

[연습문제 - 모범답안]

[8-1] 예외처리의 정의와 목적에 대해서 설명하시오.

[정답]

정의 - 프로그램 실행 시 발생할 수 있는 예외의 발생에 대비한 코드를 작성하는 것

목적 - 프로그램의 비정상 종료를 막고, 정상적인 실행상태를 유지하는 것

[해설] 프로그램의 실행도중에 발생하는 에러는 어쩔 수 없지만, 예외는 프로그래머가 이에 대한 처리를 미리 해주어야 한다.

에러(error) - 프로그램 코드에 의해서 수습될 수 없는 심각한 오류

예외(exception) - 프로그램 코드에 의해서 수습될 수 있는 다소 미약한 오류

예외처리(exception handling)란, 프로그램 실행 시 발생할 수 있는 예기치 못한 예외의 발생에 대비한 코드를 작성하는 것이며, 예외처리의 목적은 예외의 발생으로 인한 실행 중인 프로그램의 갑작스런 비정상 종료를 막고, 정상적인 실행상태를 유지할 수 있도록 하는 것이다.

예외처리(exception handling)의

정의 - 프로그램 실행 시 발생할 수 있는 예외의 발생에 대비한 코드를 작성하는 것

목적 - 프로그램의 비정상 종료를 막고, 정상적인 실행상태를 유지하는 것

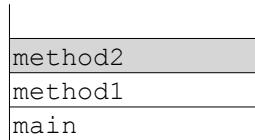
[8-2] 다음은 실행도중 예외가 발생하여 화면에 출력된 내용이다. 이에 대한 설명 중 옳지 않은 것은?

```
java.lang.ArithmetricException : / by zero
    at ExceptionEx18.method2(ExceptionEx18.java:12)
    at ExceptionEx18.method1(ExceptionEx18.java:8)
    at ExceptionEx18.main(ExceptionEx18.java:4)
```

- a. 위의 내용으로 예외가 발생했을 당시 호출스택에 존재했던 메서드를 알 수 있다.
- b. 예외가 발생한 위치는 method2 메서드이며, ExceptionEx18.java 파일의 12번째 줄이다.
- c. 발생한 예외는 ArithmetricException이며, 0으로 나누어서 예외가 발생했다.
- d. method2메서드가 method1메서드를 호출하였고 그 위치는 ExceptionEx18.java 파일의 8 번째 줄이다.

[정답] d

[해설] 예외의 종류는 ArithmetricException이고 0으로 나눠서 발생하였다. 예외가 발생한 곳은 method2이고 ExceptionEx18.java의 12번째 줄이다. 예외가 발생했을 당시의 호출스택을 보면 아래의 그림과 같다. 호출스택은 맨 위에 있는 메서드가 현재 실행 중인 메서드이고 아래 있는 메서드가 바로 위의 메서드를 호출한 것이다. 그래서 main → method1 → method2의 순서로 호출되었음을 알 수 있다.



괄호안의 내용은 예외가 발생한 소스와 라인인데, method1()의 경우 예외가 발생한 곳이 method2() 호출한 라인이고 main의 경우 method1()을 호출한 라인이다.

method1()에서 봤을 때는 method2()를 호출한 곳에서 예외가 발생한 것이기 때문이다. main에서도 역시 마찬가지.

[8-3] 다음 중 오버라이딩이 잘못된 것은? (모두 고르시오)

```
void add(int a, int b)
    throws InvalidNumberException, NotANumberException {}

class NumberException extends Exception {}
class InvalidNumberException extends NumberException {}
class NotANumberException extends NumberException {}
```

- a. void add(int a, int b) throws InvalidNumberException, NotANumberException {}
- b. void add(int a, int b) throws InvalidNumberException {}
- c. void add(int a, int b) throws NotANumberException {}
- d. void add(int a, int b) throws Exception {}
- e. void add(int a, int b) throws NumberException {}

[정답] d, e

[해설] 오버라이딩(overriding)을 할 때, 조상 클래스의 메서드보다 많은 수의 예외를 선언할 수 없다.

