

## [ 연습문제 - 모범답안 ]

**[7-1]** 셋다카드 20장을 포함하는 셋다카드 한 벌(SutdaDeck클래스)을 정의한 것이다. 셋다카드 20장을 담는 SutdaCard배열을 초기화하시오. 단, 셋다카드는 1부터 10까지의 숫자가 적힌 카드가 한 쌍씩 있고, 숫자가 1, 3, 8인 경우에는 둘 중의 한 장은 광(Kwang)이어야 한다. 즉, SutdaCard의 인스턴스변수 isKwang의 값이 true이어야 한다.

### 【연습문제】/ch7/Exercise7\_1.java

```
class SutdaDeck {
    final int CARD_NUM = 20;
    SutdaCard[] cards = new SutdaCard[CARD_NUM];

    SutdaDeck() {
        for(int i=0;i < cards.length;i++) {
            int num = i%10+1;
            boolean isKwang = (i < 10) && (num==1 || num==3 || num==8);

            cards[i] = new SutdaCard(num,isKwang);
        }
    }

    class SutdaCard {
        int num;
        boolean isKwang;

        SutdaCard() {
            this(1, true);
        }

        SutdaCard(int num, boolean isKwang) {
            this.num = num;
            this.isKwang = isKwang;
        }

        // info() 대신 Object클래스의 toString()을 오버라이딩했다.
        public String toString() {
            return num + ( isKwang ? "K":"" );
        }
    }

    class Exercise7_1 {
        public static void main(String args[]) {
            SutdaDeck deck = new SutdaDeck();

            for(int i=0; i < deck.cards.length;i++)
                System.out.print(deck.cards[i]+",");
        }
    }
}
```

### 【실행결과】

```
1K,2,3K,4,5,6,7,8K,9,10,1,2,3,4,5,6,7,8,9,10,
```

**[해설]** SutdaDeck 클래스에 cards라는 SutdaCard 배열이 정의되어 있다. 이 배열을 생성했다고 해서 SutdaCard 인스턴스가 생성된 것은 아니다. 그저 SutdaCard 인스턴스를 저장하기 위한 공간을 생성한 것일 뿐이다. 객체 배열을 생성할 때, 배열만 생성해 놓고 객체를 생성하지 않는 실수를 하지 않도록 주의하자.

```
SutdaCard[] cards = new SutdaCard[CARD_NUM];
```

생성자를 통해 객체 배열 SutdaCard에 SutdaCard 인스턴스를 생성해서 저장할 차례다. 아래와 같이 반복문을 이용해서 배열의 크기만큼 SutdaCard 인스턴스를 생성하면 되는데, 이때 num의 값과 isKwang의 값을 어떻게 계산해 넣 것인지를 고민해야 한다.

```
SutdaDeck() {
    for(int i=0;i < cards.length;i++) {
        int num = ???;
        boolean isKwang = ???;

        cards[i] = new SutdaCard(num,isKwang);
    }
}
```

아래의 표에서 볼 수 있는 것처럼, i의 값이 0~19까지 변하는 동안 우리가 원하는 num의 값을 얻기 위해서는  $i \% 10 + 1$ 과 같은 계산식을 사용하면 된다.

i	$i \% 10$	$i \% 10 + 1$
0	0	1
1	1	2
2	2	3
3	3	4
4	4	5
5	5	6
6	6	7
7	7	8
8	8	9
9	9	10
10	0	1
11	1	2
12	2	3
13	3	4
14	4	5
15	5	6
16	6	7
17	7	8
18	8	9
19	9	10

그리고 num의 값이 1, 3, 8일 때, 한 쌍의 카드 종에서 하나는 광(kwang)이어야 하므로 아래와 같은 조건식이 필요하다. AND(&&)가 OR(||)보다 우선순위가 높기 때문에 괄호를 꼭 사용해야 한다.

```
boolean isKwang = (i < 10) && (num==1 || num==3 || num==8);
```

만일 i의 값이 2이고 num의 값이 3이라면 위의 조건식은 다음과 같은 계산과정을 거쳐서 isKwang에는 true가 저장된다.

```
boolean isKwang = (2 < 10) && (3==1 || 3==3 || 3==8);
→ boolean isKwang = (true) && (false || true || false);
→ boolean isKwang = (true) && (true || false);
→ boolean isKwang = (true) && (true);
→ boolean isKwang = true;
```

**[7-2]** 문제7-1의 SutdaDeck클래스에 다음에 정의된 새로운 메서드를 추가하고 테스트 하시오.

**[주의]** Math.random()을 사용하는 경우 실행결과와 다를 수 있음.

1. 메서드명 : shuffle

기능 : 배열 cards에 담긴 카드의 위치를 뒤섞는다.(Math.random()사용)  
 반환타입 : 없음  
 매개변수 : 없음

2. 메서드명 : pick

기능 : 배열 cards에서 지정된 위치의 SutdaCard를 반환한다.  
 반환타입 : SutdaCard  
 매개변수 : int index - 위치

3. 메서드명 : pick

기능 : 배열 cards에서 임의의 위치의 SutdaCard를 반환한다.(Math.random()사용)  
 반환타입 : SutdaCard  
 매개변수 : 없음

**[연습문제]/ch7/Exercise7\_2.java**

```
class SutdaDeck {
    final int CARD_NUM = 20;
    SutdaCard[] cards = new SutdaCard[CARD_NUM];

    SutdaDeck() {
        for(int i=0;i < cards.length;i++) {
            int num = i%10+1;
            boolean isKwang = (i < 10) && (num==1 || num==3 || num==8);

            cards[i] = new SutdaCard(num,isKwang);
        }
    }

    void shuffle() {
        for(int i=0; i<cards.length;i++) {
            int j = (int)(Math.random()*cards.length);

            // cards[i] 와 cards[j] 의 값을 서로 바꾼다.
            SutdaCard tmp = cards[i];
            cards[i] = cards[j];
            cards[j] = tmp;
        }
    }

    SutdaCard pick(int index) {
        if(index < 0 || index >= CARD_NUM) // index의 유효성을 검사한다.
            return null;
        return cards[index];
    }
}
```

```

        SutdaCard pick() {
            int index = (int) (Math.random() * cards.length);
            return pick(index); // pick(int index)를 호출한다.
        }
    } // SutdaDeck

    class SutdaCard {
        int num;
        boolean isKwang;

        SutdaCard() {
            this(1, true);
        }

        SutdaCard(int num, boolean isKwang) {
            this.num = num;
            this.isKwang = isKwang;
        }

        public String toString() {
            return num + (isKwang ? "K":"" );
        }
    }

    class Exercise7_2 {
        public static void main(String args[]) {
            SutdaDeck deck = new SutdaDeck();

            System.out.println(deck.pick());
            System.out.println(deck.pick());
            deck.shuffle();

            for(int i=0; i < deck.cards.length;i++)
                System.out.print(deck.cards[i]+",");
            System.out.println();
            System.out.println(deck.pick());
        }
    }
}

