



JAVA Exception Handling

Prof. Youngchul Jung



YEUNGJIN UNIVERSITY

프로그램 오류

- 컴파일 에러(compile-time error)와 런타임 에러(runtime error)

- ✓ 컴파일 에러 : 컴파일 시 발생하는 에러
- ✓ 런타임 에러 : 실행 시 발생하는 에러

Ex8 소스코드 컴파일

```
C:\W\WINDOWS\system32\cmd.exe
C:\W\jdk1.5\work>javac Ex8.java
Ex8.java:6: cannot find symbol
symbol : method println(int)
location: class java.io.PrintStream
    System.out.println(i);
                   ^
1 error
```

Ex8 클래스 실행

```
C:\W\WINDOWS\system32\cmd.exe
C:\W\jdk1.5\work>java Ex8
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 0
    at Ex8.main(Ex8.java:6)

C:\W\jdk1.5\work>
```

- Java의 런타임 에러 – 에러(error)와 예외(exception)

에러(error) - 프로그램 코드에 의해서 수습될 수 없는 심각한 오류

예외(exception) - 프로그램 코드에 의해서 수습될 수 있는 다소 미약한 오류



예외처리의 정의와 목적

- 에러(error)는 어쩔 수 없지만, 예외(exception)는 처리해야 한다.

에러(error) - 프로그램 코드에 의해서 수습될 수 없는 심각한 오류

예외(exception) - 프로그램 코드에 의해서 수습될 수 있는 다소 미약한 오류

- 예외처리의 정의와 목적

예외처리(exception handling)의

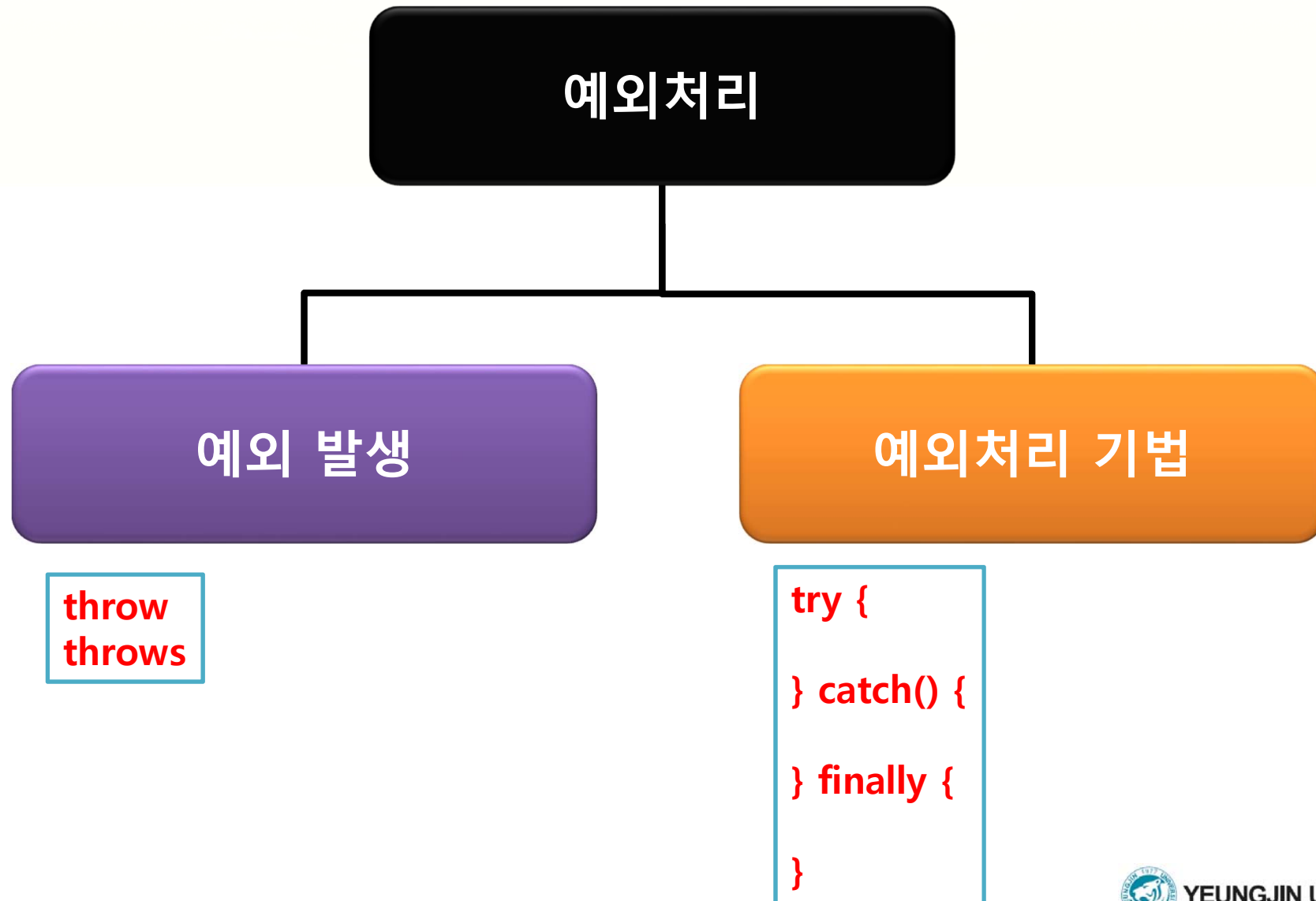
정의 - 프로그램 실행 시 발생할 수 있는 예외의 발생에 대비한 코드를 작성하는 것

