

# 제10장 예외 처리

10.1 예외(Exceptions)

10.2 Python 예외

10.3 Java 예외

10.4 예외 검사

## 10.1 예외 Exceptions

# 예외 Exception (외수)

- 예외 (상황)
  - 심각하지 않은 오류 혹은 비정상적 상황
  - 예외 발생 시 계속 수행할 수 있도록 처리해야 한다
  - 발생한 예외를 처리하지 못하면 프로그램 종료
- 예외에 대한 적절한 처리
  - 안전한 프로그램 실행을 위해 매우 중요하다.
  - 따라서 최신 언어들은 예외 관련 기능을 제공한다.
  - Java, C++, Python, ML, ...
- 필요한 예외 관련 구문
  - 예외 정의
  - 예외 발생
  - 예외 처리

# 예외 처리 모델

```
try {  
    raise E;    (1)    // 예외 발생  
    (2)  
} catch (E) {  
    (3)    // 예외 처리  
}  
(4)
```

## ① ● 재개 모델(resumption model)

- 예외가 발생하면 예외 처리 후 예외를 발생시킨 코드로 재개하여 계속 실행
- 즉 (1) 지점에서 예외가 발생했을 때 (1) -> (3) -> (2) 순으로 계속 실행된다.

## ② ● 종료 모델(termination model) → java

- 예외가 발생하면 예외 처리 후 예외를 발생시킨 코드로 재개하지 않고
- try 문을 끝내고 다음 문장을 실행한다. 즉 (1) -> (3) -> (4) 순으로 실행.

# 언어 S의 예외

- 언어 S에 예외 추가

`<command> → ... | exc id;` // 예외 정의  
`<stmt> → ...`  
`| raise id;` // 예외 발생  
`| try <stmt> catch(id) <stmt>` // 예외 처리

- 예외 정의

- 새로운 이름의 예외를 정의한다.

- 예외 발생

- 예외를 발생시킨다.

- 예외 처리

- try 블록에서 발생한 예외가 id 예외이면 이를 처리한다.
- 발생한 예외를 처리하지 못하면 프로그램은 종료한다.
- 예외가 발생하지 않으면 다음 문장을 실행한다.

## [예제 1]

---

```
exc InvalidInput;
let
  int x=0; int y=1;
in
  read x;
  if (x < 0) then
    raise InvalidInput;
  else
    while (x != 0) {
      y = y * x;
      x = x - 1;
    }
  print y;
end;
```

## [예제 2]

**exc** InvalidInput;

let

int x=0; int y=1;

in

read x;

**try** {

if (x < 0) then

**raise** InvalidInput;

else

while (x != 0) {

y = y \* x;

x = x - 1;

}


print y;

} **catch**(InvalidInput) print "invalid input";

end;

# 여러 개의 catch 절

- 예외의 종류에 따라 다르게 처리함.



```
try
  S1
catch (E1) S2
catch (E2) S3
```

*syntactic sugar*

- 이 문장은 다음과 같이 try 문을 중첩하여 작성할 수 있음

```
try
  try
    S1
  catch (E1) S2
catch (E2) S3
```



## 10.2 Python 예외

# 예외 발생

---

```
>>> x = 0
```

```
>>> print(10/x)
```

Traceback (most recent call last):

File "<pyshell#2>", line 1, in <module>

print(10/x)

ZeroDivisionError: division by zero

예외와 재발하지 않음

# 예외 처리

---

- 예외 처리

`try:`  
실행 코드  
`except:`  
예외 처리 코드

- 예제

```
try:  
    x = int(input('나눌 숫자를 입력하세요: '))  
    y = 10 / x  
    print(y)  
except: # 예외가 발생했을 때 실행됨  
    print('예외가 발생했습니다.')
```

실행 결과  
나눌 숫자를 입력하세요: 0  
예외가 발생했습니다

# 특정 예외 처리

- 구문

`try:`

실행 코드

`except` 예외이름:

예외 처리 코드

...

`except` 예외이름:

예외 처리 코드

`[finally:`


예외 발생 여부와 관계없이 실행할 코드]

- 예외에 따라 처리하는 코드 다름
- `finally` 절은 옵션으로 예외의 발생 및 처리 여부와 관계없이 마지막으로 실행된다.

# 특정 예외 처리

- 예제

```
try:
    x = int(input('나눌 숫자를 입력하세요: '))
    print(10 / x)
except ZeroDivisionError: # 숫자를 0으로 나눌 때 실행됨
    print('숫자를 0으로 나눌 수 없습니다.')
```



- 실행결과

나눌 숫자를 입력하세요: 0  
숫자를 0으로 나눌 수 없습니다.

