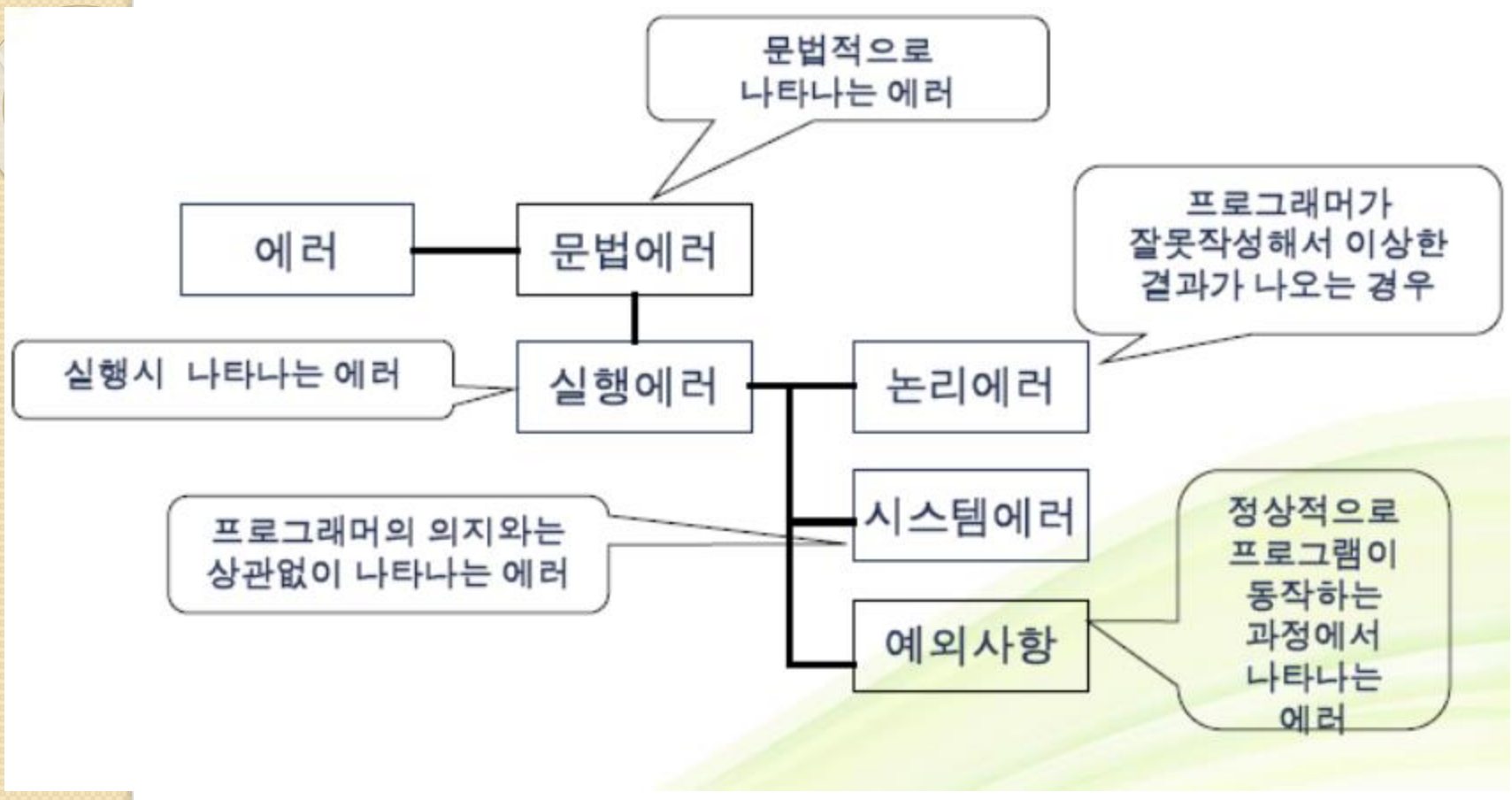
10. 예외처리

강사 강태광

] 에러 유형


문법적으로
나타나는 에러
에러
문법에러
프로그래머가
잘못작성해서 이상한
결과가 나오는 경우
실행시 나타나는 에러
실행에러
논리에러
시스템에러
프로그래머의 의지와는
상관없이 나타나는 에러
정상적으로
프로그램이
동작하는
과정에서
나타나는
에러
예외사항

