

# Object – Oriented Programming

---

LAB #9-1. Exception Handling

# Exception

---

- Exception은 프로그램의 정상적인 수행 도중에 발생할 수 있는 오류를 나타냄
- Exception이 발생하면, 프로그램이 종료됨
- Exception handling을 통해서 종료되지 않고 실행이 유지되도록 할 수 있음
- 프로그램에서, 프로그래머는 exceptional case를 다루는 코드를 제공해야 한다.

# Not Catching Exceptions

---

```
Scanner keyboard = new Scanner(System.in);  
System.out.print("Enter age: ");  
int age = keyboard.nextInt();  
System.out.println("Your age is " + age);
```

사용자가 10 대신 'Ten' 을 입력하면...

```
Exception in thread "main" java.util.InputMismatchException  
    at java.base/java.util.Scanner.throwFor(Scanner.java:939)  
    at java.base/java.util.Scanner.next(Scanner.java:1594)  
    at java.base/java.util.Scanner.nextInt(Scanner.java:2258)  
    at java.base/java.util.Scanner.nextInt(Scanner.java:2212)  
    at practice09/practice09.test01.main(test01.java:9)
```

# Not Catching Exceptions

---

- 위의 상황에서 두 가지 일이 발생한다.
  - Java가 InputMismatchException을 발생시킴
  - 프로그램이 exception catch를 실패하여 충돌이 일어남
- try-catch 문으로 이 상황을 해결할 수 있다.

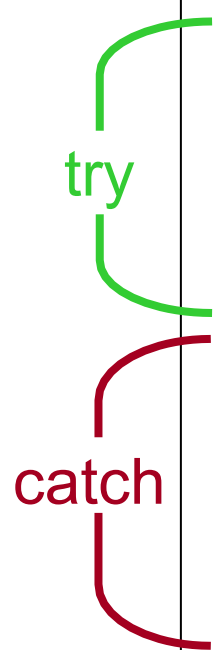
# try-throw-catch Basic

---

```
try {  
  
    // some code to attempt  
    // this code may throw an exception  
  
} catch (Exception e){  
  
    // catch the exception if it is thrown  
    // do whatever you want with it.  
  
}
```

# Catching an Exception

---



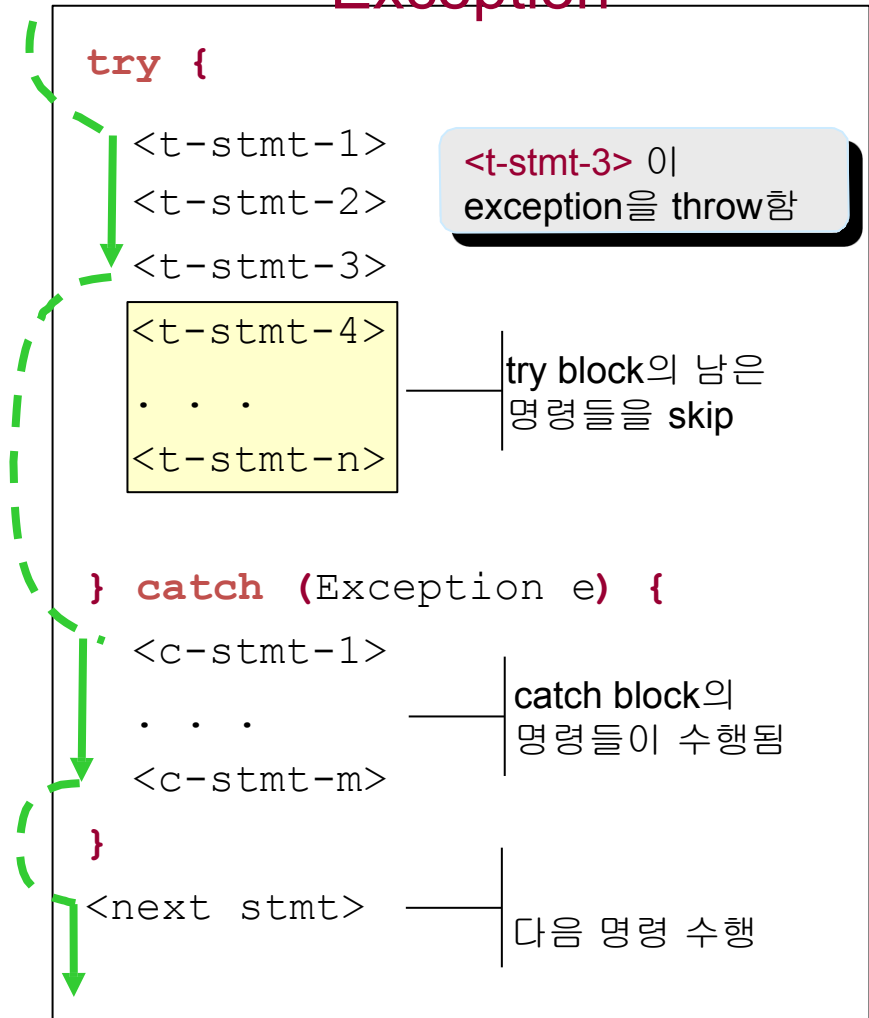
```
System.out.print("Enter age: ");

try {
    age = scanner.nextInt( );
    System.out.println("Your age is " + age);
} catch (InputMismatchException
e) {
    System.out.println("Invalid Entry.
    "
        + "Please enter digits
        only");
}
```

