



9장. 예외처리

한림대학교 소프트웨어융합대학 양은샘.

	이것이 자바다 신용권의 Java 프로그래밍 정복
저자	신용권
출판	한빛미디어 2015.1.5
페이지수	1,224 사이즈 183*235mm
판매가	서적 27,000원
구매이벤트	IT독자 설문이벤트 외 6건

	혼자 공부하는 자바 JAVA 8 & 11 지원/무료 동영상 강의 제공
저자	신용권
출판	한빛미디어 2019.6.10.
페이지수	708 사이즈 188*257mm
판매가	서적 21,600원

9장. 예외처리

- ❖ 안녕하세요? 여러분!
- ❖ 오늘은 자바의 예외처리 단원을 학습 합니다.
- ❖ 이번 장에서는
 - 예외처리의 개념과 사용법에 대해 알아보도록 하겠습니다.
 - 자바에서의 예외처리는 프로그램의 정상 작동을 위해 언제나 사용하게 되는 중요한 부분입니다.
- ❖ 지난 시간에 학습한 내용을 리뷰한 후 학습을 시작하도록 하겠습니다.

지난 시간 Review

- ❖ 인터페이스의 역할
- ❖ 인터페이스 선언
- ❖ 인터페이스 구현
- ❖ 인터페이스 사용
- ❖ 타입변환과 다형성
- ❖ 인터페이스 상속
- ❖ 디폴트 메소드와 인터페이스 확장

학습 목차

1절. 예외와 예외 클래스

2절. 실행 예외

3절. 예외 처리 코드

4절. 예외 종류에 따른 처리 코드

5절. 자동 리소스 닫기

6절. 예외 처리 떠넘기기

7절. 사용자 정의 예외와 예외 발생

학습 목표

- ❖ 예외의 종류와 발생 경우를 안다.
- ❖ 예외처리의 의미와 사용법(try-catch-finally, throws)을 안다.
- ❖ 사용자 정의 예외를 선언하고 발생시킬 수 있다.
- ❖ 예외처리를 이용하여 오류를 처리할 수 있다.

에러(Error)와 예외(Exception)

- ❖ 컴퓨터 하드웨어 관련 고장으로 인해 응용프로그램 실행 오류가 발생하는 것은 에러.
- ❖ 그 외 프로그램 자체에서 발생하는 오류는 예외.

- ❖ 오류의 종류
 - 에러(Error)
 - 하드웨어의 잘못된 동작 또는 고장으로 인한 오류.
 - 에러가 발생되면 프로그램 종료.
 - 정상 실행 상태로 돌아갈 수 없음.

 - 예외(Exception)
 - 사용자의 잘못된 조작 또는 개발자의 잘못된 코딩으로 인한 오류.
 - 예외가 발생되면 프로그램 종료.
 - 예외 처리 프로그램 통해 정상 실행 상태 유지 가능.

예외(Exception)의 종류

❖ 일반 예외(Exception)

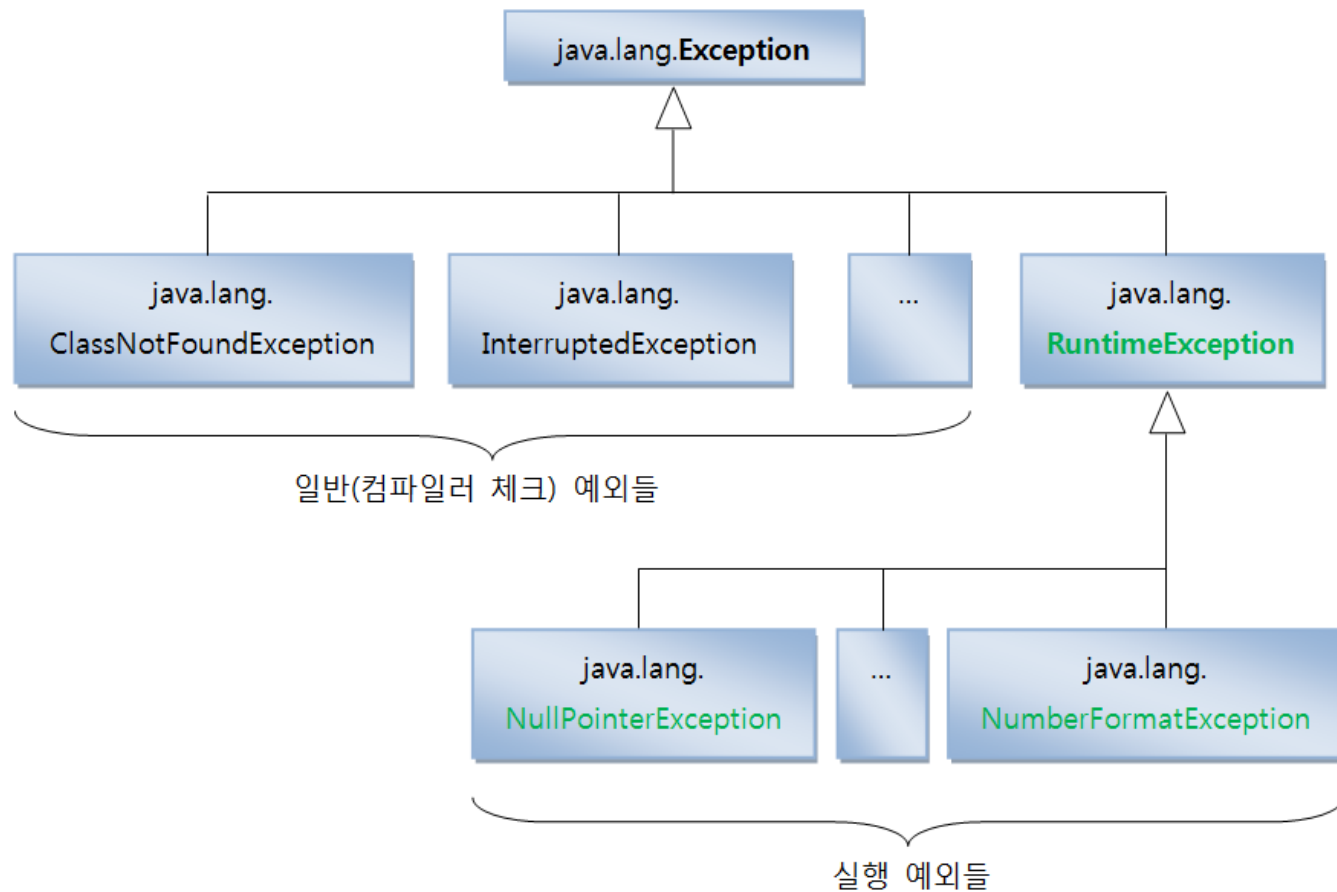
- 컴파일 체크 예외.
- 자바 소스 컴파일 과정에서 해당 예외 처리 코드 있는지 검사하게 됨.
- 예외 처리 코드가 없으면 컴파일 오류 발생.

