

14. 예 외 처 리

오류란 무엇인가요?

- 컴파일 오류(compile error) : 프로그램 코드 작성 중 발생하는 문법적 오류
- 실행 오류(runtime error) : 실행 중인 프로그램이 의도 하지 않은 동작을 하거나(bug) 프로그램이 중지되는 오류
- 실행 오류 시 비정상 종료는 서비스 운영에 치명적
- 오류가 발생할 수 있는 경우에 로그(log)를 남겨 추후 이를 분석하여 원인을 찾아야 함
- 자바는 예외 처리를 통하여 프로그램의 비정상 종료를 막고 log를 남길 수 있음

오류와 예외 클래스

- **시스템 오류(error) : 가상 머신에서 발생, 프로그래머가 처리 할 수 없음**
 - 동적 메모리 없는 경우, 스택 오버 플로우 등
- **예외 (Exception) : 프로그램에서 제어 할 수 있는 오류**
 - 읽어 들이려는 파일이 존재하지 않는 경우, 네트워크 연결이 끊어진 경우



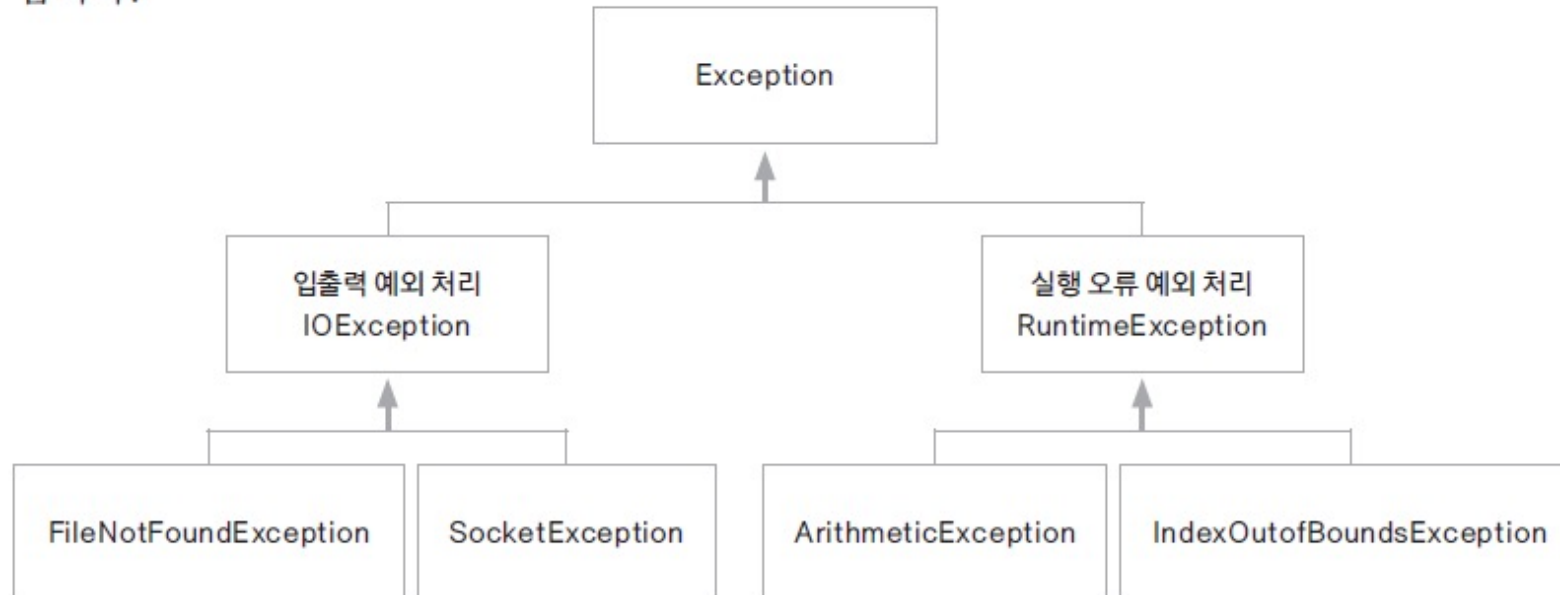
예외 클래스의 종류

- 모든 예외 클래스의 최상위 클래스는 Exception
- 다양한 예외 클래스가 제공 됨

Class Exception

```
java.lang.Object  
    java.lang.Throwable  
        java.lang.Exception
```

입니다.



