

능동적 사고 방식의

java

강사 박주병

Park Ju Byeong

Park Ju Byeong



# Part12 예외처리

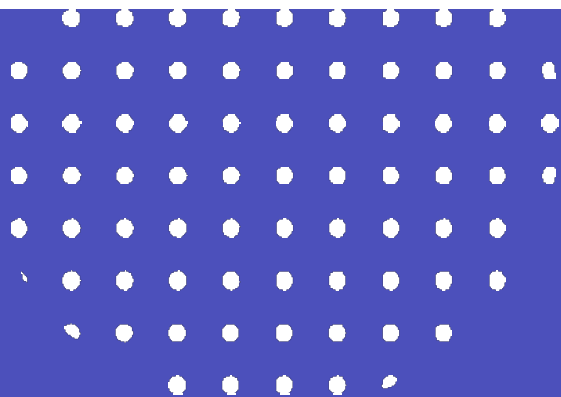
---

01 에러의종류

02 예외처리

03 사용자 정의 예외

04 실습 문제

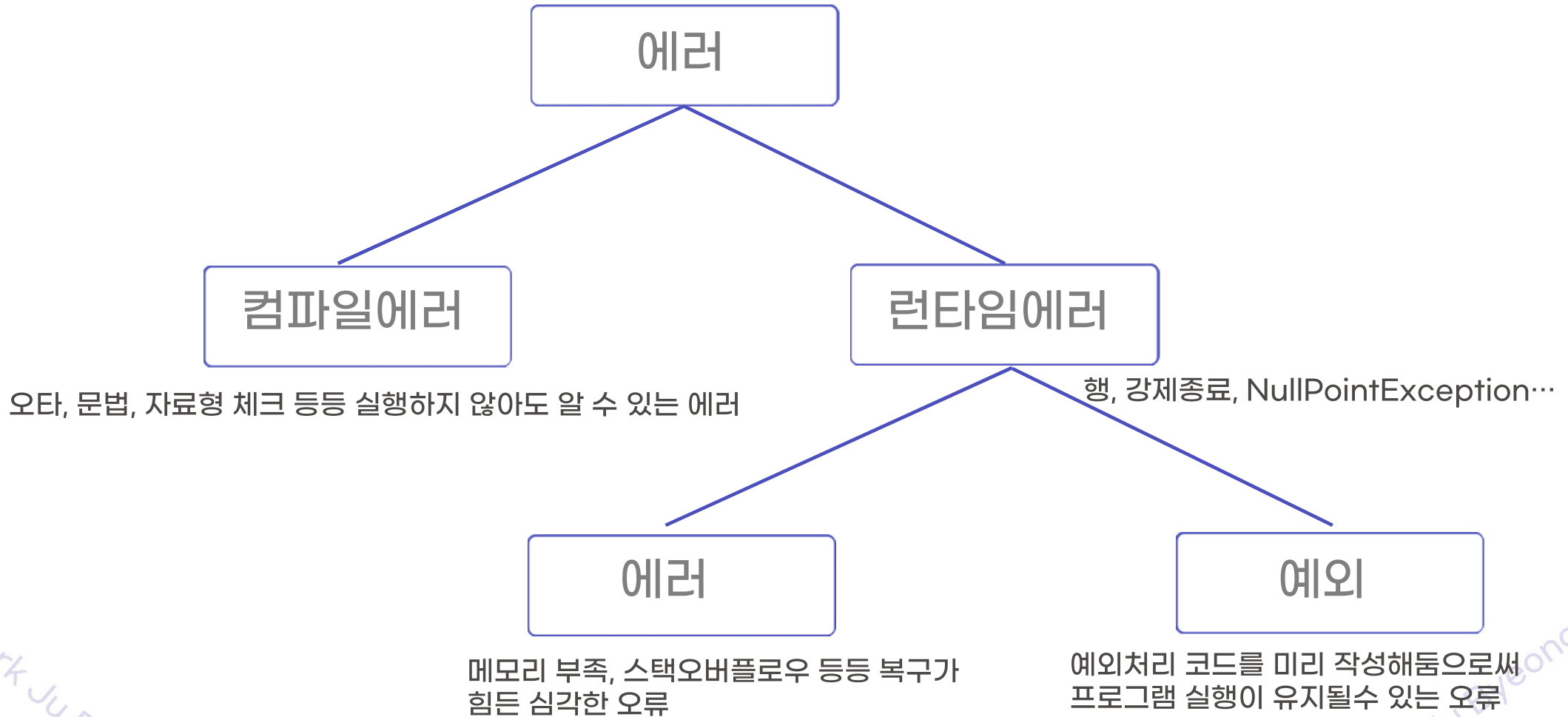


01

에러의 종류



# 에러의 종류

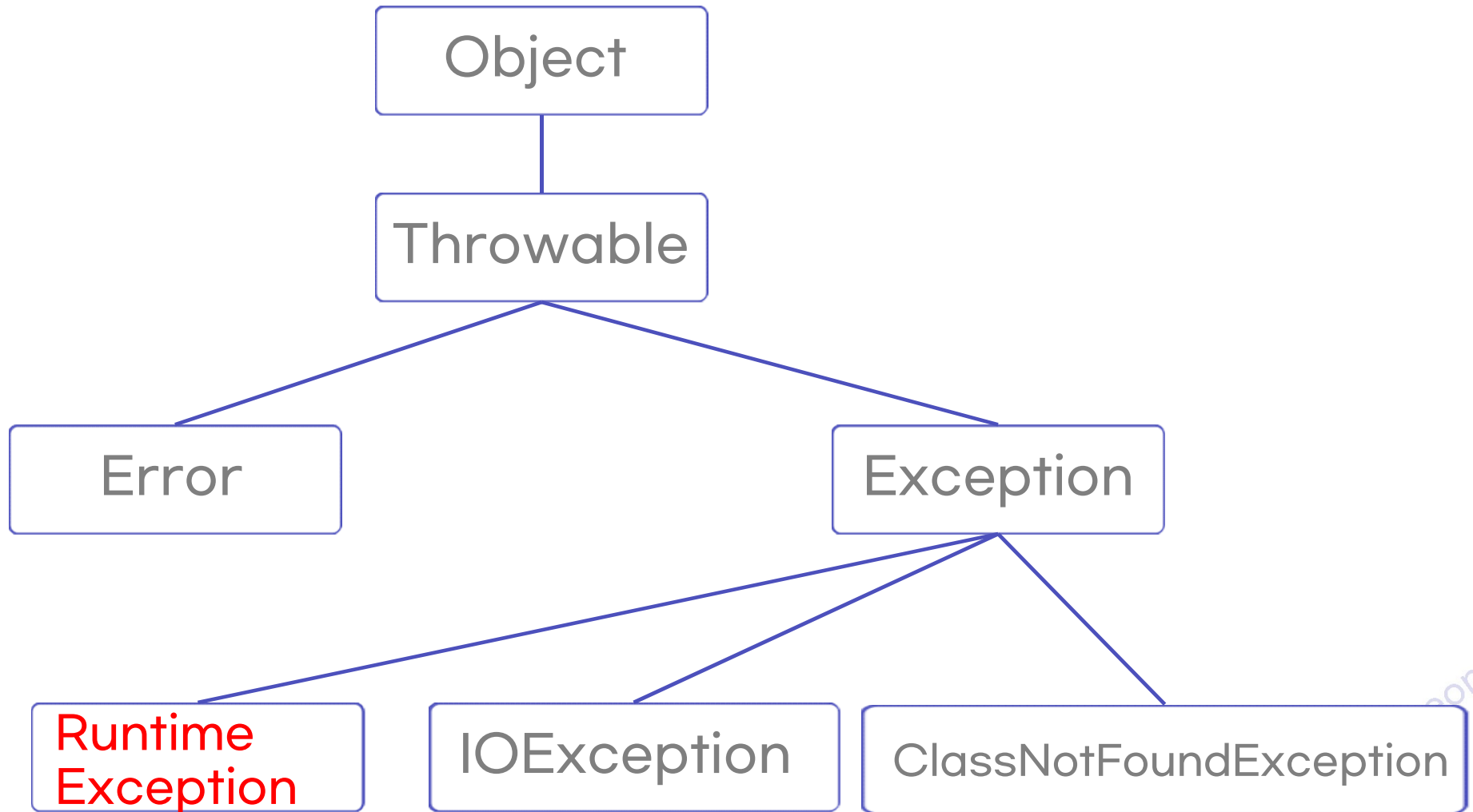


# 에러의 발생타이밍

---



# 에러의 클래스 관계도



# RuntimeException 클래스

```
int[] test = new int[5];  
  
test[5] = 10;
```

```
Problems Javadoc Declaration Search Console X Git Staging History Debug  
<terminated> Main [Java Application] C:\Users\zest1\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_17.0.5.v20221102-0933\jre\bin\javaw.exe (2023. 3. 10. 오후  
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 5  
at JavaLecture/joo.twelve.Main.main(Main.java:12)
```

```
Student[] test = new Student[5];  
  
test[2].name = "학생1";
```

```
Console X Problems Debug Shell  
<terminated> Main [Java Application] C:\Users\zest1\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_17.0.5.v20221102-0933\jre\bin\javaw.exe (2023. 3. 10. 오후  
Exception in thread "main" java.lang.NullPointerException:  
at JavaLecture/joo.twelve.Main.main(Main.java:14)
```

```
int a = 10/0;
```

```
Problems Javadoc Declaration Console X Git Staging History Debug  
<terminated> Main (1) [Java Application] C:\Users\USER545\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_17.0.5.v20221102-0933\jre\bin\javaw.exe (2023. 3. 10. 오후  
Exception in thread "main" java.lang.ArithmeticException: / by zero  
at JavaLecture/joo.twelve.Main.main(Main.java:13)
```

개발자의 실수로 발생하는 에러가 많다.

