

## 10-2. 예외 처리

혼자 공부하는 자바 (신용권 저)

- 시작하기 전에
- 예외 처리 코드
- 예외 종류에 따른 처리 코드
- 예외 떠넘기기
- 키워드로 끝내는 핵심 포인트
- 확인문제

## 시작하기 전에

[핵심 키워드] : 예외 처리, try-catch-finally 블록, 다중 catch 블록, throws 키워드

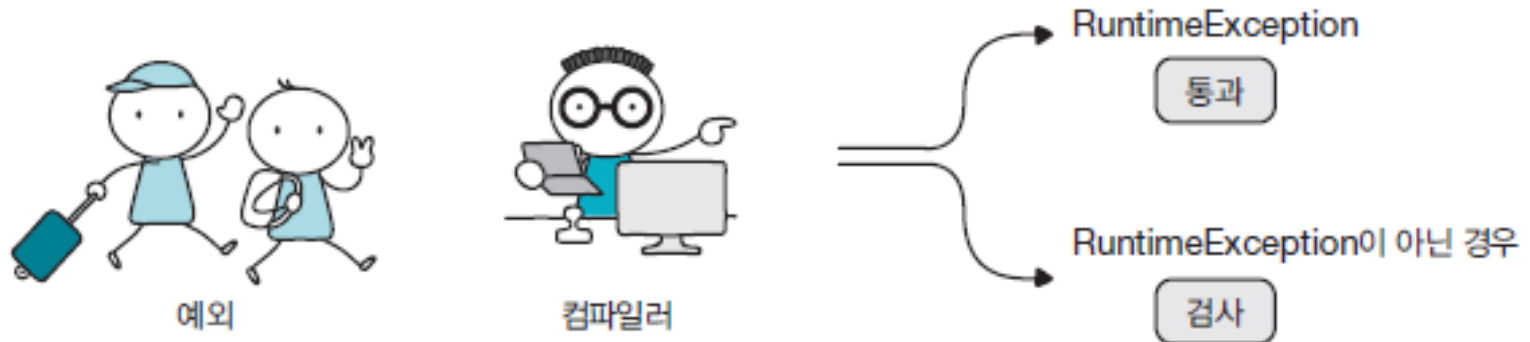
### [핵심 포인트]

프로그램에서 예외가 발생했을 경우 프로그램의 갑작스러운 종료를 막고, 정상 실행을 유지할 수 있도록 예외 처리를 해야 한다. 예외 처리를 하는 방법에 대해 알아본다.