예외 처리하기

- try-catch 문

```
try {  
    예외가 발생할 수 있는 코드 부분  
} catch(처리할 예외 타입 e) {  
    try 블록 안에서 예외가 발생했을 때 예외를 처리하는 부분  
}
```

- try- catch 문 사용

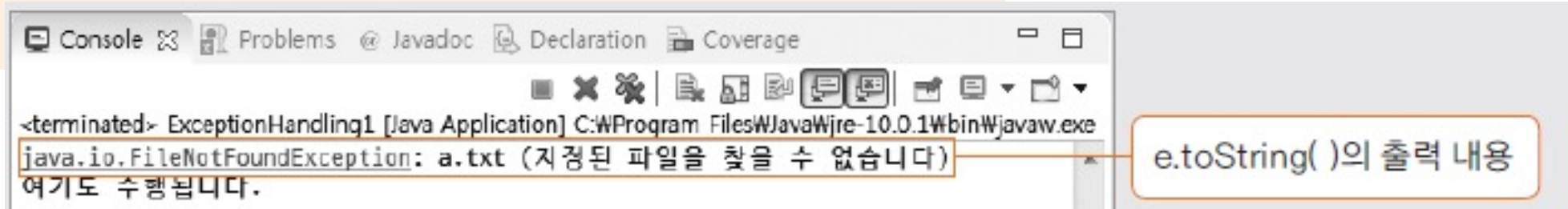
```
try {  
    for(int i = 0; i <= 5; i++) {  
        arr[i] = i;  
        System.out.println(arr[i]);  
    }  
} catch(ArrayIndexOutOfBoundsException e) {  
    System.out.println(e);  
    System.out.println("예외 처리 부분");  
}
```

예외가 발생할 수 있으므로 try 블록에 작성

예외가 발생하면 catch 블록 수행

try-catch문 예제

```
public class ExceptionHandling1 {  
    public static void main(String[] args) {  
        try {  
            FileInputStream fis = new FileInputStream("a.txt");  
        } catch (FileNotFoundException e) {  
            System.out.println(e); //예외 클래스의 toString( ) 메서드 호출  
        }  
        System.out.println("여기도 수행됩니다."); //정상 출력  
    }  
}
```



e.toString()의 출력 내용

- 비정상 종료되지 않아 “여기도 수행됩니다” 부분 출력 됨

try-catch-finally 문

- finally 에서 프로그램 리소스를 정리 함
- try{} 블록이 실행되면 finally{} 블록은 항상 실행 됨
- 리소스를 정리하는 코드를 각 블록에서
- 처리하지 않고 finally에서 처리 함

```
try {  
    예외가 발생할 수 있는 부분  
} catch(처리할 예외 타입 e) {  
    예외를 처리하는 부분  
} finally {  
    항상 수행되는 부분  
}
```

