FileIO, Exception Handling

514770
2018년 가을학기
9/17/2018
박경신

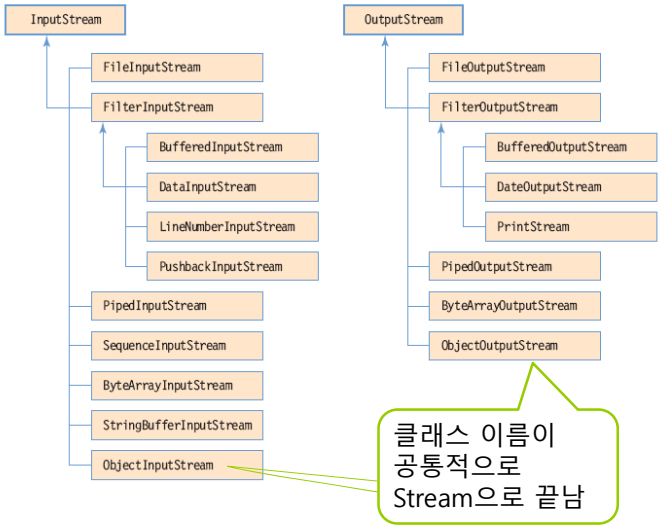
스트림

- 자바의 스트림
  - 자바 스트림은 입출력 장치와 자바 응용 프로그램 연결
  - 입출력 장치와 프로그램 사이의 데이터 흐름을 처리하는 소프트웨어 모듈
  - 입력 스트림
    - 입력 장치로부터 자바 프로그램으로 데이터를 전달하는 소프트웨어 모듈
  - 출력 스트림
    - 자바 프로그램에서 출력 장치로 데이터를 보내는 소프트웨어 모듈
- 입출력 스트림 기본 단위 : 바이트
- 자바 입출력 스트림 특징
  - 단방향 스트림, 선입선출 구조

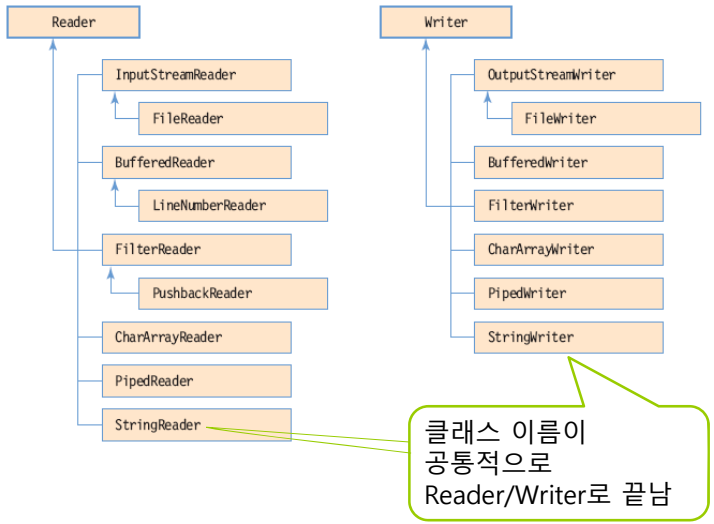
자바의 입출력 스트림 종류

- 바이트 입출력 스트림과 문자 입출력 스트림
  - 바이트 입출력 스트림
    - 입출력되는 데이터를 단순 바이트의 스트림으로 처리
    - 예) 바이너리 파일을 읽는 입력 스트림
  - 문자 입출력 스트림
    - 문자만 입출력하는 스트림
    - 문자가 아닌 바이너리 데이터는 스트림에서 처리하지 못함
    - 예) 텍스트 파일을 읽는 입력 스트림
- JDK는 입출력 스트림을 구현한 다양한 클래스 제공

JDK의 바이트 스트림 클래스 계층 구조



JDK의 문자 스트림 클래스 계층 구조



바이트 스트림 클래스

- 바이트 스트림
  - 바이트 단위의 바이너리 값을 읽고 쓰는 스트림
- 바이트 스트림 클래스
  - java.io 패키지에 포함
  - InputStream/OutputStream
    - 추상 클래스
    - 바이트 스트림을 다루는 모든 클래스의 슈퍼 클래스
  - **FileInputStream/FileOutputStream**
    - 파일로부터 바이트 단위로 읽거나 저장하는 클래스
    - 바이너리 파일의 입출력 용도
  - DataInputStream/DataOutputStream
    - 자바의 기본 데이터 타입의 값(변수)을 바이너리 값 그대로 입출력
    - 문자열도 바이너리 형태로 입출력

