12강

JAVA_PROGRAMMING

Exception

❖ Exception – 예외

- Error는 프로그램 자체적인 문제(치명적) – 컴파일 에러 또는 실행 중 에러
- Exception는 코드 수정으로 해결이 가능한 동작 중의 문제
- 실행 중 예외상황을 Java에서는 예외클래스들로 정의
- Exception 최상위 클래스
- Exception을 상속하여 실행 중 일어날 수 있는 에러를 클래스로 정의

```
BufferedReader in = new BufferedReader(
        new InputStreamReader(System.in));
Integer.parseInt(in.readLine());
```

```
test
Exception in thread "main" java.lang.NumberFormatException: For input string: "test"
```

```
Scanner in = new Scanner(System.in);
in.nextInt();
```

```
test
Exception in thread "main" java.util.InputMismatchException
```

예외 발생 예제

❖ 다음 예제를 실행하면 어떤 예외가 발생되는가?

```
public static void main(String[] args){
    Scanner in = new Scanner(System.in);
    System.out.print("수 입력 : ");
    int n1 = in.nextInt();
    System.out.print("수 입력 : ");
    int n2 = in.nextInt();
    int ret = n1 / n2;
    System.out.println(n1 + "÷" + n2 + "=" + ret);
}
```

ArithmeticException

❖ 정수를 0으로 나눌 때 발생되는 예외

- 앞의 예제를 실행하여 다음과 같이 입력한다.

```
수 입력 : 10
수 입력 : 0
Exception in thread "main" java.lang.ArithmeticException: / by zero
```

- / by zero 메시지를 출력하고 프로그램이 종료된다.

InputMismatchException

❖ Scanner를 이용하여 입력 받을 때 자료형이 맞지 않을 경우 발생되는 예외

- 입력을 받을 때 자료형이 맞지 않으면 예외를 발생시키고 즉시 프로그램 종료

```
수 입력 : q
Exception in thread "main" java.util.InputMismatchException
```

```
수 입력 : 10
수 입력 : #
Exception in thread "main" java.util.InputMismatchException
```

예외 발생

- 예외가 발생되면 해당 예외로 정의되어 있는 Exception객체 생성
- 예외 객체가 생성되면 JVM은 프로그램을 종료시킨다.

- 예외사용
  - 개발자가 놓칠 수 있는 부분에 대해 예외 객체를 자동 생성하도록 함
  - 클래스로 정의된 예외가 발생되므로 개발자가 조절 가능
  - 개발자는 해당 예외가 발생되면 어떻게 할 것인지를 처리하면 됨

예외 처리

- 예외 처리에 사용되는 구문
- try{ … }
  - 예외가 발생되는 범위를 블록으로 지정
- catch{ … }
  - try블록 뒤에 사용
  - try블록에서 발생된 예외 객체를 잡아서 처리하는 블록
  - 예외객체가 생성되면 매치되는 catch()를 호출한다고 이해하면 됨
- finally{ … }
  - 필요 시 사용하는 블록
  - 예외발생 여부와 관계없이 무조건적으로 실행할 코드를 적는 블록
- throw
  - 예외 객체를 catch로 전달하는 구문
- throws
  - 해당 메서드에서 발생되는 예외클래스를 선언(예외처리가 아닌 예외 전가)
  - throws가 선언된 메서드를 사용하는 클래스에서 예외처리 필요(try~catch)

예외 처리

❖ 예외 처리 구문 사용 예제

```
public static void main(String[] args){
    Scanner in = new Scanner(System.in);
    int n1 = 0;
    System.out.print("수 입력 : ");
    try{
        n1 = in.nextInt();
    }catch(InputMismatchException ex){
        System.out.println("예외발생 - 1!");
    }
    System.out.print("수 입력 : ");
    int n2 = in.nextInt();
        System.out.println("예외발생 - 2!");
    int ret = n1 / n2;
    System.out.println(n1 + "÷" + n2 + "=" + ret);
}
```

Quiz

❖ 발생되는 예외를 모두 try블록에 묶고 catch에서 예외를 출력하도록 적용

```
수 입력 : a
숫자 입력 에러!
0÷0=0

수 입력 : 10
수 입력 : a
숫자 입력 에러!
10÷0=0

수 입력 : 10
수 입력 : 0
0으로 나눌 수 없음!
10÷0=0
```

finally

- finally구문은 예외 발생 여부와 상관없이 무조건 적으로 실행하는 블록이다.
  - 보통 입·출력 stream을 닫을 때 사용

```
public static void main(String[] args){
    Scanner in = new Scanner(System.in);
    try{
        System.out.print("수 입력 : ");
        int data = in.nextInt();
    }catch(InputMismatchException e){
        System.err.println("입력 형식 예외!");
    }finally{
        System.out.println("finally블록은");
        System.out.println("무조건 실행!");
    }
}
```

