Chapter 12

지네릭스, 열거형, 애너테이션

Generics, Enumeration, Annotation

[연습문제]

[12-1] 클래스 Box가 다음과 같이 정의되어 있을 때, 다음 중 오류가 발생하는 문장은? 경고가 발생하는 문장은?

```
class Box<T> { // 지네릭 타입 T를 선언
    T item;

    void setItem(T item) { this.item = item; }
    T getItem() { return item; }
}
```

a. Box<Object> b = new Box<String>(); •대입된 타입이 반드시 같아야 한다.
b. Box<Object> b = (Object)new Box<String>();
c. new Box<String>().setItem(new Object()); 대입된 타입이 String이므로, setItem(T item)의 매개변수 역시, String타입만 허용된다.
d. new Box<String>().setItem("ABC");

Object타입을 Box<Object>타입의 참조변수에 저장불가

[12-2] 지네릭 메서드 makeJuice()가 아래와 같이 정의되어 있을 때, 이 메서드를 올바르게 호출한 문장을 모두 고르시오. (Apple과 Grape는 Fruit의 자손이라고 가정하자.)

```
class Juicer {
    static <T extends Fruit> String makeJuice(FruitBox<T> box) {
        String tmp = "";
        for(Fruit f : box.getList()) tmp += f + " ";
        return tmp;
    }
}
```

a. Juicer.<Apple>makeJuice(new FruitBox<Fruit>());
b. Juicer.<Fruit>makeJuice(new FruitBox<Grape>());
c. Juicer.<Fruit>makeJuice(new FruitBox<Fruit>());
d. Juicer.makeJuice(new FruitBox<Apple>());
e. Juicer.makeJuice(new FruitBox<Object>());

제네릭 메소드에 대입된 타입이 Apple이므로, 이 메소드의 매개변수는 'FruitBox<Apple>'타입이 된다. new FruitBox<Fruit>()는 매개변수의 타입과 일치하지 않으며, 자동형변환도 불가능한 타입이다.

제네릭 메소드의 타입 호출이 생략되지 않았다면, 'Juicer.<Object>makeJuice(new FruitBox<Object>());'과 같다. d번의 경우와 같이 타입이 일치하긴 하지만, <T extends Fruit>로 제 걸려있으므로, 타입 T는 Fruit의 자손이어야 한다. Object는 Fruit의 자손이 아니다

[12-3] 다음 중 올바르지 않은 문장을 모두 고르시오.

```
class Box<T extends Fruit> { // 지네릭 타입 T를 선언
    T item;

    void setItem(T item) { this.item = item; }
    T getItem() { return item; }
}
```

a. Box<?> b = new Box();
b. Box<?> b = new Box<>();
c. Box<?> b = new Box<Object>();
d. Box<Object> b = new Box<Fruit>();
e. Box b = new Box<Fruit>();
f. Box<? extends Fruit> b = new Box<Apple>();
g. Box<? extends Object> b = new Box<? extends Fruit>();

[12-4] 아래의 메서드는 두 개의 ArrayList를 매개변수로 받아서, 하나의 새로운 ArrayList로 병합하는 메서드이다. 이를 지네릭 메서드로 변경하시오.

<T extends Product> ArrayList<T>

```
public static ArrayList<? extends Product> merge(
    ArrayList<? extends Product> list, ArrayList<? extends Product> list2) {
        ArrayList<? extends Product> newList = new ArrayList<>(list);
                                        지운 자리에 <T> 추가
        newList.addAll(list2);

        return newList;
}
```

[12-5] 아래는 예제7-3에 열거형 Kind와 Number를 새로 정의하여 적용한 것이다. (1)에 알맞은 코드를 넣어 예제를 완성하시오. (Math.random()을 사용했으므로 실행결과가 달라질 수 있다.)

