clone 재정의는 주의해서 진행하라

이펙티브 자바 item13

유현우

목차

clone 메서드 사용법

Object 클래스에 protected clone( )

```java
public class Entry {
    String key;
    String value;

    public Entry(String key, String value) {
        this.key = key;
        this.value = value;
    }

    @Override
    protected Object clone() throws CloneNotSupportedException {
        return super.clone();
    }
}
```

```java
@Test
public void cloneTest() throws CloneNotSupportedException {
    // given
    Entry entry = new Entry("key", "value");
    // when
    Entry clone = (Entry) entry.clone();
    // then
    assertThat(entry.key).isEqualTo(clone.key);
    assertThat(entry.value).isEqualTo(clone.value);
}
```

```
java.lang.CloneNotSupportedException: com.example.test.item13.Entry

    at java.base/java.lang.Object.clone(Native Method)
    at com.example.test.item13.Entry.clone(Entry.java:18)
    at com.example.test.item13.EntryTest.cloneTest(EntryTest.java:14) <1 internal line>
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)
```

Clonable의 역할

```
package java.lang;

public interface Cloneable {
}
```

- 복제해도 되는 클래스임을 나타내는 믹스인 인터페이스이다.
- Cloneable 인터페이스는 clone( ) 메서드의 동작방식을 결정한다.
- Cloneable을 구현하지 않은 인스턴스에서 clone( )을 호출하면 CloneNotSupportedException을 던진다.

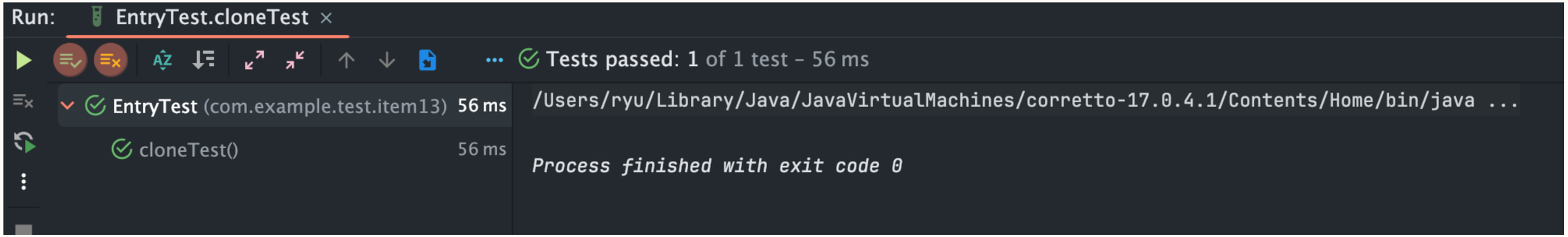
믹스인 은 객체 지향 언어에서 범용적으로 쓰이는 용어로, 다른 클래스들의 메서드 조합을 포함하는 클래스를 의미합니다.

clone 메서드 사용법

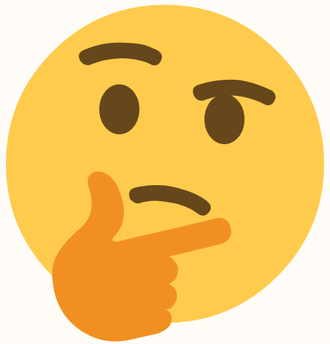
Cloneable을 구현하면 예외가 발생하지 않는다.

```
public class Entry implements Cloneable {
    ...

    @Override
    protected Object clone() throws CloneNotSupportedException {
        return super.clone();
    }
}
```



checked Exception 보다는 Unchecked Exception으로 만들어야 했다.



Clonable이 왜 필요하지..?

Cloneable은 사용하지 말라고요?

— 복사 생성자 와 복사 팩터리는 조금 있다가~

6 Answers

Sorted by: Highest score (default)

173

The first thing you should know about `Cloneable` is - don't use it.

It is very hard to implement cloning with `Cloneable` right, and the effort is not worth it.

Instead of that use some other options, like apache-commons `SerializationUtils` (deep-clone) or `BeanUtils` (shallow-clone), or simply use a copy-constructor.

See here for the views of Josh Bloch about cloning with `Cloneable`, which explains the many drawbacks of the approach. (Joshua Bloch was a Sun employee, and led the development of numerous Java features.)

