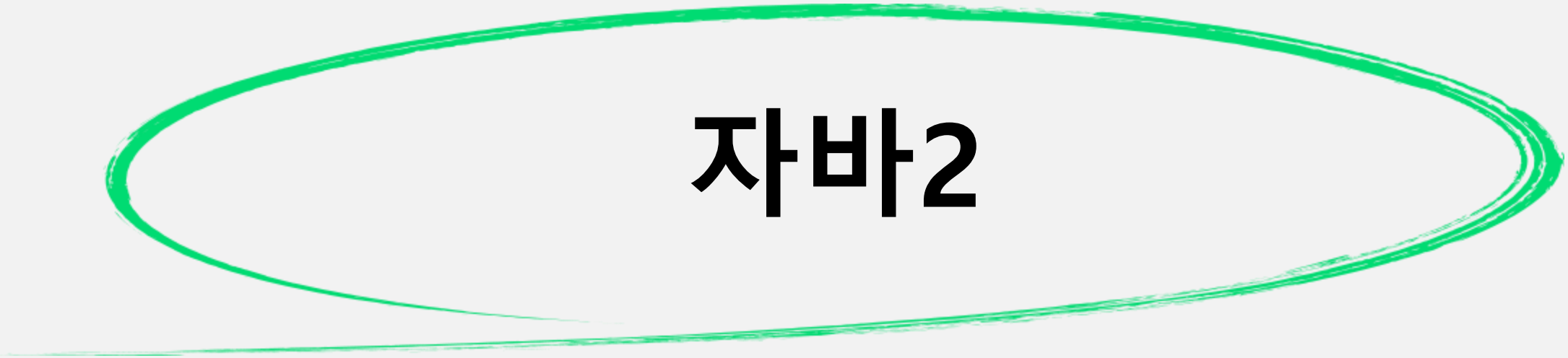


자바2

(예외 처리)

Chapter 01
예외 처리

Chapter01의 학습목표

- 예외 처리의 필요성과 사용법에 대해 학습한다.

예외 처리

예외와 오류

- 예외(Exception) : 프로그램에 존재하는 오류(error)를 의미
  개발자가 직접 처리할 수 있는 간단한 문제
- 에러(Error) :개발자가 처리할 수 없는 복잡한 문제

종류	설명	예
에러	개발자가 조취를 취할 수 없는 수준	메모리 부족, JVM 동작이상, 하드웨어 에러
컴파일 에러	컴파일 시에 발생하는 에러	오타, 잘못된 자료형
런타임 에러	프로그램이 실행하는 도중에 발생하는 에러	프로그래밍 버그
로직 에러	실행은 되지만 의도와는 다르게 동작하는 에러	-
예외	다른 방식으로 처리 가능한 오류	입력 값 오류

예외 처리

예외 처리(Exception Handling)

- 예외가 발생했을 때 이를 적절히 처리하여
  프로그램이 비정상적으로 종료되는 것을 막는 방법

예) 2/0 (0으로 나누기를 시도)

예외 처리의 목적

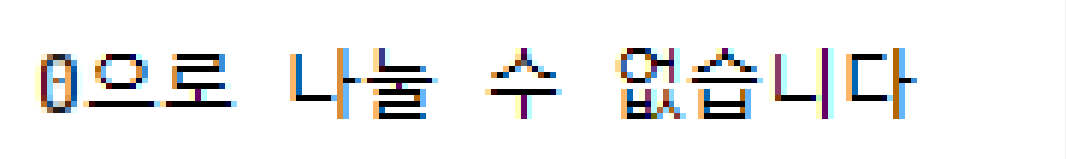
발생한 문제로 인해
프로그램이 비정상적으로 종료되는 것을 막고,
사용자에게 발생한 문제에 대한 정보를 전달하기 위함

1) 예외 처리를 하지 않았을 경우

```
int a = 0;
int b = 2;
int c = b/a;
```

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

2) 예외 처리를 했을 경우



예외 처리

고전적인 예외 처리

고전적인 예외 처리의 문제점:

1) 어떤 문제가 발생할 지 예상할 수 있어야 대비 가능
2) 어떤 문제가 발생할지 예상해도 대비할 수 없음
3) 프로그램의 규모가 커질수록 모든 문제를 대비하기 힘듦