2] 예외의 개요

1) 예외 : **프로그램 실행 중에 발생하는 예기치 않은 사건**

2) 예외가 발생하는 예시

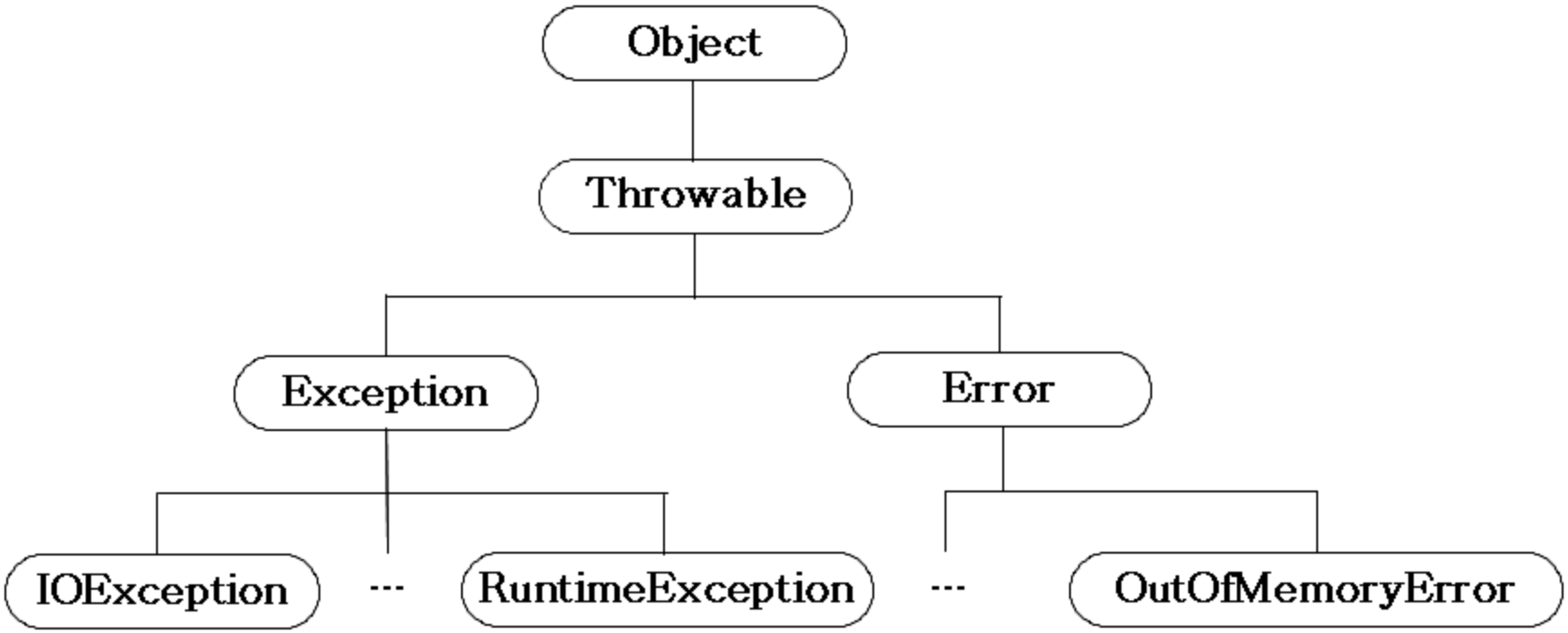
- 정수를 0으로 나누는 경우
- 배열의 첨자가 음수 또는 범위를 벗어나는 경우
- 부적절한 형변환이 일어나는 경우
- 입출력을 위한 파일이 없는 경우 등

