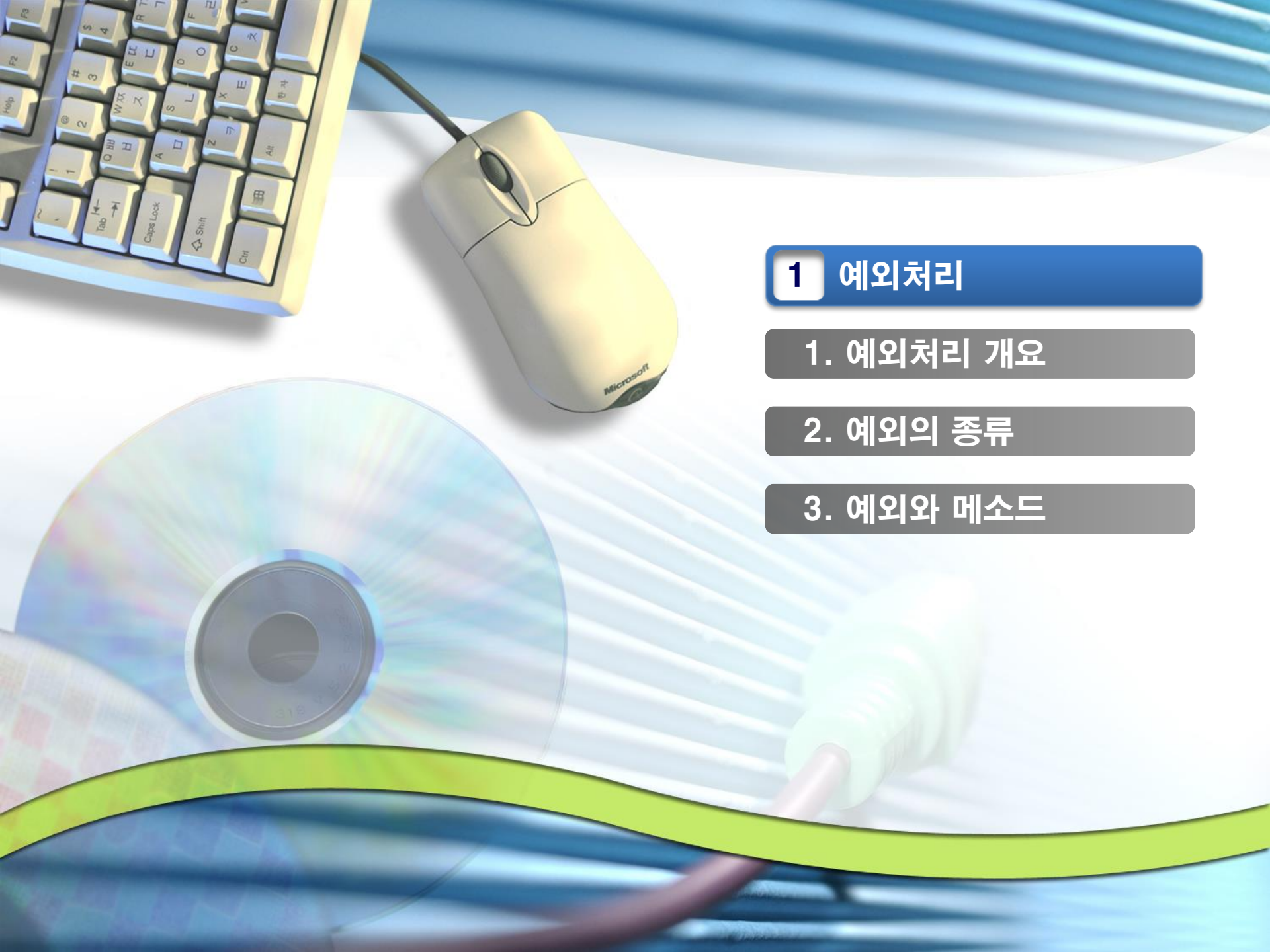

JAVA
예외처리



1 예외처리

1. 예외처리 개요

2. 예외의 종류

3. 예외와 메소드

프로그램의 오류

프로그램이 정상적으로 실행되지 못하고 오작동을 하거나 비정상적으로 종료될 수 있다 이를 프로그램 오류라 한다.

- **오류**
  - **문법적 오류** : 문법이 올바르지 않은 상태
  - **실행 오류**
    - **오류** : 시스템이 정상적인 기능을 수행할 수 없게 된 상태(수습 불가)
    - **예외** : 실행 중 예기치 않은 문제가 발생한 상태(수습 가능)
  - **논리적 오류** :개발자가 원하는 결과가 산출되지 않은 상태

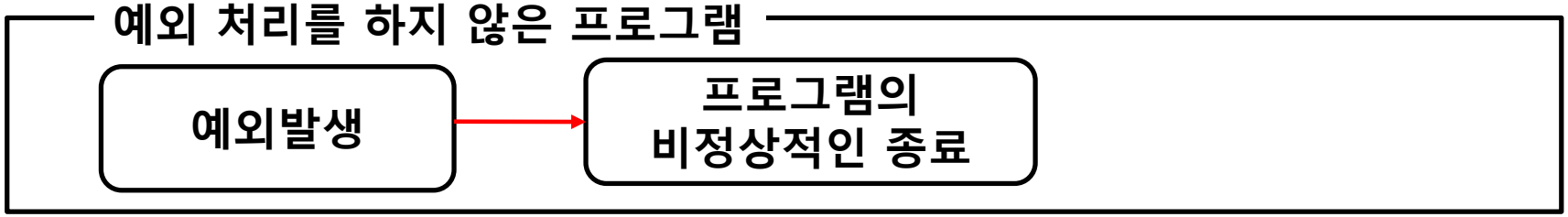
오류의 종류

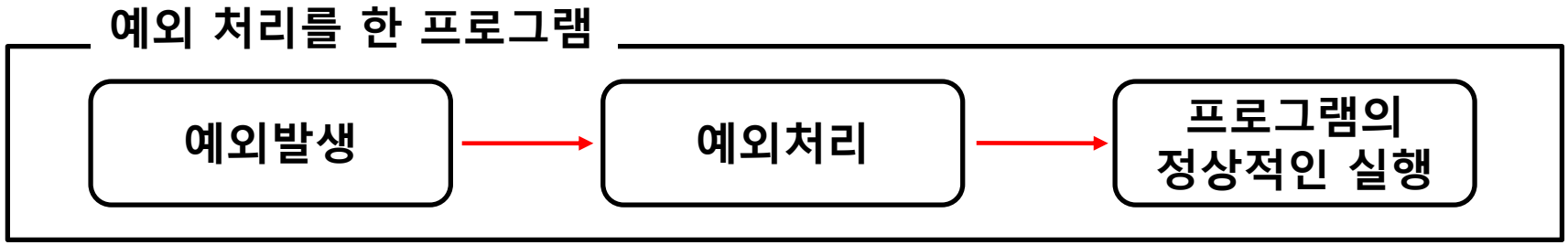
- 에러(Error)
  - 하드웨어의 잘못된 동작 또는 고장으로 인한 오류.
  - 에러가 발생되면 프로그램 종료.
  - 정상 실행 상태로 돌아갈 수 없음.