목적 - 프로그램의 비정상 종료를 막고, 정상적인 실행상태를 유지하는 것

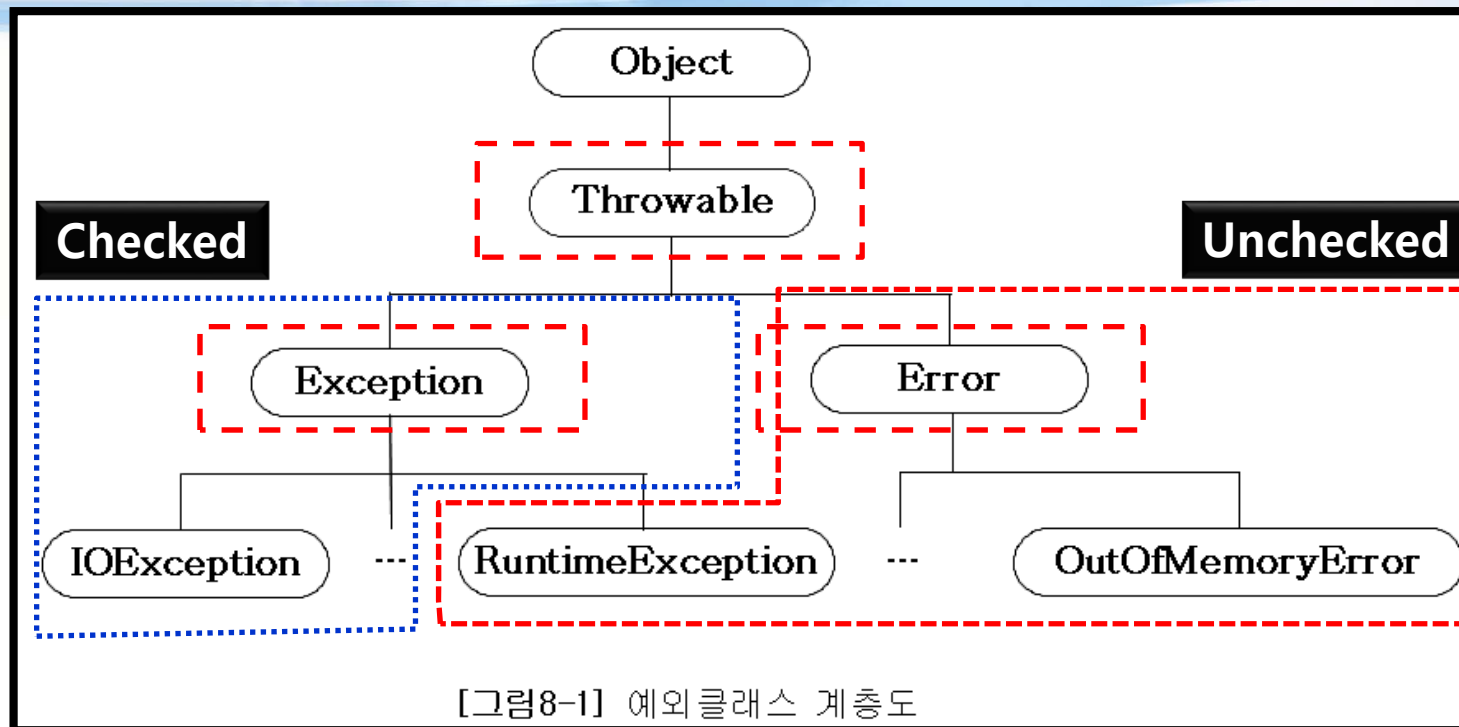
[참고] 에러와 예외는 모두 실행 시(runtime) 발생하는 오류이다.