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

System.out.print("제수를 입력하세요 >>> ");
int a = input.nextInt();
System.out.print("피제수를 입력하세요 >>> ");
int b = input.nextInt();

if (b == 0) {
    System.out.println("0으로 나누는 것은 불가능합니다.");
}
else {
    System.out.println(a + "/" + b + " = " + (a/b));
}
```

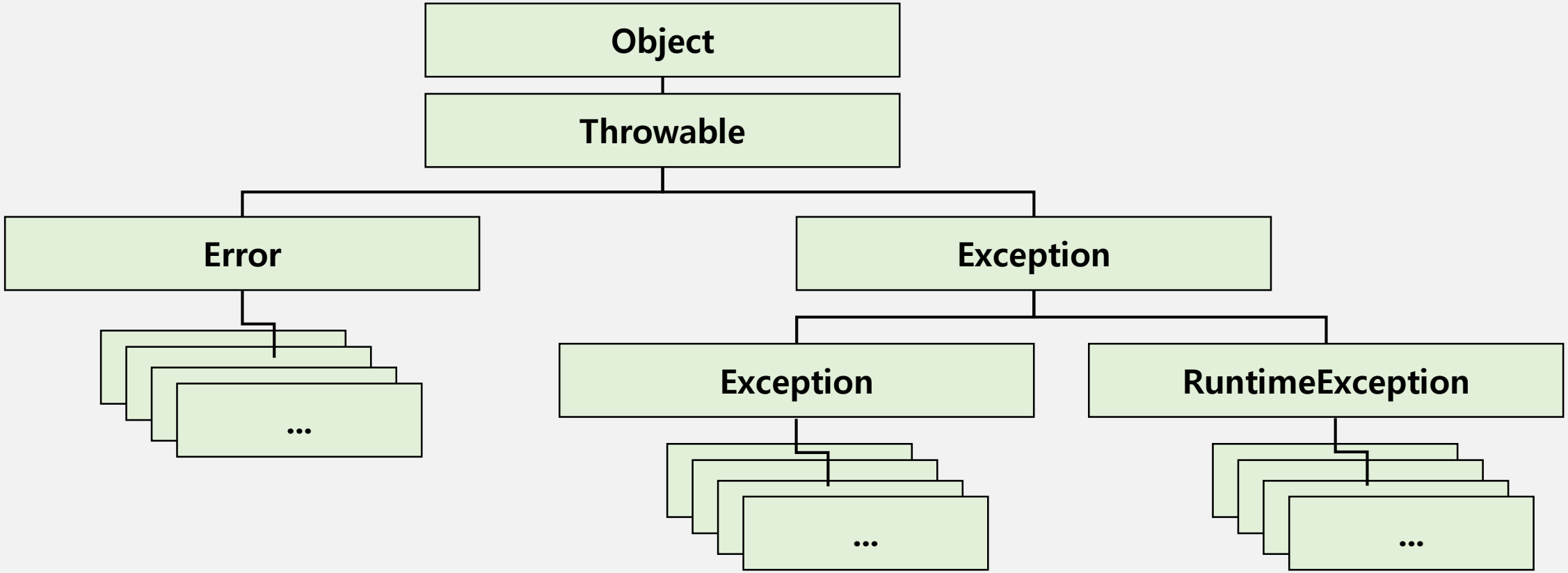
정수가 아닌 값을 입력(실수나 문자 등)하면 문제가 발생

프로그램의 규모가 커질수록 작성할 코드가 길어짐

예외 처리

자바에서의 예외 처리

- 자바에서의 예외는 Exception이라는 클래스 객체로 동작
- 예외 클래스들도 서로 상속의 관계로 묶여져 있음
- 문제가 발생할 시 자동으로 해당 문제의 예외 인스턴스가 발생



예외 처리

런타임 예외의 종류

- 프로그래밍 버그 혹은 논리 오류
- Error + RuntimeException = Unchecked exception

분류	예외	설명
Runtime Exception	ArthmeticException	어떤 수를 0으로 나눌 경우
	NullPointerException	널 객체를 참조할 경우
	ClassCastException	적절치 못하게 클래스를 형변환하는 경우
	NegativeArraySizeException	배열의 크기가 음수 값인 경우
	OutOfMemoryException	사용 가능한 메모리가 없는 경우
	NoClassDefFoundException	원하는 클래스를 찾지 못하였을 경우
	ArrayIndexOutOfBoundsException	배열을 참조하는 인덱스가 잘못된 경우

예외 처리

예외 처리 방식 : try-catch

1) 모든 예외를 처리하는 방식

