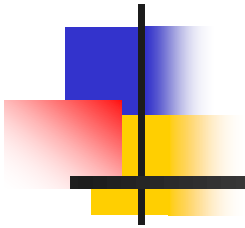
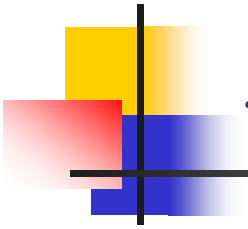


java 12강-예외처리

양 명 숙
[now4ever7@gmail.com]



목차

- 예외처리
  - try~catch
    - 다중 예외처리
  - throw
  - try~catch~finally
  - throws
  - 사용자 정의 예외 만들기

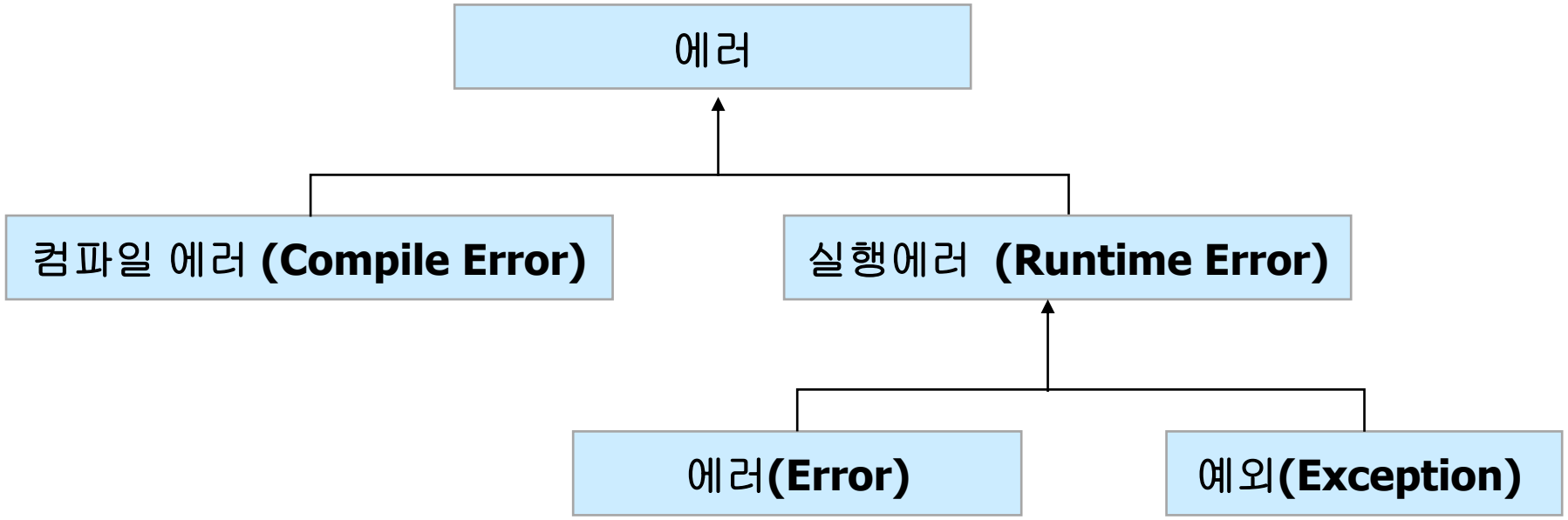
예외(Exception)의 의미

```
두 개의  정수 입력: 10
0
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at DivideByZero.main(DivideByZero.java:14)
```

- 예외
  - 예외
    - 프로그램의 **실행도중에 발생하는** 예상치 못한 **오류**(문제 상황, 예외적인 상황)
    - 컴파일 시 발생하는 문법적인 에러는 예외의 범주에 포함되지 않음
  - 예외 상황의 예
    - 나눗셈을 두 개의 정수를 입력 받는데, 나누는 수로 0 이 입력됨
    - 나이를 입력하라고 했는데, 0보다 작은 값이 입력됨
    - 주민번호 13자리만 입력하라고 했더니, 중간에 - 를 포함하여 14자리를 입력함
- 처리되지 않은 예외는 프로그램의 실행을 중단시키는 원인
- 신뢰도 및 안정성 측면에서 매우 중요

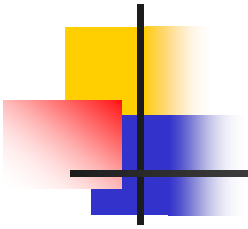
에러

- 에러
  - [1] 컴파일 에러 (Compile Error)
  - [2] 실행에러 (Runtime Error)
    - 에러(Error)
    - 예외(Exception)



예외처리(Exception)

- 에러
  - [1] 컴파일 에러 (Compile Error)
    - 소스코드를 컴파일하는 과정에서 생기는 에러, 실행파일을 생성시키지 못함
    - 오타이거나 문법적 오류인 경우
  - [2] 실행에러 (Runtime Error)
    - 프로그램을 실행시켜 사용하는 도중에 발생되는 에러
- 자바에서는 실행 시 발생할 수 있는 프로그램 오류를 '에러(Error)'와 '예외(Exception)' 두 가지로 구분
  - 에러(Error) - 메모리 부족, 스택오버플로우와 같이 일단 발생하면 복구할 수 없는 심각한 오류
  - 예외(Exception) - 발생하더라도 수습될 수 있는 비교적 덜 심각한 오류
    - => 프로그램의 비정상적인 종료를 막을 수 있다



예외처리

- 예외처리(Exception Handling)란
  - 프로그램 실행 시 발생할 수 있는 예기치 못한 **예외의 발생에 대비한 코드를 작성**하는 것
- 예외 처리의 목적
  - 예외의 발생으로 인한 실행중인 프로그램의 갑작스런 비정상 종료를 막고, 정상적인 실행상태를 유지할 수 있도록 하는 것

예외처리 구문

- 형식

```
try
{
        예외가 발생할만한 코드들이 배치
        (프로그램의 정상적인 실행을 시도하다가 실패하게 되면 예외를 던짐)
}
catch(Exception e)
{
        try블록에서 발생한 예외를 잡아 실패에 대한 처리를 하는 코드
}
```