# 시작하기 전에

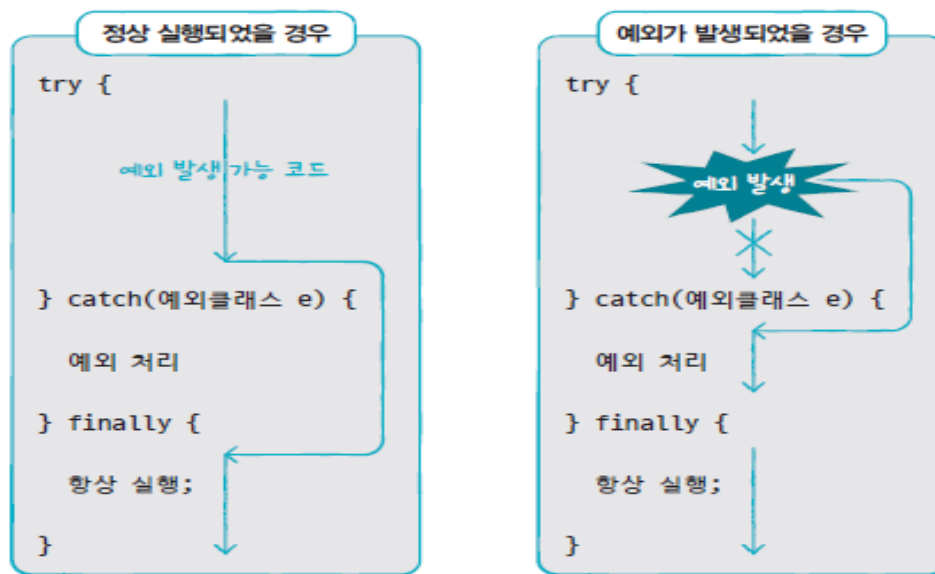
## ❖ 예외 처리 코드

- 자바 컴파일러는 소스 파일 컴파일 시 일반 예외 발생할 가능성이 있는 코드를 발견하면 컴파일 에러를 발생시켜 개발자에게 예외 처리 코드 작성을 요구
- 실행 예외의 경우 컴파일러가 체크하지 않으므로 개발자가 경험을 바탕으로 작성해야 함



## ❖ try-catch-finally 블록

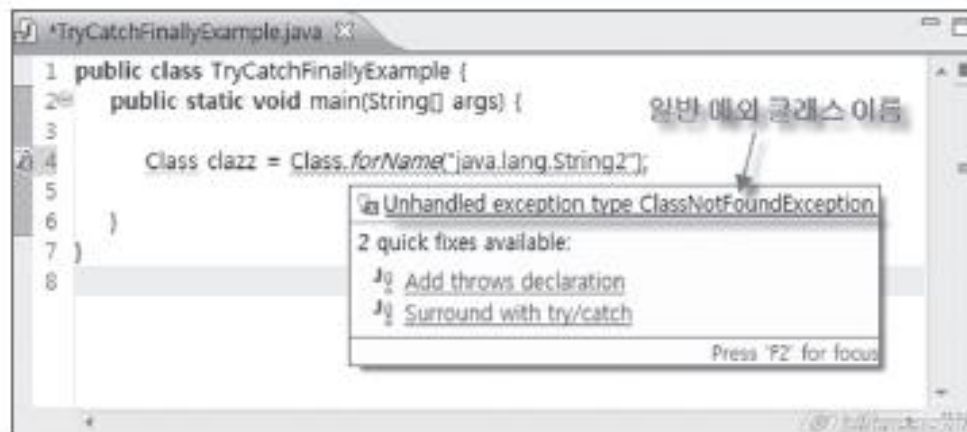
- 생성자 및 메소드 내부에서 작성되어 일반예외와 실행예외가 발생할 경우 예외 처리 가능하게 함



- try 블록에는 예외 발생 가능 코드가 위치
- try 블록 코드가 예외발생 없이 정상실행되면 catch 블록의 코드는 실행되지 않고 finally 블록의 코드를 실행. try 블록의 코드에서 예외가 발생한다면 실행 멈추고 catch 블록으로 이동하여 예외 처리 코드 실행. 이후 finally 블록 코드 실행
- finally 블록은 생략 가능하며, 예외와 무관하게 항상 실행할 내용이 있을 경우에만 작성.

# 예외 처리 코드

- 빨간색 밑줄로 예외 처리 코드 필요성 알림



## ❖ 예시 – 일반 예외 처리

```
01 package sec02.exam01;  
02  
03 public class TryCatchFinallyExample {  
04     public static void main(String[] args) {  
05         try {  
06             Class clazz = Class.forName("java.lang.String2");  
07         } catch (ClassNotFoundException e) {  
08             System.out.println("클래스가 존재하지 않습니다.");  
09         }  
10     }  
11 }
```

실행결과

클래스가 존재하지 않습니다.