# 그외 Exception 클래스들

---

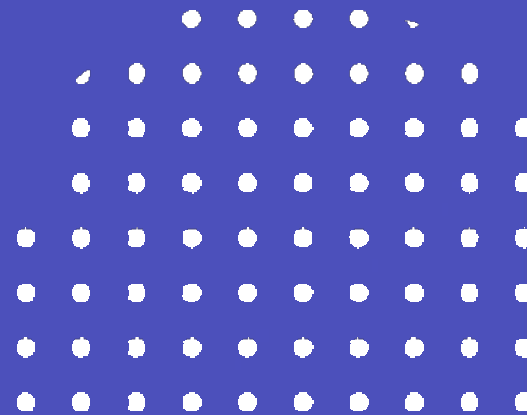
외부적 요인에 의해 주로 발생한다.

1. FileNotFoundException : 외부파일 을 찾지 못함.
2. ClassNotFoundException : 외부에서 클래스파일을 참조시 이름이 잘못
3. DataFormatException : 사용자가 잘못된 데이터를 입력



# — 02

## 예외처리



# try catch문

예외 발생시 별도의 처리를 하지 않으면 프로그램이 비정상 종료된다.

```
try
{
    int a =10/0;
} catch (NullPointerException ex)
{
    System.out.println("객체를 생성하고 쓰세요");
}
catch (ArithmeticException ex)
{
    System.out.println("0으로 나누지 마세요 --");
}
```

1줄 이더라도 중괄호 생략 불가

```
<terminated> Main (1) [Java Application]
0으로 나누지 마세요 --
```

# try catch문 중첩

중첩 사용 가능

변수명 중복

```
try
{
    int a = 10/0;

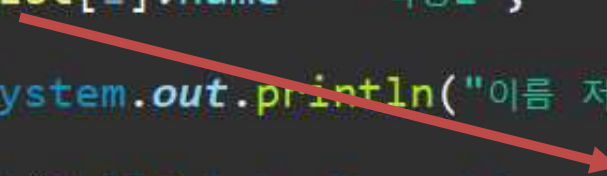
    try
    {
        System.out.println("try catch 중첩 사용가능");
    } catch (Exception ex)
    {
    }

} catch (NullPointerException ex)
{
    try
    {
        System.out.println("try catch 중첩 사용가능");
    } catch (Exception ex)
    {
    }

    System.out.println("객체를 생성하고 쓰세요");
}
catch (ArithmeticException ex)
{
    System.out.println("0으로 나누지 마세요 --");
}
```

# try catch문 실행흐름

```
try
{
    Student[] list = new Student[5];
    list[1].name = "학생1";
    System.out.println("이름 저장 완료!");
} catch (NullPointerException ex)
{
    System.out.println("객체를 생성하고 쓰세요");
} catch (ArithmeticException ex)
{
    System.out.println("0으로 나누지 마세요 ——");
}
System.out.println("try catch문 이후");
```



```
<terminated> main [Java Application]
객체를 생성하고 쓰세요
try catch문 이후
```

# 예외의 최상위 클래스 Exception

```
try
{
    Student[] list = new Student[5];
    list[1].name = "학생1";

    System.out.println("이름 저장 완료!");
} catch (Exception ex)
{
    System.out.println("모든 예외를 다 받는다.");
}
catch (ArithmeticException ex)
{
    System.out.println("0으로 나누지 마세요 ——");
}
```

실행될 일이 없는 코드  
이므로 컴파일 에러

## 예외 클래스의 객체

```
try
{
    Student[] list = new Student[5];
    list[1].name = "학생1";

    System.out.println("이름 저장 완료!");
} catch (Exception ex)
{
    System.out.println("모든 예외를 다 받는다.");
}
```

try 내부에서 예외 발생시 객체를  
생성하여 ex참조변수에 넣어준다.

## printStackTrace()

예외가 발생한 메서드의 정보 및  
예외 메시지를 화면에 출력

## getMessage()

예외클래스의 인스턴스에 저장된  
메시지를 String으로 반환

```
try
{
    System.out.println(1);
    Student[] list = new Student[5];

    System.out.println(2);
    list[1].name = "학생1";

    System.out.println("이름 저장 완료!");
} catch (Exception ex)
{
    ex.printStackTrace();

    System.out.println(ex.getMessage());
}
```

```
1
2
java.lang.NullPointerException: Cannot assign field "name" because "list[1]" is null
    at JavaLecture/joo.twelve.Main.main(Main.java:16)
Cannot assign field "name" because "list[1]" is null
```



# 멀티 catch 블록

```
try
{
    int a = 10/0;
} catch (NullPointerException ex)
{
    System.out.println("객체를 생성하고 쓰세요");
}
catch (ArithmeticException ex)
{
    System.out.println("0으로 나누지 마세요 --");
}
```



```
try
{
    System.out.println(1);
    Student[] list = new Student[5];

    System.out.println(2);
    list[1].name = "학생1";

    System.out.println("이름 저장 완료!");
} catch (NullPointerException | ArithmeticException ex)
{
    ex.printStackTrace();

    System.out.println(ex.getMessage());
}
```



# 멀티 catch 블록

```
try
{
    System.out.println(1);
    Student[] list = new Student[5];

    System.out.println(2);
    list[1].name = "학생1";

    System.out.println("이름 저장 완료!");
} catch (NullPointerException | RuntimeException ex)
{
    ex.printStackTrace();

    System.out.println(ex.getMessage());
}
```

RuntimeException은 NullPointerException의 부모이다.