- 예외(Exception)
  - 사용자의 잘못된 조작 또는 부정확한 데이터 처리 또는 개발자의 잘못된 코딩으로 인한 오류.
  - 예외가 발생되면 프로그램 종료
  - 예외 처리 추가하면 정상 실행 상태로 돌아갈 수 있음.

오류의 종류

예외란 프로그램 실행 중에 발생하며 정상적으로 실행되는 흐름을 중단시키는 것을 말한다. 중단된다는 점은 오류와 같지만 이런 여기치 못한 상황에 대해서, 즉 예외에 대해서 미리 코드를 작성해서 대처함으로써 프로그램이 비정상적으로 종료되는 것을 막고, 정상적으로 종료되도록 할 수 있다. 예외란 이렇게 대처가 가능하다는 점에서 오류와 다르며, **미리 코드를 작성해서 대처하는 과정을 예외처리(exception handling)**라 한다.





예외 처리를 하면 프로그램 실행 중 예기치 않는 상황이 발생하더라도 프로그램이 비정상적으로 종료되지 않고 대처해 놓은 코드를 실행해서 정상적인 상태를 유지

예외란?

예외란 " Exceptional Event"의 약자이다.

- 자바에서는 예외도 하나의 객체로 취급한다.
- 오류가 발생하면 발생된 오류를 설명하는 객체를 생성하게 되는데 그 객체를 예외객체(exception object)라고 한다.
- 예외객체는 오류정보, 오류의 타입과 오류 발생 시의 프로그램의 상태 등의 정보를 포함하고 있다.
- 예외객체를 생성하는 것을 흔히 예외를 던진다(throw)고 하고 예외 객체를 처리하는 것을 예외를 잡는다(catch)고 한다.

예외의 예

실행되지 않음!

```
public class DivideByZeroTest {
        public static void main(String[] args) {
                int x = 1;
                int y = 0;
                int result = x / y;          // 예외 발생!
                System.out.println("result :"+result);
        }
}
```

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at DivideByZeroTest.main(DivideByZeroTest.java:7)
```

예외의 예

실행되지 않음!

```
public class BadIndex {
        public static void main(String[] args) {
                int[] array = new int[10];
                for (int i = 0; i < 10; i++)
                        array[i] = i+1;
                int result = array[10];
                System.out.println(result);
                System.out.println("과연 이 문장이
                                        실행될까요?");
        }
}
```

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 10
        at exam_try_catch_finally.BadIndex.main(BadIndex.java:8)
```

예외 처리기의 개요

- 예외 처리 코드
  - 예외 발생시 프로그램 종료 막고, 정상 실행 유지할 수 있도록 처리
  - try ~ catch ~ finally블록 이용해 예외 처리 코드 작성

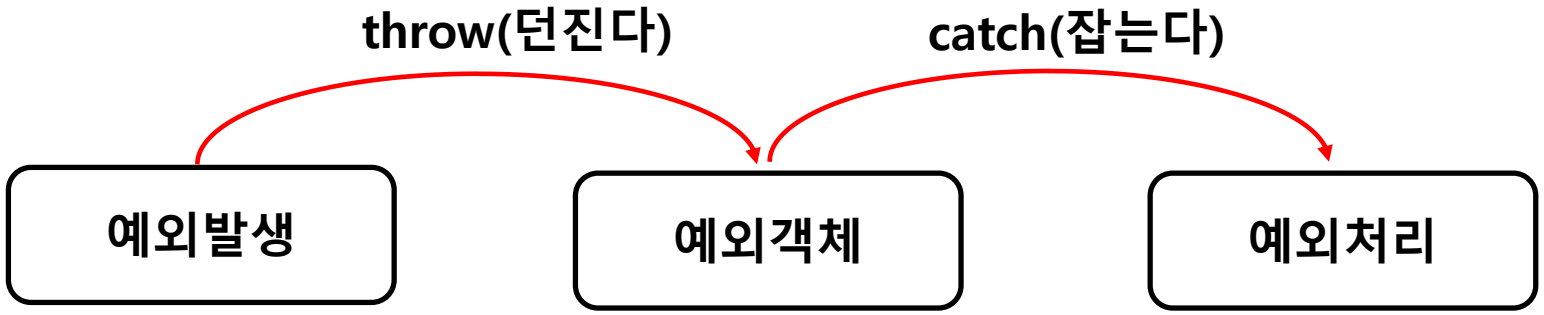
예외 처리기의 개요

오류가 발생하였건 발생하지 않았건 항상
실행되어야 하는 코드는 finally 블록에 넣을 수 있다.

- 예외 처리 코드

```
try {
      // 예외가 발생할 수 있는 코드
} catch (예외종류 참조변수) {
     // 예외를 처리하는 코드
} finally {
    // 해당 코드는 try 블록이 끝나면 무조건 실행된다.
}
```

예외에 대한 정보



예외객체는 예외 유형과 예외 발생시 프로그램 상태 등 예외 정보를 가지고 있다.

