05_3 Copy Constructor

Object-Oriented Programming

05-3 Copy Constructor에 대해 강의하겠습니다.

Copy Constructor

- Constructor with a single argument of the same type as the class
- Should create an object that is a separate, independent object
- Values of instance variables are the same as original

페이지 2

Copy constructor는 parameter로 같은 class type의 object를 넘겨주어
내용을 그대로 copy하면서 새로운 object를 생성하게 하는 constructor입니다.
새로 만들어지는 object는 parameter로 넘겨진 object와는
완전히 분리된 새로운 object여야 합니다.
단, instance variable들의 value들은 parameter object의
그것들과 같게 됩니다.

Copy Constructor Example

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

    // Copy Constructor
    public Person(Person person) {
        this.name = person.name;
        this.age = person.age;
    }
```

```java
    // Original Person object
    Person original = new Person("John Doe", 30);

    // Using copy constructor
    Person copy = new Person(original);
```


페이지 3

여기 copy constructor를 define하고
사용하는 example을 살펴 봅니다.
Person class에는 String name과
int age라는 두 개의 instance variable들이 있습니다.
Copy constructor는 Person object인 person을
parameter로 받습니다.
그리고 새로 만들어지는 object의 instance variable들의 값인
this.name과 this.age를 각각 parameter인
person.name과 person.age의 값으로 assign합니다.

original이라는 Person object하나를 생성했는데
name은 "John Doe", age는 30 입니다.
이 original을 copy하여 새로운 object를 생성하기 위해
copy constructor를 사용하면 됩니다.
Person copy = new Person(original);
이렇게 copy constructor를 call하면
독립적인 새 object이지만 instance variable의 값은 같은
하나의 object가 생성됩니다.

Example: Person.java (1/4)

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

    // default constructor
    public Person() { }

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

    // Copy Constructor
    public Person(Person person) {
        this.name = person.name;
        this.age = person.age;
    }
```


페이지 4

이제 copy constructor를 사용하는 예제 프로그램을 보겠습니다.
class Person에는 String name과 int age의
두 개의 instance variable이 있습니다.
default constructor는 아무일도 하지 않지만
권장하는 대로 하나 만들어 놓았구요.
name과 age의 초기값을 parameter로 받는
constructor도 하나 만들었습니다.
여기에서 this.name과 this.age는 instance variable
name과 age는 parameter를 뜻하고 있습니다.
이제 copy constructor인데
Person object 인 person을 parameter로 받아
새로 만들어지는 object의 instance variable들인
this.name과 this.age에
person.name과 person.age를 assign했습니다.

Example: Person.java (2/4)

```java
    // Accessors and Mutators
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

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


페이지 5

Instance variable들인 name과 age가 private이기 때문에
accessor와 mutator들이 필요합니다.
getName, getAge가 accessor이고
setName, setAge가 mutator들입니다.
toString method가 redefine 되어 있습니다.
Person object의 name과 age를 print하게 되어 있습니다.

Example: Person.java (3/4)

```
    public static void main(String[] args) {

        Person original = new Person("John Doe", 30);
        System.out.println("Original: " + original);

        // Using copy constructor
        Person copy = new Person(original);
        System.out.println("Copy: " + copy);

        // Test the equality between original and copy objects
        System.out.println("Are the objects equal? " + original.equals(copy));

        // Test the privacy leak
        copy.setName("Jane Doe");
        copy.setAge(25);
        System.out.println("Modified Copy: " + copy);
        System.out.println("Original after modifying copy: " + original);
    }
}
```


페이지 6

이제 main method를 보겠습니다.
먼저 "John Doe" 라는 name과 age 30을 가진
Person object인 original을 생성하였습니다.
Copy constructor를 사용하여
original을 copy하여 새 object인 copy를 생성하였습니다.
여기서 original.equals(copy)를 test하였는데
copy는 original과 내용은 같지만
다른 object로 새로 만들어졌기 때문에
다른 reference를 가집니다.
equals는 redefine하지 않으면
두 object의 reference가 같은지만 비교하기 때문에
false가 나오게 될 것입니다.
그 아래에는 copy object의 name과 age를 각각
"Jane Doe" 와 25로 mutator를 이용하여 변경하였습니다.
그리고 나서 copy와 original을
toString을 이용하여 print해 보게 되면
copy의 내용이 바뀌어도 original의 내용은 바뀌지 않은 것을
알 수 있습니다.
즉, copy constructor에 의해 새로운 object가 만들어졌기 때문에
privacy leak을 방지할 수 있었습니다.

Example: Person.java (4/4)

```
OUTPUT:

Original: Person{name='John Doe', age=30}
Copy: Person{name='John Doe', age=30}
Are the objects equal? false  // because the two objects' reference are different
Modified Copy: Person{name='Jane Doe', age=25}
Original after modifying copy: Person{name='John Doe', age=30}
// privacy leak prevented
```


페이지 7

이 slide는 앞의 프로그램의 output들을 보여주고 있습니다.

Example: Person2.java (1/7)

