Chapter 17. abstract와 interface 그리고 inner class

17-1. abstract 클래스

■ 인스턴스의 생성을 허용 안 하는 abstract 클래스


```
class Friend
{
    . . . . // 앞부분 생략
    public void showData()
    {
        System.out.println("이름 : "+name);
        System.out.println("전화 : "+phoneNum);
        System.out.println("주소 : "+addr);
    }
    public void showBasicInfo() { }
}
```

텅 빈 정의!

앞서 상속 관련 예제에서 정의한 Friend 클래스! 이 클래스는 UnivFriend와 HighFriend를 상속의 관계로 연결하기 위해 정의한 클래스 즉, 인스턴스화에 목적이 없다! 달리 말해서 인스턴스화 된다면, 이는 실수다!

추상화! 인스턴스 생성을 막음

```
abstract class Friend
{
    . . . .
    public void showData()
    {
        System.out.println("이름 : "+name);
        System.out.println("전화 : "+phoneNum);
        System.out.println("주소 : "+addr);
    }
    public abstract void showBasicInfo();
}
```

하나 이상의 메소드가 abstract면, 클래스도 abstract

메소드를 완성시키지 않는다는 선언

showBasicInfo 메소드는 비어있었다. 이렇듯 오버라이딩의 관계 유지를 목적으로 하는 메소드는 abstract로 선언이 가능하다.

- 하나 이상 abstarct 메소드를 포함하는 클래스는 abstract로 선언되어야 하며, 인스턴스 생성은 불가!
- 인스턴스 생성은 불가능하나, 참조변수 선언 가능하고, 오버라이딩의 원리 그대로 적용됨!

■ abstract 클래스를 상속하는 하위 클래스

나 정말 JAVA를 공부한 적 없는데요

```
abstarct class AAA
{
    void methodOne() { . . . }
    abstract void methodTwo();
}

class BBB extends AAA
{
    void methodThree() { . . . }
}
```

그대로 하위 클래스로 내려오는 꼴이 된다!

컴파일 에러 발생 BBB 클래스도 abstract 로 선언되어야 에러 발생 않는다!

위의 경우BBB 클래스는 AAA 클래스의 abstract 메소드를 상속한다. 그런데 오버라이딩 하지 않았으므로, abstract 상태 그대로 놓이기 된다. 결국 BBB 클래스는 하나 이상의 abstract 메소드를 포함한 셈이니, abstract로 선언되어야 하며, 인스턴스의 생성도 불가능하게 된다.

17-2. interface

■ 문제의 상황

나 정말 JAVA를 공부한 적 없는데요

프로젝트 담당자인 홍만군의 요구사항 1.

- 이름과 주민등록 번호를 저장하는 기능의 클래스가 필요하다.
- 이 클래스에는 주민등록 번호를 기반으로 사람의 이름을 찾는 기능이 포함되어야 한다.

프로젝트 담당자인 홍만군의 요구사항 2.

- 주민등록번호와 이름의 저장 → void addPersonalInfo(String perNum, String name)
- 주민등록번호를 이용한 검색 → String searchName(String perNum)

홍만군이 생각한 프로젝트 진행의 문제점

➡ 문제 1

"나도 프로젝트를 진행해야 하는데, A사가 클래스를 완성할 때까지 기다리고만 있을 수는 없잖아! 그리고 나중에 내가 완성한 결과물과 A사가 완성한 결과물을 하나로 묶을 때 문제가 발생하면 어떻게 하지? A사와 나 사이에 조금 더 명확한 약속이 필요할 것 같은데"

➡ 문제 2

"내가 요구한 기능의 메소드들이 하나의 클래스에 담겨있지 않으면 어떻게 하지? A사에서 몇 개의 클래스로 기능을 완성하건, 나는 하나의 인스턴스로 모든 일을 처리하고 싶은데! 무엇보다 나는 A사가 완성해 놓은 기능들을 활용만 하고 싶다고! 어떻게 구현했는지는 관심 없다고!"