Clonable의 문제점

- 믹스인으로 의도해서 만들었는데 일반적인 인터페이스의 동작방식과 다르게 상위 클래스에 메서드를 오버라이드 해야 한다. (Clonable에 clone 메서드가 존재하지 않는다.)
- Cloneable만 사용하면 당연히 복제가 이뤄질 줄 알았는데 생각보다 복잡한 구조를 이해하고 있어야 한다.
- 자바의 기본 의도와 다르게 생성자를 호출하지 않고 객체를 생성할 수 있게 되어버린다.
- clone( ) 메서드의 일반 규약은 약간 허술하다.

clone() 메서드의 일반 규약

x.clone() != x (true)

- 복사된 객체가 원본이랑 같은 주소를 가지면 안된다.

x.clone().getClass() == x.clone().getClass() (true)

- 복사된 객체가 같은 클래스여야 한다.

x.clone().equals() (true가 필수 X)

- 복사된 객체가 논리적 동치는 일치해야 한다는 뜻이다. (필수 X)

clone() 메서드의 문제점

1. super.clone()을 사용하지 않을 때 문제
2. 가변 상태를 참조하는 클래스의 clone() 문제

super.clone()을 사용하지 않을 때 문제

clone() 메서드의 문제점

```java
public class Parent implements Cloneable {
    private String field;

    public Parent(String field) {
        this.field = field;
    }

    @Override
    protected Parent clone() {
//        return new Child(field); 통과
        return new Parent(field); // java.lang.ClassCastException 발생
    }
}

class Child extends Parent implements Cloneable {
    public Child(String field) {
        super(field);
    }

    @Override
    protected Child clone() {
        return (Child) super.clone();
    }
}
```

```java
public static void main(String[] args) {
        Parent parent = new Parent("hi");
        Parent cloneParent = parent.clone();
        // class com.example.test.item13.Parent
        System.out.println(cloneParent.getClass());

        Child child = new Child("hello");
        Child cloneChild = child.clone();
        // class com.example.test.item13.Child
        System.out.println(cloneChild.getClass());
}
```

해결 방법

super.clone()을 사용하지 않을 때 문제

상속 못 하게 하기

```
public final class Parent implements Cloneable {
    @Override
    protected Parent clone() {
        try {
            return (Parent) super.clone();
        } catch (CloneNotSupportedException e) {
            throw new RuntimeException(); // UnCheckedException
        }
    }
}
```

상위 부모에서 재정의 못 하게 하기

```
public class Parent implements Cloneable {
    private String field;

    public Parent(String field) {
        this.field = field;
    }

    @Override
    final protected Parent clone() {
        try {
            return (Parent) super.clone();
        } catch (CloneNotSupportedException e) {
            throw new RuntimeException(); // UnCheckedException
        }
    }
}

class Child extends Parent implements Cloneable {
    public Child(String field) {
        super(field);
    }
}
```

```
Child child = new Child("hello");
Child clone = (Child) child.clone();
```

해결 방법

super.clone()을 사용하지 않을 때 문제

super.clone() 사용하기

```java
public class Parent implements Cloneable {
    private String field;

    public Parent(String field) {
        this.field = field;
    }

    @Override
    protected Parent clone() {
        try {
            return (Parent) super.clone();
        } catch (CloneNotSupportedException e) {
            throw new RuntimeException(); // UnCheckedException
        }
    }
}

class Child extends Parent implements Cloneable {
    public Child(String field) {
        super(field);
    }

    @Override
    protected Child clone() {
        return (Child) super.clone();
    }
}
```

가변 객체를 참조하는 클래스의 clone() 문제

clone() 메서드의 문제점

```
public class Stack implements Cloneable{
    private Object[] elements;

    ...

    @Override
    protected Stack clone() {
        try {
            return (Stack) super.clone();
        } catch (CloneNotSupportedException e) {
            throw new RuntimeException();
        }
    }

}
```

```
public static void main(String[] args) {
    Stack stack = new Stack();
    stack.push("test");
    Stack clone = stack.clone();
    clone.pop();
    System.out.println(stack); // size = 1
    System.out.println(clone);
}
```

```
Stack{elements=[null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null], size=1}
Stack{elements=[null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null, null], size=0}

Process finished with exit code 0
```

가변객체란, instance 생성 이후에도 내부 상태 변경이 가능한 객체를 말한다.

가변 객체를 참조하는 클래스의 clone() 문제

clone() 메서드의 문제점

```java
public class MyPointer implements Cloneable{
    private Point point;

    public MyPointer(Point point) {
        this.point = point;
    }

    public void updatePoint(int x, int y) {
        point.setX(x);
        point.setY(y);
    }

    @Override
    protected MyPointer clone() {
        try {
            MyPointer clone = (MyPointer) super.clone();
            return clone;
        } catch (CloneNotSupportedException e) {
            throw new RuntimeException();
        }
    }
}
```

```java
@Setter
class Point {
    int x;
    int y;

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
```

```java
public static void main(String[] args) {
        MyPointer pointer = new MyPointer(new Point(1, 3));
        MyPointer clone = pointer.clone();
        clone.updatePoint(3, 1);
        System.out.println(pointer); // Stack{point=Point{x=3, y=1}}
        System.out.println(clone); // Stack{point=Point{x=3, y=1}
    }
```

해결 방법

primitive 타입(할당)의 값은 올바르게 복제되지만 reference 타입(참조)의 값은 동일한 주소 값을 가지게 된다.

