컴파일 에러와 런타임 에러, [자바] 예외처리

런타임

컴파일 과정을 마친 응용 프로그램이 사용자에 의해서 실행되어지는 때

런타임 에러

이미 컴파일이 완료되어 프로그램이 실행중임에도 불구하고, 의도치 않은 예외 상황으로 인하여 프로그램 실행중에 발생하는 오류 형태

- NullPointerException(생성되지 않은 객체를 참조할 때 발생)
- infinite loop(무한 루프)
- arithmeticException(0으로 나눴을 때 발생)
- 메모리 누수

→ try catch문으로 에러를 해결할 수 잇음

런타임 에러는 논리 에러와 시스템 에러가 있음

- 논리 에러: 프로그래머의 논리적 실수에 의해 발생하는 에러
- 시스템 에러: 프로그램 동작 중에 운영체제 또는 하드웨어에 문제가 발생해 프로그램이 정상적으로 동작하지 않는 경우에 발생하는 에러

컴파일타임

개발자에 의해 소스코드가 작성되며, 컴파일 과정을 통해 컴퓨터가 인식할 수 있는 기계어 코드로 변환되어 실행 가능한 프로그램이 되는 과정

컴파일타임 에러

소스코드가 컴파일되는 과정에서 발생하는 Syntax error, 파일 참조 오류 등과 같은 같은 문제들로 인해 컴파일이 방해되어 발생하는 오류

문법을 잘못 작성해 프로그램을 컴파일 할 수 없는 에러

에러 메시지를 통해 에러가 발생한 부분을 확인해 비교적 쉽게 해결이 가능함

- ;이 누락된 경우 문법 에러
- 괄호가 맞지 않는 구문 에러
- interface 사용시 함수의 구체적인 내용을 적지 않는 에러
- 타입 불일치
- 선언되지 않는 식별자

런타임과 컴파일의 차이점

런타임과 컴파일타임은 소프트웨어 프로그램개발의 서로 다른 계층을 나타냄

컴파일 에러는 프로그램이 성공적으로 컴파일링되는 것을 방해하는 Syntax error나 파일참조 오류와 같은 문제

컴파일러는 컴파일 타임 에러를 발생시키고 문제를 일으킨 소스코드 라인을 제시해줌

자바 예외 처리

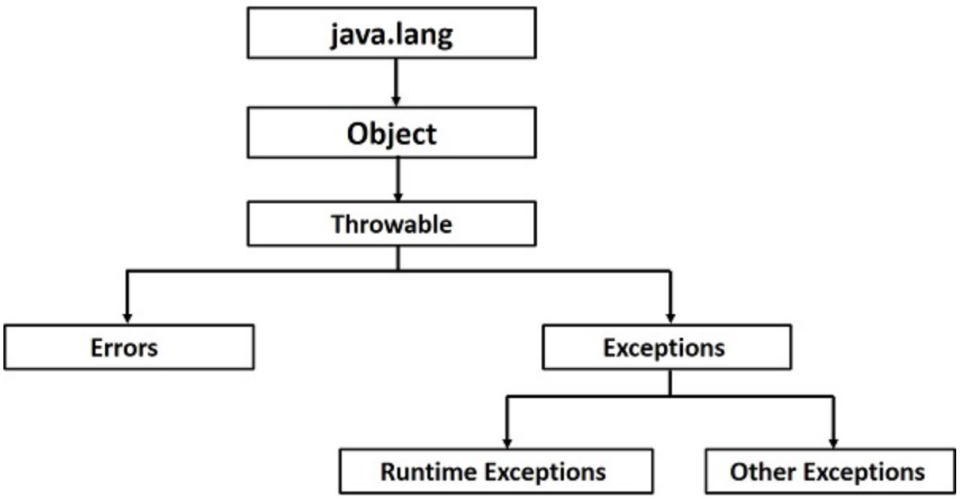
❓ 에러와 예외의 차이

에러는 한 번 발생하면 복구하기 어려운 수준의 문제를 의미하고, 대표적으로 메모리 부족인 OutOfMemoryError와 스택 오버플로우 StackOverFlowError가 있음

