

## JAVA 프로그래밍 4 ~ 7강 보충학습

### 제 3 장 - 객체지향 프로그래밍 연습 문제

1. 다음 프로그램의 두 가지 문제점을 지적하라.

```
final class First {  
    private int a = 1;  
    int b = 2;  
}  
class Second extends First {  
    public void method() {  
        System.out.println(a + b);  
    }  
}
```

(답) 클래스 Second는 final 클래스인 First를 상속할 수 없다. 클래스 Second의 메소드 method()에서 클래스 First의 private 필드 a에 접근할 수 없다.

2. 다음 프로그램은 컴파일되지 않는다. 이유는 무엇인가?

```
public class Fred {  
    public int x = 0;  
    public fred (int x) {  
        this.x = x;  
    }  
}
```

(답) fred는 생성자가 아니기 때문에 반환형이 명시되어야 하기 때문이다.

3. 다음 프로그램의 출력 결과가 무엇인지 설명하라.

```
class Mystery {  
    String s;  
    public static void main(String[] args) {  
        Mystery m = new Mystery(); // ①  
        m.go();  
    }  
    void Mystery() { // ②  
        s = "constructor";  
    }  
    void go() {  
        System.out.println(s);  
    }  
}
```

(답) 여기서 Mystery 메소드는 생성자가 아니므로 실행되지 않는다. 따라서 s의 초기값인 null이 출력된다.

☞ ②의 void Mystery()는 클래스와 이름이 같은 메소드이나 반환형이 명시되어 있으므로 생성자가 아닌 일반 메소드이다. 클래스 Mystery에는 명시적인 생성자 정의가 없으므로 인

자 없는 기본 생성자가 컴파일러에 의해 자동으로 제공된다. 따라서 ①에 의해 객체가 생성될 때는 void Mystery()가 아닌 컴파일러가 제공하며 아무 것도 수행하지 않는 기본 생성자가 실행될 뿐이다. 객체가 생성될 때 필드의 초기 값이 주어지지 않으면 0이나 null로 자동 초기화된다. 따라서 객체 m이 가지고 있는 필드 s가 참조형이므로 그것의 초기 값인 null이 출력된다.

생성자는 객체가 생성될 때 암묵적이면서 자동으로 실행되는 메소드로, 일반적인 메소드 선언 방식과 같지만 반환 자료형을 정의할 수 없으며, 생성자의 이름은 클래스의 이름과 같다.

4. 다음 프로그램의 문제점은 무엇인가?

```
protected class Fred {
    private int x = 0;
    private Fred (int xval) {
        x=xval;
    }
}
```

(답) protected라는 클래스 접근 제어자는 여기서 사용할 수 없다.

5. 다음 프로그램은 컴파일되지 않는다. 이유는 무엇인가?

```
public class Test {
    int x;
    public static void main (String args[]) {
        x = 8;
    }
}
```

(답) static 함수 main에서 static이 아닌 x를 참조하고 있기 때문이다.

6. 다음 프로그램을 컴파일 할 때 생성되는 클래스 파일(.class)은 몇 개인가?

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class MouseInnerDemo extends Applet {
    ...
    public void init() {
        addMouseListener(new MouseAdapter() { ... });
        addMouseMotionListener(new MouseMotionAdapter() { ... });
    } // init() 메소드의 끝

    public void paint(Graphics g) { ... }
}
```

(답) 3개이며 생성되는 파일은 MouseInnerDemo.class, MouseInnerDemo\$1.class, 그리고 MouseInnerDemo\$2.class 이다.

①과 ②에서 예외처리에 사용되는 2개의 익명 클래스가 존재하므로 전부 3개의 클래스 파일이 생성된다. 익명 클래스는 클래스를 따로 정의하지 않고 바로 객체를 생성시키는 것을 말한다.

다. 익명 클래스는 클래스를 상속하거나 인터페이스를 구현할 때 적용할 수 있다. 컴파일 결과로 만들어지는 파일의 이름은 각각 MouseInnerDemo.class, MouseInnerDemo\$1.class, 그리고 MouseInnerDemo\$2.class 이다.

보통의 경우 객체 생성을 위한 명령은 “new 클래스이름()”와 같은 형식이다. 만약 “new 클래스이름() { }”와 같은 형식이 있다면 중괄호 블럭이 익명 클래스의 정의를 의미한다. 이 때 나오는 클래스 이름은 익명 클래스의 슈퍼 클래스가 된다.

7. 다음 프로그램의 실행 결과는 무엇인가?