FileInputStream을 이용한 파일 읽기

- 파일 전체를 읽어 화면에 출력하는 코드 샘플

C:₩test.txt 파일을 열고 파일과 입력 바이트 스트림 객체 fin 연결

```
FileInputStream fin = new FileInputStream("c:₩₩test.txt");

int c;

while((c = fin.read()) != -1) {

  System.out.print((char)c);

}

fin.close();
```

파일 끝까지 바이트씩 c에 읽어 들임. 파일의 끝을 만나면 read()는 -1 리턴

바이트 c를 문자로 변환하여 화면에 출력

스트림을 닫음. 파일도 닫힘. 스트림과 파일의 연결을 끊음. 더 이상 스트림으로부터 읽을 수 없음

문자 스트림 클래스

- 문자 스트림
  - 유니 코드로 된 문자를 입출력 하는 스트림
    - 문자로 표현되지 않는 데이터는 다루지 않음
    - 문자 스트림은 이미지, 동영상과 같은 바이너리 데이터는 입출력 할 수 없음 - 문자 스트림은 문자 데이터만 입출력 가능
- 문자 스트림을 다루는 클래스
  - Reader/Writer
    - java.io 패키지에 포함
    - 추상 클래스. 문자 스트림을 다루는 모든 클래스의 슈퍼 클래스
  - InputStreamReader/OutputStreamWriter
    - 바이트 스트림과 문자 스트림을 연결시켜주는 다리 역할
    - 지정된 문자집합 이용
    - InputStreamReader : 바이트를 읽어 문자로 인코딩
    - OutputStreamWriter : 문자를 바이트로 디코딩하여 출력
  - **FileReader/FileWriter**
    - 텍스트 파일에서 문자 데이터 입출력

FileReader를 이용한 텍스트 파일 읽기

- system.ini 파일 전체를 읽어 화면에 출력하는 코드 샘플

```
import java.io.*;
public class FileReaderEx {
  public static void main(String[] args) {
    FileReader in = null;
    try {
      // 파일로부터 문자 입력 스트림 생성
      in = new FileReader("c:₩₩windows₩₩system.ini");
      int c;
      while ((c = in.read()) != -1) { // 한 문자씩 읽는다.
            System.out.print((char)c);
      }
      in.close();
    }
    catch (IOException e) {
      System.out.println("입출력 오류");
    }     }   }
```

C:₩Windows₩system.ini 파일을 열고 문자입력스트림 연결

파일의 끝을 만나면 read()는 -1 리턴

버퍼 입출력 스트림 클래스

- 버퍼 스트림
  - 버퍼를 가진 스트림으로써 입출력 데이터를 일시적으로 저장하는 버퍼를 이용하여 입출력 효율 개선
  - 입출력 시 운영체제의 API 호출 횟수를 줄여 입출력 성능 개선
    - 출력 시 여러 번 출력되는 데이터를 버퍼에 모아두고 한 번에 장치로 출력
    - 입력 시 입력 데이터를 버퍼에 모아두고 한번에 프로그램에게 전달
- **바이트 버퍼 스트림 클래스**
  - BufferedInputStream와 BufferedOutputStream
  - 바이트 단위의 바이너리 데이터(Binary Data)를 처리하는 버퍼 스트림
- **문자 버퍼 스트림 클래스**
  - **BufferedReader와 BufferedWriter**
  - **유니코드의 문자 데이터(Text Data)만 처리하는 버퍼 스트림**

텍스트 파일 읽기

- BufferedReader 클래스를 사용한 텍스트 파일 read

```
import java.io.*;
public class BufferedReaderExample {
   public static void main(String args[]) throws Exception{
        FileReader fr = new FileReader("C:/test.txt");
        BufferedReader br = new BufferedReader(fr);
        int ch;
        while((ch=br.read())!=-1){
        System.out.print((char)ch);
        }
        br.close();
        fr.close();
   }
}
```

