09_1 Object Class

Object-Oriented Programming

Object class에 대해 강의하겠습니다.

java.lang package revisited

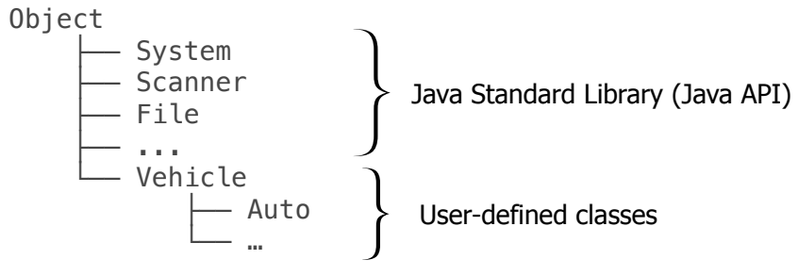
- **Object class**
- System class
- Class class
- String, StringBuilder, StringTokenizer
- Wrapper classes: Byte, Short, Character, Integer, Float, Double, Boolean
- Math class

페이지 two

java.lang package에 여러가지 중요한 class들이 포함되어 있고
그 중에 Object class에 대해 자세히 알아보겠습니다

Object class

- Inherited by default by all classes (in all packages) without 'extends' keyword
- Every class (including user-defined class) in java is a child or descendant of Object.



페이지 3

Object class는 모든 package 안의 모든 class들의 ancestor가 되는 class입니다
User defined class들도 명시적으로 extends를 하지 않지만
자동으로 Object class의 descendant가 되는 것입니다

Methods in Object class (1/2)

- boolean equals(Object obj)
  - Testing whether two objects (caller and obj)'s references are the same or not
  - 'equals(obj)' is overridden and rewritten for almost all classes
- String toString()
  - Return the string representing the object's information
  - toString() is overridden and rewritten for almost all classes

페이지 4

Object class는 모든 class들이 상속받는 중요한 method들을 가지고 있습니다
우리는 이미 equals()와 toString()에 대해 알고 있습니다
equals()는 override되지 않는 다면 reference 값의 동일함 여부를 test하며
흔히 overriding되어 두 object의 내용이 같은지를 비교하는데 사용됩니다
toString()은 object에 대한 정보를 String으로 return하는데
주로 instance variable들의 value들을 정보로 포함하게 됩니다

Methods in Object class (2/2)

- int hashCode()
  - Return the hash code (an integer identifying the object) of the object
  - The same objects should return the same hashCode()
  - The hashCode() method of Object class usually returns the memory address of the object
  - Overridden hashCode() usually implemented with the value of other instance variables
  - hashCode() is mainly used with hash-based collection such as 'HashMap', 'HashSet', and 'HashTable' (see later chapter for collections)
  - Multiplying prime number is a technique to reduce the hash collision

페이지 5

hashCode() methyod는 각 object마다 고유하게 가진 번호를 return합니다.
때문에 같은 object들은 같은 hashCode() 값을 return해야 합니다.
원래 대문자 Object class의 hashCode() 는
그 object의 memory address를 return합니다.
hashCode()가 overriding 되면
해당 object의 instance variable 값들을 조합하여
새로운 hashCode 값을 만들어 냅니다.
hashCode() 는 주로 hash 기반의 collection들인
HashMap, HashSet, HashTable 등에 주로 사용됩니다.
이 collection들에 대해서는 뒤의 Collections chapter에서 자세히 다룰 것입니다.
새로 만들어 내는 hashCode 값은 지금까지 나온 적이 없는
새로운 hashCode 값이 되는 것이
hash 기반 collection에서 data 위치들의 충돌을 방지할 수 있기 때문에
prime number를 instance variable value에 곱하는
기법을 자주 활용합니다.