- 아래의 코드를 보면 Child클래스의 parentMethod()에 선언된 예외의 개수가 조상인 Parent클래스의 parentMethod()에 선언된 예외의 개수보다 적으므로 바르게 오버라이딩 되었다.

```
Class Parent {
    void parentMethod() throws IOException, SQLException {
        ...
    }
}

Class Child extends Parent {
    void parentMethod() throws IOException {
        ...
    }
    ...
}
```

여기서 주의해야할 점은 단순히 선언된 예외의 개수의 문제가 아니라는 것이다.

```
Class Child extends Parent {
    void parentMethod() throws Exception {
        ...
    }
    ...
}
```

만일 위와 같이 오버라이딩을 하였다면, 분명히 조상클래스에 정의된 메서드보다 적은 개수의 예외를 선언한 것처럼 보이지만 Exception은 모든 예외의 최고 조상이므로 가장 많

은 개수의 예외를 던질 수 있도록 선언한 것이다.

그래서 예외의 개수는 적거나 같아야 한다는 조건을 만족시키지 못하는 잘못된 오버라이딩인 것이다.

아래의 코드로 이 문제를 직접 테스트할 수 있다.

```
class NumberException extends Exception {}  
class InvalidNumberException extends NumberException {}  
class NotANumberException extends NumberException {}  
  
class Parent {  
    int a;  
    int b;  
  
    Parent() {  
        this(0,0);  
    }  
  
    Parent(int a, int b) {  
        this.a = a;  
        this.b = b;  
    }  
  
    void add(int a, int b)  
        throws InvalidNumberException, NotANumberException {}  
}  
  
class Child extends Parent {  
    Child() {}  
    Child(int a, int b) {  
        super(a,b);  
    }  
  
    void add(int a, int b)  
        throws InvalidNumberException, NotANumberException {}  
}
```

[8-4] 다음과 같은 메서드가 있을 때, 예외를 잘못 처리한 것은? (모두 고르시오)

```
void method() throws InvalidNumberException, NotANumberException {}

class NumberException extends RuntimeException {}
class InvalidNumberException extends NumberException {}
class NotANumberException extends NumberException {}
```

- a. try {method();} catch(Exception e) {}
- b. try {method();} catch(NumberException e) {} catch(Exception e) {}
- c. try {method();} catch(Exception e) {} catch(NumberException e) {}
- d. try {method();} catch(InvalidNumberException e) {
 } catch(NotANumberException e) {}
- e. try {method();} catch(NumberException e) {}
- f. try {method();} catch(RuntimeException e) {}

[정답] c

[해설] try블럭 내에서 예외가 발생하면, catch블럭 중에서 예외를 처리할 수 있는 것을 차례대로 찾아내려간다. 발생한 예외의 종류와 일치하는 catch블럭이 있으면 그 블럭의 문장들을 수행하고 try-catch문을 빠져나간다. 일치하는 catch블럭이 없으면 예외는 처리되지 않는다.

발생한 예외의 종류와 일치하는 catch블럭을 찾을 때, instanceof로 검사를 하기 때문에 모든 예외의 최고조상인 Exception이 선언된 catch블럭은 모든 예외를 다 처리할 수 있다. 한 가지 주의할 점은 Exception을 처리하는 catch블럭은 모든 catch블럭 중 제일 마지막에 있어야 한다는 것이다.

```
try {
    method();
} catch(Exception e) { // 컴파일 에러 발생!!!
} catch(NumberException e) {

}
```

위의 코드에서는 Exception을 선언한 catch블럭이 마지막 catch블럭이 아니기 때문에 컴파일 에러가 발생한다.

[8-5] 아래의 코드가 수행되었을 때의 실행결과를 적으시오.

