 1987, 12, 15);
  boss.setBonus(5000);
  System.out.println("boss.toString(): " + boss);
  System.out.println("carl.equals(boss): " + carl.equals(boss));
   System.out.println("alice1.hashCode(): " + alice1.hashCode());
  System.out.println("alice3.hashCode(): " + alice3.hashCode());
  System.out.println("bob.hashCode(): " + bob.hashCode());
  System.out.println("carl.hashCode(): " + carl.hashCode());

 } // end of main()
} //end of EqualsTest class
```

Example: Employee

```
package equals;
import java.time.*;
import java.util.Objects;
public class Employee
{
  private String name;
  private double salary;
  private LocalDate hireDay;

 public Employee(String name, double salary, int year,
     int month, int day)
 {
    this.name = name;
    this.salary = salary;
    hireDay = LocalDate.of(year, month, day);
 }
 … methods here …
```

```
public boolean equals(Object otherObject)
{
   // a quick test to
   if (this == otherObject) return true;
   if (otherObject == null) return false;
   // to test class match
    if (getClass() != otherObject.getClass())
    return false;
   // now otherObject is a non-null Employee
    Employee other = (Employee) otherObject;      // test  field by field
    return Objects.equals(name, other.name)
              && salary == other.salary
             && Objects.equals(hireDay, other.hireDay);
}
```

Example: Employee

```
public int hashCode()
{
   return Objects.hash(name, salary, hireDay);
 }
 public String toString()
 {
   return getClass().getName() + "[name=" + name + ",salary="
   + salary + ",hireDay=" + hireDay + "]";
  } // end of ToString
} // End of Employee
```

Example: Manager

```
package equals;
public class Manager extends Employee
{
  private double bonus;
  public Manager(String name, double salary, int year,
     int month, int day)
  {
    super(name, salary, year, month, day);
    bonus = 0;
  }
  public double getSalary()
  {
    double baseSalary = super.getSalary();
    return baseSalary + bonus;
  }
  public void setBonus(double bonus)
  {
    this.bonus = bonus;
  }
```

```
public boolean equals(Object otherObject)
{
   if (!super.equals(otherObject)) return false;
   Manager other = (Manager) otherObject;
   // super.equals compare class of  this and other
   return bonus == other.bonus;
}
 public int hashCode()
 {
   return super.hashCode() + 17 * new Double(bonus).hashCode();
 }
 public String toString()
 {
   return super.toString() + "[bonus=" + bonus + "]";
 }
} // End of manger class
```

Design Hints for Inheritance

- Place common operations and fields in the superclass.
- Don’t use protected fields.
- Use inheritance to model the “is–a” relationship:

```
public class Rectangle extends Point
{
   private inte height;
   private inte width;
   …
}
```

  - Is a rectangle a point?
  - No, it has a point!

```
public class Rectangle
{
   private Point p;
   private inte height;
   private inte width;
    …
}
```

- Don’t use inheritance unless all inherited methods make sense:

```
class Holiday extends GregorianCalendar { . . . } // ???
. . .
Holiday christmas;
christmas.add(Calendar.DAY_OF_MONTH, 12);
```

Design Hints for Inheritance

- Use polymorphism, not type information:

```
if (x is of type 1)
  action1(x);
else if (x is of type 2)          ⇒  x.action();
  action2(x);
```