• 실행에러가 발생할 가능성이 있는 코드부분에 보호막을 씌워서 에러가 발생했을 경우에 처리해야 할 일들을 정해 줄 수 있음

- 에러가 발생될 가능성이 있는 부분에 try로 블록을 지어놓고, 그 try블록 안에서 에러가 발생한다면 catch블록 안의 코드가 실행이 됨

• 예외를 처리한다는 의미 - 프로그램 실행도중 발생한 에러를 처리, 에러에 의해서 비정상적으로 프로그램이 종료되는 것을 막아줌

try ~ catch 블록

- 예외처리
  - 예외가 발생할만한 구문을 try절에 놓음
  - try절에서 예외가 발생하면 catch절이 실행됨
  - try절에서 예외가 발생하지 않는다면 catch문은 실행되지 않고 지나가게 됨

```
try
{
        System.out.println("숫자를 입력하세요");
        Scanner sc=new Scanner(System.in);
        int i=sc.nextInt();
}
catch (Exception e)
{
        System.out.println("예외발생: "+e);
}
System.out.println("다음 문장…");
```

Program logic

Error handling

```
숫자를 입력하세요
abc
예외발생 : java.util.InputMismatchException
```

```
숫자를 입력하세요
27000000000000
예외발생 : java.util.InputMismatchException: For input string: "27000000000000"
```

예제-try catch 이용

```
두 개의  정수 입력: 5 0
나눗셈 불가능
/ by zero
프로그램을 종료합니다.
```

```
import java.util.Scanner;
class DivideByZero
{
	public static void main(String[] args)
	{
		System.out.print("두 개의  정수 입력: ");
		Scanner sc=new Scanner(System.in);
		int num1=sc.nextInt();
		int num2=sc.nextInt();

		try
		{
			System.out.println("나눗셈 결과의 몫: "+(num1/num2));
			System.out.println("나눗셈 결과의 나머지: "+(num1%num2));
		}
		catch(ArithmeticException e)
		{
			System.out.println("나눗셈 불가능");
			System.out.println(e.getMessage());
		}

		System.out.println("프로그램을 종료합니다.");
	}
}
```

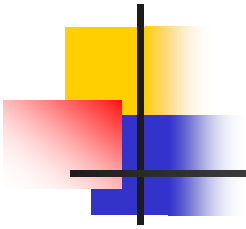
try문은 에러가 발생할 가능성이 있는 곳에서 사용

1. 예외발생

2. 참조값 전달하면서 **catch** 영역 실행

3. catch 영역 실행 후, **try~catch** 다음 문장을 실행

- 예외를 처리했기 때문에 메시지만 출력되고 프로그램은 계속 실행되며 다운되지는 않음



예제-try catch 이용

- 실행의 흐름이 catch 영역으로 이동하는 과정
  - 자바 가상머신이 0 으로 나누는 예외상황이 발생했음을 인식
  - 이 상황을 위해 정의된 ArithmeticException 클래스의 인스턴스를 생성
  - 가상머신에 의해 생성된 인스턴스의 참조값을 catch영역에 선언된 매개변수에 전달
- e.getMessage()
  - 예외상황이 발생한 이유를 담은 문자열을 반환하는 메소드

예외처리 – if문 이용

```
피제수 입력: 5
제수 입력: 2
나눗셈 결과 : 2
피제수 입력: 7
제수 입력: 0
제수는 0이 될 수 없습니다.
피제수 입력: 7
제수 입력: 3
나눗셈 결과 : 2
```

```
import java.util.Scanner;
class ExceptionHandleUseIf{
	public static void main(String[] args)	{
		Scanner sc=new Scanner(System.in);
		int result=0;

		for(int i=0; i<2; i++){
			System.out.print("피제수 입력: ");
			int num1=sc.nextInt();

			System.out.print("제수 입력: ");
			int num2=sc.nextInt();

			if(num2==0){
				System.out.println("제수는 0 이 될 수 없습니다.");
				i--;
				continue;
			}

			result=num1/num2;
			System.out.println("나눗셈 결과 : "+result);
		}//for
	}
}
```

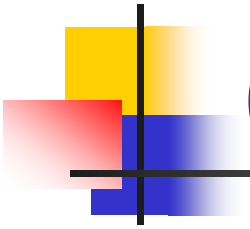
예외처리 – try~catch 이용

```
import java.util.Scanner;
class ExceptionHandleUseTryCatch{
	public static void main(String[] args)	{
		Scanner sc=new Scanner(System.in);

		int result=0;
		for(int i=0; i<2; i++){
			try{
				System.out.print("피제수 입력: ");
				int num1=sc.nextInt();

				System.out.print("제수 입력: ");
				int num2=sc.nextInt();

				result=num1/num2;
				System.out.println("나눗셈 결과 : "+result);
			}catch(ArithmeticException e){
				System.out.println("제수(나누는 수)는 0 이 될 수 없습니다.");
				i--;
				continue;
			}
		}
	}
}
```

예외처리

```
try{
    //try 영역
}
catch(AAA e){
    //catch 영역
}
```

- if문을 이용해서 예외상황 처리
  - if문은 예외처리 이외의 용도로도 사용되기 때문에, 프로그램 코드 상에서 **예외처리 부분을 구분하기가 쉽지 않음**
  - 프로그램의 주 흐름을 구성하는 코드와 예외상황을 처리하는 코드의 구분이 어려워짐
- try~catch 기반의 예외처리 방식
  - try는 예외상황이 발생할 만한 영역을 감싸는 용도로 사용
  - catch는 발생한 예외의 처리를 위한 코드를 묶어두기 위한 용도로 사용됨
  - catch 영역에서 예외상황이 처리되기 때문에, **소스코드상에서 예외상황의 처리를 위한 코드를 쉽게 구분**할 수 있음
  - try 영역에서 발생한 AAA에 해당하는 예외상황은 이어서 등장하는 catch 영역에서 처리한다.
    - catch 영역에서 모든 예외상황을 처리하는 것은 아님
    - (AAA e) 이 부분에 명시되어 있는 유형의 예외상황만 처리가 가능



예외 타입

• 자주 발생하는 예외들을 모아서 예외 클래스로 제공

예외타입	설명
ArrayIndexOutOfBoundsException	배열의 접근에 잘못된 인덱스값을 사용했을 때 던져짐
NumberFormatException	int형 숫자로 변경될 수 없을때 예외발생
ArithmeticException	0으로 나눗셈을 하는 등의 수학적 연산이 불가능한 상황
ClassCastException	허용할 수 없는 형변환 연산을 진행하는 예외상황
NullPointerException	참조변수가 null 로 초기화된 상황에서 메소드를 호출하는 예외상황
NegativeArraySizeException	배열선언 과정에서 배열의 크기를 음수로 지정하는 예외상황