【연습문제】/ch8/Exercise8_5.java

```
class Exercise8_5 {
    static void method(boolean b) {
        try {
            System.out.println(1);
            if(b) throw new ArithmeticException();
            System.out.println(2); // 예외가 발생하면 실행되지 않는 문장
        } catch(RuntimeException r) {
            System.out.println(3);
            return; // 메서드를 빠져나간다. (finally블럭을 수행한 후에)
        } catch(Exception e) {
            System.out.println(4);
            return;
        } finally {
            System.out.println(5); // 예외발생여부에 관계없이 항상 실행되는 문장
        }

        System.out.println(6);
    }

    public static void main(String[] args) {
        method(true);
        method(false);
    } // main
}
```

[정답]

【실행결과】

```
1
3
5
1
2
5
6
```

[해설] 예외가 발생하면 1,3,5가 출력되고 예외가 발생하지 않으면, 1,2,5,6이 출력된다. `ArithmaticException`은 `RuntimeException`의 자손이므로 `RuntimeException`이 정의된 `catch`블럭에서 처리된다. 이 `catch`블럭에 `return`문이 있으므로 메서드를 종료하고 빠져나가게 되는데, 이 때도 `finally`블럭이 수행된다.

[8-6] 아래의 코드가 수행되었을 때의 실행결과를 적으시오.

[연습문제]/ch8/Exercise8_6.java

```
class Exercise8_6 {
    public static void main(String[] args) {
        try {
            method1();
        } catch(Exception e) {
            System.out.println(5);
        }
    }

    static void method1() {
        try {
            method2();
            System.out.println(1);
        } catch(ArithmaticException e) {
            System.out.println(2);
        } finally {
            System.out.println(3);
        }

        System.out.println(4);
    } // method1()

    static void method2() {
        throw new NullPointerException();
    }
}
```

[정답]

[실행결과]

```
3
5
```

[해설] main에서드가 method1()을 호출하고, method1()은 method2()를 호출한다.

method2()에서 NullPointerException이 발생했는데, 이 예외를 처리해줄 try-catch블럭이 없으므로 method2()는 종료되고, 이를 호출한 method1()으로 되돌아갔는데 여기에는 try-catch블럭이 있긴 하지만 NullPointerException을 처리해줄 catch블럭이 없으므로 method1()도 종료되고, 이를 호출한 main에서드로 돌아간다. 이 때 finally블럭이 수행되어 '3'이 출력된다.

main에서드에서는 모든 예외를 처리할 수 있는 Exception이 선언된 catch블럭이 있으므로 예외가 처리되고 '5'가 출력된다.

[8-7] 아래의 코드가 수행되었을 때의 실행결과를 적으시오.

【연습문제】/ch8/Exercise8_7.java

```
class Exercise8_7 {
    static void method(boolean b) {
        try {
            System.out.println(1);
            if(b) System.exit(0);
            System.out.println(2);
        } catch(RuntimeException r) {
            System.out.println(3);
            return;
        } catch(Exception e) {
            System.out.println(4);
            return;
        } finally {
            System.out.println(5);
        }

        System.out.println(6);
    }

    public static void main(String[] args) {
        method(true);
        method(false);
    } // main
}
```

[정답]

【실행결과】

1

[해설] 변수 b의 값이 true이므로 System.exit(0);이 수행되어 프로그램이 즉시 종료된다. 이럴 때는 finally블럭이 수행되지 않는다.

[8-8] 다음은 1~100사이의 숫자를 맞추는 게임을 실행하던 도중에 숫자가 아닌 영문자를 넣어서 발생한 예외이다. 예외처리를 해서 숫자가 아닌 값을 입력했을 때는 다시 입력을 받도록 보완하라.

```
1과 100사이의 값을 입력하세요 :50
더 작은 수를 입력하세요 .
1과 100사이의 값을 입력하세요 :asdf
Exception in thread "main" java.util.InputMismatchException
    at java.util.Scanner.throwFor(Scanner.java:819)
    at java.util.Scanner.next(Scanner.java:1431)
    at java.util.Scanner.nextInt(Scanner.java:2040)
    at java.util.Scanner.nextInt(Scanner.java:2000)
    at Exercise8_8.main(Exercise8_8.java:16)
```