[연습문제]/ch12/Exercise12_5.java

```
class DeckTest {
   public static void main(String args[]) {
        Deck d = new Deck();     // 카드 한 벌(Deck)을 만든다.
        Card c = d.pick(0);      // 섞기 전에 제일 위의 카드를 뽑는다.
        System.out.println(c);   // System.out.println(c.toString());과 같다.

        d.shuffle();             // 카드를 섞는다.
        c = d.pick(0);           // 섞은 후에 제일 위의 카드를 뽑는다.
        System.out.println(c);
   }
}

class Deck {
   final int CARD_NUM = Card.Kind.values().length
                                    * Card.Number.values().length; // 카드의 개수
   Card cardArr[] = new Card[CARD_NUM];  // Card객체 배열을 포함

   Deck () {
        /*
            (1
        */
   }

   Card pick(int index) {     // 지정된 위치(index)에 있는 카드 하나를 꺼내서 반환
        return cardArr[index];
   }

   Card pick() {              // Deck에서 카드 하나를 선택한다.
        int index = (int)(Math.random() * CARD_NUM);
        return pick(index);
   }

   void shuffle() { // 카드의 순서를 섞는다.
        for(int i=0; i < cardArr.length; i++) {
            int r = (int)(Math.random() * CARD_NUM);
```

```
int i=0;
for(Card.Kind kind : Card.Kind.values()) {
    for(Card.Number num : Card.Number.values()) {
        cardArr[i++] = new Card(kind, num);
    }
}
```

```
            Card temp = cardArr[i];
            cardArr[i] = cardArr[r];
            cardArr[r] = temp;
        }
    }
} // Deck클래스의 끝

// Card클래스
class Card {
    enum Kind   { CLOVER, HEART, DIAMOND, SPADE }
    enum Number {
        ACE, TWO, THREE, FOUR, FIVE,
        SIX, SEVEN, EIGHT, NINE, TEN,
        JACK, QUEEN, KING
    }

    Kind   kind;
    Number num;

    Card() {
        this(Kind.SPADE, Number.ACE);
    }

    Card(Kind kind, Number num) {
        this.kind  = kind;
        this.num   = num;
    }

    public String toString() {
        return "[" + kind.name() + "," + num.name() + "]";
    } // toString()의 끝
} // Card클래스의 끝
```

[실행결과]

```
[CLOVER,ACE]
[HEART,TEN]
```

[12-6] 다음 중 메타 애너테이션이 아닌 것을 모두 고르시오.

a. Documented
b. Target
c. Native
d. Inherited

어노테이션 표 참고

애너테이션	설명
@Override	컴파일러에게 오버라이딩하는 메서드라는 것을 알린다.
@Deprecated	앞으로 사용하지 않을 것을 권장하는 대상에 붙인다.
@SuppressWarnings	컴파일러의 특정 경고메시지가 나타나지 않게 해준다.
@SafeVarargs	지네릭스 타입의 가변인자에 사용한다.(JDK1.7)
@FunctionalInterface	함수형 인터페이스라는 것을 알린다.(JDK1.8)
@Native	native메서드에서 참조되는 상수 앞에 붙인다.(JDK1.8)
@Target*	애너테이션이 적용가능한 대상을 지정하는데 사용한다.
@Documented*	애너테이션 정보가 javadoc으로 작성된 문서에 포함되게 한다.
@Inherited*	애너테이션이 자손 클래스에 상속되도록 한다.
@Retention*	애너테이션이 유지되는 범위를 지정하는데 사용한다.
@Repeatable*	애너테이션을 반복해서 적용할 수 있게 한다.(JDK1.8)

[12-7] 애너테이션 TestInfo가 다음과 같이 정의되어 있을 대, 이 애너테이션이 올바르게 적용되지 않은 것은?

```
@interface TestInfo {
    int count() default 1;
    String[] value() default "aaa";
}
```

a. @TestInfo class Exercise12_7 {}
b. @TestInfo(1) class Exercise12_7 {}
c. @TestInfo("bbb") class Exercise12_7 {}
d. @TestInfo("bbb","ccc") class Exercise12_7 {}

b. 요소의 이름이 value가 아닌 경우에는 요소의 이름을 생략할 수 없다.
@TestInfo(count=1)

d. 요소의 타입이 배열이고, 지정하려는 값이 여러 개인 경우 괄호{ }가 필요.
@TestInfo({"bbb", "ccc"}) 또는 @TestInfo(value={"bbb", "ccc"})