

10장 예외 처리

10.1 예외와 예외 클래스

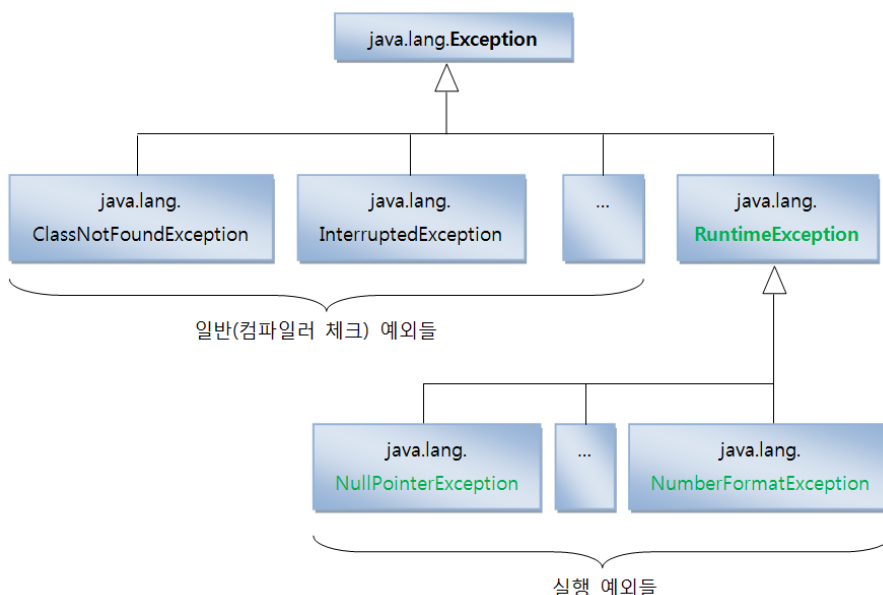
■ 오류의 종류

- 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, 즉 프로그램이 실행되는 동안에 발생하는 예기치 않은 에러를 의미한다.
 - 예외가 발생되면 프로그램 종료
 - 예외 처리 추가하면 정상 실행 상태로 돌아갈 수 있음
- 예외처리(Exception Handling)
 - 예외가 발생했을 경우에 프로그램의 비 정상적인 종료를 막고 정상 실행을 유지할 수 있도록 처리하는 것을 의미한다.

■ 예외의 종류

- 일반(컴파일 체크) 예외(Exception)
 - 예외 처리 코드 없으면 컴파일 오류 발생
- 실행 예외(RuntimeException)
 - 예외 처리 코드를 생략하더라도 컴파일이 되는 예외
 - 경험 따라 예외 처리 코드 작성 필요

■ 예외 클래스



10.2 실행 예외

10.2.1 NullPointerException

- 객체가 없는 상태에서 객체를 사용할 경우에 해당 예외가 발생한다.

[NullPointerExceptionExample.java] NullPointerException이 발생하는 경우

```
package sec02_runtime_exception;

public class NullPointerExceptionExample {
    public static void main(String[] args) {
        String data = null;
        System.out.println(data.toString());
    }
}
```

10.2.2 ArrayIndexOutOfBoundsException

- 배열에서 인덱스 범위 초과하여 사용할 경우에 발생한다.

[ArrayIndexOutOfBoundsExceptionExample.java]

```
package sec02_runtime_exception;

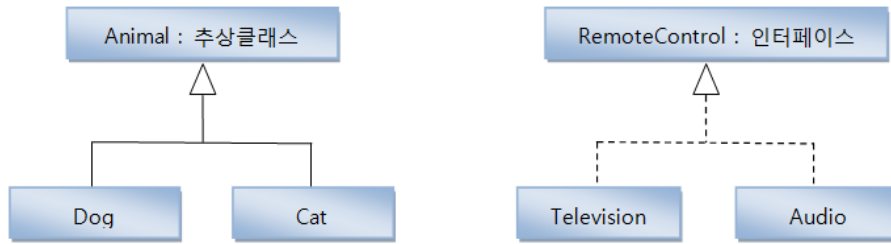
public class ArrayIndexOutOfBoundsExceptionExample {
    public static void main(String[] args) {
        if (args.length == 2) {
            String data1 = args[0];
            String data2 = args[1];
            System.out.println("args[0]: " + data1);
            System.out.println("args[1]: " + data2);
        } else {
            System.out.println("[실행 방법]");
            System.out.print("java ArrayIndexOutOfBoundsExceptionExample ");
            System.out.print("값1 값2");
        }
    }
}
```

10.2.3 NumberFormatException

- 문자열로 되어 있는 데이터를 숫자로 변경하는 경우에 발생한다.