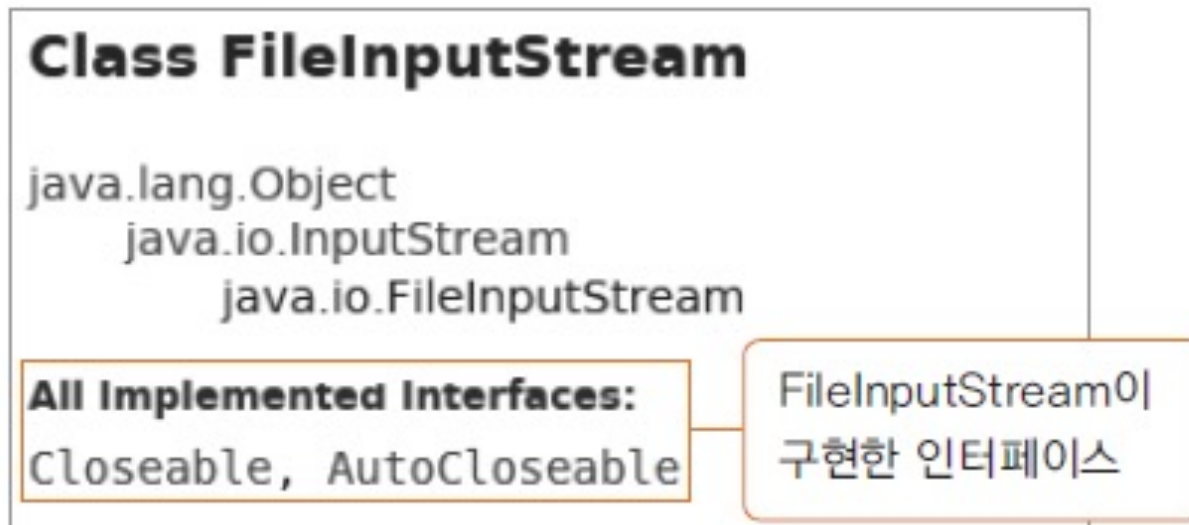
try-catch-finally문 예제

```
public static void main(String[ ] args) {  
    FileInputStream fis = null;  
  
    try {  
        fis = new FileInputStream("a.txt");  
    } catch (FileNotFoundException e) {  
        System.out.println(e);  
        return;  
    } finally {  
        if(fis != null) {  
            try {  
                fis.close( );  
            } catch (IOException e) {  
                //TODO Auto-generated catch block  
                e.printStackTrace( );  
            }  
        }  
        System.out.println("항상 수행됩니다.");  
    }  
    System.out.println("여기도 수행됩니다.");  
}
```

파일 입력 스트림 닫기

try-with-resources문

- 리소스를 자동 해제 하도록 제공하는 구문
- 자바 7 부터 제공 됨
- `close()`를 명시적으로 호출하지 않아도 `try{}` 블록에서 열린 리소스는 정상적인 경우, 예외 발생한 경우 모두 자동 해제 됨
- 해당 리소스가 `AutoCloseable`을 구현 해야 함
- `FileInputStream` 의 경우 `AutoCloseable`을 구현 하고 있음



AutoCloseable 인터페이스

- AutoCloseable 인터페이스 구현한 클래스 만들기

```
public class AutoCloseObj implements AutoCloseable {
```

```
    @Override
```

```
    public void close( ) throws Exception {
```

```
        System.out.println("리소스가 close( ) 되었습니다");
```

```
    }
```

```
}
```

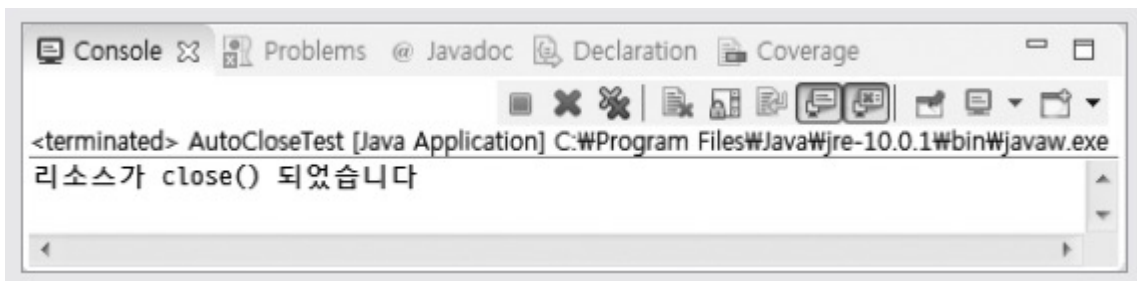
close() 메서드 구현

try-with-resources문 사용하기(1)

