```java
//* 섯다카드 20장을 포함하는 섯다카드 한 벌(SutdaDeck클래스)을 정의한 것이다.
// 섯다카드 20장을 담는 SutdaCard배열을 초기화하시오.
// 단, 섯다카드는 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.
// */


class SutdaDeck {
    final int CARD_NUM = 20;
    SutdaCard[] cards = new SutdaCard[CARD_NUM];
    // SutdaCard 객채배열 생성

    SutdaDeck() {
        for (int i = 0; i < cards.length; i++) {
            // i는 0부터 19까지 증가한다.
            // i 0~9까지 한쌍, 10~19까지 한상.
            // i%10으로 나누면 i가0일때 0...9일때 9, 이렇게 1씩
            // 0~9까지 증가하고 i가 10이 되는 순간 다시 0~9가 된다.
            // 카드의 숫자는 1-10까지 이므로 +1을 하면 된다.
            int num = i % 10 + 1;
            // num의 값이 1,3,8일 때, 한 쌍의 카드 중에서 하나는 광(kwang)
            // 첫번째 세트인 i가 0~9일때, 그리고 num이 1, 3, 8일때만 true인 걸로
            // AND가 OR보다 우선순위가 높기 때문에 괄호는 필수다.
            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]+",");
        // 1K,2,3K,4,5,6,7,8K,9,10,1,2,3,4,5,6,7,8,9,10,
    }
}
```

```java
//* 섯다카드 20장을 포함하는 섯다카드 한 벌(SutdaDeck클래스)을 정의한 것이다.
// 섯다카드 20장을 담는 SutdaCard배열을 초기화하시오.
// 단, 섯다카드는 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.
// */


class SutdaDeck {
    final int CARD_NUM = 20;
    SutdaCard[] cards = new SutdaCard[CARD_NUM];
    // SutdaCard 객채배열 생성

    // 생성자 안에서 cards 객체배열의 객체 만들어서 초기화 시켜주기
    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);
        }
    }

    // 배열 cards에 담긴 카드의 위치를 뒤섞는다.(Math.random()사용)
    void shuffle() {
        for (int i = 0; i < cards.length; i ++) {
            int random = (int) (Math.random() * cards.length);
            SutdaCard temp = cards[i];
            cards[i] = cards[random];
            cards[random] = temp;
        }
    }

    // 배열 cards에서 지정된 위치의 SutdaCard를 반환한다.
    SutdaCard pick(int index) {
        // index 유효성 검사
        if (!(index >= 0 && index < CARD_NUM)) {
            return null;
        }
        return cards[index];
    }

    // 배열 cards에서 임의의 위치의 SutdaCard를 반환한다.(Math.random()사용)
    SutdaCard pick() {
        // pick(int index)를 호출한다
        int index = (int) (Math.random() * cards.length);
        return pick(index);
    }

}

class SutdaCard {
// 섯다카드의 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.
// 따라서 인스턴스 변수 앞에 fianl을 붙여 상수로 만들고, 변수명을 대문자로 한다.
```

```
// 원래 상수는 선언과 초기화를 동시에 하지만, 인스턴스변수의 경우 선언시에 초기화 하지 않고,
// 생성자에서 초기화 할 수 있다. 생성할 때 지정된 값이 변하지 않도록 할 수 있는 것이다.
// 상수이므로 한번 초기화 한 이후로는 값을 바꿀 수 없다.
    final int NUM;
    final boolean IS_KWANG;

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

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

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

class Exercise7_2 {
    public static void main(String args[]) {
        SutdaDeck deck = new SutdaDeck();
        System.out.println(deck.pick(0));
        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(0));
    }
}
```

매개변수가 있는 메서드는 반드시 작업 전에 유효성 검사를 해야 한다. (index 범위를 넘어서면 ArrayIndexOutOfBoundsException 오류가 발생할 수 있다).

[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.

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.

[7-6] 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.

[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 인스턴스변수 x는 200이 된다. getX()는 조상인 Parent클래스에 정의된 것이라서, getX()에서 x는 Parent클래스의 인스턴스변수 x를 의미한다. 그래서 x=200이 출력된다.