```

**【실행결과】**

```

1K
7
2,6,10,1K,7,3,10,5,7,8,5,1,2,9,6,9,4,8K,4,3K,
2

```

**[해설]** shuffle메서드에 대한 것은 이미 문제6-20에서 이미 설명했으므로 설명을 생략하겠다. pick(int index)메서드는 매개변수 index에 대한 유효성검사가 필요하다. 그렇지 않으면 배열의 index범위를 넘어서서 ArrayIndexOutOfBoundsException이 발생할 수 있다. 매개변수가 있는 메서드는 반드시 작업 전에 유효성검사를 해야 한다는 것을 기억하자.

```
SutdaCard pick(int index) {
    if(index < 0 || index >= CARD_NUM) // index의 유효성을 검사한다.
        return null;
    return cards[index];
}

SutdaCard pick() {
    int index = (int)(Math.random()*cards.length);
    return pick(index); // pick(int index)를 호출한다.
}
```

pick()메서드의 경우, cards배열에 있는 임의의 카드를 꺼내야하므로 Math.random()을 이용해서 유효한 index범위 내의 한 값을 얻어서 다시 pick(int index)메서드를 호출한다. 다소 비효율적이지만 코드의 중복을 제거하고 재사용성을 높이기 위해 이처럼 하는 것이다. 그러나 너무 객체지향적인 측면에 얹매여서 코드를 짤 필요는 없다고 생각한다. 상황에 맞는 적절한 코드를 작성하면 그것으로 좋지 않을까.

```
SutdaCard pick() {
    int index = (int)(Math.random()*cards.length);
    return cards[index];
}
```

**[7-3]** 오버라이딩의 정의와 필요성에 대해서 설명하시오.

**[정답]** 오버라이딩(overriding)이란, ‘조상 클래스로부터 상속받은 메서드를 자손 클래스에 맞게 재정의 하는 것’을 말한다.

조상 클래스로부터 상속받은 메서드를 자손 클래스에서 그대로 사용할 수 없는 경우가 많기 때문에 오버라이딩이 필요하다.

[7-4] 다음 중 오버라이딩의 조건으로 옳지 않은 것은? (모두 고르시오)

- a. 조상의 메서드와 이름이 같아야 한다.
- b. 매개변수의 수와 타입이 모두 같아야 한다.
- c. 접근 제어자는 조상의 메서드보다 좁은 범위로만 변경할 수 있다.
- d. 조상의 메서드보다 더 많은 수의 예외를 선언할 수 있다.

[정답] c, d

[해설]

자손 클래스에서 오버라이딩하는 메서드는 조상 클래스의 메서드와

- 이름이 같아야 한다.
- 매개변수가 같아야 한다.
- 리턴타입이 같아야 한다.

[참고] JDK1.5부터 '공변 반환타입(covariant return type)'이 추가되어, 반환타입을 자손 클래스의 타입으로 변경하는 것은 가능하도록 조건이 완화되었다. p.457

조상 클래스의 메서드를 자손 클래스에서 오버라이딩할 때

1. 접근 제어자를 조상 클래스의 메서드보다 좁은 범위로 변경할 수 없다.
2. 예외는 조상 클래스의 메서드보다 많이 선언할 수 없다.
3. 인스턴스메서드를 static메서드로 또는 그 반대로 변경할 수 없다.

**[7-5]** 다음의 코드는 컴파일하면 에러가 발생한다. 그 이유를 설명하고 에러를 수정하기 위해서는 코드를 어떻게 바꾸어야 하는가?

**【연습문제】/ch7/Exercise7\_5.java**

```
class Product
{
    int price;           // 제품의 가격
    int bonusPoint;     // 제품구매 시 제공하는 보너스점수

    Product() {}

    Product(int price) {
        this.price = price;
        bonusPoint =(int)(price/10.0);
    }
}

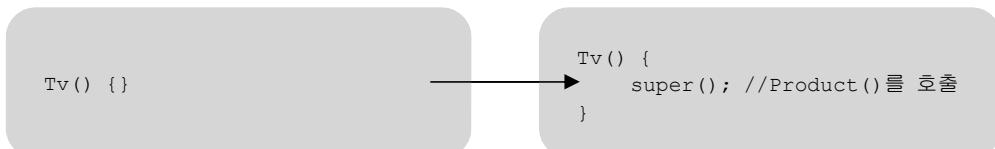
class Tv extends Product {
    Tv() {}

    public String toString() {
        return "Tv";
    }
}

class Exercise7_5 {
    public static void main(String[] args) {
        Tv t = new Tv();
    }
}
```

**[정답]** Product클래스에 기본 생성자 Product()가 없기 때문에 에러가 발생한다. Product 클래스에 기본 생성자 Product() {}를 추가해 줘야한다.

**[해설]** Tv클래스의 인스턴스를 생성할 때, 생성자 Tv()가 호출되고 Tv()는 조상 생성자 super()를 호출한다. 실제 코드에서는 super()를 호출하는 곳이 없지만 컴파일러가 자동적으로 추가해 준다. 그래서 컴파일을 하고 나면 아래의 오른쪽 코드와 같이 변경된다.



추가된 super()는 조상클래스인 Product의 기본 생성자 Product()를 호출하는 것인데, Product클래스에는 기본 생성자 Product()가 정의되어 있지 않다. 정의되어 있지 않은 생성자를 호출하니까 에러가 발생하는 것이다. Product클래스에는 이미 Product(int price)라는 생성자가 정의되어 있기 때문에 컴파일러가 자동적으로 추가해 주지도 않으므로 직접 Product클래스에 Product(){}를 넣어주면 문제가 해결된다.

**[7-6]** 자손 클래스의 생성자에서 조상 클래스의 생성자를 호출해야하는 이유는 무엇인가?

**[정답]** 조상에 정의된 인스턴스 변수들이 초기화되도록 하기 위해서.

**[해설]** 자손클래스의 인스턴스를 생성하면 조상으로부터 상속받은 인스턴스변수들도 생성되는데, 이 상속받은 인스턴스변수들 역시 적절히 초기되어야 한다. 상속받은 조상의 인스턴스변수들을 자손의 생성자에서 직접 초기화하기보다는 조상의 생성자를 호출함으로써 초기화되도록 하는 것이 바람직하다.

각 클래스의 생성자는 해당 클래스에 선언된 인스턴스변수의 초기화만을 담당하고, 조상 클래스로부터 상속받은 인스턴스변수의 초기화는 조상클래스의 생성자가 처리하도록 해야 하는 것이다.

