Chapter 4

Class & Interface

발 표 자

한 주 연

목차

Review

CONTENTS

01 정보 은닉

정보 은닉
(캡슐화)

모든 클래스의 멤버의 접근성을 가능한 한 좁혀야 한다.

* Item 15. 접근 권한 최소화
* Item 16. public 클래스
* Item 17. 불변 클래스

Item 15

01 정보 은닉의 방법

정보 은닉
(캡슐화)

모든 클래스의 멤버의 접근성을 가능한 한 좁혀야 한다.

* Access Modifier
  - private
  - package-private(default)
  - protected
  - public

] 공개 API

* Inner Class

* Module System

Item. 16

01 Public Class

불변성이 깨진 클래스

```
public static class C {
    // 가변 인스턴스
    public static final int[] arr = {1,2,3,4,5};
    public char ch = 'c';
    // 가변 객체
    public List<Integer> list = new ArrayList<Integer>();
}

public static class D {
    C c = new C();

    public void changeC() {
        // 객체 C의 불변성 보장 실패
        c.arr[3] = 6;
        c.ch = 'd';
        c.list.add(11);
    }
}
```

Item. 16

01 Public Class

좋은 예

```
public static class A {
    private int a;
    private List<B> b;
    private static final int[] arr = {1,2,3,4,5};

    private static class B {    //private static으로 중첩
        private int b;
    }

    public int getA() {
        return a;
    }

    public B getB(int idx) {
        return b.get(idx);
    }

    public static final int[] getArr() {
        return arr.clone();
    }
}
```

Item. 17

01 불변 클래스

완벽한 불변 클래스

```java
// 2번 : 상속 불가
final public class InvariantClass{
    // 3,4번 : 모든 필드는 final private
    private int num;
    final private char[] charArr;

    // 2번 : 생성자를 private, 정적 팩터리 메서드 제공
    public final InvariantClass instance = new InvariantClass();
    private InvariantClass() {
        num = 10;
        charArr = new char[10];
    }
    public InvariantClass getInstance() {
        return instance;
    }
    // 5번 : 가변 객체의 접근을 막는다
    public int getNum() {
        return num;
    }
    // 5번 : 방어적 복사를 수행한다
    public char[] getCharArr() {
        return charArr.clone();
    }
    // 1번 : 변경자를 제공하지 않는다
    private void setNum(int num) {
        this.num = num;
    }
}
```

02 Inheritance

상속

상속은 하위 클래스의 캡슐화를 깨뜨린다.

* Item 18. Composition - √
* Item 19. 상속의 문서화
* Item 20. 추상 골격 구현 - √
  - Abstract
  - Interface
* Item 21. Default 메소드
* Item 22. Interface의 올바른 용도 - √

Item. 18

02 Inheritance

상속에 의해 캡슐화가 깨짐

```
public class WrongSet<E> extends HashSet<E> {
      private int addCount = 0; // 추가된 원소의 개수

      @Override
      public boolean add(E e) {
          addCount++;
          return super.add(e);
      }

      @Override
      public boolean addAll(Collection<? extends E> c) {
          addCount = addCount + c.size();
          return super.addAll(c);    // add() 호출!
      }

      public int getAddCount() {
          return addCount;
      }
}
```

```
public boolean addAll(Collection<? extends E> c) {
    boolean modified = false;
    for (E e : c)
        if (add(e))
            modified = true;
    return modified;
}
```

Item. 18

02 Inheritance

Composition

```
public class GoodSet<E> extends ForwardingSet<E>  {
    private int addCount = 0;

    public GoodSet(Set<E> set) {
        super(set);
    }

    @Override
    public boolean add(E e) {
        addCount++;
        return super.add(e);
    }

    @Override
    public boolean addAll(Collection<? extends E> collection) {
        addCount = addCount + collection.size();
        return super.addAll(collection);
    }

    public int getAddCount() {
        return addCount;
    }
}
// 전달 클래스
public class ForwardingSet<E> implements Set<E> {
    private final Set<E> set;          // 상속 대신 인스턴스를 참조한다.
    public ForwardingSet(Set<E> set) { this.set = set; }
    public void clear() { set.clear(); }
    public boolean isEmpty() { return set.isEmpty(); }
    public boolean add(E e) { return set.add(e); }
    public boolean addAll(Collection<? extends E> c) { return set.addAll(c); }
    // ... 생략
```

Item. 20

02 Inheritance

Abstract

* 다른 클래스들에게서 공통으로 가져야하는 메소드들의 집합
* 단일 상속
* 구현 클래스는 반드시 하위 클래스가 되야 하는 제약(공통 조상)