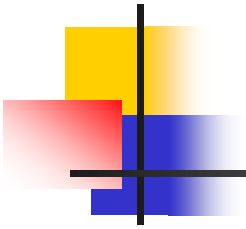
예제

```
예외발생  : 3
java.lang.ArrayIndexOutOfBoundsException: 3

예외발생  : [I cannot be cast to java.lang.String
java.lang.ClassCastException: [I cannot be cast to java.lang.String

예외발생  : null
java.lang.NegativeArraySizeException

예외발생  : null
java.lang.NullPointerException
```

```
class RuntimeExceptionCase{
    public static void main(String[] args)
        try
        {
            int[] arr=new int[3];
            arr[3]=20;
        }
        catch(ArrayIndexOutOfBoundsException e)
        {
            System.out.println("예외발생  : "+ e.getMessage()+"₩n"+e+"₩n");
            //e.printStackTrace();
        }

        try
        {
            Object obj=new int[10];
            String str=(String)obj; //배열과 String 클래스는 형변환이 불가능한,아무상관이
    없는 클래스 => 예외상황

            //Circle c = (Circle)new Shape();
        }
        catch(ClassCastException e)
        {
            System.out.println("예외발생  : "+ e.getMessage()+"₩n"+e+"₩n");
        }
```

예제

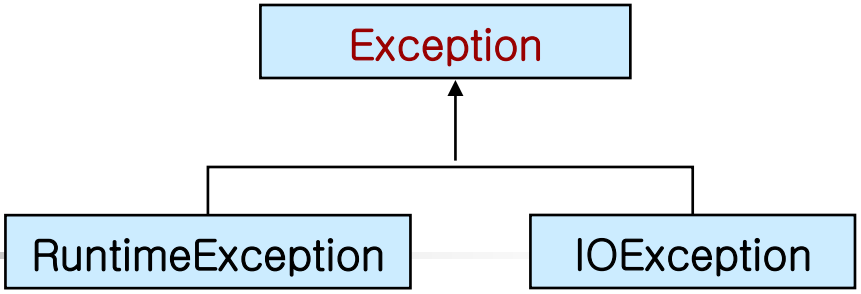
```
        try
        {
                int[] arr=new int[-10];
        }
        catch(NegativeArraySizeException e)
        {
                System.out.println("예외발생 : "+ e.getMessage()+"₩n"+e+"₩n");
        }

        try
        {
                String str=null;
                int len=str.length();
        }
        catch(NullPointerException e)
        {
                System.out.println("예외발생 : "+ e.getMessage()+"₩n"+e+"₩n");
        }
    }
}
```



다중 예외 처리

- 다중 catch문
  - try문에서 여러 개의 예외가 발생할 수 있을 때는 발생 가능한 모든 예외에 대해 여러 개의 catch문을 나열하고 발생한 예외에 따라 각각 다르게 처리함
  - 여러 가지 종류의 에러에 대해서 어떤 에러가 발생했는지 구분하고 그 종류에 따라 처리하는 코드를 다르게 정할 수 있음
  - 발생한 예외의 종류와 일치하는 단 한 개의 catch 블럭만 수행됨
  - 예외 타입
    - 예외종류에 따라 다르게 처리할 수 있음

다중 예외 처리



- 다중 catch문
  - 예외상황이 발생되어서 예외 클래스의 인스턴스가 생성되고 나면, 위에서 아래로 적절한 catch 영역을 찾게 됨
  - Exception 계층구조에서 상위의 예외처리 객체가 위에 올라와서는 안됨
    - 하위(자식)에서 상위(부모)순으로 와야 함

```
catch (Exception e){}
catch (IOException e){}
```

에러 - IOException 의 catch 영역은 실행되지 않습니다

  - 특수하고 상세한 예외가 앞쪽에 와야 하며, 일반적인 예외가 뒤쪽에서 처리되어야 함
  - 동일한 예외처리 객체를 두 번 이상 반복할 수 없음

예제

```
숫자를 입력하세요
abcd
숫자만 입력: null
```

```
숫자를 입력하세요
0
0으로 나누지 마: / by zero
```

```
import java.util.*;
class MultiTryTest2{
	public static void main(String[] args) {
		try {
			System.out.println("숫자를 입력하세요");
			Scanner sc=new Scanner(System.in);
			int num=sc.nextInt();
			int r = 100/num;
		}
		catch (ArithmeticException e){
			System.out.println("0으로 나누지 마: "+e.getMessage());
		}
		catch (InputMismatchException e){
			System.out.println("숫자만 입력: "+e.getMessage());
		}
		catch(Exception e) {
			System.out.println(e.getMessage());
		}
	}
}
```

```
원하는 배열의 갯수를 입력하세요: abc
숫자만 입력: null
```

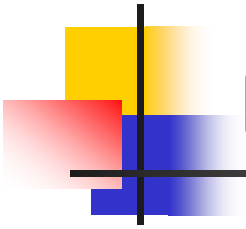
```
원하는 배열의 갯수를 입력하세요: 7
인덱스 오류: 8
```

예제2

```
import java.util.*;
class MultiTryTest3
{
	public static void main(String[] args){
		try{
			System.out.print("원하는 배열의 갯수를 입력하세요: ");
			Scanner sc=new Scanner(System.in);
			int size=sc.nextInt();
			int[] arr = createArray(size);
			for (int n : arr)
			{
				System.out.println(n + "₩t");
			}
		}
		catch (ArrayIndexOutOfBoundsException e)
		{
			System.out.println("인덱스 오류: "+e.getMessage());
			e.printStackTrace();
		}
```

예제2

```
            catch (InputMismatchException e)
            {
                        System.out.println("숫자만 입력: "+e.getMessage());
                        e.printStackTrace();
            }
            catch(Exception e){
                        System.out.println("에러 메세지: " + e.getMessage());
                        e.printStackTrace();
            }
    } //main

    public static int[] createArray(int size)
    {
            int[] num = new int[size];
            num[size+1] = 50;  //에러 발생 상황

            return num;
   }
}
```