**[7-7]** 다음 코드의 실행했을 때 호출되는 생성자의 순서와 실행결과를 적으시오.

**[연습문제]/ch7/Exercise7\_7.java**

```

class Parent {
    int x=100;

    Parent() {
        this(200); // Parent(int x)를 호출
    }

    Parent(int x) {
        this.x = x;
    }

    int getX() {
        return x;
    }
}

class Child extends Parent {
    int x = 3000;

    Child() {
        this(1000); // Child(int x)를 호출
    }

    Child(int x) {
        this.x = x;
    }
}

class Exercise7_7 {
    public static void main(String[] args) {
        Child c = new Child();

        System.out.println("x="+c.getX());
    }
}

```

**[정답]** Child() → Child(int x) → Parent() → Parent(int x) → Object()의 순서로 호출된다.

**[실행결과]**

x=200

**[해설]** 컴파일러는 생성자의 첫 줄에 다른 생성자를 호출하지 않으면 조상의 기본 생성자를 호출하는 코드 'super();'를 넣는다. 그래서 왼쪽의 코드는 컴파일 후 오른쪽과 같은 코드로 바뀐다. Child클래스의 조상은 Parent이므로 super()는 Parent()를 의미한다.

```

Child(int x) {
    this.x = x;
} → Child(int x) {
    super(); // Parent()를 호출
    this.x = x;
}

```

마찬가지로 Parent(int x) 역시 컴파일러가 Parent의 조상인 Object클래스의 기본 생성자를 호출하는 코드 'super();'를 넣는다.

```
Parent(int x) {  
    this.x = x;  
}
```

```
Parent(int x) {  
    super(); // Object()를 호출  
    this.x = x;  
}
```

Child() → Child(int x) → Parent() → Parent(int x) → Object()의 순서로 호출되니까, Child클래스의 인스턴스변수 x는 1000이 되고, Parent클래스의 인스턴스 변수 x는 200이 된다. getX()는 조상인 Parent클래스에 정의된 것이라서, getX()에서 x는 Parent클래스의 인스턴스변수 x를 의미한다. 그래서 x=200이 출력된다.

[7-8] 다음 중 접근제어자를 접근범위가 넓은 것에서 좁은 것의 순으로 바르게 나열한 것은?

- a. **public-protected-(default)-private**
- b. **public-(default)-protected-private**
- c. **(default)-public-protected-private**
- d. **private-protected-(default)-public**

[정답] a

[해설]

접근 제어자가 사용될 수 있는 곳 - 클래스, 멤버변수, 메서드, 생성자

- |                  |   |
|------------------|---|
| <b>private</b>   | - 같은 클래스 내에서만 접근이 가능하다.                     |
| <b>default</b>   | - 같은 패키지 내에서만 접근이 가능하다.                     |
| <b>protected</b> | - 같은 패키지 내에서, 그리고 다른 패키지의 자손클래스에서 접근이 가능하다. |
| <b>public</b>    | - 접근 제한이 전혀 없다.                             |

접근 범위가 넓은 쪽에서 좁은 쪽의 순으로 왼쪽부터 나열하면 다음과 같다.

**public > protected > default > private**

제어자	같은 클래스	같은 패키지	자손클래스	전체
<b>public</b>				
<b>protected</b>				
<b>default</b>				
<b>private</b>				

**[7-9]** 다음 중 제어자 final을 붙일 수 있는 대상과 붙였을 때 그 의미를 적은 것이다.  
옳지 않은 것은? (모두 고르시오)

- 지역변수 - 값을 변경할 수 없다.
- 클래스 - 상속을 통해 클래스에 새로운 멤버를 추가할 수 없다.
- 메서드 - 오버로딩을 할 수 없다. ← 오버라이딩(overriding)을 할 수 없다.
- 멤버변수 - 값을 변경할 수 없다.

**[정답]** c

**[해설]** 제어자 final은 '마지막의' 또는 '변경될 수 없는'의 의미를 가지고 있으며 거의 모든 대상에 사용될 수 있다.

제어자	대상	의 미
final	클래스	변경될 수 없는 클래스, 확장될 수 없는 클래스가 된다. 그래서 final로 지정된 클래스는 다른 클래스의 조상이 될 수 없다.
	메서드	변경될 수 없는 메서드, final로 지정된 메서드는 오버라이딩을 통해 재정의 될 수 없다.
	멤버변수 지역변수	변수 앞에 final이 붙으면, 값을 변경할 수 없는 상수가 된다.

**[7-10]** MyTv2클래스의 멤버변수 isPowerOn, channel, volume을 클래스 외부에서 접근할 수 있도록 제어자를 붙이고 대신 이 멤버변수들의 값을 어디서나 읽고 변경할 수 있도록 getter와 setter메서드를 추가하라.

**[연습문제]/ch7/Exercise7\_10.java**

```
class MyTv2 {
    private boolean isPowerOn;
    private int     channel;
    private int     volume;

    final int MAX_VOLUME  = 100;
    final int MIN_VOLUME  = 0;
    final int MAX_CHANNEL = 100;
    final int MIN_CHANNEL = 1;

    public void setVolume(int volume) {
        if(volume > MAX_VOLUME || volume < MIN_VOLUME)
            return;

        this.volume = volume;
    }

    public int getVolume() {
        return volume;
    }

    public void setChannel(int channel) {
        if(channel > MAX_CHANNEL || channel < MIN_CHANNEL)
            return;

        this.channel = channel;
    }

    public int getChannel() {
        return channel;
    }
}

class Exercise7_10 {
    public static void main(String args[]) {
        MyTv2 t = new MyTv2();

        t.setChannel(10);
        System.out.println("CH:"+t.getChannel());
        t.setVolume(20);
        System.out.println("VOL:"+t.getVolume());
    }
}
```

**[실행결과]**

```
CH:10
VOL:20
```

**[해설]** 별로 어렵지 않은 문제라 별도의 설명이 필요없을 것이다. 다만 매개변수가 있는 메서드는 반드시 작업 전에 넘겨받은 값의 유효성검사를 해야 한다는 것을 잊지 말자.

**[7-11]** 문제7-10에서 작성한 MyTv2클래스에 이전 채널(previous channel)로 이동하는 기능의 메서드를 추가해서 실행결과와 같은 결과를 얻도록 하시오.

**[Hint]** 이전 채널의 값을 저장할 멤버변수를 정의하라.

메서드명 : gotoPrevChannel

기 능 : 현재 채널을 이전 채널로 변경한다.