Example) ObjectClassTest (1/4)

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

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

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }
```


페이지 6

이제 Object를 상속받는 user defined class Person을 살펴보겠습니다
Person은 name과 age, 두 개의 private instance variable들을 가지고 있습니다
Constructor는 name과 age값을 parameter로 받아
instance variable들에 assign해 주고 있습니다.
Instance variable들이 private이기 때문에
name을 위한 accessor method인 getName()과
age를 위한 accessor method인 getAge()가 존재합니다.

Example) ObjectClassTest (2/4)

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

        if (this == other) return true;
        if (other == null || getClass() != other.getClass()) return false;

        Person person = (Person) other; // downcasting

        return age == person.age &&
           (name == null ?
            person.name == null :
            name.equals(person.name));

    }
```


페이지 7

equals method가 overriding되어 있습니다.
equals는 원래 Object class에 존재하던 대로
Object type parameter를 넘겨 받아야 합니다.
equals에서는 먼저 현재 object가 null인지와
현재 object의 getClass()와 other object의 getClass()가
다른지를 test합니다.
만약 이 조건이 하나라도 만족한다면
this와 other는 내용을 볼 필요도 없이 다르다고 판단할 수 있기 때문에
false를 return하고 method를 끝냅니다.
그렇지 않다면 other는 Person class의 object일 것이므로
먼저 할 일은 large Object type인 other를
Person class type으로 down casting 하는 것입니다.
이 down casting이 가능한 이유는
위의 test에서 this.getClass()가 other.getClass()와 같다고 했고
따라서 other는 원래 Person class object로 생성된 것이기 때문입니다.
Down casting이 되었다면 이제 할일은
두 object들의 instance variable들의 값이 모두 같은지를 test하는 일 뿐입니다.
age가 같아야 하고, name이 둘다 null 이거나,
null이 아니라면 두 name이 같은지는
이미 잘 정의되어 있는 String class의 equals method를 활용하여
알아보면 되겠습니다.

Example) ObjectClassTest (3/4)

```
    @Override
    public int hashCode() {
        int result = ((name != null) ? name.hashCode() : 0);
        result = 31 * result + age; // multiplying the prime number 31
        return result;
    }

    @Override
    public String toString() {
        return "Person{" +
                "name='" + name + '\'' +
                ", age=" + age +
                '}';
    }
}
```


페이지 8

hashCode() 의 overriding은 처음 나오는 것인데요.
새로운 hashCode 값은 31 곱하기 name의 hashCode 더하기 age로 계산하였습니다.
여기서 31은 prime number로서
이렇게 prime number를 instance variable 값에 곱해 줌으로써
서로 다른 object들의 hashCode가 같은 값이 나올 확률을 줄여줄 수 있습니다.
toString() 은 instance variable value들을 보여주도록
overriding되었습니다.

Example) ObjectClassTest (4/4)

```java
public class ObjectClassTest {

    public static void main(String[] args) {
        Person person1 = new Person("John", 25);
        Person person2 = new Person("John", 25);
        Person person3 = new Person("Jane", 30);

        System.out.println("person1 equals person2: "+person1.equals(person2)); // true
        System.out.println("person1 equals person3: "+person1.equals(person3)); // false

        System.out.println("person1 hashCode: " + person1.hashCode()); // 71750734
        System.out.println("person2 hashCode: " + person2.hashCode()); // 71850734
        System.out.println("person3 hashCode: " + person3.hashCode()); // 71339152

        System.out.println("person1 toString: " + person1.toString()); //...
        System.out.println("person2 toString: " + person2.toString());
        System.out.println("person3 toString: " + person3.toString());
    }
}
```


페이지 9

Person class를 test해 보는 code 입니다.
먼저 person1, person2, person3의 세 object들을 생성하였습니다.
person1과 person2는 equal하다고 나오고
person1과 person3는 equal하지 않다고 맞는 결과가 나오고 있습니다.
세 object들의 hashCode들도 모두 다르게 나와 충돌을 피하고 있습니다.
마지막으로 toString들을 test해 보았습니다.