예외처리 기술 구성요소



예외 클래스의 계층구조



- 예외 클래스는 크게 두 그룹으로 나뉜다
- **Unchecked** : RuntimeException + Error
 - ✓ 프로그래머의 실수로 발생하는 예외
 - ✓ 대부분 논리적 오류, 오류 수정이 어려움
- **Checked** : RuntimeException을 제외한 Exception 의 모든 자손 클래스
 - ✓ 사용자(User)와 같은 외적인 요인에 의해 발생하는 예외
 - ✓ 반드시 예외처리 구문 필요 (try-catch, throws)

예외생성 실습 - 1

```
class A {  
    void L1(){  
        L2();  
    }  
  
    void L2() {  
        L3();  
    }  
  
    void L3() {  
        // 예외 객체 생성  
        RuntimeException e = new RuntimeException("Exception test");  
  
        System.out.println("1");  
  
        // 예외 객체를 던진다.  
        throw e;  
  
        // 아래 문장이 실행 될까요?  
        //System.out.println("2");  
    }  
}
```

throw (Exception 객체 명);

- 발생한 예외객체를 메소드 밖으로 던진다.
- throwing 동시에 메소드 종료

```
public class MainClass{  
  
    public static void main(String args[]) {  
        A a = new A();  
        a.L1();  
    }  
}
```

예외생성 실습 - 2

```
class A {  
    void L1(){  
        L2();  
    }  

```

```
    void L2() {  
        L3();  
    }  

```

```
RuntimeException e = new RuntimeException("Exception test");
```

```
    void L3() {
```

```
        // 예외 객체 생성
```

```
        Exception e = new Exception("Exception test");
```

```
        System.out.println("1");
```

```
        // 예외 객체를 던진다.
```

```
        throw e;
```

```
        // 아래 문장이 실행 될까요?
```

```
        //System.out.println("2");
```

```
    }
```

```
}
```



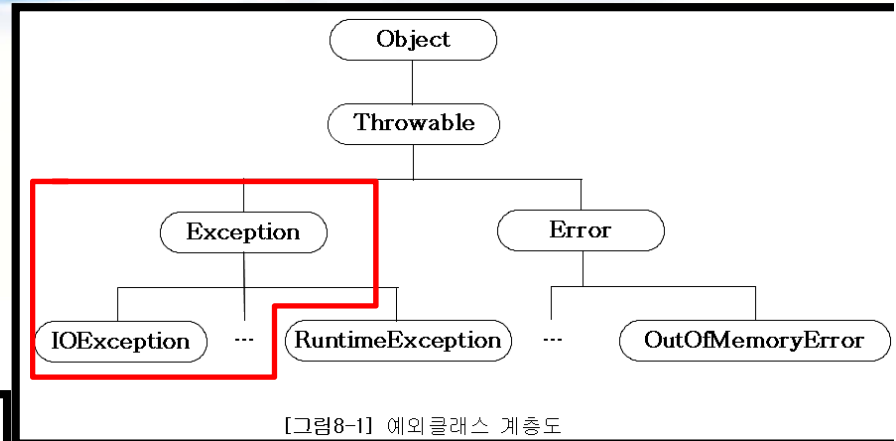
빨간 펜 선생님 등장 이유?