```java
public class Person2 {
    private String name;
    private int age;
    private Address address; // Person2 has an Address: class variable

    // class variable
    private static String country = "Unknown";

    // default constructor
    public Person2() { }

    // constructor
    public Person2(String name, int age, Address address) {
        this.name = name;
        this.age = age;
        this.address = address;
    }
```


페이지 8

이 프로그램 Person2.java에서는
shallow copy와 deep copy의 차이점을 보여주고 있습니다.
Person2 class에는 instance variable들로
String name과 int age가 있고
Address class type의 address가 있습니다.
static variable로 country가 있는데
초기값은 “Unknown” 으로 되어 있습니다.
country가 static으로 되어 있는 것으로 보아
아마도 이 Person2의 object들은
같은 국적을 가질 것으로 예상됩니다.
모든 object가 static variable인 country를
공유하게 되기 때문입니다.
역시 아무것도 하지 않지만
형식적인 default constructor를 하나 만들어 두었고
name, age, address의 세개의 parameter를 받아
instance variable들에 assign하는
constructor가 하나 있습니다.

Example: Person2.java (2/7)

```
    // Shallow Copy Constructor
    public Person2(Person2 person) {
        this.name = person.name;
        this.age = person.age;
        this.address = person.address; // Shallow copy (just copy the reference)
    }

    // Deep Copy Constructor
    public Person2 deepCopy(Person2 person) {
        return new Person2(person.name, person.age, new Address(person.address));
    }

    // Accessors and Mutators
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
```


페이지 9

이제 copy constructor를 define합니다.
name과 age는 이전과 같이
instance variable에 assign해 주면 됩니다.
그런데 마지막 line에서
parameter로 들어온 person object의 address를
그대로 새로 만들어지는 object의 address에 assign했네요.
이렇게 하면 person.address의 reference가
그대로 this.address로 assign되기 때문에
person.address와 this.address가
같은 memory location을 가리키게 됩니다.
이와 같이 reference variable을 그대로 assign하는 것을
shallow copy라고 합니다.
예상이 되지만 shallow copy 때문에
두 reference가 같은 memory location을 가리키게 되고
한쪽의 내용이 바뀐다면 다른 쪽의 내용도 바뀌기 때문에
privacy leak이 일어나게 됩니다.
그 아래는 constructor는 아니지만
deepCopy라는 method를 하나 만들어
parameter로 주어진 object의 내용을
deep copy하면서 새로운 object를 생성하여 return하게 했습니다.
따라서 return type은 Person2 입니다.
이 deepCopy method에서는
앞 slide에서 정의된
parameter 세개를 주는 constructor를 사용하고 있습니다.
new Person2(person.name, person.age, new Address(person.address)) 로
constructor를 call했는데
맨 마지막 parameter가 그냥 person.address 가 아니라
Address(person.address), 즉, Address class의 copy constructor를 call 한 것입니다.
이렇게 하면 person.address와 내용은 같으나
새로운 Address object가 생성되면서
그 reference를 parameter로 하기 때문에
deepCopy는 shallow copy가 아니라 deep copy version을 완성하게 되고
privacy leak도 방지하게 되는 것입니다.
그 아래에는 accessor와 mutator들이 있습니다.

Example: Person2.java (3/7)

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

public void setAge(int age) {
    this.age = age;
}

public Address getAddress() {
    return address;
}

public void setAddress(Address address) {
    this.address = address;
}

public static String getCountry() {
    return country;
}
```


페이지 10

accessor와 mutator들이 계속 정의되고 있습니다.
여기서 하나 눈여겨 보아야 할 점은
setAddress인데
만일 copy.setAddress(original.address); 로
mutator를 call했다면 privacy leak 문제가 발생할 수도 있습니다.
왜 그러할지는 여러분들이 한번 생각해 보시기 바랍니다.
또 어떻게 하면 방지할 수 있을지도 한번 생각해 보시기 바랍니다.
이 slide에는 class Address의 구현은 지면상 제약때문에 생략되어 있습니다.
class Address의 구현은 배포된 source code를 참조하기 바랍니다.

Example: Person2.java (4/7)

```java
public static void setCountry(String country) {
    Person2.country = country;
}

// toString
public String toString() {
    return "Person2{name='" + name + "', age=" + age + ", address=" + address
            + ", country=" + country + '}';
}

public static void main(String[] args) {
    // original object creation
    Address address = new Address("Seoul", "1234 Street");
    Person2 original = new Person2("John Doe", 30, address);
    System.out.println("Original: " + original);


    // shallow copy
    Person2 shallowCopy = new Person2(original);
    System.out.println("Shallow Copy: " + shallowCopy);
```


페이지 11

mutator인 setCountry가 define되어 있고
Person2의 toString이 define 되어 있습니다.
toString이 return하는 String의 중간에
", address=" + address 라는 부분이 있는데
여기서 뒤의 address는 Address class의
toString() 을 call하여 return되는 String이라는 것을
다시 강조합니다.
main method에서는 먼저 address object를 하나 생성하고
Person2 object인 original을 생성합니다.
미리 만들어 둔 address object가
Person2의 constructor의 parameter로 pass되는 것을
볼 수 있습니다.
그 아래에는 Person2의 copy constructor를 call하여
shallowCopy object를 생성하였습니다.