Exception 정의하기

- ❖ 정의된 Exception이 아닌 사용자가 정의하는 예외를 클래스로 정의할 수 있다.
- ❖ 이 때 Exception으로 관리가 가능하도록 Exception클래스를 상속한다.

```
class MyException extends Exception{
    public MyException(String message){
        super(message);
    }
    public String toString(){
        return "MyException";
    }
}
```

Exception 발생 시키기 throw

- ❖ 앞에서 정의한 Exception을 코드에서 발생시킨다.
- ❖ throw는 예외객체를 catch로 던진다.

```
public static void main(String[] args){
    Scanner in = new Scanner(System.in);
    int val = 0;
    try{
        System.out.print("수 입력 : ");
        val = in.nextInt();
        if(val > 100){
            MyException exception = new MyException("100이상을 입력함");
            throw exception;
        }
        System.out.println("입력한 값 : " + val);
    }catch(MyException ex){
        String msg = ex.getMessage();
        System.err.println(ex.toString() + " : " + msg);
    }
}
```

Exception 메서드

❖ Exception은 Throwable클래스를 상속했으므로 Throwable의 함수를 사용한다.

- 일반적으로 사용하는 메서드는 printStackTrace(), getMessage() 이다.

Throwable클래스의 메서드 목록

메소드의 개요	
Throwable	**fillInStackTrace** () 실행 스택 트레이스를 묻습니다.
Throwable	**getCause** () 원인이 존재하지 않는가 불명한 경우에, 이 Throwable 또는 null 의 원인을 돌려줍니다.
String	**getLocalizedMessage** () 이 throw 가능 객체의, 로컬라이즈 된 기술을 작성합니다.
String	**getMessage** () 이 Throwable 객체의 상세 메세지 캐릭터 라인을 돌려줍니다.
StackTraceElement []	**getStackTrace** () printStackTrace() 에 의해 제공되는 스택 트레이스 정보에 프로그램으로 액세스 할 수 있도록(듯이) 합니다.
Throwable	**initCause** (Throwable cause) 지정된 값에 대한 이 Throwable 의 「원인」을 초기화합니다.
void	**printStackTrace** () 이 throw 가능 객체 및 그 백 트레이스를 표준 에러 스트림에 출력합니다.
void	**printStackTrace** (PrintStream s) 이 throw 가능 객체와 그 백 트레이스가 지정된 인쇄 스트림에 출력합니다.
void	**printStackTrace** (PrintWriter s) 이 throw 가능 객체와 그 백 트레이스가 지정된 프린트 라이터에 출력합니다.
void	**setStackTrace** (StackTraceElement [] stackTrace) getStackTrace() 에 의해 돌려주어지고 printStackTrace() 에 의해 출력되는 스택 트레이스 요소와 관련 메소드를 설정합니다.
String	**toString** () 이 throw 가능 객체의 짧은 기술을 돌려줍니다.

throws 예외전가

❖ 입출력 사용 시 메서드에 선언했던 throws IOException같은 형태는 예외전가
- 해당 함수를 사용하는 곳에서 직접 처리를 해야만 한다.

```
public static void main(String[] args){
    System.out.println("문자 입력 : ");
    int val = System.in.read();
    System.out.println("입력한 문자 : " + (char)val);
}
```

```
public static void main(String[] args) throws IOException{
    System.out.println("문자 입력 : ");
    int val = System.in.read();
    System.out.println("입력한 문자 : " + (char)val);
}
```

read() 메서드 throws 예

❖ System 클래스의 in static객체는 InputStream으로 만들어져 있다.

```
static InputStream                  in
                                    The "standard" input stream.
```

❖ InputStream 클래스의 read()메서드는 다음과 같이 throws 선언이 되어있다.

```
public abstract int read()
                  throws IOException
```

- 해당 메서드에서 예외가 발생할 수 있다.
- 예외를 현재 메서드에서 처리하지 않고 사용하는 쪽에서 처리하도록 선언하는 것

❖ 즉, InputStream의 read()메서드를 사용한다면 사용하는 메서드에서 try~catch 구문을 이용한 예외처리가 필요한 것이다.