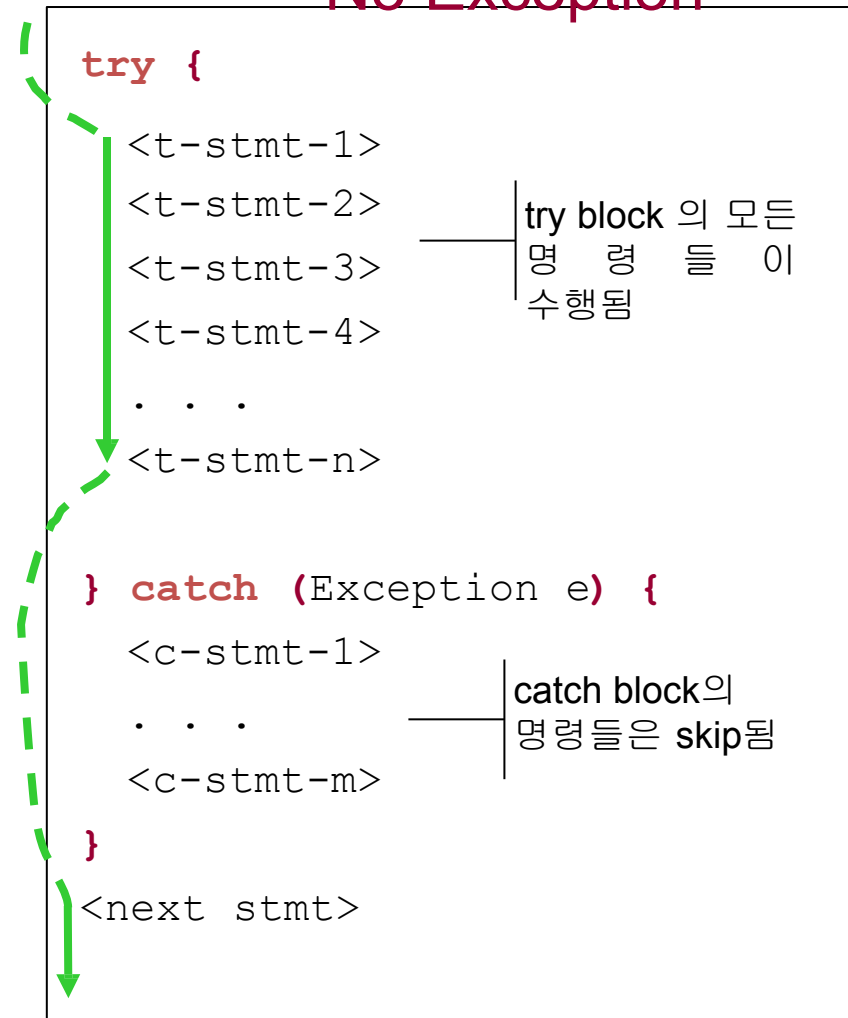
The diagram illustrates the structure of a try-catch block. A green curly brace on the left is labeled 'try' and encompasses the code within the try block. A red curly brace on the left is labeled 'catch' and encompasses the code within the catch block.

