

예외처리

18-1. 자바 예외처리의 기본

자바에서 말하는 예외

예외(Exception)

‘예외적인 상황’을 줄여서 ‘예외’라 한다.

단순한 문법 오류가 아닌 실행 중간에 발생하는 ‘정상적이지 않은 상황’을 뜻한다.

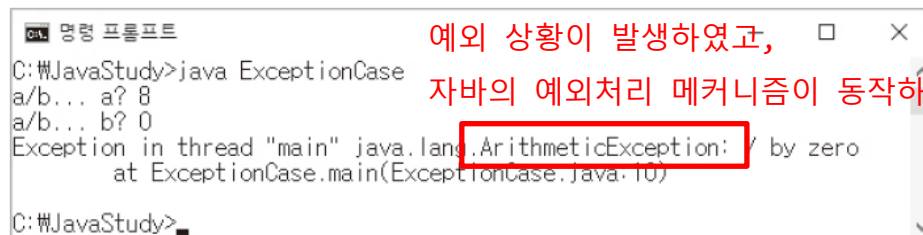
예외처리

예외 상황에 대한 처리를 의미한다.

자바는 예외처리 메커니즘을 제공한다.

예외의 상황의 예

```
public static void main(String[] args) {  
    Scanner kb = new Scanner(System.in);  
  
    System.out.print("a/b...a? ");  
    int n1 = kb.nextInt();    // int형 정수 입력  
  
    System.out.print("a/b...b? ");  
    int n2 = kb.nextInt();    // int형 정수 입력  
  
    System.out.printf("%d / %d = %d \n", n1, n2, n1 / n2);  
    System.out.println("Good bye~~!");  
}
```



```
C:\JavaStudy>java ExceptionCase  
a/b... a? 8  
a/b... b? 0  
Exception in thread "main" java.lang.ArithmeticException: / by zero  
    at ExceptionCase.main(ExceptionCase.java:10)  
C:\JavaStudy>
```

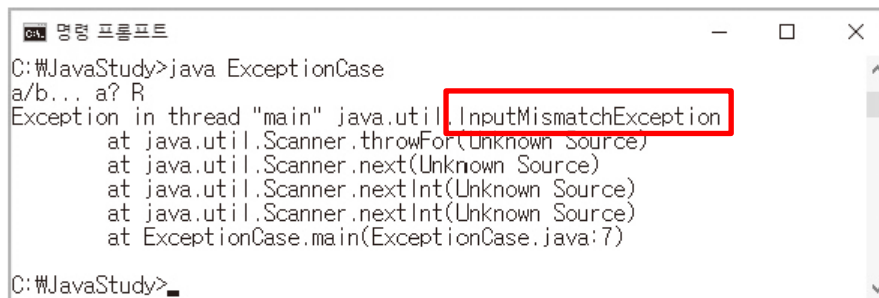
예외 상황이 발생하였고,
자바의 예외처리 메커니즘이 동작하였다.

예외 상황의 또 다른 예

```
public static void main(String[] args) {  
    Scanner kb = new Scanner(System.in);  
  
    System.out.print("a/b...a? ");  
    int n1 = kb.nextInt();    // int형 정수 입력  
  
    System.out.print("a/b...b? ");  
    int n2 = kb.nextInt();    // int형 정수 입력  
  
    System.out.printf("%d / %d = %d \n", n1, n2, n1 / n2);  
    System.out.println("Good bye~~!");  
}
```

예외에 대한 처리 방법은 프로그래머가 결정할 수 있다.

자바의 기본 예외처리 메커니즘은
문제가 발생한 지점에 대한 정보 출력과
프로그램 종료이다!



```
명령 프롬프트  
C:\JavaStudy>java ExceptionCase  
a/b... a? R  
Exception in thread "main" java.util.InputMismatchException  
    at java.util.Scanner.throwFor(Unknown Source)  
    at java.util.Scanner.next(Unknown Source)  
    at java.util.Scanner.nextInt(Unknown Source)  
    at java.util.Scanner.nextInt(Unknown Source)  
    at ExceptionCase.main(ExceptionCase.java:7)  
C:\JavaStudy>
```

