**[ 14 ] 예외(Exception) 처리**

* 예외란
* 예외처리의 필요성
* 예외처리의 문법
* 일반적으로 많이 보게 되는 예외들

1. 예외란?

문제 없을 것 같은 프로그램도 외부환경요인 등에 의해서 문제가 발생하곤 합니다.

프로그램에서 문제가 발생될 만한 곳을 예상하여 사전에 "문제가 발생하면 이렇게 해라"라고 프로그래밍 하는 것을 예외 처리라고 합니다.

|  |  |  |
| --- | --- | --- |
| 에  러 | 문법에러 | 문법적으로 나타나는 에러 |
| 실행에러  (실행 시 에러) | 논리에러 : 프로그래머가 잘못 작성해서 이상한 결과가 나오는 경우 |
| 시스템에러 : 프로그래머의 의지와 상관없이 나타나는 에러 |
| 예외사항 : 정상적으로 프로그램이 동작하는 과정에서 나타나는 에러  프로그램 실행 중에 발생하는 예기치 않는 사건  **ex. 정수를 0으로 나누는 경우, 배열 인덱스 벗어난 경우,**  **부적절한 형변환이 일어나는 경우**  **입출력을 위한 파일이 없는 경우** |

2. 예외처리의 필요성

프로그래머는 예외 처리를 왜 해야 될까요? 실행 중인 시스템이 단순한 어떤 이유로 작동을 하지 않다면 프로그램의 프로세스가 멈춥니다. 다시 재가동을 시키지 않으면 움직이지 않는 상태가 된 겁니다.

예외의 필요성은 어느 한 부분에서 예외가 발생하더라도 계속해서 프로그램이 동작되도록 하는데 목적이 있습니다. 어떤 문제가 하나 발생되었다고 그 프로세스가 완전히 정지되어 시스템을 재가동하기 전까지는 구동을 아예 못한다고 한다면 너무 많은 손실이 클 것입니다. 그래서 사소한 문제 같은 경우에는 그 문제를 우회해서 가는 방법으로 프로세스가 죽지 않고 계속 구동하도록 프로그래밍하는 방법이 필요합니다

예를 들어,

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

Scanner scanner = **new** Scanner(System.***in***);

System.***out***.print("원하는 첫번째 정수 i는 ? ");

**int** i = scanner.nextInt();

System.***out***.print("원하는 두번째 정수 j는 ? ");

**int** j = scanner.nextInt();

System.***out***.println("i/j = "+ (i/j));

System.***out***.println("i\*j = "+ (i\*j));

System.***out***.println("i+j = "+ (i+j));

System.***out***.println("i-j = "+ (i-j));

scanner.close();

}

사용자한테 두 번의 숫자를 입력 받아 곱셈, 나눗셈, 덧셈, 뺄셈을 하는 프로그램이 있습니다. 사용자가 ‘10’과 ‘2’를 입력하면 결과는 다음과 같을 것입니다.

10 \* 2 = 20, 10 / 2 = 5, 10 + 2 = 12, 10 - 2 = 8

이 경우에는 아무런 이상이 없습니다. 하지만 사용자가 10과 0을 입력하는 순간 문제가 발생합니다.

10 \* 0 = 0, 10 / 0 = ?

0으로 나눌 수 없어 프로그램이 중단될 것입니다. 이런 경우 덧셈과 뺄셈은 이상이 없어도 사용자는 프로그램 동작이 중단되어 결과를 볼 수가 없습니다.

만약, 나눗셈 처리에 예외처리를 해 두었다면, 나눗셈의 결과는 못 보더라도 덧셈과 뺄셈의 결과는 볼 수 있었을 것입니다. 이것이 바로 예외처리가 필요한 이유입니다.

3. 예외처리 문법(try ~ catch)

**try {**

**try블럭 ; 익셉션이 발생할 가능성이 있는 명령문들(문제가 발생할 수 있는 로직을 기술)**

**}catch(익셉션타입 익셉션변수) {**

**그 익셉션을 처리하는 명령문(try블록안에서 문제가 발생했을 때 대처방안 기술);**

**} finally {**

**익셉션 발생 여부와 상관없이 맨 마지막에 실행할 명령문;**

**}**

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

Scanner scanner = **new** Scanner(System.***in***);

System.***out***.print("원하는 첫번째 정수 i는 ? ");

**int** i = scanner.nextInt();

System.***out***.print("원하는 두번째 정수 j는 ? ");

**int** j = scanner.nextInt();

**try**{

System.***out***.println("i/j = "+ (i/j)); //문제가 발생할 수 있는 로직을 try 처리 }**catch** (ArithmeticException e) {

System.***out***.println(e.getMessage());//문제가 발생할 경우 처리하는 부분

e.printStackTrace();//예외 메시지를 좀 더 자세히 볼 수 있다

//문제가 발생하여도 catch로 처리하고 나머지 작업 계속 진행

}

System.***out***.println("i\*j = "+ (i\*j));

System.***out***.println("i+j = "+ (i+j));

System.***out***.println("i-j = "+ (i-j));

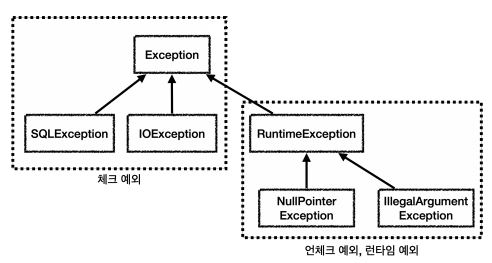
scanner.close();

}

자바는 예외를 객체로 취급. 예외가 발생하면 예외 객체가 자동 생성된다.