3) 자바 언어는 프로그램에서 예외를 처리할 수 있는 기법을 제공

3] 예외 Class의 계층구조1

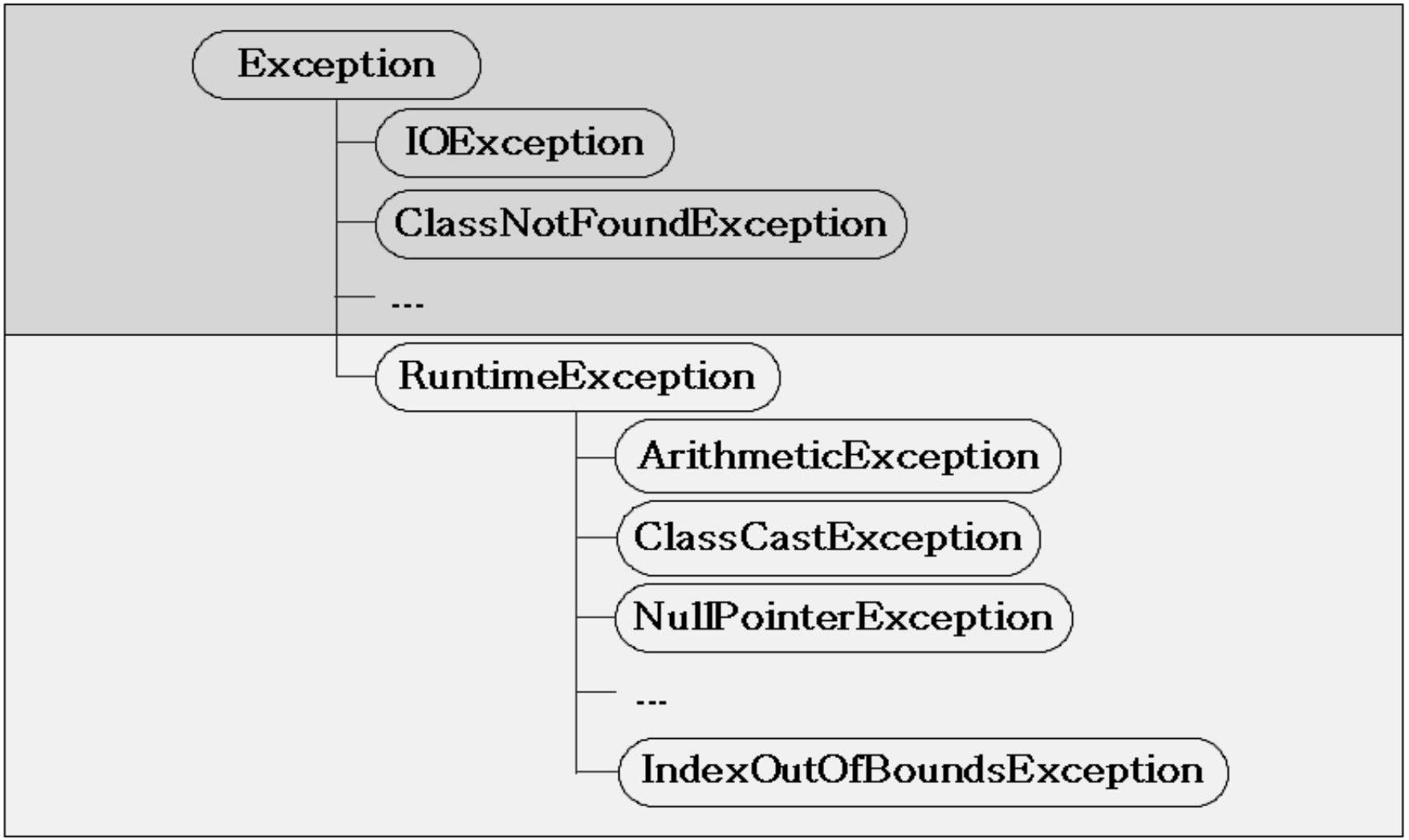
- 예외 클래스는 크게 두 그룹으로 나뉜다

RuntimeException클래스들 - 프로그래머의 실수로 발생하는 예외
Exception클래스들 - 사용자의 실수와 같은 외적인 요인에 의해 발생하는 예외



