

예외처리 (Exception)

▶ 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 상의 문법 에러, 소스 구문을 수정하여 해결 ->
2. 런타임 에러 : 입력 값이 틀렸거나, 배열의 인덱스 범위를 벗어났거나, 계산식의 오류 등 주로 if문 사용으로 에러 처리
3. 논리 에러 : 문법상 문제가 없고, 런타임 에러도 발생하지 않지만, 개발자의 의도대로 작동하지 않음.
4. 시스템 에러 : 컴퓨터 오작동으로 인한 에러, 소스 구문으로 해결 불가

▶ Exception(예외)

✓ Exception(예외)

프로그램 오류 중 적절한 코드에 의해서 수습될 수 있는 미약한 오류.
예외발생상황을 예측해서 미리 예외처리코드를 작성해 둬
NullPointerException, ArithmeticException, IOException 등.

✓ 예외 처리

프로그램을 만든 프로그래머가 상정한 정상적인 처리에서 벗어나는
경우에 이를 처리하기 위한 방법
예측 가능한 에러를 처리하는 것
프로그램의 비정상적인 종료를 막고, 정상적인 실행상태를 유지하기
위함

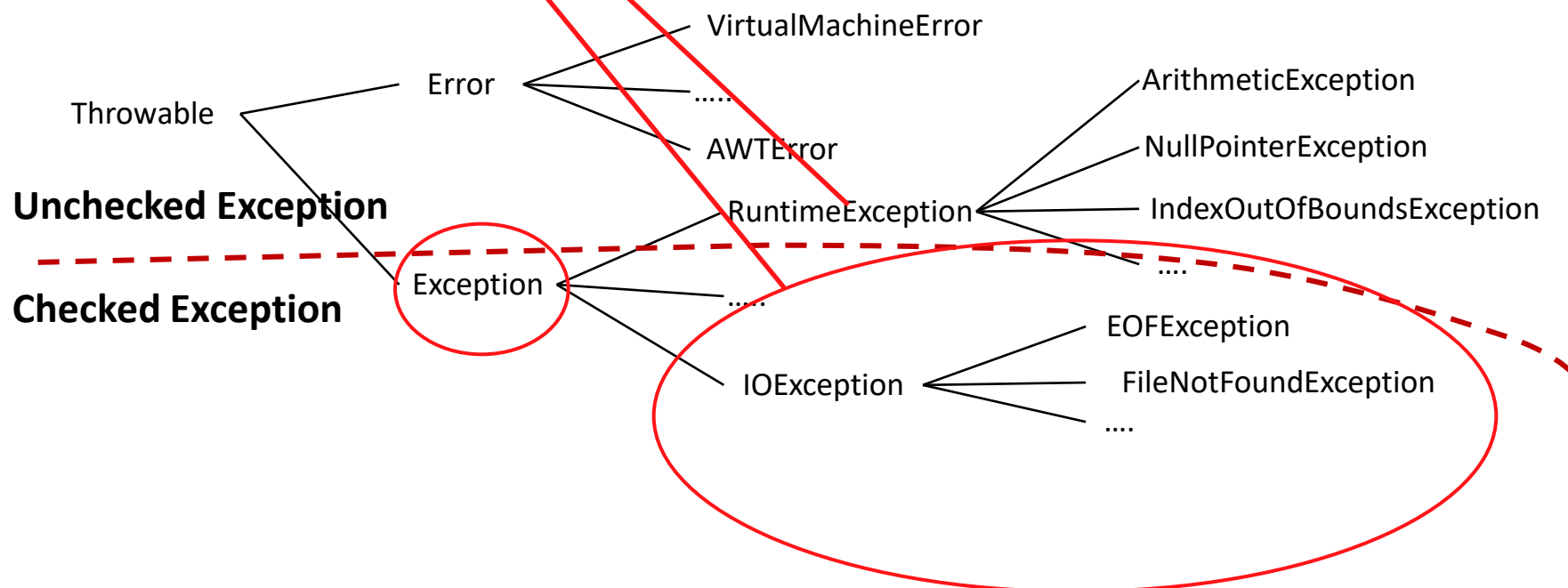
▶ 예외 클래스 계층 구조

Exception과 Error 클래스 모두 Object 클래스의 자손

모든 예외의 최고 조상은 **Exception** 클래스

Checked Exception : 소스코드 상에서 반드시 개발자가 처리해야 하는 예외

Unchecked Exception : 소스코드 상에서 개발자가 다룰 필요가 없는 예외



▶ RuntimeException 클래스

Unchecked Exception으로 주로 프로그래머의 부주의로 인한 오류인 경우가 많기 때문에 예외 처리보다 코드를 수정해야 하는 경우가 많음