【연습문제】/ch8/Exercise8_8.java

```
import java.util.*;

class Exercise8_8
{
    public static void main(String[] args)
    {
        // 1~100사이의 임의의 값을 얻어서 answer에 저장한다.
        int answer = (int)(Math.random() * 100) + 1;
        int input = 0; // 사용자입력을 저장할 공간
        int count = 0; // 시도횟수를 세기 위한 변수

        do {
            count++;
            System.out.print("1과 100사이의 값을 입력하세요 :");

            // input = new Scanner(System.in).nextInt();
            try {
                input = new Scanner(System.in).nextInt();
            } catch(Exception e) {
                System.out.println("유효하지 않은 값입니다. "
                    +"다시 값을 입력해주세요.");
                continue;
            }

            if(answer > input) {
                System.out.println("더 큰 수를 입력하세요.");
            } else if(answer < input) {
                System.out.println("더 작은 수를 입력하세요.");
            } else {
                System.out.println("맞췄습니다.");
                System.out.println("시도횟수는 "+count+"번입니다.");
                break; // do-while문을 벗어난다
            }
        } while(true); // 무한반복문
    } // end of main
} // end of class HighLow
```

【실행결과】

```
1과 100사이의 값을 입력하세요 :50
더 작은 수를 입력하세요.
1과 100사이의 값을 입력하세요 :asdf
유효하지 않은 값입니다. 다시 값을 입력해주세요.
1과 100사이의 값을 입력하세요 :25
더 큰 수를 입력하세요.
1과 100사이의 값을 입력하세요 :38
더 큰 수를 입력하세요.
1과 100사이의 값을 입력하세요 :44
맞췄습니다.
시도횟수는 5번입니다.
```

[해설] 사용자로부터 값을 입력받는 경우에는 유효성검사를 철저하게 해야 한다. 사용자가 어떤 값을 입력할지 모르기 때문이다.

여기서는 간단하게 화면으로부터 값을 입력받는 부분에 try-catch구문으로 예외처리를 해주기만 하면 된다. 값을 입력받을 때 예외가 발생하면, 값을 다시 입력하라는 메세지를 보여주고 다시 입력 받으면 된다.

```
input = new Scanner(System.in).nextInt();  
  
try {  
    input = new Scanner(System.in).nextInt();  
} catch(Exception e) {  
    System.out.println("유효하지 않은 값입니다. 다시 값을 입력해주세요.");  
    continue;  
}
```

[8-9] 다음과 같은 조건의 예외클래스를 작성하고 테스트하시오.

[참고] 생성자는 실행결과를 보고 알맞게 작성해야한다.

- * 클래스명 : UnsupportedFuctionException
- * 조상클래스명 : RuntimeException
- * 멤버변수 :
 - 이 름 : ERR_CODE
 - 저장값 : 에러코드
 - 타입 : int
 - 기본값 : 100
 - 제어자 : final private
- * 메서드 :
 1. 메서드명 : getErrorCode
 - 기 능 : 에러코드(ERR_CODE)를 반환한다.
 - 반환타입 : int
 - 매개변수 : 없음
 - 제어자 : public
 2. 메서드명 : getMessage
 - 기 능 : 메세지의 내용을 반환한다.(Exception클래스의 getMessage()를 오버라이딩)
 - 반환타입 : String
 - 매개변수 : 없음
 - 제어자 : public