# 예외 처리 코드

## ❖ 예시 – 실행 예외 처리

```
01 package sec02.exam02;
02
03 public class TryCatchFinallyRuntimeExceptionExample {
04     public static void main(String[] args) {
05         String data1 = null;
06         String data2 = null;
07         try {
08             data1 = args[0];
09             data2 = args[1];
10         } catch (ArrayIndexOutOfBoundsException e) {
11             System.out.println("실행 매개값의 수가 부족합니다.");
12             return;
13         }
14
15         try {
16             int value1 = Integer.parseInt(data1);
17             int value2 = Integer.parseInt(data2);
18             int result = value1 + value2;
19             System.out.println(data1 + "+" + data2 + "=" + result);
20         } catch (NumberFormatException e) {
21             System.out.println("숫자로 변환할 수 없습니다.");
22         } finally {
23             System.out.println("다시 실행하세요.");
24         }
25     }
26 }
```

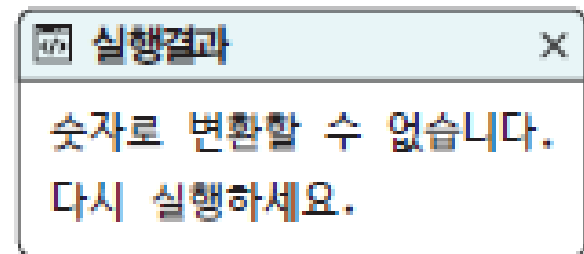
 실행결과

X

실행 매개값의 수가 부족합니다.

## 예외 처리 코드

- 이클립스 – [Run] – [Run Configuration] 메뉴 선택
  - 2개의 실행 매개값 주되 첫 번째 실행 매개값에 숫자가 아닌 문자 넣고 실행
  - 16라인에서 예외 발생
  - 21라인에서 예외처리 후 마지막으로 23라인 실행