파일 전체를 읽어 화면에 출력

텍스트 파일 읽기

- BufferedReader 클래스를 사용한 텍스트 파일 readLine
  (파일을 한 줄씩 읽어서 lines 배열에 저장)

```
import java.io.*;
public class BufferedReaderExample2 {
   static String[] lines = new String[10]; // 10개의 라인 배열

   public static void main(String args[]) throws Exception{
        BufferedReader br = new BufferedReader(new FileReader("C:/test.txt"));
        int i = 0;
        String line = "";
        while ((line=br.readLine())!= null){
          lines[i++] = line;
        }
        br.close();
        for (String l : lines) System.out.println(l); // lines 배열 출력
   }
}
```

한 줄씩 읽어 lines 배열에 저장

File 클래스

- File 클래스
  - 파일의 경로명을 다루는 클래스
    - java.io.File
    - 파일과 디렉터리 경로명의 추상적 표현
  - 파일 이름 변경, 삭제, 디렉터리 생성, 크기 등 파일 관리
    - File 객체는 파일 읽고 쓰기 기능 없음
  - 파일 입출력은 파일 입출력 스트림 이용

File 클래스 생성자와 주요 메소드

메소드	설명
File(File parent, String child)	parent 디렉터리에 child 이름의 디렉터리나 파일을 나타내는 File 객체 생성
File(String pathname)	pathname이 나타내는 File 객체 생성
File(String parent, String child)	parent 디렉터리에 child 이름의 디렉터리나 파일을 나타내는 File 객체 생성
File(URI uri)	file:URI를 추상 경로명으로 변환하여 File 객체 생성

메소드	설명
boolean mkdir()	새로운 디렉터리 생성
String[] list()	디렉터리 내의 파일과 디렉터리 이름의 문자열 배열 리턴
File[] listFiles()	디렉터리 내의 파일 이름의 File 배열 리턴
boolean renameTo(File dest)	dest가 지정하는 파일 이름 변경
boolean delete()	파일 또는 디렉터리 삭제
long length()	파일의 크기 리턴. 디렉터리나 장치 파일인 경우 0 리턴
String getPath()	파일 경로명 전체를 문자열로 변환하여 리턴
String getName()	파일 또는 디렉터리 이름을 문자열로 리턴
boolean isFile()	일반 파일이면 true 리턴
boolean isDirectory()	디렉터리이면 true 리턴
long lastModified()	파일이 마지막으로 변경된 시간 리턴
boolean exists()	파일 또는 디렉터리가 존재하면 true 리턴

File 클래스 예제

파일 객체 생성

```
File f = new File("c:₩₩test.txt");
```

파일인지 디렉터리인지 구분

```
File f = new File("c:₩₩windows₩₩system.ini");
String res;
if(f.isFile()) // 파일 타입이면
  res = "파일";
else // 디렉터리 타입이면
  res = "디렉터리";
System.out.println(f.getPath() + "은 " + res + "입니다.");
```

```
c:\windows\system.ini은 파일입니다.
```

서브 디렉터리 리스트 얻기

```
File f = new File("c:₩₩tmp₩₩java_sample");
String[] filenames = f.list(); // 파일명 리스트 얻기
for (int i=0; i<filenames.length; i++) {
  File sf = new File(f, filenames[i]);
  System.out.print(filenames[i]);
  System.out.print("₩t파일 크기: " + sf.length());
}
```

예외와 예외 클래스

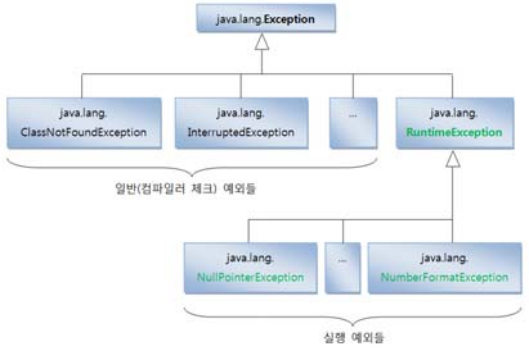
- 오류의 종류
  - 에러(Error)
    - 하드웨어의 잘못된 동작 또는 고장으로 인한 오류
    - 에러가 발생되면 JVM실행에 문제가 있으므로 프로그램 종료
    - 정상 실행 상태로 돌아갈 수 없음
  - 예외(Exception)
    - 사용자의 잘못된 조작 또는 개발자의 잘못된 코딩으로 인한 오류
    - 예외가 발생되면 프로그램 종료
    - "예외 처리" 추가하면 정상 실행 상태로 돌아갈 수 있음