Interface

* 추상 메서드들의 집합
* 다중 상속
* 추상 클래스에 비해 상속에 자유롭다(메서드만 정의)
* 믹스인

Item. 20

02 Inheritance

Interface

```
public interface Singer {
    AudioClip sing(Song song);
}
public interface SongWiter {
    Song compose(int chartPosition);
}
// 2개의 인터페이스를 확장하여 제 3의 인터페이스를 정의한다.
public interface SingerSongWiter extends Singer, SongWiter {
    AudioClip strum();
    void actSensitive();
}
```

Abstract

```
public abstract class Singer {
    abstract void sing(String s);
}
public abstract class SongWriter {
    abstract void compose(int chartPosition);
}

public abstract class SingerSongWriter {
    abstract void strum();
    abstract void actSensitive();
    abstract void Compose(int chartPosition);
    abstract void sing(String s);
}
```

Item. 20

02 Inheritance

Interface

```
public interface Car{
    void start();
    void accelPedal();
    void breakPedal();
    void stop();
    void process();
}
```

중복되는 메서드

```
public class Sonata implements Car{
    int speed = 0;
    @Override
    public void start() {
        System.out.println("start Engine");
    }

    @Override
    public void accelPedal() {
        speed += 10;
    }

    @Override
    public void breakPedal() {
        speed -= 10;
    }

    @Override
    public void stop() {
        System.out.print("Stop Engine");
    }

    @Override
    public void process() {
        start();
        accelPedal();
        breakPedal();
        stop();
    }
}
```

```
public class Grandeur implements Car{
    int speed = 0;
    @Override
    public void start() {
        System.out.println("start Engine");
    }

    @Override
    public void accelPedal() {
        speed += 20;
    }

    @Override
    public void breakPedal() {
        speed -= 20;
    }

    @Override
    public void stop() {
        System.out.print("Stop Engine");
    }

    @Override
    public void process() {
        start();
        accelPedal();
        breakPedal();
        stop();
    }
}
```

Item. 20

02 Inheritance

추상 골격 구현 클래스

```
// 추상 골격 구현 클래스
public abstract class AbstractCar implements Car{
    int speed = 0;
    @Override
    public void start() {
        System.out.println("start Engine");
    }
    @Override
    public void stop() {
        System.out.print("Stop Engine");
    }
    @Override
    public void process() {
        start();
        accelPedal();
        breakPedal();
        stop();
    }
}
```

구현

```
public class Sonata extends AbstractCar implements Car{
    @Override
    public void accelPedal() {
        speed += 10;
    }

    @Override
    public void breakPedal() {
        speed -= 10;
    }
}

public class Grandeur extends AbstractCar implements Car{
    @Override
    public void accelPedal() {
        speed += 20;
    }

    @Override
    public void breakPedal() {
        speed -= 20;
    }
}
```

Item. 20

02 Inheritance

자동차 제조사

```
public class CarManufacturer{
    public void ManufacturerName() {
        System.out.println("hyundai");
    }
}
```

Inner Class를 통한
추상 골격 구현

```
public class genesis extends CarManufacturer implements Car{
    InnerAbstractCar iac = new InnerAbstractCar();
    @Override
    public void start() {
        iac.start();            // 추상 골격 클래스에서 구현한 메서드 호출
    }

    @Override
    public void accelPedal() {
        iac.accelPedal();
    }

    @Override
    public void breakPedal() {
        iac.breakPedal();
    }

    @Override
    public void stop() {
        iac.stop();
    }

    @Override
    public void process() {
        ManufacturerName();     // 상속받은 클래스의 메서드 사용
        iac.process();
    }
    // Car에서 중 미구현 된 메서드를 구현한다.
    private class InnerAbstractCar extends AbstractCar{

        @Override
        public void accelPedal() {
            speed += 30;
        }

        @Override
        public void breakPedal() {
            speed -= 30;
        }
    }
}
```

Item. 22

02 Inheritance

Interface의 올바른 용도

Interface의 용도

인터페이스는 자신을 구현한 클래스의 인스턴스를 참조할 수 있는 **타입**의 역할을 한다.

```java
public class item22_ex {
    public interface A{
        public void B();
    }
    public static class AA implements A{
        @Override
        public void B() {
            System.out.println("hi");
        }
    }
    public static void main(String[] args) {
        A a = new AA();   // 클래스 AA의 타입으로 interface A가 사용되었다.
        a.B();
    }
}
```

03 그 외의 주의사항

클래스 사용 관례

클래스 사용의 관례와 방법

* Item 23. 태그 클래스 사용 금지
* Item 24. 중첩 클래스
  - 정적 멤버 클래스
  - 비정적 멤버 클래스
  - 익명 클래스
  - 지역 클래스
* Item 25. 톱 레벨 클래스

감사합니다

THANK YOU

JooYeon Han
한 주 연