# 예외 던지기

```
try
{

    int a = 10/1;

    //예외 생성
    ArithmeticException e = new ArithmeticException("1로 나누면 예러 발생!");
    //예외 던지기
    throw e;

} catch (ArithmeticException ex )
{
    ex.printStackTrace();

    System.out.println(ex.getMessage());
}
```

# checked VS unchecked

```
public static void main(String[] args) {  
    // TODO Auto-generated method stub  
  
    int a = 10/1;  
  
    //예외 생성  
    Exception e = new Exception("1로 나누면 에러 발생!");  
    //예외 던지기  
    throw e;  
}
```

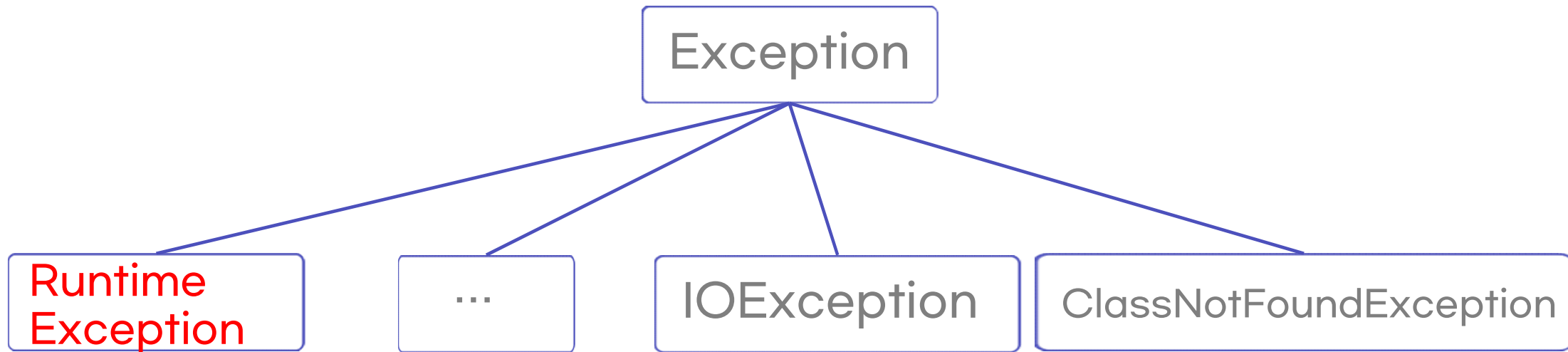
checked 예외 : 반드시 예외처리 해야 한다.

```
public static void main(String[] args) {  
    // TODO Auto-generated method stub  
  
    int a = 10/1;  
  
    //예외 생성  
    RuntimeException e = new RuntimeException("1로 나누면 에러 발생!");  
    //예외 던지기  
    throw e;  
}
```

unchecked 예외 : 컴파일은 통과된다.

# checked VS unchecked

---



```
13
14     int[] test = new int[5];
15
16     test[5] = 10;
17
18
19
20 }
21
22 }
23
```

Problems Javadoc Declaration Search Console X Git Staging History Debug

<terminated> main [Java Application] C:\Users\Wzest1\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86\_64\_17.0.5.v20221102-0933\jre\bin\javaw.exe (2023. 3. 11. 오전 11:00:00)

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 5  
at Javalecture/joo.twelve.Main.main(Main.java:16)

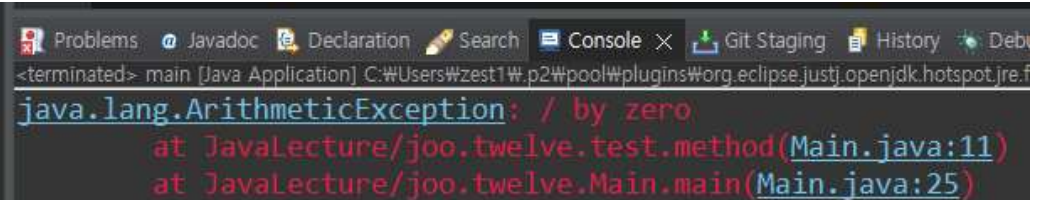
RuntimeException 이기에 그동안 예외처리 없이 컴파일이 가능했다.