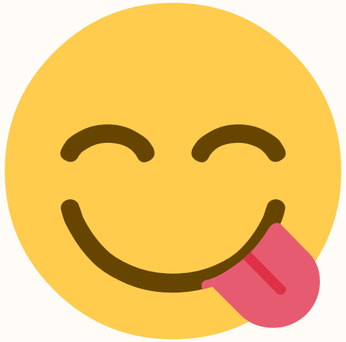
- 반복문이든 재귀든 객체를 연쇄적으로 계속 같은 값을 지닌 새로운 객체로 복사해줘야 한다.

```
@Override
protected Stack clone() {
    try {
        Stack clonedStack = (Stack) super.clone();
        clonedStack.elements = this.elements.clone();
        return clonedStack;
    } catch(CloneNotSupportedException e) {
        throw new RuntimeException();
    }
}
```

elements가 final로 선언되어 있었다면 clonedStack.elements = this.elements.clone(); 이 구문을 실행할 수 없다. 즉, 가변 객체를 참조하는 필드는 final로 선언하라는 일반 용법과 충돌한다.

clone() 메서드 주의사항

- Object.clone( )은 동기화를 신경쓰지 않았기 때문에 **동시성 문제**가 발생할 수 있다.
- **기본 타입이나 불변 객체 참조**만 가지면 아무것도 **수정**할 필요가 없지만 **일련번호 혹은 고유 ID**와 같은 값을 가지고 있다면, 비록 불변일지라도 **새롭게 수정**해주어야 할 것이다.



복사 생성자와 복사 팩터리를 사용하자

복사 생성자

복사 생성자와 복사 팩터리를 사용하자

복사 생성자는 기존 개체의 복사본으로 새 개체를 만들기 위한 특수 생성자로 클래스 개체를 복사할 때 컴파일러에서 수행하는 작업을 정의하면 된다.

- 많은 필드가 있는 복잡한 개체에서 복사 생성자를 사용하는 것이 훨씬 더 간단하다.
- 복사 생성자는 우리에게 구현을 강요하지 않는다. (Cloneable 또는 Serializable)

복사 생성자

복사 생성자와 복사 팩터리를 사용하자

```
class Main
{
    public static void main(String[] args)
    {
        Student student =
          new Student("Jon Snow", 22, new HashSet<String>(
                                Arrays.asList("Maths", "Science", "English"))
                            );

        // 복사 생성자 호출
        Student s = new Student(student);
        System.out.println("Using Copy Constructor: " + s.toString());

        // s의 지도에 대한 변경 사항은 학생의 지도에 반영되지 않습니다.
        s.getSubjects().add("History");
        System.out.println(student.getSubjects());
    }
}
```

```
Using Copy Constructor: [Jon Snow, 22, [Maths, English, Science]]
[Maths, English, Science]


Process finished with exit code 0
```

```
class Student
{
    private String name;
    private int age;
    private Set<String> subjects;

    public Student(String name, int age, Set<String> subjects)
    {
        this.name = name;
        this.age = age;
        this.subjects = subjects;
    }

    // 복사 생성자
    public Student(Student student)
    {
        this.name = student.name;
        this.age = student.age;

        // 얕은 복사
        this.subjects = student.subjects;
        // 깊은 복사 - `HashSet`의 새 인스턴스 생성
        this.subjects = new HashSet<>(student.subjects);
    }
}
```

복사 팩터리

복사 생성자와 복사 팩터리를 사용하자

정적 복사 팩토리 메서드를 사용하여 본질적으로 복사 생성자 메서드와 동일한 작업을 수행할 수 있다.

```
class Student
{
    ...
    // 복사 생성자
    public Student(Student student)
    {
        this.name = student.name;
        this.age = student.age;
        // deep copy
        this.subjects = new HashSet<>(student.subjects);
    }

    // 팩토리 복사
    public static Student newInstance(Student student) {
        return new Student(student);
    }
}
```

```
public static void main(String[] args)
    {
        Student student = ...

        // 복사 팩토리 사용
        Student s = Student.newInstance(student);
        System.out.println("using copy factory: " + s);

        // s의 지도에 대한 변경 사항은 학생의 지도에 반영되지 않습니다.
        s.getSubjects().add("History");
        System.out.println(student.getSubjects());
    }
}
```

결론

Clone을 사용할거면

- super.clone()을 사용하자
- 가변 상태를 참조하는 클래스인 경우 반복문으로 같은 값을 지닌 새로운 객체로 복사해줘야 한다.
- 일련번호 혹은 고유 ID와 같은 값을 가지고 있다면 불변일지라도 새롭게 수정해서 복사해야 한다.

복사 생성자와 복사 팩터리를 사용하자

📚Reference

stackoverflow: https://stackoverflow.com/questions/4081858/how-does-cloneable-work-in-java-and-how-do-i-use-it

checked, unChecked: https://devlog-wjdrbs96.tistory.com/351

복사 생성자와 복사 팩터리: https://www.techiedelight.com/ko/copy-constructor-factory-method-java/

이펙티브 자바 아이템 13: https://sysgongbu.tistory.com/172

Thank you

궁금한 점을 물어보세요