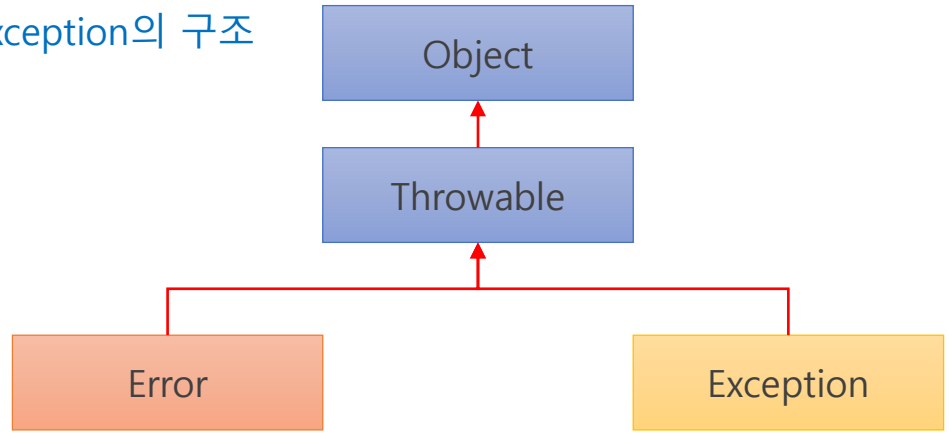
예외 처리

Error와 Exception

- Error
  - ✓ 시스템 레벨의 심각한 오류
  - ✓ 시스템 수정으로 문제를 해결
- Exception
  - ✓ 프로그램 레벨의 일반적인 오류
  - ✓ 코드 수정으로 문제를 해결

- 일반적으로 발생하는 문제는 Error가 아닌 Exception임

Error와 Exception의 구조



예외

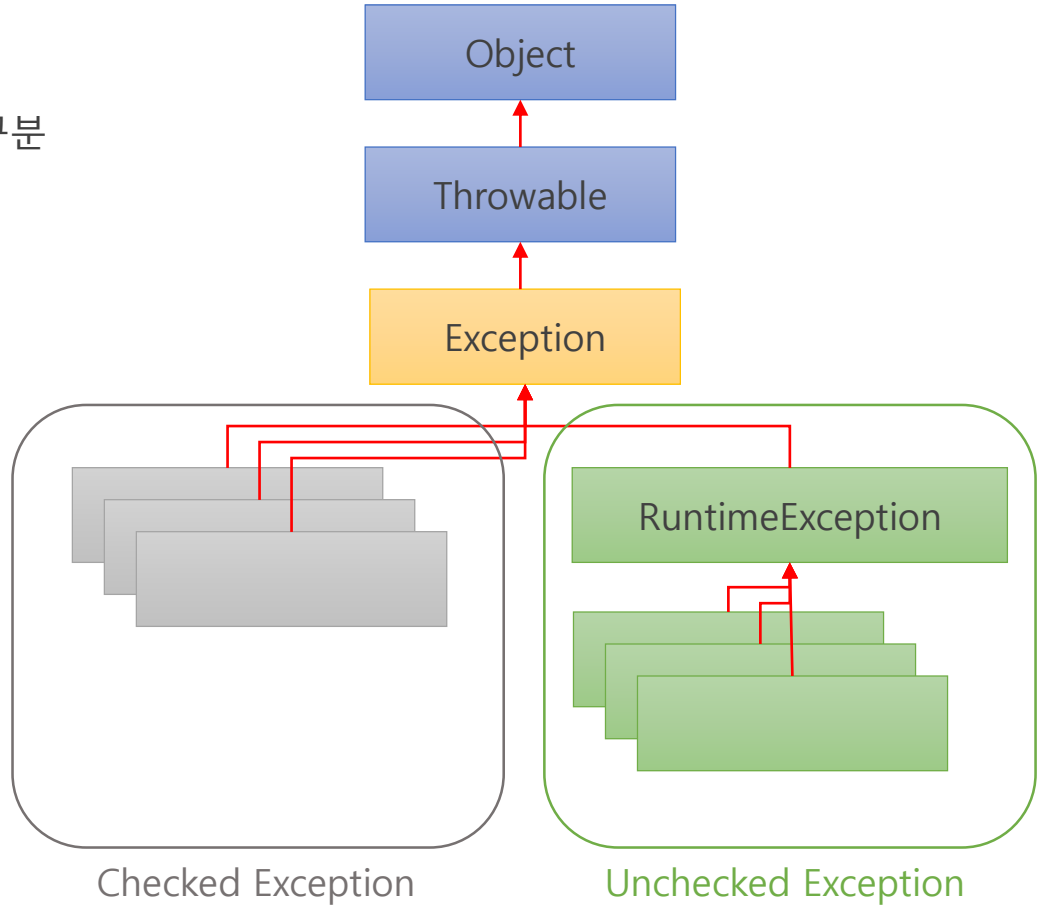
- 예외(Exception)
  - ✓ 프로그램의 실행 중 발생하는 여러 가지 오류를 의미함
  - ✓ 예외가 발생하면 프로그램이 강제로 종료됨
  - ✓ 예외가 발생하더라도 프로그램이 종료되지 않도록 예외처리를 해야 함
  - ✓ try - catch문을 이용해서 예외 발생으로 인해 프로그램이 종료되는 것을 회피할 수 있음