```
class myInt {
    int x;
    public myInt(int x) { this.x = x; }
}
public class Example {
    public static void main(String args[]) {
        myInt x1 = new myInt(10); myInt x2 = new myInt(10);
        method(x1, x2);
        System.out.println(x1.x + ", " + x2.x);
    }
    public static void method(myInt v1, myInt v2) {
        v2.x = 20;
        v1=v2;
    }
}
```

(답) 10,20

8. 어떤 클래스에서 다음과 같은 메소드가 존재할 때, 하위 클래스에서 오버라이딩을 못하게 하는 두 개를 고르고 설명하라.

- a) final void methoda() {}                      b) void final methoda() {}
- c) static void methoda() {}                    d) static final void methoda() {}
- e) final abstract void methoda() {}

(답) a와 d가 그러하다. b와 e는 문법적으로 오류가 있다.

☞ final 메소드는 구현되어 있으며, 상속은 가능하지만 서브 클래스에서 오버라이딩이 불가능한 메소드이다. 따라서 abstract와 사용될 수 없으며 e)는 잘못된 문장이다. 그리고 b)는 void와 final을 바꾸어 썼으므로 잘못된 문장이다. c)는 재정의 가능하다.

9. 생성자에서 슈퍼 클래스의 기본 생성자를 호출하는 문장을 써라.

(답) super()

☞ 교재 91p의 예제 3-35를 다음과 같이 수정하였다.

```
class SuperClass{
    public double x;
    public SuperClass(){ //SuperClass 클래스의 기본 생성자를 정의
```

```

        this.x=0;
    }
    public SuperClass(double new_x){ //SuperClass 클래스의 다른 생성자를 정의
        this.x=new_x*10;
    }
}
class SubClass extends SuperClass{
    double x;
    public SubClass(){ //SubClass 클래스의 기본 생성자 정의
        super();      //슈퍼 클래스의 기본 생성자를 호출
        this.x=0;
    }
    public SubClass(double new_x){ //SubClass 클래스의 다른 생성자 정의
        super(new_x);    //슈퍼 클래스의 생성자를 호출
        this.x=new_x;
    }
    public double getSuper(){ return super.x; }
    public double getSub(){ return this.x; }
}

```

super()는 슈퍼 클래스의 생성자 호출을, this()는 자신의 다른 생성자 호출을 의미한다. 여러 생성자가 있을 수 있으므로 기본 생성자 호출이 아닌 경우에는 필요한 인자를 괄호 안에 써 주어야 한다. this()와 super()는 생성자 몸체(중괄호 블록)에서만 사용할 수 있으며 몸체의 맨 앞에서 호출되어야 한다.

위 프로그램에서 정의된 SubClass 유형으로 객체가 생성된다면 이 객체는 상속받은 필드 x와 자신의 클래스에서 정의된 x 즉, 같은 이름으로 두 개의 x를 가지게 된다. 이것을 구분하기 위해 메소드의 몸체에서 this.x와 super.x를 사용할 수 있다.

10. 다음 프로그램은 컴파일 되지 않는다. 그 이유는 무엇인가?