- 실행결과

나눌 숫자를 입력하세요: 10.0  
Traceback (most recent call last):

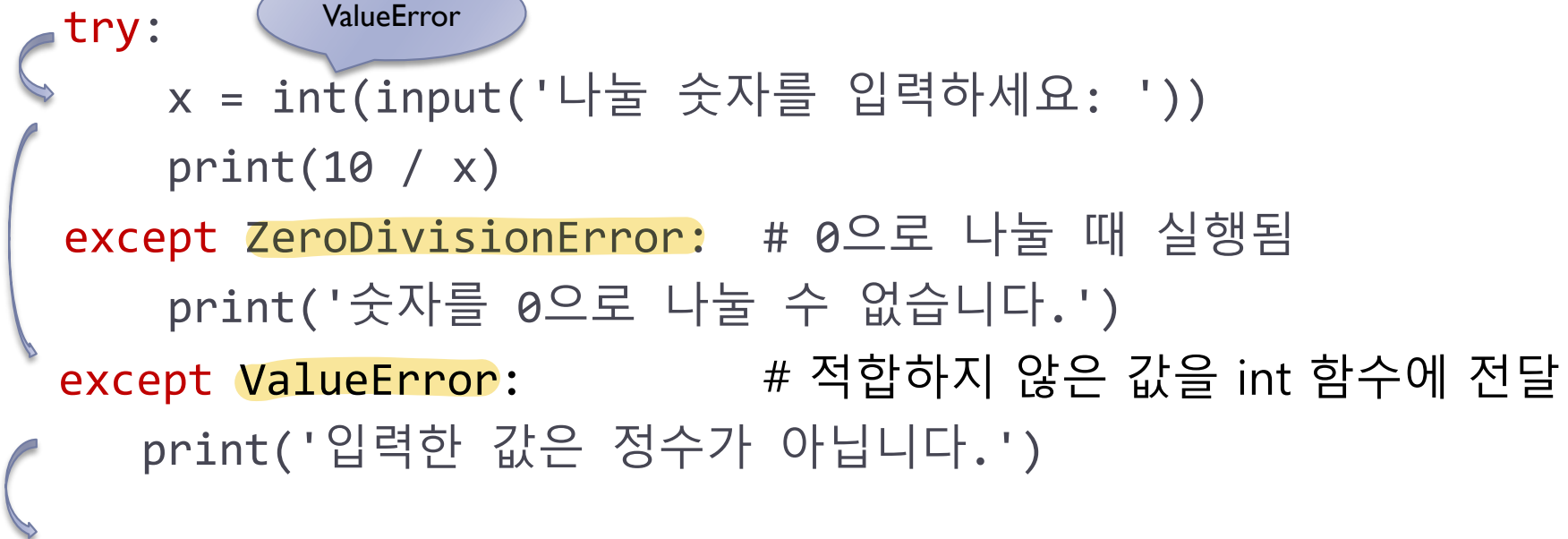
File "<pyshell#15>", line 2, in <module>  
x = int(input('나눌 숫자를 입력하세요: '))

① ValueError: invalid literal for int() with base 10: '10.0'  
여기도 처리해야함!

# 여러 개의 except 절

- 예제

```
try:
    x = int(input('나눌 숫자를 입력하세요: '))
    print(10 / x)
except ZeroDivisionError: # 0으로 나눌 때 실행됨
    print('숫자를 0으로 나눌 수 없습니다.')
except ValueError:        # 적합하지 않은 값을 int 함수에 전달
    print('입력한 값은 정수가 아닙니다.')
```



- 실행결과

나눌 숫자를 입력하세요: 10.0  
입력한 값은 정수가 아닙니다.

# try 문 중첩

---

- 예제

```
try:
```

```
    try:
```

```
        x = int(input('나눌 숫자를 입력하세요: '))
```

```
        print(10 / x)
```

```
    except ZeroDivisionError: # 0으로 나눌 때 실행
```

```
        print('숫자를 0으로 나눌 수 없습니다.')
```

```
except ValueError: # 적합하지 않은 값을 int 함수에 전달
```

```
    print('입력한 값은 정수가 아닙니다.')
```

# try-except-as

- 예외의 오류 메시지 받아오기

try:

```
x = int(input('나눌 숫자를 입력하세요: '))  
print(10 / x)
```

except ZeroDivisionError as e:

```
print('숫자를 0으로 나눌 수 없습니다: ', e)
```

except ValueError as e:

```
print('입력 값은 정수가 아닙니다: ', e)
```

변수 e에 오류 메시지 받음

- 실행 결과

나눌 숫자를 입력하세요: 0

숫자를 0으로 나눌 수 없습니다: division by zero

- 실행 결과

나눌 숫자를 입력하세요: 10.0

입력 값은 정수가 아닙니다 : invalid literal for int() with base 10: '10.0'