반환타입 : 없음

매개변수 : 없음

**[연습문제]/ch7/Exercise7\_11.java**

```
class MyTv2 {
    private boolean isPowerOn;
    private int     channel;
    private int     volume;
    private int     prevChannel; // 이전 채널(previous channel)

    final int MAX_VOLUME = 100;
    final int MIN_VOLUME = 0;
    final int MAX_CHANNEL = 100;
    final int MIN_CHANNEL = 1;

    public void setVolume(int volume) {
        if(volume > MAX_VOLUME || volume < MIN_VOLUME)
            return;

        this.volume = volume;
    }

    public int getVolume() {
        return volume;
    }

    public void setChannel(int channel) {
        if(channel > MAX_CHANNEL || channel < MIN_CHANNEL)
            return;

        prevChannel = this.channel; // 현재 채널을 이전 채널에 저장한다.
        this.channel = channel;
    }

    public int getChannel() {
        return channel;
    }

    public void gotoPrevChannel() {
        setChannel(prevChannel); // 현재 채널을 이전 채널로 변경한다.
    }
}

class Exercise7_11 {
    public static void main(String args[]) {
        MyTv2 t = new MyTv2();
```

```

        t.setChannel(10);
        System.out.println("CH:"+t.getChannel());
        t.setChannel(20);
        System.out.println("CH:"+t.getChannel());
        t.gotoPrevChannel();
        System.out.println("CH:"+t.getChannel());
        t.gotoPrevChannel();
        System.out.println("CH:"+t.getChannel());
    }
}

```

**【실행결과】**

```

CH:10
CH:20
CH:10
CH:20

```

**[해설]** 먼저 이전 채널을 저장할 변수(prevChannel)를 하나 추가해야 한다. 그리고 채널이 바뀔 때마다 이 변수에 바뀌기 전의 채널을 저장해야 한다. 문제7-10의 코드에 아래의 붉은 색 코드를 추가했다.

```

public void setChannel(int channel){
    if(channel > MAX_CHANNEL || channel < MIN_CHANNEL)
        return;

    prevChannel = this.channel; // 현재 채널을 이전 채널에 저장한다.
    this.channel = channel;
}

```

이제 gotoPrevChannel()에서는 setChannel()을 호출해주기만 하면 된다.

```

public void gotoPrevChannel() {
    setChannel(prevChannel); // 현재 채널을 이전 채널로 변경한다.
}

```

[7-12] 다음 중 접근 제어자에 대한 설명으로 옳지 않은 것은? (모두 고르시오)

- a. `public`은 접근제한이 전혀 없는 접근 제어자이다.
- b. `(default)`가 붙으면, 같은 패키지 내에서만 접근이 가능하다.
- c. 지역변수에도 접근 제어자를 사용할 수 있다.
- d. `protected`가 붙으면, 같은 패키지 내에서도 접근이 가능하다.
- e. `protected`가 붙으면, 다른 패키지의 자손 클래스에서 접근이 가능하다.

[정답] c

[해설]

접근 제어자가 사용될 수 있는 곳 – 클래스, 멤버변수, 메서드, 생성자

**private** – 같은 클래스 내에서만 접근이 가능하다.  
**default** – 같은 패키지 내에서만 접근이 가능하다.  
**protected** – 같은 패키지 내에서, 그리고 다른 패키지의 자손클래스에서 접근이 가능하다.  
**public** – 접근 제한이 전혀 없다.

제어자	같은 클래스	같은 패키지	자손클래스	전체
<b>public</b>				
<b>protected</b>				
<b>default</b>				
<b>private</b>				

**[7-13]** Math클래스의 생성자는 접근 제어자가 `private`이다. 그 이유는 무엇인가?

**[정답]** Math클래스의 모든 메서드가 `static`메서드이고 인스턴스변수가 존재하지 않기 때문에 객체를 생성할 필요가 없기 때문

**[해설]** Math클래스는 몇 개의 상수와 `static`메서드만으로 구성되어 있기 때문에 인스턴스를 생성할 필요가 없다. 그래서 외부로부터의 불필요한 접근을 막기 위해 다음과 같이 생성자의 접근 제어자를 `private`으로 지정하였다.

```
public final class Math {  
    private Math() {}  
    //...  
}
```

**[7-14]** 문제7-1에 나오는 셋다카드의 숫자와 종류(isKwang)는 사실 한번 값이 지정되면 변경되어서는 안 되는 값이다. 카드의 숫자가 한번 잘못 바뀌면 똑같은 카드가 두 장이 될 수 도 있기 때문이다. 이러한 문제점이 발생하지 않도록 아래의 SutdaCard를 수정하시오.

**[연습문제]/ch7/Exercise7\_14.java**

```
class SutdaCard {
    final int NUM;
    final boolean IS_KWANG;

    SutdaCard() {
        this(1, true);
    }

    SutdaCard(int num, boolean isKwang) {
        this.NUM = num;
        this.IS_KWANG = isKwang;
    }

    public String toString() {
        return NUM + ( IS_KWANG ? "K":"");
    }
}

class Exercise7_14 {
    public static void main(String args[]) {
        SutdaCard card = new SutdaCard(1, true);
    }
}
```

**[해설]** 원래 변수 앞에 `final`을 붙일 때는 선언과 초기화를 동시에 해야 한다.

```
final int MAX_VOLUME = 100;
```

그러나 인스턴스변수의 경우, 선언시에 초기화 하지 않고 생성자에서 초기화할 수 있다. 생성할 때 지정된 값이 변하지 않도록 할 수 있는 것이다. 상수이므로 한번 초기화한 이후로는 값을 바꿀 수 없다.

```
final int NUM;
final boolean IS_KWANG;

SutdaCard(int num, boolean isKwang) {
    this.NUM = num;           // 생성자에서 단 한 번의 초기화만 가능
    this.IS_KWANG = isKwang; // 생성자에서 단 한 번의 초기화만 가능
}
```

카드게임에서 카드의 숫자와 무늬가 게임도중에 변경되는 것이 가능하다면, 실수로 같은 카드가 두 장이 되는 일이 일어날 수 있기 때문에 이를 방지하기 위해서 숫자와 무늬는 한번 지정되면 변경할 수 없도록 하는 것이 바람직하다.

[7-15] 클래스가 다음과 같이 정의되어 있을 때, 형변환을 올바르게 하지 않은 것은?  
(모두 고르시오.)

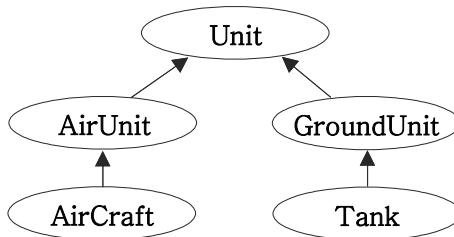
```
class Unit {}
class AirUnit extends Unit {}
class GroundUnit extends Unit {}
class Tank extends GroundUnit {}
class AirCraft extends AirUnit {}

Unit u = new GroundUnit();
Tank t = new Tank();
AirCraft ac = new AirCraft();
```

- a. u = (Unit)ac;
- b. u = ac;
- c. GroundUnit gu = (GroundUnit)u;
- d. AirUnit au = ac;
- e. t = (Tank)u; ← 조상타입의 인스턴스를 자손타입으로 형변환 할 수 없다.
- f. GroundUnit gu = t;

[정답] e

[해설] 클래스간의 상속관계를 그림으로 그려보면 쉽게 알 수 있다.