# 예외 넘기기

```
void method() throws ArithmeticException, NullPointerException  
{  
    int a = 10/0;  
    System.out.println("1");  
}
```

메서드 내부에서 해당 예외 발생시 메서드를 호출한 쪽으로 예외를 넘긴다.

```
public static void main(String[] args) {  
    // TODO Auto-generated method stub  
  
    try  
    {  
        test t= new test();  
        t.method();  
    } catch (ArithmeticException ex)  
    {  
        ex.printStackTrace();  
    }  
}
```



The screenshot shows the Eclipse IDE's console window. It displays a stack trace for a `java.lang.ArithmeticException: / by zero`. The stack trace indicates the exception was thrown at `Main.java:11` in the `test.method()` call, which was invoked from `Main.java:25` in the `Main.main()` method. The console window has tabs for Problems, Javadoc, Declaration, Search, Console, Git Staging, History, and Debug Console.

```
<terminated> main [Java Application] C:\Users#wzest1\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.f  
java.lang.ArithmeticException: / by zero  
    at JavaLecture/joo.twelve.test.method(Main.java:11)  
    at JavaLecture/joo.twelve.Main.main(Main.java:25)
```

```
void method() throws ArithmeticException, NullPointerException, IOException
{
    int a = 10/0;
    System.out.println("1");
}
```

```
24
25     test t= new test();
26     t.method();
27
28
```

Unhandled exception type IOException



```
try
{
    test t= new test();
    t.method();
}
catch(IOException ex)
{
}
```

unchecked 예외를 넘긴다면 호출하는쪽에서 반드시 예외처리를 해줘야 된다.

```
class test
{
    void method() throws ArithmeticException, NullPointerException, IOException
    {
        int a = 10/0;
        System.out.println("1");
    }
}

public class Main {

    public static void main(String[] args) throws Exception{
        // TODO Auto-generated method stub

        test t= new test();
        t.method();
    }
}
```

method가 IOException을 처리하지 않고 넘겼으나 main 메서드 역시 처리하지 않고 본인을 호출한쪽으로 예외를 넘기고 있다.



# finally

- 예외와 상관없이 반드시 실행되는 코드영역이다.

```
try
{
    System.out.println(1);

    int a = 10/0;
    System.out.println("마무리 작업하는 코드");
}
catch(Exception ex)
{
    System.out.println(2);
    return;
}
System.out.println(3);
```

예외가 발생하면 실행될수 없다.

# finally

```
try
{
    System.out.println(1);

    int a = 10/0;
} catch (Exception ex)
{
    System.out.println(2);
    System.out.println("마무리 작업하는 코드");
    return;
}

System.out.println(3);
```

예외가 발생하지 않으면 실행될 수 없다.

# finally

```
try
{
    System.out.println(1);

    int a = 10/0;
} catch (Exception ex)
{
    System.out.println(2);
    return;
}

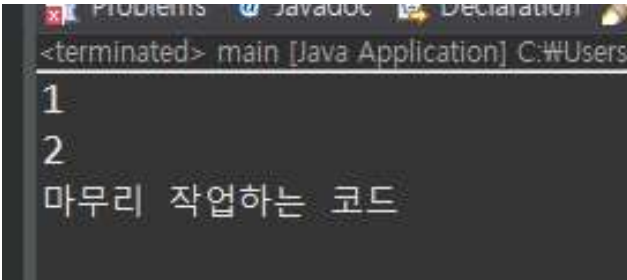
System.out.println(3);
System.out.println("마무리 작업하는 코드");
```

예외가 발생하면 실행 될 수 없다.

# finally

```
try
{
    System.out.println(1);

    int a = 10/0;
} catch (Exception ex)
{
    System.out.println(2);
    return;
} finally
{
    System.out.println("마무리 작업하는 코드");
}
System.out.println(3);
```



```
<terminated> main [Java Application] C:\Users
1
2
마무리 작업하는 코드
```

catch에서 return을 만나더라도 finally는 무조건 실행이 된다.

# finally

```
FileReader fileReader = null;

try
{
    File file = new File("C:\\Users\\zest1\\git\\Java");
    fileReader = new FileReader(file);

    int fileContent=0;
    while((fileContent = fileReader.read()) != -1)
    {
        System.out.print((char)fileContent);
    }
} catch (Exception ex)
{
}

finally
{
    fileReader.close();
}
```

→ close는 반드시 수행되어야 한다

그런데 close 하다가 예외가 발생하면...??

# finally

```
}finally
{

    try
    {
        if(fileReader != null)
            fileReader.close();
    }catch(IOException ex)
    {
        System.out.println("파일 close 예외 발생");
    }

}
```

finally 안에서 다시 예외 처리를 해줘야 한다 ㅠㅠ

# try with resource문

- SDK 1.7부터 사용가능

```
File file = new File("C:\\Users\\zest1\\git\\JavaLecture\\J  
  
try(FileReader fileReader = new FileReader(file))  
{  
  
    int fileContent=0;  
    while((fileContent = fileReader.read()) != -1)  
    {  
        System.out.print((char)fileContent);  
    }  
}catch(Exception ex)  
{  
  
}
```

→ Close()를 자동으로  
호출해준다.