❖ 실행 예외(RuntimeException)

- 예외 처리 코드를 생략하더라도 컴파일이 되는 예외.
- 실행 시 예측할 수 없이 갑자기 발생하기 때문에 컴파일 과정에서 예외처리코드 검사하지 않음.
- 경험 따라 예외 처리 코드를 작성할 필요가 있음.

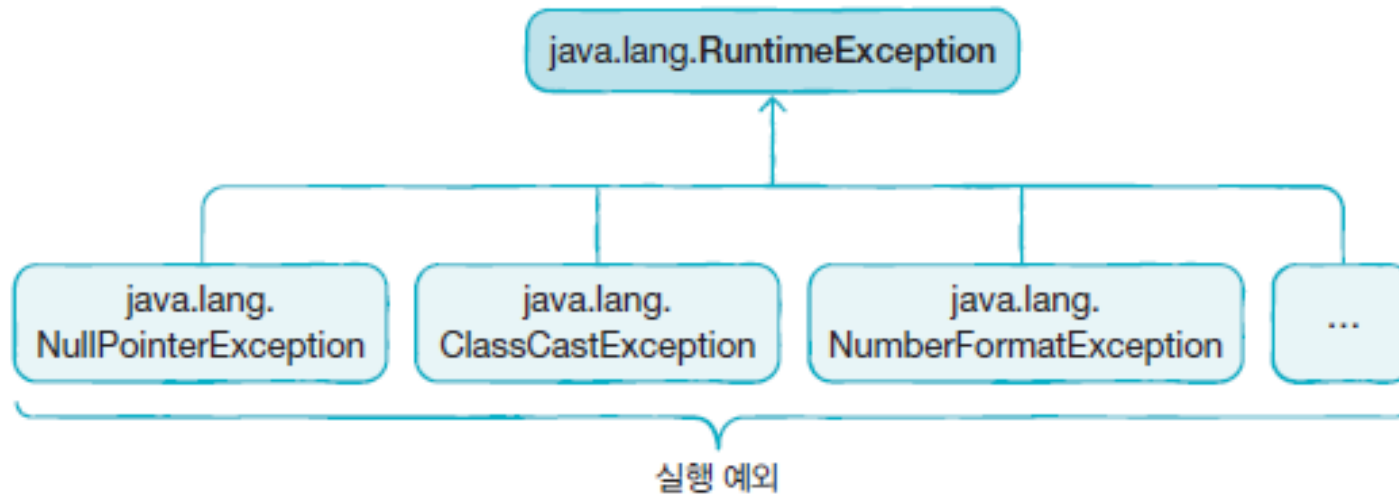
예외 클래스

❖ 자바에서는 예외를 클래스로 관리



실행 예외(RuntimeException)

- ❖ RuntimeException 클래스 기준으로 일반 및 실행 예외 클래스를 구분.
 - 개발자의 경험에 의해서 예외 처리 코드 작성해야 함.
 - 예외처리코드가 없을 경우 해당 예외 발생 시 프로그램 종료.