- 정상적으로 수행 된 경우 : close() 가 호출 됨

```
public class AutoCloseTest {  
    public static void main(String[ ] args) {  
        try(AutoCloseObj obj = new AutoCloseObj( )) {  
        } catch(Exception e) {  
            System.out.println("예외 부분입니다");  
        }  
    }  
}
```

사용할 리소스 선언

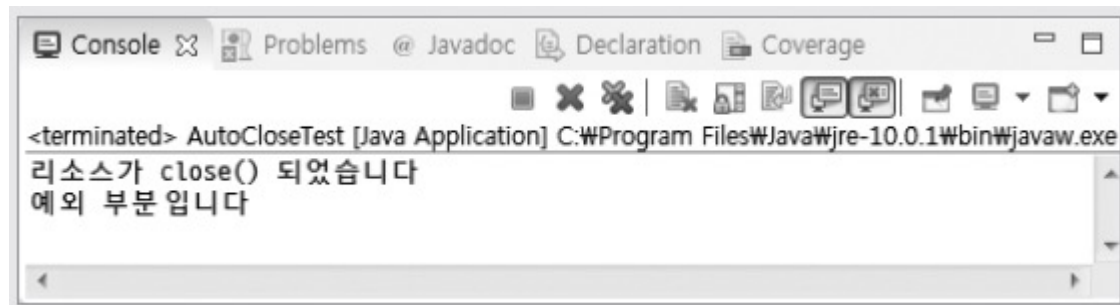


try-with-resources문 사용하기(1)

- 예외가 발생한 경우 : close() 가 호출 됨

```
public class AutoCloseTest {  
    public static void main(String[] args) {  
        try (AutoCloseObj obj = new AutoCloseObj( )) {  
            throw new Exception( );  
        } catch (Exception e) {  
            System.out.println("예외 부분입니다");  
        }  
    }  
}
```

강제 예외 발생



항상된 try-with-resources 문

- 자바 9 에서 제공되는 구문
- 외부에 선언 된 리소스도 변수만 사용 가능
- 자바 9 이전

```
AutoCloseObj obj = new AutoCloseObj( );  
try (AutoCloseObj obj2 = obj)  
    throw new Exception( );  
}catch(Exception e) {  
    System.out.println("예외 부분입니다");  
}
```

다른 참조 변수로 다시 선언해야 함

- 자바 9 이후

```
AutoCloseObj obj = new AutoCloseObj( );  
try(obj) {  
    throw new Exception( );  
} catch(Exception e) {  
    System.out.println("예외 부분입니다");  
}
```

외부에서 선언한 변수를 그대로 쓸 수 있음

예외 처리 미루기

- throws 를 사용하여 예외처리 미루기
- 메서드 선언부에 throws 를 추가
- 예외가 발생한 메서드에서 예외 처리를 하지 않고
 - 이 메서드를 호출한 곳에서 예외 처리를 한다는 의미
- main() 에서 throws를 사용하면 가상머신에서 처리 됨

예외 처리 미루기 예제

```
public class ThrowsException {
```

```
    public Class loadClass(String fileName, String className) throws
```

```
        FileNotFoundException, ClassNotFoundException {
```

```
        FileInputStream fis = new FileInputStream(fileName);
```

```
        Class c = Class.forName(className);
```

```
        return c;
```

```
    }
```

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

```
        ThrowsException test = new ThrowsException( );
```

```
        test.loadClass("a.txt", "java.lang.String");
```

```
    }
```

```
}
```

두 예외를 메서드가 호출될 때 처리하도록 미룸

throws

FileNotFoundException
발생 가능

ClassNotFoundException 발생 가능

메서드를 호출할 때 예외를 처리함

예외 처리 미루기 예제

- 모든 예외를 한 블록에서 처리 하기

```
public static void main(String[ ] args) {  
    ThrownException test = new ThrownException( );  
    try {  
        test.loadClass("a.txt", "java.lang.String");  
    } catch (FileNotFoundException | ClassNotFoundException e) {  
        // TODO Auto-generated catch block  
        e.printStackTrace( );  
    }  
}
```