```

class A {
    int i;
    A(int i) {
        this.i = i * 2;
    }
}
class B extends A {
    public static void main(String[] args) {
        B b = new B(2);
    }
    B(int i) {
        System.out.println(i);
    }
}

```

(답) B의 생성자에서 묵시적으로 A의 기본 생성자를 호출하나 존재하지 않기 때문이다.

☞ B의 생성자 몸체에서 명시적으로 슈퍼 클래스 생성자를 호출하고 있지 않다. 이런 경우 묵시적으로 슈퍼 클래스인 A의 기본 생성자를 호출하나 A에는 기본 생성자가 존재하지 않기 때문이다.

클래스 정의시 아무런 생성자 정의도 하지 않으면, 인자를 갖지 않는 기본 생성자가 자동으로 만들어진다. 그러나 클래스 A에는 생성자가 하나 있으므로 기본 생성자는 자동으로 만들어지지 않는다.

11. 아래 프로그램에서 modify 메소드는 static 데이터 a의 값을, 인자로 전달되는 값만큼 증가시킨다. 밑줄을 완성하고 출력 결과를 설명하라.

```
public class X{
    private static int a=2;
    public static void main(String args[]){
        modify(a);
        System.out.println(a);
    }
    public static void modify(int a){
                
    }
}
```

(답) X.a += a; 출력 결과는 4이다.

12. 다음 프로그램의 실행 결과를 예상하라.

```
class Super {
    int index = 5;
    public void printVal () {
        System.out.println("Super");
    }
}
class Sub extends Super {
    int index = 2;
    public void printVal () {
        System.out.println("Sub");
    }
}
public class Runner {
    public static void main(String[] args) {
        Super sup=new Sub();
        System.out.print(sup.index + " "); // ①
        sup.printVal (); // ②
    }
}
```

(답) 5 Sub

☞ 위 프로그램에서 정의된 클래스인 Sub 유형으로 객체가 생성된다면 이 객체는 index라는 이름으로 두 개의 필드를 가진다. 또한 클래스 Sub에서는 상속된 메소드 printVal()을 재정의하고 있다.

main 함수에서 변수 sup의 선언 유형은 Super이다. 이 참조 변수는 Super 유형의 객체를 참조할 수도 있으며 Sub 유형의 객체도 참조할 수 있다. 위 프로그램의 경우 sup는 Sub 유형의 객체를 참조하고 있다.

①에서 sup.index는 sub가 참조하는 객체가 가지고 있는 두 개의 index 필드 중에서 상속받은 필드를 의미한다. 왜냐하면 필드를 따질 때는 sup의 선언 유형을 따르기 때문이다. 따라서 5가 출력된다(정적 바인딩).

sup가 참조하는 객체의 실제 유형은 Sub이다. 따라서 ②에 의해 호출되는 메소드는 Sub의 printVal()이다. 왜냐하면 메소드를 따질 때는 sup의 실제 유형을 따르기 때문이다. 따라서 출력되는 값은 Sub이다(동적 바인딩).

13. 다음에 주어진 인터페이스를 구현하는 클래스 B의 가장 간단한 정의를 작성하여라.

```
interface A {
    int method1(int i);
    int method2(int j);
}
```

(답)

```
class B implements A {
    public int method1(int i) { return 1; }
    public int method2(int j) { return 2; }
}
```

클래스가 인터페이스를 상속하는 경우 'implements' 키워드를 사용한다. 이것을 “클래스가 인터페이스를 구현한다”고도 하는데 클래스에서 인터페이스에 정의된 모든 추상 메소드를 구현해 주어야 한다. 이 때 주의할 점은 인터페이스에서 그 형식이 명시된 메소드의 접근 제어자가 public이라는 점이다. 따라서 인터페이스를 구현하는 클래스를 정의할 때, 구현하고 있는 메소드의 접근 제어자를 public으로 해야 한다.

14. try 블록 다음에 반드시 나와야 하는 블록은 무엇인가?

(답) 일반적으로 catch 블록이 나와야 하나, 예외가 전파된다면 finally 블록만 나올 수도 있다.

자바에서는 try - catch 구문을 이용하여 예외를 처리(exception handling)하는 코드를 넣을 수 있다. 예외가 발생할 가능성이 있는 명령문은 try 블록에 넣고, 예외가 발생한 경우에 이것의 처리를 위한 명령문은 catch 블록에 넣는다. 일반적으로 try 블록 다음에 catch 블록이 나와야 한다. 그러나 예외를 전파시킬 수만 있다면 catch 블록 없이 finally 블록만 나올 수도 있다. finally 블록은 예외 발생과 무관하게 항상 실행된다.

결론적으로 try 블록 다음에 catch 블록이나 finally 블록 중 하나는 나와야 한다. 즉 다음 구조 중 하나가 된다. try - catch , try - catch - finally, try - finally

15. 예외가 발생한 곳에서 try-catch 절을 이용하여 예외처리를 할 수도 있지만, 현재 메소드를 호출한 메소드에게 발생한 예외를 전달하여 처리할 수도 있다. 이와 같이 예외 처리의 위임을 나타낼 때 사용되는 자바 키워드는 무엇인가?

(답) throws

16. 예외 상황을 정의하기 위해 상속받아야 하는 클래스는 무엇인가?

(답) Exception

17. 예외 상황을 발생시키기 위해서 사용하는 키워드는 무엇인가?

(답) throw

18. 다음 프로그램을 실행하면 NullPointerException이 발생한다. 그 이유를 설명하라.

