

예외 처리 (Exception Handling)

▶ 프로그램 오류



프로그램 수행 시 치명적 상황이 발생하여 비정상 종료 상황이 발생한 것, 프로그램 에러라고도 함

✓ 오류의 종류

- 1. 컴파일 에러 : 프로그램의 실행을 막는 소스 코드상의 문법 에러. 소스 코드 수정으로 해결.
- 2. 런타임 에러 : 프로그램 실행 중 발생하는 에러. **주로 if문 사용으로 에러 처리** (ex. 배열의 인덱스 범위를 벗어났거나, 계산식의 오류)
- 3. 시스템 에러 : 컴퓨터 오작동으로 인한 에러, 소스 코드 수정으로 해결 불가

✓ 오류 해결 방법

소스 코드 수정으로 해결 가능한 에러를 예외(Exception)라고 하는데 이러한 예외 상황(예측 가능한 에러) 구문을 처리 하는 방법인 예외 처리를 통해 해결





✓ Exception 확인하기

Java API Document에서 해당 클래스에 대한 생성자나 메소드를 검색하면

- 그 메소드가 어떤 Exception을 발생시킬 가능성이 있는지 확인 가능.
- -> 발생하는 예외를 미리 확인하여 상황에 따른 예외 처리 코드를 작성할 수 있음

✓ 예시

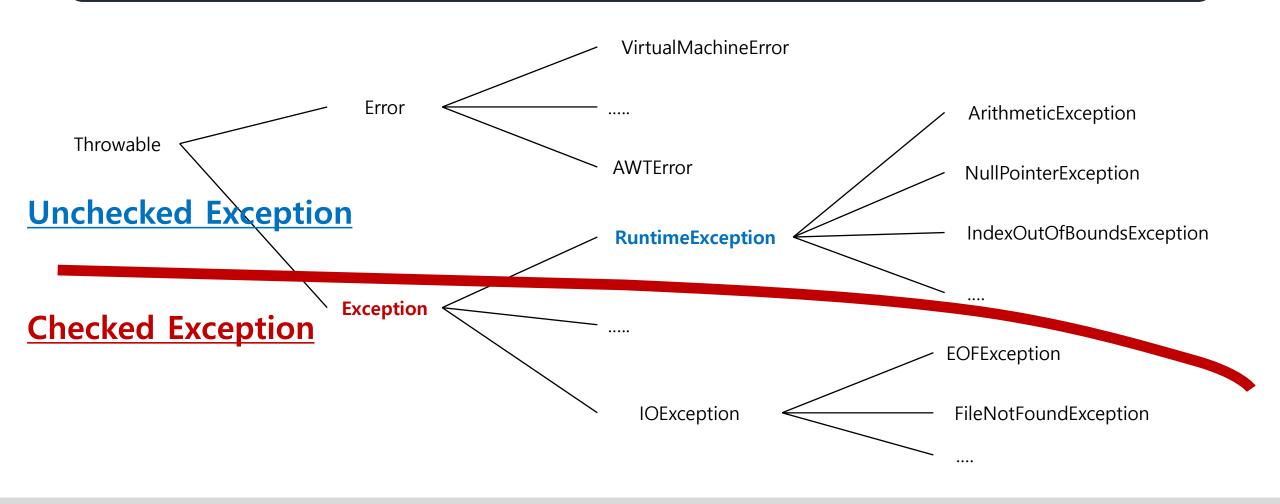
java.io.BufferedReader의 readLine() 메소드

readLine

▶ 예외 클래스 계층 구조



Exception과 Error 클래스 모두 Object 클래스의 **자손이며 모든 예외의 최고 조상은 Exception** 클래스 반드시 예외 처리해야 하는 Checked Exception과 해주지 않아도 되는 Unchecked Exception으로 나뉨





Unchecked Exception

✓ RuntimeException 클래스

Unchecked Exception으로 주로 <u>프로그래머의 부주의로 인한 오류</u>인 경우가 많기 때문에 예외 처리보다는 코드를 수정해야 하는 경우가 많음