Exception 클래스

- Exception – 모든 예외의 기반 클래스
  - getMessage() 메소드
    - String형 반환
    - 예외가 발생한 원인에 대한 설명을 담고 있음
    - 발생한 예외 클래스의 인스턴스에 저장된 메세지를 얻을 수 있음
  - printStackTrace() 메소드
    - 예외발생 당시의 호출스택에 있었던 메서드의 정보와 예외 메시지를 화면에 출력함
    - 예외가 발생한 위치에 대한 정보
    - 예외가 발생해서 전달되는 과정 출력

```
두 실수를 입력하세요
12.5
47.6
덧셈 연산 결과 : 12.5+47.6 = 60.1
```

```
두 실수를 입력하세요
10
ok
null
java.util.InputMismatchException
        at java.util.Scanner.throwFor(Scanner.java:840)
        at java.util.Scanner.next(Scanner.java:1461)
        at java.util.Scanner.nextDouble(Scanner.java:2387)
        at Ex7_1.main(Ex7_1.java:18)
```

실습

- 1. 두 실수의 합을 구하는 메서드 만들기
- main()에서 사용자로부터 두 실수를 입력 받아서, 메서드 호출하여 두 실수의 합을 구한 후 결과를 출력
  - 문자열을 입력하면 예외처리 되도록
  - main()에서 try~catch 로 예외처리하기

- 2. 두 수의 나머지를 구하는 메서드 만들기
- main()에서 사용자로부터 두 수를 입력 받아서, 메서드 호출하여 나머지를 구한 후 결과를 출력
  - 나누는 수가 0 이면 예외처리 되도록
  - main()에서 try~catch 로 예외처리하기

```
두 수를 입력하세요
15
0
/ by zero
```

throw 구문

- 키워드 throw 를 사용해서 개발자가 **고의로 예외를 직접 발생**시킬 수 있음

```
throw new Exception("에러 메시지") ;
```

```
if (minute < 1 || minute >= 60)
{
   throw new InvalidTimeException(minute + " : 분은 잘못된 시간임!");
}
```

1. 먼저, 연산자 new 를 이용해서 발생시키려는 예외 클래스의 객체를 만든 다음
   Exception ex = new Exception("고의로 발생시켰음!");
2. 키워드 throw 를 이용해서 예외를 발생시킨다
   throw ex;

throw – 예외 발생시키기

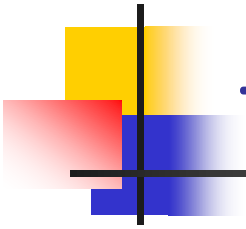
- throw문 (던지기)
  - 프로그램 실행에서 비정상적인 상황을 알리기 위해 예외를 강제로 발생시킴
  - throw문을 써서 해당 예외처리를 던지면 catch문의 코드가 실행됨
  - 개발자가 고의로 예외를 던질 수도 있음

- throw 문은 언제 사용?

  - 자바 자체에서 출력하는 예외처리 메시지가 아닌 **다른 메시지를 보여주고자** 할 때

  - 자바 가상머신에 의해 인식될 수 있는 예외 상황이 아니지만, 프로그램의 성격에 따라 **개발자가 정의한 예외상황**인 경우 (예: 나이 입력시 0 보다 작은 값이 입력되었다)

- throw new Exception ("에러!");



throw

- 프로그램의 성격에 따라 개발자가 정의한 예외상황인 경우
  - 예외상황이 발생되었음을 알리고
  - 이에 필요한 예외 클래스의 정의도 해야 함
    - Exception 클래스를 상속하여 사용자 정의 예외 클래스 정의

예제

```
에러 메시지 : 고의로 발생시켰음!
java.lang.Exception: 고의로 발생시켰음!
        at ThrowTest.main(ThrowTest.java:6)

프로그램이 정상 종료되었음.
```

```
class ThrowTest {
    public static void main(String args[])  {
        try {
            Exception e = new Exception("고의로 발생시켰음!");
            throw e;     // 예외를 발생시킴
            //throw new Exception("고의로 발생시켰음!");  //한 줄로 줄여 쓸 수 있음

        } catch (Exception e) {
            System.out.println("에러 메시지 : " + e.getMessage());
            e.printStackTrace();
        }

        System.out.println("₩n프로그램이 정상 종료되었음.");
    }
}
```

public **Exception**(String message)

- Exception 인스턴스를 생성할 때, 생성자에 String을 넣어 주면, 이 String이 Exception 인스턴스에 메시지로 저장됨
- 이 메시지는 **getMessage()**를 이용해서 얻을 수 있음

예제2

```
나이를 입력하세요
-5
예외 발생 : 나이는 양수를 입력해야 합니다!
java.lang.Exception: 나이는 양수를 입력해야 합니다!
        at ThrowTest2.main(ThrowTest2.java:13)

프로그램 정상 종료~~
```

```
나이를 입력하세요
20
나이 : 20

프로그램 정상 종료~~
```

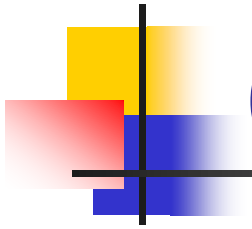
```
import java.util.*;
class ThrowTest2{
    public static void main(String[] args) {
        try{
            System.out.println("나이를 입력하세요");
            Scanner sc = new Scanner(System.in);
            int age = sc.nextInt();
            if (age<0){
                throw new Exception("나이는 양수를 입력해야 합니다!");
            }

            System.out.println("나이 : " + age);
        }catch (Exception e){
            System.out.println("예외 발생 : " + e.getMessage());
            e.printStackTrace();
        }

        System.out.println("₩n프로그램 정상 종료~~");
    }
}
```

예제3

```
숫자를 두 개 입력하세요
10
0
제수가 0 이 되면 안됩니다!!!
```

```
숫자를 두 개 입력하세요
47
6
47/6=7
```

```
import java.util.*;
class ThrowTest2{
    public static void main(String[] args){
        try
        {
            System.out.println("숫자를 두 개 입력하세요");
            Scanner sc = new Scanner(System.in);
            int x =sc.nextInt();
            int y =sc.nextInt();

            if (y==0)
            {
                throw  new Exception("제수가 0 이 되면 안됩니다!!!");
            }
            int r = divide(x, y);
            System.out.printf("%d/%d=%d", x, y, r);
        }
        catch(Exception e)
        {
            System.out.println(e.getMessage());
        }
    }
```



예제3

```
    public static int divide(int a, int b)
    {
            int c= a / b;

            return c;
    }
}
```

실습

```
주민번호를 입력하세요
990102
주민번호를 잘못 입력했습니다. 13자리를 입력하세요
java.lang.Exception: 주민번호를 잘못 입력했습니다. 13자리를 입력하세요
        at Ex7_5.main(Ex7_5.java:15)

프로그램이 정상적으로 종료되었습니다!!
```