예외와 예외 클래스

- 예외의 종류
  - 일반 예외(컴파일 체크 Exception)
    - 컴파일하는 과정에서 예외 처리 코드가 필요한지 검사
    - 예외 처리 코드 없으면 컴파일 오류 발생
  - **실행 예외(RuntimeException)**
    - 예외 처리 코드를 생략하더라도 컴파일이 되는 예외
    - '경험'따라 예외 처리 코드 작성 필요

예외 클래스

- 예외 클래스
  - **Java는 예외를 클래스로 관리**
  - JVM이 프로그램을 실행하는 도중에 예외가 발생하면 해당 예외 클래스로 객체를 생성
    - 예외 처리코드에서 예외 객체를 이용



자주 발생하는 예외

예외 종류	예외 발생 경우
ArithmeticException	정수를 0으로 나눌 때 발생
NullPointerException	null 레퍼런스를 참조할 때 발생
ClassCastException	변환할 수 없는 타입으로 객체를 변환할 때 발생
OutOfMemoryError	메모리가 부족한 경우 발생
ArrayIndexOutOfBoundsException	배열의 범위를 벗어난 접근 시 발생
IllegalArgumentException	잘못된 인자 전달 시 발생
IOException	입출력 동작 실패 또는 인터럽트 시 발생
NumberFormatException	문자열이 나타내는 숫자와 일치하지 않는 타입의 숫자로 변환 시 발생

실행 예외(RuntimeException)

- **NullPointerException**
  - 객체 참조가 없는 상태
    - null 값 갖는 참조변수로 객체 접근 연산자인 도트(.) 사용했을 때 발생

```
String data = null;
System.out.println(data.toString());
```

- **ArrayIndexOutOfBoundsException**
  - 배열에서 인덱스 범위 초과하여 사용할 경우 발생

```
public static void main(String[] args) {
  String data1 = args[0];
  String data2 = args[1];

  System.out.println("args[0]: " + data1);
  System.out.println("args[1]: " + data2);
}
```

실행시 매개값을 주지 않을 경우 예외 발생

실행 예외(RuntimeException)

- **NumberFormatException**
  - 숫자로 변환될 수 없는 문자가 포함되어 있는 문자열을 숫자로 변경할 경우
    - Integer.parseInt(String s)
    - Double.parseDouble(String s)

```
public class NumberFormatExceptionExample {
	public static void main(String[] args) {
		String data1 = "100";
		String data2 = "a100";

		int value1 = Integer.parseInt(data1);
		int value2 = Integer.parseInt(data2);

		int result = value1 + value2;
		System.out.println(data1 + "+" + data2 + "=" + result);
	}
}
```

실행 예외(RuntimeException)

- **ClassCastException**
  - 타입 변환이 되지 않을 경우 발생

  - 정상 코드

Animal animal = new Dog(); Dog dog = (Dog) animal;	RemoteControl rc = new Television(); Television tv = (Television) rc;

  - 예외 발생 코드

Animal animal = new Dog(); Cat cat = (Cat) animal;	RemoteControl rc = new Television(); Audio audio = (Audio) rc;

실행 예외(RuntimeException)

- **ClassCastException**
  - 타입 변환이 되지 않을 경우 발생

```
public class NumberFormatExceptionExample {
	public static void main(String[] args) {
		Dog dog = new Dog();
		changeDog(dog);
		Cat cat = new Cat();
		changeDog(cat);
	}
	public static void changeDog(Animal animal) {
		//if(animal instanceof Dog) {
		    Dog dog = (Dog) animal;      //ClassCastException 발생 가능
		//}
	}
}
class Animal {}
class Dog extends Animal {}

class Cat extends Animal {}
```