NullPointerException

- ❖ RuntimeException -> java.lang.NullPointerException
- ❖ 가장 빈번하게 발생하는 실행 예외.
- ❖ 객체 참조가 없는 상태의 참조 변수로 객체 접근 연산자 도트(.)를 사용할 경우 발생.

```
01 package sec01.exam01;
02
03 public class NullPointerExceptionExample {
04     public static void main(String[] args) {
05         String data = null;
06         System.out.println(data.toString());
07     }
08 }
```

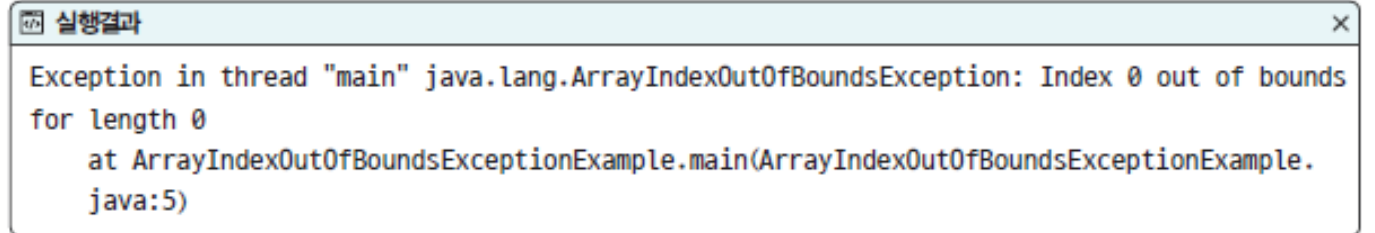
실행결과

```
Exception in thread "main" java.lang.NullPointerException
    at NullPointerExceptionExample.main(NullPointerExceptionExample.java:6)
```

ArrayIndexOutOfBoundsException

- ❖ RuntimeException -> java.lang.ArrayIndexOutOfBoundsException
- ❖ 배열에서 인덱스 범위를 초과할 경우.

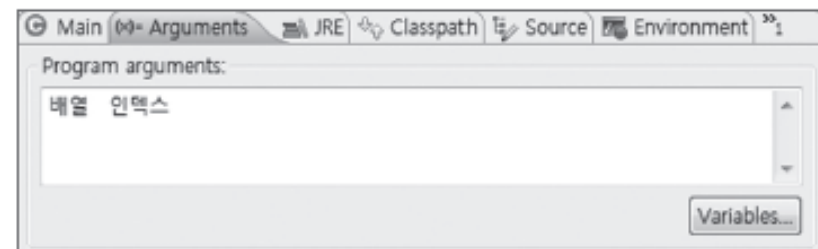
```
01 package sec01.exam02;  
02  
03 public class ArrayIndexOutOfBoundsExceptionExample {  
04     public static void main(String[] args) {  
05         String data1 = args[0];  
06         String data2 = args[1];  
07  
08         System.out.println("args[0]: " + data1);  
09         System.out.println("args[1]: " + data2);  
10     }  
11 }
```



실행결과

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 0 out of bounds for length 0
at ArrayIndexOutOfBoundsExceptionExample.main(ArrayIndexOutOfBoundsExceptionExample.java:5)

- ❖ 이클립스 – [Run] – [Run Configuration] – [Arguments] – [Program arguments]에
- ❖ 아래와 같이 입력하여 해결.



NumberFormatException

- ❖ RuntimeException -> java.lang.NumberFormatException
- ❖ 숫자가 변환될 수 없는 문자가 포함된 경우 java.lang.NumberFormatException 발생.
- ❖ 문자열을 숫자로 변환하는 경우.

리턴 타입	메소드 이름(매개 변수)	설명
int	Integer.parseInt(String s)	주어진 문자열을 정수로 변환해서 리턴
double	Double.parseDouble(String s)	주어진 문자열을 실수로 변환해서 리턴

```
01 package sec01.exam04;
02
03 public class NumberFormatExceptionExample {
04     public static void main(String[] args) {
05         String data1 = "100";
06         String data2 = "a100";
07
08         int value1 = Integer.parseInt(data1);
09         int value2 = Integer.parseInt(data2);    //NumberFormatException 발생
10
11         int result = value1 + value2;
12         System.out.println(data1 + "+" + data2 + "=" + result);
13     }
14 }
```

실행결과

```
Exception in thread "main" java.lang.NumberFormatException: For input string: "a100"
    at java.base/java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)
    at java.base/java.lang.Integer.parseInt(Integer.java:652)
    at java.base/java.lang.Integer.parseInt(Integer.java:770)
    at NumberFormatExceptionExample.main(NumberFormatExceptionExample.java:9)
```

ClassCastException

- ❖ RuntimeException -> ClassCastException
- ❖ 상위 및 하위 클래스 그리고 구현 클래스와 인터페이스 간 타입 변환이 되지 않을 경우에 발생.
- ❖ instanceof 연산자로 타입 변환 가능 여부 확인 필요.