- 예외발생 예시

```
Exception in thread "main" java.lang.ClassCastException: class Person cannot be cast to class Student at
MainClass.main(MainClass.java:10)
```

Exception의 구조

- Exception 클래스
  - ✓ 모든 예외 클래스의 슈퍼클래스
  - ✓ 명시적 예외 처리 여부에 따라 Checked/Unchecked Exception으로 구분
- Checked Exception
  - ✓ RuntimeException 클래스의 자식클래스가 아닌 모든 예외 클래스
  - ✓ 반드시 try - catch문으로 예외 처리를 해야 함
  - ✓ IOException, SQLException 등
- Unchecked Exception
  - ✓ RuntimeException 클래스의 자식클래스로 등록된 모든 예외 클래스
  - ✓ try - catch문이 없어도 실행 가능
  - ✓ NullPointerException, NumberFormatException 등



예외 처리

- try - catch문
  - ✓ 예외를 처리할 때 사용하는 코드
  - ✓ 실행할 코드는 try 블록에 두고 예외를 처리할 코드는 catch 블록에 두는 방식
  - ✓ try 블록의 코드를 실행하다가 예외가 발생되면 발생된 예외는 자동으로 catch 블록으로 전달됨
  - ✓ 모든 예외는 자바 클래스로 만들어져 있음

- 형식

```
try {

    실행 코드

} catch(예외 타입 선언) {

    예외 처리 코드

}
```

예외 처리 흐름

- 정상 흐름

  ①→②→③→⑤ 순으로 진행

```
try {

    ① 정상 실행 코드
    ② 정상 실행 코드
    ③ 정상 실행 코드

} catch(예외 타입 선언) {

    ~~④ 예외 처리 코드~~

}

⑤ try - catch 이후 코드
```

- 예외 흐름

  ①→②→④→⑤ 순으로 진행

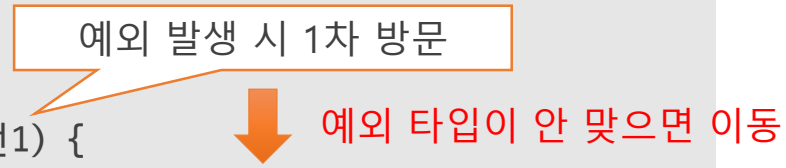
```
try {

    ① 정상 실행 코드
    ② 예외 발생 코드
    ~~③ 정상 실행 코드~~

} catch(예외 타입 선언) {

    ④ 예외 처리 코드

}

⑤ try - catch 이후 코드
```

다중 catch 블록

- 하나의 try 블록에 여러 개의 catch 블록이 추가될 수 있음
- try 블록에서 발생된 예외는 우선 첫 번째 catch 블록을 방문함
- 첫 번째 catch 블록이 처리하지 못하면 다시 두 번째 catch 블록을 방문함