```
public class X {
    public static void main(String[] args) {
        Object[] foo = new Object[1]; //①
        Object bar = foo[0]; //②
        String baz = bar.toString(); //③
    }
}
```

(답) bar(또는 foo[0])이 null 이기 때문이다. 문장 foo[0]=new Object(); 이 필요하다.

①의 문장을 실행하면 foo[0]이 만들어지며 이것의 참조값은 null이 된다. ②의 문장을 실행하면 bar도 null 이 된다. ③에서 null 객체를 가지고 메소드를 호출하였으므로 NullPointerException이 발생하게 된다.

①과 ② 사이에 “foo[0]=new Object();”와 같은 명령으로 객체 생성을 하고, 이것의 참조값을 foo[0]이 가지게 하면 예외가 발생하지 않을 것이다.

19. 다음 프로그램의 실행 결과는 무엇인가?

```
public class A {
    public void problem() throws RuntimeException {
        throw new RuntimeException();
    }
    public void trythis() {
        try {
            System.out.print("1");
            problem();
        } catch (RuntimeException x) {
            System.out.print("2");
        } catch (Exception x) {
            System.out.print("3");
        } finally {
            System.out.print("4");
        }
        System.out.print("5");
    }
    public static void main(String[] args){
        A a=new A();
        a.trythis();
    }
}
```

```
    }
}
```

(답) 1245

20. 다음 프로그램의 실행(또는 컴파일) 결과를 예상하라.

```
class SuperClass {}
class SubClass extends SuperClass {}
public class Test {
    public static void test(SuperClass foo) {
        SubClass bar = (SubClass) foo; //5번 라인
    }
    public static void main(String args[]) {
        SuperClass sup1=new SubClass();
        test(sup1);
        SuperClass sup2=new SuperClass();
        test(sup2); //11번 라인
    }
}
```

(답) 11번 라인의 호출로 인한 5번 라인에서 ClassCastException 이 발생한다.

21. 다음 프로그램에서 밑 줄 부분에 들어갈 문제가 없는 것은?

```
public class Test { }
```

- a) package kim.java;
- b) import java.lang.\*;
- c) public class Test1 { }
- d) class Test1
- e) int i=0 ;

(답) a, b, d가 들어 갈 수 있는데, 함께 들어간다면 a, b, d 순으로 들어가야 한다.

자바의 패키지는 다른 프로그래밍 언어의 라이브러리(library)와 비슷한 개념으로 하나의 패키지는 관련된 클래스 파일들과 서브 패키지들로 구성된다. 사용자 정의 패키지를 만들고 여기에 클래스 파일을 넣고 싶을 때는 소스 코드의 맨 앞쪽에 'package <패키지 이름>' 구문을 넣어주고, 패키지를 사용하고 싶을 때는 'import <패키지 이름>.\*'의 형식으로 패키지들을 지정하여 주어야 한다.

위 프로그램에서 c)가 포함될 수 없는 이유는 이미 public 클래스가 존재하기 때문이다.

22. import 문으로 포함시키지 않아도 자동적으로 포함되는 패키지는 무엇인가?

(답) java.lang