10.2.4 ClassCastException

- 타입 변환(Casting)은 상위 클래스와 하위 클래스 간에 발생하고 구현 클래스와 인터페이스 간에도 발생한다. 이러한 관계가 아니라면 클래스는 다른 클래스로 타입 변환할 수 없다. 억지로 타입 변환을 시도할 경우에 예외가 발생한다.



정상코드

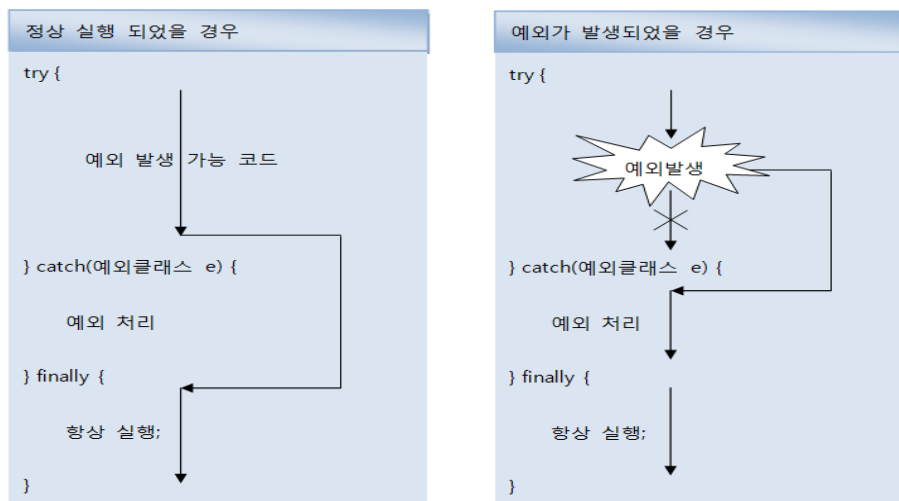
Animal animal = new Dog(); Dog dog = (Dog) animal;	RemoteControl rc = new Television(); Television tv = (Television) rc;
---	--

예외 발생 코드

Animal animal = new Dog(); Cat cat = (Cat) animal;	RemoteControl rc = new Television(); Audio audio = (Audio) rc;
---	---

10.3 예외 처리 코드

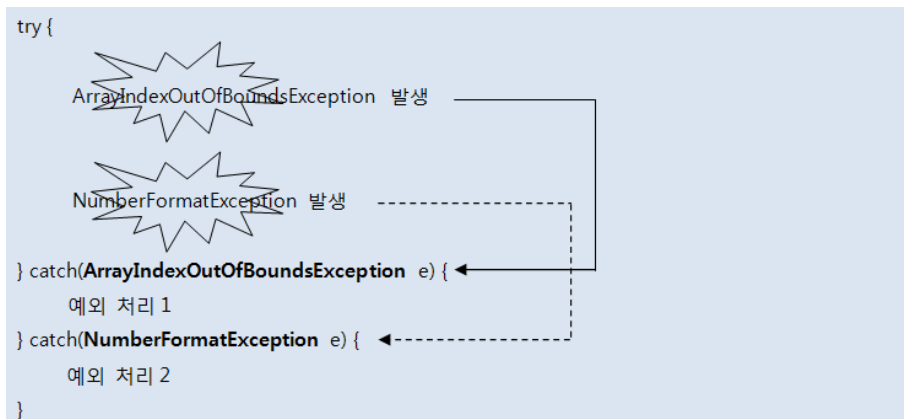
- 예외 발생시 프로그램 종료 막고, 정상 실행 유지할 수 있도록 처리
 - 일반 예외: 반드시 작성해야 컴파일 가능
 - 실행 예외: 컴파일러가 체크해주지 않으며 개발자 경험 의해 작성
- try - catch - finally 블록 이용해 예외 처리 코드 작성
 - try 블록의 코드가 예외 발생 없이 정상 실행되면 catch 블록의 코드는 실행되지 않고 finally 블록의 코드를 실행한다.
 - try 블록의 코드에서 예외가 발생하면 즉시 실행을 멈추고 catch 블록으로 이동하여 예외 처리 코드를 실행한다.
 - try 블록과 catch 블록에서 return문을 사용하더라도 finally 블록은 항상 실행된다.