# try-catch Control Flow

## Exception



## No Exception



# try-catch

---

```
public static void main(String[] args) {  
    try {  
        System.out.println(1);  
        System.out.println(2);  
        System.out.println(0/0);  
        System.out.println(3);  
        System.out.println(4);  
    } catch (Exception e) {  
        System.out.println(5);  
        System.out.println(e.getMessage());  
    }  
    System.out.println(6);  
}
```

```
1  
2  
3  
4  
6
```

```
1  
2  
5  
/ by zero  
6
```



# Exception Object

---

- Exception 객체에 관하여 가장 중요한 2가지
  - exception의 type (i.e., exception class)
  - exception이 가지고 있는 message
- message는 instance variable로 exception 객체와 함께 전달된다.
- 이는 accessor 메소드인 getMessage()를 통해 불러올 수 있다.
  - > catch block에서 이 message를 이용 가능

# Exception Class

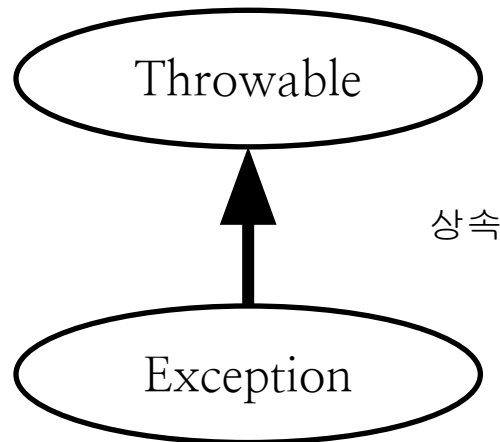
---

- Java의 표준 패키지에는 사전에 정의된 수많은 exception class들이 있다.
  - 예를 들면
    - IOException
    - NoSuchMethodException
    - FileNotFoundException
- 이러한 exception class들을 사용하기 위해서는 import 해야한다.
  - Import java.io.IOException

# Exception Message

---

- getMessage() Method
  - 모든 exception은 exception이 발생한 이유를 식별하는 message를 포함한 String instance variable을 가지고 있다.
  - getMessage() 메소드는 자세한 message 를 반환한다.



# Programmer-defined Exceptions

---

- Exception class는 프로그래머가 정의할 수 있으나,  
이미 존재하는 Exception class로 부터 파생된 Class여야 한다.
- 다른 Exception class가 적합하지 않다면, 클래스 'Exception'을 base class로 사용할 수 있다.
- 적어도 2개 이상의 생성자가 정의되어야 한다.
- 해당 Exception class는 getMessage()를 상속 받는다.

# Exception Message type

---

- 다른 타입의 인자를 받는 Exception class 생성자를 정의 할 수 있다.
  - 이 생성자는 값을 instance variable에 저장한다.
  - 이 instance variable에 접근하기 위해 accessor 메소드를 정의해야 한다.

# Programmer-defined Exceptions

---

```
Public class MyException extends Exception{
```

```
    // variables
```

```
    public MyException(){  
        super("default message");  
        //perform other tasks  
    }
```

```
    public MyException(String message){  
        super(message);  
        //perform other tasks  
    }
```

```
    //other methods if needed  
}
```

# Example

---

**Display 9.5    An Exception Class with an int Message**

```
1  public class BadNumberException extends Exception
2  {
3      private int badNumber;

4      public BadNumberException(int number)
5      {
6          super("BadNumberException");
7          badNumber = number;
8      }

9      public BadNumberException()
10     {
11         super("BadNumberException");
12     }

13     public BadNumberException(String message)
14     {
15         super(message);
16     }

17     public int getBadNumber()
18     {
19         return badNumber;
20     }
21 }
```

# Multiple catch Blocks

---

- try-catch 문은 여러 개의 catch block을 가질 수 있다.

```
try {  
    ...  
    n = keyboard.nextInt();  
    ...  
} catch (InputMismatchException e) {  
    ...  
} catch (NumberFormatException e) {  
    ...  
}
```



# Multiple catch Blocks

---

- 여러 개의 catch 문을 사용할 때는 구체적인 exception을 먼저 catch해야 한다.

```
catch (BadNumberException e)
```

```
{ ... }
```

```
catch (Exception e)
```

```
{ ... }
```

- BadNumberException 이 Exception 보다 구체적이므로 먼저 catch되어야 한다.