```
try{
    코드 작성 영역}
catch(Exception e){
    예외 발생 시 처리 영역}}
```

```java
try {
    System.out.print("제수를 입력하세요 >>> ");
    int a = input.nextInt();
    System.out.print("피제수를 입력하세요 >>> ");
    int b = input.nextInt();
    System.out.println(a/b);
}
catch (Exception e) {
    // TODO: handle exception
    System.out.println("예외가 발생했습니다.");
}
```

- 0으로 나눌 때

```
제수를 입력하세요 >>> 5
피제수를 입력하세요 >>> 0
예외가 발생했습니다.
```

- 정수가 아닌 값 입력

```
제수를 입력하세요 >>> test
예외가 발생했습니다.
```

- 정상 값 입력

```
제수를 입력하세요 >>> 10
피제수를 입력하세요 >>> 5
2
```

예외 처리

예외 처리 방식 : try-catch

2) 특정 예외만 처리하는 방식

1. 0으로 나누려고 하는 경우 → 0으로 나눌 수 없습니다.
2. 정수가 아닌 값을 입력할 경우 → 정수만 입력할 수 있습니다.

```
try {
    System.out.print("제수를 입력하세요 >>> ");
    int a = input.nextInt();
    System.out.print("피제수를 입력하세요 >>> ");
    int b = input.nextInt();
    System.out.println(a/b);
}
catch (ArithmeticException e) {
    System.out.println("0으로 나눌 수 없습니다.");
}
catch (InputMismatchException e) {
    System.out.println("정수만 입력이 가능합니다.");
}
```

- 예외는 순서대로 except처리

```
try{
    코드 작성 영역}
catch(예외1){
    예외1 발생 시 처리 영역}
catch(예외2){
    예외1이 발생하지 않았고,
    예외2 발생 시 처리 영역}
```

예외 처리

예외 처리 방식 : try-catch-finally

- 오류가 발생하든 하지 않든 간에 무조건 실행되는 구문

```
try{
    코드 작성 영역}
catch(예외1){
    예외1 발생 시 처리 영역}
catch(예외2){
    예외1이 발생하지 않았고,
    예외2 발생 시 처리 영역}
finally{
    예외 여부와 관계없이 실행되는 영역}
```

```java
Scanner input = new Scanner(System.in);

try {
    System.out.print("제수를 입력하세요 >>> ");
    int a = input.nextInt();
    System.out.print("피제수를 입력하세요 >>> ");
    int b = input.nextInt();
    System.out.println(a/b);
}
catch (ArithmeticException e) {
    System.out.println("0으로 나눌 수 없습니다.");
}
catch (InputMismatchException e) {
    System.out.println("정수만 입력이 가능합니다.");
}
finally {
    System.out.println("시스템 종료");
}
```

예외 처리

예외 처리 방식

3) 예외 메시지 처리하기

- catch문의 객체 참조 변수 e를 통해 예외 메시지를 받아 오기
- System.err.println(); 구문을 통해 에러가 났을 경우 에러 메세지 출력하기

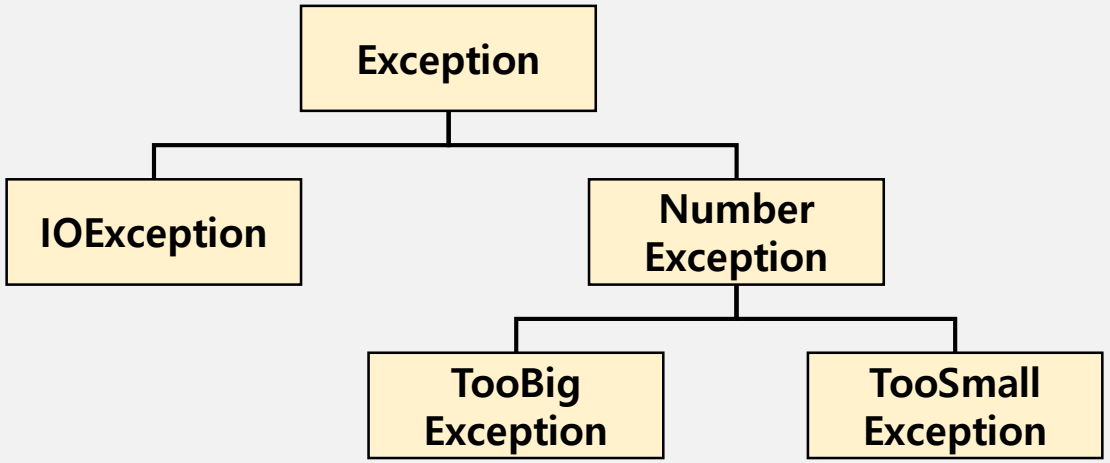
```
catch (ArithmeticException e) {
    System.out.println(e);
}
catch (InputMismatchException e) {
    System.out.println(e);
}
```