Unit클래스는 나머지 네 개 클래스의 조상이므로 형변환이 가능하며, 심지어는 생략할 수도 있다.

```
AirCraft ac = new AirCraft();
u = (Unit)ac; // u는 AirCraft의 조상인 Unit타입이므로 형변환이 가능하다.
u = ac;       // 업캐스팅(자손→조상)이므로 형변환을 생략할 수 있다.
```

조상타입의 참조변수로 자손타입의 인스턴스를 참조하는 것이 가능하기 때문에 아래의 코드는 모두 가능하다.

```
Unit u = new GroundUnit();
GroundUnit gu = (GroundUnit)u; // u가 참조하는 객체가 GroundUnit이므로 OK
GroundUnit gu = (GroundUnit)new GroundUnit(); // 위의 두 줄을 한 줄로 합침

AirCraft ac = new AirCraft();
AirUnit au = ac; // AirCraft가 AirUnit의 자손이므로 가능. 형변환 생략됨
AirUnit au = new AirCraft(); // 위의 두 줄을 한 줄로 합치면 이렇게 쓸 수 있음

Tank t = new Tank();
GroundUnit gu = t; // 조상타입의 참조변수로 자손타입의 인스턴스를 참조. OK
GroundUnit gu = new Tank(); // 위의 두 줄을 한 줄로 합치면 이렇게 쓸 수 있음
```

그러나 조상인스턴스를 자손타입으로 형변환하는 것은 허용하지 않는다. 참조변수 u는 실제로 GroundUnit인스턴스를 참조하고 있다. (Tank)u는 GroundUnit인스턴스를 자손타입인 Tank로 형변환하는 것인데, 자손타입으로 형변환은 허용되지 않으므로 실행시 에러가 발생한다.

**[참고]** 컴파일 시에는 타입만을 체크하기 때문에 에러가 발생하지 않을 수도 있지만, 실행시에 에러가 발생한다.

```
Unit u = new GroundUnit();
Tank t = new Tank();

t = (Tank)u; // 조상인스턴스 (GroundUnit)를 자손 (Tank) 으로 형변환할 수 없다.
Tank t = (Tank)new GroundUnit; // 허용되지 않음
```

[7-16] 다음 중 연산결과가 true가 아닌 것은? (모두 고르시오)

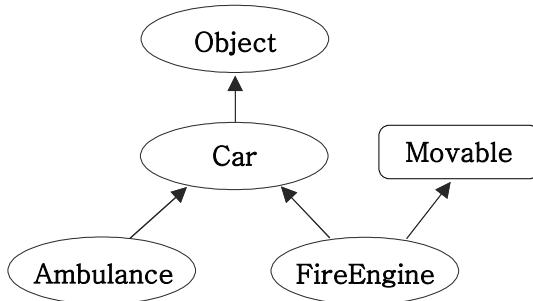
```
class Car {}
class FireEngine extends Car implements Movable {}
class Ambulance extends Car {}

FireEngine fe = new FireEngine();
```

- a. fe instanceof FireEngine
- b. fe instanceof Movable
- c. fe instanceof Object
- d. fe instanceof Car
- e. fe instanceof Ambulance

[정답] e

[해설] instanceof연산자는 실제 인스턴스의 모든 조상이나 구현한 인터페이스에 대해 true를 반환한다. 그래서, 아래 그림에서 알 수 있듯이 FireEngine인스턴스는 Object, Car, Movable, FireEngine타입에 대해 instanceof연산을 하면 결과로 true를 얻는다. 어떤 타입에 대해 instanceof연산결과가 true라는 것은 그 타입으로 형변환이 가능하다는 것을 뜻한다. 참조변수의 형변환을 하기 전에 앞서 instanceof연산자로 형변환이 가능한지 미리 확인해 보는 것이 좋다.



**[7-17]** 아래 세 개의 클래스로부터 공통부분을 뽑아서 Unit이라는 클래스를 만들고, 이 클래스를 상속받도록 코드를 변경하시오.

```

class Marine {    // 보병
    int x, y;      // 현재 위치
    void move(int x, int y) { /* 지정된 위치로 이동 */ }
    void stop()        { /* 현재 위치에 정지 */ }
    void stimPack()     { /* 스팀팩을 사용한다. */ }
}

class Tank {      // 탱크
    int x, y;      // 현재 위치
    void move(int x, int y) { /* 지정된 위치로 이동 */ }
    void stop()        { /* 현재 위치에 정지 */ }
    void changeMode()   { /* 공격모드를 변환한다. */ }
}

class Dropship { // 수송선
    int x, y;      // 현재 위치
    void move(int x, int y) { /* 지정된 위치로 이동 */ }
    void stop()        { /* 현재 위치에 정지 */ }
    void load()         { /* 선택된 대상을 태운다. */ }
    void unload()       { /* 선택된 대상을 내린다. */ }
}

```

**[정답]** 각 클래스의 공통부분을 뽑아서 Unit클래스를 생성하면 된다. 클래스마다 이동하는 방법이 다르므로 move메서드는 추상메서드로 정의하였다. 책에도 같은 내용이 있기 때문에 자세한 설명은 생략하겠다.

```

abstract class Unit {
    int x, y;
    abstract void move(int x, int y); // 추상클래스
    void stop() { /* 현재 위치에 정지 */ }
}

class Marine extends Unit { // 보병
    void move(int x, int y) { /* 지정된 위치로 이동 */ }
    void stimPack()     { /* 스팀팩을 사용한다. */ }
}

class Tank extends Unit { // 탱크
    void move(int x, int y) { /* 지정된 위치로 이동 */ }
    void changeMode()   { /* 공격모드를 변환한다. */ }
}

class Dropship extends Unit { // 수송선
    void move(int x, int y) { /* 지정된 위치로 이동 */ }
    void load()         { /* 선택된 대상을 태운다. */ }
    void unload()       { /* 선택된 대상을 내린다. */ }
}

```

**[7-18]** 다음과 같은 실행결과를 얻도록 코드를 완성하시오.

[Hint] instanceof연산자를 사용해서 형변환한다.

메서드명 : action

기 능 : 주어진 객체의 메서드를 호출한다.

DanceRobot인 경우, dance()를 호출하고,

SingRobot인 경우, sing()을 호출하고,

DrawRobot인 경우, draw()를 호출한다.

반환타입 : 없음

매개변수 : Robot r – Robot인스턴스 또는 Robot의 자손 인스턴스

**【연습문제】/ch7/Exercise7\_18.java**

```
class Exercise7_18 {
    public static void action(Robot r) {
        if(r instanceof DanceRobot) {
            DanceRobot dr = (DanceRobot)r;
            dr.dance();
        } else if(r instanceof SingRobot) {
            SingRobot sr = (SingRobot)r;
            sr.sing();
        } else if(r instanceof DrawRobot) {
            DrawRobot dr = (DrawRobot)r;
            dr.draw();
        }
    }

    public static void main(String[] args) {
        Robot[] arr = { new DanceRobot(), new SingRobot(), new DrawRobot() };

        for(int i=0; i< arr.length;i++)
            action(arr[i]);
    } // main
}

class Robot {}

class DanceRobot extends Robot {
    void dance() {
        System.out.println("춤을 춥니다.");
    }
}

class SingRobot extends Robot {
    void sing() {
        System.out.println("노래를 합니다.");
    }
}

class DrawRobot extends Robot {
    void draw() {
        System.out.println("그림을 그립니다.");
    }
}
```

**[실행결과]**

춤을 춥니다.  
노래를 합니다.  
그림을 그립니다.