## try with resource문

---

```
public interface AutoCloseable {  
    /**
```

AutoCloseable 인터페이스를 구현한 클래스들만  
try with resource문에 들어갈수 있다.



# 1-1 실습문제 (normal)

아래의 코드가 강제 종료되지 않게 예외처리 해보자.

```
int[] list = new int[5];  
  
for(int i=0 ;i<=5;i++)  
    list[i] = i;
```



Problems Javadoc Search  
<terminated> Main (14) [Java Application] C  
배열의 길이를 넘었습니다.

## 1-1 문제풀이 (normal)

---

Park Ju Byeong

Park Ju Byeong

## 1-2 실습문제 (hard)

1~100 숫자 맞추기 게임 중 숫자가 아닌 값 입력 시 다시 입력하도록 예외처리 해보자

- nextInt() 메서드는 숫자 이외의 값이 들어오면 InputMismatchException이 발생한다.

1~100 랜덤숫자 얻기

```
int answer = (int)(Math.random() * 100) + 1;
```

키보드로 숫자 입력받기

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

```
main [Java Application] C:\Users\zest1\p2\pool\plugins\org.e
```

```
1 ~ 100 사이의 값을 입력하세요 : 한글넣기
```

```
비정상적인 값입니다. 다시입력하세요
```

```
1 ~ 100 사이의 값을 입력하세요 :kkk
```

```
비정상적인 값입니다. 다시입력하세요
```

```
1 ~ 100 사이의 값을 입력하세요 :50
```

```
더 작은 수를 입력하세요
```

```
1 ~ 100 사이의 값을 입력하세요 :25
```

```
더 작은 수를 입력하세요
```

```
1 ~ 100 사이의 값을 입력하세요 :
```

# 1-2 문제풀이 (hard)

---

# 1-3 실습문제 (hard)

Main 클래스에 test 메서드를 아래와 같이 만들어서 사용해보자

- 빈칸 부분을 적절하게 채워 넣어 124561356 이 출력 되도록 실행흐름을 만들어보자
- test 메서드 사용시 반드시 예외처리를 하도록 강제할것

```
public static void test(boolean isThrowYn)
{
    try
    {
        System.out.println(1);
        if(isThrowYn)
        {
            throw ;
        }

        System.out.println(2);

        throw ;
    } catch(RuntimeException ex)
    {
        System.out.println(3);
    }
    catch(Exception ex)
    {
        System.out.println(4);
    }

    {
        System.out.println(5);
    }

    System.out.println(6);
}
```

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

    try
    {
        test( );
        test( );
    } catch(Exception ex)
    {
        System.out.println(7);
    }

}
```

```
<terminated> main [J
124561356
```

## 1-3 문제풀이 (hard)

---

# 03

## 사용자 정의 예외

# 사용자 정의 예외

- 예외 클래스를 상속받아 구현한다.
- 부모가 checked 이면 자식도 checked 이다.

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

```
try  
{  
  
    throw new MyException("내가만든 예외");  
  
} catch (Exception ex)  
{  
    System.out.println(ex.getMessage());  
}
```

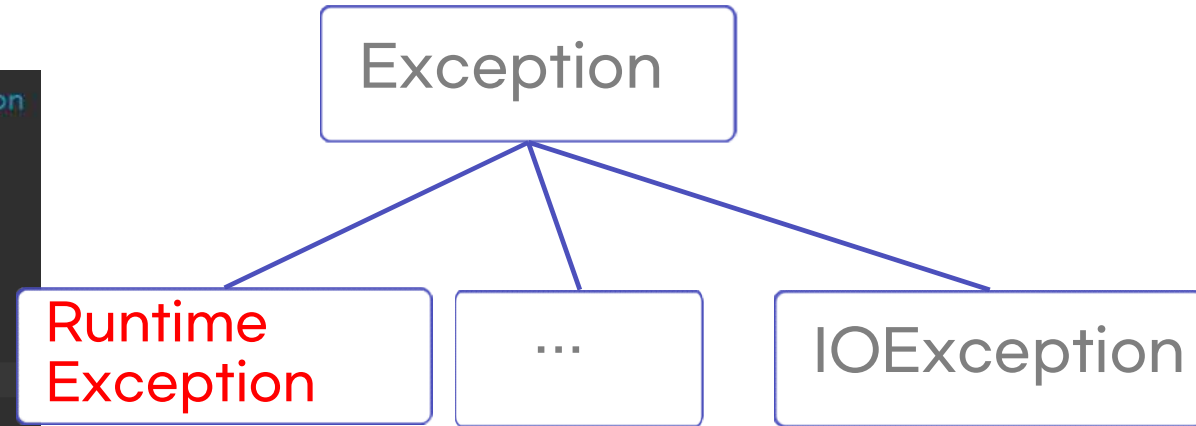
```
<terminated> main [Java  
내가만든 예외
```



# 사용자 정의 예외

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