❖ 정상 코드

```
Animal animal = new Dog();  
Dog dog = (Dog) animal;
```

```
RemoteControl rc = new Television();  
Television tv = (Television) rc;
```

```
Animal animal = new Dog() ;  
if(animal instanceof Dog) {  
    Dog dog = (Dog) animal;  
} else if(animal instanceof Cat) {  
    Cat cat = (Cat) animal;  
}
```

❖ 예외 발생 코드

```
Animal animal = new Dog();  
Cat cat = (Cat) animal;
```

```
RemoteControl rc = new Television();  
Audio audio = (Audio) rc;
```

```
Remocon rc = new Audio();  
if(rc instanceof Television) {  
    Television tv = (Television) rc;  
} else if(rc instanceof Audio) {  
    Audio audio = (Audio) rc;  
}
```

ClassCastException 예

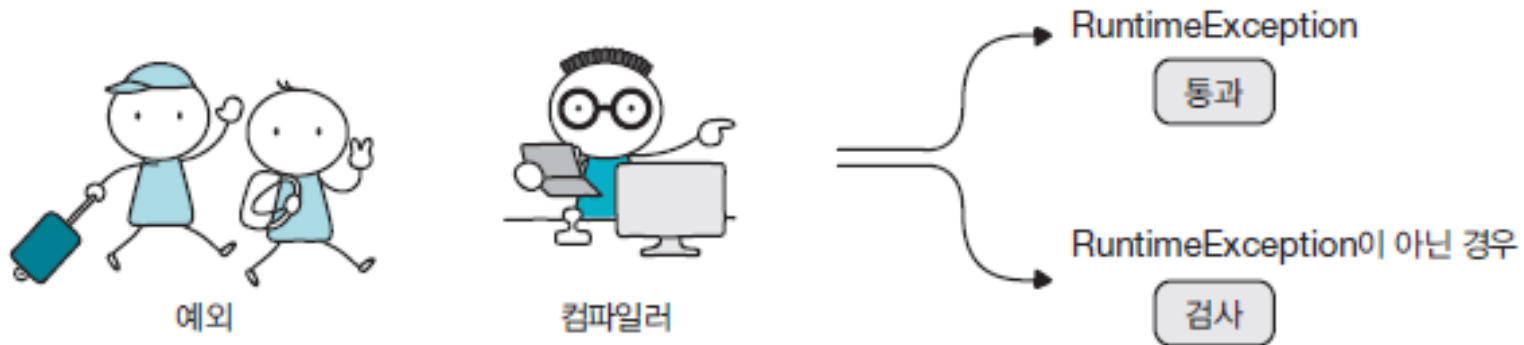
```
01 package sec01.exam05;
02
03 public class ClassCastExceptionExample {
04     public static void main(String[] args) {
05         Dog dog = new Dog();
06         changeDog(dog);
07
08         Cat cat = new Cat();
09         changeDog(cat);
10     }
11
12     public static void changeDog(Animal animal) {
13         //if(animal instanceof Dog) {
14             Dog dog = (Dog) animal;    //ClassCastException 발생 가능
15         //}
16     }
17 }
18
19 class Animal {}
20 class Dog extends Animal {}
21 class Cat extends Animal {}
```

실행결과

Exception in thread "main" java.lang.ClassCastException: class Cat cannot be cast to class Dog (Cat and Dog are in unnamed module of loader 'app')
at ClassCastExceptionExample.changeDog(ClassCastExceptionExample.java:14)
at ClassCastExceptionExample.main(ClassCastExceptionExample.java:9)

예외 처리 코드

- ❖ 자바 컴파일러는 소스 파일 컴파일 시 일반 예외가 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.



예외 처리 코드(try-catch-finally 블록)

❖ try-catch-finally 블록

- 생성자 및 메소드 내부에서 작성되어 일반예외와 실행예외가 발생할 경우 예외처리를 가능하게 함.

❖ try 블록

- 예외 발생 가능 코드가 위치.