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	변수 앞에 final이 붙으면, 값을 변경할 수 없는 상수가 된다.
	지역변수	

```
class MyTv2 {
    private boolean isPowerOn;
    private int channel;
    private int previous_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 setPower(boolean power) {
        isPowerOn = power;
    }

    public boolean getPower() {
        return isPowerOn;
    }

    public void setChannel(int channel) {
        previous_channel = this.channel; // 현재 채널을 이전 채널에 저장
        if (!(channel >= MIN_CHANNEL && channel <= MAX_CHANNEL)) {
            return;
        }
        this.channel = channel;
    }

    // 이전 채널로 이동하는 기능의 메서드
    public void gotoPrevChannel() {
        setChannel(previous_channel); // 현재 채널을 이전 채널로 변경
    }

    public int getChannel() {
        return channel;
    }

    public void setVolume(int volume) {
        if (!(volume >= MIN_VOLUME && volume <= MAX_VOLUME)) {
            return;
        }
        this.volume = volume;
    }

    public int getVolume() {
        return volume;
    }
```

```
}

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());
    }
}
```

[7-13] Math클래스의 생성자는 접근 제어자가 private이다. 그 이유는 무엇인가?
[정답] Math클래스의 모든 메서드가 static메서드이고 인스턴스변수가 존재하지 않기 때문에 객체를 생성할 필요가 없기 때문

[해설] Math클래스는 몇 개의 상수와 static메서드만으로 구성되어 있기 때문에 인스턴스를 생성할 필요가 없다. 그래서 외부로부터의 불필요한 접근을 막기 위해 다음과 같이 생성자의 접근 제어자를 private으로 지정하였다.

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

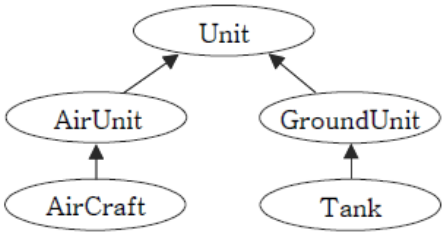
[7-15] 클래스가 다음과 같이 정의되어 있을 때, 형변환을 올바르게 하지 않은 것은? (모두 고르시오.)

```
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
```

a,b) Aircraft타입 참조변수 ac가 참조하고 있는 인스턴스(AirCraft)를 참조변수 u가 참조하도록 한다. u는 Unit타입의 참조변수로 인스턴스AirCraft의 조상이다. 조상타입의 참조변수로 자손타입의 인스턴스를 참조하는 것이 가능하기 때문에 형변환이 가능하며, 업캐스팅(자손->조상)이므로 생략도 가능하다.

c) Unit타입 참조변수 u가 참조하고 있는 인스턴스(GroundUnit)를 참조변수 gu가 참조하도록 한다. gu는 Ground타입의 참조변수로 인스턴스(GroundUnit)과 타입이 일치하기 때문에, 형변환이 가능하다. 다운캐스팅(조상->자손)이므로 형변환 생략이 불가능하다.

d) AirCraft타입 참조변수 ac가 가르키고 있는 인스턴스(AirCraft)를 참조변수 au가 참조하도록 한다. au는 AirUnit 타입의 참조변수로 인스턴스AirCraft의 조상이다. 조상타입의 참조변수로 자손타입의 인스턴스를 참조하는 것이 가능하기 때문에 형변환이 가능하며, 업캐스팅(자손->조상)이므로 생략도 가능하다.

e) Unit타입 참조변수 u가 참조하고 인스턴스(GroundUnit)을 참조변수 t가 참조하게 한다. t는 Tank타입의 참조변수로 인스턴스GroundUnit의 자손이다. **자손타입의 참조변수로 조상 타입의 인스턴스를 참조하게 하는 것은 허용하지 않으므로, 실행시 에러가 발생한다**.

f) 참조변수 t가 참조하고 있는 인스턴스(Tank)를 참조변수 gu가 참조하게 한다. gu는 GroundUnit타입의 참조변수로 인스턴스Tank의 조상이다. 조상타입의 참조변수로 자손타입의 인스턴스를 참조하는 것이 가능하기 때문에 형변환이 가능하며, 업캐스팅(자손->조상)이므로 생략도 가능하다.

Resources: 자바의 정석