생성됨

여러 예외를 한 문장으로
처리함

예외 처리 미루기 예제

- 각 상황마다 예외 처리 하기

```
public static void main(String[ ] args) {  
    ThrowsException test = new ThrowException( );  
    try {  
        test.loadClass("a.txt", "java.lang.String");  
    } catch (FileNotFoundException e) {  
        //TODO Auto-generated catch block  
        e.printStackTrace( );  
    } catch (ClassNotFoundException e) {  
        //TODO Auto-generated catch block  
        e.printStackTrace( );  
    }  
}
```

생성됨

각 예외 상황마다
다른 방식으로 처리함

다중 예외 처리 시 주의 사

- 예외가 다양한 경우 가장 최상위 클래스인 Exception 클래스에서 예외를 처리 할 수 있음

```
public static void main(String[ ] args) {  
    ThrowsException test = new ThrowsException( );  
    try {  
        test.loadClass("a.txt", "java.lang.String");  
    } catch (FileNotFoundException e) {  
        e.printStackTrace( );  
    } catch (ClassNotFoundException e) {  
        e.printStackTrace( );  
    } catch(Exception e) {  
        e.printStackTrace( );  
    }  
}
```

Exception 클래스로
그 외 예외 상황 처리

- 단 Exception 클래스는 모든 예외 클래스의 최상위 클래스 이므로 가장 마지막 블록에 위치 해야 함

사용자 정의 예외

- JDK 에서 제공되는 예외 클래스 외에 사용자가 필요에 의해 예외 클래스를 정의하여 사용
- 기존 JDK 예외 클래스 중 가장 유사한 클래스에서 상속
- 기본적으로 Exception 에서 상속해도 됨

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

생성자의 매개변수로 예외 상황 메시지를 받음

사용자 정의 예외 클래스 예제

- 전달 받은 아이디의 값이 null 이거나 8자 이하 20자 이상인 경우 예외를 발생 시킴

```
public class IDFormatTest {  
    private String userID;  
  
    public String getUserID( ) {  
        return userID;  
    }  
  
    public void setUserID(String userID) throws IDFormatException {  
        if(userID == null) {  
            throw new IDFormatException("아이디는 null일 수 없습니다");  
        }  
        else if(userID.length( ) < 8 || userID.length( ) > 20) {  
            throw new IDFormatException("아이디는 8자 이상 20자 이하로 쓰세요");  
        }  
        this.userID = userID;  
    }  
}
```

아이디에 대한 제약 조건 구현

IDFormatException 예외를 setUserID() 메서드가 호출될 때 처리하도록 미룸

강제로 예외 발생시킴

사용자 정의 예외 클래스 예제

```
public static void main(String[] args) {  
    IDFormatTest test = new IDFormatTest( );
```

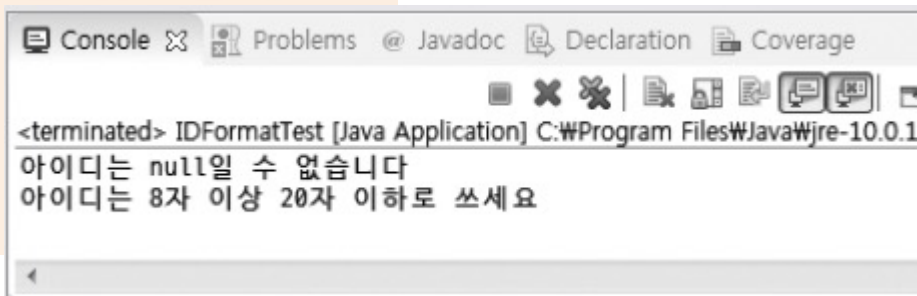
```
    String userID = null;  
    try {  
        test.setUserID(userID);  
    } catch (IDFormatException e) {  
        System.out.println(e.getMessage( ));  
    }
```

아이디 값이 null인 경우

```
    userID = "1234567";  
    try {  
        test.setUserID(userID);  
    } catch (IDFormatException e) {  
        System.out.println(e.getMessage( ));  
    }
```

아이디 값이 8자 이하인 경우

```
    }  
}
```



감사합니다.