예외 상황을 알리기 위한 클래스

`java.lang.ArithmeticException`

→ 수학 연산에서의 오류 상황을 의미하는 예외 클래스

`java.util.InputMismatchException`

→ 클래스 Scanner를 통한 값의 입력에서의 오류 상황을 의미하는 예외 클래스

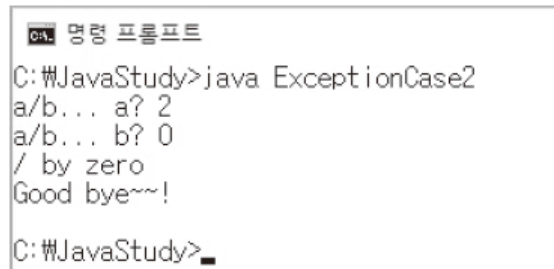
예외의 처리를 위한 try ~ catch

```
try {  
    ...관찰 영역...  
}  
  
catch(ArithmeticException e) {  
    ...처리 영역...  
}
```

예외의 처리를 위한 코드를 별도로 구분하기 위해 디자인된 예외처리 메커니즘이 try ~ catch 이다.

try ~ catch의 예

```
public static void main(String[] args) {  
    Scanner kb = new Scanner(System.in);  
  
    try {  
        System.out.print("a/b...a? ");  
        int n1 = kb.nextInt();  
        System.out.print("a/b...b? ");  
        int n2 = kb.nextInt();  
        System.out.printf("%d / %d = %d \n", n1, n2, n1 / n2);    // 예외 발생 지점  
    }  
    catch(ArithmeticException e) {  
        System.out.println(e.getMessage());  
    }  
  
    System.out.println("Good bye~~!");  
}
```



```
C:\JavaStudy>java ExceptionCase2  
a/b... a? 2  
a/b... b? 0  
/ by zero  
Good bye~~!  
C:\JavaStudy>
```


예외 발생 이후의 실행 흐름

```
try {  
    1. ...  
    2. 예외 발생 지점  
    3. ...  
}  
catch(Exception e) {  
    ...  
}
```

4. 예외 처리 이후 실행 지점

try로 감싸야 할 영역의 결정

```
public static void main(String[] args) {  
    Scanner kb = new Scanner(System.in);
```

입력 오류에 대한 예외의 관점에서 보았을 때 이는 하나의 작업

```
try {  
    System.out.print("a/b...a? ");  
    int n1 = kb.nextInt();  
    System.out.print("a/b...b? ");  
    int n2 = kb.nextInt();  
    System.out.printf("%d / %d = %d \n", n1, n2, n1/n2);  
}  
catch (InputMismatchException e) {  
    e.getMessage();  
}  
  
System.out.println("Good bye~~!");  
}
```

둘 이상의 예외 처리를 위한 구성1

```
public static void main(String[] args) {  
    Scanner kb = new Scanner(System.in);  
  
    try {  
        System.out.print("a/b...a? ");  
        int n1 = kb.nextInt();  
        System.out.print("a/b...b? ");  
        int n2 = kb.nextInt();  
        System.out.printf("%d / %d = %d \n", n1, n2, n1 / n2);  
    }  
    catch(ArithmeticException e) {  
        e.getMessage();  
    }  
    catch(InputMismatchException e) {  
        e.getMessage();  
    }  
  
    System.out.println("Good bye~~!");  
}
```

둘 이상의 예외 처리를 위한 구성2

```
public static void main(String[] args) {  
    Scanner kb = new Scanner(System.in);  
  
    try {  
        System.out.print("a/b...a? ");  
        int n1 = kb.nextInt();  
        System.out.print("a/b...b? ");  
        int n2 = kb.nextInt();  
        System.out.printf("%d / %d = %d \n", n1, n2, n1 / n2);  
    }  
    catch(ArithmeticException | InputMismatchException e) {  
        e.getMessage();  
    }  
  
    System.out.println("Good bye~~!");  
}
```

Throwable 클래스

java.lang.Throwable 클래스

모든 예외 클래스의 최상위 클래스: 물론 Throwable도 Object를 상속한다.