예외 처리기의 개요

- 약간의 주의할 점을 살펴보자.
  - try와 catch블록은 별도의 독립된 블록이다.
    따라서 try 블록에서 정의된 변수는 catch블록에서 사용할 수 없다.
  - 예외의 종류에 따라서 여러 개의 catch 블록이 있을 수 있으며, 이 중에서 발생한 예외의 종류와 일치하는 catch 블록만 실행된다. 만약 일치되는 catch블록이 없으면 발생된 예외는 처리되지 않는다.

try/catch 블록에서의 실행 흐름

정상 실행 되었을 경우	예외가 발생되었을 경우

try{
예외 발생 가능 코드
} catch(예외클래스 e){
예외 처리
} finally {
항상 실행
}
	try{
	예외 발생
	} catch(예외클래스 e){
	예외 처리
	} finally {
	항상 실행
	}

try/catch 블록에서의 실행 흐름

```
int x = 6, y = 2;
try {
    int result = x / y;

} catch (Exception e) {
    System.out.println("오류발생");
}
System.out.println("try/catch 통과");
```

예외가 발생하지 않은 경우

```
int x = 6, y = 0;
 try {
    int result = x / y;

 } catch (Exception e) {
    System.out.println("오류발생");
 }
System.out.println("try/catch 통과");
```

예외가 발생한 경우

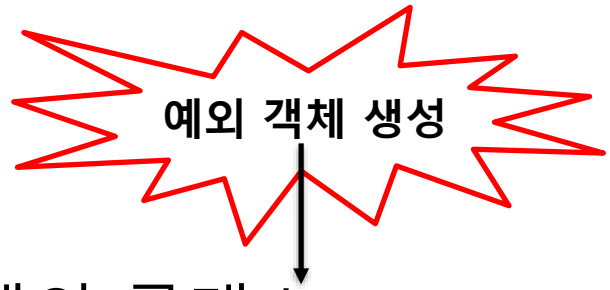
예외 정보 얻기

- **printStackTrace()**
  - **예외 발생 코드 추적한 내용을 모두 콘솔에 출력**
  - **프로그램 테스트하면서 오류 찾을** 때 유용하게 활용

```
try{

} catch (예외 클래스 e){
        // 예외가 가지고 있는 message 얻기
        String message = e.getMessage();

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



예외 처리 메서드

모든 예외 객체는 Throwable를 상속받는다. Throwable에 선언된 메서드 중 예외 처리에 자주 사용되는 메서드가 다음과 같다.

제어자 및 타입	메서드	설명
String	**getMessage()**	발생한 예외 객체의 메시지를 추출
void	**printStackTrace()**	예외가 발생하기까지 호출된 순서를 거꾸로 출력
String	**toString()**	발생한 예외 객체를 문자열로 추출

예제

```
public class DivideByZeroTest {
    public static void main(String[] args) {
        int x = 1;
        int y = 0;
        try {
            int result = x / y; // 예외 발생!
            System.out.println("result :"+result);
        } catch (ArithmeticException e) {
            System.out.println("0으로 나눌 수 없습니다.");
        }
        System.out.println("프로그램은 계속 진행됩니다.");
    }
}
```

[실행결과]

0으로 나눌 수 없습니다.

프로그램은 계속 진행됩니다.

예제

```
public class BadIndex2 {
        public static void main(String[] args) {
                int[] array = new int[10];
                for (int i = 0; i < 10; i++)
                        array[i] = 0;
                try {
                        int result = array[12];
                        System.out.println(result);
                } catch (ArrayIndexOutOfBoundsException e) {
                        System.out.println("배열의 인덱스가 잘못
                                                되었습니다.");
                }
                System.out.println("과연 이 문장이 실행될까요?");
        }
}
```

[실행결과]
배열의 인덱스가 잘못되었습니다.
과연 이 문장이 실행될까요?

finally 블록

finally문은 try 블록에서 사용했던 자원을 해제하기 위해서이다. try 블록에서 파일열기, 네트워크 연결, 데이터베이스 연결과 같은 작업을 했다면 작업이 완료된 후에는 파일닫기, 네트워크 연결 종료, 데이터베이스 연결 종료와 같은 자원 해제 작업을 해야 하는데 만일 자원 해제 작업을 하지 않으면 다른 프로그램에서 자원들이 필요할 때 사용할 수 없는 상황이 발생할 수도 있다.

예외 발생 차단

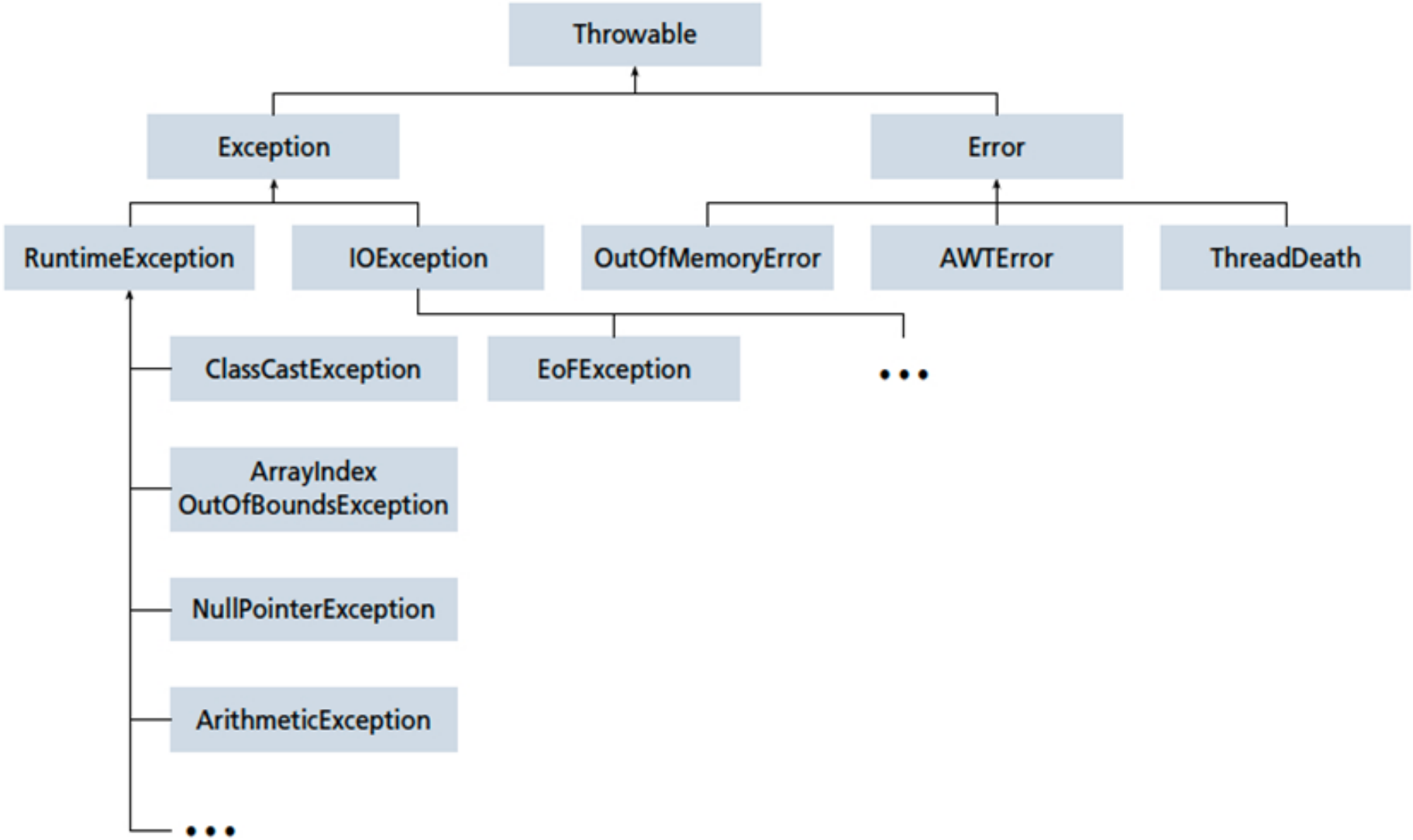
try ~ catch 구문은 유용하고 필요한 도구이다.

하지만 가장 최선의 방법은 역시 예외가 일어나지 않도록 하는 것이다. 물론 이것이 불가능한 경우도 많지만 미리 데이터를 테스트하여서 예외가 일어나지 않도록 하는 것이 가능하다.

위의 예제에서 나눗셈을 할 때 분모가 0인지를 검사하면 된다.