# try-finally 예제

```
y = [10, 20, 30]
```

```
try:
```

```
    index = int(input('인덱스를 입력하세요: '))
```

```
    x = int(input('나눌 숫자를 입력하세요: '))
```

```
    print(y[index] / x)
```

```
except ZeroDivisionError as e:
```

```
    print('숫자를 0으로 나눌 수 없습니다: ', e)
```

```
except IndexError as e:
```

```
    print('잘못된 인덱스입니다: ', e)
```

```
except ValueError as e:
```

```
    print('입력한 값은 정수가 아닙니다: ', e)
```

```
finally:
```

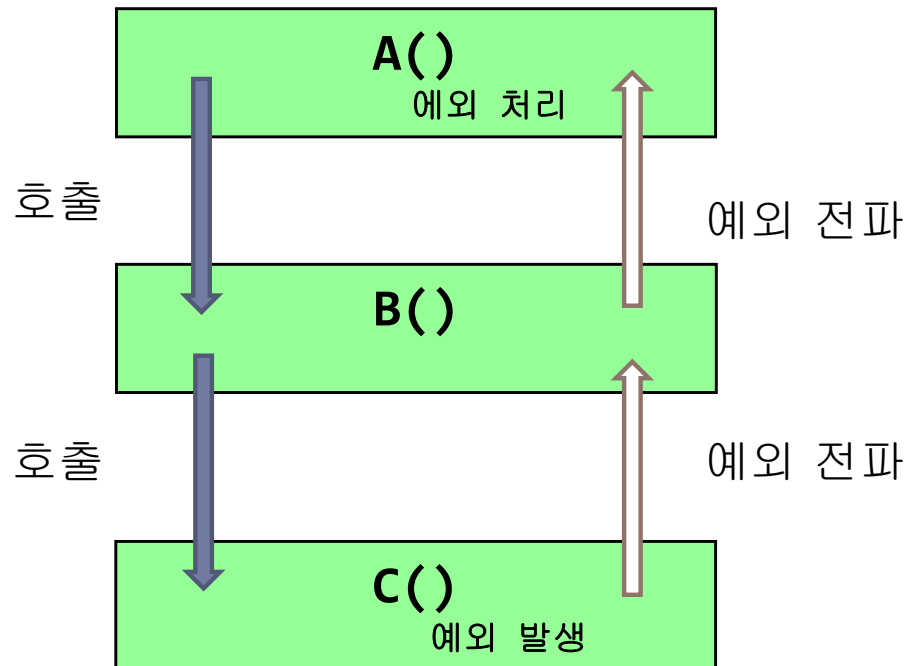
```
    print('try 문의 끝입니다.')
```

# 예외 전파

- 예외 전파(exception propagation)

- 호출된 함수 내에서 발생한 예외는 그 함수 내에서 처리되지 않으면 호출의 역순으로 처리될 때까지 호출자 함수에 전파된다.

반드시 그 함수내에서 처리되도록  
잘하도록!



## [예제 9]

---

```
def C(x):  
    return 8 / x      # x가 0인 경우 오류 발생
```

```
def B(y):  
    return C(y - 1)   # y가 1인 경우 오류 발생
```

```
def A( ):  
    print(B(int(input())))
```

```
A( )
```

- 실행결과

1

```
Traceback (most recent call last):  
File "<pyshell#6>", line 1, in <module>  
A()  
File "<pyshell#5>", line 2, in c  
print(B(int(input())))  
File "<pyshell#3>", line 2, in b  
return C(y - 1)  
File "<pyshell#1>", line 2, in a  
return 8 / x  
ZeroDivisionError: division by zero
```

## [예제 10]

---

```
def C(x):  
    return 8 / x # x가 0인 경우 오류 발생  
def B(y):  
    return C(y - 1) # y가 1인 경우 오류 발생  
def A( ):  
    try:  
        print(B(int(input())))  
    except ZeroDivisionError:  
        print('0으로는 나눌 수 없습니다.')  
A( )
```