Throwable 클래스의 메소드 둘

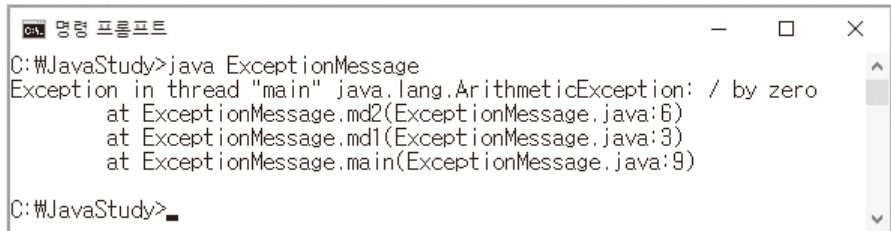
public String getMessage() : 예외의 원인을 담고 있는 문자열을 반환

public void printStackTrace() : 예외가 발생한 위치와 호출된 메소드의 정보를 출력

예외의 전달

```
class ExceptionMessage {  
    public static void md1(int n) {  
        md2(n, 0);    // 아래의 메소드 호출  
    }  
  
    public static void md2(int n1, int n2) {  
        int r = n1 / n2;    // 예외 발생 지점  
    }  
  
    public static void main(String[] args) {  
        md1(3);  
        System.out.println("Good bye~~!");  
    }  
}
```

예외 발생 지점에서 예외를 처리하지 않으면 해당 메소드를 호출한 영역으로 예외가 전달된다.



```
명령 프롬프트  
C:\JavaStudy>java ExceptionMessage  
Exception in thread "main" java.lang.ArithmeticException: / by zero  
    at ExceptionMessage.md2(ExceptionMessage.java:6)  
    at ExceptionMessage.md1(ExceptionMessage.java:3)  
    at ExceptionMessage.main(ExceptionMessage.java:9)  
C:\JavaStudy>
```

printStackTrace 메소드 관련 예제

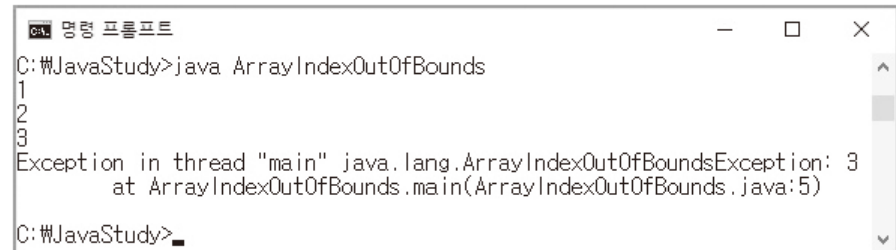
```
class ExceptionMessage2 {  
    public static void md1(int n) {  
        md2(n, 0);    // 이 지점으로 md2로부터 예외가 넘어온다.  
    }  
    public static void md2(int n1, int n2) {  
        int r = n1 / n2;    // 예외 발생 지점  
    }  
    public static void main(String[] args) {  
        try {  
            md1(3);    // 이 지점에서 md1으로부터 예외가 넘어온다.  
        }  
        catch(Throwable e) {  
            e.printStackTrace();  
        }  
  
        System.out.println("Good bye~~!");  
    }  
}
```



```
CA\ 명령 프롬프트  
C:\JavaStudy>java ExceptionMessage2  
java.lang.ArithmeticException: / by zero  
    at ExceptionMessage2.md2(ExceptionMessage2.java:6)  
    at ExceptionMessage2.md1(ExceptionMessage2.java:3)  
    at ExceptionMessage2.main(ExceptionMessage2.java:10)  
Good bye~~!  
C:\JavaStudy>
```

ArrayIndexOutOfBoundsException

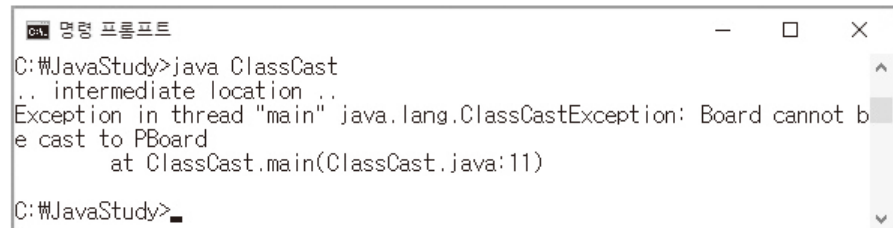
```
class ArrayIndexOutOfBoundsException {  
    public static void main(String[] args) {  
        int[] arr = {1, 2, 3};  
  
        for(int i = 0; i < 4; i++)  
            System.out.println(arr[i]); // 인덱스 값 3은 예외를 발생시킴  
    }  
}
```



The screenshot shows a Windows command prompt window titled "명령 프롬프트". The command executed is `C:\#JavaStudy>java ArrayIndexOutOfBoundsException`. The output shows the numbers 1, 2, and 3 printed on separate lines, followed by an exception message: `Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 3` and `at ArrayIndexOutOfBoundsException.main(ArrayIndexOutOfBoundsException.java:5)`. The prompt then returns to `C:\#JavaStudy>`.

