08_1 Exception Class

Object-Oriented Programming

Exception Class에 대해 강의하겠습니다.

Errors and Exceptions

- Errors: Serious errors that **cannot be handled** by program code
  - Compile-time Error
  - Run-time Error
  - Logic Error
- Exceptions
  - Minor errors that **can be handled** by program code
- Exception Handling
  - Preventing abnormal termination of the program **by coding**
  - Maintaining normal execution

페이지 2

먼저 Error와 Exception을 구분할 필요가 있습니다.
Error는 program code로 처리가 되지 않는 심각한 error를 말합니다.
Compile-time error는
프로그래밍 언어의 문법에 어긋난 코드에서 발생하며
compile이 실행되지 않게하는 error입니다.
Run-time error는 compile time에는 감지되지 않으며
프로그램이 실행중에 예기치 않게 종료되는 경우입니다.
예를 들어 memory 부족, stack overflow, 하드웨어 결함 등이 있는데
이것은 프로그램의 오류 보다는 system의 자원 부족등으로 나타나기 때문에
예측할 수 없다는 공통점이 있습니다.
Logic error는 compile도 되고 run도 되지만
원하는 답이 나오지 않는 경우로
프로그램에 사용된 알고리즘을 잘못 설계했을 경우에 일어납니다.
한편, 복구하기 어려운 Error에 비해
Exception은 그 발생을 예상하는 program의 code 지점에
exception을 처리할 code를 추가함으로써 처리될 수 있는
비교적 가벼운 error를 말합니다.
그렇다면 Exception을 handling 한다는 것은 무엇을 말할까요?
program의 비정상적인 종료를 coding을 통해 방지하여
정상적인 프로그램 실행을 유지하도록 하는 것을 말합니다.
여기서 비정상적인 종료를 방지한다는 것은
프로그램을 절대 종료시키지 않는다는 말이 아니라
적어도 프로그래머가 이 프로그램이 왜 여기서 종료되었는지를
알 수 있도록 코딩할 수 있다는 뜻이기도 합니다.

Types of Exception

- General Exception (= compile-time checked exception)
  - Compiler checks for existence of appropriate exception handling code
  - If there is no exception handling code, a compile error occurs

- Runtime Exception (= compile-time unchecked exception)
  - Does not check for exception handling code at compile time
  - But, to handle the exception, exception-handling code must be written by programmer
  - If runtime exception occurs
    - If the program has exception-handling code, then the code is executed
    - If no exception-handling code, the program stops immediately

페이지 3

Exception의 Type에 대해 알아보겠습니다.
먼저 General Exception은
compile-time checked exception 이라고도 불리는데
컴파일러가 적절한 exception handling code의 존재를
컴파일하면서 체크하여
만약 exception handling code가 존재하지 않으면
compile error를 발생시키는 경우를 말합니다.
Exception 에 대해 상당히 엄격하게
그 처리를 요구하는 경우라 할 수 있습니다.
다른 type으로 Runtime Exception이 있는데
이것은 compile-time unchecked exception 이라고도 불립니다.
이 exception type은
compiler가 exception handling code의 존재여부를
compile time에 체크하지는 않습니다.
그러나 programmer는
Runtime Exception type의 exception을 handling하기 위해서
exception handling code를 직접 써 넣을 수도 있습니다.
runtime exception이 발생한 경우
만일 이미 coding되어 있는 exception handling code가 존재하는 경우에는
그 code에 따라 exception이 handling되며
exception-handling code가 존재하지 않는 경우에는
program이 즉시 stop 됩니다.

Exception Classes

- java.lang.Exception
  - java.lang.ClassNotFoundException
  - java.io.IOException
  - ….
  - java.lang.RuntimeException
    - java.lang.ArithmeticException
    - java.lang.ClassCastException
    - java.lang.NullPointerException
    - java.lang.IndexOutOfBoundsException
    - …

} General Exception
(Compile-time checked Exception)

} Runtime Exception
(Compile-time unchecked Exception)

페이지 4

java.lang.Exception은 모든 exception class 중에
가장 ancestor class입니다.
모든 predefined exception들과
또한 user defined exception들까지도
대문자로 시작하는 Exception class의 descendant로 정의됩니다.
Exception의 children class중에
java.lang.RuntimeException이 아닌
예를 들면 ClassNotFoundException,
java.io.IOException등은
앞 슬라이드에서 정의한 General Exception에 속하기 때문에
Compile time에 exception handling code의 존재가 check되고
coding할 때 exception handling code도
함께 coding해야 합니다.
한편 Exception의 children class 중에
java.lang.RuntimeException은
앞 슬라이드에서 정의한 Runtime Exception에 속합니다.
java.lang.RuntimeException은
많은 children class들을 가지는데
이들이 모두 Runtime Exception 들이고
따라서 compile time에
exception handling code의 존재가 check되지 않으며
runtime에 exception이 발생할 경우에
exception handling code가 있는 때에만
exception을 handling 합니다.