```
if ( y != 0 ) {
      result = x / y;
}
```

예외의 종류


Throwable
Exception
Error
RuntimeException
IOException
OutOfMemoryError
AWTError
ThreadDeath
ClassCastException
EoFException
• • •
ArrayIndex
OutOfBoundsException
NullPointerException
ArithmeticException
• • •

Error와 Exception

예외
(Throwable)

Error — JVM 내에서 발생하는 예외로서 프로그램 내에서 처리가 불가능하다는 것이다

Exception — 프로그램 내에서 발생하는 예외로서 프로그램 내에서 처리가 가능하다

- RunTimeException
- IOException

RunTimeException과 IOException

분류	예외	설명
RuntimeException	ArithmeticException	어떤 수를 0으로 나눌 때 발생한다.
	NullPointerException	Null 객체를 참조할 때 발생한다.
	ClassCastException	적절치 못한 클래스로 형변환하는 경우에 발생한다.
	NegativeArraySizeException	배열의 크기가 음수 값인 경우 발생한다.
	ArrayIndexOutOfBoundsException	배열을 참조하는 인덱스가 잘못된 경우 발생한다.

RunTimeException과 IOException

분류	예외	설명
IOException	IOException	파일의 끝을 지나쳐서 읽으려고 하는 경우나 잘못된 URL을 사용하는 경우에 발생한다.

RunTimeException 예외 클래스

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

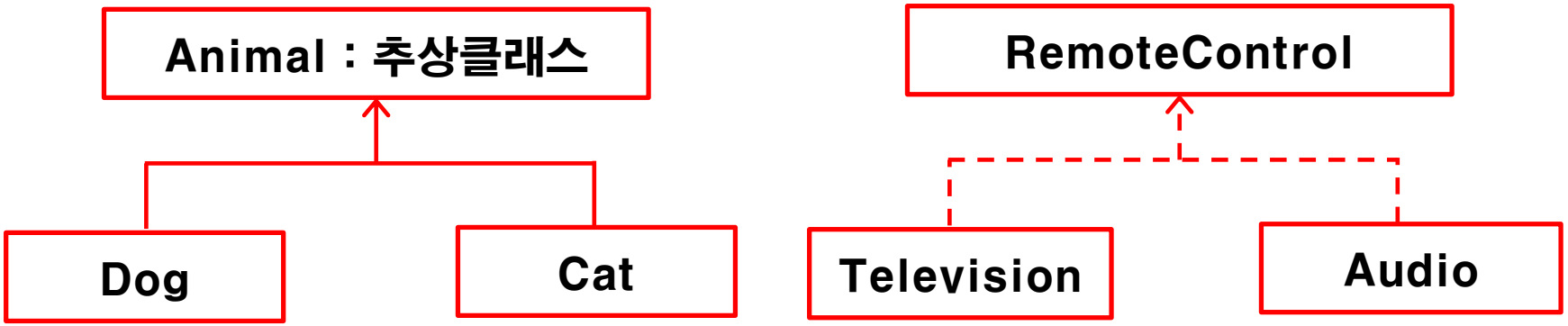
```
String data = null;
System.out.println(data.length());
```

- ArrayIndexOutOfBoundsException
  - 배열에서 인덱스 범위 초과하여 사용할 경우 발생