# Throwing Exceptions

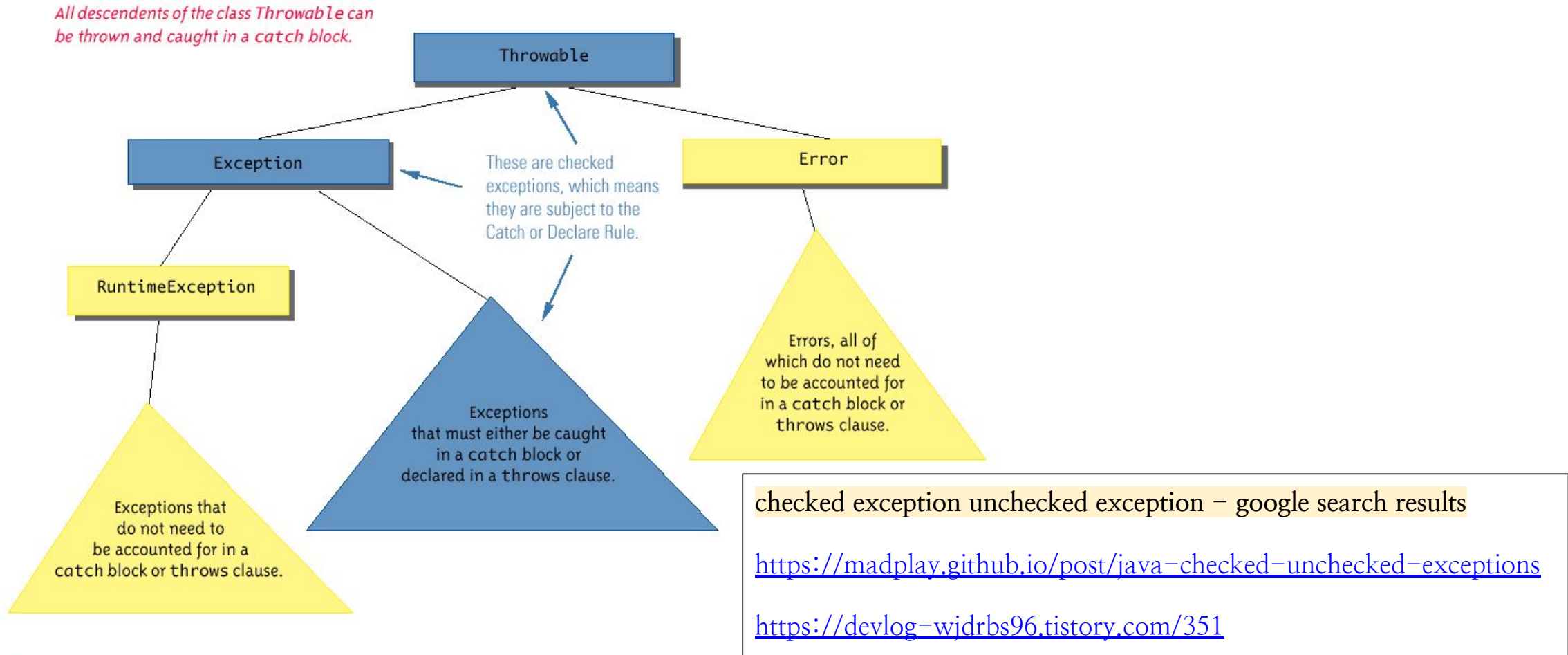
---

- method 내에서 exception을 handling 하지 않고 throw 할 수 있음
  - 해당 exception이 checked exception이라면 호출한 method에서 처리해주어야 함
  - 호출한 method에서도 throw 할 수 있지만, 결국에는 어디에서든 try-catch를 통해서 handling되어야 함

```
public static void isBadNumber(int num) throws BadNumberException {  
    if (num <= 0)  
        throw new BadNumberException(num);  
}
```

# Checked Exception vs Unchecked Exception

Display 9.10 Hierarchy of Throwable Objects



# Checked Exception vs Unchecked Exception

**nextInt**

...