General Exception: ClassNotFoundException (1/3)

```java
class TempClass0 { }

public class ClassClass0 {
    public static void main(String[] args) {
        TempClass0 t0 = new TempClass0();
        System.out.println("t0.getClass() returns: " + t0.getClass());
    }
}
```

```
OUTPUT: t0.getClass() returns: class TempClass0
```


페이지 5

이제 General Exception 중의 하나인
ClassNotFoundException이 발생하는 경우에 대해
알아보려고 합니다.
ClassNotFoundException에 대해서 이해하기 위해서는
대문자로 시작하는 Class class에 대해 이해해야 합니다.
이 프로그램에는 class TempClass0가 정의되어 있습니다.
main method에서 TempClass0의 object인 t0를 생성하였습니다.
t0.getClass()를 call하여 java.lang.Class type의 object를 return합니다.
이 대문자로 시작하는 Class object는 현재 실행중인 java application 내의
특정 class, 여기서는 t0의 class 이니까
TempClass0에 대한 정보를 가지고 있습니다.
OUTPUT을 보니 t0.getClass()에서 return 된 object는
class TempClass0로 print되었습니다.

General Exception: ClassNotFoundException (2/3)

```
class TempClass0 { }

public class ClassClass0 {
    public static void main(String[] args) {
        TempClass0 t0 = new TempClass0();
        System.out.println("t0.getClass() returns: " + t0.getClass());
        Class classInfo;
        classInfo = Class.forName("TempClass0");
        System.out.println("classInfo: " + classInfo);
        System.out.println("classInfo.getName() returns: " + classInfo.getName());
    }
}
```

```
java: unreported exception java.lang.ClassNotFoundException; must be caught or declared
to be thrown
```


페이지 6

앞 슬라이드의 예와 같이 TempClass0 object인 t0를 생성하였고
t0.getClass()를 call하여 보았습니다.
그리고 나서 Class.forName("TempClass0") 를 call 하는데
forName은 Class의 static method로서
"TempClass0" 를 이름으로 가지는 class를 찾아서 그 info인 Class object를 return합니다.
즉 t0.getClass()가 return하는 Class object와 같은 것을 return 해야 하는 것입니다.
그러나 아래 box에서 보이듯이 이 프로그램은 compile error가 나게 되는데
Class.forName("TempClass0") 을 call하는 부분에서
그 이름을 가진 class가 현재 application 내에 존재하지 않을 경우를 대비하여
Exception을 handling하는 code를 쓰지 않았기 때문에
compile error가 나게 됩니다.
이렇게 compile error가 나는 것은 ClassNotFoundException이
General Exception이기 때문입니다.

General Exception: ClassNotFoundException (3/3)

```
class TempClass1 { }

public class ClassClass {
    public static void main(String[] args) {
        TempClass1 t1 = new TempClass1();
        Class classInfo;
        System.out.println("t1.getClass() returns: " + t1.getClass());
        try {
            classInfo = Class.forName("TempClass1");
            System.out.println("classInfo: " + classInfo);
            System.out.println("classInfo.getName() returns: " + classInfo.getName());
        } catch (ClassNotFoundException e) {
            System.out.println("Class not found \"TempClass1\"");
        }
    }
}
```

```
t1.getClass() returns: class TempClass1
classInfo: class TempClass1
classInfo.getName() returns: TempClass1
```


페이지 7

이제 ClassNotFoundException handling을 제대로 해 둠으로써
compile error를 방지하고자 합니다.
TempClass1 object인 t1을 생성했고
t1.getClass() 도 정상적으로 실행되었습니다.
그 다음에 Class.forName("TempClass1")을 실행하는 부분을
try block으로 감쌌습니다.
이 handling code 때문에 compile error는 일어나지 않았으며
Class.forName("TempClass1") 은 classInfo를 정상적으로 return하여
class TempClass1 이라는 output을 print하게 됩니다.
또 classInfo.getName() call은
다시 class의 이름인 String "TempClass1" 을 return하게 됩니다.
만일 TempClass1이 현재 application에 존재하지 않는 경우 발생하는
ClassNotFoundException 을 catch block에서 catch하여
exception handling을 하는 code를 추가했습니다.
try-catch mechanism 에 대해서는
나중에 더 자세히 공부할 것입니다.