예외처리를 강제화 하지 않음

✓ RuntimeException 후손 클래스

5
0
5

4

- **ArithmeticException**
0으로 나누는 경우 발생
if문으로 나누는 수가 0인지 검사
- **ArrayIndexOutOfBoundsException**
배열의 index범위를 넘어서 참조하는 경우
배열명.length를 사용하여 배열의 범위 확인
- **NegativeArraySizeException**
배열 크기를 음수로 지정한 경우 발생
배열 크기를 0보다 크게 지정해야 함
- **ClassCastException**
Cast연산자 사용 시 타입 오류
instanceof연산자로 객체타입
확인 후 cast연산
- **NullPointerException**
Null인 참조 변수로 객체 멤버
참조 시도 시 발생
객체 사용 전에 참조 변수가 null인지 확인

▶ Exception 확인하기

CHECKED EXCEPTION

Java API Document에서 해당 클래스에 대한 생성자나 메소드를 검색하면 그 메소드가 어떤 Exception을 발생시킬 가능성이 있는지 확인 가능.

해당 메소드를 사용하려면 반드시 뒤에 명시된 예외 클래스를 처리해야 함

가

✓ 예시

API

java.io.BufferedReader의 readLine() 메소드

readLine

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

java api 8

IOException

가
가 ->

1. if
2. try - catch
3. throws

▶ 예외처리 방법1

✓ Exception이 발생한 곳에서 직접 처리

try~catch문을 이용하여 예외처리

try : exception 발생할 가능성이 있는 코드를 안에 기술

가

catch : try 구문에서 exception 발생 시 해당하는 exception에 대한 처리 기술

여러 개의 exception처리가 가능하나 exception간의 상속 관계 고려

finally : exception 발생 여부와 관계없이 꼭 처리해야 하는 로직 기술

multi catch

중간에 return문을 만나도 finally구문은 실행되지만

가

finally System.exit();를 만나면 무조건 프로그램 종료

try - catch
finally
return
finally

finally 주로 java.io나 java.sql 패키지의 메소드 처리 시 이용

▶ 예외처리 방법1(try~catch)

✓ try~catch 표현식

```
try{  
    //반드시 예외 처리를 해야 하는 구문 작성  
}catch(처리해야할예외클래스명 참조형변수명){  
    //잡은 예외 클래스에 대한 처리 구문 작성  
}finally{  
    //실행 도중 해당 Exception이 발생을 하던,  
    //안하던 마지막에 반드시 실행해야 하는 구문 작성  
}
```


▶ 예외처리 방법1(try~catch)

✓ try~catch 구문 예시1

```
try {  
    System.out.println(100/0);  
} catch (Exception e) {  
    System.out.println("0으로 나눌 수 없습니다.");  
    System.out.println("예외 메시지 : "+e.getMessage());  
}  
System.out.println("프로그램 종료");
```

▶ 예외처리 방법1(try~catch)

✓ try~catch 구문 예시2

```
try {  
    System.out.println(100/0);  
} catch (Exception e) {  
    System.out.println("0으로 나눌 수 없습니다.");  
    System.out.println("예외 메시지 : "+e.getMessage());  
} finally {  
    System.out.println("프로그램 종료");  
}
```

▶ 예외처리 방법1(try~catch)

✓ try~catch 구문 예시3

```
Scanner sc = new Scanner(System.in);
System.out.println("----- 나눗셈 프로그램 -----");
System.out.print("첫번째 수 입력 : ");
int data1 = sc.nextInt();
System.out.print("두번째 수 입력 : ");
int data2 = sc.nextInt();
try {
    int result = data1 / data2;
    System.out.println("결과 : " + result);
} catch (Exception e) {
    System.out.println("0으로는 나눌 수 없습니다.");
}
```