■ 인터페이스의 정의

나 정말 JAVA를 공부한 적 없는데

홍만군이 판단한 해결책!

"클래스를 하나 정의해야겠다. 그리고 A사에는 이 클래스를 상속해서 기능을 완성해 달라고 요구하고, 난 이 클래스를 기준으로 프로젝트를 진행해야겠다!"

문제의 해결을 위해서 정의한 클래스, A사에 전달!

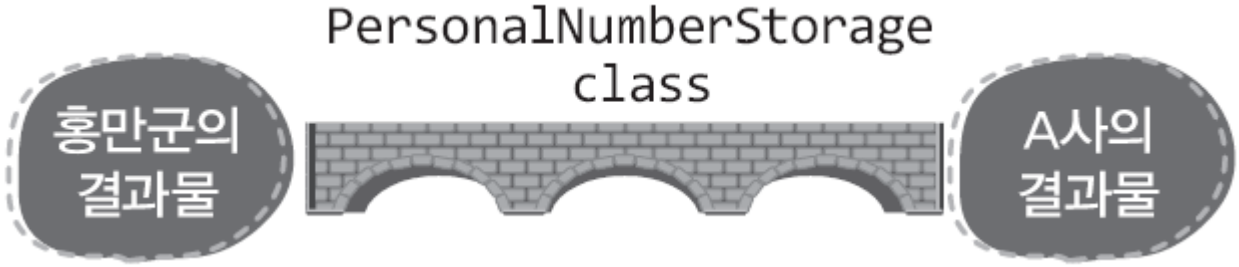
```
abstract class PersonalNumberStorage
{
    public abstract void addPersonalInfo(String perNum, String name);
    public abstract String searchName(String perNum);
}
```

```
class AbstractInterface
{
    public static void main(String[] args)
    {
        PersonalNumberStorage storage=new (A사가 구현할 클래스 이름);
        storage.addPersonalInfo("김기동", "950000-1122333");
        storage.addPersonalInfo("장산길", "970000-1122334");

        System.out.println(storage.searchName("950000-1122333"));
        System.out.println(storage.searchName("970000-1122334"));
    }
}
```

클래스 PersonalNumberStorage의 정의로 인해서 왼쪽의 형태로 홍만군은 프로젝트를 완료할 수 있게 되었다.

A사의 프로젝트 진행에 상관없이 말이다!

■ 홍만군의 사례로 본 인터페이스에 대한 고찰



- ✓ 클래스 PersonalNumberStorage는 인터페이스의 역할을 하는 클래스이다.
- ✓ 인터페이스는 두 결과물의 연결고리가 되는 일종의 약속 역할을 한다.
- ✓ 인터페이스의 정의로 인해서 홍만군은 홍만군대로,
  A사는 A사대로 더 이상의 추가 논의 없이 프로젝트를 진행할 수 있었다.
- ✓ 인터페이스의 정의되었기 때문에 프로젝트를 하나로 묶는 과정도 문제가 되지 않는다.

interface의 활용


```
abstract class PersonalNumberStorage
{
    public abstract void addPersonalInfo(String perNum, String name);
    public abstract String searchName(String perNum);
}
```

모든 메소드가 abstract로 선언된 abstract 클래스는 다음과 같이 정의 가능하다!

```
interface PersonalNumberStorage
{
    void addPersonalInfo(String perNum, String name);
    String searchName(String perNum);
}
```

interface로 선언되는 클래스는 다음의 특징을 지니는 특별한 유형의 클래스!