❖ try 블록 코드가 예외 발생 없이 정상 실행되면

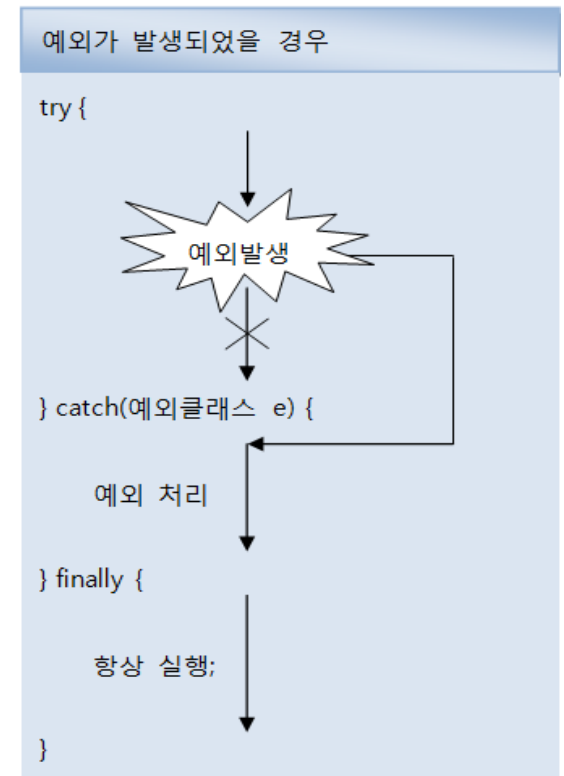
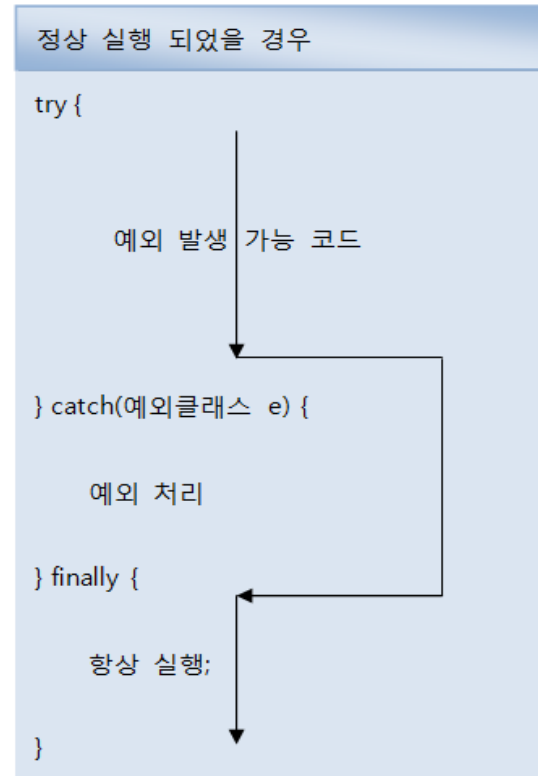
- catch 블록의 코드는 실행되지 않고 finally 블록의 코드를 실행.

❖ try 블록의 코드에서 예외가 발생하면

- 실행 멈추고 catch 블록으로 이동하여 예외 처리 코드 실행. 이후 finally 블록 코드 실행.

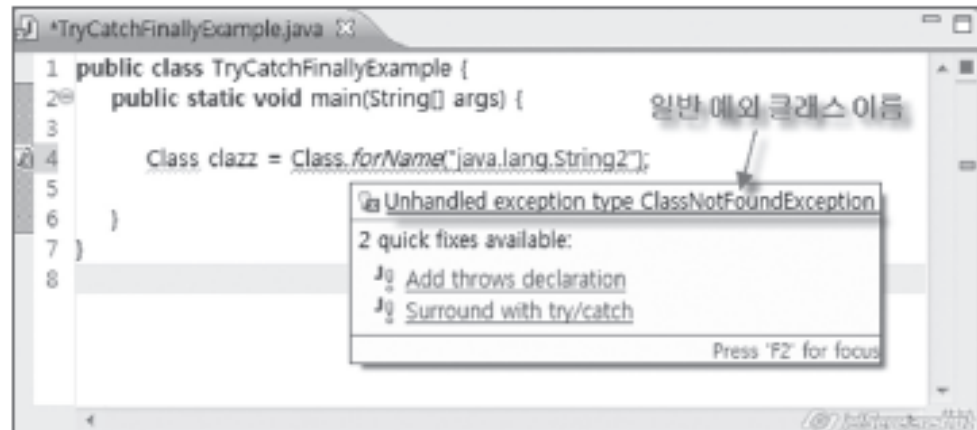
❖ finally 블록

- 생략 가능하며, 예외와 무관하게 항상 실행할 내용이 있을 경우에만 작성.



예외 처리 코드 예

❖ 빨간색 밑줄로 예외 처리 코드 필요성 알림.



❖ 일반 예외 처리 예

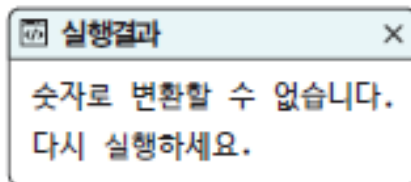
```
01 package sec02.exam01;
02
03 public class TryCatchFinallyExample {
04     public static void main(String[] args) {
05         try {
06             Class clazz = Class.forName("java.lang.String2");
07         } catch(ClassNotFoundException e) {
08             System.out.println("클래스가 존재하지 않습니다.");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

실행결과

클래스가 존재하지 않습니다.

예외 처리 코드 예

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

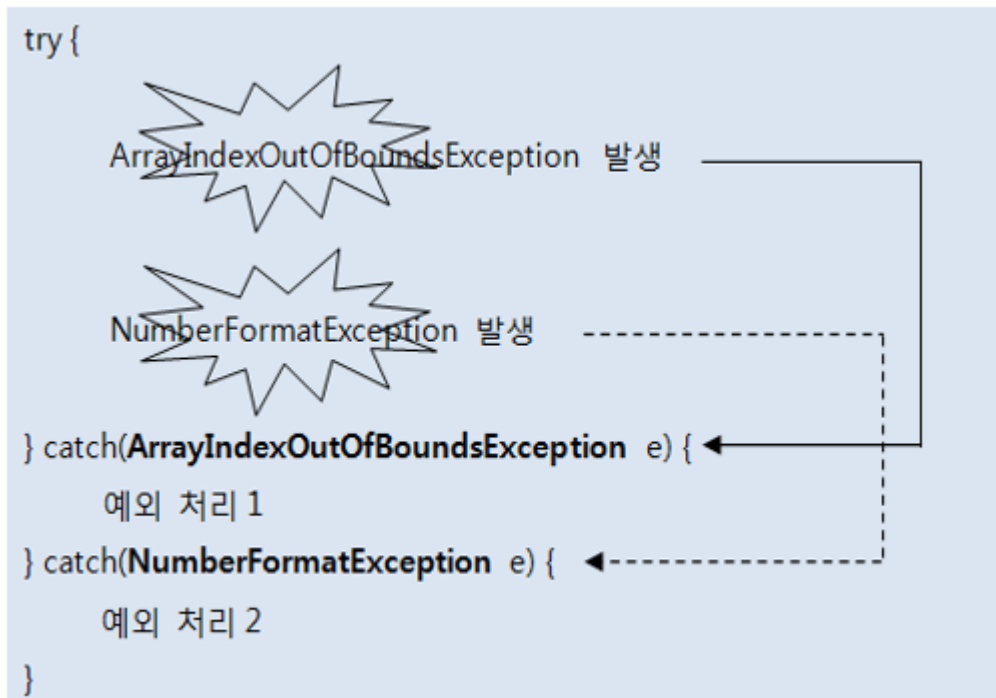


