[8-1] 예외처리의 정의와 목적에 대해서 설명하시오.

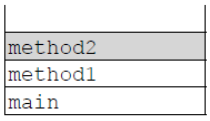
[정답]
정의 - 프로그램 실행 시 발생할 수 있는 예외의 발생에 대비한 코드를 작성하는 것
목적 - 프로그램의 비정상 종료를 막고, 정상적인 실행상태를 유지하는 것

[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
```

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

[해설] 예외의 종류는 ArithmeticException이고 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. main메서드 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.

[해설] try블럭 내에서 예외가 발생하면, catch블럭 중에서 예외를 처리할 수 있는 것을을 차례대로 찾아 내려간다. 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
```

[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();
  }
}
```

[정답]

[실행결과]

```
3
5
```

[해설] main메서드가 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 main메서드로 돌아간다. 이 때 finally블럭이 수행되어 '3'이 출력된다.

main메서드에서는 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.

```java
import java.util.*;
class Ex
{
    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 :");
            try {
                input = new Scanner(System.in).nextInt();
            } catch (InputMismatchException ime) {
                System.out.println("유효하지 않은 값입니다. 다시 값을 입력해 주세요.");
                count--;
                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 Ex
```

```java
import java.util.*;
class UnSupportedFunctionException extends RuntimeException {
    final private int ERR_CODE;

    public int getERR_CODE() {
        return ERR_CODE;
    }

    UnSupportedFunctionException(String message, int ERR_CODE) {
        super(message);  // 조상의 생성자 Exception(String msg) 호출
                         // Exception클래스는 생성 시에 String값을 받아서 메시지로 저장할 수
        this.ERR_CODE = ERR_CODE;
    }

    public String getMessage() { // Exception의 getMessage()를 오버라이딩.
        return "[" + getERR_CODE() + "]" + super.getMessage();
        // 조상의 메서드를 오버라이딩 할 때는, 가능하다면 조상의 메서드를 재활용하는 것이 좋다.
    }
}

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

[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 main메서드에 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);
    }
```