```
public static void myExceptionTest() throws MyException  
{  
    throw new MyException("내가만든 예외");  
}  
  
public static void main(String[] args) {  
    myExceptionTest();  
}
```



Runtime Exception을 상속받아 만들어 예외처리를 선택적으로 사용할수 있게 만드는데 일반적이다.

그런데 에러를 일부러 생성해서 발생 시킬 이유가 있는가..?

게다가 자체적으로 새롭게 만든 에러까지 만들어서 발생시켜서 어디다 써먹는걸까..?

에러는 만나게 해야하는건데 왜 더 발생시키려는걸까?

# 사용자 정의 예외 활용

- 아래 코드의 문제점을 생각해보자.

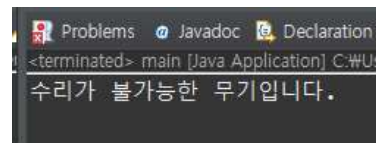
```
public void reparaire(Weapon weapon)
{
    if(weapon instanceof Repairable)
        weapon.durability+=10;
    else
        System.out.println("수리가 불가능한 무기입니다.");
}
```

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

    BlackSmith bs = new BlackSmith();

    Weapon weapon= new Punch(13,20);

    bs.repaire(weapon);
}
```



수리가 불가능 할 때 나는 다른 문구를 쓰고 싶은데... BlackSmith 클래스에서 자기 맘대로 문구를 정해놓고 출력까지 해버리네?

## 사용자 정의 예외 활용

```
public Boolean repair(Weapon weapon)
{
    if(weapon instanceof Repairable)
        weapon.durability+=10;
    else
        return false;
    |
    return true;
}
```

그래 알겠어 나는 수리 가능, 불가능 여부만 boolean으로 반환해줄게  
문구는 사용하는쪽에서 정해!

## 사용자 정의 예외 활용

```
BlackSmith bs = new BlackSmith();  
Weapon weapon= new Punch(13,20);  
Boolean result = bs.repaire(weapon);  
  
if(!result)  
{  
    System.out.println("수리 안됨.");  
}
```

이제 어떤 문구를 어떻게 표시 할 것인지는 BlackSmith에서 제거 되었다.

그런데 수리비용이 부족해서 절반만큼만 수리가 됐으면???... 혹은 그 외에 다양한 수리결과를 받아보고 싶다면...????  
예를들어 수리 대성공으로 인해 공격력이 올라간다던가...?

# 사용자 정의 예외 활용

- 메서드의 반환 데이터로는 메서드 내부의 상황을 알리기 힘들다.

```
public Boolean repaire(Weapon weapon)
{
    if(weapon instanceof Repairable)
        weapon.durability+=10;
    else
        return false;

    return true;
}
```

반환값으로는 다양한 정보를 전달하는 것에 한계가 있다.

물론 그런 정보를 담을 용도의 클래스를 만들어 객체를 돌려주면 가능은 하다.

# 사용자 정의 예외 활용

- 메서드의 return은 일반적으로 데이터를 반환하는 목적이다(메시지전달이 아니다)
- 예외처리를 통해 클래스를 사용하려는 개발자에게 다양한 메시지를 전달할수 있다.

```
public class CanNotRepairException extends RuntimeException
{
    public CanNotRepairException(String msg)
    {
        super(msg);
    }
}

public Boolean repaire(Weapon weapon)
{
    if(weapon instanceof Repairable)
        weapon.durability+=10;
    else
        throw new CanNotRepairException("수리불가");

    return true;
}
```

반환값 보다 더욱 풍성하게 메서드 내부의 상황을 외부에 알려줄수 있다.

```
try
{
    BlackSmith bs = new BlackSmith();
    Weapon weapon= new Punch(13,20);
    Boolean result = bs.repaire(weapon);

    System.out.println("수리가 되어야지만 진행가능한 로직들...");
} catch (BlackSmith.CanNotRepairException ex)
{
    System.out.println("수리가 불가능 합니다.");
}
```

반환값 만으로 외부와 커뮤니케이션 하려면 이 아래부터 IF문이 잔뜩 들어가야 한다.

# 사용자 정의 예외 활용

```
public static void main(String[] args) {  
    try  
    {  
        BlackSmith bs = new BlackSmith();  
  
        Weapon weapon= new Punch(13,20);  
  
        Boolean result = bs.repaire(weapon);  
  
        System.out.println("수리가 되어야지만 진행가능한 로직들...");  
    }catch(BlackSmith.CanNotRepairException ex)  
    {  
        System.out.println("수리가 불가능 합니다.");  
        throw ex;  
    }  
}
```

일부만 처리하고 해당 로직의 상위로 또다시 예외를 던져 객체를 반환 하는 것보다 상당히 유연한 코드 구조를 가질수 있다.