```
01 package sec02.exam02;
02
03 public class TryCatchFinallyRuntimeExceptionExample {
04     public static void main(String[] args) {
05         String data1 = null;
06         String data2 = null;
07         try {
08             data1 = args[0];
09             data2 = args[1];
10         } catch(ArrayIndexOutOfBoundsException 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 }
```

예외 종류에 따른 처리 코드 - 다중 catch

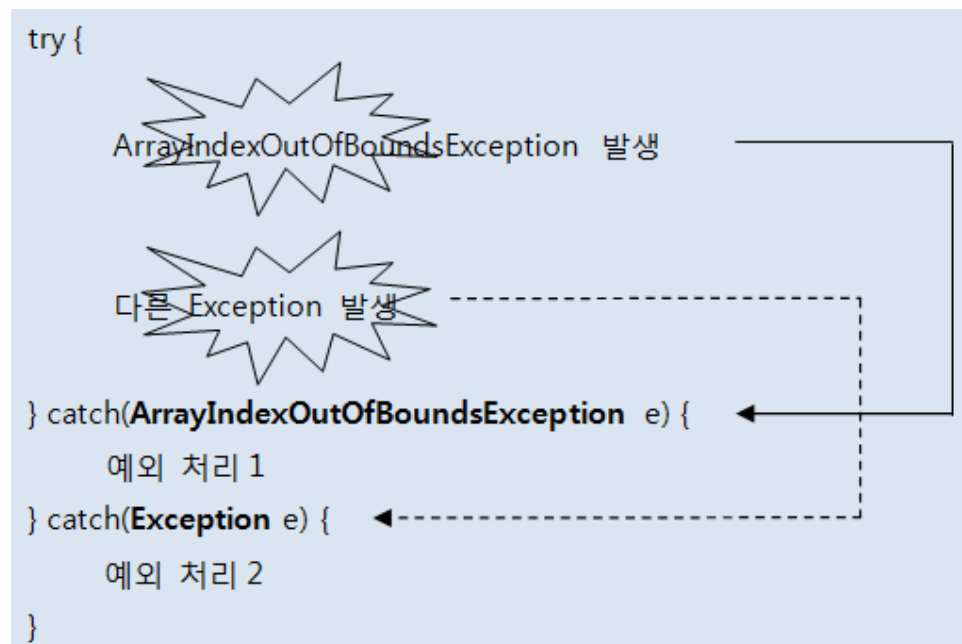
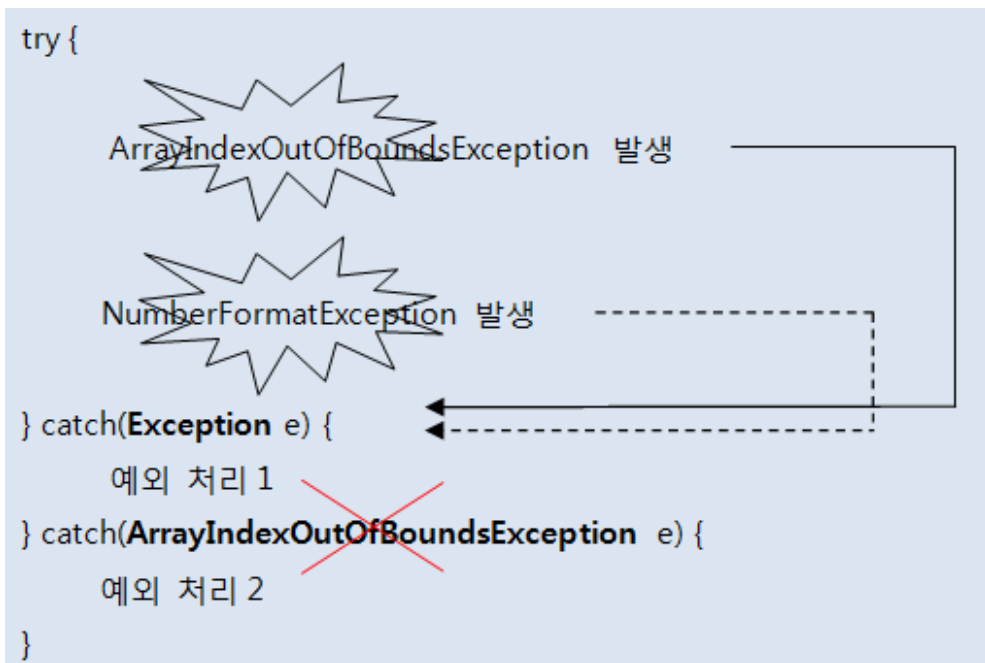
❖ 다중 catch

- 발생하는 예외 별로 예외 처리 코드를 다르게 하는 다중 catch 블록.
- catch 블록의 예외 클래스 타입은 try 블록에서 발생한 예외의 종류를 말함.
- try 블록에서 해당 타입의 예외가 발생하면 catch 블록을 실행.



예외 종류에 따른 처리 코드 – catch 순서

❖ catch 순서 – 상위 클래스가 아래에 위치해야 함.

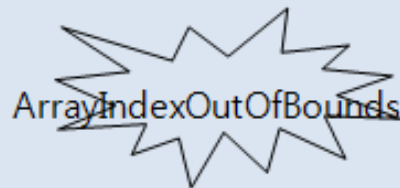


예외 종류에 따른 처리 코드 – multi catch

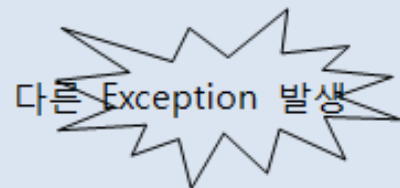
❖ 멀티(multi) catch

- 자바 7부터는 하나의 catch 블록에서 여러 개의 예외 처리가 가능.
 - 동일하게 처리하고 싶은 예외를 | (or연산자) 로 연결