```
int[] arr=new int[5];
arr[5]=100;
```

RunTimeException 예외 클래스

- 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

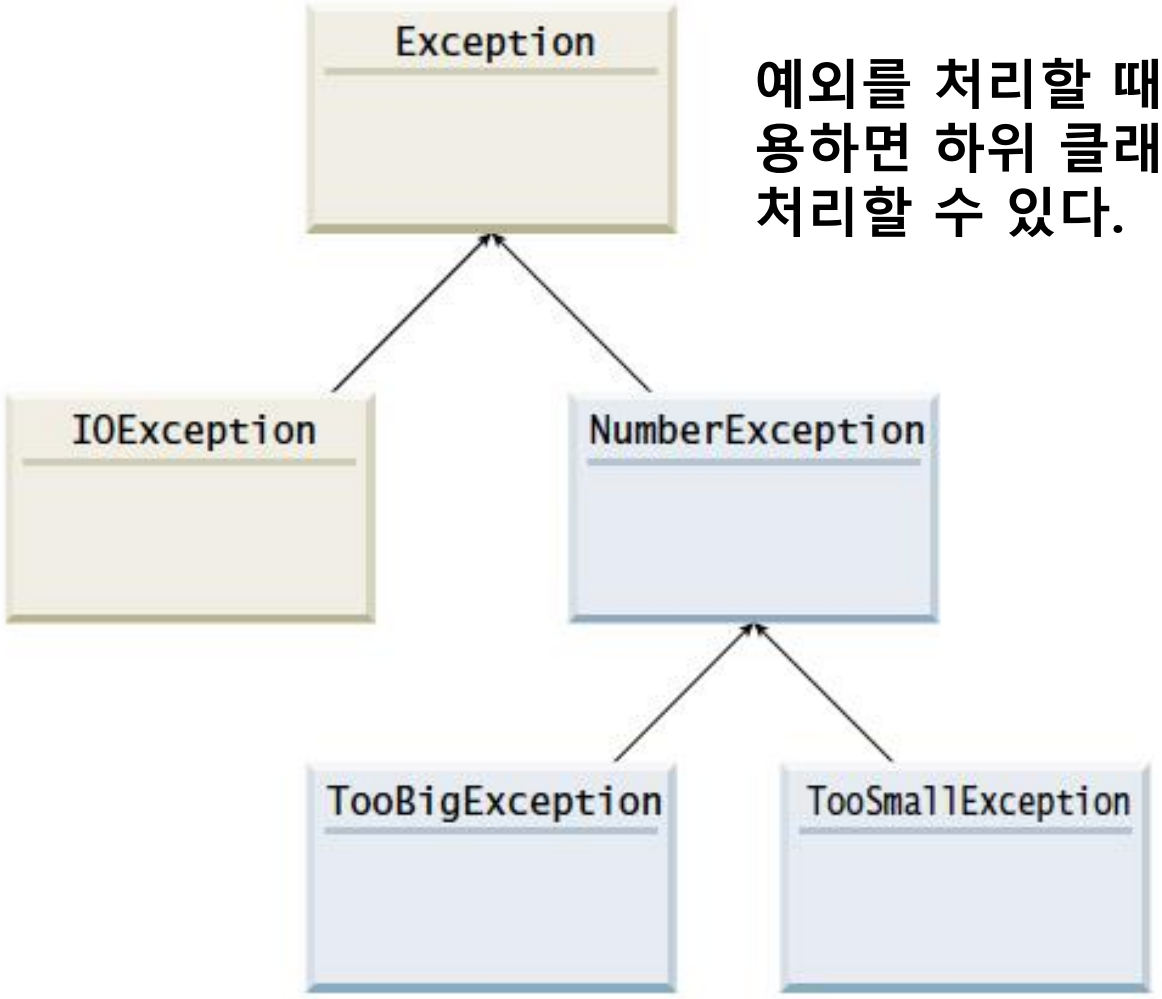
Animal animal = new Dog(); Cat cat = (Cat) animal;	RemoteControl rc = new Television(); Audio audio = (Audio)rc;

체크 예외와 비체크 예외

- RuntimeException은 대부분 버그에서 비롯되기 때문에 프로그래머가 신경쓰면 얼마든지 해결할 수 있기 때문에 이러한 오류들은 컴파일러가 처리했는지를 확인하지 않는다. 이것을 **비체크 예외(unchecked exception)**라고 한다. 비체크 예외는 코드에서 반드시 처리하지 않아도 된다.
- 반면에 RuntimeException을 제외한 모든 다른 예외는 컴파일러가 처리를 확인한다. 이것을 **체크 예외(checked exception)**라고 한다. 체크예외는 코드에서 반드시 처리를 해주어야 한다.

예외 처리 코드 없으면 컴파일 오류 발생

다형성과 예외



예외를 처리할 때에 상위 클래스를 이용하면 하위 클래스까지 포괄적으로 처리할 수 있다.

다형성과 예외

예를 들어서 아래와 같이 선언하면 TooBigException이나 TooSmallException도 잡을 수 있다.

```
try{
        getInput();
} catch (NumberException e){
      // NumberException의 하위 클래스를 모두 예외 처리
      // 할 수 있다.
}
```

```
try{
        getInput();
} catch (Exception e){
      // Exception의 모든 하위 클래스를 예외 처리하기 때문
     // 에 정확히 구분하기가 힘들다.
}
```

다형성과 예외

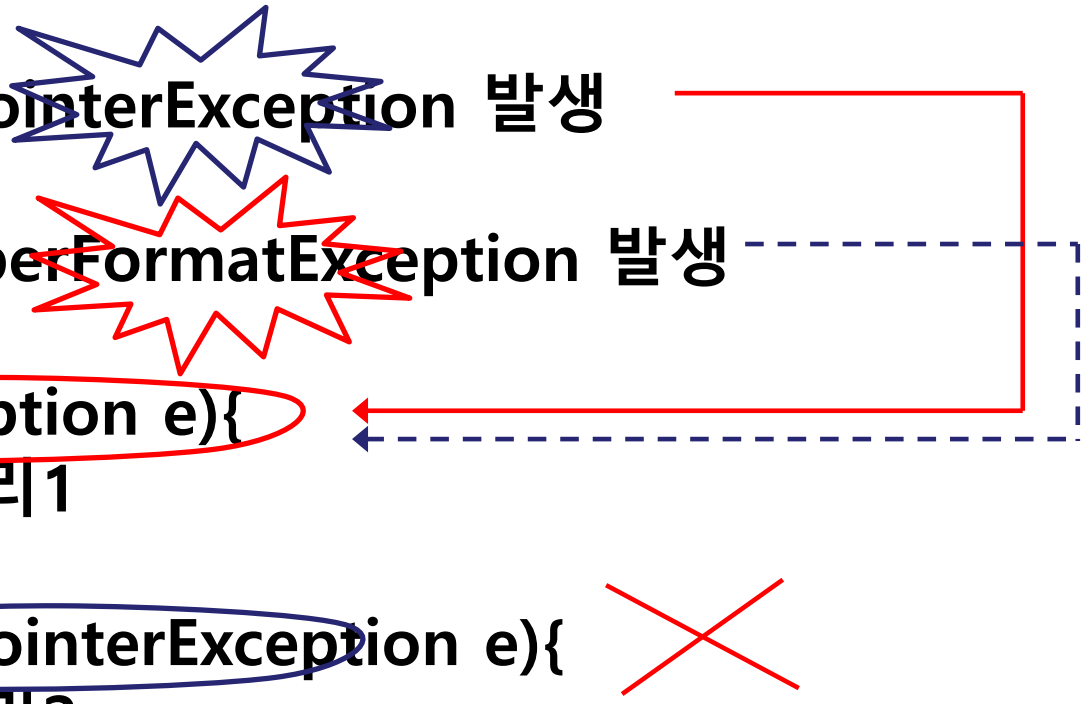
```
try{
        getInput();
} catch (TooSmallException e){
        // TooSmallException 예외만 처리
}catch (NumberException e){
        // TooSmallException을 제외한 나머지 예외 처리
}
```