ClassCastException

```
class Board { }  
class PBoard extends Board { }  
  
class ClassCast {  
    public static void main(String[] args) {  
        Board pbd1 = new PBoard();  
        PBoard pbd2 = (PBoard)pbd1;    // OK!  
  
        System.out.println(".. intermediate location ..");  
        Board ebd1 = new Board();  
        PBoard ebd2 = (PBoard)ebd1;    // Exception!  
    }  
}
```



```
C:\#JavaStudy>java ClassCast  
.. intermediate location ..  
Exception in thread "main" java.lang.ClassCastException: Board cannot be  
cast to PBoard  
    at ClassCast.main(ClassCast.java:11)  
C:\#JavaStudy>
```

NullPointerException

```
class NullPointer {  
    public static void main(String[] args) {  
        String str = null;  
        System.out.println(str);    // null 출력  
        int len = str.length();    // Exception!  
    }  
}
```



A screenshot of a Windows command prompt window titled "명령 프롬프트". The window shows the execution of a Java program. The command entered is "C:\JavaStudy>java NullPointer". The output is "null" followed by an exception message: "Exception in thread "main" java.lang.NullPointerException at NullPointer.main(NullPointer.java:5)". The prompt "C:\JavaStudy>" is visible at the bottom.

```
명령 프롬프트  
C:\JavaStudy>java NullPointer  
null  
Exception in thread "main" java.lang.NullPointerException  
    at NullPointer.main(NullPointer.java:5)  
C:\JavaStudy>
```

18-2.

예외처리에 대한 나머지 설명들

예외 클래스의 구분

- `Error` 클래스를 상속하는 예외 클래스
- `Exception` 클래스를 상속하는 예외 클래스
- `RuntimeException` 클래스를 상속하는 예외 클래스
 - `RuntimeException` 클래스는 `Exception` 클래스를 상속한다.

Error 클래스를 상속하는 예외 클래스들의 특성

- **Error** 클래스를 상속하는 예외 클래스
- Exception 클래스를 상속하는 예외 클래스
- RuntimeException 클래스를 상속하는 예외 클래스

Error 클래스를 상속하는 예외 클래스의 예외 상황은 시스템 오류 수준의 예외 상황으로 프로그램 내에서 처리 할 수 있는 수준의 예외가 아니다.

ex) `VirtualMachineError`

가상머신에 심각한 오류 발생

`IOError`

입출력 관련해서 코드 수준 복구가 불

가능한 오류 발생

RuntimeException 클래스를 상속하는 예외 클래스들의 특성

- Error 클래스를 상속하는 예외 클래스
- Exception 클래스를 상속하는 예외 클래스
- RuntimeException 클래스를 상속하는 예외 클래스

코드 오류로 발생하는 경우가 대부분이다. 따라서 이 유형의 예외 발생시 코드의 수정을 고려해야 한다.

ex) ArithmeticException

ClassCastException

IndexOutOfBoundsException

NegativeArraySizeException

NullPointerException

ArrayStoreException

배열 생성시 길이를 음수로 지정하는 예외의 발생

배열에 적절치 않은 인스턴스를 저장하는 예외의 발생

Exception 클래스를 상속하는 예외 클래스들의 특성

- Error 클래스를 상속하는 예외 클래스
- **Exception** 클래스를 상속하는 예외 클래스
- RuntimeException 클래스를 상속하는 예외 클래스

코드의 문법적 오류가 아닌, 프로그램 실행 과정에서 발생하는 예외적 상황을 표현하기 위한 클래스들이다.

따라서 예외의 처리를 어떻게 할 것인지 반드시 명시해 주어야 한다.

ex) `java.io.IOException`

입출력 관련 예외 상황을 표현하는 예외 클래스

Exception을 상송하는 예외의 예

```
public static void main(String[] args) {  
    Path file = Paths.get("C:\\javastudy\\Simple.txt");  
    BufferedWriter writer = null;  
  
    try {  
        writer = Files.newBufferedWriter(file);    // IOException 발생 가능  
        writer.write('A');    // IOException 발생 가능  
        writer.write('Z');    // IOException 발생 가능  
  
        if(writer != null)  
            writer.close();    // IOException 발생 가능  
    }  
    catch(IOException e) {  
        e.printStackTrace();  
    }  
}
```

Exception을 상속하는 예외는 반드시 처리를 해야 한다.
그렇지 않으면 컴파일 오류로 이어진다.