Throws:

`InputMismatchException` - if the next token does not match the *Integer* regular expression, or is out of range  
`NoSuchElementException` - if input is exhausted  
`IllegalStateException` - if this scanner is closed

**FileReader**

...

Throws:

`FileNotFoundException` - if the named file does not exist, is a directory rather than a regular file, or

## Class InputMismatchException

```
java.lang.Object
  java.lang.Throwable
    java.lang.Exception
      java.lang.RuntimeException
        java.util.NoSuchElementException
          java.util.InputMismatchException
```

## Class FileNotFoundException

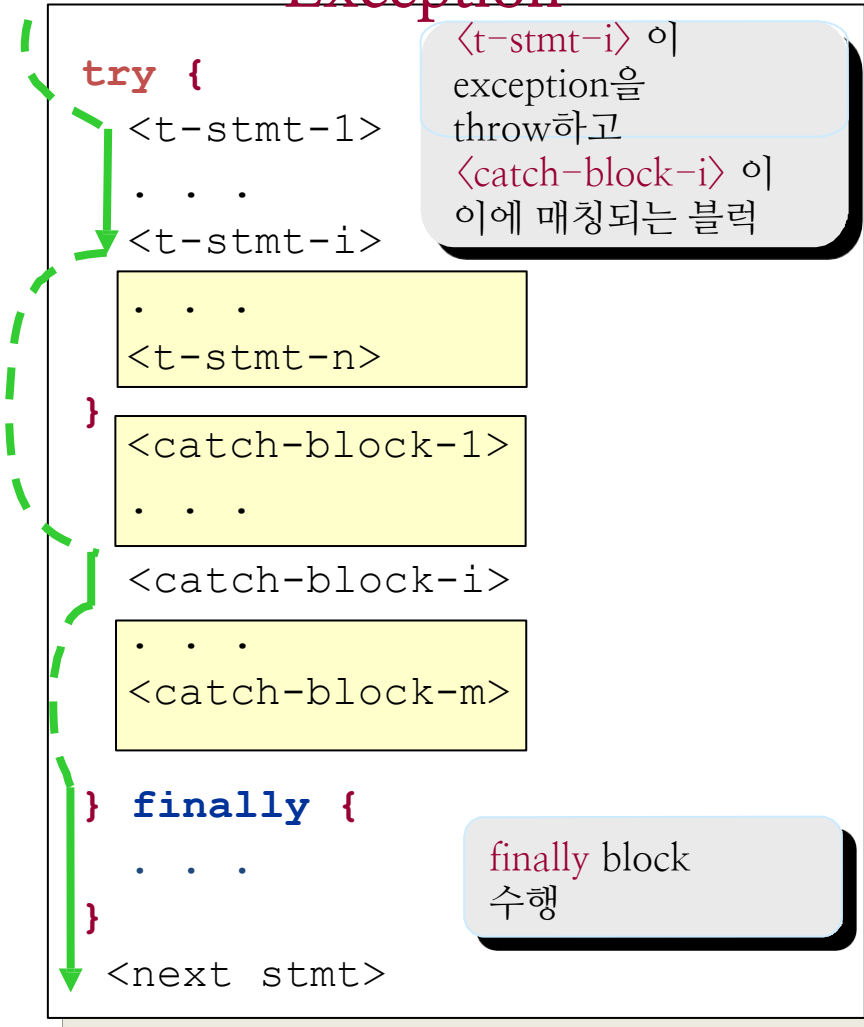
```
java.lang.Object
  java.lang.Throwable
    java.lang.Exception
      java.io.IOException
        java.io.FileNotFoundException
```

```
int age = keyboard.nextInt();
```