Checked exceptions

- Checked exception의 경우 반드시 발생한 예외를 처리. 아래 양자 택일!!
 - ✓ 메소드 내 Try-catch 문 내에서 처리
 - ✓ 메서드 내에서 처리 불가시 메소드 밖으로 예외 전달

```
class A {  
    void L1(){  
        L2();  
    }  
  
    void L2() {  
        L3();  
    }  
  
    void L3() throws Exception {  
        // 예외 객체 생성  
        Exception e = new Exception("Exception test");  
  
        System.out.println("1");  
  
        // 예외 객체를 던진다.  
        throw e;  
  
        // 아래 문장이 실행 될까요?  
        //System.out.println("2");  
    }  
}
```



throws???



throws

- 메서드 내 발생할 수 있는 예외를 선언
- Checked exception이 메서드 내에서 Throwing 될 경우 반드시 선언!!

notify

```
public final void notify()
```

Wakes up a single thread that is waiting on this object's monitor. If any threads are waiting, the thread that has been waiting the longest is awakened, at the discretion of the implementation. A thread waits on an object's monitor until it is awakened.

The awakened thread will not be able to proceed until the current thread releases its lock on this object. There may be many other threads waiting on the object; for each of these threads, the JVM will become the owner of the lock this object.

This method should only be called by a thread that is the owner of this object's monitor.

- By executing a synchronized instance method of that object.
- By executing the body of a synchronized statement that synchronizes on the object.
- For objects of type `Class`, by executing a synchronized static method of that class.

Only one thread at a time can own an object's monitor.

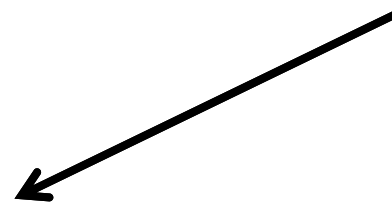
Throws:

`IllegalMonitorStateException` - if the current thread is not the owner of the monitor.

See Also:

`notifyAll()`, `wait()`

메서드에서 발생하는 예외 종류 설명



자신만의 예외정의 하기

- 기존의 예외 클래스를 상속받아서 새로운 예외 클래스를 정의할 수 있다

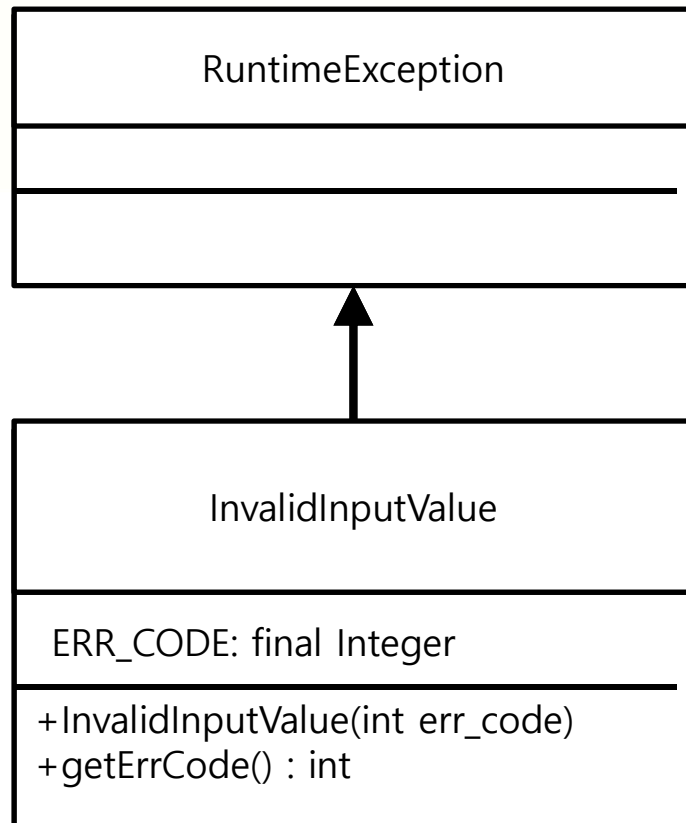
```
class MyException extends Exception {  
    MyException(String msg) { // 문자열을 매개변수로 받는 생성자  
        super(msg); // 조상인 Exception 클래스의 생성자를 호출한다.  
    }  
}
```

- 에러코드를 저장할 수 있게 ERR_CODE와 getErrCode()를 멤버로 추가

```
class MyException extends Exception {  
    // 에러 코드 값을 저장하기 위한 필드를 추가 했다.  
    private final int ERR_CODE;  
  
    MyException(String msg, int errCode) { // 생성자.  
        super(msg);  
        ERR_CODE = errCode;  
    }  
  
    MyException(String msg) { // 생성자.  
        this(msg, 100); // ERR_CODE를 100 (기본값) 으로 초기화한다.  
    }  
  
    public int getErrCode() { // 에러 코드를 얻을 수 있는 메서드도 추가했다.  
        return ERR_CODE; // 이 메서드는 주로 getMessage()와 함께 사용될 것이다.  
    }  
}
```