- ✓ 인터페이스 내에 선언된 변수는 무조건 public static final로 선언된다.
- ✓ 인터페이스 내에 선언된 메소드는 무조건 public abstract로 선언된다.
- ✓ 인터페이스도 참조변수 선언 가능하고, 메소드 오버라이딩 원칙 그대로 적용된다!

interface의 특성

```
public interface MyInterface
{
    public void myMethod();
}

public interface YourInterface
{
    public void yourMethod();
}
```

```
Class OurClass implements MyInterface, YourInterface
{
    public void myMethod() { . . . }
    public void yourMethod() { . . . }
}
```

인터페이스는 둘 이상을 동시에 구현 가능

인터페이스의 상속(구현)은 extends가 아닌 implements를 사용한다.

```
public interface SuperInterf
{
    public void supMethod();
}

public interface SubIntef extends SuperInterf
{
    public void subMethod();
}
```

인터페이스 간 상속 가능 단 이 때는

implements가 아닌 extends를 사용한다.

interface 기반의 상수표현

```
public class Week
{
    public static final int MON=1;
    public static final int TUE=2;
    public static final int WED=3;
    public static final int THU=4;
    public static final int FRI=5;
    public static final int SAT=6;
    public static final int SUN=7;
}
```

인터페이스 내에 선언된 변수는 무조건 public static final로 선언이 되므로,

이 둘은 완전히 동일한 의미를 갖는다.

```
public interface Week
{
    int MON=1, TUE=2, WED=3, THU=4, FRI=5, SAT=6, SUN=7;
}
```

interface 기반의 상수표현 예제

```
interface Week
{
    int MON=1, TUE=2, WED=3, THU=4, FRI=5, SAT=6, SUN=7;
}
```

```
public static void main(String[] args)
{
    . . . . . .
    switch(sel)
    {
    case Week.MON :
        System.out.println("주간회의가 있습니다.");
        break;
    case Week.TUE :
        System.out.println("프로젝트 기획 회의가 있습니다.");
        break;
    case Week.WED :
        System.out.println("진행사항 보고하는 날입니다.");
        break;
    case Week.THU :
        System.out.println("사내 축구시합이 있는 날입니다.");
        break;
    case Week.FRI :
        System.out.println("프로젝트 마감일입니다.");
        break;
    case Week.SAT :
        System.out.println("가족과 함께 즐거운 시간을 보내세요");
        break;
    case Week.SUN :
        System.out.println("오늘은 휴일입니다.");
    }
}
```

자바 interface의 또 다른 가치

```
interface UpperCasePrintable
{
    // 비어 있음
}

class ClassPrinter
{
    public static void print(Object obj)
    {
        String org=obj.toString();
        if(obj instanceof UpperCasePrintable)
        {
            org=org.toUpperCase();
        }

        System.out.println(org);
    }
}
```

UpperCasePrintable의 성격을 표시하는 용도!

```
class PointOne implements UpperCasePrintable
{
    private int xPos, yPos;

    PointOne(int x, int y)
    {
        xPos=x;
        yPos=y;
    }
    public String toString()
    {
        String posInfo="[x pos : "+xPos + ", y pos : "+yPos+"]";
        return posInfo;
    }
}
```

✓무엇인가를 표시하는(클래스의 특성을 표시하는) 용도로도 인터페이스는 사용된다.

✓ 이러한 경우, 인터페이스의 이름은 ~able로 끝나는 것이 보통이다.

✓ 이러한 경우, 인터페이스는 비어 있을 수도 있다.

✓ instanceof 연산자를 통해서 클래스의 특성을 파악할 수 있다.

interface를 통한 다중상속의 효과

```
public static void main(String[] args)
{
    IPTV iptv=new IPTV();
    iptv.powerOn();

    TV tv=iptv;
    Computer comp=iptv;
}
```

이 부분만 놓고 보면 IPTV 클래스가 TV 클래스를, 그리고 Computer 클래스를 동시에 상속하고 있는 것처럼 보인다. 그러나 자바는 다중상속을 지원하지 않는다!

```
class TV
{
    public void onTV()
    {
        System.out.println("영상 출력 중");
    }
}

interface Computer
{
    public void dataReceive();
}

class ComputerImpl
{
    public void dataReceive()
    {
        System.out.println("영상 데이터 수신 중");
    }
}
```

```
class IPTV extends TV implements Computer
{
    ComputerImpl comp=new ComputerImpl();

    public void dataReceive()
    {
        comp.dataReceive();
    }
    public void powerOn()
    {
        dataReceive();
        onTV();
    }
}
```

실제로는, 인터페이스를 통해서 다중상속이 된 것과 같은 효과를 보이고 있다.