[그림8-1] 예외클래스 계층도

3] 예외 Class의 계층구조2



[그림8-2] Exception클래스와 RuntimeException클래스 중심의 상속계층도

4] 예외 관련 클래스 I

- 자바는 예외를 객체로 취급
- 예외 관련 클래스를 **java.lang 패키지에서 제공**
- 자바 프로그램에서는 **Error, RuntimeException 클래스의 하위** 클래스들을 제외한 모든 예외를 처리하여야 한다
- 일반적으로 **Error, RuntimeException 클래스(하위 클래스 포함)** 들과 연관된 예외는 프로그램에서 처리하지 않는다

-> 이유 **: 예외를 처리하여 얻는 이득보다 예외를 처리하기 위한** 노력이 너무 크기 때문

Exception 클래스의 하위 클래스

- NoSuchMethodException : 메소드가 존재하지 않을 때
- ClassNotFoundException : 클래스가 존재하지 않을 때
- CloneNotSupportedException : 객체의 복제가 지원되지 않는 상황에서의 복제 시도
- IllegalAccessException : 클래스에 대한 부정 접근
- InstantiationException : 추상클래스나 인터페이스로부터 객체 생성하려 할 때
- InterruptedException : 스레드가 인터럽트 되었을 때
- RuntimeException : 실행시간 예외가 발생할 때

4] 예외 관련 클래스2

RuntimeException 클래스의 하위 클래스

- ArithmeticException : 0으로 나누는 등의 산술적인 예외
- NegativeArraySizeException :배열의 크기를 지정할 때 음수의 사용
- NullPointerException : null 객체의 메소드나 멤버 변수에 접근할 때
- IndexOutOfBoundException : 배열이나 스트링의 범위를 벗어난 접근
  . 하위클래스로 ArrayIndexOutOfBoundsException 클래스와 StringIndexOutOfBoundsException 클래스를 제공
- SecurityException :보안을 이유로 메소드를 수행할 수 없을 때

```
class ExceptionEx2 {
	public static void main(String  args[]) {
		int number  = 100;
		int result  = 0;
		for(int i=0; i < 10; i++) {
			result  = number / (int)(Math.random()  * 10);
			// 7번째 라인
			System.out.println(result);
		}
	}
}
```

```
Exception in thread "main"
java.lang.ArithmeticException: / by zero
at
ch10.ExceptionEx2.main(ExceptionEx2.java:8)
```

5] 예외를 처리하는 방법 I

1. 예외를 처리하는 방법은 두 가지
   1) 예외가 발생된 메소드 내에서 처리하는 방법(**try, catch 절 사용)**
   2) 예외가 발생된 메소드를 호출한 메소드에게 예외의 처리를 넘겨주는 방법(**throws 절 사용)**

1. 예외(Exception)이 발생한 메소드 내에서 직접 처리 (try-catch-finally)

```
try{
    예외 발생 가능성이 있는 문장들;
}catch(예외 타입1 매개변수명){
    예외타입1의 예외가 발생할 경우 처리 문장들;
}catch(예외 타입 n 매개변수명){
    예외타입 n의 예외가 발생할 경우 처리 문장들;
}finally{
    항상 수행할 필요가 있는 문장들;
```

5] 예외를 처리하는 방법 I

1) try 블록은 예외가 발생할 가능성이 있는 범위를 지정하는 블록이다. try 블록은 최소한 하나의 catch 블록이 있어야 하며, catch 블록은 try 블록 다음에 위치한다.

2) catch 블록의 매개변수는 예외 객체가 발생했을 때 참조하는 변수명으로 반드시 java.lang.Throwable 클래스의 하위 클래스 타입으로 선언되어야 한다.

3) 지정된 타입의 예외 객체가 발생하면 try 블록의 나머지 문장들은 수행되지 않고, 자바 가상 머신은 발생한 예외 객체를 발생시키며 발생한 예외 객체 타입이 동일한 catch 블록을 수행한다.