끝

자바의 예외 처리

- **컴파일 오류**

- 문법에 맞지 않게 작성된 코드
- 컴파일할 때 발견

- **예외(Exception)**

- 오동작이나 결과에 악영향을 미칠 수 있는 실행 중 발생한 오류
 - 정수를 0으로 나누는 경우
 - 배열보다 큰 인덱스로 배열의 원소를 접근하는 경우
 - 존재하지 않는 파일을 읽으려고 하는 경우
 - 정수 입력을 기다리는 코드가 실행되고 있을 때, 문자가 입력된 경우
- 자바에서 예외 처리 가능
 - 예외 발생 -> 자바 플랫폼 인지 -> 응용프로그램에서 전달
 - 응용프로그램이 예외를 처리하지 않으면, 응용프로그램 강제 종료

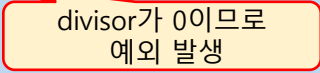
예제 8 : 0으로 나누기 예외 발생으로 프로그램이 강제 종료되는 경우

두 정수를 입력 받아 나눗셈을 하고 몫을 구하는 프로그램 코드이다. 사용자가 나누는 수에 0을 입력하면 `ArithmeticException` 예외가 발생하여 프로그램이 강제 종료된다.

```
import java.util.Scanner;

public class DivideByZero {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int dividend; // 나뉘는 수
        int divisor; // 나눗수

        System.out.print("나뉘는 수를 입력하십시오:");
        dividend = scanner.nextInt(); // 나뉘는 수 입력
        System.out.print("나눗수를 입력하십시오:");
        divisor = scanner.nextInt(); // 나눗수 입력
        System.out.println(dividend + "를 " + divisor + "로 나누면 몫은 " +
            dividend/divisor + "입니다.");
        scanner.close();
    }
}
```



나뉘는 수를 입력하십시오:100

나눗수를 입력하십시오:0

Exception in thread "main" java.lang.ArithmeticException: / by zero
at DivideByZero.main(ExceptionExample1.java:14)

예외 처리, try-catch-finally 문

- 예외 처리

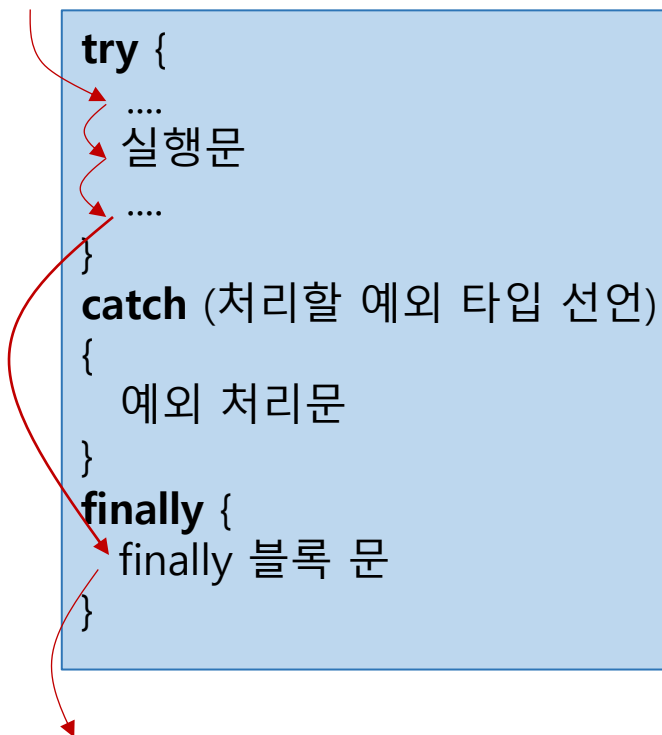
- 예외가 발생할 때 대응하는 응용프로그램 코드
- try-catch-finally 문 사용
 - finally 블록은 생략 가능

```
try {  
    예외가 발생할 가능성이 있는 실행문 (try 블록)  
}  
catch (처리할 예외 타입 선언) {  
    예외 처리문 (catch 블록)  
}  
finally {  
    예외 발생 여부와 상관없이 무조건 실행되는 문장  
    (finally 블록)  
}
```