```
try{
        getInput();
} catch (NumberException e){
        // NumberException 하위 모든 예외 처리
}catch (TooSmallException e){
        // 아무런 예외 처리도 할 수 없다.
}
```

예외 종류에 따른 처리 코드

```
try{
        NullPointerException 발생
        NumberFormatException 발생
} catch (Exception e){
    // 예외 처리1

}catch (NullPointerException e){
    // 예외 처리2
}
```



예외 종류에 따른 처리 코드

다중 catch

- 예외 별로 예외 처리 코드 다르게 구현

```
try{
        ArithmeticException 발생
        InputMismatchException 발생
} catch (ArithmeticException e){
    // 예외 처리1
}catch (InputMismatchException e){
    // 예외 처리2
} catch (Exception e){
    // 예외 처리
}
```

예외 종류에 따른 처리 코드

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

```
try{
        ArithmeticException 또는 NullPointerException발생

        또 다른 Exception 발생

} catch (ArithmeticException | NullPointerException e){
    //예외 처리1
} catch (Exception e){
    // 예외 처리
}
```

중간 점검 문제

1. Error와 Exception은 어떻게 다른가?

2. 자바 코드에서 반드시 처리하여야 하는 예외는 어떤 것들인가?

3. RuntimeException을 처리했는지를 왜 컴파일러에서는 검사하지 않는가?

예외와 메소드

자바에서 예외를 처리하는 방법에는 한 가지가 더 있다.

- 예외를 잡아서 그 자리에서 처리하는 방법(기존) :
  try~catch 블록을 사용하여서 예외를 잡고 처리한다.
- 메소드 내부에서 처리하지 않고 메소드를 호출할 때 에러를 처리하도록 만드는 기법 :
  throws(에러 처리를 미룰 때 사용하는 키워드)를 사용하여, 다른 메서드한테 예외 처리를 맡긴다.

```
int method(int n) throws IOException {
        …
}
```

예외 떠 넘기기

- throws
  - 메소드 선언부 끝에 작성

```
리턴타입 메소드명(매개변수) throws 예외클래스1, 예외클래스2 {
    …
}
```

  - 예외 처리를 자신이 담당하지 않고 자신을 호출한 쪽으로 던져버리는 방법이다.
  - 메서드 내에서 예외가 발생하면 예외를 자신을 호출한 상위 메서드로 예외를 전달할 수 있다.
  - throws가 선언된 메서드는 메서드를 호출한 곳에서 throws 다음에 선언된 예외들을 처리하도록 위임한다.
  - throws는 메서드를 호출하는 곳의 상황에 맞게 동적인 오류 처리를 하기 위해 사용하는 명령문이다.

예외 떠 넘기기

- throws
  - 메소드에서 처리하지 않은 예외를 호출한 곳으로 떠넘기는 역할

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

public void method2()  throws ClassNotFoundException  {
  Class c1 = Class.forName("exam_class.Car");
}
```

호출한 곳에서 예외처리

예외와 메소드

예로 FileWriterEx.java의 writeList() 메소드 안에서 예외를 처리하지 않으면 컴파일 오류가 발생한다.

```
public void writeList(){
        FileWriter fw = new FileWriter("javafile.txt");
        fw.write("java");
        fw.close();
}
```

예외와 메소드

예로 FileWriterEx.java의 writeList() 메소드 안에서 예외를 처리하는 방법은 다음과 같다.

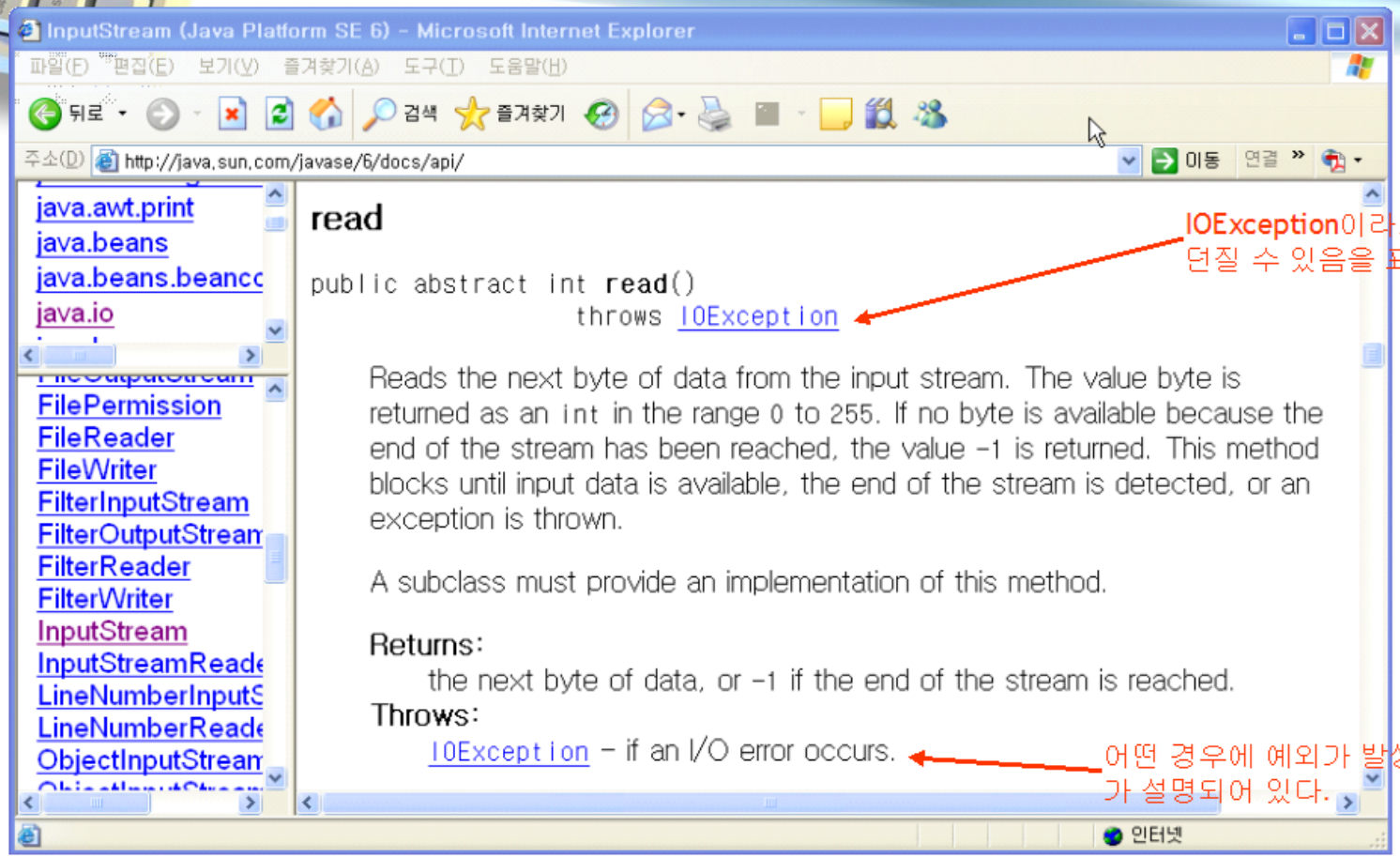
```
public void writeList() {
    FileWriter fw = null;
    try{
        fw = new FileWriter("javafile.txt");
        fw.write("java");
    }catch(IOException e){
        System.out.println("파일 저장에 문제가 있음.");
    }finally{
        if(fw!=null) fw.close();
    }
}
```