4. finally 블록은 필수 블록은 아니다.
finally 블록이 사용되면 finally 블록의 내용은 예외 발생 유무나 예외 catch 유무와 상관 없이 무조건 수행. 따라서, 데이터베이스나 파일을 사용한 후 닫는 기능과 같이 항상 수행해야 할 필요가 있는 경우에 사용한다

5] 예외를 처리하는 방법 - try, catch 1

```
class ExceptionEx3 {
	public static void main(String args[]) {
		int number = 100;
		int result = 0;
		for(int i=0; i < 10; i++) {
			try {
				result = number / (int)(Math.random() * 10);
				System.out.println(result);
			} catch (ArithmeticException e) {
				// ArithmeticException이 발생하면 실행되는 코드
				System.out.println("0");
			} // try-catch의 끝
		} // for의 끝
	}
}
```

5] 예외를 처리하는 방법 - try, catch 2

```
class ExceptionEx5 {
    public static void main(String args[]) {
        // 0으로 나눠서 고의로 ArithmeticException을 발생시킨다.
        System.out.println(1);
        System.out.println(2);
        try {
            System.out.println(3);
            System.out.println(0/0);
            System.out.println(4);  // 실행되지 않는다.
        } catch (ArithmeticException ae) {
            System.out.println(5);
        } // try-catch의 끝
        System.out.println(6);
    } // main메서드의 끝
}
```

5] 예외를 처리하는 방법 - throws 1

1 예외(Exception)이 발생한 메소드를 호출 한 곳으로 예외 객체를 넘기는 방법 (throws)

```
public class ExceptionTest {

    static void callDriver()  throws ClassNotFoundException{
        Class.forName("oracle.jdbc.driver.OracleDriver");
        System.out.println("완료");
    }

    public static void main(String  args[]){

        try{
            callDriver();
        }catch(ClassNotFoundException  e){
            System.out.println("클래스를  찾을 수 없습니다.");
        }catch(Exception  e){
            System.out.println(e.getMessage());
        }finally{
            System.out.println("시스템  종료.");
        }

    }
}
```

5] 예외를 처리하는 방법 - throws 1

1 예외(Exception)이 발생한 메소드를 호출 한 곳으로 예외 객체를 넘기는 방법 (throws)

1) main 함수에서 callDriver() 함수를 실행시킨다.
2) callDriver() 함수에서는 "oracle.jdbc.driver.OracleDriver" 클래스를 가져온다.
3) 해당 클래스를 찾지 못하면 ClassNotFoundException이 발생하는데, callDriver()에서는
4) throws ClassNotFoundException 처리로 호출한 main 함수로 예외를 넘긴다.
5) main에서는 ClassNotFoundException을 받아 catch 문에서 잡아서
   "클래스를 찾을 수 없습니다." 메시지를 출력.
6) 마지막으로 finally가 실행되며 "시스템 종료" 를 출력

5] 예외를 처리하는 방법 - throws 1

2 **사용자 정의 예외 생성 (throw)**

1) 기존의 예외 클래스로 예외를 처리할 수 없다면 사용자만의 예외 클래스를 작성하여 발생시킬 수 있다.
사용자가 예외 클래스를 정의하려면 예외 클래스의 최상위 클래스인 java.lang.Exception 클래스를 상속받아 클래스를 정의한다.
- class 예외 클래스 이름 extends Exception

2) 자바 가상 머신은 프로그램 수행중에 예외가 발생하면 자동으로 해당하는 예외 객체를 발생시킨 다음 catch 블록을 수행한다.
하지만 예외는 사용자가 강제적으로 발생시킬 수도 있다. 자바는 예외를 발생시키기 위해 throw 예약어를 사용
- **throw new 예외 클래스 이름(매개변수);**
- throw를 이용한 예외 발생시에도 try-catch-finally 구문을 이용한 예외 처리를 하거나, throws를 이용하여 예외가 발생한 메소드를 호출한 다른 메소드로 넘기는 예외 처리 방법을 사용해야 한다.