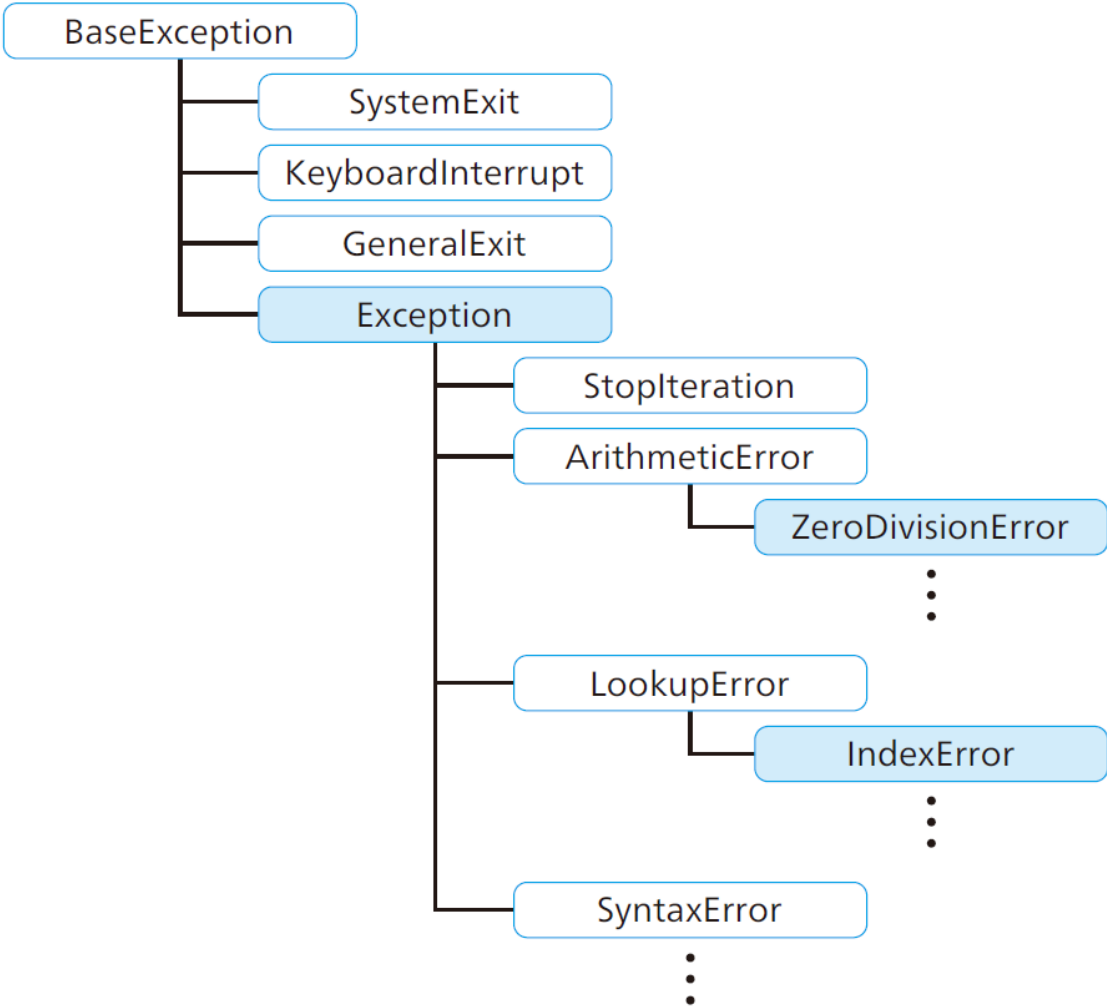
- 실행결과

1

0으로는 나눌 수 없습니다.

---

## 예외 계층 구조



### 그림 10.2 예외 계층 구조

# 포괄적 예외 처리

- 포괄적 예외 처리

- try

- ...

- except `BaseException`:

- try

- ...

- except:

- try

- ...

- except `Exception`:

- 특정 예외에 맞는 적절한 처리를 하기 힘들다.

# 예외 발생

- 예외 발생

`raise` 예외이름 또는 예외이름(메시지)  
예외설명

- 예

```
>>> raise ZeroDivisionError('0으로 나눌 수 없음')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: 0으로 나눌 수 없음
```

# 예외 정의

상속  
↓

```
class NegativeInputException(Exception):
```

```
    pass → 이쯤만 정의
```

```
def input_total( ):
```

```
    try:
```

```
        total = 0
```

```
        while True:
```

```
            score = int(input( ))
```

```
            if score < 0:
```

```
                raise NegativeInputException
```

```
            total = total + score
```

```
except NegativeInputException as e:
```

```
    print(e)
```

```
    return total
```

실행결과

input\_total()

10

20

30

-1

60



# 오류 메시지 만들기

- 메시지와 함께 예외를 생성하여 예외를 발생시킴  
`raise NegativeInputException("음수 입력입니다.")`
- 예외 클래스에 `__str__` 메소드를 구현
  - `__str__` 메서드는 `print(e)`처럼 오류 메시지를 출력할 때 호출되는 메소드

```
class NegativeInputException(Exception):  
    def __str__(self):  
        return "음수 입력입니다."
```