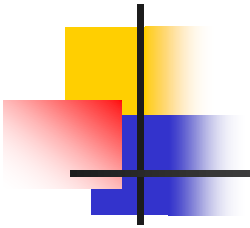
- 1.사용자로부터 주민번호를 입력 받아서, 13자리가 아니면 throw 문을 이용하여 예외처리 되도록 한다.

```
주민번호를 입력하세요
9901021117777

주민번호 : 9901021117777

프로그램이 정상적으로 종료되었습니다!!
```

- 2.두 수의 나머지를 구하는 메서드 만들기
- main()에서 사용자로부터 두 수를 입력 받아서, 메서드 호출하여 나머지를 구한 후 결과를 출력
  - 나누는 수가 0 이면 예외처리 되도록 throw 문 이용
  - main() 에서 try~catch 로 예외처리하기

```
숫자를 두 개 입력하세요
14
0
제수가 0 이 되면 안됩니다!!!
```

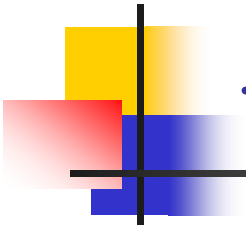
```
숫자를 두 개 입력하세요
47
5
47%5=2
```



finally 절

- 예외 발생여부와 상관없이 반드시 실행되어야 하는 구문을 입력하는 곳
- 예외가 발생해도 호출되며, 그렇지 않아도 호출됨
- try 영역으로 일단 들어가면 무조건 실행되는 영역

```
try {
    예외가 발생할 가능성이 있는 문장들을 넣는다.
}
catch(){
    예외 처리를 위한 문장을 넣는다
}
finally {
    예외의 발생여부에 관계없이 항상 수행되어야 하는 문장들을 넣는다.
}
```



finally

- finally문
  - 예외가 발생하든 안 하든 무조건 finally블록 안에 있는 코드는 실행시키라는 것
  - 예외의 발생여부에 상관없이 최종적으로 해야 할 일을 지정하는 곳
  - 예) 사용했던 자원을 해제 : DB Close, 파일 닫기

• try-catch문 다음에 최종적으로 반드시 실행되어야 하는 명령을 입력

```
class FinallyTest {
    public static void main(String args[]) {
        try {
            startInstall();          // 프로그램 설치에 필요한 준비를 한다.
            copyFiles();             // 파일들을 복사한다.
            deleteTempFiles();       // 프로그램 설치에 사용된 임시파일들을 삭제한다.
        } catch (Exception e) {
            e.printStackTrace();
            deleteTempFiles();   // 프로그램 설치에 사용된 임시파일들을 삭제한다.
        }
    } // main

    public static void startInstall() {
        System.out.println("프로그램 설치에 필요한 준비를 합니다!");
    }
    public static void copyFiles() {
        System.out.println("파일들을 복사합니다");
    }
    public static void deleteTempFiles() {
        System.out.println("임시파일들을 삭제합니다");
    }

}//class
```

```
class FinallyTest2 {
    public static void main(String args[]) {
        try {
            startInstall();                 // 프로그램 설치에 필요한 준비를 한다.
            copyFiles();                    // 파일들을 복사한다.
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            deleteTempFiles();    // 프로그램 설치에 사용된 임시파일들을 삭제한다.
        }
    }// main

    public static void startInstall() {
        System.out.println("프로그램 설치에 필요한 준비를 합니다!");
    }
    public static void copyFiles() {
        System.out.println("파일들을 복사합니다");
    }
    public static void deleteTempFiles() {
        System.out.println("임시파일들을 삭제합니다");
    }
}
```

finally

```
0이나 2를 선택해주세요.
0
에러 발생
이것만큼은 꼭 실행되어야 돼!!.
```

```
0이나 2를 선택해주세요.
2
50
이것만큼은 꼭 실행되어야 돼!!.
```

```
import java.util.*;
class FinallyTest2{
    public static void main(String[] args){
        System.out.println("0 이나 2를 선택해주세요.");
        try {
            Scanner sc=new Scanner(System.in);
            int n=sc.nextInt();
            int m = 100/n;
            System.out.println(m);  //에러시 실행 안됨
        }catch(Exception e)  {
            System.out.println("에러 발생");
        }finally {
            System.out.println("이것만큼은 꼭 실행되어야 해!!.");
        }
    }
}
```

예제

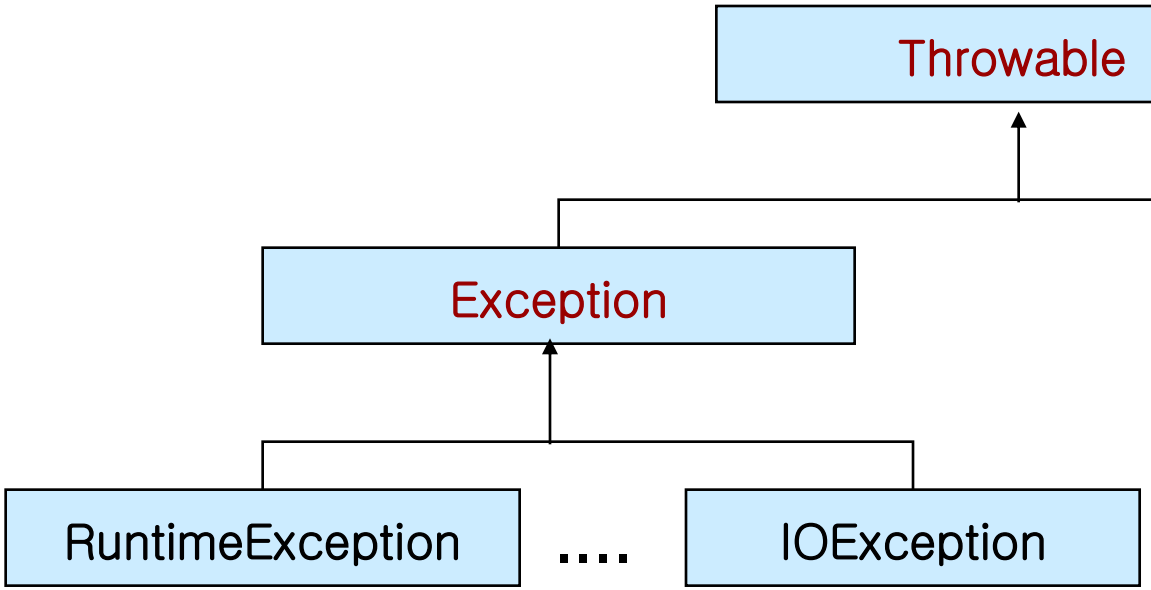
```
나눗셈 결과는 2
finally 영역 실행
연산 성공

/ by zero
finally 영역 실행
연산 실패
```

```
class FinallyTest{
    public static void main(String[] args){
        boolean divOK=divider(4, 2);
        if(divOK)
            System.out.println("연산 성공₩n");
        else
            System.out.println("연산 실패₩n");

        divOK=divider(4, 0);
        if(divOK)
            System.out.println("연산 성공₩n");
        else
            System.out.println("연산 실패₩n");
    }
    public static boolean divider(int num1, int num2)  {
        try{
            int result=num1/num2;
            System.out.println("나눗셈 결과는 "+result);
            return true;
        }
        catch(ArithmeticException e){
            System.out.println(e.getMessage());
            return false;
        }
        finally{
            System.out.println("finally 영역 실행");
        }
    }
}
```