5] 예외를 처리하는 방법 - throws 1

3 사용자 정의 예외 생성 (throw) 예시

```
class MyException extends Exception{
    public MyException(){
        super("내가 만든 예외");
    }
}

public class ExceptionTest {
   static void callException() throws MyException{
        throw new MyException();
    }

    public static void main(String args[]){
        try{
                callException();
               }catch(MyException e){
                        System.out.println(e.getMessage());
                }catch(Exception e){
                         System.out.println(e.getMessage());
                }finally{
                        System.out.println("시스템 종료.");
               }
    }
}
```

5] 예외를 처리하는 방법 - throws 1

3 사용자 정의 예외 생성 (throw) 예시 설명

1. MyException 이라는 Exception을 상속한 예외를 만들었다.
   그리고 MyException은 "내가 만든 예외" 라는 메시지를 갖는다.

2. ExceptionTest의 main 함수가 실행되면 callException() 함수를 호출한다.

3. callException 함수는 **MyException()**을 **new**로 생성한 후 callException() 함수를 호출한 main 함수로 던진다.

4. **MyException**을 받은 main 함수는 catch에서 해당 예외를 받아서, 예외의 메시지를 출력-> "내가 만든 예외"

5. 최종적으로 finally가 실행되어 "시스템 종료"가 출력된다

5] 예외를 처리하는 방법 - throws 1

```
import java.io.*;
class Exception1 {
	public static void main(String []args)throws Exception {
		BufferedReader in = new BufferedReader(new
		InputStreamReader(System.in));

		while(true) {
			System.out.print("첫 번째 값을 입력하세요 => ");
			int num1 = Integer.parseInt(in.readLine());
			System.out.print("두 번째 값을 입력하세요 => ");
			int num2 = Integer.parseInt(in.readLine());
			System.out.println(num1 + " / " + num2 +" = " + num1/num2);
		}
	}
}
```

5] 예외를 처리하는 방법 - throws 2

```
package ch10;

import java.io.*;
class Exception3 {
	public static void main(String []args)throws Exception {
		BufferedReader in = new BufferedReader(new InputStreamReader(System.in));

		while(true) {
			try {
				System.out.print("첫 번째 값을 입력하세요 => ");
				int num1 = Integer.parseInt(in.readLine());
				System.out.print("두 번째 값을 입력하세요 => ");
				int num2 = Integer.parseInt(in.readLine());
				System.out.println(num1 + " / " + num2 +" = " + num1/num2);
			}catch(Exception e) {
				System.out.println( "값을 잘못 입력했습니다.");
			}
		}
	}
}
```

6] 다중Try/catch문에 의한 예외처리

형식

```
try{
예외가 발생할 만한 코드
}catch(해당 Exception){
예외처리를 위한 루틴
} catch(해당 Exception){
예외처리를 위한 루틴
} catch(해당 Exception){
예외처리를 위한 루틴
}
```

```
import java.io.*;
class Exception4 {
   public static void main(String[]  args) throws Exception {
      BufferedReader  in =
          new  BufferedReader(new  InputStreamReader(System.in));
      while(true)  {
            try {
            System.out.print("첫  번째 값을 입력 하세요 => ");
             int num1  = Integer.parseInt(in.readLine());
            System.out.print("두  번째 값을 입력 하세요 => ");
            int num2  = Integer.parseInt(in.readLine());
            System.out.println(num1  + "/" + num2
            +"="+num1/num2);
            } catch(NumberFormatException  e) {
            System.out.println(  "숫자를 입력해야 합니다.");
            } catch(ArithmeticException  e) {
            System.out.println(  "0으로 나누는 계산은 처리할 수 없습
            니다.");
            }
      }
   }
}
```