General Exception: java.io.IOException

```java
public class IOExceptionExample {
    public static void main(String[] args) {
        BufferedReader reader = null;
        try {
            reader = new BufferedReader(new FileReader("example.txt"));
            String line;
            while ((line = reader.readLine()) != null) {
                System.out.println(line);
            }
        }
        catch (IOException e) {
            System.out.println("파일을 read 중에 오류가 발생: " + e.getMessage());
            e.printStackTrace();
        }
```

```
파일을 read 중에 오류가 발생: example.txt (No such file or directory)
java.io.FileNotFoundException: example.txt (No such file or directory)
        at java.base/java.io.FileInputStream.open0(Native Method)
        at java.base/java.io.FileInputStream.open(FileInputStream.java:213)
        at java.base/java.io.FileInputStream.<init>(FileInputStream.java:152)
        at java.base/java.io.FileInputStream.<init>(FileInputStream.java:106)
        at java.base/java.io.FileReader.<init>(FileReader.java:60)
        at IOExceptionExample.main(IOExceptionExample.java:9)
```


페이지 8

이 example에서는 지면 관계로 몇 개의 import 문을 생략했으며
완전한 code는 배포된 code를 참고하시기 바랍니다.
또다른 General Exception인 java.io.IOException에 대해 살펴 보겠습니다.
FileReader object를 "example.txt" 파일을 open하면서 생성하는 부분에서
"example.txt" 파일이 .class 파일과 같은 folder안에 존재하지 않는 경우
IOException이 발생하게 됩니다.
IOException은 General Exception이기 때문에
IOException의 handling code를 쓰지 않은 경우에는
compile error가 나게 됩니다.
따라서 이 경우에도 FileReader의 생성을 try block안에서 실행하고
IOException이 일어날 경우 그것을 handling하는 code를
catch block 안에 두어야 합니다.
이 example program의 경우 "example.txt" 파일이 존재하지 않아서
FileReader를 생성할 때 실제로 IOException이 일어나게 됩니다.
따라서 catch block 안에서 IOException을 handling하게 되는데
먼저 "파일을 read 중에 오류가 발생" 이라는 메시지와
IOException 안에 기존에 포함되어 있는 e.getMessage() 인
"example.txt (No such file or directory)" 를 print합니다.
e.printStackTrace() 를 call하면
program에서 IOException이 발생한 위치까지
어떤 method들이 차례로 실행되고 call되었는지를 나타내는
stack trace가 print됩니다.
stack trace에서 보면 가장 아랫줄에
IOExceptionExample.java의 9번째 line에 있는
IOExceptionExample.main method에서
java.io.FileNotFoundException이 발생합니다.
FileNotFoundException은 IOException의 descendant입니다.
그 위로 역순으로 call된 method들이 차례로 나열되게 됩니다.

Runtime Exception: NullPointerException

- Occurs when the dot (.) operator is used on a reference variable without the object it references
- ex)

```java
public class NullPointerExceptionDemo {
    public static void main(String[] args) {
        String str1;
        String str2 = null;
        // System.out.println(str1.toString()); // compile error!!
        System.out.println(str2.toString()); // no compile error
                                             // but runtime exception
    }
}
```

```
Exception in thread "main" java.lang.NullPointerException: Cannot
invoke "String.toString()" because "str2" is null
    at NullPointerExceptionDemo.main(NullPointerExceptionDemo.java:6)
```


페이지 9

이제 Runtime Exception들을 살펴 보도록 하겠습니다.
NullPointerException은 이미 우리가 reference type에 대해 처음 배울때
한번 본적이 있는 것입니다.
String str1은 declare만 하였고 어떤 object도 reference하지 않고 있으며
String str2는 null로 initialize되어 있습니다.
이제 str1.toString() 을 coding하는 단계에서는
compile error가 나게 됩니다.
str1이 가리키는 것이 어떤 의미있는 object가 아니기 때문입니다.
한편 str2.toString()을 call하는 것은
compile error가 나지는 않습니다.
str2가 무엇인가로 (여기서는 null로) 초기화 되어 있기 때문입니다.
하지만 null object는 dereference되지 못하며
의미있는 toString() method를 가지고 있지 않기 때문에
실행 도중 이 명령을 만났을 때
runtime exception인 NullPointerException이 발생하게 됩니다.
하지만 이 프로그램에서는 NullPointerException을 handling하는
아무런 handling code가 없기 때문에
즉시 프로그램이 중단되면서
아래 box의 "Exception in thread "main" java.lang.NullPointerException...."
과 같은 message가 프린트 됩니다.
이와 같은 runtime exception의 handling에 대해서는
다음 노트에서 자세히 살펴보도록 하겠습니다.