```
try {
```



ArrayIndexOutOfBoundsException 또는 NumberFormatException 발생



다른 Exception 발생

```
} catch(ArrayIndexOutOfBoundsException | NumberFormatException e) {
```

예외 처리 1

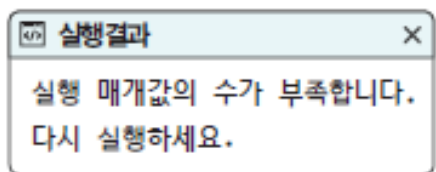
```
} catch(Exception e) {
```

예외 처리 2

```
}
```

catch 블록의 순서 예

```
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 떠 넘기는 역할.
- throws 키워드 뒤에는 떠넘길 예외 클래스를 쉼표로 구분하여 나열.

```
리턴타입 메소드명(매개변수,...) throws 예외클래스 1, 예외클래스 2, ... {  
}
```

```
public void method1() {  
    try {  
        method2();  
    } catch(ClassNotFoundException e) {  
        //예외 처리 코드  
        System.out.println("클래스가 존재하지 않습니다.");  
    }  
}  
  
public void method2() throws ClassNotFoundException {  
    Class clazz = Class.forName("java.lang.String2");  
}
```

호출한 곳에서 예외 처리

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

예외 처리 떠넘기기 예

```
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 }
```

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


사용자 정의 예외 선언

- ❖ 사용자 정의 예외 클래스 선언
 - 자바 표준 API에서 제공하지 않는 예외.
- ❖ 애플리케이션 서비스와 관련된 예외
 - Ex) 잔고 부족 예외, 계좌 이체 실패 예외, 회원 가입 실패 예외....
- ❖ 사용자 정의 예외 클래스 선언 방법
 - 상속을 이용.

```
public class XXXException extends [ Exception | RuntimeException ] {  
    public XXXException() { }  
    public XXXException(String message) { super(message); }  
}
```

사용자 정의 예외 발생

- ❖ 예외를 발생시키는 방법.

```
throw new XXXException()  
throw new XXXException("메시지");
```

- ❖ 발생한 예외는 try-catch-finally를 사용하여 직접 처리도 가능하며,
- ❖ 상위 메소드로 예외 처리를 떠 넘길 수도 있음.

- ❖ 예외 발생 후 호출된 곳으로 예외를 처리를 넘기는 예.

```
public void method() throws XXXException {  
    throw new XXXException("메시지");  
}
```

사용자 정의 예외 예

```
public class BalanceException extends Exception {
    public BalanceException() { }
    public BalanceException(String message) {
        super(message);
    }
}
public class Account {
    private long balance;

    public Account() { }

    public long getBalance() { return balance; }

    public void deposit(int money) { balance += money; }

    public void withdraw(int money) throws BalanceException {
        if(balance < money) {
            throw new BalanceException("잔고부족:"+(money-balance)+" 모자람");
        }
        balance -= money;
    }
}
```

```
public class AccountExample {
    public static void main(String[] args) {
        Account account = new Account();
        //예금하기
        account.deposit(10000);
        System.out.println("예금액: " + account.getBalance());

        //출금하기
        try {
            account.withdraw(30000);
        } catch(BalanceException e) {
            String message = e.getMessage();
            System.out.println(message);
            System.out.println();
            e.printStackTrace();
        }
    }
}
```