예제 : ArithmeticException 예외 처리

try-catch문을 이용하여 정수를 0으로 나누려고 할 때 "0으로 나룰 수 없습니다."라는 경고 메시지를 출력하도록 프로그램을 작성하시오.

```
import java.util.Scanner;
public class ExceptionExample2 {
   public static void main (String[] args) {
      Scanner rd = new Scanner(System.in);
      int divisor = 0;
      int dividend = 0;
      System.out.print("나뉨수를 입력하시오:");
      dividend = rd.nextInt();
      System.out.print("나눗수를 입력하시오:");
      divisor = rd.nextInt();
      try {
         System.out.println(dividend+"를 "+divisor+"로 나누면 몫은 "+
dividend/divisor+"입니다.");
      } catch (ArithmeticException e) {
         System.out.println("0으로 나눌 수 없습니다.");
      }
   }
}
```

ArithmeticException 예외 발생

```
나뉨수를 입력하시오:100
나눗수를 입력하시오:0
0으로 나눌 수 없습니다.
```

예제 : 범위를 벗어난 배열의 접근

배열의 인덱스가 범위를 벗어날 때 발생하는 ArrayIndexOutOfBoundsException을 처리하는 프로그램을 작성하시오.

```
public class ArrayException {
  public static void main (String[] args) {
    int[] intArray = new int[5];
    intArray[0] = 0;
    try {
      for (int i = 0; i < 5; i++) {
        intArray[i+1] = i+1 + intArray[i];
        System.out.println("intArray["+i+"]"+"="+intArray[i]);
      }
    }
    catch (ArrayIndexOutOfBoundsException e) {
      System.out.println("배열의 인덱스가 범위를 벗어났습니다.");
    }
  }
}
```

i가 4일 때 ArrayIndexOutOfBoundsException 예외 발생

```
intArray[0]=0
intArray[1]=1
intArray[2]=3
intArray[3]=6
배열의 인덱스가 범위를 벗어났습니다.
```

예제 : 정수가 아닌 문자열을 정수로 변환할 때 예외 발생

문자열을 정수로 변환할 때 발생하는 NumberFormatException을 처리하는 프로그램을 작성하라.

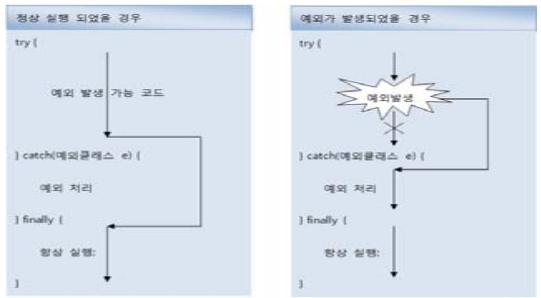
```
public class NumException {
  public static void main (String[] args) {
    String[] stringNumber = {"23", "12", "998", "3.14
    try {
      for (int i = 0; i < stringNumber.length; i++) {
        int j = Integer.parseInt(stringNumber[i]);
        System.out.println("숫자로 변환된 값은 " + j);
      }
    }
    catch (NumberFormatException e) {
      System.out.println("정수로 변환할 수 없습니다.");
    }
  }
}
```

"3.141592"를 정수로 변환할 때 NumberFormatException 예외 발생

```
숫자로 변환된 값은 23
숫자로 변환된 값은 12
숫자로 변환된 값은 998
정수로 변환할 수 없습니다.
```

예외처리 코드 (try-catch-finally)

- 예외처리 코드
  - 예외 발생시 프로그램 종료 막고, 정상 실행 유지할 수 있도록 처리
    - 일반 예외: 반드시 작성해야 컴파일 가능
    - 실행 예외: 컴파일러가 체크해주지 않으며 개발자 경험 의해 작성
  - **try – catch – finally** 블록 이용해 예외 처리 코드 작성