10.4 예외 종류에 따른 처리 코드

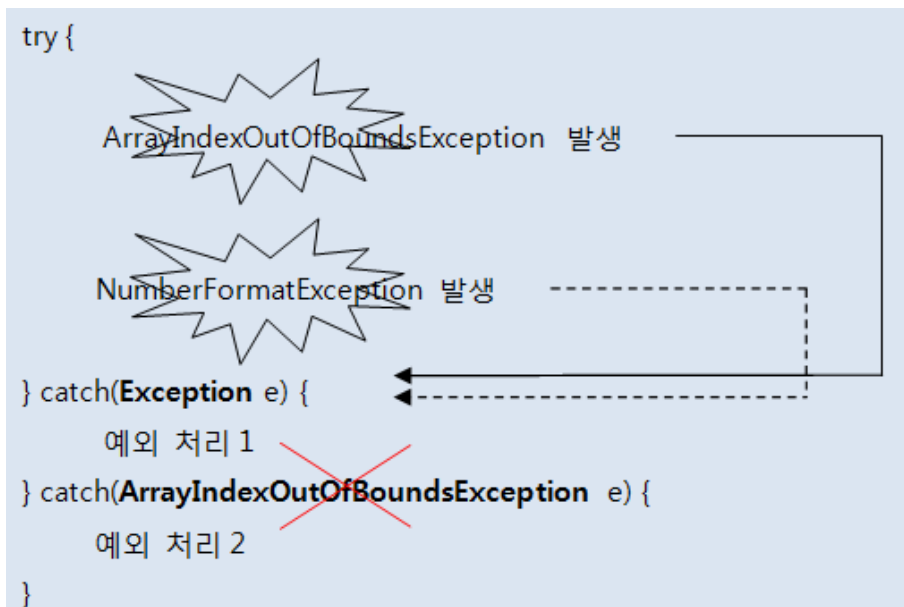
10.4.1 다중 catch

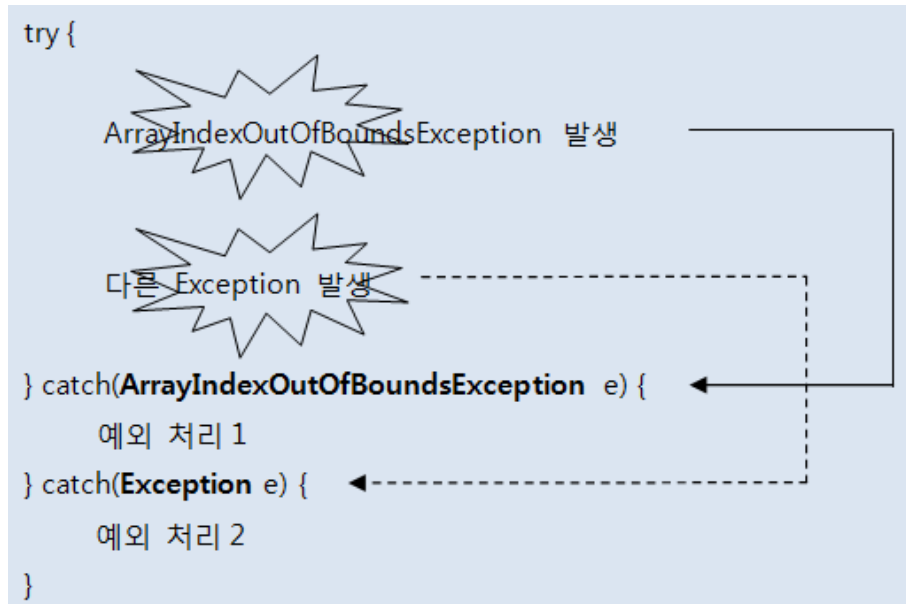
■ 예외 별로 예외 처리 코드 다르게 구현



10.4.2 catch 순서

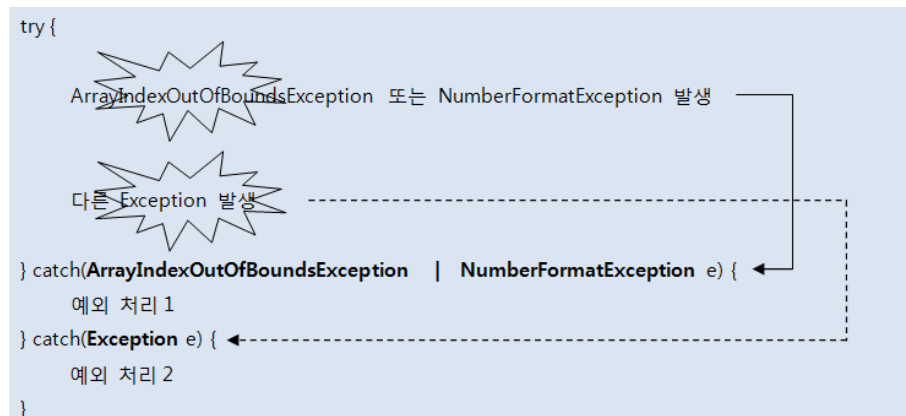
- 다중 catch 블록을 작성할 때 주의할 점은 상위 예외 클래스가 하위 예외 클래스보다 아래쪽에 위치해야 한다.