예외와 메소드

예로 FileWriterEx.java의 writeList() 메소드 안에서 예외를 처리하는 방법은 다음과 같다.

```
public void writeList() throws IOException {
        FileWriter fw = new FileWriter("javafile.txt");
        fw.write("java");
        fw.close();
}
```

예외가 발생하는 메소드



처리 방법

- 예외를 try/catch로 처리하는 방법
- 예외를 상위 메소드로 전달하는 방법

예외가 발생하는 메소드

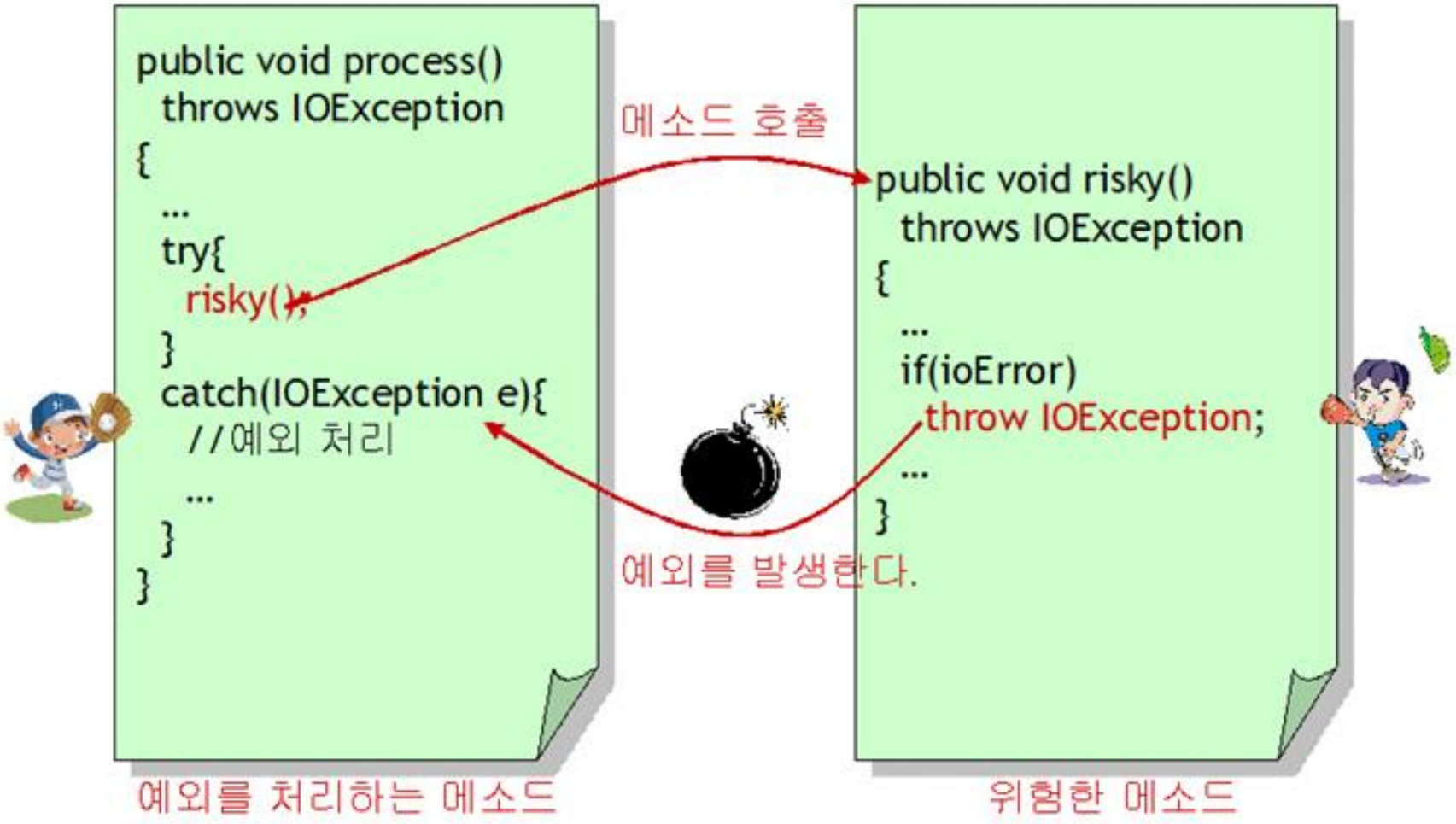
```
public class ExceptionMethod {
        public static void main(String[] args) {
                System.out.println(readString());  }
        public static String readString() {
             byte[] buf = new byte[100];
             System.out.println("문자열을 입력하시오:");
             System.in.read(buf);
             return new String(buf);   }
}
```