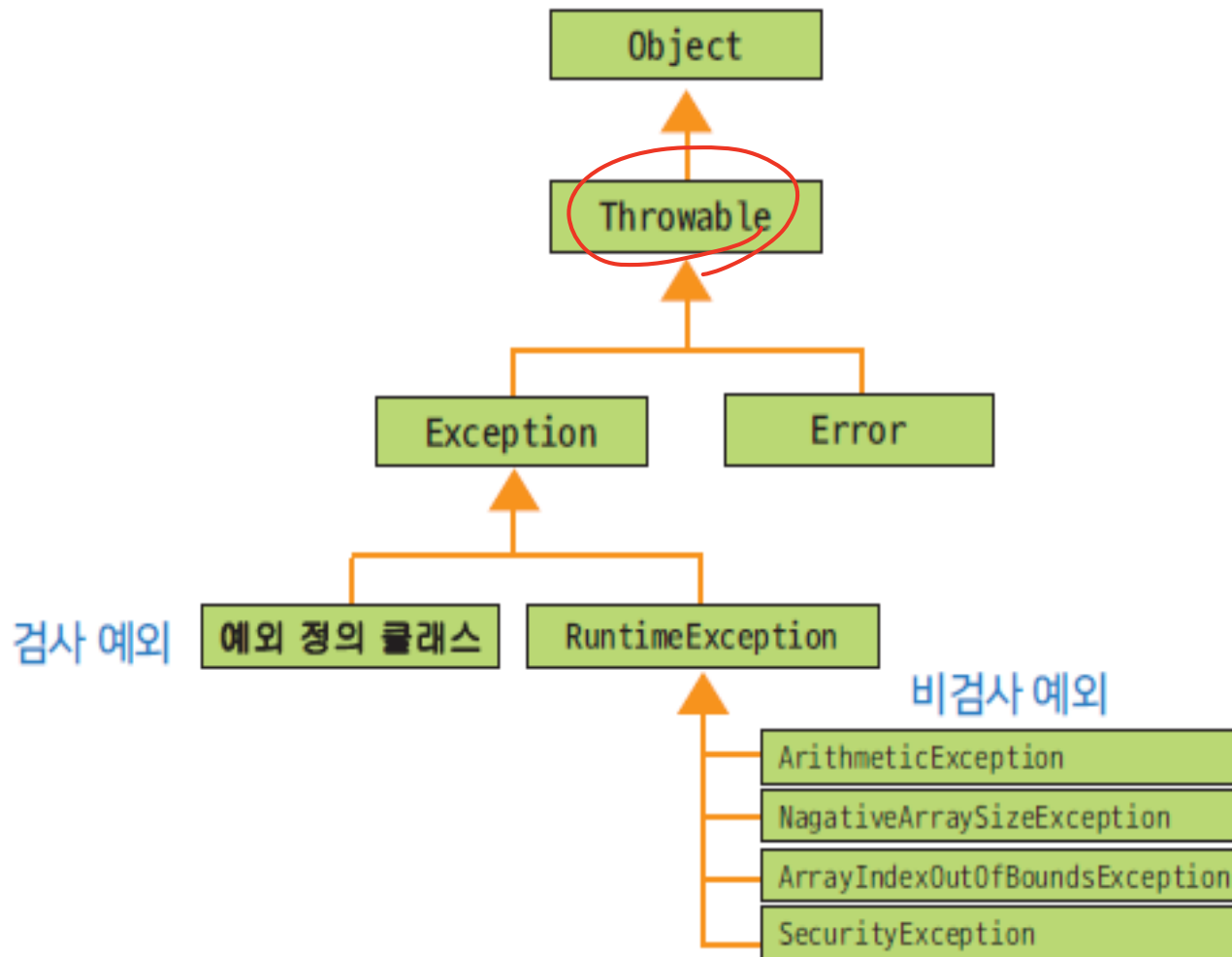
## 10.3 Java 예외

# Java에서 예외 선언

---

- 자바에서 예외 타입은 클래스로 선언
  - Exception 클래스나 서브클래스로부터 상속 받아 정의한다.
- 예외타입은 클래스이다
  - 생성자, 멤버 필드, 메소드 등을 가질 수 있다.
- Exception object is a first-class object
  - 일반 object처럼 클래스를 이용하여 정의되고
  - 일반 object처럼 사용될 수 있다.
  - 일반 object와 차이점 : throw 될 수 있다.