17-3. Inner 클래스

Inner 클래스와 Nested 클래스


```
class OuterClass
{
    . . . .
    class InnerClass
    {
        . . . .
    }
}
```

왼쪽에서와 같이 클래스의 정의가 다른 클래스의 내부에 삽입될 수 있다. 이 때 외부의 클래스를 가리켜 **Outer 클래스**라 하고, 내부의 클래스를 가리켜 **Inner 클래스**라 한다.

```
class OuterClass
{
    . . . .
    static class NestedClass
    {
        . . . .
    }
}
```

Inner 클래스의 형태에 static 선언이 삽입되면, 이를 가리켜 **static Inner 클래스** 또는 간단히 **Nested 클래스**라 한다.

■ 예제를 통한 Nested 클래스의 이해

나 정말 JAVA를 공부한 적 없는데

```
class OuterClassOne
{
    OuterClassOne()
    {
        NestedClass nst=new NestedClass();
        nst.simpleMethod();
    }
    static class NestedClass
    {
        public void simpleMethod()
        {
            System.out.println("Nested Instance Method One");
        }
    }
}
```

클래스 내부에서는 직접 생성 가능

클래스 외부에서 이 클래스의 이름은 OuterClassOne.NestedClass 가 된다!

```
public static void main(String[] args)
{
    OuterClassOne one=new OuterClassOne();

    OuterClassOne.NestedClass nst1=new OuterClassOne.NestedClass();
    nst1.simpleMethod();
}
```

Nested 클래스의 인스턴스 생성방법

NestedClass가 private으로 선언되면, 선언된 클래스 내부에서만 인스턴스를 생성할 수 있다.

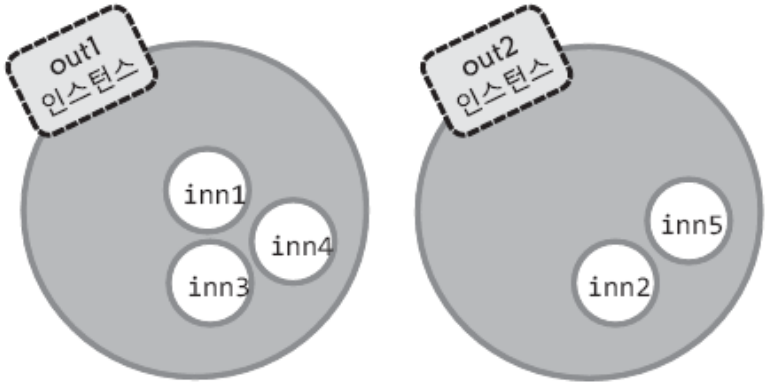
■ 예제를 통한 Inner 클래스의 이해


```
class OuterClass
{
    private String myName;
    private int num;

    OuterClass(String name)
    {
        myName=name;
        num=0;
    }

    public void whoAreYou()
    {
        num++;
        System.out.println(myName+ " OuterClass "+num);
    }

    class InnerClass
    {
        InnerClass()
        {
            whoAreYou();
        }
    }
}
```

```
public static void main(String[] args)
{
    OuterClass out1=new OuterClass("First");
    OuterClass out2=new OuterClass("Second");
    out1.whoAreYou();
    out2.whoAreYou();

    OuterClass.InnerClass inn1=out1.new InnerClass();
    OuterClass.InnerClass inn2=out2.new InnerClass();
    OuterClass.InnerClass inn3=out1.new InnerClass();
    OuterClass.InnerClass inn4=out1.new InnerClass();
    OuterClass.InnerClass inn5=out2.new InnerClass();
}
```

Inner 클래스의 인스턴스는

Outer 클래스의 인스턴스에 종속적이다!