Runtime Exception: ArrayIndexOutOfBoundsException

- Occurs when the index range is exceeded in the array

```
public class ArrayIndexOutOfBoundsExceptionDemo {
    public static void main(String[] args) {
        String[] strArray = new String[3];
        strArray[0] = "Korea";
        strArray[3] = "Seoul";
    }
}
```

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException:
Index 3 out of bounds for length 3 at
ArrayIndexOutOfBoundsExceptionDemo.main(ArrayIndexOutOfBoundsExceptionDemo.java:5)
```


페이지 10

이번에는 length 3인 String array인 strArray를 고려합니다.
첫번째 element인 strArray[0]에는 "Korea" 를 assign 했습니다.
그 다음에 strArray[3]에 "Seoul" 을 assign 하려 했는데
이 실행과정에서 java.lang.ArrayIndexOutOfBoundsException이
발생하게 됩니다.
왜냐하면 strArray는 length가 3이고
strArray[2]가 마지막 element이기 때문입니다.
이렇게 array의 index 범위를 벗어났을 때 발생하는 runtime exception이
java.lang.ArrayIndexOutOfBoundsException 입니다.
아래의 box 안에서 exception 발생으로 프로그램이 끝난 이후
print되는 message를 볼 수 있습니다.

Runtime Exception: NumberFormatException

- Occurs when trying to convert a string into a numeric type data, such as int or double, and the string cannot be converted to a number (e.g. "23asdf", "?%^*", ...)

```
public class NumberFormatExceptionDemo {
    public static void main(String[] args) {
        String str1 = "132.68";
        String str2 = "abcde";
        int num1 = Integer.parseInt(str1);
        double num2 = Double.parseDouble(str2);
    }
}

Exception in thread "main" java.lang.NumberFormatException: For input string: "132.68"
    at java.base/java.lang.NumberFormatException.forInputString(NumberFormatException.java:67)
    at java.base/java.lang.Integer.parseInt(Integer.java:588)
    at java.base/java.lang.Integer.parseInt(Integer.java:685)
    at NumberFormatExceptionDemo.main(NumberFormatExceptionDemo.java:5)
```


페이지 11

이번에는 java.lang.NumberFormatException 입니다.
str1에는 string “132.68” 이 assign 되었고
str2에는 string “abcde” 가 assign 되었습니다.
Wrapper class인 Integer.parseInt(str1) 을 이용하여
“132.68” 을 int 로 convert하려고 하는 순간
NumberFormatException이 일어납니다.
“132.68” 은 소수점 이하가 존재하는 실수를 나타내는 String 이므로
이를 int 로 바꾸는 것이 exception을 발생시키는 것입니다.
이 때 프로그램은 즉시 중단됩니다.
그러나 만일 Integer.parseInt() 가 없었다면
그 아래 줄의 Double.parseDouble(“abcde”) 도
같은 exception인 NumberFormatException을 발생시킬 것입니다.
“abcde”는 double type의 숫자로 바꿀 수 없기 때문입니다.
아래쪽의 메시지는 Integer.parseInt()에서 프로그래밍 중단된 후
프린트 되는 메시지를 보여줍니다.
이 프로그램도 역시 exception handling은 하지 않았고
어떻게 handling을 할 수 있을지는
다음 노트에서 자세하게 학습할 것입니다.

Runtime Exception: InputMismatchException

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

public class InputMismatchExceptionExample {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Input an integer: ");
        int number = scanner.nextInt();  // assume input "abcd"
        System.out.println("Input Integer: " + number);
        scanner.close();
    }
}
```

```
Input an integer: abcd
Exception in thread "main" java.util.InputMismatchException
       at java.base/java.util.Scanner.throwFor(Scanner.java:964)
       at java.base/java.util.Scanner.next(Scanner.java:1619)
       at java.base/java.util.Scanner.nextInt(Scanner.java:2284)
       at java.base/java.util.Scanner.nextInt(Scanner.java:2238)
       at InputMismatchExceptionExample.main(InputMismatchExceptionExample.java:8)
```