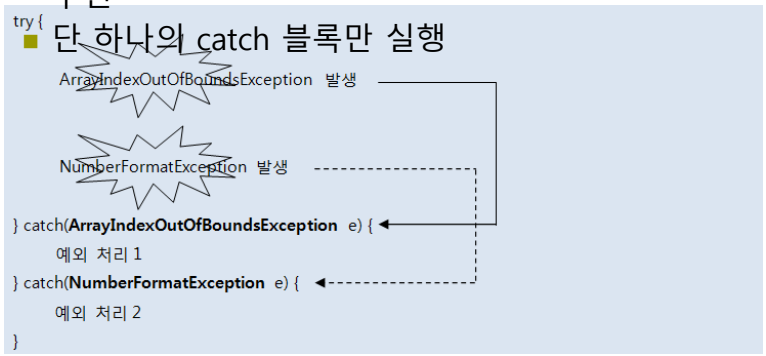


예외 처리 코드(try-catch-finally)

```
public class TryCatchFinallyRuntimeExceptionExample {
    public static void main(String[] args) {
        String data1 = null;
        String data2 = null;
        try {                                   실행시 매개값을 주지 않을 경우 예외 발생
            data1 = args[0];
            data2 = args[1];
        } catch(ArrayIndexOutOfBoundsException e) {
            System.out.println("실행 매개값의 수가 부족합니다.");
            System.out.println("[실행 방법]");
            System.out.println("java TryCatchFinallyRuntimeExceptionExample num1 num2");
            return;
        }
        try {
            int value1 = Integer.parseInt(data1);  실행시 매개값을 잘못 주었을 경우 예외 발생
            int value2 = Integer.parseInt(data2);
            int result = value1 + value2;
            System.out.println(data1 + "+" + data2 + "=" + result);
        } catch(NumberFormatException e) {
            System.out.println("숫자로 변환할 수 없습니다.");
        } finally {
            System.out.println("다시 실행하세요.");
        }
    }
}
```

예외 종류에 따른 처리 코드

- **다중 catch**
  - 하나의 try 블록 내부에서 다양한 종류의 예외 발생시
  - 각 예외 별로 예외 처리 코드(catch 블록) 다르게 구현
  - 단 하나의 catch 블록만 실행



예외 종류에 따른 처리 코드

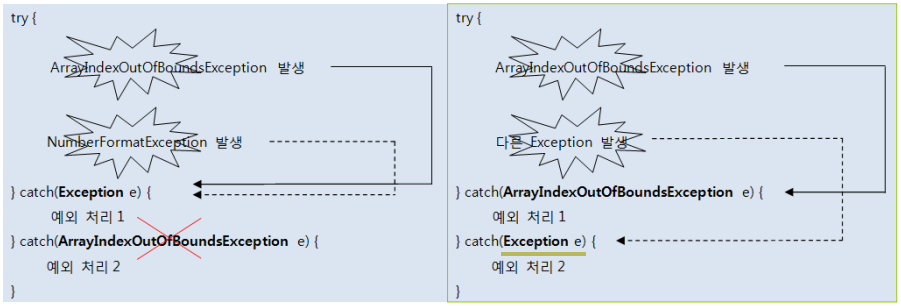
```
public class CatchByExceptionKindExample {
    public static void main(String[] args) {
        try {
            String data1 = args[0];
            String data2 = args[1];
            int value1 = Integer.parseInt(data1);
            int value2 = Integer.parseInt(data2);
            int result = value1 + value2;
            System.out.println(data1 + "+" + data2 + "=" + result);
        } catch(ArrayIndexOutOfBoundsException e) {
            System.out.println("실행 매개값의 수가 부족합니다.");
            System.out.println("[실행 방법]");
            System.out.println("java CatchByExceptionKindExample  num1  num2");
        } catch(NumberFormatException e) {
            System.out.println("숫자로 변환할 수 없습니다.");
        } finally {
            System.out.println("다시 실행하세요.");
        }
    }
}
```