# Java 예외 클래스 계층구조



# Java 예외 클래스 예

---

- **ArithmeticException**
  - 0으로 나누는 경우에 주로 발생하는 예외 상황
  - RuntimeException으로부터 상속받은 예외 클래스
- **ArrayIndexOutOfBoundsException**
  - 배열의 크기보다 큰 원소를 접근하려고 할 때 발생하는 예외
- **NegativeArraySizeException**
  - 배열의 크기가 음수로 된 경우에 발생하는 예외
- **NullPointerException**
  - 생성되지 않은 객체를 이용해서 객체의 멤버를 접근하는 경우에 발생하는 예외
- ...

# 사용자 정의 예외 클래스: 예

---

```
class NegativeInputException extends Exception {  
    private String reason="Negative input";  
    NegativeInputException( ) {  
        System.out.println(reason + "is received");  
    }  
}
```

# 예외 발생

---

- 예외는 throw문을 이용해서 발생시킨다
  - **throw** 예외 객체;
  - **throws**와는 다름
- 예외 발생 시 처리하지 않는 경우
  - 프로그램은 메시지를 내고 종료한다.
  - 메시지는 호출 스택 트레이스(*call stack trace*)를 포함한다.
    - main 메소드부터 호출과정

# 예외 처리: try-catch 문

- try-catch 문

- 예외가 발생할 수 있는 문장들을 try문 블록에 기술한다.
- 예외가 발생하면 해당 매칭되는 catch 문으로 제어가 넘어 간다.

- 구문 구조

```
try {
```



예외

실행 코드

```
}
```

```
catch (E1 x) { 예외 처리 코드 }
```

...

```
catch (En x) { 예외 처리 코드 }
```

```
[finally { 예외 발생과 관계없이 실행할 코드 }]
```



## [예제 12]

```
class Grade {  
    int newGrade, total;  
    void totalGrade( ) {  
        Scanner scan = new Scanner (System.in);  
        try {  
            while (true) {  
                System.out.println("Please input a grade");  
                newGrade = scan.nextInt( );  
                if (newGrade < 0)  
                    throw new NegativeInputException( );  
                if (newGrade <= 100)  
                    total = total + newGrade;  
                else System.out.println("Out of range input !");  
            }  
        } catch(NegativeInputException x) {  
            System.out.println(x);  
            System.out.println("Total:" + total);  
        }  
    }  
}
```

## [예제 13]

---

```
public static void main(String[] args) {  
    Grade grade = new Grade();  
    grade.totalGrade();  
}
```

### [실행결과]

Please input a grade

85

Please input a grade

90

Please input a grade

75

Please input a grade

800

Out of range input !

Please input a grade

80

Please input a grade

-1

Negative input is received

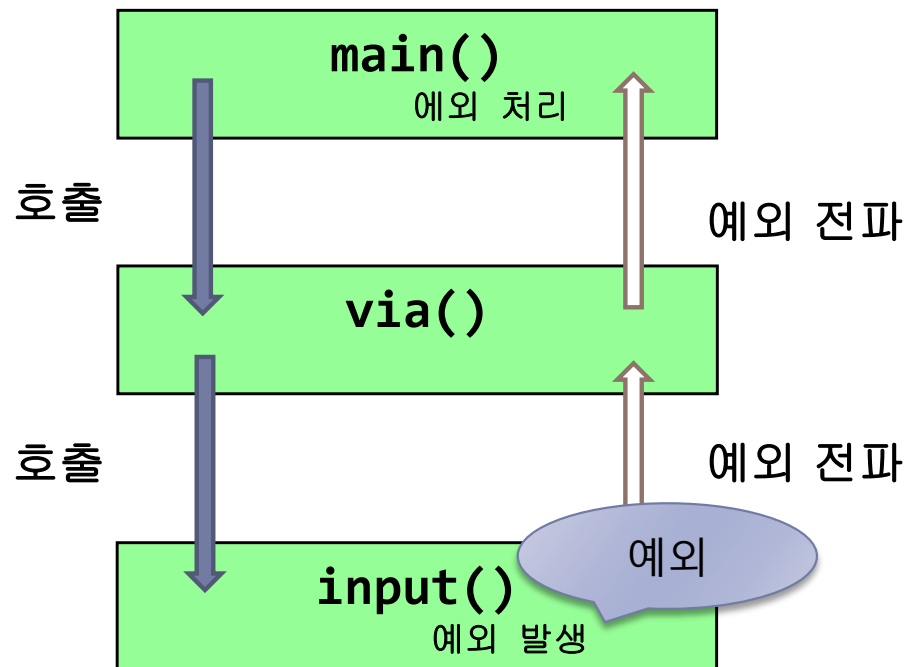
NegativeInputException

Total: 330

# 예외 전파 Exception Propagation

- 처리되지 않는 예외 전파
  - 일차적으로 호출자에게 전파된다.
  - 호출의 역순으로 처리될 때까지 전파된다.
  - 처리되지 않으면 main() 메소드까지 전파되고
  - main()에서도 처리되지 않으면 프로그램 종료