▶ 예외처리 방법1(try~catch)

✓ 멀티catch 표현식

```
try{  
    //반드시 예외 처리를 해야 하는 구문 작성함  
}catch(예외클래스명3 e){  
  
    } catch(예외클래스명2 e){  
  
    } catch(예외클래스명1 e){  
  
    }
```

※ catch절의 순서는 상속관계를 따라 작성해야 함
후손클래스가 부모클래스보다 먼저 기술되어야 함
ex) FileNotFoundException -> IOException -> Exception

▶ 예외처리 방법1(try~catch)

✓ 멀티catch 구문 예시

```
BufferedReader br = null;
try {
    br = new BufferedReader(new FileReader("C:/data/text.txt"));
    String s;
    while((s = br.readLine()) != null) {
        System.out.println(s);
    }
} catch(FileNotFoundException e) {
    System.out.println("파일이 없습니다.");
} catch(IOException e) {
    e.printStackTrace();
} finally {
    try {
        if(br != null) br.close();
    } catch(IOException e) {}
}
```

▶ 예외처리 방법1(try~catch)

✓ try~with~resource 구문

자바 7에서 추가된 기능으로, finally에서 작성되었던 close()처리를 생각하고 자동으로 close처리 되게 하는 문장

✓ try~with~resource 표현식

```
try(반드시 close 처리 해야 하는 객체에 대한 생성 구문){  
    //예외 처리를 해야 하는 구문 작성함  
}catch(예외클래스명 레퍼런스){  
    //잡은 예외 클래스에 대한 처리 구문 작성함  
}finally{  
    //실행 도중 해당 Exception이 발생을 하던,  
    //안하던 반드시 실행해야 하는 구문 작성함  
}
```

▶ 예외처리 방법1(try~catch)

✓ try~with~resource 구문 예시

```
try (BufferedReader br=new BufferedReader(new
    FileReader("C:/data/text.txt"))){
    String s;
    while((s = br.readLine()) != null) {
        System.out.println(s);
    }
} catch(FileNotFoundException e) {
    System.out.println("파일이 없습니다.");
} catch(IOException e) {
    e.printStackTrace();
} catch(Exception e) {
    e.printStackTrace();
}
```

▶ 예외처리 방법2

✓ Exception 처리를 호출한 메소드에게 위임

메소드 선언 시 **throws** ExceptionName문을 추가하여 호출한 상위 메소드에게 처리 위임

계속 위임하면 main()메소드까지 위임하게 되고 거기서도 처리되지 않는 경우 비정상 종료 됨

▶ 예외처리 방법2(throws)

✓ 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("종료");  
    }  
}
```

```
public void methodA() throws IOException {  
    methodB();  
}
```

```
public void methodB() throws IOException {  
    methodC();  
}
```

```
public void methodC() throws IOException {  
    throw new IOException();  
    //Exception 발생  
}
```

▶ Exception과 오버라이딩

오버라이딩 시 throws하는 Exception 은 같거나 더 구체적인 것(후손), 처리 범위가 좁아야 함

[Object](#)

[Exception](#)

[SystemException](#)

[ArgumentException](#)

[ArgumentNullException](#)

[ArgumentOutOfRangeException](#)

[InvalidEnumArgumentException](#)

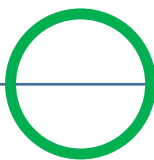
[DuplicateWaitObjectException](#)

API Document 참조


▶ Exception과 오버라이딩

```
public class TestA{  
    public void methodA() throws IOException{  
        ...  
    }  
}
```

```
public class TestB extends TestA{  
    public void methodA()  
        throws EOFException{  
        ...  
    }  
}
```



```
public class TestC extends TestA{  
    public void methodA()  
        throws Exception{  
        ...  
    }  
}
```



▶ 사용자 정의 예외

Exception 클래스를 상속받아 예외 클래스를 작성하는 것으로
Exception 발생하는 곳에서 throw new 예외클래스명()으로 발생

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

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

```
public class Run {  
    public static void main(String[] args) {  
        UserExceptionTest uet  
            = new UserExceptionTest();  
        try {  
            uet.method();  
        } catch(UserException e) {  
            e.printStackTrace();  
        }  
    }  
}
```