중간에 return을 하더라도 finally 영역은 실행되고 나서 메서드를 빠져나가게 됨

예외 클래스의 계층구조

- 예외 클래스의 계층구조



- 심각한 오류의 상황을 표현하기 위해 정의된 클래스
- 프로그램의 실행을 멈춰야 할 정도의 심각한 오류상황 표현시 사용
- 예) VirtualMachineError, OutOfMemoryError, StackOverflowError

예외 클래스의 계층구조

- 예외 클래스
  - 1. RuntimeException 클래스와 그 자식 클래스들
    - 개발자의 실수에 의해서 발생할 수 있는 예외들
      - 예) 배열의 범위를 벗어나는 경우(IndexOutOfBoundsException)
      - 값이 null 인 참조변수의 멤버를 호출하려 하는 경우(NullPointerException)
      - 클래스간의 형변환을 잘못한 경우(ClassCastException)
      - 정수를 0으로 나누려 하는 경우에 발생하는 예외들(ArithmeticException)
    - try~catch 문을 사용하기 보다는 주의 깊게 작성하여 예외가 발생하지 않도록 하는 것이 좋다
  - 2. Exception 클래스와 그 자식 클래스들
    - 사용자의 실수와 같은 외적인 요인에 의해 발생하는 예외
      - 예) 존재하지 않는 파일의 이름을 입력하는 경우(FileNotFoundException)
      - 실수로 클래스의 이름을 잘못 적은 경우(ClassNotFoundException)
      - 입력한 데이터 형식이 잘못된 경우(DataFormatException)
    - 반드시 예외처리를 해주어야 함, 그렇지 않으면 컴파일 시에 에러가 발생함

처리하지 않아도 되는 RuntimException의 하위 클래스

- Exception 의 하위 클래스 중 RuntimeException 클래스는 그 성격이 Error 클래스와 비슷
  - (이는 Exception 을 상속하는 다른 예외 클래스들과의 차이점임)
  - RuntimeException 을 상속하는 예외 클래스도 Error를 상속하는 예외 클래스 처럼 **try~catch, throws 절을 이용한 예외처리를 필요로 하지 않음**
- Error 의 하위 클래스들과 구분되는 특징
  - RuntimeException을 상속하는 예외 클래스는 Error 를 상속하는 예외 클래스처럼 치명적인 상황을 표현하지 않음
  - 따라서 예외 발생 이후에도 프로그램의 실행을 이어가기 위해서 try~catch 문으로 해당 예외를 처리하기도 함
- RuntimeException의 하위 클래스
  - ArithmeticException
  - IndexOutOfBoundsException
  - ClassCastException
  - NegativeArraySizeException
  - NullPointerException

메서드에 예외 선언하기

- 예외를 처리하는 방법
  - [1] try~ catch 문을 사용하는 것
  - [2] 예외를 메서드에 선언하는 방법(예외 전달, 예외 떠넘기기)
- 메서드에 예외를 선언
  - 메서드의 선언부에 키워드 throws 를 사용해서 메서드 내에서 발생할 수 있는 예외를 적어주기만 하면 됨
  - 예외가 여러 개일 경우에는 쉼표(,)로 구분

```
void method() throws Exception1, Exception2, ...ExceptionN{
        //메서드의 내용
}
```

  - 메서드를 작성할 때 메서드내에서 발생할 가능성이 있는 예외를 메서드의 선언부에 명시하여 이 메서드를 **사용하는 쪽에서는 이에 대한 처리를 하도록 강요**함

메서드에 예외 선언하기

- 메서드에 예외를 선언할 때 일반적으로 RuntimeException 클래스들은 적지 않는다.

- 예외를 메서드의 throws 에 명시하는 것은 예외를 처리하는 것이 아니라, 자신(예외가 발생할 가능성이 있는 메서드)을 호출한 메서드에게 **예외를 전달하여 예외처리를 떠맡기는 것**임
- 예외를 전달받은 메서드가 또다시 자신을 호출한 메서드에게 전달할 수 있음
- 이런 식으로 계속 호출스택에 있는 메서드들을 따라 전달되다가 제일 마지막에 있는 main 메서드에서도 예외가 처리되지 않으면, main 메서드 마저 종료되어 프로그램 전체가 종료됨

예제 1

```
Exception in thread "main" java.lang.Exception
        at ThrowsTest1.method2(ThrowsTest1.java:11)
        at ThrowsTest1.method1(ThrowsTest1.java:7)
        at ThrowsTest1.main(ThrowsTest1.java:3)
```

```
class ThrowsTest1 {
    public static void main(String[] args) throws Exception {
        method1();              // 같은 클래스내의 static멤버이므로 객체생성없이 직접 호출가능.
    }       // main메서드의 끝

    static void method1() throws Exception {
        method2();
    }       // method1의 끝

    static void method2() throws Exception {
        throw new Exception();
    }       // method2의 끝
}
```

- 예외가 발생한 곳은 제일 윗줄에 있는 method2()
- main 메서드가 method1()을, 그리고 method1()은 method2()를 호출했다
- method2()에서 'throw new Exception(); 문장에 의해 예외가 강제적으로 발생했으나, try-catch문으로 예외처리를 해주지 않았으므로, method2()는 종료되면서 예외를 자신을 호출한 method1()에게 넘겨줌
- method1()에서도 역시 예외처리를 해주지 않았으므로 종료되면서 main메서드에게 예외를 넘겨줌
- main메서드에서도 예외처리를 해주지 않았으므로 main메서드가 종료되어 프로그램이 예외로 인해 비정상적으로 종료됨

