

Chap 09. Exceptions

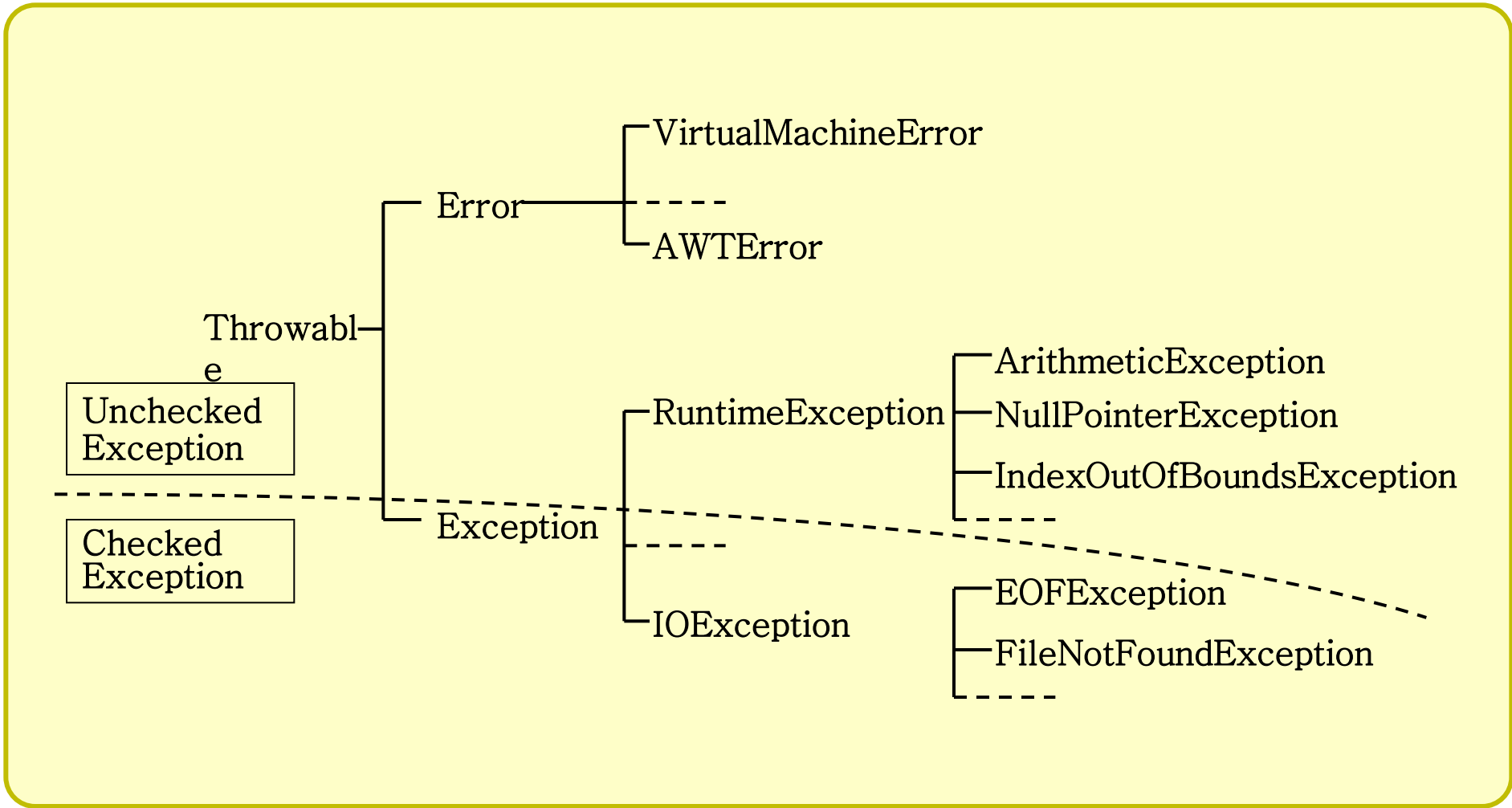
1. Exception Category
2. Exception 처리방법
3. Method overriding & Exception
4. 사용자 정의 Exception