처리하거나 아니면 넘기거나

```
public static void main(String[] args) {  
    try {  
        md1();  
    }  
    catch(IOException e) {  
        e.printStackTrace();  
    }  
}
```

메소드의 throws절 선언을 통해 예외의 처리를 넘길 수 있다.

```
public static void md1() throws IOException {    // IOException 예외 넘긴다고 명시!  
    md2();  
}  
public static void md2() throws IOException {    // IOException 예외 넘긴다고 명시!  
    Path file = Paths.get("C:\\javastudy\\Simple.txt");  
    BufferedWriter writer = null;  
    writer = Files.newBufferedWriter(file);    // IOException 발생 가능  
    writer.write('A');    // IOException 발생 가능  
    writer.write('Z');    // IOException 발생 가능  
  
    if(writer != null)  
        writer.close();    // IOException 발생 가능  
}
```

둘 이상의 예외 넘김에 대한 선언

```
public void simpleWrite() throws IOException, IndexOutOfBoundsException {  
  
    ....  
  
}
```

프로그래머가 정의하는 예외 클래스

Exception 클래스를 상속하는 것이 예외 클래스의 유일한 조건

```
class ReadAgeException extends Exception {    // Exception을 상속하는 것이 핵심

    public ReadAgeException() {

        super("유효하지 않은 나이가 입력되었습니다.");

    }    Throwable 클래스의 getMessage 메소드가 반환할 문자열 지정

}
```

프로그래머 정의 예외 클래스의 예

```
class MyExceptionClass {
    public static void main(String[] args) {
        System.out.print("나이 입력: ");

        try {
            int age = readAge();
            System.out.printf("입력된 나이: %d \n", age);
        }
        catch(ReadAgeException e) {
            System.out.println(e.getMessage());
        }
    }
    public static int readAge() throws ReadAgeException {
        Scanner kb = new Scanner(System.in);
        int age = kb.nextInt();

        if(age < 0)
            throw new ReadAgeException(); // 예외의 발생

        return age;
    }
}
```

```
class ReadAgeException extends Exception {
    . . .
}
```

명령 프롬프트

```
C:\JavaStudy>java MyExceptionClass
나이 입력: 12
입력된 나이: 12
```

```
C:\JavaStudy>java MyExceptionClass
나이 입력: -7
유효하지 않은 나이가 입력되었습니다.
```

```
C:\JavaStudy>_
```

잘못된 catch 구분의 구성

```
try {  
    ....  
}  
  
catch(FirstException e) {...}  
  
catch(SecondException e) {...}  
  
catch(ThirdException e) {...}
```

```
class FirstException extends Exception {...}  
  
class SecondException extends FirstException {...}  
  
class ThirdException extends SecondException {...}
```

컴파일러는 친절하게도

이 부분에 대해 컴파일 오류를 전달해 준다!

finally

```
try {...  
}  
finally {...    // 코드의 실행이 try 안으로 진입하면, 무조건 실행된다.  
}
```

```
try {...  
}  
catch(Exception name) {...  
}  
finally {...    // 코드의 실행이 try 안으로 진입하면, 무조건 실행된다.  
}
```

finally 구분의 사용의 예

```
public static void main(String[] args) {
    Path file = Paths.get("C:\\javastudy\\Simple.txt");
    BufferedWriter writer = null;
    try {
        writer = Files.newBufferedWriter(file); // IOException 발생 가능
        writer.write('A'); // IOException 발생 가능
    }
    catch(IOException e) {
        e.printStackTrace();
    }
    finally {
        try {
            if(writer != null)
                writer.close(); // IOException 발생 가능
        }
        catch(IOException e) {
            e.printStackTrace();
        }
    }
}
```

실행의 흐름이 try 구문 안에 들어왔을 때 반드시 실행해야 하는 문장을 finally 구문에 둘 수 있다.
그러나! 보다 멋진 대안이 등장했다!

try-with-resources

```
try(BufferedWriter writer = Files.newBufferedWriter(file)) {  
    writer.write('A');  
    writer.write('Z');  
}                               try 구문을 빠져 나갈 때 다음 문장을 안정적으로 자동 실행  
                                writer.close();  
  
catch(IOException e) {  
    e.printStackTrace();  
}
```

try-with-resources 기반의 오픈 및 종료 대상이 되기 위한 조건은 다음과 같다.

java.lang.AutoCloseable 인터페이스의 구현