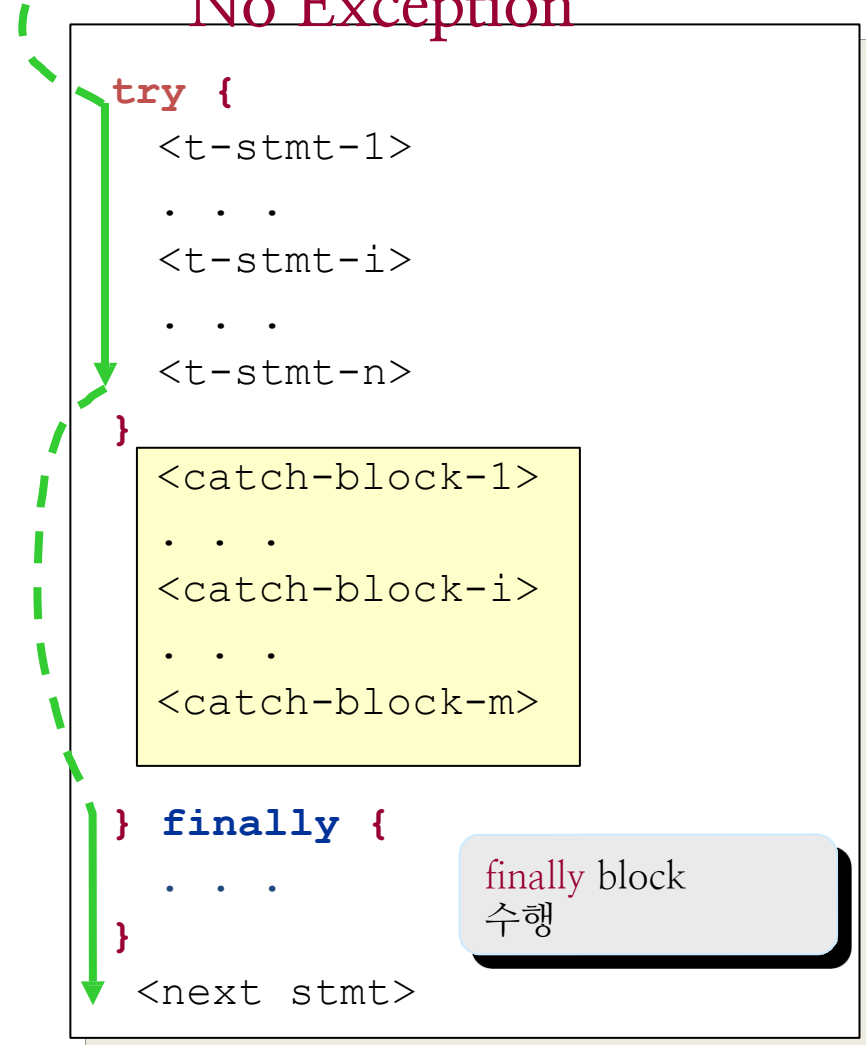
```
BufferedReader inputStream = new BufferedReader(new FileReader("morestuff2.txt"));
```

# The finally Block

## Exception



## No Exception



# 실습 과제

---

NegativeException, TooMuchExpenseException, Wallet class를 생성한다.

NegativeException: 음수 값을 입력하면 발생하는 exception

인자가 없는 생성자 작성

default message는 “price must be positive”

String 인자를 받는 생성자 작성 ( 받은 인자를 message로 함 )

TooMuchExpenseException: 100를 초과하는 숫자를 입력하면 발생하는 exception

한 개의 instance variable을 가짐 ( private int inputNum )

인자가 없는 생성자 작성

default message는 “Not enough balance.”