Example) EqualsWithPolymorphism (1/3)

```java
class AClass {
    private int x;

    public AClass(int x) { this.x = x; }

    @Override
    public boolean equals(Object obj) {
        if (obj == null) return false;
        if (obj instanceof AClass) {
            AClass other = (AClass) obj;
            if (x == other.x) return true;
            return false;
        }
        return false;
    }
}
```

Polymorphism:
Descendant object can be assigned to Ancestor type reference variable

페이지 10

이번에는 hierarchy가 있는 class들간에 polymorphism을 고려할 때
equals method의 구현에 주의해야 하는 점을 예를 들어 설명하겠습니다.
polymorphism의 개념을 간단히 복습해 보면
descendant type object를 ancestor type의 reference variable에
assign할 수 있다는 것이며
이 때 descendant type class에서 overriding 된 method를
그 type들에 따라 다르게 적용할 수 있다는 것이었습니다.
이제 base class로 AClass를 고려해 보겠습니다
AClass에는 int x 가 유일한 instance variable 입니다.
Constructor가 정의되어 있고
equals는 적절한 test를 거친 후 this.x와 other.x가 같은지 비교하는 것으로
구현되어 override되어 있습니다.

Example) EqualsWithPolymorphism (2/3)

```
class BClass extends AClass {
    private int y;
    public BClass(int x, int y) { super(x); this.y = y; }

    public boolean equals(Object obj) {
        if (obj == null) return false;
        if (obj instanceof BClass) {
            BClass other = (BClass) obj; // downcasting
            if (super.equals(obj) && y == other.y)
                return true; // using super.equals
            return false;
        }
        return false;
    }
}

class CClass {
    private String name;
    public CClass(String name) { this.name = name; }
}
```


페이지 11

AClass의 child인 BClass에는
int y 하나가 더 instance variable로 추가되었습니다.
overriding된 equals method에서는
super.equals를 이용하여 x가 같은지를 확인하고
this.y 와 other.y가 같은지를 더 체크하여
결과를 return합니다.
CClass는 AClass와 BClass의 hierarchy에 속해있지 않은
독자적인 class로서
String name 하나만을 instance variable로 가지고 있습니다

Example) EqualsWithPolymorphism (3/3)

```
public class EqualWithPolymorphism {
    public static void main(String[] args) {
        AClass a1 = new AClass(3);
        AClass a2 = new AClass(7);
        BClass b1 = new BClass(3, 5);
        BClass b2 = new BClass(7, 9);
        CClass c = new CClass("Korea");
        System.out.println(a1.equals(a1));  // true
        System.out.println(a1.equals(b1));  // true
        System.out.println(a1.equals(b2));  // false
        System.out.println(a2.equals(c));   // false
    }
}
```


페이지 12

Main class에서 먼저 AClass object인 a1과 a2을 생성하여
x값에 각각 3과 7을 assign 하였습니다.
BClass object인 b1과 b2에는 x와 y를 각각 (3, 5) 그리고 (7,9) 를 assign했습니다.
CClass인 object c는 "Korea" 를 String name에 assign 하였습니다.
먼저 a1.equals(a1) 을 call 했는데, a1 자신과 자신을 비교하는 것이니까
당연히 같게 되고, 답은 true가 나옵니다.
a1.equals(b1) 을 call 할 때 a1.equals의 parameter type이 AClass이고
BClass가 AClass의 child이기 때문에
polymorphism이 적용되어 BClass object인 b1을
parameter로 passing 받을 수 있게 됩니다.
a1.equals 내부에서 b1은 AClass type으로 down casting되어
x 값 만이 같은지를 비교하게 됩니다.
따라서 a1.equals(b1) 은 a1과 b1 모두 x가 3이기 때문에
true를 return하게 됩니다.
그러나 a1.equals(b2)를 하면, b2의 x value는 7이기 때문에
false를 return하게 됩니다.
한편 a2.equals(c) 는 당연히 false가 되는데
AClass인 a2와 CClass인 c가 서로 class가 다르며
c를 AClass로 down casting 할 수도 없고
따라서 비교가 불가능하기 때문입니다.