❑ Error

- 프로그램 수행시, 치명적 상황 발생
- 처리가 불가능
- 예) 메모리 부족 등

❑ Exception

- 비교적 가벼운 에러
- 미리 예상하여 처리 가능
- 예) 파일 access시 파일 없음,
부적절한 parameter에 의한 메소드 호출 등



Exception 확인하기

API Document에서 해당 클래스에 대한 생성자나 메소드를 검색하면, 그 메소드가 어떠한 Exception을 발생시킬 가능성이 있는지 확인할 수 있다.

java.io.BufferedReader 의 readLine() 메소드 경우

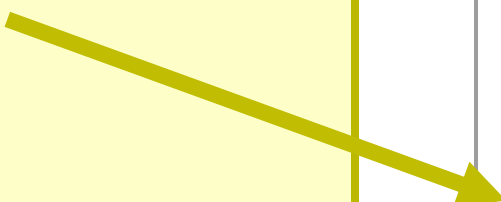
The screenshot shows the Java API documentation for the `readLine()` method of the `BufferedReader` class. The browser window title is "BufferedReader (Java 2 Platform SE v1.4.2) - Microsoft Internet Explorer". The address bar shows the URL "C:\wj2sdk\docs\api\index.html". The left sidebar lists various Java classes, with `BufferedReader` selected. The main content area displays the `readLine` method signature: `public String readLine() throws IOException`. Below the signature, there is a description: "Read a line of text. A line is considered to be terminated by any one of a line feed ('\\n'), a carriage return ('\\r'), or a carriage return followed immediately by a linefeed." It also includes "Returns:" and "Throws:" sections. The "Throws:" section states: "IOException - If an I/O error occurs". An orange arrow points from the `IOException` in the signature to the "Throws:" section. The bottom of the window shows the taskbar with the text "내 컴퓨터".

`readLine`

```
public String readLine()  
        throws IOException
```


- Unchecked Exception이며, 주로 프로그래머의 부주의로 인한 Bug인 경우가 많기 때문에 Exception 처리보다는 코드를 수정해야하는 경우가 많다.

- ArithmeticException
- NullPointerException
- NegativeArraySizeException
- ArrayIndexOutOfBoundsException
- ClassCastException



```
int sum = 0;
int total = 100;
int count = 0;

sum = total / count;
```



```
int sum = 0;
int total = 0;
int count = 0;

if ( count != 0 )
    sum = total / count;
```

- `ArithmeticException` : 0으로 나누는 경우 발생
→ if 문으로 먼저 나누는 수가 0인지를 검사.
- `NullPointerException` : null 인 ref.변수로 객체 멤버 참조 시도
→ 객체를 사용하기 전에 ref. 변수가 null인지 먼저 확인.
- `NegativeArraySizeException` : 배열 크기를 음수로 준 경우
→ 배열 size를 0보다 크게 지정.
- `ArrayIndexOutOfBoundsException` : 배열의 index 범위를 넘어서서 참조 하는 경우
→ 배열이름.length 를 써서 배열의 범위를 확인.
- `ClassCastException` : Cast 연산자 사용시 타입오류
→ instanceof 연산자를 이용하여, 먼저 객체 타입을 확인하고 Cast 연산자를 사용.

```
1 public class ExceptionTest {  
2     public static void main(String[] args) {  
3         String str = null;  
4  
5         String str2 = str.concat( "This is displayed" );  
6  
7         System.out.println( "Contents of str : " + str2 );  
8     }  
9 }
```



NullPointerException


실행결과

```
java.lang.NullPointerException  
    at ExceptionTest.main(ExcetionTest.java:5)  
Exception in thread "main"
```



```
1 public class ExceptionTest {  
2     public static void main(String[] args){  
3         String str = null;  
4         String str2 = null;  
5  
6         try {  
7             str2 = str.concat( "This is displayed" );  
8         } catch ( NullPointerException e ) {  
9             System.out.println( "null값입니다." );  
10        }  
11  
12        System.out.println( "Contents of str2 : " + str2 );  
13    }  
14 }
```

NullPointerException



finally 구문에는 오류 발생 여부에 관계 없이 처리 되어져야 하는 로직을 기술한다.

```
public String readFile() {  
  
    try {  
        파일을 연다.  
        파일을 읽는다.  
  
        if ( 파일오픈상태 == true )  
            파일을 닫는다.  
    } catch ( Exception e ) {  
        System.out.println( “개발자 문의” );  
    }  
  
    읽은 것을 return한다.  
}
```



수정

```
public String readFile() {  
  
    try {  
        파일을 연다.  
        파일을 읽는다.  
    } catch ( Exception e ) {  
        System.out.println( “개발자 문의” );  
    } finally {  
        if ( 파일오픈상태 == true )  
            파일을 닫는다.  
    }  
  
    읽은 것을 return한다.  
}
```

발생한 곳에서 직접 처리

- try

Exception 발생할 가능성이 있는 코드를 try 구문 안에 기술

- catch

try 구문에서 Exception 발생시 해당하는 Exception에 대한 처리 기술.