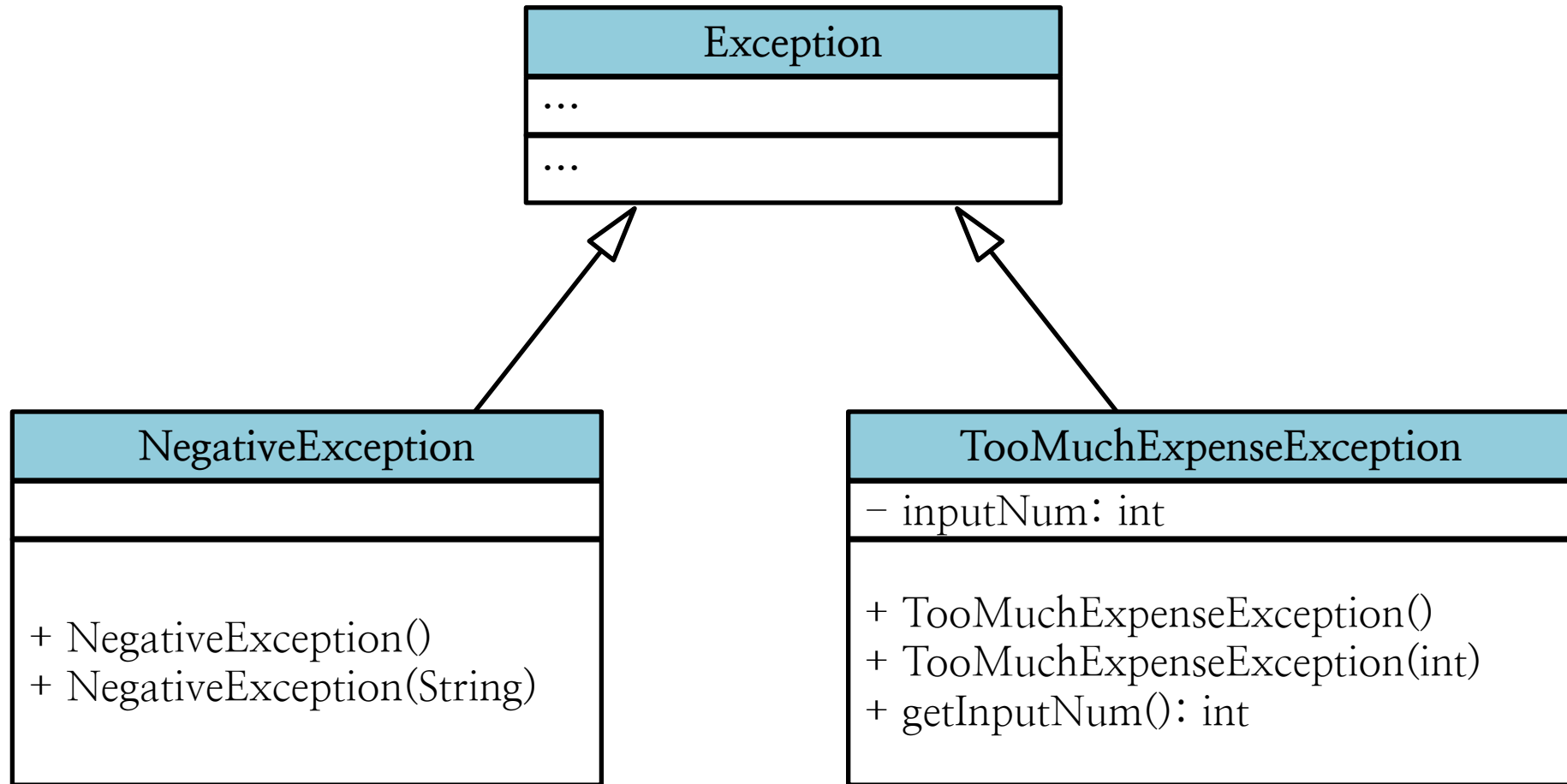
int 인자를 받는 생성자 작성 ( inputNum을 초기화 )

message는 “Over the limit!”

inputNum을 반환하는 getter 작성

# 실습 과제

---



# 실습 과제

## Wallet class

3개의 private instance variable를 가진다.

```
String name  
int balance  
int txIndex
```

name을 인자로 받고 instance variable의 값을 초기화하는 생성자 작성

```
txIndex = 0, balance = 100으로 초기화
```

balance에 대한 getter를 작성한다.