10.4.3 멀티 catch

- 자바 7부터는 하나의 catch 블록에서 여러 개의 예외 처리 가능 -> 동일하게 처리하고 싶은 예외를 |로 연결한다.



10.5 자동 리소스 닫기

- try-with-resources
 - 예외 발생 여부와 상관 없음
 - 사용했던 리소스 객체의 close() 메소드 호출해 리소스 닫음
 - 리소스 객체
 - 각종 입출력스트림, 서버소켓, 소켓, 각종 채널
 - java.lang.AutoCloseable 인터페이스 구현하고 있어야 함

[FileInputStream.java]

```

package sec05_try_with_resources;

public class FileInputStream implements AutoCloseable {
    
```

```
private String file;

public FileInputStream(String file) {
    this.file = file;
}

public void read() {
    System.out.println(file + "을 읽습니다.");
}

@Override
public void close() throws Exception {
    System.out.println(file + "을 닫습니다.");
}
}
```

[TryWithResourceExample.java]

```
package sec05_try_with_resources;

public class TryWithResourceExample {
    public static void main(String[] args) {
        try (FileInputStream fis = new FileInputStream("file.txt")) {
            fis.read();
            throw new Exception(); // 강제로 예외 발생시킴
        } catch (Exception e) {
            System.out.println("예외 처리 코드가 실행되었습니다.");
        }
    }
}
```

10.6 예외 떠넘기기

- throws
 - 메소드 선언부 끝에 작성
 - throws 키워드 뒤에는 떠넘길 예외 클래스를 쉼표로 구분해서 나열한다.
 - throws Exception만으로 모든 예외를 간단히 떠넘길 수도 있다.

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

- 메소드에서 처리하지 않은 예외를 호출한 곳으로 떠넘기는 역할

```
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");
}
```

10.7 사용자 정의 예외와 예외 발생

10.7.1 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
public class XXXException extends [ Exception | RuntimeException ] {
    public XXXException() {}
    public XXXException(String message) { super(message); }
}
```

[BalanceInsufficientException.java]

```
package sec07_user_define_exception;

public class BalanceInsufficientException extends Exception {
    public BalanceInsufficientException() { }
    public BalanceInsufficientException(String message) {
        super(message);
    }
}
```

10.7.2 예외 발생시키기

- 코드에서 예외 발생시키는 법

```
// 1. 사용자 정의 예외 발생: throw 키워드 뒤에는 예외 객체 생성 코드를 놓는다.
throw new XXXException();
throw new XXXException("메시지");

// 2. 사용자 정의 예외 떠넘기기: throws 키워드로 예외를 떠넘긴다.
public void method() throws XXXException {
    throw new XXXException("메시지");
}

// 3. 사용자 정의 예외 메시지 얻기: throws 키워드를 포함하고 있는 메소드는 호출한 곳에서 try-catch 블록으로
예외 처리를 해주어야 한다.
try {
    method();
} catch(XXXException e) {
    //예외 처리
}
```

[Account.java] 사용자 정의 예외 발생시키기

```
package sec07_user_define_exception;

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 BalanceInsufficientException { // 사용자 정의 예외 떠넘기기
    if(balance < money) { // 사용자 정의 예외 발생
        throw new BalanceInsufficientException("잔고부족: "+(money-balance)+" 모자람");
    }
    balance -= money;
}
}
```

[과제] 사용자가 정의한 예외 사용하기

- 0으로 나눌 경우 '0으로 나누지 마라'라고 알려주는 예외를 만들어 보자. (힌트: Exception 클래스를 상속하여 만든다.)

[MyZeroException.java]

```
package verify.exam00;
...
```

[UsingEx.java]

```
package verify.exam00;
public class UsingEx
{
    public int add(int a,int b){
        return (a+b);
    }
    public int sub(int a,int b) throws MyZeroException{
        if(b==0){throw new MyZeroException("0으로 나누지 마라");}
        return (a/b);
    }
    public int subs(int a,int b) throws MyZeroException{
        return sub(a,b);
    }
    public int adds(int a,int b){
        return (a/b);
    }
}
```

[UsingExMain.java]