Example: Person2.java (5/7)

```
// Using Deep Copy Constructor
Person2 deepCopy = original.deepCopy(original);
System.out.println("Deep Copy: " + deepCopy);


// Test the equality of original and copy
System.out.println("Are the shallow copy objects equal?"+ original.equals(shallowCopy));
System.out.println("Are the deep copy objects equal? " + original.equals(deepCopy));


// Test the privacy leak
shallowCopy.setName("Jane Doe");
shallowCopy.setAge(25);
shallowCopy.getAddress().setCity("Busan");
shallowCopy.getAddress().setStreet("5678 Avenue");
System.out.println("Modified Shallow Copy: " + shallowCopy);
System.out.println("Original after modifying shallow copy: " + original);
```


페이지 12

original.deepCopy(original) 을 실행하여
deepCopy object를 생정하였습니다.
original.equals(shallowCopy) 를 실행해 보았는데
shallowCopy는 Person2 level에서 된 것이 아니라
그 안의 instance variable인 address가 shallow copy된 것이라
이 equals test는 false가 나오게 됩니다.
물론 그 아래의 original.equals(deepCopy) 도 false가 나오게 되죠.
이번에는 privacy leak을 test 해 보려 합니다.
shallowCopy의 name, age와 address를
모두 mutator를 이용하여 새로운 값으로 바꾸었습니다.
이때 shallowCopy와 origianal을 print 해 보면
original의 address도 함께 바뀐 것을 알 수 있습니다.
즉 privacy leak이 발생한 것이죠.

Example: Person2.java (6/7)

```
        deepCopy.setName("Alice Smith");
        deepCopy.setAge(28);
        deepCopy.getAddress().setCity("Incheon");
        deepCopy.getAddress().setStreet("91011 Boulevard");
        System.out.println("Modified Deep Copy: " + deepCopy);
        System.out.println("Original after modifying deep copy: " + original);

        // Test the class variable change
        Person2.setCountry("Korea");
        System.out.println("Original after changing country: " + original);
        System.out.println("Shallow Copy after changing country: " + shallowCopy);
        System.out.println("Deep Copy after changing country: " + deepCopy);
    }
}
```


페이지 13

이번에는 deepCopy object의 name, age, address를 바꾼 뒤
deepCopy와 original을 print 해 보았습니다.
deepCopy 이기 때문에 privacy leak이 방지 되어
deepCopy의 address를 바꾸더라도
original address는 바뀌지 않는 것을 알 수 있습니다.
그 아래에는 static 즉 class variable인 country를
"Korea" 로 바꾸어 보았는데
original, shallowCopy, deepCopy 의 세 object의 country가
모두 "Unknown" 에서 "Korea" 로 바뀐 것을 볼 수 있습니다.

Example: Person2.java (7/7)

OUTPUT:

```
Original: Person2{name='John Doe', age=30, address=Address{city='Seoul', street='1234 Street'}, country=Unknown}
Shallow Copy: Person2{name='John Doe', age=30, address=Address{city='Seoul', street='1234 Street'}, country=Unknown}
Deep Copy: Person2{name='John Doe', age=30, address=Address{city='Seoul', street='1234 Street'}, country=Unknown}
Are the shallow copy objects equal? false
Are the deep copy objects equal? false
Modified Shallow Copy: Person2{name='Jane Doe', age=25, address=Address{city='Busan', street='5678 Avenue'}, country=Unknown}
Original after modifying shallow copy: Person2{name='John Doe', age=30, address=Address{city='Busan', street='5678 Avenue'},
country=Unknown}
Modified Deep Copy: Person2{name='Alice Smith', age=28, address=Address{city='Incheon', street='91011 Boulevard'},
country=Unknown}
Original after modifying deep copy: Person2{name='John Doe', age=30, address=Address{city='Busan', street='5678 Avenue'},
country=Unknown}
Original after changing country: Person2{name='John Doe', age=30, address=Address{city='Busan', street='5678 Avenue'},
country=Korea}
Shallow Copy after changing country: Person2{name='Jane Doe', age=25, address=Address{city='Busan', street='5678 Avenue'},
country=Korea}
Deep Copy after changing country: Person2{name='Alice Smith', age=28, address=Address{city='Incheon', street='91011 Boulevard'},
country=Korea}
```


페이지 14

이 slide는 Person2.java 프로그램의 output을 보여주고 있습니다.
shallow copy와 deep copy의 차이 그리고 static variable의 변경 부분을 잘 살펴보시기 바랍니다.