✓ RuntimeException 후손 클래스

- ArithmeticException
 0으로 나누는 경우 발생
 if문으로 나누는 수가 0인지 검사
- NullPointerException

 Null인 참조 변수로 객체 멤버 참조 시도 시 발생

 객체 사용 전에 참조 변수가 null인지 확인
- NegativeArraySizeException 배열 크기를 음수로 지정한 경우 발생 배열 크기를 0보다 크게 지정해야 함

- ArrayIndexOutOfBoundsException 배열의 index범위를 넘어서 참조하는 경우 배열명.length를 사용하여 배열의 범위 확인
- ClassCastException
 Cast연산자 사용 시 타입 오류
 instanceof 연산자로 객체타입 확인 후 cast연산
- InputMismatchException
 Scanner를 사용하여 데이터 입력 시
 입력 받는 자료형이 불일치할 경우 발생



▶ 사용자 정의 예외

Java API에서 제공하는 Exception Class 만으로는 처리할 수 없는 예외가 있을 경우 사용자의 필요에 의해 생성하는 Exception Class.

Exception 발생하는 곳에서 throw new 예외클래스명()으로 발생

```
public class UserException extends Exception{
  public UserException() {}
  public UserException(String msg) {
     super(msg);
  }
}
```

```
public class UserExceptionController {
   public void method() throws UserException{
     throw new UserException("사용자정의 예외발생");
   }
}
```

```
public class Run {
   public static void main(String[] args) {
     UserExceptionController uc
         = new UserExceptionController();
     try {
         uc.method();
      } catch(UserException e) {
         System.out.println(e.getMessage());
```





1. Exception이 발생한 곳에서 직접 처리

try~catch문을 이용하여 예외 처리

- try: Exception 발생할 가능성이 있는 코드를 안에 기술
- catch: try 구문에서 Exception 발생 시 해당하는 Exception에 대한 처리 기술 여러 개의 Exception 처리가 가능하나 Exception간의 상속 관계 고려해야 함
- **finally**: Exception 발생 여부와 관계없이 꼭 처리해야 하는 로직 기술 중간에 return문을 만나도 finally구문은 실행되지만 System.exit();를 만나면 무조건 프로그램 종료 주로 java.io나 java.sql 패키지의 메소드 처리 시 이용



▶ 예외 처리 방법

✓ try ~ catch로 예외 처리

```
public void method() {
   BufferedReader br = null;
   try {
        br = new BufferedReader(new InputStreamReader(System.in));
       System.out.print("입력 : ");
       String str = br.readLine();
       System.out.println("입력된 문자열 : " + str);
    } catch (IOException e) {
        e.printStackTrace();
```



▶ 예외 처리 방법

✓ try ~ catch ~ finally로 예외 처리 후 반드시 수행해야 하는 로직 처리

```
public void method() {
    BufferedReader br = null;
    try {
        br = new BufferedReader(new InputStreamReader(System.in));
        System.out.print("입력 : ");
        String str = br.readLine();
        System.out.println("입력된 문자열 : " + str);
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        try {
            System.out.println("BufferedReader 반환");
            br.close();
        } catch (IOException e) {
            e.printStackTrace();
```

▶ 예외 처리 방법



2. Exception 처리를 호출한 메소드에게 위임

- 메소드 선언 시 throws Exception명을 추가하여 호출한 상위 메소드에게 처리 위임

- 계속 위임하면 main() 메소드까지 위임하게 되고 main() 메소드에서도 처리되지 않는 경우 프로그램이 비정상 종료됨.





✓ throws로 예외 던지기

```
public static void main(String[] args) {
        ThrowsTest tt = new ThrowsTest();
        try {
                tt.methodA();
        } catch (IOException e) {
                e.printStackTrace();
        } finally {
               System.out.println("프로그램 종료");
```

```
public void methodA() throws IOException {
        methodB()
public void methodB() throws IOException {
        methodC();
public void methodC() throws IOException {
         throw new IOException();
        // IOException 강제 발생
```

▶ Exception과 오버라이딩



오버라이딩 시 throws하는 Exception의 개수와 상관없이 처리 범위가 같거나 후손 이여아 함

* Exception 클래스는 상속이 될 수록 상위 클래스 보다 예외의 내용이 더 상세하게 기술됨.

EOFException은 IOException의 후손

Exception은 IOException의 부모