Chapter 8

예외처리

Exception Handling

[연습문제]

[8-1] 예외처리의 정의와 목적에 대해서 설명하시오.

예외처리란?
프로그래머가 예상하는 일외에 뜻하지 않은 일들이 생길 수 있는데, 이렇게 예상하지 못한 일들을 '예외'라고 하고 예외를 대비하고 준비하는 것을 '예외처리'라고 한다.
예외처리를 통해 프로그램의 비정상적인 종료를 막고 발생한 예외에 대한 처리를 하여 정상적인 프로그램 실행을 계속 진행할 수 있도록 하는 것이 목적이다.

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

```
java.lang.ArithmeticException : / 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.java파일의 12번째 줄이다.
c. 발생한 예외는 ArithmeticException이며, 0으로 나누어서 예외가 발생했다.
✔. method2메서드가 method1메서드를 호출하였고 그 위치는 ExceptionEx18.java파일의 8번째 줄이다.

예외의 종류는 ArithmeticException이고 0으로 나눠서 발생했다.
예외가 발생한 곳은 method2이고 ExceptionEx18.java의 12번째 줄이다.
호출 순서는 main -> method1 -> method2 순서로 호출되었다.

[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 {}
✔. void add(int a, int b) throws Exception {}
✔. void add(int a, int b) throws NumberException {}

오버라이딩을 할 때, 부모 클래스의 메소드보다 많은 수의 예외를 선언할 수 없다.
주의할 점으로 단순히 선언된 예외의 개수가 문제가 아니다.
Exception은 모든 예외의 최고 상위이므로 가장 많은 개수의 예외를 던질 수 있다.
이런 문제점을 실수하지 않도록 한다.

[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) {}

Exception이 모든 예외처리를 했기 때문에 NumberException는 예외처리를 할 수 없다.
모든 예외의 최고 상위인 Exception은 catch블럭 중 제일 마지막에 있어야 한다.

[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);
        } 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
}
```

[실행 결과]

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

예외가 발생하면 1, 3, 5 / 예외가 발생하지 않으면 1, 2, 5, 6이 출력된다.
ArithmeticException은 RuntimeException의 자식이므로 RuntimeException이 정의된 catch블럭에서 처리된다.
이 catch블럭에 return문이 있어서 메소드를 종료하고 빠져나갈 때, finally가 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(ArithmeticException e) {
            System.out.println(2);
        } finally {
            System.out.println(3);
        }

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

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

[실행 결과]
3
5

main > mehotd1() > method2() 호출
method2()에서 NullPointerException이 발생하는데, 이 예외를 처리해줄 try-catch블럭이 없다.
따라서 method2()는 종료되고, 이를 호출한 method1()로 되돌아간다.
method1()에는 try-catch블럭이 있지만, NullPointerException을 처리해줄 catch블럭이 없으므로
method1()도 종료되고, 이를 호출한 main메소드로 돌아간다.
종료되면서 finally가 실행되고, 3이 출력된다.
main메소드에는 모든 예외처리를 할 수 있는 Exception이 선언된 catch블럭이 있으므로 예외처리 후 5를 출력한다.

[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
}
```

[실행 결과]
1

변수 b의 값이 true이므로 System.exit(0);이 수행되어 프로그램이 즉시 종료된다.
이 경우 finally블럭은 수행되지 않는다.

System.exit(i)는 사용하지 않는 것이 바람직하다.

[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();
                                                  nextInt는 정수를 받을 때 사용, 다른 값이 들어오면 에러발생
                  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.
```

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

    continue;
}
```

```
1과 100사이의 값을 입력하세요 :38
더 큰 수를 입력하세요.
1과 100사이의 값을 입력하세요 :44
맞췄습니다.
시도횟수는 5번입니다.
```

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

[실행결과]

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

[8-10] 아래의 코드가 수행되었을 때의 실행결과를 적으시오.

[연습문제]/ch8/Exercise8_10.java

```
class Exercise8_10 {
   public static void main(String[] args) {
       try {
           method1();
           System.out.println(6);
       } 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;
       } catch(Exception e) {
           System.out.println(3);
       } finally {
           System.out.println(4);
       }

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

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