```
catch (Exception e) {
    System.out.println(e);
    System.err.println();
}
```

예외 처리

다형성과 예외

- 다형성의 원칙에 따라 상위 클래스의 참조 변수는 하위 클래스의 객체를 참조할 수 있음



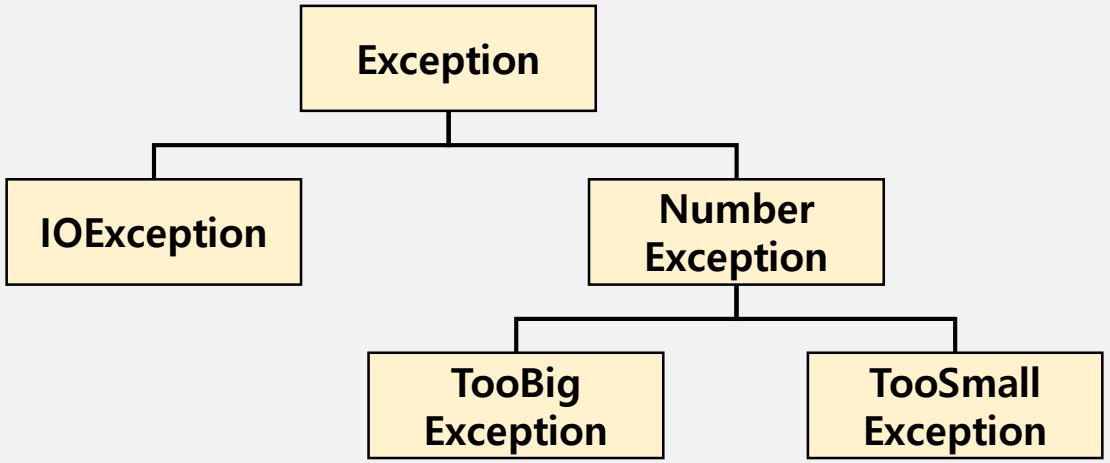

```
try{
    getInput(); //예외를 발생하는 메소드
}
catch(NumberException e){
    //NumberException의 하위 클래스를 모두 잡을 수 있음
}
```

```
try{
    getInput(); //예외를 발생하는 메소드
}
catch(Exception e){
    //Exception의 하위 클래스를 모두 잡을 수 있으나
    예외를 분간할 수 없음
}
```

예외 처리

다형성과 예외

- 다형성의 원칙에 따라 상위 클래스의 참조 변수는 하위 클래스의 객체를 참조할 수 있음