【연습문제】/ch8/Exercise8_9.java

```
class UnsupportedFunctionException extends RuntimeException {
    private final int ERR_CODE;

    UnsupportedFunctionException(String msg, int errCode) { // 생성자
        super(msg);
        ERR_CODE = errCode;
    }

    UnsupportedFunctionException(String msg) { // 생성자
        this(msg, 100); // ERR_CODE를 100(기본값)으로 초기화한다.
    }

    public int getErrorCode() { // 에러 코드를 얻을 수 있는 메서드도 추가했다.
        return ERR_CODE; // 이 메서드는 주로 getMessage()와 함께 사용될 것이다.
    }

    public String getMessage() { // Exception의 getMessage()를 오버라이딩한다.
        return "["+getErrorCode()+"]" + super.getMessage();
    }
}

class Exercise8_9 {
    public static void main(String[] args) throws Exception
    {
        throw new UnsupportedFunctionException("지원하지 않는 기능입니다.", 100);
    }
}
```

【실행결과】

```
Exception in thread "main" UnsupportedFuctionException: [100] 지원하지 않는 기능
입니다.
at Exercise8_9.main(Exercise8_9.java:5)
```

[해설] 여러 메시지를 저장하는 인스턴스 변수 msg는 상속받은 것이므로 조상의 생성자를 호출해서 초기화되도록 해야 한다. ERR_CODE는 한 번 값이 지정되면 바뀌는 값이 아니라서 final을 붙여서 상수로 했다. 그리고 생성자를 통해 초기화하였다.

```
UnsupportedFunctionException(String msg, int errCode) { // 생성자
    super(msg); // 조상의 생성자 RuntimeException(String msg)를 호출
    ERR_CODE = errCode;
}
```

getMessage() 역시 조상으로부터 상속받은 것이며, ERR_CODE도 같이 출력되도록 하기 위해 오버라이딩했다. 조상의 메서드를 오버라이딩할 때는, 가능하다면 조상의 메서드를 재활용하는 것이 좋다.

```
public String getMessage() { // Exception의 getMeesage()를 오버라이딩한다.
    return "["+getErrCode()+"]" + super.getMessage();
}
```

[8-10] 아래의 코드가 수행되었을 때의 실행결과를 적으시오.

[연습문제]/ch8/Exercise8_10.java

```

class Exercise8_10 {
    public static void main(String[] args) {
        try {
            method1(); // 예외 발생!!!
            System.out.println(6); // 예외가 발생해서 실행되지 않는다.
        } catch(Exception e) {
            System.out.println(7);
        }
    }

    static void method1() throws Exception {
        try {
            method2();
            System.out.println(1);
        } catch(NullPointerException e) {
            System.out.println(2);
            throw e; // 예외를 다시 발생시킨다. 예외 되던지기(re-throwing)
        } catch(Exception e) {
            System.out.println(3);
        } finally {
            System.out.println(4);
        }

        System.out.println(5);
    } // method1()

    static void method2() {
        throw new NullPointerException(); // NullPointerException을 발생시킨다.
    }
}

```

[정답]

[실행결과]

```

2
4
7

```

[해설] method2()에서 발생한 예외를 method1()의 try-catch문에서 처리했다가 다시 발생시킨다.

```

} catch(NullPointerException e) {
    System.out.println(2);
    throw e; // 예외를 다시 발생시킨다. 예외 되던지기(re-throwing)
} catch(Exception e) {
}

```

예외가 발생한 catch블럭 내에 이 예외(NullPointerException)를 처리할 try-catch블럭이 없기 때문에 method1()이 종료되면서 main에서드에 예외가 전달된다. 이 때 예외가 처리되진 않았지만, finally블럭의 문장이 수행되어 4가 출력된다.

main메서드의 try-catch블럭은 method1()으로부터 전달된 예외를 처리할 catch블럭이 있으므로 해당 catch블럭이 수행되어 7을 출력하고 try-catch블럭을 벗어난다. 그리고 더 이상 수행할 코드가 없으므로 프로그램이 종료된다.

```
try {
    method1(); // NullPointerException 발생!!!
    System.out.println(6); // 예외가 발생해서 실행되지 않는다.
} catch(Exception e) { // 모든 종류의 예외를 처리할 수 있다.
    System.out.println(7);
}
```