Day15-JAVA

태그	
날짜	@2022년 10월 20일

예외 처리 Exception

어떤 언어든지 프로그램을 만들다 보면 수많은 예외/오류가 발생하게 된다.

하지만 그 오류를 무시할 때도 있고 적절하게 처리해줄 때도 있다.

예외처리를 잘하게 되면 문제점을 미리 파악하고 전체적인 흐름을 판단할 줄 아는 사람이라는 의미가 될 수도 있다.

- 자바에서의 예외 2가지
  1. 컴파일 할 때 발생하는 예외로 아예 프로그램 실행 조차 되지 않는 예외
  2. 컴파일도 잘 되서 프로그램이 정상적으로 실행되다가 중간에 에러가 발생하는 경우 (runtime exception)

```
package Exception_exam;

public class exception {

  public static void main(String[] args) {
    // TODO Auto-generated method stub



    //예외 처리 예
    //어떤 수를 0으로 나눴을때
    int c = 4 / 0;
    System.out.println(c);
  }

}


// 어떤 수를 0으로 나누면 에러가 발생한다.
```

```
package Exception_exam;

public class exception {

  public static void main(String[] args) {
    // TODO Auto-generated method stub

    //index 에러
    int[] a = {1, 2, 3};
    System.out.println(a[3]);
  }
}

// 인덱스 에러
// 존재하지 않는 순번을 지정했을 때 예외가 발생한다.
```

- 예외 처리 방법

TRY

```
try{
    실행 구문
} catch (예외 1){
  예외 발생 시 실행
}
```

try 실행구문에서 예외가 발생하지 않으면 catch 구문은 아예 실행되지 않는다.

하지만 try 구문에서 예외가 발생을 하면 catch 구문이 실행된다.

```
package Exception_exam;

public class exception {

  public static void main(String[] args) {
    // TODO Auto-generated method stub

    int c;
```

```
    try {
     c = 4 / 0;
    }catch(ArithmeticException e) {
      c= -1;
      System.out.println(c);
    }

  }

}

// 예외가 즉 에러가 발생하면 에러 메세지를 출력하고 강제 종료하지 않고 특정 메세지 여기서는
// -1을 출력하여 강제 종료되지 않게 할 수 있다.
```

Finally

프로그램 수행 도중에 예외가 발생하면 프로그램이 중지되거나 또는 예외 구문이 실행되도록 할 수 있다.

하지만 어떤 예외가 발생하더라도 반드시 실행되어야 하는 부분이 있다면 어떻게 해야할까

```
package Exception_exam;

public class Sample {

  public void shouldBeRun() {
    System.out.println("Thanks..");
  }

  public static void main(String [] args) {
    Sample sample = new Sample();

    int c;

    try {
      c = 4 / 2;
    }catch(ArithmeticException e) {
      c = -1;
    } finally {
    sample.shouldBeRun();
    }

    System.out.println(c);

  }

}
```

Finally 구문은 try 구문 실행중 예외 여부와는 관계없이 항상 무조건 마지막에 실행된다.

예외를 적극적으로 발생 시켜 보기

```
package Exception_exam;

public class ThrowExam {

  public void sayNick(String nick) {
  if("fool".equals(nick)) {

    return;
  }
  System.out.println("당신의 별명은" + nick + "입니다.");
  }
  public static void main(String[] args) {
    ThrowExam sample = new ThrowExam();
    sample.sayNick("fool");
    sample.sayNick("genious");
  }

}
```

예외 발생 Throw

```
FoolException.java

package Exception_exam;

public class FoolException extends RuntimeException{

}
```

```
ThrowExam.java

package Exception_exam;
```

```
public class ThrowExam {

  public void sayNick(String nick) {
  if("fool".equals(nick)) {

    throw new FoolException(); // 에러를 강제로 발생 시킴
  }
  System.out.println("당신의 별명은" + nick + "입니다.");
  }
  public static void main(String[] args) {
    ThrowExam sample = new ThrowExam();
    sample.sayNick("fool");
    sample.sayNick("genious");
  }

}
```

예외 던지기 Throws

위에서 보면 sayNick 매서드에서 FoolException 을 발생시키고 예외 처리도 sayNick매서드에서 했는데 이렇게 하지 않고 호출한 곳에서 직접 예외처리를 할수 있도록 예외를 던져야 할 필요도 있다.

```
package Exception_exam;

public class ThrowsExam {

  public void sayNick(String nick) throws FoolException{

    if("fool".equals(nick)) {
        throw new FoolException(); // 에러를 강제로 발생 시킴
      }
      System.out.println("당신의 별명은" + nick + "입니다.");
}
  public static void main(String[] args) {
    ThrowsExam sample = new ThrowsExam();


    try {
    sample.sayNick("fool");
    sample.sayNick("genious");
    }catch (FoolException e) {
      System.out.println("FoolException 발생...!!");
```

```
    }
  }

}

//예외 처리를 예외가 발생한 메서드에서 직접하지 않고 그 매서드를 호출한 지점으오 예외를 던진다.
// 따라서 예외 처리 또한 호출한 지점에서 하게된다.
// 이렇게 되면 위에서 sample.sayNick("fool")에서 예외가 발생함에 따라 두번째 메서드는 동작하지
//않게 된다. 물론 예외처리를 하였기 때문에 시스템이 불완전하게 중지되는 경우는 발생하지 않는다.
```

여기서 생각해 볼것은 Transaction이다.

Transaction은 하나의 전체 작업 단위를 뜻한다.

예를 들면

- 송장 출력
- 제품 검수
- 제품 포장
- 제품 발송

위 4가지의 전체 작업을 트랜젝션이라 한다.

만약 위의 4가지 작업 중 하나라도 문제가 발생된다면 배송이 시작되어서는 안된다.

상품발송 () {

try{

- 송장 출력()
- 제품 검수()
- 제품 포장()
- 제품 발송()

} catch (예외)

제품 발송 취소 //위의 단계에서 하나라도 에러가 발생하면 전체 발송이 취소 되어야 한다.

송장 출력 ()throws 예외 {

}

제품 검수 ()throws 예외{

}

제품 포장 ()throws 예외{

}

제품 발송 ()throws 예외{

}