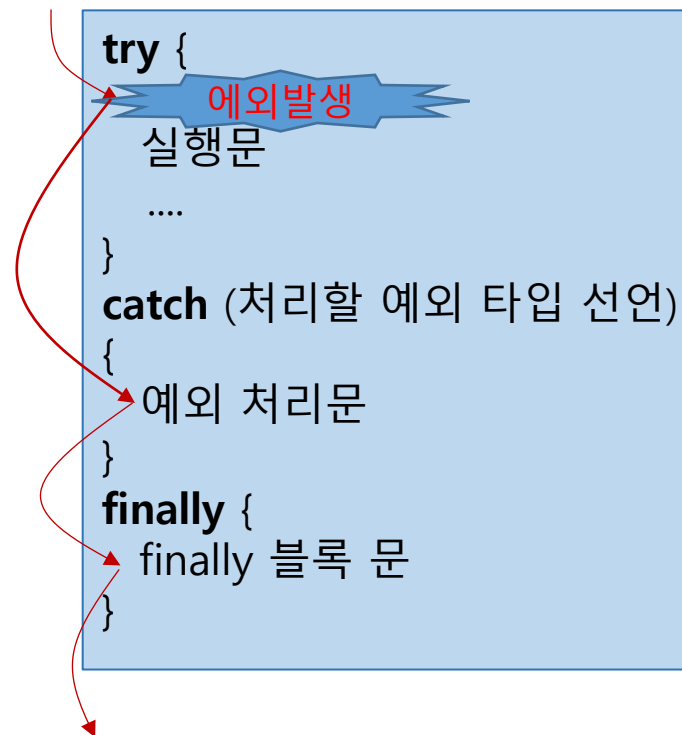
생략
가능

예외에 따른 제어의 흐름

try블록에서 예외가 발생하지 않은 정상적인 경우



try블록에서 예외가 발생한 경우



자바의 예외 클래스

• 자주 발생하는 예외

예외 타입(예외 클래스)	예외 발생 경우	패키지
ArithmeticException	정수를 0으로 나눌 때 발생	java.lang
NullPointerException	null 레퍼런스를 참조할 때 발생	java.lang
ClassCastException	변환할 수 없는 타입으로 객체를 변환할 때 발생	java.lang
OutOfMemoryError	메모리가 부족한 경우 발생	java.lang
ArrayIndexOutOfBoundsException	배열의 범위를 벗어난 접근 시 발생	java.lang
IllegalArgumentException	잘못된 인자 전달 시 발생	java.lang
IOException	입출력 동작 실패 또는 인터럽트 시 발생	java.io
NumberFormatException	문자열이 나타내는 숫자와 일치하지 않는 타입의 숫자로 변환 시 발생	java.lang
InputMismatchException	Scanner 클래스의 nextInt()를 호출하여 정수로 입력받고자 하였지만, 사용자가 'a' 등과 같이 문자를 입력한 경우	java.util

예제 9 : 0으로 나눌 때 발생하는 ArithmeticException 예외 처리

try-catch 블록을 이용하여 예제 3-14를 수정하여, 정수를 0으로 나누는 경우에 "0으로 나눌 수 없습니다!"를 출력하고 다시 입력 받는 프로그램을 작성하라.

```
import java.util.Scanner;

public class DevideByZeroHandling {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        while(true) {
            System.out.print("나뉘수를 입력하시오:");
            int dividend = scanner.nextInt(); // 나뉘수 입력
            System.out.print("나눗수를 입력하시오:");
            int divisor = scanner.nextInt(); // 나눗수 입력
            try {
                System.out.println(dividend + "를 " + divisor + "로 나누면 몫은 " + dividend/divisor + "입니다.");
                break; // 정상적인 나눗기 완료 후 while 벗어나기
            }
            catch(ArithmeticException e) { // ArithmeticException 예외 처리 코드
                System.out.println("0으로 나눌 수 없습니다! 다시 입력하세요");
            }
        }
        scanner.close();
    }
}
```