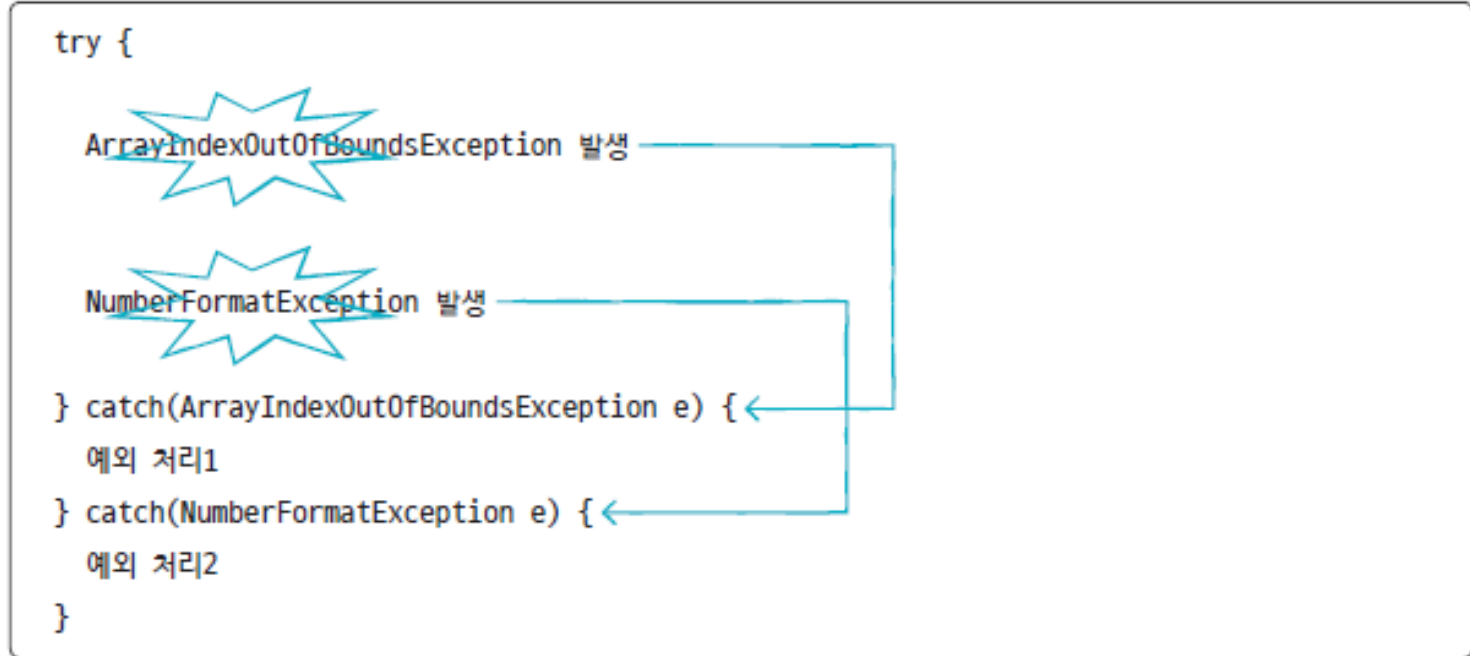




# 예외 종류에 따른 처리 코드

## ❖ 다중 catch

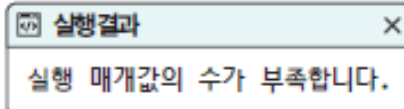
- 발생하는 예외별로 예외 처리 코드를 다르게 하는 다중 catch 블록
- catch 블록의 예외 클래스 타입은 try 블록에서 발생한 예외의 종류 말함
- try 블록에서 해당 타입 예외가 발생하면 catch 블록을 실행



# 예외 종류에 따른 처리 코드

## ❖ 예시



```
01 package sec02.exam03;
02
03 public class CatchByExceptionKindExample {
04     public static void main(String[] args) {
05         try {
06             String data1 = args[0];
07             String data2 = args[1];
08             int value1 = Integer.parseInt(data1);
09             int value2 = Integer.parseInt(data2);
10             int result = value1 + value2;
11             System.out.println(data1 + "+" + data2 + "=" + result);
12         } catch (ArrayIndexOutOfBoundsException e) {
13             System.out.println("실행 매개값의 수가 부족합니다.");
14         } catch (NumberFormatException e) {
15             System.out.println("숫자로 변환할 수 없습니다.");
16         } finally {
17             System.out.println("다시 실행하세요.");
18         }
19     }
20 }
```



# 예외 종류에 따른 처리 코드

## ❖ catch 순서

- 다중 catch 블록 작성 시 상위 예외 클래스가 하위 예외 클래스보다 아래 위치해야 함
- 잘못된 예

```
try {  
     ArrayIndexOutOfBoundsException 발생  
     NumberFormatException 발생  
} catch (Exception e) {  
    예외 처리1  
} catch (ArrayIndexOutOfBoundsException e) {  
    예외 처리2  
}
```

## 예외 종류에 따른 처리 코드

### ■ 올바른 예

```
try {
```

ArrayIndexOutOfBoundsException 발생

다른 Exception 발생

```
} catch(ArrayIndexOutOfBoundsException e) { <
```

예외 처리1

```
} catch(Exception e) { <
```

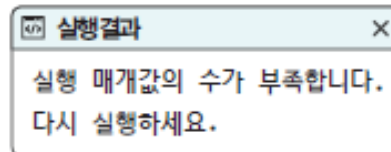
예외 처리2

```
}
```

# 예외 종류에 따른 처리 코드

## ❖ 예시 - catch 블록의 순서

```
01 package sec02.exam01;
02
03 public class CatchOrderExample {
04     public static void main(String[] args) {
05         try {
06             String data1 = args[0];
07             String data2 = args[1];
08             int value1 = Integer.parseInt(data1);
09             int value2 = Integer.parseInt(data2);
10             int result = value1 + value2;
11             System.out.println(data1 + "+" + data2 + "=" + result);
12         } catch (ArrayIndexOutOfBoundsException e) {
13             System.out.println("실행 매개값의 수가 부족합니다.");
14         } catch (Exception e) {
15             System.out.println("실행에 문제가 있습니다.");
16         } finally {
17             System.out.println("다시 실행하세요.");
18         }
19     }
20 }
```