Exception

* IOException
* ClassNotFoundException
* ….
* RuntimeException
  + ArithmeticException
  + ClassCaseException
  + NullPointerException
  + …
  + IndexOutOfBoundsException



* 체크 예외(RuntimeException외 Exception) : 컴파일러가 체크하는 예외 . 컴파일 시점에 try-catch를 않으면 항상 throws에 던지는 예외 선언해야 한다.
* 언체크 예외(RuntimeException) : 컴파일러가 체크하지 않는 예외. try-catch를 하지 않아도 throws를 생략할 수 있다.

<catch문에 예외처리를 여러 개 할 수도 있다>

try {

try블럭 ; 익셉션이 발생할 가능성이 있는 명령문들

}catch(**ArrayIndexOutOfBoundsException a**) {

**ArrayIndexOutOfBounds 익셉션을 처리하는 명령문**

} catch(**NumberFormatException n**) {

**NumberFormat 익셉션을 처리하는 명령문**

} catch(**Exception e**) {

**그 외 익셉션을 처리하는 명령문**

} finally {

익셉션 발생 여부와 상관없이 맨 마지막에 실행할 명령

}

<예제>

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

Scanner scanner = **new** Scanner(System.***in***);

**int** i=0, j=0;

System.***out***.print("원하는 첫번째 정수 i는 ? ");

**try**{

i = scanner.nextInt();

}**catch** (InputMismatchException e) {

System.***out***.println("숫자만 입력하셔야지 문자를 입력하시면 끝내요");

System.*exit*(-1);

}

System.***out***.print("원하는 두번째 정수 j는 ? ");

**try**{

j = scanner.nextInt();

}**catch** (InputMismatchException e) {

System.***out***.println("숫자만 입력하셔야지 문자를 입력하시면 끝내요");

System.*exit*(-1);

}

System.***out***.println("i="+i+"\tj="+j);

System.***out***.println("i\*j = "+ (i\*j));

**try**{

System.***out***.println("i/j = "+ (i/j));

}**catch** (ArithmeticException e) {

System.***out***.println(e.getMessage());

e.printStackTrace();

}

System.***out***.println("i+j = "+ (i+j));

System.***out***.println("i-j = "+ (i-j));

scanner.close();

}

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

**int** i=0, j=0;

Scanner scanner = **new** Scanner(System.***in***);

System.***out***.print("원하는 첫번째 정수 i는 ? ");

**try**{

i = scanner.nextInt();

System.***out***.print("원하는 두번째 정수 j는 ? ");

j = scanner.nextInt();

System.***out***.println("i="+i+"\tj="+j);

System.***out***.println("i\*j = "+ (i\*j));

System.***out***.println("i/j = "+ (i/j));

}**catch**(InputMismatchException e){

System.***out***.println("숫자만 입력하셔야 되요. 문자입력시 종료합니다");

System.*exit*(-1);

}**catch** (ArithmeticException e) {

System.***out***.print(e.getMessage());

System.***out***.println(" 0으로 나누면 안 된다고요");

}**catch** (Exception e) {

System.***out***.println(e.getMessage()+" 뭔 에러일까?");

}

System.***out***.println("i+j = "+ (i+j));

System.***out***.println("i-j = "+ (i-j));

scanner.close();

}

※ finally 끝내 실행합니다. finally 블럭은 try와 catch문의 영향 없이 무조건 실행됩니다

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

**int**[] iArr = {0,1,2};

**try**{

System.***out***.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.***out***.println(iArr[3]);

System.***out***.println("============================");

} **catch**(Exception e) {

System.***out***.println(e.getMessage()); // 예외 객체가 가지고 있는 메시지를 문자열로 return

//e.printStackTrace(); 예외의 발생 경로를 추적내용 출력

} **finally** {

System.***out***.println("무조건 실행합니다");

}

}

4. 예외처리 문법(throws)

try ~ catch 예외 처리 방법은 예외가 발생했을 때 자체적으로 catch문을 이용해서 해결 했습니다.

이번에 살펴볼 throws의 경우에는 나를 호출한(실행시키는) 쪽으로 예외를 던져버리는 방식 입니다.