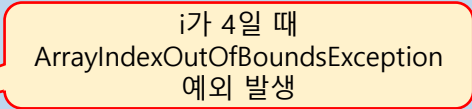
ArithmeticException
예외 발생

```
나뉘수를 입력하시오:100
나눗수를 입력하시오:0
0으로 나눌 수 없습니다! 다시 입력하세요
나뉘수를 입력하시오:100
나눗수를 입력하시오:5
100를 5로 나누면 몫은 20입니다.
```

예제 10 : 범위를 벗어난 배열의 접근

배열의 인덱스가 범위를 벗어날 때 발생하는 `ArrayIndexOutOfBoundsException`을 처리하는 프로그램을 작성하시오.

```
public class ArrayException {
    public static void main (String[] args) {
        int[] intArray = new int[5];
        intArray[0] = 0;
        try {
            for (int i=0; i<5; i++) {
                intArray[i+1] = i+1 + intArray[i];
                System.out.println("intArray["+i+"]"+"="+intArray[i]);
            }
        }
        catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("배열의 인덱스가 범위를 벗어났습니다.");
        }
    }
}
```



```
intArray[0]=0
intArray[1]=1
intArray[2]=3
intArray[3]=6
배열의 인덱스가 범위를 벗어났습니다.
```

예제 12 : 입력오류시 발생하는 예외(InputMismatchException)

3개의 정수를 입력받아 합을 구하는 프로그램을 작성하라. 사용자가 정수가 아닌 문자를 입력할 때 발생하는 InputMismatchException 예외를 처리하여 다시 입력받도록 하라.

```
import java.util.Scanner;
import java.util.InputMismatchException;

public class InputException {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("정수 3개를 입력하세요");
        int sum=0, n=0;
        for(int i=0; i<3; i++) {
            System.out.print(i+">>");
            try {
                n = scanner.nextInt(); // 정수 입력
            }
            catch(InputMismatchException e) {
                System.out.println("정수가 아닙니다. 다시 입력하세요!");
                scanner.next(); // 입력 스트림에 있는 정수가 아닌 토큰을 버린다.
                i--; // 인덱스가 증가하지 않도록 미리 감소
                continue; // 다음 루프
            }
            sum += n; // 합하기
        }
        System.out.println("합은 " + sum);
        scanner.close();
    }
}
```

사용자가 문자를 입력하면
InputMismatchException 예외 발생

정수 3개를 입력하세요

0>>5

1>>R

정수가 아닙니다. 다시 입력하세요!

1>>4

2>>6

합은 15

예제 13: 정수가 아닌 문자열을 정수로 변환할 때 예외 발생 (NumberFormatException)

문자열을 정수로 변환할 때 발생하는 NumberFormatException을 처리하는 프로그램을 작성하라.

```
public class NumException {  
    public static void main (String[] args) {  
        String[] stringNumber = {"23", "12", "3.141592", "998"};  
  
        int i=0;  
        try {  
            for (i=0; i<stringNumber.length; i++) {  
                int j = Integer.parseInt(stringNumber[i]);  
                System.out.println("숫자로 변환된 값은 " + j);  
            }  
        }  
        catch (NumberFormatException e) {  
            System.out.println(stringNumber[i] + "는 정수로 변환할 수 없습니다.");  
        }  
    }  
}
```

"3.141592"를 정수로 변환할 때
NumberFormatException
예외 발생

숫자로 변환된 값은 23
숫자로 변환된 값은 12
3.141592는 정수로 변환할 수 없습니다.