예외 정보 얻기

- ❖ getMessage()
 - 예외 발생시킬 때 생성자 매개 값으로 사용한 메시지 리턴

```
throw new XXXException("예외 메시지");
```

- ❖ 원인 세분화하기 위해 예외 코드도 포함 가능(예: 데이터베이스 예외 코드).
- ❖ catch() 절에서 활용

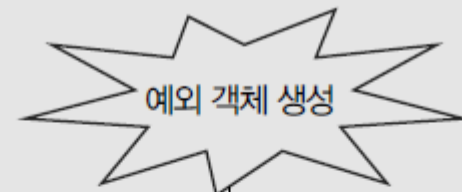
```
} catch(Exception e) {  
    String message = e.getMessage();  
}
```

예외 정보 얻기

❖ printStackTrace()

- 예외 발생 코드를 추적한 내용을 모두 콘솔에 출력.
- 프로그램을 테스트하면서 오류 찾을 때 유용하게 활용.

```
try {
```



```
} catch(예외클래스 e) {  
    //예외가 가지고 있는 Message 얻기  
    String message = e.getMessage();  
  
    //예외의 발생 경로를 추적  
    e.printStackTrace();  
}
```

학습 정리 1

- ❖ 예외 : 사용자의 잘못된 조작 또는 개발자의 잘못된 코딩으로 인해 발생하는 프로그램 오류.
- ❖ 예외 처리 : 프로그램에서 예외가 발생하는 경우 프로그램의 갑작스러운 종료 막고 정상 실행 상태를 유지할 수 있도록 처리하는 것. 예외 발생 시 프로그램이 곧바로 종료되나, 예외 처리 통해 정상 실행 상태를 유지할 수 있음.
- ❖ 예외 클래스 : 자바에서는 예외를 클래스로 관리함. 프로그램 실행 중 예외가 발생하면 해당 예외 클래스로 객체를 생성하고 예외 처리 코드에서 예외 객체를 이용할 수 있도록 해줌.
- ❖ 일반 예외 : 컴파일러 체크 예외. 프로그램 실행 시 예외 발생 가능성 높기 때문에 자바 소스 컴파일 과정에서 해당 예외 처리 코드 있는지 검사함.
- ❖ 실행 예외 : 컴파일러가 체크하지 않는 예외. 실행 시 예측할 수 없이 갑자기 발생하기 때문에 컴파일 과정에서 예외 처리 코드 존재 여부를 검사하지 않음.

학습 정리 2

- ❖ try-catch-finally 블록 : 생성자 내부와 메소드 내부에서 작성되어, 일반 예외 또는 실행 예외가 발생하는 경우 예외를 처리 할 수 있도록 함.
- ❖ 다중 catch 블록 : catch 블록이 여러 개라도 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, 처리하지 않은 예외를 해당 메소드를 호출한 곳으로 떠넘기는 역할.
- ❖ 사용자 정의 예외 선언 방법 : Exception 또는 RuntimeException을 상속 받아 선언.
- ❖ 사용자 정의 예외 발생 방법 : throw new 사용자_정의_예외;
- ❖ 예외 정보를 얻는 방법 : getMessage(), toString() ...

확인 문제 1

- ❖ 예외에 대한 아래 설명 중 틀린 것을 고르세요.
 - 예외는 사용자의 잘못된 조작, 개발자의 잘못된 코딩으로 인한 프로그램 오류를 말한다.
 - RuntimeException의 하위 클래스는 컴파일러가 예외 처리 코드를 체크하지 않는다.
 - 예외는 클래스로 관리된다.
 - Exception의 하위 클래스는 모두 일반 예외에 해당한다.
- ❖ 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.

확인 문제 2

- ❖ 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 }
```

- ❖ 입력 받은 사용자의 이름이 "Daniel"이 아닐 경우, UserNameException을 발생하도록 사용자 정의 예외를 선언하고 예외를 발생시켜 보세요.

예외 중요 소스

❖ java_09장_예외_중요소스.pdf

적용 확인 학습 & 응용 프로그래밍

❖ 다음 파일에 있는 문제들의 해답을 스스로 작성 해 보신 후 개념 & 적용 확인 학습 영상을 학습 하시기 바랍니다.

■ java_09장_예외_ex.pdf

Q & A

- ❖ “예외”에 대한 학습이 모두 끝났습니다.
- ❖ 모든 내용을 이해 하셨나요?
- ❖ 아직 이해가 안되는 내용이 있다면 다시 한번 복습하시기 바랍니다.
- ❖ 질문은 한림 SmartLEAD 쪽지 또는 e-mail 또는 전화상담을 이용하시기 바랍니다.



- ❖ 퀴즈와 과제가 출제되었습니다. 마감시간에 늦지 않도록 주의해 주세요.
- ❖ 수고하셨습니다.^^