사용자 정의형 예외 만들기 실습 (1)



입력 값이 지정된 범위를 초과 할 경우,
아래와 같이 에러코드 생성

Err_code

18 : 입력 값이 0보다 적을 경우
218 : 입력 값이 100보다 클 경우



사용자 정의형 예외 만들기 실습 (2)

```
public class MainClass {  
    public static void main(String args[]) {  
        MainClass main = new MainClass();  
  
        System.out.println("입력 값: " + main.getInputValue());  
    }  
  
    int getInputValue() throws InvalidInputValue{  
        Scanner scn = new Scanner(System.in);  
  
        int inputValue = scn.nextInt();  
  
        if(inputValue < 0)  
            throw new InvalidInputValue(18);  
        else if(inputValue > 100)  
            throw new InvalidInputValue(218);  
  
        return inputValue;  
    }  
}
```

발생된 예외는 어떻게 처리하지??



예외처리구문 – try-catch (1)

- 예외를 처리하려면 **try-catch**문을 사용해야 한다.

```
try {  
    // 예외가 발생할 가능성이 있는 문장들을 넣는다.  
} catch (Exception1 e1) {  
    // Exception1이 발생했을 경우, 이를 처리하기 위한 문장을 적는다.  
} catch (Exception2 e2) {  
    // Exception2가 발생했을 경우, 이를 처리하기 위한 문장을 적는다.  
...  
} catch (ExceptionN eN) {  
    // ExceptionN이 발생했을 경우, 이를 처리하기 위한 문장을 적는다.  
}
```

```
class ExceptionEx11 {  
    public static void main(String args[]) {  
        System.out.println(1);  
        System.out.println(2);  
        try {  
            System.out.println(3);  
            System.out.println(0/0);  
            System.out.println(4);  
        } catch (ArithmeticException ae) {  
            if (ae instanceof ArithmeticException)  
                System.out.println("true");  
            System.out.println("ArithmeticException");  
        } catch (Exception e) {  
            System.out.println("Exception");  
        } // try-catch의 끝  
        System.out.println(6);  
    } // main메서드의 끝  
}
```

0으로 나뉘어서
ArithmeticException을
발생시킨다.

ArithmeticException을
제외한 모든 예외가 처리된
다.



예외처리구문 – try-catch (2)

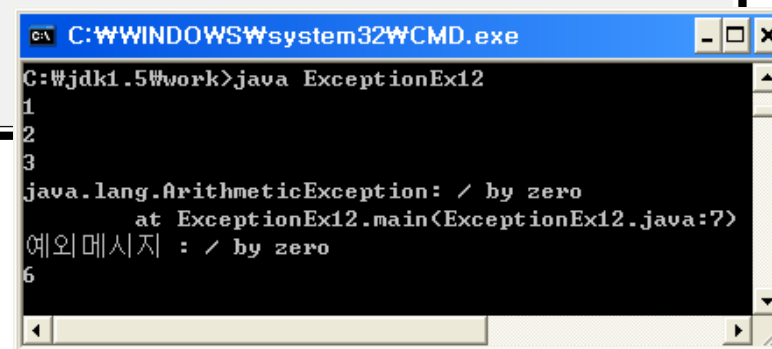
- 발생한 예외 객체를 catch 블록의 참조변수로 접근

`printStackTrace()` - 예외발생 당시의 호출스택(Call Stack)에 있었던 메서드의 정보와 예외 메시지를 화면에 출력한다.