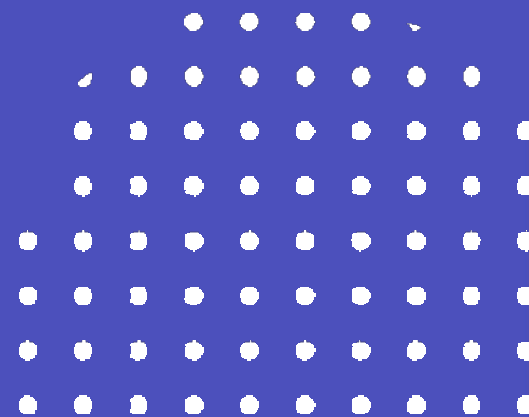


주의점: 정보 전달을 메인 목적으로 예외를 사용하는건 옳지 않다.

외부로 전달하려는 정보들이 프로그래밍 코드로써 대응 해야하는  
문제점들인지 객체를 반환하는것보다 코드의 가독성이 좋아 지는지 등을  
고려 해야 한다!

# — 04

## 실습문제



## 2-1 실습문제 (normal)

MyMath 클래스를 만든후 int add(int a, int b) 메서드를 만들어보자

- 0을 더할경우 직접만든 CanNotAddZeroException 예외를 발생시켜보자(사용자에게 예외처리를 강제하지 말것!)
- CanNotAddZeroException 는 직접만든 사용자 정의 예외이다.

```
MyMath m = new MyMath();  
  
m.add(10, 0);
```

```
<terminated> main [Java Application] C:\Users\zest1\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64.jre\bin\java.exe  
Exception in thread "main" joo.twelve.MyMath$CanNotAddZeroException  
    at JavaLecture/joo.twelve.MyMath.add(MyMath.java:16)  
    at JavaLecture/joo.twelve.Main.실습문제2_2(Main.java:99)  
    at JavaLecture/joo.twelve.Main.main(Main.java:25)
```

## 2-1 문제풀이 (normal)

---

Park Ju Byeong

Park Ju Byeong

## 2-2 실습문제 (normal)

아래의 코드는 에러가 발생한다. 왜 그런지 생각해보고 수정을 해보자

```
try
{
    MyMath m = new MyMath();

    m.minus(10, 0);
} catch (Exception ex)
{
}

} catch (MyMath.CanNotMinusZeroException ex1)
{
}

}
```

## 2-2 문제풀이 (normal)

---

Park Ju Byeong

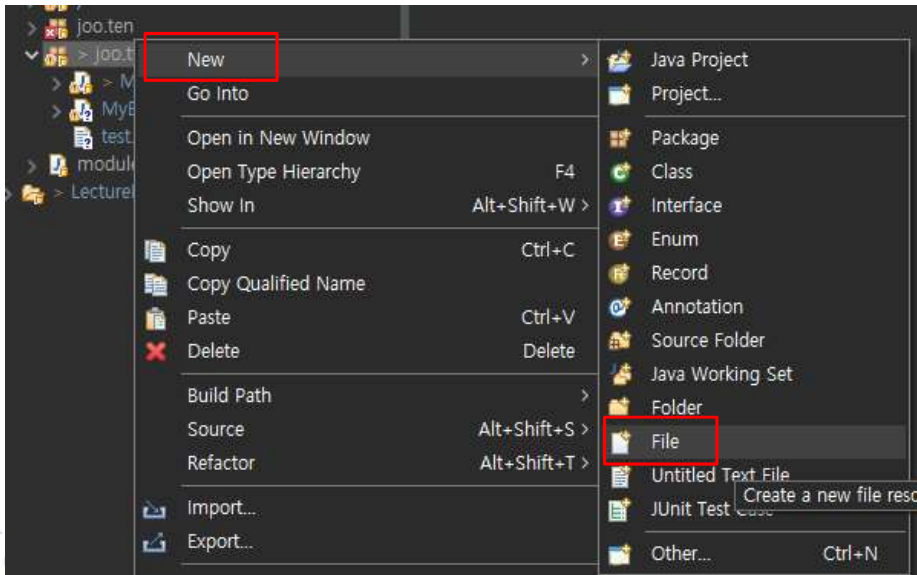
Park Ju Byeong

## 2-3 실습문제 (hard)

test.txt 파일을 읽어서 콘솔화면에 출력해보자.

FileReader 객체는 다 사용하였다면 close() 메서드를 이용하여 닫아 줘야한다.

파일경로: 생성된 파일 우클릭 -> Properties -> 파일 경로 나옴



```
File file = new File("읽어들일 파일경로");
```

```
FileReader fileReader = new FileReader(file)
```

```
int fileContent=0;  
while((fileContent = fileReader.read()) != -1)  
{  
    System.out.print((char)fileContent);  
}
```

파일 내용의 끝이 나올때까지  
1글자씩 가져온다

Park Ju Byeong

## 2-3 문제풀이 (hard)

---

Par,

u

ik Ju Byeong



## 2-3 문제풀이 (hard)

---



# THANK YOU



강사 박주병