txIndex를 1만큼 증가시키는 void increaseIndex() method 작성

balance를 expense 만큼 감소시키는 void decreaseBalance(int expense) method 작성

```
toString() "name: "+this.name+", #"+this.txIndex+", balance: "+this.balance
```

지갑에 돈이 없으면 집에 가야하는 메소드인 empty() 작성

```
balance <= 0 일 때 Exception("Go Home")을 발생시키도록 작성
```

Wallet
<ul style="list-style-type: none"><li>- name: String</li><li>- balance: int</li><li>- txIndex: int</li></ul>
<ul style="list-style-type: none"><li>~ Wallet(String)</li><li>~ getBalance(): int</li><li>~ increaseIndex(): void</li><li>~ decreaseBalance(int): void</li><li>+ toString(): String</li><li>+ empty(int): void</li></ul>



# 실습 과제

---

Market class를 생성하고 main method에서 다음과 같은 코드를 작성한다.

Wallet 객체를 하나 생성한다. 사용자 입력을 받을 준비를 한다.

무한 반복문을 사용하여 다음을 수행한다.

Wallet 클래스의 empty메소드를 호출하여 객체의 지갑 잔고를 체크한다.

사용할 금액을 입력한다. ex) “Enter price : ”

입력한 금액에 따라 다른 결과를 출력한다.

입력 값이 0보다 작을 경우, NegativeException() 을 발생시킨다.

입력 값이 100보다 클 경우,

TooMuchExpenseException(expense)을 발생시킨다. (expense는 입력 받은 지출 비용 값)

입력 값이 잔고보다 클 경우 TooMuchExpenseException()을 발생시킨다.

입력값이 0~balance 일 경우, 다음 작업을 수행한다.

Wallet 객체의 Index 증가

Wallet 객체의 잔고를 입력값 만큼 감소(decreaseBalance)

# 실습 과제

---

(계속)

발생한 예외에 따른 **catch** 문을 작성한다.

NegativeException일 경우, 해당 Exception의 Message를 출력한다.

TooMuchExpenseException일 경우,

해당 Exception의 Message를 출력한다.

getMessage 를 이용하여 Message가 “Over the limit!” 일 경우에는  
inputNum과 함께 message를 출력한다.

ex) “you pay”+ inputNum (getter 사용)

Exception일 경우, 해당 Exception의 Message를 출력하고 Scanner 객체를 닫고 프로그램을 종료한다. ( return 사용 )

**finally** 문에서 위의 모든 결과와는 무관하게 항상 출력되는 문자열을 출력한다.

지갑 객체의 현 잔고를 출력한다. (toString)

트랜잭션 완료 메시지를 출력한다. (example: “---transaction complete--- \n”)

# 실습 과제

NegativeException.java,  
TooMuchExpenseException.java,  
Wallet.java,  
Market.java 를 제출

```
// printStackTrace 사용시
// 이클립스 콘솔에서 에러메시지 표시가 제대로 안될때

System.out.println(e);
System.out.println("\tat "+e.getStackTrace()[0]);
```

```
Enter price: 50
peace~~
name: my wallet, #1, balance: 50
---transaction complete---

Enter price: 100
wallets.TooMuchExpenseException: Not enough balance.
    at wallets.Market.main(Market.java:24)
oh, my!
name: my wallet, #1, balance: 50
---transaction complete---

Enter price: 150
wallets.TooMuchExpenseException: Over the limit!
    at wallets.Market.main(Market.java:21)
you pay 150
oh, my!
name: my wallet, #1, balance: 50
---transaction complete---

Enter price: 0
peace~~
name: my wallet, #2, balance: 50
---transaction complete---

Enter price: -10
wallets.NegativeException: price must be positive
    at wallets.Market.main(Market.java:27)
oh, sorry!
name: my wallet, #2, balance: 50
---transaction complete---

Enter price: 20
peace~~
name: my wallet, #3, balance: 30
---transaction complete---

Enter price: 30
peace~~
name: my wallet, #4, balance: 0
---transaction complete---

java.lang.Exception: Go Home
    at wallets.Wallet.empty(Wallet.java:41)
the end...
name: my wallet, #4, balance: 0
---transaction complete---
```