```
try{
    getInput(); //예외를 발생하는 메소드
}
catch(NumberException e){
    //모든 NumberException이 잡힌다
}
catch(TooSmallException e){
    //아무것도 잡히지 않는다
}
```

예외 처리

강제로 예외 발생시키기 (throw)

- 자바에서는 예외로 인식하지 못하지만 프로그램 구현에서 예외가 발생할 때 사용
  ex) 어떤 사람의 나이를 입력 받을 때 : -1000
- 예외 클래스의 인스턴스를 생성한 다음 throw 키워드를 사용하여 예외를 발생시킴
- 예외 메시지는 Exception 클래스의 생성자에 전달

Exception e = new Exception(“Exception”);
throw e; //예외 발생

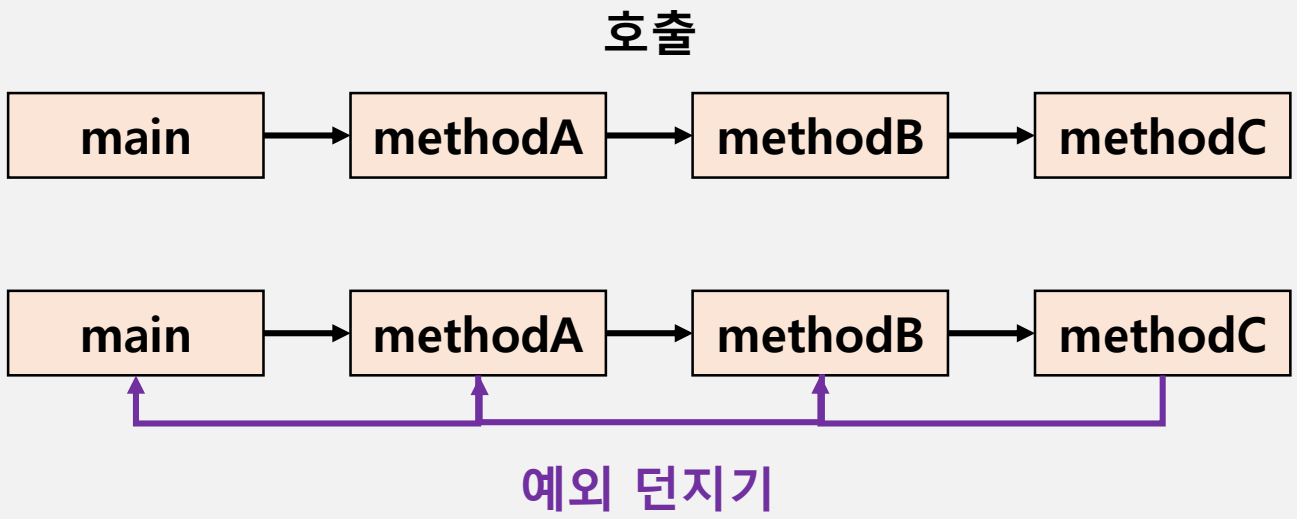
```
try {
    Exception e = new Exception("고의 예외");
    throw e;
}
catch (Exception e) {
    System.out.println("예외 발생");
    System.out.println(e);
    System.out.println(e.getMessage());
}
```

예외 처리

예외 던지기 (throws)

- 예외가 발생했을 경우 현재 메소드가 예외를 처리하지 않고 자신을 호출한 쪽으로 예외 처리에 대한 책임을 넘기는 것
- 예외를 넘겨받은 쪽은 반드시 직접 예외처리를 하거나 자신도 예외를 던져야 함

```
void method() throws Exception{
}
```




```java
public class Test{
    public static void methodA() throws Exception{
        methodB();
    }
    public static void methodB() throws Exception{
        methodC();
    }
    public static void methodC() throws Exception{
        Exception e = new Exception();
        throw e;
    }

    public static void main(String[] args) {
        int age = -19;

        try {
            methodA();
        }
        catch (Exception e) {
            System.out.println("메인에서 처리");
        }
    }
}
```

예외 처리

사용자 예외 클래스

- 예외 클래스를 직접 만들어서 사용
- Exception 클래스를 상속받은 클래스를 새로 생성
- Exception 클래스의 생성자에 message를 전달

```
class 예외 클래스 extends Exception{
    public 예외클래스(){}
    public 예외클래스(String message){
        super()('예외 메시지');}
}
```

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

public class Test{
    public static void ticketing(int age) throws AgeException {
        if(age < 0) {
            throw new AgeException("나이 입력이 잘못되었습니다.");
        }
    }

    public static void main(String[] args) {
        int age = 19;

        try {
            ticketing(age);
        }
        catch (AgeException e) {
            e.printStackTrace();
        }
    }
}
```

예외처리 예제

입력받은 이름이 2~6자 사이가 아니면 NameError 예외를 발생시키고 '이름은 2~6자 사이로 입력해주세요'라는 예외 메시지를 출력하는 프로그램을 구현

```java
import java.util.Scanner;

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

```java
public class Test{
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        try {
            System.out.print("이름을 입력하세요 >>> ");
            String name = input.next();
            if(name.length() < 2 || name.length() > 6) {
                NameError e = new NameError("이름은 2-6자 사이로 입력해주세요.");
                throw e;
            }
            else {
                System.out.println("입력된 이름은 " + name + "입니다.");
            }
        }
        catch (NameError e) {
            System.out.println(e);
        }

    }
}
```

수고하셨습니다.