● 예



// 발생한 예외의 전파과정을 보인다.

```
public class Propagate
```

```
{
```

```
    C void input() throws NegativeInputException {
```

```
        int age;
```

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

```
        System.out.println("input 메소드 시작");
```

```
        age = scan.nextInt( );
```

```
        if (age < 0)
```

```
            throw new NegativeInputException( );
```

```
        System.out.println("input 메소드 끝");
```

```
    }
```

```
    B void via() throws NegativeInputException {
```

```
        System.out.println("via 메소드 시작");
```

```
        input();
```

```
        System.out.println("via 메소드 끝");
```

```
    }
```

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

```
        Propagate p = new Propagate();
```

```
        System.out.println("main 메소드 시작");
```

```
        try {
```

```
            p.via();
```

```
        } catch (NegativeInputException m) {
```

```
            System.out.println(m);
```

```
        }
```

```
        System.out.println("main 메소드 끝");
```

```
    }
```

```
}
```

## 10.4 예외 검사

# 예외 검사의 필요성

---

## [예제 프로그램 6]

- NegativeInputException이 발생할 수 있지만 이를 처리할 수 있는 try-catch 문이 없다.
- 따라서 이 메소드는 실행되면 오류가 발생할 것이다.

```
void input() {  
    int age;  
    Scanner scan = new Scanner(System.in);  
    System.out.println("input 메소드 시작");  
    age = scan.nextInt( );  
    if (age < 0)  
        throw new NegativeInputException( );  
    System.out.println("input 메소드 끝");  
}
```

# 컴파일러의 예외 검사

---

- 예외가 발생하면 처리할 수 있는지 컴파일 시간에 미리 검사
  - 발생한 예외를 처리할 수 있는 try-catch 문이 없으면
  - 실제 실행에서 예외가 발생하면 프로그램은 갑자기 종료한다.
- 이를 미리 예방하기 위해서 컴파일-시간 예외 검사
  - 발생 가능한 예외가 메소드 내에서 try-catch 문으로 처리될 수 있는지
  - 아니면 메소드 헤더에 throws로 선언되었는지 컴파일-시간에 검사
  - 오류 메시지

OOOException must be caught or declared to be thrown.

# 예외 명세

---

- 발생한 예외를 처리할 수 있는 try-catch 문을 추가하든지
- 아니면 처리되지 않은 검사 예외들을 메소드 이름 뒤에 명세
- throws 절 이용

```
메소드이름(...) throws E1, E2, E3
{
    . . .
}
```



## [예제 16]

// 발생한 예외의 전파과정을 보인다.

```
public class Propagate
```

```
{
```

```
    void input() throws NegativeInputException {
```

```
        int age;
```

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

```
        System.out.println("input 메소드 시작");
```

```
        age = scan.nextInt( );
```

```
        if (age < 0)
```

```
            throw new NegativeInputException( );
```

```
        System.out.println("input 메소드 끝");
```

```
    }
```

```
    void via() throws NegativeInputException {
```

```
        System.out.println("via 메소드 시작");
```

```
        input();
```

```
        System.out.println("via 메소드 끝");
```

```
    }
```

처음은 안하긴 선언했으니  
아무것도 안주나!

## [예제 16]

---

```
public static void main(String[] args) {  
    Propagate p = new Propagate();  
    System.out.println("main 메소드 시작");  
    try {  
        p.via(); ✓  
    } catch (NegativeInputException m) {  
        System.out.println(m);  
    }  
    System.out.println("main 메소드 끝");  
}
```

# 예외 검사

---

- Java 컴파일러의 예외 검사

- 메소드 내에서 발생 가능한 예외가 처리될 수 있는지
- 아니면 메소드 헤더에 선언되었는지 검사한다.

- 예외 검사 과정

1. 메소드 내에서 발생 가능한 예외

- 이를 처리할 수 있는 try-catch 문이 있는지 검사.
- 없으면 그 예외가 메소드 헤더에 throws로 선언되어 있는지 검사.