여러 Exception 처리 가능하나, Exception간 상속 관계 고려 해야 함

(자식 → 부모 순 기술, 부모 먼저 기술하면 컴파일 에러)

- finally

Exception 발생 여부에 관계 없이, 꼭 처리해야 하는 logic은 finally 에서 구현한다.

중간에 return문을 만나도, finally 구문은 실행한다.

단, System.exit(); 를 만나면 무조건 프로그램 종료한다.

주로 java.io , java.sql 패키지에 있는 메소드 처리시 많이 이용


Exception Handling 방법 1 – finally example

```
1  import java.io.*;
2  public class ReadFile {
3      public static void main ( String args[] ) {
4          BufferedReader in = null;
5          try {
6              in = new BufferedReader( new FileReader( "c:/data/test.txt" ) );
7              String s;
8              while ( ( s = in.readLine() ) != null ) {
9                  System.out.println( s );
10             }
11         } catch ( FileNotFoundException e ) {
12             System.out.println("파일이 없습니다.");
13         } catch ( IOException e) {
14             e.printStackTrace();
15         } finally {
16             try {
17                 if ( in != null )
18                     in.close();
19             }
20             catch ( IOException e ) { }
21         }
22     }
23 }
```

□ Exception 처리를 호출한 메소드에게 위임 : Call Stack Mechanism

- 메소드 선언시 throws *Exception_Name* 문을 추가하여 호출한 상위 메소드에게 처리 위임
- 계속적으로 위임하면, main() 까지 위임하게 되고 main() 까지 가서 try catch 없으면 비정상 종료된다.

```
public class ThrowTest {  
    public static void main( String[] args ) {  
        ThrowTest t = new ThrowTest();  
  
        try {  
            t.methodA();  
            System.out.println( "정상수행" );  
        } catch (IOException e) {  
            System.out.println( "IOException이 발생" );  
        }  
    }  
  
    public void methodA() throws IOException {  
        methodB();  
    }  
  
    public void methodB() throws IOException {  
        methodC();  
    }  
  
    public void methodC() throws IOException {  
        throw new IOException();  
    }  
}
```



- ❑ Overriding 시 throws하는 Exception은 같거나, 더 구체적인 것(자식)이어야 한다.

```
public class TestA {  
    public void methodA() throws IOException {  
        ...  
    }  
}  
  
public class TestB1 extends TestA {  
    public void methodA() throws EOFException { // → OK  
        ...  
    }  
}  
  
public class TestB2 extends TestA {  
    public void methodA() throws Exception { //→ compile error  
        ...  
    }  
}
```

```
public class Example {
    public static void main(String[] args) {
        SuperClass superClass = new SubClass();

        try {
            superClass.method(); //자식클래스의 Exception 발생
        } catch ( IOException ioe ) { // Exception을 잡을 수 없다.
            System.out.println( "IOException occurred!" ) ;
        }
    }
}

class SuperClass {
    public void method() throws IOException {
        throw new IOException();
    }
}

class SubClass extends SuperClass {
    //컴파일 오류가 안난다고 가정
    public void method() throws Exception {
        throw new Exception(" Exception");
    }
}
```


- ❑ Exception 클래스를 상속하여 작성
- ❑ Exception 발생할 곳에서, `throw new UserDefineException()`

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

```
public class MyExceptionTest {  
    public static void main( String[] args ) {  
        int age = 20;  
  
        try {  
            if ( age < 19 ) {  
                throw new MyException( "좀 더 크고 오세요" );  
            } else {  
                System.out.println( "즐감" );  
            }  
        } catch ( MyException me ) {  
            System.out.println( me );  
        }  
    }  
}
```