`getMessage()` - 발생한 예외클래스의 인스턴스에 저장된 메시지를 얻을 수 있다.

```
class ExceptionEx12 {  
    public static void main(String args[]) {  
        System.out.println(1);  
        System.out.println(2);  
        try {  
            System.out.println(3);  
            System.out.println(0/0); // 예외발생!!!  
            System.out.println(4);    // 실행되지 않는다.  
        } catch (ArithmeticException ae) {  
            ae.printStackTrace();  
            System.out.println("예외메시지 : " + ae.getMessage());  
        } // try-catch의 끝  
        System.out.println(6);  
    } // main메서드의 끝  
}
```

참조변수 ae를 통해, 생성된 ArithmeticException인스턴스에 접근할 수 있다.



```
C:\WINDOWS\system32\cmd.exe  
C:\jdk1.5\work>java ExceptionEx12  
1  
2  
3  
java.lang.ArithmeticException: / by zero  
    at ExceptionEx12.main(ExceptionEx12.java:?)  
예외메시지 : / by zero  
6
```

사용자 정의형 예제 완성하기!! (1)

```
public class MainClass {  
    public static void main(String args[]) {  
        MainClass main = new MainClass();  
  
        System.out.println("입력 값: " + main.getInputValue());  
    }  
  
    int getInputValue() throws InvalidInputValue {  
        Scanner scn = new Scanner(System.in);  
  
        int inputValue = scn.nextInt();  
  
        if(inputValue < 0)  
            throw new InvalidInputValue(18);  
        else if(inputValue > 100)  
            throw new InvalidInputValue(218);  
  
        return inputValue;  
    }  
}
```

프로그램 실행 후 콘솔화면

1000

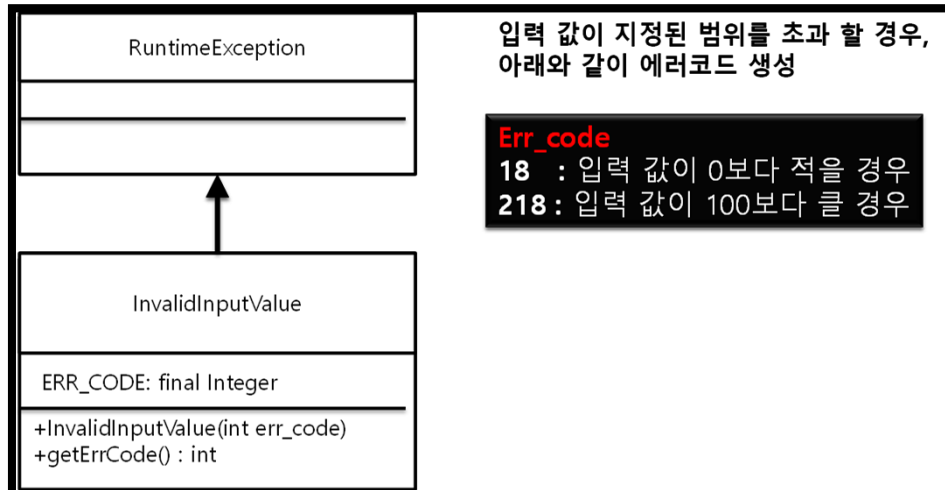
입력 값은 0 ~ 100 사이 입니다.
다시 입력하세요

-100

입력 값은 0 ~ 100 사이 입니다.
다시 입력하세요

50

입력 값: 50



사용자 정의형 예제 완성하기!! (2)

```
public static void main(String args[]) {  
    MainClass main = new MainClass();  
    boolean flag = true;  
  
    while (flag) {  
        try {  
            System.out.println("입력 값: " + main.getInputValue());  
            flag = false;  
        } catch (InvalidInputValue e) {  
            System.out.println("입력 값은 0 ~ 100 사이 입니다.");  
            System.out.println("다시 입력하세요");  
        }  
    }  
}
```



예외처리구문 – try-catch-finally (1)

- 예외의 발생여부와 관계없이 실행되어야 하는 코드를 넣는다
- 선택적으로 사용할 수 있으며, try-catch-finally의 순서로 구성된다.
- 예외 발생시, try → catch → finally의 순서로 실행되고
- 예외 미 발생시, try → finally의 순서로 실행된다.
- try 또는 catch 블록에서 return문을 만나도 finally 블록은 수행된다.

```
try {  
    // 예외가 발생할 가능성이 있는 문장들을 넣는다.  
} catch (Exception1 e1) {  
    // 예외처리를 위한 문장을 적는다.  
} finally {  
    // 예외의 발생여부에 관계없이 항상 수행되어야하는 문장들을 넣는다.  
    // finally블록은 try-catch문의 맨 마지막에 위치해야한다.  
}
```