```
package verify.exam00;
public class UsingExMain
{
    public static void main(String[] args)
    {
        UsingEx exe=new UsingEx();
        //System.out.println(exe.adds(5,0));
        //System.out.println(exe.sub(5,6));//unreported exception;must be caught
    }
}
```



```

        try{
            System.out.println(exe.sub(5,2));
            System.out.println(exe.subs(5,3));
            System.out.println(exe.sub(5,0));
        }catch(MyZeroException ee){
            System.out.println(ee);
        }
    }
}

```

[실행결과]

```

2
1
verify.exam00.MyZeroException: 0으로 나누지 마라

```

10.8 예외 정보 얻기

■ getMessage()

- 예외 발생시킬 때 생성자 매개값으로 사용한 메시지 리턴

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

- 원인 세분화하기 위해 예외 코드 포함(예: 데이터베이스 예외 코드)
- catch() 절에서 활용

```

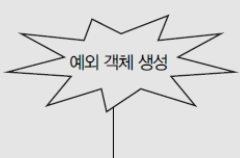
} catch(Exception e) {
    String message = e.getMessage();
}

```

■ printStackTrace()

- 예외 발생 코드 추적한 내용을 모두 콘솔에 출력
- 프로그램 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();
}

```

[AccountExample.java] 사용자 정의 예외 발생시키기

```

package sec07_user_define_exception;

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 (BalanceInsufficientException e) {
    String message = e.getMessage();
    System.out.println(message);
    System.out.println();
    e.printStackTrace();
}
}
```

[과제] 확인문제

1. 예외에 대한 설명 중 틀린 것은 무엇입니까?

- (1) 예외는 사용자가 잘못된 조작, 개발자의 잘못된 코딩으로 인한 프로그램 오류를 말한다.
- (2) RuntimeException의 하위 예외는 컴파일러가 예외 처리 코드를 체크하지 않는다.
- (3) 예외는 try-catch 블록을 사용해서 처리된다.
- (4) 자바 표준 예외만 프로그램에서 처리할 수 있다.

2. try-catch-finally 블록에 대한 설명 중 틀린 것은 무엇입니까?

- (1) try {} 블록에는 예외가 발생할 수 있는 코드를 작성한다.
- (2) catch {} 블록은 try {} 블록에서 발생한 예외를 처리하는 블록이다.
- (3) try {} 블록에서 return문을 사용하면 finally {} 블록은 실행되지 않는다.
- (4) catch {} 블록은 예외의 종류별로 여러 개를 작성할 수 있다.

3. throws에 대한 설명으로 틀린 것은 무엇입니까?

- (1) 생성자나 메소드의 선언 끝 부분에 사용되어 내부에서 발생한 예외를 떠넘긴다.
- (2) throws 뒤에는 떠넘겨야 할 예외를 쉼표(,)로 구분해서 기술한다.
- (3) 모든 예외를 떠넘기기 위해 간단하게 throws Exception으로 작성할 수 있다.
- (4) 새로운 예외를 발생시키기 위해 사용된다.

4. throw에 대한 설명으로 틀린 것은 무엇입니까?

- (1) 예외를 최초로 발생시키는 코드이다.
- (2) 예외를 호출한 곳으로 떠넘기기 위해 메소드 선언부에 작성된다.
- (3) throw로 발생한 예외는 일반적으로 생성자나 메소드 선언부에 throws로 떠넘겨진다.
- (4) throw 키워드 뒤에는 예외 객체 생성 코드가 온다.

5. 다음과 같은 메소드가 있을 때 예외를 잘못 처리한 것은 무엇입니까?

```
public void method1() throws NumberFormatException, ClassNotFoundException { ... }
```

- (1) try { method1(); } catch(Exception e) {}
- (2) void method2() throws Exception { method1(); }
- (3) try { method1(); } catch(Exception e) {} catch(ClassNotFoundException e) {}
- (4) try { method1(); } catch(ClassNotFoundException e) {} catch(NumberFormatException e) {}

6. 다음 코드가 실행되었을 때 출력 결과는 무엇입니까?

[TryCatchFinallyExample.java]

```

01  package verify.exam06;
02
03  public class TryCatchFinallyExample {
04      public static void main(String[] args) {
05          String[] strArray = { "10", "2a" };
06          int value = 0;
07          for(int i=0; i<=2; i++) {
08              try {
09                  value = Integer.parseInt(strArray[i]);
10              } catch(ArrayIndexOutOfBoundsException e) {
11                  System.out.println("인덱스를 초과했음");
12              } catch(NumberFormatException e) {
13                  System.out.println("숫자로 변환할 수 없음");
14              } finally {
15                  System.out.println(value);
16              }
17          }
18      }
19  }
    
```