2. 메소드 내에서 다른 메소드를 호출하는 경우

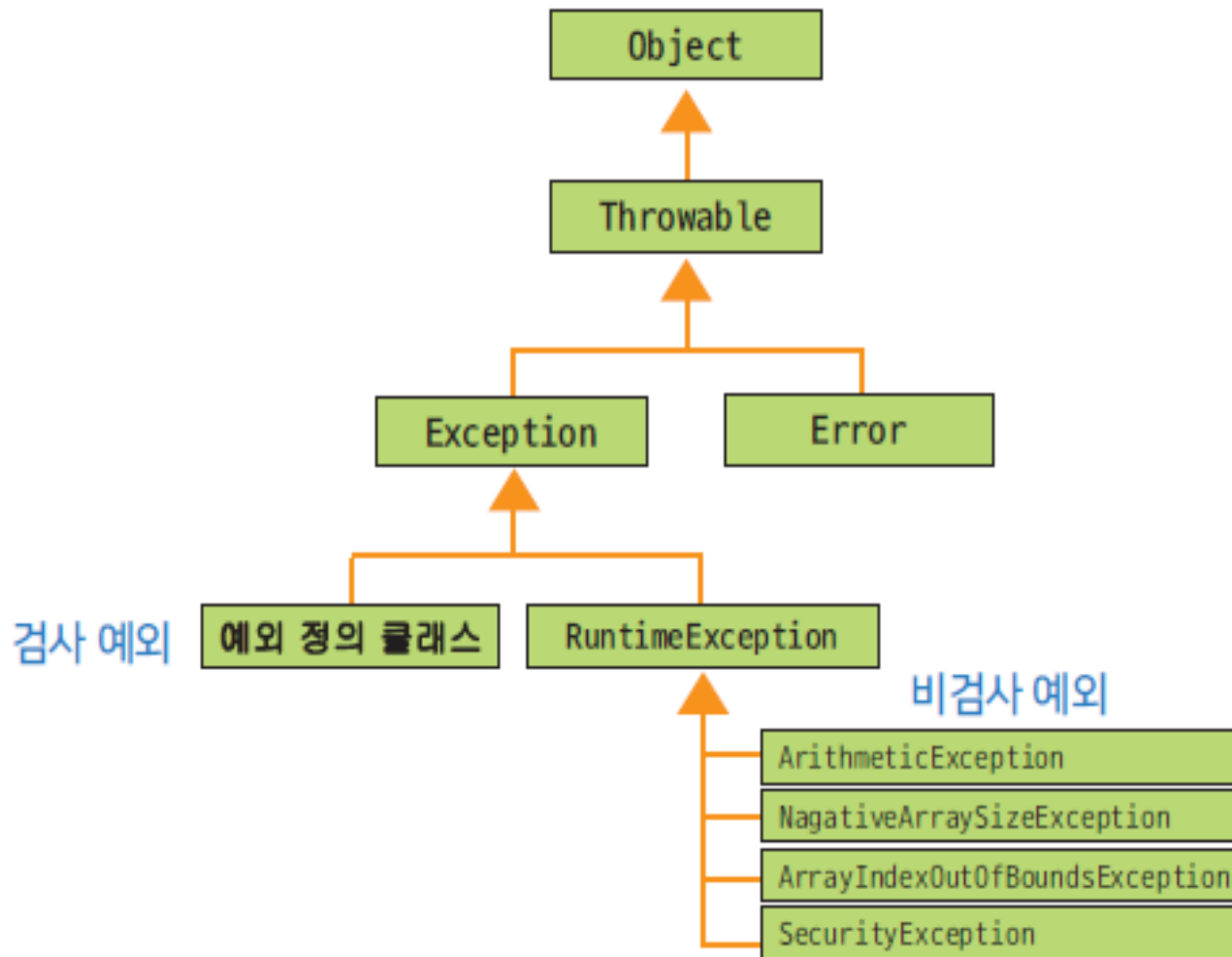
- 피호출자 메소드 헤더에 throws로 선언된 예외 정보를 참조.
- 이 예외를 처리할 수 있는 try-catch 문이 있는지 검사 .
- 없으면 호출자 메소드 헤더에 이 예외가 선언되어 있는지 검사.

# 검사 예외 vs 비검사 예외

---

- 검사 예외(`checked exception`)
  - 예외가 발생되면 처리될 수 있는지 컴파일러가 미리 검사한다.
  - 런타임 예외를 제외한 예외는 모두 검사 예외
  - 메소드 내에서 처리되지 않는 예외는 메소드 헤더 부분에 `throws`를 이용하여 선언되어야 한다.
- 비검사 예외(`unchecked exception`)
  - `RuntimeException`로부터 상속 받는 표준 런타임 예외
  - 예외 처리 여부를 컴파일러가 미리 검사하지 않는다.
  - 대신 필요에 따라 예외를 처리하도록 프로그래머가 알아서 프로그램을 작성해야 한다.

# Java: 검사 예외와 비검사 예외



# 실습문제

---

1. 10.1절에서 살펴본 예외 관련 기능을 인터프리터에 추가하여 구현하시오.

```
<command> → ... | exc id;           // 예외 정의
<stmt> → ...
        | raise id;                   // 예외 발생
        | try <stmt> catch(id) <stmt> // 예외 처리
```