<예제>

**public** **class** TestMain {

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

**new** ThrowsEx();

}

}

**public** **class** ThrowsEx {

**public** ThrowsEx() {

actionB();

}

**private** **void** actionB(){

System.***out***.println("actionB 전반부");

**try** {

actionA();

}**catch** (Exception e) {

System.***out***.println("예외는 여기서 처리할께요"+e.getMessage());

}

System.***out***.println("actionB 후반부");

}

**private** **void** actionA() **throws** ArrayIndexOutOfBoundsException{

System.***out***.println("actionA 전반부");

**int**[] arr = {0, 1, 2};

System.***out***.println(arr[3]);

System.***out***.println("actionA 후반부");

}

}

☞ 실행결과 :

actionB 전반부

actionA 전반부

예외는 여기서 처리할께요Index 3 out of bounds for length 3

actionB 후반부

5. 일반적으로 많이 보게 되는 예외들

예외(Exception)의 종류는 아주 많습니다. 정말 많습니다.

Exception클래스는 많은 예외들(자식클래스)를 가지고 있습니다. 모든 예외를 습득한다는 것은 불가능 하며, 그럴 필요도 없습니다.

프로그래밍을 하다 보면, 자주 보게 되는 예외들이 있습니다. 대표적인 몇 가지만 살펴보도록 하겠습니다.

JAVA API문서를 통해서 Exception클래스가 얼마나 많은 자식 클래스를 가지고 있는지 알 수 있습니다.

※ **ArrayIndexOutOfBoundsException** > 배열을 사용시 존재하지 않는 index값을 호출하면 발생 합니다.

int[] iArr = {0,1,2,3};

System.out.println(iArr[4]);

※ **NullPointerException** > 존재하지 않는 객체를 가리킬 때 발생 합니다

String str = null;

char c = str.charAt(0);

**public** **class** Friend {

**private** String name;

**private** String tel;

**public** Friend() {

// **TODO** Auto-generated constructor stub

}

**public** Friend(String name, String tel) {

**this**.name = name;

**this**.tel = tel;

}

@Override

**public** String toString() {

String post = tel.substring(tel.lastIndexOf("-")+1);

**return** name+"(\*\*\*-\*\*\*\*-"+post+")";

}

}

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

Friend hong = **new** Friend();

Friend hong2 = **new** Friend("홍2", "011-325-8888");

System.***out***.println(hong2);

System.***out***.println(hong);

}

※ **NumberFormatException** > 숫자로 변경할 수 없는 문자열을 변경하려고 함.

public static void main(String[] args) {

String str = "aa";

String str1 = "10 ";

int i = 10;

int j = Integer.parseInt(str)+i;

j = Integer.parseInt(str1.trim())+i;

}

※ DB 관련 Exception

ClassNotFoundException : 드라이브 이름을 찾지 못했을 때

SQLException : db url, id, pw가 올바르지 않을 때

등등 Exception의 종류만 봐도 왜 예외가 일어났는지 직관적으로 알 수 있다.

6. 강제로 예외 발생시키기(코드에서 예외 발생시키기)

|  |
| --- |
| **throw** **new** Exception(“에러메세지”); |

**public** **class** Account {

**private** String accountNo;

**private** String ownerName;

**private** **int** balance;

**public** Account(){}

**public** Account(String accountNo, String ownerName){

**this**.accountNo = accountNo;

**this**.ownerName = ownerName;

**this**.balance = 0;

}

**public** Account(String accountNo, String ownerName, **int** balance){

**this**.accountNo = accountNo;

**this**.ownerName = ownerName;

**this**.balance = balance;

}

**public** **void** deposit(**int** amount){//예금

System.***out***.println(**this**);

System.***out***.println(amount+"예금함");

balance += amount;

System.***out***.println(**this**);

}

**public** **void** withdraw(**int** amount) **throws** Exception{

System.***out***.println(**this**);

**if**(balance<amount)

**throw** **new** Exception(amount+"를 인출하기엔 잔액("+balance+")이 부족합니다");

balance -= amount;

System.***out***.println(amount+"인출함");

System.***out***.println(**this**);

}

@Override

**public** String toString() {

String temp = "계좌번호 : "+ accountNo;

temp += "\t예금주 :"+ownerName;

temp += "\t잔액:"+balance;

**return** temp;

}

}

**public** **class** MainClass {

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

Account ob1 = **new** Account("11-11", "홍길동");

Account ob2 = **new** Account("22-22", "김철수", 1000);

ob1.deposit(1000);

**try** {

ob2.withdraw(1000);

} **catch** (Exception e) {

System.***out***.println("예외메세지 : "+e.getMessage());

// 관리자에게 경고를 보냅니다

}

**try** {

ob1.withdraw(3000);

} **catch** (Exception e) {

System.***out***.println("예외메세지 : "+e.getMessage());

// 관리자에게 경고를 보냅니다

}

}

}

예외처리는 보험이다.

예외가 발생 했을 때 프로세스가 죽지 않게 예외처리를 자체적으로 처리할 수도 있고(try~catch), 내가 처리 안하고 나를 부른데서 알아서 할테니 걍 던져 버릴 수도 있다(throws)