예외는 개발자의 잘못된 사용으로 인해 발생하는 에러는 상대적으로 약한 문제의 수준

즉, 개발자의 실수로 인해 발생하는 것

예외 클래스의 상속 계층도



자바에서는 예외가 발생하면 예외 클래스로부터 객체를 생성하여 해당 인스턴스를 통해 예외처리를 한다

자바의 모든 에러와 예외 클래스는 Throwable 클래스로부터 확장되며, 모든 예외의 상위 클래스는 Exception 클래스이다

또한 exception 클래스는 실행 예외 클래스(runtime exception)와 일반 예외 클래스(other exception)로 구분된다

- 일반 예외 클래스: 컴파일러가 코드 실행 전에 예외 처리 코드 여부를 검사하기 때문에 Checked 예외라고도 한다, 주로 잘못된 클래스명(ClassNotFoundExcecption)이나 데이터 형식(DataFormatException) 등 사용자의 편의 실수로 발생하는 경우가 많음
- 실행 예외 클래스: 컴파일러가 예외 처리 코드 여부를 검사하지 않아서 Unchecked 예외라고 부르기도 한다

  클래스 간의 형 변환 오류(ClassCastException), 배열의 범위를 벗어난 오류(ArrayIndexOutOfBoundsException), NPE 등이 있다

try-catch문

```
try {
    // 예외가 발생할 가능성이 있는 코드
} catch(Exception e){
    // 예외가 발생했을 경우 실행할 코드
} finally {
```

```
 // 예외 발생 여부와 상관없이 항상 실행
}
```

- try: 작성한 코드가 예외 없이 정상적으로 동작하면 catch는 실행되지 않고, 예외처리를 종료하거나 finally 볼록이 실행됨
- catch: 괄호 안에 Exception 클래스를 작성한다면, 모든 예외에 대해 처리가 가능하다
- finally: 항상 실행됨

만약, 문자열을 입력받아 대문자로 변환하여 출력해주는 코드에서 null값을 넣게 된다면, NullPointerException 예외가 발생함

```
public class ExceptionTest {
    static void printStr(String str) {
        String upperStr = str.toUpperCase();
        System.out.println(upperStr);
    }
    public static void main(String[] args) {
        printStr(null);
    }
}
```

```
Exception in thread "main" java.lang.NullPointerException
```

이를 try-catch문으로 예외 처리해보자

```
public class ExceptionTest {
    static void printStr(String str) {
        String upperStr = str.toUpperCase();
        System.out.println(upperStr);
    }
    public static void main(String[] args) {
        try {
            printStr("Hello");
```

```
            printStr(null);
        } catch (NullPointerException e) {
            System.out.println("예외 발생");
        } finally {
            System.out.println("finally 실행");
        }
    }
}
```

프로그램이 실행되는 도중에 null값을 printStr메소드에 넘기면서 호출되는 부분에서 예외가 발생한다

발생한 예외는 NPE으로 catch문의 조건에서 걸린 예외에 대하여 catch블록의 코드를 실행

예외 발생시 검사는 instanceOf 연산자를 통해 검사한다

이후 finally 블록을 실행하여 해당 내용인 'finally 실행' 문자열을 출력함

❗ catch가 여러 개일때, 순차적으로 검사를 진행한다

따라서, 상위 catch 블록에서 예외 처리를 하게 되면 하위에 존재하는 catch문은 실행되지 않는다

예외 전가

try-catch 문 외에 예외를 호출한 곳으로 다시 예외를 떠넘기는 것

```
void eat() throws NullPointerException, ArrayIndexOutOfBoundsExc
}
```

throws를 사용하면 자바 JVM이 최종적으로 예외의 내용을 콘솔에 출력하여 예외처리를 수행한다

```
public class ThrowsExcpetion {
    public static void main(String[] args) {
        try {
                throwException();
```

```
        } catch(ClassNotFoundException e) {
            System.out.println(e.getMessage());
        }
    }
    static void throwException() throws ClassNotFoundException,
        Class.forName("예외 발생");
    }
}
```

```
예외 발생
```

throws 키워드를 사용하여 해당 예외를 발생한 메소드 안에서 처리하는 것이 아닌, 메소드를 호출한 곳으로 다시 떠넘긴다. 따라서 예외 처리 책임은 throwException 메소드가 아닌 main 메소드가 가지게 된다

예외를 의도적으로 발생

throws 키워드와 유사한 throw키워드를 사용하면, 의도적으로 예외를 발생시킬 수 있다

```
public class ThrowException {
    public static void main(String[] args) {
        try {
                Exception intendedException = new Exception("의도
                throw intendedException;
            } catch(Exception e) {
                System.out.println(e.getMessage());
            }
        }
    }
```

```
의도적인 예외
```

classcastexception: 엔티티 클래스를 만들면

→ instanceof 연산자로 처리

numberformatexception, dataformexception

1. 로그 수집법
2. System.out.print: printStream 객체 , System.out.write: printWriter 비교하기

형변환 : int num = (int) 3.1414; → 예외처리가 아니라 형변환임

BufferedReader