실행시 매개값을 주지 않을 경우 예외 발생

실행시 매개값을 잘못 주었을 경우 예외 발생

예외 종류에 따른 처리 코드

- 하위 예외는 상위 예외를 상속
  - 하위 예외는 상위 예외 타입도 됨
- **catch** 순서 – 상위 예외가 아래에 위치해야



상위 예외 Exception이 위에 있을 경우

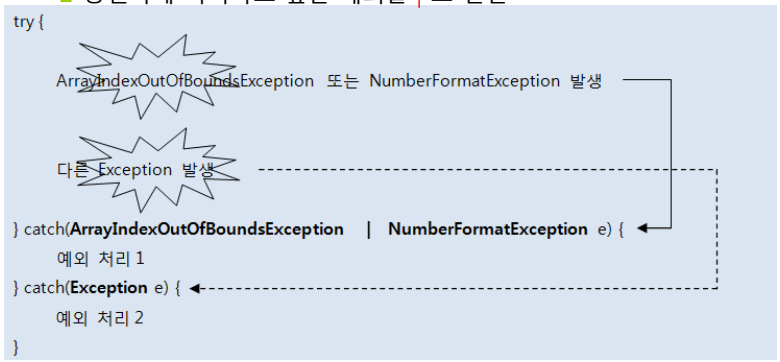
상위 예외 Exception이 아래에 있을 경우

예외 종류에 따른 처리 코드

```
public class CatchOrderExample {
    public static void main(String[] args) {
        try {
            String data1 = args[0];
            String data2 = args[1];
            int value1 = Integer.parseInt(data1);
            int value2 = Integer.parseInt(data2);
            int result = value1 + value2;
            System.out.println(data1 + "+" + data2 + "=" + result);
        } catch(ArrayIndexOutOfBoundsException e) {
            System.out.println("실행 매개값의 수가 부족합니다.");
        } catch(Exception e) {
            System.out.println("실행에 문제가 있습니다.");
        } finally {
            System.out.println("다시 실행하세요.");
        }
    }
}
```

예외 종류에 따른 처리 코드

- **멀티(multi) catch**
  - 자바 7부터는 하나의 catch 블록에서 여러 개의 예외 처리 가능
    - 동일하게 처리하고 싶은 예외를 | 로 연결



예외 종류에 따른 처리 코드

```
public class MultiCatchExample {
	public static void main(String[] args) {
		try {
			String data1 = args[0];
			String data2 = args[1];
			int value1 = Integer.parseInt(data1);
			int value2 = Integer.parseInt(data2);
			int result = value1 + value2;
			System.out.println(data1 + "+" + data2 + "=" + result);
		} catch(ArrayIndexOutOfBoundsException | NumberFormatException e) {
			System.out.println("실행 매개값의 수가 부족하거나 숫자로 변환할 수
없습니다..");
		} catch(Exception e) {
			System.out.println("알수 없은 예외 발생");
		} finally {
			System.out.println("다시 실행하세요.");
		}
	}
}
```

실행시 매개값을 주지 않을 경우 예외 발생

실행시 매개값을 잘못 주었을 경우 예외 발생

자동 리소스 닫기

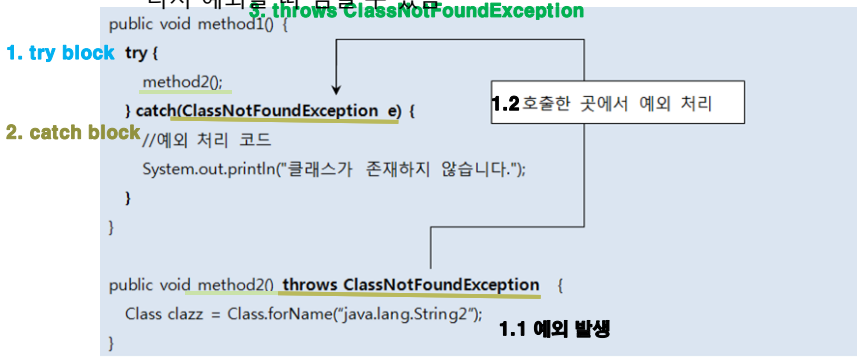
- try-with-resources
  - 예외 발생 여부와 상관 없음
  - 사용했던 리소스 객체의 close() 메소드 호출해 리소스 닫음
  - 리소스 객체
    - 각종 입출력스트림, 서버소켓, 소켓, 각종 채널
    - java.lang.AutoCloseable 인터페이스 구현하고 있어야 함

예외 떠 넘기기(throws)

- **throws**
  - 메소드 선언부 끝에 작성
  - 메소드내에서 처리하지 않은 예외를 메소드 호출한 곳(calling method)으로 떠 넘기는 역할

예외 떠 넘기기(throws)