- Outer 클래스의 인스턴스 생성 후에야 Inner 클래스의 인스턴스 생성이 가능하다.
- Inner 클래스 내에서는 Outer 클래스의 멤버에 직접 접근이 가능하다.
- Inner 클래스의 인스턴스는 자신이 속할 Outer 클래스의 인스턴스를 기반으로 생성된다.

Inner 클래스의 성격

17-4. Local 클래스와 Anonymous 클래스

Local 클래스

나 정말 JAVA를 공부한 적 없는데

- Outer 클래스의 인스턴스 생성 후에야 Inner 클래스의 인스턴스 생성이 가능하다.
- Inner 클래스 내에서는 Outer 클래스의 멤버에 직접 접근이 가능하다.
- Inner 클래스의 인스턴스는 자신이 속할 Outer 클래스의 인스턴스를 기반으로 생성된다.

Inner 클래스의 성격을 그대로 유지한다!

Local 클래스는 메소드 내에 정의가 되어서, 메소드 내에서만 인스턴스의 생성 및 참조 변수의 선언이 가능하다는 특징이 있다!

```
class OuterClass
{
   . . . .
   public LocalClass createLocalClassInst( )
   {
      class LocalClass
      {
         . . . .
      }

      return new LocalClass( );
   }
}
```

문제 없는 반환형인가?

Local 클래스

왼쪽에 정의된 LocalClass 클래스는 로컬 클래스이다! 그러나 반환형의 선언이 문제가 된다. 반환하는 로컬 클래스를 외부에서 참조할 수 없기 때문이다. 참조변수의 선언은 클래스가 정의된 메소드 내에서만 가능하므로...

Local 클래스의 적절한 사용 모델


```
interface Readable
{
    public void read();
}

class OuterClass
{
    private String myName;

    OuterClass(String name)
    {
        myName=name;
    }
    public Readable createLocalClassInst()
    {
        class LocalClass implements Readable
        {
            public void read()
            {
                System.out.println("Outer inst name : "+myName);
            }
        }
        return new LocalClass();
    }
}
```

```
public static void main(String[] args)
{
    OuterClass out1=new OuterClass("First");
    Readable localInst1=out1.createLocalClassInst();
    localInst1.read();

    OuterClass out2=new OuterClass("Second");
    Readable localInst2=out2.createLocalClassInst();
    localInst2.read();
}
```

이렇게 인터페이스의 구현을 기반으로 로컬 클래스를 정의하면 외부에 정의된 인터페이스의 참조변수를 통해서 인스턴스의 참조가 가능하다!

■ Local 클래스의 지역변수, 매개변수 접근


```
public Readable createLocalClassInst(final int instID)
{
    class LocalClass implements Readable
    {
        public void read()
        {
            System.out.println("Outer inst name : "+myName);
            System.out.println("Local inst ID : "+instID);
        }
    }
    return new LocalClass();
}
```

- ✓ 메소드가 반환하는 순간 매개변수와 지역변수는 소멸된다.
- ✓ 따라서 매개변수와 지역변수의 접근은 논리적으로 맞지 않는다!
- ✓ 단, final로 선언이 변수의 접근은 허용한다.
- ✓ 접근의 허용을 위해서 final 변수를 로컬 클래스의 인스턴스가 접근 가능한 영역에 복사한다.

Anonymous 클래스

클래스의 이름이 정의되어 있지 않다는 사실에서만 Local 클래스와 차이를 보인다!

```
public Readable createLocalClassInst(final int instID)
{
    return new Readable()
    {
        public void read()
        {
            System.out.println("Outer inst name : "+myName);
            System.out.println("Local inst ID : "+instID);
        }
    };
}
```