예외처리구문 – try-catch-finally (2)

```
class FinallyTest {
    public static void main(String args[]) {
        try {
            startInstall();          // 프로그램 설치에 필요한 준비를 한다.
            copyFiles();             // 파일들을 복사한다.
            deleteTempFiles();       // 프로그램 설치에 사용된 임시파일들을 삭제한다.
        } catch (Exception e) {
            e.printStackTrace();
            deleteTempFiles();       // 프로그램 설치에 사용된 임시파일들을 삭제한다.
        } // try-catch의 끝
    } // main의 끝

    static void startInstall() {
        /* 프로그램 설치에 필요한 준비를 하는 코드를 적는다.*/
    }

    static void copyFiles() { /* 파일들을 복사하는 코드를 적는다. */ }
    static void deleteTempFiles() { /* 임시파일들을 삭제하는 코드를 적는다.*/ }
}
```

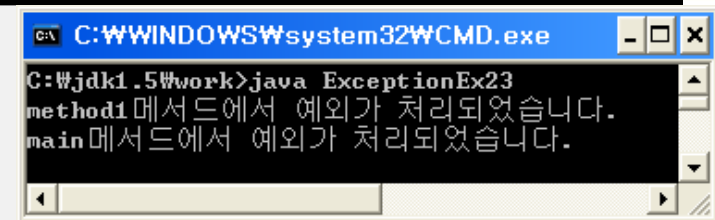
```
try {
    startInstall();
    copyFiles();
} catch (Exception e) {
    e.printStackTrace();
} finally {
    deleteTempFiles();
} // try-catch의 끝
```



예외처리구문 – 예외 되 던지기

- 예외를 처리한 후에 다시 예외를 생성해서 호출한 메서드로 전달하는 것
- 예외가 발생한 메서드와 호출한 메서드, 양쪽에서 예외를 처리해야 하는

```
class ExceptionEx23 {  
    public static void main(String[] args)  
    {  
        try {  
            method1();  
        } catch (Exception e) {  
            System.out.println("main메서드에서 예외가 처리되었습니다.");  
        }  
    } // main메서드의 끝  
  
    static void method1() throws Exception {  
        try {  
            throw new Exception();  
        } catch (Exception e) {  
            System.out.println("method1메서드에서 예외가 처리되었습니다.");  
            throw e; // 다시 예외를 발생시킨다.  
        }  
    } // method1메서드의 끝  
}
```



예외처리구문 – try-catch 사용시 주의사항

```
public static void main(String[] args)
{
    try {
        try {    } catch (Exception e) {
            //...
        }
    } catch (Exception e) {
        try {    } catch (Exception e) { // 컴파일 에러 발생 !!!
            //...
        }
    } // try-catch의 끝
} // main메서드의 끝
```



왜 그렇지?

그럼 해결 방법은?



실습문제

```
public class MainClass {  
    public static void main(String args[]) {  
        int answer = (int) (Math.random() * 100) + 1;  
        int input = 0;  
        int count = 0;
```

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

```
        do {  
            1~100사이의 임의의 수를 입력하세요
```

```
            50
```

```
            50 보다 작은 수를 입력하세요
```

```
            1~100사이의 임의의 수를 입력하세요
```

```
            10
```

```
            10 보다 큰 수를 입력하세요
```

```
            1~100사이의 임의의 수를 입력하세요
```

```
            cyka
```

```
            Exception in thread "main" java.util.InputMismatchException
```

```
                at java.util.Scanner.throwFor(Unknown Source)
```

```
                at java.util.Scanner.next(Unknown Source)
```

```
                at java.util.Scanner.nextInt(Unknown Source)
```

```
                at java.util.Scanner.nextInt(Unknown Source)
```

```
                at tt.MainClass.main(MainClass.java:17)
```

```
        } while
```

```
    }
```

```
}
```