- **throws**
  - throws 선언된 메소드를 호출하는 메소드(calling method)는
    1. 반드시 try 블록 내에서 호출
    2. catch 블록에서 떠 넘겨 받은 예외를 처리함
    3. try-catch 블록으로 예외처리를 하지 않고 throws 키워드로 자신도 다시 예외를 떠 넘길 수 있음




```
public void method1() {
  try {
    method2();
  } catch(ClassNotFoundException e) {
    //예외 처리 코드
    System.out.println("클래스가 존재하지 않습니다.");
  }
}

public void method2() throws ClassNotFoundException {
  Class clazz = Class.forName("java.lang.String2");
}
```

사용자 정의 예외와 예외 발생

- 사용자 정의 예외(user-defined exception) 클래스 선언
  - 자바 표준 API에서 제공하지 않는 예외
  - 애플리케이션 서비스와 관련된 예외, Application Exception
    - E.g. 잔고 부족 예외, 계좌 이체 실패 예외, 회원 가입 실패 예외....
  - 사용자 정의 예외 클래스 선언 방법
    1. 예외 클래스 상속
       - 일반 예외 : Exception class 상속
       - 실행 예외 : RuntimeException class 상속
    2. 생성자 정의
       - 매개변수 없는 기본 생성자, String 타입의 매개변수를 갖는 생성자

```
public class XXXException extends [ Exception | RuntimeException ] {
    public XXXException() { }
    public XXXException(String message) { super(message); }
}
```

사용자 정의 예외와 예외 발생

- 예외 발생 시키기(throw)
  - 코드에서 예외 발생시키는 법 - 예외 객체 생성

```
throw new XXXException()
throw new XXXException("메시지");
```

  - 호출한 곳에서 발생한 예외를 처리하도록

```
public void method() throws XXXException {
    throw new XXXException("메시지");
}
```

사용자 정의 예외와 예외 발생

```
public class BalanceInsufficientException extends Exception {
    public BalanceInsufficientException() { }
    public BalanceInsufficientException(String message) {
        super(message);
    }
}
```

1. Exception class 상속
2. constrctors 생성

```
public class Account {
    private long balance;

    public Account() { }

    public long getBalance() {
        return balance;
    }
    public void deposit(int money) {
        balance += money;
    }
    public void withdraw(int money) throws BalanceInsufficientException {
        if(balance < money) {
            throw new BalanceInsufficientException("잔고부족:"+(money-balance)+" 모자람");
        }
        balance -= money;
    }
}
```

4. 호출한 곳에서 발생한 예외를 처리하도록 함
3. 사용자 정의 예외 발생 – 예외 객체 생성

사용자 정의 예외와 예외 발생

```
public class AccountExample {
	public static void main(String[] args) {

		Account account = new Account();

		//예금하기
		account.deposit(10000);
		System.out.println("예금액: " + account.getBalance());

		//출금하기
		try {
			account.withdraw(30000);
		} catch(BalanceInsufficientException e) {
			String message = e.getMessage();
			System.out.println(message);
			System.out.println();
			e.printStackTrace();
		}
	}
}
```

5. try block에서 메소드 호출

6. catch block에서 사용자정의 예외 처리

7. 예외 정보 얻기(8절)

예외 정보 얻기

- **getMessage()**
  - 예외 발생시킬 때 생성자 매개값으로 사용한 메시지 리턴

```
throw new XXXException("예외 메시지");
```

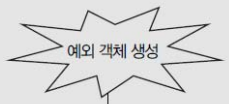
  - 원인 세분화하기 위해 예외 코드 포함(예: 데이터베이스 예외 코드)
  - catch() 절에서 활용

```
} catch(Exception e) {
    String message = e.getMessage();
}
```

예외 정보 얻기

- **printStackTrace()**
  - 예외 발생 코드를 추적하여 모두 콘솔에 출력

```
try {
```




```
} catch(예외클래스 e) {
    //예외가 가지고 있는 Message 얻기
    String message = e.getMessage();

    //예외의 발생 경로를 추적
    e.printStackTrace();
}
```

Problems Javadoc Declaration Console

<terminated> AccountExample [Java Application] C:₩Program Files₩Java₩jdk1.8.0_73₩bin₩javaw.exe (2

```
예금액: 10000
잔고부족:20000 모자람

BalanceInsufficientException: 잔고부족:20000 모자람
        at Account.withdraw(Account.java:14)
        at AccountExample.main(AccountExample.java:9)
```