# 예외 떠넘기기

## ❖ throws 키워드

- 메소드 선언부 끝에 작성되어 메소드에서 처리하지 않은 예외를 호출한 곳으로 넘기는 역할
- throws 키워드 뒤에는 떠넘길 예외 클래스를 쉼표로 구분하여 나열

```
리턴타입 메소드이름(매개변수,...) throws 예외클래스1, 예외클래스2, ... {  
}
```

```
리턴타입 메소드이름(매개변수,...) throws Exception {  
}
```

# 예외 떠넘기기

```
public void method1() {  
    try {  
        method2();  
    } catch(ClassNotFoundException e) {  
        //예외 처리 코드  
        System.out.println("클래스가 존재하지 않습니다.");  
    }  
}
```

호출한 곳에서 예외 처리

```
public void method2() throws ClassNotFoundException {  
    Class clazz = Class.forName("java.lang.String2");  
}
```

- method1()에서 try-catch 블록으로 예외 처리하지 않고 throws 키워드로 다시 예외 떠넘기는 경우

```
public void method1() throws ClassNotFoundException {  
    method2();  
}
```

# 예외 떠넘기기

## ❖ 예시 – 예외 처리 떠넘기기

```
01 package sec02.exam02;
02
03 public class ThrowsExample {
04     public static void main(String[] args) {
05         try {
06             findClass();
07         } catch(ClassNotFoundException e) {
08             System.out.println("클래스가 존재하지 않습니다.");
09         }
10     }
11
12     public static void findClass() throws ClassNotFoundException {
13         Class clazz = Class.forName("java.lang.String2");
14     }
15 }
```

- main() 메소드에서 throws 키워드 사용하여 예외 떠넘기는 경우

```
public static void main(String[] args) throws ClassNotFoundException {
    findClass();
}
```



## 키워드로 끝내는 핵심 포인트

- **예외 처리** : 프로그램에서 예외 발생하는 경우 프로그램의 갑작스러운 종료 막고 정상 실행 상태 유지할 수 있도록 처리하는 것.
- **try-catch-finally 블록** : 생성자 내부와 메소드 내부에서 작성되어 일반 예외와 실행 예외 발생하는 경우 예외 처리 할 수 있도록 함
- **다중 catch 블록** : catch 블록이 여러 개이더라도 하나의 catch 블록만 실행함. try 블록에서 동시다발적으로 예외가 발생하지 않고, 하나의 예외 발생했을 때 즉시 실행 멈추고 해당 catch 블록으로 이동하기 때문.
- **throws 키워드** : 메소드 선언부 끝에 작성되어 메소드에서 처리하지 않은 예외를 호출한 곳으로 떠넘기는 역할.

- ❖ try-catch-finally 블록에 대한 설명 중 틀린 것을 고르세요
  - try {} 블록에는 예외가 발생할 수 있는 코드를 작성한다
  - catch {} 블록은 try{} 블록에서 발생한 예외를 처리하는 블록이다
  - try {} 블록에서 return문을 사용하면 finally{} 블록은 실행되지 않는다
  - catch {} 블록은 예외의 종류별로 여러 개를 작성할 수 있다
- ❖ 다음 코드가 실행되었을 때 출력 결과는 무엇입니까?

소스 코드 TryCatchFinallyExample.java

```
01 String[] strArray = { "10", "2a" };
02 int value = 0;
03 for(int i=0; i<=2; i++) {
04     try {
05         value = Integer.parseInt(strArray[i]);
06     } catch(ArrayIndexOutOfBoundsException e) {
07         System.out.println("인덱스를 초과했음");
08     } catch(NumberFormatException e) {
09         System.out.println("숫자로 변환할 수 없음");
10     } finally {
11         System.out.println(value);
12     }
13 }
```

# Thank You !

혼자 공부하는 자바 (신용권 저)