예제2

```
method1메서드에서 예외가 처리되었습니다.
java.lang.Exception
        at ThrowsTest2.method1(ThrowsTest2.java:8)
        at ThrowsTest2.main(ThrowsTest2.java:3)
```

```
class ThrowsTest2 {
    public static void main(String[] args) {
        method1();
    }       // main메서드의 끝

    static void method1() {
        try {
            throw new Exception();
        } catch (Exception e) {
            System.out.println("method1메서드에서 예외가 처리되었습니다.");
            e.printStackTrace();
        }
    }       // method1의 끝
}
```

- 예외가 발생한 메서드에서 예외처리를 하지 않고 자신을 호출한 메서드에게 예외를 넘겨줄 수는 있지만, 이것으로 예외가 처리된 것은 아니고 예외를 단순히 전달만 하는 것임
- 결국 어디서든 간에 반드시 try-catch문을 사용해서 예외처리를 해주어야 함

예제3

```
main메서드에서 예외가 처리되었습니다.
java.lang.Exception
        at ThrowsTest3.method1(ThrowsTest3.java:12)
        at ThrowsTest3.main(ThrowsTest3.java:4)
```

```
class ThrowsTest3 {
    public static void main(String[] args) {
        try  {
                method1();
        } catch (Exception e) {
                System.out.println("main메서드에서 예외가 처리되었습니다.");
                e.printStackTrace();
        }
    }       // main메서드의 끝

    static void method1() throws Exception {
        throw new Exception();
    }       // method1()의 끝
} // class의 끝
```

System.in.read()

```
import java.io.IOException;
class SwitchTest{
	public static void main(String[] args) throws IOException{
		System.out.println("성별을 입력하세요. F/M");
		char gender = (char)System.in.read();  //사용자가 입력한 아스키코드값을 반환해 줌
                                                       //앞의 1바이트만 읽는다
		String str = "";
		switch(gender){
		case 'M':
				str = "남자이시군요";
				break;
		case 'F':
				str = "여자이시군요";
				break;
		default:
				str = "잘못 입력! ";
				break;
		}
		System.out.println(str);
	}
}
```

```
public abstract int read() throws IOException

System.in.read();
```

IOException

- IOException
  - input/output 관련 작업을 처리할 때 발생하는 예외상황
  - 예) 파일을 생성하고 어떤 데이터를 쓰거나(output) 읽을 때(input) 발생함
- RuntimeException은 프로그램 내부의 원인에 의해서 발생하는 예외지만, IOException은 주로 JVM 외부의 원인에 의해서 발생함
- IOException 클래스의 자식 클래스 – EOFException, FileNotFoundException,InterruptedIOException 등
  - EOFException – EOF 는 End Of File 의 첫 글자를 딴 것으로, 예상하지 않은 파일의 종료 혹은 스트림의 종료가 발생하여 더 이상 처리할 수 없을 때 발생하는 예외
  - FileNotFoundException – 지정된 경로의 파일을 찾을 수 없을 때 발생하는 예외
  - InterruptedIOException – 입출력(IO)관련 처리를 하는 도중 다른 영향을 받아서 인터럽트(interrupt)가 발생하는 예외
    - => 이들 예외는 미디어(Media)에 데이터를 쓰거나 읽을 때 발생하는 예외상황들임
    - 대상 매체는 파일(file)이나 네트워크(network) 소켓 통신이 될 수 있음

예외 전달

• clone (Object 클래스의 인스턴스 메소드)
protected Object clone() throws CloneNotSupportedException

- clone() 은 상황에 따라서 CloneNotSupportedException 예외를 전달하는 메소드

```
public void simpleMethod1(int n) throws CloneNotSupportedException
{
        MyClass my = new MyClass();
        my.clone();
}

public void simpleMethod(int n)
{
        MyClass my = new MyClass();
        try
        {
                my.clone();
        }
        catch (CloneNotSupportedException e)
        {
        }
}
```

사용자 정의 예외 만들기

- 기존의 정의된 예외 클래스 외에 필요에 따라 개발자가 새로운 예외 클래스를 정의하여 사용할 수 있음

```
class MyException extends Exception{
        MyException(String msg)
        {
                super(msg);  //부모인 Exception 클래스의 생성자를 호출함
        }
}
```

• Exception 클래스의 생성자

Exception(String message)
- Exception 클래스는 생성 시에 String 값을 받아서 메시지로 저장할 수 있음

사용자 정의 예외 만들기

```
class MyException extends Exception{
        //에러 코드 값을 저장하기 위한 필드 추가
        private final int errorCode = 100;

        MyException(String msg)
        {
                super(msg);  //부모인 Exception 클래스의 생성자를 호출함
        }

        public int getCode() //에러 코드를 얻을 수 있는 메서드 추가
        {
                return errorCode; //주로 getMessage()와 함께 사용
        }
}
```

MyException이 발생했을 때, catch 블럭에서 getMessage()와 getCode()를 사용해서 에러 코드와 메시지를 모두 얻을 수 있다.

예제

```
import java.io.*;
class MyException extends Exception {
	private final int errorCode = 100;

	public MyException(String msg) {
		super(msg);
	}

	public int  getCode() {
		return errorCode;
	}
}

public class MyExceptionTest {
	public static void main(String[] args) {
		try{
			throw new MyException("사용자 정의 예외가 발생하였습니다!!!");
		}
		catch (MyException e)		{
			System.out.println("에러 메시지 : " + e.getMessage());
			System.out.println("에러 코드 : " + e.getCode());

			e.printStackTrace();
		}

		System.out.println("₩n--------The End-------");
	}
}
```

예제2

```
나이를 입력하세요: 20
당신은 20세입니다.
```

```
나이를 입력하세요: -5
유효하지 않은 나이가 입력되었습니다.
```

```
import java.util.Scanner;

class AgeInputException extends Exception
{
    public AgeInputException()
    {
        super("유효하지 않은 나이가 입력되었습니다.");
    }
}

class ProgrammerDefineException{
    public static void main(String[] args)
    {
        System.out.print("나이를 입력하세요: ");

        try
        {
            int age=readAge();
            System.out.println("당신은 "+age+"세입니다.");
        }
        catch(AgeInputException e)
        {
            System.out.println(e.getMessage());
        }
    }
```

Exception 클래스의 생성자 호출을 통해서 전달된 문자열이 getMessage()의 호출을 통해서 반환됨

throws에 의해 이동된 예외처리 포인트!