**[해설]** action메서드의 매개변수가 Robot타입이므로 Robot클래스의 자손클래스인 DanceRobot, SingRobot, DrawRobot의 인스턴스는 모두 매개변수로 가능하다.

```
Robot[] arr = { new DanceRobot(), new SingRobot(), new DrawRobot() };

for(int i=0; i< arr.length;i++)
    action(arr[i]);
```

action메서드 내에서는 실제로 받아온 인스턴스가 어떤 것인지 알 수 없다. 단지 Robot클래스 또는 그 자손클래스의 인스턴스일 것이라는 것만 알 수 있다. 그래서 instanceof연산자를 이용해야만 실제 인스턴스의 타입을 확인할 수 있다.

```
public static void action(Robot r) {
    if(r instanceof DanceRobot) {
        DanceRobot dr = (DanceRobot)r;
        dr.dance();
    } else if(r instanceof SingRobot) {
        SingRobot sr = (SingRobot)r;
        sr.sing();
    } else if(r instanceof DrawRobot) {
        DrawRobot dr = (DrawRobot)r;
        dr.draw();
    }
}
```

**[7-19]** 다음은 물건을 구입하는 사람을 정의한 Buyer 클래스이다. 이 클래스는 멤버변수로 돈(money)과 장바구니(cart)를 가지고 있다. 제품을 구입하는 기능의 buy메서드와 장바구니에 구입한 물건을 추가하는 add메서드, 구입한 물건의 목록과 사용금액, 그리고 남은 금액을 출력하는 summary메서드를 완성하시오.

1. 메서드명 : buy

기능 : 지정된 물건을 구입한다. 가진 돈(money)에서 물건의 가격을 빼고, 장바구니(cart)에 담는다.

만일 가진 돈이 물건의 가격보다 적다면 바로 종료한다.

반환타입 : 없음

매개변수 : Product p - 구입할 물건

2. 메서드명 : add

기능 : 지정된 물건을 장바구니에 담는다.

만일 장바구니에 담을 공간이 없으면, 장바구니의 크기를 2배로 늘린 다음에 담는다.

반환타입 : 없음

매개변수 : Product p - 구입할 물건

3. 메서드명 : summary

기능 : 구입한 물건의 목록과 사용금액, 남은 금액을 출력한다.

반환타입 : 없음

매개변수 : 없음

**[연습문제]/ch7/Exercise7\_19.java**

```
class Exercise7_19 {
    public static void main(String args[]) {
        Buyer b = new Buyer();
        b.buy(new Tv());
        b.buy(new Computer());
        b.buy(new Tv());
        b.buy(new Audio());
        b.buy(new Computer());
        b.buy(new Computer());
        b.buy(new Computer());

        b.summary();
    }
}

class Buyer {
    int money = 1000;
    Product[] cart = new Product[3]; // 구입한 제품을 저장하기 위한 배열
    int i = 0; // Product배열 cart에 사용될 index

    void buy(Product p) {
        // 1.1 가진 돈과 물건의 가격을 비교해서 가진 돈이 적으면 메서드를 종료한다.
        if(money < p.price) {
            System.out.println("잔액이 부족하여 "+ p +"을/를 살수 없습니다.");
            return;
        }
    }
}
```

```

//      1.2 가진 돈이 충분하면, 제품의 가격을 가진 돈에서 빼고
//      money -= p.price;
//      1.3 장바구니에 구입한 물건을 담는다. (add메서드 호출)
//      add(p);
}

void add(Product p) {
//      1.1 i의 값이 장바구니의 크기보다 같거나 크면
//          if(i >= cart.length) {
//              1.1.1 기존의 장바구니보다 2배 큰 새로운 배열을 생성한다.
//                  Product[] tmp = new Product[cart.length*2];
//              1.1.2 기존의 장바구니의 내용을 새로운 배열에 복사한다.
//                  System.arraycopy(cart, 0, tmp, 0, cart.length);
//              1.1.3 새로운 장바구니와 기존의 장바구니를 바꾼다.
//                  cart = tmp;
//          }
//      1.2 물건을 장바구니 (cart)에 저장한다. 그리고 i의 값을 1 증가시킨다.
//          cart[i++]=p;
} // add(Product p)

void summary() {
    String itemList = "";
    int sum = 0;

    for(int i=0; i < cart.length;i++) {
        if(cart[i]==null)
            break;
//        1.1 장바구니에 담긴 물건들의 목록을 만들어 출력한다.
//        itemList += cart[i] + ",";
//        1.2 장바구니에 담긴 물건들의 가격을 모두 더해서 출력한다.
//        sum += cart[i].price;
    }

//    1.3 물건을 사고 남은 금액 (money)를 출력한다.
    System.out.println("구입한 물건:"+itemList);
    System.out.println("사용한 금액:"+sum);
    System.out.println("남은 금액:"+money);
} // summary()
}

class Product {
    int price;           // 제품의 가격

    Product(int price) {
        this.price = price;
    }
}

class Tv extends Product {
    Tv() { super(100); }

    public String toString() { return "Tv"; }
}

class Computer extends Product {
    Computer() { super(200); }
}

```

```
    public String toString() { return "Computer"; }  
}  
  
class Audio extends Product {  
    Audio() { super(50); }  
  
    public String toString() { return "Audio"; }  
}
```

**【실행결과】**

잔액이 부족하여 Computer을/를 살수 없습니다.  
구입한 물건:Tv, Computer, Tv, Audio, Computer, Computer,  
사용한 금액:850  
남은 금액:150

**[해설]** 자신이 스스로 로직을 작성할 수 있으면 가장 좋겠지만, 적어도 주어진 로직대로 코드를 구현할 수 있는 능력은 갖추어야 한다. 그런 능력을 향상시키기 위한 문제이다. 이 문제가 쉽게 느껴지는 사람은 로직(주석)을 안보고 코드를 다시 작성해보기 바란다. 책에 있는 내용을 복습하는 문제이기 때문에 자세한 설명은 생략하겠다. 책을 참고하길 바란다.

**[7-20]** 다음의 코드를 실행한 결과를 적으시오.

```
[연습문제]/ch7/Exercise7_20.java
class Exercise7_20 {
    public static void main(String[] args) {
        Parent p = new Child();
        Child c = new Child();

        System.out.println("p.x = " + p.x);
        p.method();

        System.out.println("c.x = " + c.x);
        c.method();
    }
}

class Parent {
    int x = 100;

    void method() {
        System.out.println("Parent Method");
    }
}

class Child extends Parent {
    int x = 200;

    void method() {
        System.out.println("Child Method");
    }
}
```

### [정답]

#### **[실행결과]**

```
p.x = 100
Child Method
c.x = 200
Child Method
```

**[해설]** 조상 클래스에 선언된 멤버변수와 같은 이름의 인스턴스변수를 자손 클래스에 중복으로 정의했을 때, 조상타입의 참조변수로 자손 인스턴스를 참조하는 경우와 자손타입의 참조변수로 자손 인스턴스를 참조하는 경우는 서로 다른 결과를 얻는다.