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
        Unhandled exception type IOException

        at ExceptionMethod.readString(ExceptionMethod.java:9)
        at ExceptionMethod.main(ExceptionMethod.java:4)
```

예외 생성하기

예외는 **throw** 문장을 이용하여 생성한다.



사용자 정의 예외와 예외 발생

1) 사용자 정의 예외 클래스 선언

- 자바 표준 API에서 제공하지 않는 예외

- 애플리케이션 서비스와 관련된 예외
  - Ex) 잔고 부족 예외, 계좌 이체 실패 예외, 회원 가입 실패 예외....

- 사용자 정의 예외 클래스 선언 방법

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

사용자 정의 예외와 예외 발생

❖ 예외 발생 시키기

- 코드에서 예외 발생시키는 법
- **throw은 예외 객체를 생성해서 강제로 던질 수 있는 예약어이다. 프로그래머가 의도적으로 예외를 발생시키고자 할 때 사용하는 예약어로 사용자 정의 예외를 만든 후 예외를 사용자가 발생시킬 때 주로 사용한다.**

호출된 곳에서 발생한 예외를 처리하도록

```
throw new 예외클래스명();
throw new 예외클래스명("출력 메시지");
```

```
public void method() throws 예외클래스명{
        throw new 예외클래스명();
}
```

예외 객체는 예외 상황이 발생하면 JVM이 자동으로 객체를 생성한다. 그러나 사용자 정의 예외 객체는 개발자가 필요해서 생성한 것으로 JVM이 예외 상황을 인지하여 자동으로 예외 객체를 생성할 수 없다.

사용자 정의 예외와 예외 발생

- **사용자 정의 예외 정의하기**

  사용자 정의 예외 클래스를 정의하려면 우선 Exception 클래스를 상속 받아야 한다.

```
class MyException extends Exception {
        public MyException(String msg){
                super(msg);
        }
}
```

사용자 정의 예외와 예외 발생

- **사용자 정의 예외 던지기**

  사용자 정의 예외 클래스를 정의하고 예외 발생시 해당 상황을 알 수 있도록 혹은 처리할 수 있도록 인수로 메시지를 받아 예외 객체를 생성한다

```
class MyScore {
        private int score;
        public void setScore (int score)
                                    throws MyException {
                if(score < 0){
                    throw new MyException("점수는
                                    음수가 될 수 없습니다");
                } else {
                    this.score = score;
        }
}
```

사용자 정의 예외와 예외 발생

❖ 사용자 정의 예외의 발생과 처리

```
class MyScoreTest {
        public static void main(String[] args){
            MyScore obj = new MyScore();
            try{
                    obj.setScore(-10);
            } catch(MyException e){
                    System.out.print(e.getMessage());
            }
        }
}
```

사용자 정의 예외와 예외 발생

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

```
throw new 예외클래스명("출력 메시지");
```

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

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

예외 생성하기

2) 연속적인 예외 발생

어떤 에플리케이션은 예외를 처리하면서 다른 예외를 발생시킨다. catch 블럭에서 예외를 처리하지 않고 다른 예외 처리기로 작업을 위임할 경우에는 예외를 다시 발생시켜서 전달할 수 있다.

```
try{
      ....
} catch (IOException e) {
    throw new UserDefException("다른 예외") ;
}
```

예외처리 전략

일반 프로그램의 패턴에서 예외 상황과 처리에 대해 정리한 표이다.

상황	처리	설명
데이터 추출	throw	데이터를 제공하는 곳에서 예외를 처리할 책임이 있다. 상위 클래스로 예외를 던진다.
유효성 검사		
처리	try ~ catch	상위 클래스에서 제공하는 데이터에서 이상이 없고, 자신이 처리하는 동안 발생한 예외이므로 자신이 처리할 책임이 있다. try 블록에서 발생한 예외를 catch 블록에서 처리한다.
결과 출력		

예외처리 전략

예외 처리 코드는 처음 절차에서부터 계획된 예외 처리 전략을 사용해야 한다.

① 예외 클래스를 정의하고, 언제, 어디서 던지고, 잡아서 처리할지를 결정한다.

② 비즈니스 로직에 전념하고 예외 처리가 필요한 곳에서 주석으로 예외 처리를 표시한다.

③ 예외 처리가 필요한 곳(주석 처리 부분)에 실제로 에러 처리 코드를 삽입한다.

예외 처리 코드를 사용하면 비즈니스 로직과 예외 처리 코드를 분리하는 장점이 있다.

중간 점검 문제

1. 사용자로부터 국어, 영어, 수학 점수를 입력 받아서 평균을 계산하는 프로그램을 작성하여 보자. 만약 사용자가 음수를 입력하면 NegativeNumberException(사용자 정의 예외 클래스)을 발생한다. 이 예외를 catch 블록으로 잡아서 처리하는 코드도 추가하라.

자동 리소스 닫기

- try-with-resources
  - 예외 발생 여부와 상관 없음
  - try-with-resources 문장은 문장의 끝에서 리소스들이 자동으로 닫혀지게 한다. 즉 사용했던 리소스 객체의 close() 메소드 호출해 리소스 닫음
  - try-with-resources 문장은 Java SE 7버전부터 추가되었다.
  - 리소스 객체
    - 각종 입출력스트림 등
    - java.lang.AutoCloseable 인터페이스 구현하고 있어야 함

```
try(리소스자료형1 참조변수 = new 리소스자료형1();
    리소스자료형2 참조변수 = new 리소스자료형2()){
}
```

try-with-resources문

1. try-with-resources문을 사용하면 문의 끝에 확실하게 자원을 자동으로 해제해 준다. 단 이렇게 처리되는 자원은 반드시 java.lang.AutoCloseable 클래스로 구현해야 한다.

2. 예외 객체는 예외 상황이 발생하면 JVM이 자동으로 객체를 생성한다. 그러나 사용자 정의 예외 객체는 개발자가 필요해서 생성한 것으로 JVM이 예외 상황을 인지하여 자동으로 예외 객체를 생성할 수 없다. 사용자 정의 예외 객체는 프로그램 내에서 예외 상황과 예외 객체 생성을 처리해주어야 한다. 예외 상황은 조건문으로 처리하고 예외 객체 생성은 throw 문으로 할 수 있다. 예외 상황을 강제로 발생하게 하는 throw문 사용법은 다음과 같다.


Thank You