예제2

AgeInputException 예외는 던져버린다

```
    public static int readAge() throws AgeInputException
    {
            Scanner sc=new Scanner(System.in);
            int age=sc.nextInt();
            if(age<0){
                    AgeInputException excpt=new AgeInputException();
                    throw excpt;
            }

            return age;
    }
}
```

예외상황의 발생지점
예외처리 포인트!

throw - 예외상황이 발생되었음을 자바 가상머신에게 알리는 키워드

예외처리 메커니즘 가동!

예제2

- throw 문이 존재하는 위치가 예외상황이 발생했음을 알리는 지점
- 이 부분을 try~catch 문으로 감싸서 예외상황을 처리할 수도 있고, throw 문이 존재하는 메서드를 호출한 영역에서 try~catch 처리할 수도 있음
- 메서드 호출영역에서 처리하는 경우 – 예외상황을 알리기 위해 생성된 AgeInputException 인스턴스의 참조 값은 main()메소드에 존재하는 catch 영역으로 전달되어 예외상황이 처리됨
  - [1] throw에 의해 생성된 예외상황이 메소드 내에서 처리되지 않으면, 메소드를 호출한 영역으로 예외의 처리가 넘어감을 (던져짐을) 반드시 명시해야 함
  - throws AgeInputException
    - 'readAge()메소드 내에서는 AgeInputException 에 대한 예외상황을 처리하지 않으니, 이 메소드를 호출하는 영역에서는 AgeInputException 에 대한 처리도 대비해야 한다' 는 선언
  - [2] throw 에 의해 생성된 예외상황은 반드시 try~catch 문에 의해 처리되거나 throws 에 의해서 넘겨져야 함
    - main()메소드 내에서도 AgeInputException 의 예외상황을 처리하지 않는다면, main()메소드 역시 throws 를 이용해서 이 예외상황의 처리를 넘겨야 함

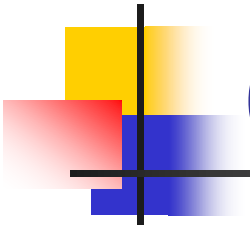
예제

```
나이를 입력하세요: -5
Exception in thread "main" AgeInputException: 유효하지 않은 나이가 입력되었습니다.
        at ThrowsFromMain.readAge(ThrowsFromMain.java:26)
        at ThrowsFromMain.main(ThrowsFromMain.java:16)
```

```
import java.util.Scanner;
class AgeInputException extends Exception{
    public AgeInputException()
    {
        super("유효하지 않은 나이가 입력되었습니다.");
    }
}
class ThrowsFromMain{
    public static void main(String[] args) throws AgeInputException
    {
        System.out.print("나이를 입력하세요: ");
        int age=readAge();
        System.out.println("당신은 "+age+"세입니다.");
    }
    public static int readAge() throws AgeInputException
    {
        Scanner sc=new Scanner(System.in);
        int age=sc.nextInt();
        if(age<0)
        {
            AgeInputException excpt=new AgeInputException();
            throw excpt;
        }
        return age;
    }
}
```

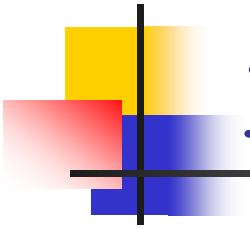
main 메소드내에서 AgeInputException의 예외상황이 발생하면, 이를 처리하지 않고 main메소드를 호출한 영역으로 넘겨버린다고 선언

readAge() 메소드내에서 AgeInputException의 예외상황이 발생하면, 이를 처리하지 않고 readAge 메소드를 호출한 영역으로 넘겨버린다고 선언



예제

- 예외상황은 main 메소드를 호출한 영역으로까지 넘어가게 됨
  - main 메소드는 가상머신이 호출하는 메소드
  - 예외상황의 처리는 가상머신에게까지 넘어가게 되고, 결과적으로 예외상황의 처리는 가상머신에 의해서 이뤄지게 됨
- 가상머신의 예외처리 방식
  - 1. getMessage() 메소드 호출
  - 2. 예외상황이 발생해서 전달되는 과정을 출력해줌
  - 3. 프로그램을 종료함



과제

- [과제1]
- 주민번호 14자리(예: 990101-2223333)가 입력되지 않은 경우 예외발생시키기
  - 사용자 정의 예외 클래스 만들기
  - 메서드에서 주민번호 입력 받기
    - 메서드에서 예외발생

과제- 전화번호 관리 프로그램 5단계

- 직접 예외의 상황을 정의하고, 해당 예외의 표현을 위한 예외 클래스를 정의해서 프로그램에 반영
- 예외 상황
  - [1] 초기 메뉴 선택에서 1, 2, 3, 4, 5 이외의 값을 입력하는 예외상황
  - [2] 데이터 입력의 과정(inputData())에서 1,2,3 이외의 값을 입력하는 예외상황
    - 두 가지 예외상황 모두 그 유형이 동일하니, MenuChoiceException 이라는 이름의 예외 클래스를 하나 정의해서 위의 두 상황에 모두 활용한다
    - 위의 두 가지 중 어느 예외상황이 발생하건, 프로그램의 흐름은 '초기 메뉴 선택'으로 이동하는 것을 원칙으로 한다

예외처리에서는 예외의 발생위치와 예외의 처리 위치를 결정하는 것이 가장 중요함

```
선택하세요...
1. 데이터 입력
2. 전체 데이터 조회
3. 데이터 검색
4. 데이터 삭제
5. 프로그램 종료
선택: 7
7에 해당하는 선택은 존재하지 않습니다.
메뉴 선택을 처음부터 다시 진행합니다.

선택하세요...
1. 데이터 입력
2. 전체 데이터 조회
3. 데이터 검색
4. 데이터 삭제
5. 프로그램 종료
선택:
```

```
선택하세요...
1. 데이터 입력
2. 전체 데이터 조회
3. 데이터 검색
4. 데이터 삭제
5. 프로그램 종료
선택: 1
데이터 입력을 시작합니다..
1. 일반, 2. 대학, 3. 회사
선택>> 5
5에 해당하는 선택은 존재하지 않습니다.
메뉴 선택을 처음부터 다시 진행합니다.

선택하세요...
1. 데이터 입력
2. 전체 데이터 조회
3. 데이터 검색
4. 데이터 삭제
5. 프로그램 종료
선택:
```