메서드의 경우 조상 클래스의 메서드를 자손의 클래스에서 오버라이딩한 경우에도 참조변수의 타입에 관계없이 항상 실제 인스턴스의 메서드(오버라이딩된 메서드)가 호출되지만, 멤버변수의 경우 참조변수의 타입에 따라 달라진다.

타입은 다르지만, 참조변수 p, c 모두 Child인스턴스를 참조하고 있다.

```
Parent p = new Child();
Child c = new Child();
```

그리고, Parent클래스와 Child클래스는 서로 같은 멤버들을 정의하고 있다.

```
class Parent {  
    int x = 100;  
    ...  
}  
  
class Child extends Parent {  
    int x = 200;  
    ...  
}
```

이 때 조상타입의 참조변수 p로 Child인스턴스의 멤버들을 사용하는 것과 자손타입의 참조변수 c로 Child인스턴스의 멤버들을 사용하는 것의 차이를 알 수 있다.

메서드인 method()의 경우 참조변수의 타입에 관계없이 항상 실제 인스턴스의 타입인 Child클래스에 정의된 메서드가 호출되지만, 인스턴스변수인 x는 참조변수의 타입에 따라서 달라진다.

**[7-21]** 다음과 같이 attack메서드가 정의되어 있을 때, 이 메서드의 매개변수로 가능한 것 두 가지를 적으시오.

```
interface Movable {  
    void move(int x, int y);  
}  
  
void attack(Movable f) {  
    /* 내용 생략 */  
}
```

**[정답]** null, Movable인터페이스를 구현한 클래스 또는 그 자손의 인스턴스

**[해설]** 매개변수의 다형성을 잘 이해하고 있는지를 확인하는 문제이다. 매개변수의 타입이 인터페이스라는 것은 어떤 의미일지 이해하지 못하는 경우가 많은데, 이것을 이해하는 것은 매우 중요하다.

언제라도 누가 ‘Movable인터페이스타입의 매개변수로 가능한 것이 무엇이냐?’고 물었을 때, 주저 없이 얘기할 수 있도록 완전히 외우고 있어야 한다.

**[7-22]** 아래는 도형을 정의한 Shape클래스이다. 이 클래스를 조상으로 하는 Circle클래스와 Rectangle클래스를 작성하시오. 이 때, 생성자도 각 클래스에 맞게 적절히 추가해야 한다.

- (1) 클래스명 : Circle  
조상클래스 : Shape  
멤버변수 : double r - 반지름
- (2) 클래스명 : Rectangle  
조상클래스 : Shape  
멤버변수 : double width - 폭  
double height - 높이  
메서드 :  
1. 메서드명 : isSquare  
기능 : 정사각형인지 아닌지를 알려준다.  
반환타입 : boolean  
매개변수 : 없음

#### 【연습문제】/ch7/Exercise7\_22.java

```
abstract class Shape {
    Point p;

    Shape() {
        this(new Point(0,0));
    }

    Shape(Point p) {
        this.p = p;
    }

    abstract double calcArea(); // 도형의 면적을 계산해서 반환하는 메서드

    Point getPosition() {
        return p;
    }

    void setPosition(Point p) {
        this.p = p;
    }
}

class Rect extends Shape {
    double width;
    double height;

    Rect(double width, double height) {
        this(new Point(0,0), width, height);
    }

    Rect(Point p, double width, double height) {
        super(p); // 조상의 멤버는 조상의 생성자가 초기화하도록 한다.
        this.width = width;
    }
}
```

```
        this.height = height;
    }

    boolean isSquare() {
        // width& height가 0이 아님과 width와 height가 같으면 true를 반환한다.
        return width*height!=0 && width==height;
    }

    double calcArea() {
        return width * height;
    }
}

class Circle extends Shape {
    double r;      // 반지름

    Circle(double r) {
        this(new Point(0,0),r); // Circle(Point p, double r)를 호출
    }

    Circle(Point p, double r) {
        super(p);      // 조상의 멤버는 조상의 생성자가 초기화하도록 한다.
        this.r = r;
    }

    double calcArea() {
        return Math.PI * r * r;
    }
}

class Point {
    int x;
    int y;

    Point() {
        this(0,0);
    }

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

    public String toString() {
        return "["+x+","+y+"]";
    }
}
```

**[7-23]** 문제7-22에서 정의한 클래스들의 면적을 구하는 메서드를 작성하고 테스트 하시오.

1. 메서드명 : sumArea  
기능 : 주어진 배열에 담긴 도형들의 넓이를 모두 더해서 반환한다.  
반환타입 : double  
매개변수 : Shape[] arr

**[연습문제]/ch7/Exercise7\_23.java**

```
class Exercise7_23
{
    static double sumArea(Shape[] arr) {
        double sum = 0;

        for(int i=0; i < arr.length;i++)
            sum+= arr[i].calcArea();

        return sum;
    }

    public static void main(String[] args)
    {
        Shape[] arr = {new Circle(5.0), new Rectangle(3,4), new Circle(1)};
        System.out.println("면적의 합:"+sumArea(arr));
    }
}
```

**[실행결과]**

면적의 합:93.68140899333463

**[해설]** 반복문으로 넘겨받은 객체배열(arr)의 객체들에 대해 calcArea()를 호출하여 면적을 구하고 누적해서 반환하도록 작성하면 된다.

Shape타입의 배열에는 Shape의 자손 인스턴스가 들어있기 때문에, Shape클래스의 추상메서드 calcArea()를 호출해도 실제로는 각 인스턴스에 완전히 구현된 calcArea()가 호출된다.

**[7-24]** 다음 중 인터페이스의 장점이 아닌 것은?

- a. 표준화를 가능하게 해준다.
- b. 서로 관계없는 클래스들에게 관계를 맺어 줄 수 있다.
- c. 독립적인 프로그래밍이 가능하다.
- d. 다중상속을 가능하게 해준다.
- e. 패키지간의 연결을 도와준다.

**[정답]** e

**[해설]** 인터페이스를 사용하는 이유와 그 장점을 정리해 보면 다음과 같다.

#### 1. 개발시간을 단축시킬 수 있다.

일단 인터페이스가 작성되면, 이를 사용해서 프로그램을 작성하는 것이 가능하다. 메서드를 호출하는 쪽에서는 메서드의 내용에 관계없이 선언부만 알면 되기 때문이다.

그리고 동시에 다른 한 쪽에서는 인터페이스를 구현하는 클래스를 작성하도록 하여, 인터페이스를 구현하는 클래스가 작성될 때까지 기다리지 않고도 양쪽에서 동시에 개발을 진행할 수 있다.