6] 다중Try/catch문과 finally문

▶ finally문
☞ 예외의 발생 여부에 상관없이
반드시 실행이 되어야 할 때 사용한다.

```
try{
        예외가 발생할 만한 코드
}catch(해당 Exception){
        예외처리를 위한 루틴
}finally{
        실행될 문장
}
```

```
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
public class Ex03 {
    public static void main(String[] args) {
        FileReader reader;
        char[] buffer = new char[100];
        String file_name = ".classpath";
        try{ reader = new FileReader(file_name);
        reader.read(buffer,0,100);
        String str = new String(buffer);
        System.out.println("읽은건 " + str);
        reader.close();
        } catch(FileNotFoundException e){
          System.out.println("그런 파일 없슴당");
        }catch(IOException e){
          System.out.println("읽다가 에러났슈");
        } finally{
         System.out.println("어쨌거나 읽었어요");
        }
    }
}
```

7] throw / throws/사용자 정의 Exception 1

- throw는 프로그래머가 임의로 예외를 발생시킬 때 사용된다.
  형식) throw 예외객체;
  throw new 예외객체타입(매개변수);
  예) throw new MyExceptionOne();
- throws는 예외를 직접처리하지 않고 자신을 호출한 메소드에게
  예외를 넘겨주는 방법에 사용
  형식) 메소드(매개변수) throws 해당Exception,.....
  예) public int add(int a) throws XXException, YYException{....}
- 사용자 정의 Exception은 상위 클래스에서 Exception을 상속받으면 된다.
  형식) class 클래스명 extends Exception{ }

7] throw / throws/사용자 정의 Exception 2

```
package ch10;

public class Throw1 {
        public static void main(String[] args) {
       System.out.println("안녕");
       Exception e1 = new Exception("내가 Error 만들었다");
           try {
                throw e1;
             //System.out.println("이건 안 출력");
           }   catch(Exception e) {
             System.out.println(e.getMessage());
           }
           System.out.println("프로그램이 정상 종료되었음.");
        }

}
```

8] 메서드에 예외 선언하기

- 예외를 처리하는 또 다른 방법
- 사실은 예외를 처리하는 것이 아니라, 호출한 메서드로 전달해주는 것
  - 호출한 메서드에서 예외처리를 해야만 할 때 사용

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

[참고] 예외를 발생시키는 키워드 throw와 예외를 메서드에 선언할 때 쓰이는 throws를 잘 구별하자.

```
public final void wait()
            throws InterruptedException
```

Causes current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object. In other words, this method behaves exactly as if it simply performs the call wait(0).

Throws:

IllegalMonitorStateException – if the current thread is not the owner of the object's monitor.
InterruptedException – if another thread interrupted the current thread before or while the current thread was waiting for a notification. The *interrupted status* of the current thread is cleared when this exception is thrown.

See Also:

notify(), notifyAll()

java.lang

Class IllegalMonitorStateException

java.lang.Object
  └ java.lang.Throwable
    └ java.lang.Exception
      └ java.lang.RuntimeException
        └ java.lang.IllegalMonitorStateException

java.lang

Class InterruptedException

java.lang.Object
  └ java.lang.Throwable
    └ java.lang.Exception
      └ java.lang.InterruptedException

9] 메서드에 예외 선언하기-예시1

```
class ExceptionEx18 {
	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();   // 일부로 예외 발생
}
```

```
class ExceptionEx23 {
	public static void main(String[] args) {
		try {    method1();
		} catch (Exception e) {
		System.out.println("main메서드에서 예외가 처리");
		}
	} // main메서드의 끝
	static void method1() throws Exception {
		try {
				throw new Exception();
		} catch (Exception e) {
				System.out.println("method1메서드에서 예외가 처리");
				throw e; // 다시 예외를 발생시킨다.
		}
	} // method1메서드의 끝
}
```

10] 사용자정의 예외 만들기

- 기존의 예외 클래스를 상속받아서 새로운 예외 클래스를 정의할 수 있다.

```
class MyException extends Exception {
     MyException(String msg) {  // 문자열을 매개변수로 받는 생성자
       super(msg); // 조상인 Exception클래스의 생성자를 호출한다.
     }
}
```

- 에러코드를 저장할 수 있게 ERR_CODE와 getErrCode()를 멤버로 추가

```
class MyException extends Exception {
    // 에러 코드 값을 저장하기 위한 필드를 추가 했다.
    private final int ERR_CODE;

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

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

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

MyExceptionTest.java 참조