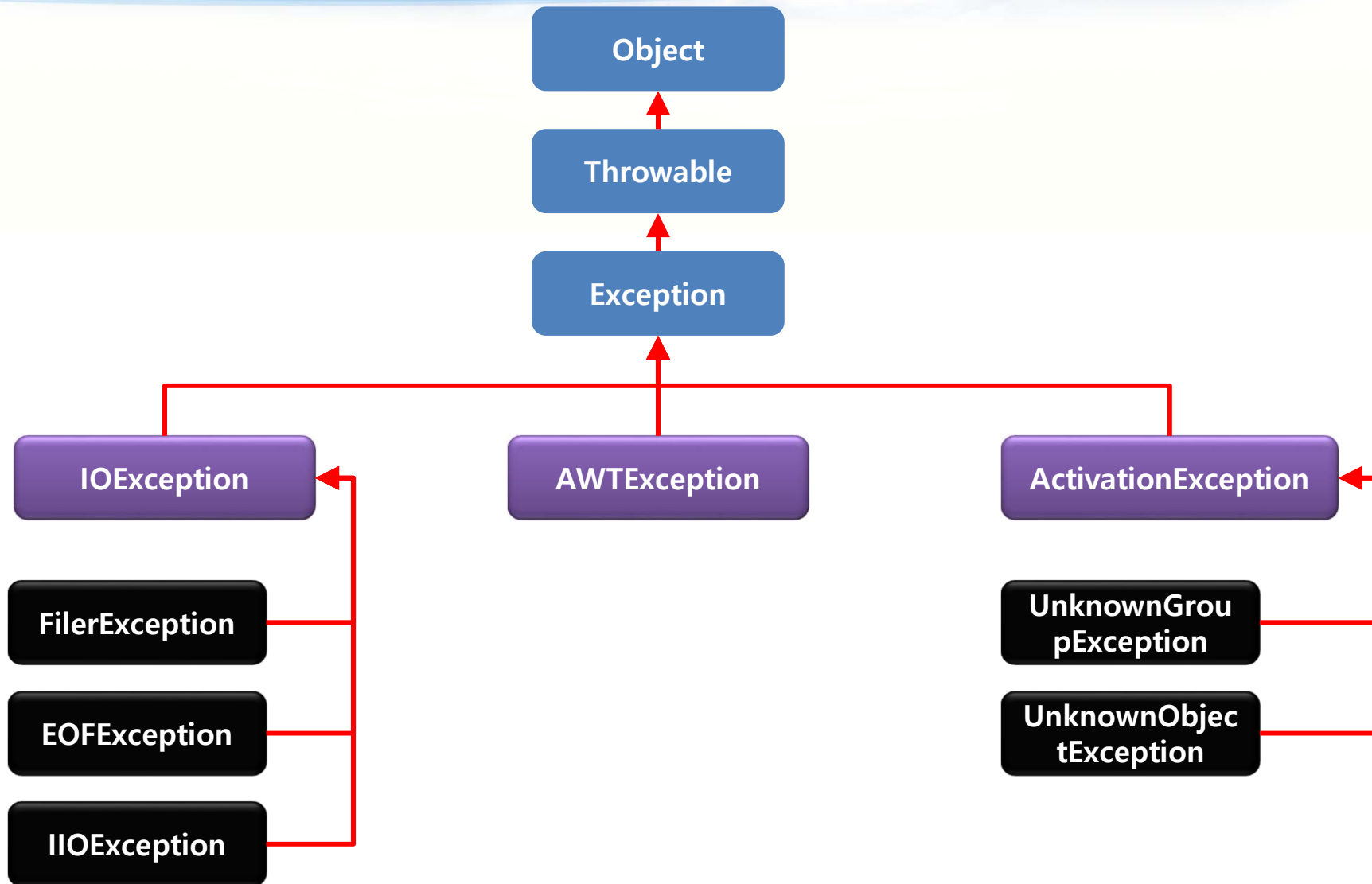
Exception Handling with Overriding

- ✓ Overriding 시 Exception throws 규칙은 checked 예외만 고려
- ✓ Overriding 되는 메서드 (Child class)에서는 부모(현 시점 상위 클래스만 적용)
메서드에 선언된 예외와 같거나 자식 예외만 선언 될 수 있다.

```
8 class Parent {
9     void myMethod() throws ActivationException, InvalidTypeException, IOException{
10    }
11 }
12
13 class Child extends Parent{
14     // 오버라이딩
15     void myMethod() throws UnknownObjectException, UnknownGroupException{
16    }
17 }
18
19 public class ExceptionTestMain {
20     public static void main(String[] args) throws Exception {
21         new Child().myMethod();
22         System.out.println("Completed");
23     }
24 }
```



예외 클래스 상속 관계도 예제





Thanks



YEUNGJIN UNIVERSITY
Prof. Youngchul Jung