#### 2. 표준화가 가능하다.

프로젝트에 사용되는 기본 틀을 인터페이스로 작성한 다음, 개발자들에게 인터페이스를 구현하여 프로그램을 작성하도록 함으로써 보다 일관되고 정형화된 프로그램의 개발이 가능하다.

#### 3. 서로 관계없는 클래스들에게 관계를 맺어 줄 수 있다.

서로 상속관계에 있지도 않고, 같은 조상클래스를 가지고 있지 않은 서로 아무런 관계도 없는 클래스들에게 하나의 인터페이스를 공통적으로 구현하도록 함으로써 관계를 맺어 줄 수 있다.

#### 4. 독립적인 프로그래밍이 가능하다.

인터페이스를 이용하면 클래스의 선언과 구현을 분리시킬 수 있기 때문에 실제구현에 독립적인 프로그램을 작성하는 것이 가능하다. 클래스와 클래스간의 직접적인 관계를 인터페이스를 이용해서 간접적인 관계로 변경하면, 한 클래스의 변경이 관련된 다른 클래스에 영향을 미치지 않는 독립적인 프로그래밍이 가능하다.

**[7-25]** Outer 클래스의 내부 클래스 Inner의 멤버변수 iv의 값을 출력하시오.

**[연습문제]/ch10/Exercise7\_25.java**

```
class Outer {          // 외부 클래스
    class Inner {      // 내부 클래스(인스턴스 클래스)
        int iv=100;
    }
}

class Exercise7_25 {
    public static void main(String[] args) {
        Outer o = new Outer();
        Outer.Inner ii = o.new Inner();
        System.out.println(ii.iv);
    }
}
```

**[실행결과]**

```
100
```

**[해설]** 내부 클래스(인스턴스 클래스)의 인스턴스를 생성하기 위해서는 먼저 외부클래스의 인스턴스를 생성해야한다. 왜냐하면 '인스턴스 클래스'는 외부 클래스의 '인스턴스 변수'처럼 외부 클래스의 인스턴스가 생성되어야 쓸 수 있기 때문이다.

[7-26] Outer클래스의 내부 클래스 Inner의 멤버변수 iv의 값을 출력하시오.

**[연습문제]/ch10/Exercise7\_26.java**

```
class Outer {           // 외부 클래스
    static class Inner { // 내부 클래스(static클래스)
        int iv=200;
    }
}

class Exercise7_26 {
    public static void main(String[] args) {
        Outer.Inner ii = new Outer.Inner();
        System.out.println(ii.iv);
    }
}
```

**[실행결과]**

```
200
```

**[해설]** 스태틱 클래스(static inner class)는 인스턴스 클래스와 달리 외부 클래스의 인스턴스를 생성하지 않고도 사용할 수 있다. 마치 static멤버를 인스턴스 생성없이 사용할 수 있는 것처럼.

**[7-27]** 다음과 같은 실행결과를 얻도록 (1)~(4)의 코드를 완성하시오.

**[연습문제]/ch10/Exercise7\_27.java**

```

class Outer {
    int value=10;           // Outer.this.value

    class Inner { // 인스턴스 클래스(instance inner class)
        int value=20;       // this.value

        void method1() {
            int value=30; // value

            System.out.println(      value);
            System.out.println(      this.value);
            System.out.println(Outer.this.value);
        }
    } // Inner클래스의 끝
} // Outer클래스의 끝

class Exercise7_27 {
    public static void main(String args[]) {
        Outer outer = new Outer();
        Outer.Inner inner = outer.new Inner();

        inner.method1();
    }
}

```

**[실행결과]**

```

30
20
10

```

**[해설]** 외부 클래스와 내부 클래스에 같은 이름의 인스턴스 변수(value)가 선언되었을 때 어떻게 구별하는가에 대한 문제이다. 외부 클래스의 인스턴스 변수는 내부 클래스에서 ‘외부클래스이름.this.변수이름’로 접근할 수 있다.

내부 클래스의 종류가 인스턴스 클래스이기 때문에 외부 클래스의 인스턴스를 생성한 다음에야 내부 클래스의 인스턴스를 생성할 수 있다.

[7-28] 아래의 EventHandler를 익명 클래스(anonymous class)로 변경하시오.

[연습문제]/ch7/Exercise7\_28.java

```
import java.awt.*;
import java.awt.event.*;

class Exercise7_28
{
    public static void main(String[] args)
    {
        Frame f = new Frame();
        f.addWindowListener(new EventHandler());
    }
}

class EventHandler extends WindowAdapter
{
    public void windowClosing(WindowEvent e) {
        e.getWindow().setVisible(false);
        e.getWindow().dispose();
        System.exit(0);
    }
}
```

[정답]

[연습문제]/ch10/Exercise7\_28\_2.java

```
import java.awt.*;
import java.awt.event.*;

class Exercise7_28_2
{
    public static void main(String[] args)
    {
        Frame f = new Frame();
        f.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                e.getWindow().setVisible(false);
                e.getWindow().dispose();
                System.exit(0);
            }
        });
    } // main
}
```

**[7-29]** 지역 클래스에서 외부 클래스의 인스턴스멤버와 static멤버에 모두 접근할 수 있지만, 지역변수는 final이 붙은 상수만 접근할 수 있는 이유 무엇인가?

**[정답]** 메서드가 수행을 마쳐서 지역변수가 소멸된 시점에도, 지역 클래스의 인스턴스가 소멸된 지역변수를 참조하려는 경우가 발생할 수 있기 때문이다.

**[해설]** 아직 쓰레드를 배우지 않았지만, 쓰레드를 사용해서 상황을 만들어 보았다.

#### 【연습문제】/ch10/Exercise7\_29.java

```
import java.awt.*;
import java.awt.event.*;

class Exercise7_29
{
    public static void main(String[] args)
    {
        final int VALUE = 10; // 외부 클래스의 지역변수

        Thread t = new Thread(new Runnable() { // 익명 클래스(내부 클래스)
            public void run() {
                for(int i=0; i < 10;i++) { // 10번 반복한다.
                    try {
                        Thread.sleep(1*1000); // 1초간 멈춘다.
                    } catch(InterruptedException e) {}

                    System.out.println(VALUE); // 외부 클래스의 지역변수를 사용
                }
            } // run()
        });

        t.start(); // 쓰레드를 시작한다.
        System.out.println("main() - 종료.");
    } // main
}
```

#### 【실행결과】

```
main() - 종료.
```

```
10
10
10
10
10
10
10
10
10
10
```

실행결과를 보면 main메서드가 종료된 후에도 지역변수 VALUE의 값을 사용하고 있다는 것을 알 수 있다. 지역변수는 메서드가 종료되면 함께 사라지지만, 상수의 경우 이미 컨스턴트 풀(constant pool, 상수를 따로 모아서 저장해 놓는 곳)에 저장되어 있기 때문에 사용할 수 있는 것이다.