페이지 12

이번에는 InputMismatchException에 대해 살펴 보겠습니다.
이 runtime exception은 Scanner를 사용하여 input하는 값이
읽어들이는 next...() method가 기대하는 type과 다른 경우,
또는 읽어들이는 value가 range를 벗어날 경우 발생하게 됩니다.
InputMismatchException은 java.util package에 있습니다.
예제 코드에서는 먼저 java.util.InputMismatchException과
java.util.Scanner를 import 하였습니다.
Scanner object인 scanner를 생성한 후
"Input an integer: " 라는 prompt를 print하였습니다.
여기에서 사용자가 "abcd" 라는 input을 입력하였다면
scanner.nextInt()를 call하는 부분에서
InputMismatchException이 발생하게 됩니다.
왜냐하면 "abcd" 를 정수로 읽어들일 수 없기 때문입니다.
이 때 프로그램은 즉시 중단되며
아래쪽 박스 안의 메시지들을 프린트하게 됩니다.
메시지 맨 끝 줄의 main method부터 시작하여
method call들이 순서대로 보여지는
runtime stack의 현재 상태가
프린트되고 있습니다.

Runtime Exception: ArithmeticException

```java
public class ArithmeticExceptionExample {
    public static void main(String[] args) {
        int dividend = 10;
        int divisor = 0;
        int result = dividend / divisor;
        System.out.println("Result: " + result);
    }
}
```

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
       at ArithmeticExceptionExample.main(ArithmeticExceptionExample.java:5)
```


페이지 13

java.lang.ArithmeticException도 또다른 runtime exception 입니다.
이 exception이 일어나는 대표적인 경우는
0으로 나누는 arithmetic operation이 시도되는 경우 입니다.
예제 프로그램에서도 0인 divisor 값으로 dividend 10을 나누려 시도했는데
아래의 box 안의 메시지가 프린트되면서
프로그램은 즉시 실행을 멈추게 됩니다.

Runtime Exception: ClassCastException (1/2)

- Occurs when type conversion between classes is not possible

```
class Vehicle { };
class Auto extends Vehicle { };
class Bicycle extends Vehicle { };

public class ClassCastExceptionDemo {
    public static void main(String[] args) {
        Vehicle vec1 = new Auto();
        Vehicle vec2 = new Bicycle();

        Auto auto1 = (Auto) vec1;    // OK.. vec1's original class is Auto
        if (vec1 instanceof Auto) {  // preventing wrong conversion
            Auto auto2 = (Auto) vec1;
        }

        Bicycle by = (Bicycle) vec2; // OK.. vec2's original class is Bicycle
        Auto auto3 = (Auto) vec2;    // ClassCastException
                                     // vec2's original class is Bicycle, not Auto
    }
}
```


페이지 14

runtime exception인 java.lang.ClassCastException에 대해 살펴봅니다.
class Vehicle이 정의되었고
class Auto와 class Bicycle은 Vehicle을 inherit한 children들입니다.
Vehicle type variable인 vec1과 vec2에
각각 Auto와 Bicycle object를 assign하였습니다.
이 때 Auto type인 auto1과 auto2에
vec1을 downcasting하는 것은 가능합니다.
vec1의 original class가 Auto이기 때문입니다.
또 Bicycle type인 by에 vec2를 downcasting하는 것도 가능합니다.
vec2의 original class가 Bicycle이기 때문입니다.
그러나 맨 마지막에 Auto type인 auto3에
vec2를 downcasting하는 것은 ClassCastException을 발생시킵니다.
vec2의 original class가 Auto가 아니라 Bicycle이었기 때문입니다.
이와 같이 ClassCastException은
hierarchy 관계로 보았을 때 assign할 수 없는 object들을
assign하려 할 때에 발생하게 됩니다.

Runtime Exception: ClassCastException (2/2)

```
Exception in thread "main" java.lang.ClassCastException: class Bicycle cannot be cast to
class Auto (Bicycle and Auto are in unnamed module of loader 'app')
    at ClassCastExceptionDemo.main(ClassCastExceptionDemo.java:16)
```


페이지 15

이 box안의 message는 ClassCastException이 발생한 당시에 프린트되는 메시지를 보여주고 있습니다.