- 형식

```
try {
    실행 코드
} catch(예외 타입 선언1) {
    예외 처리 코드1
} catch(예외 타입 선언2) {
    예외 처리 코드2
}
```



다중 catch 블록

- 예외 클래스들의 상속 관계를 파악한 뒤 catch 블록을 작성해야 함
- 슈퍼클래스 타입인 Exception이나 RuntimeException의 경우에는 마지막 catch 블록에서 처리

- 잘못된 catch 블록

```
try {


} catch(Exception e) {


} catch(IOException e) {
    절대 실행되지 않는 catch 블록
}
```

모든 예외는 Exception 클래스타입 이므로 모든 예외를 처리할 수 있음

실행할 수 없기 때문에 오류 발생

정상적인 catch 블록

```
try {
    실행 코드
} catch(IOException e) {


} catch(Exception e) {
    IOException 이외의 예외 처리 담당
}
```

IOException만 처리하고 다른 예외는 두 번째 catch 볼록이 처리함

정상 진행

finally 블록

- try - catch문 마지막에 추가할 수 있는 블록
- 예외 발생 여부와 상관 없이 항상 마지막에 실행되는 블록
- 필요 없는 경우 finally 블록은 생략할 수 있음
- 주로 어떤 자원을 반납(close)할 때 사용

- 형식

```
try {
    실행 코드
} catch(예외 타입 선언) {
    예외 처리 코드
} finally {
    무조건 실행되는 코드
}
```

throw

- 개발자가 직접 예외를 던지는 경우에 사용
- 자바는 예외로 인식하지 못하지만 실제로는 예외인 경우 개발자가 직접 예외를 발생시켜서 던짐
- 개발자가 직접 예외를 발생시킬 때는 RuntimeException을 사용하는 경우가 일반적임
- 개발자가 직접 만든 예외클래스를 던지는 것도 가능함

```
try {


    if(예외 상황이면) {
        throw new RuntimeException("예외 메시지");
    }

                          예외를  발생시켜서 던짐(throw)

} catch(RuntimeException e) {

    System.out.println(e.getMessage());  // 예외 메시지

}
```

throws

- 메소드에서 예외를 처리하는 방식
  ① 메소드 내부에 try-catch문을 두고 직접 처리하는 방식
  ② 메소드 외부로 예외를 던져서 메소드를 호출하는 곳에서 try-catch문으로 처리하는 방식

- 메소드 외부로 예외를 던질 때 throws문을 이용해 던지는 예외를 명시함
- 2개 이상의 예외를 명시할 수 있기 때문에 throw가 아닌 throws라고 함

```
public class ThrowsEx {

    public static void method() {
        try {
            메소드 내부 코드
        } catch(Exception e) {
            예외 처리 코드
        }
    }

    public static void main(String[] args) {
        method();
    }

}
```

메소드 내부에서 예외를 처리

```
public class ThrowsEx {

    public static void method() throws Exception {
        메소드 내부 코드
    }

    public static void main(String[] args) {
        try {
            method();
        } catch(Exception e) {
            예외 처리 코드
        }
    }

}
```

메소드 외부로 예외를 던짐

메소드 호출영역에서 예외를 처리

주요 예외클래스

구분	예외클래스	예외가 발생되는 경우
Unchecked Exception	ArithmeticException	산술 연산의 문제로 인해 발생(0으로 값을 나누는 경우)
	ClassCastException	어떤 객체를 변환할 수 없는 클래스타입으로 변환(Casting)하는 경우에 발생
	IndexOutOfBoundsException	배열의 인덱스가 범위를 벗어난 경우에 발생
	NullPointerException	null값을 참조하는 경우에 발생(null값으로 메소드를 호출할 때)
	NumberFormatException	String을 Number타입으로 변환하지 못하는 경우에 발생
Checked Exception	ClassNotFoundException	클래스이름을 찾을 수 없는 경우에 발생
	FileNotFoundException	파일을 찾을 수 없는 경우에 발생
	NamingException	자원(Resource)의 이름을 확인할 수 없는 경우에 발생
	IOException	입출력 동작이 실패하는 경우에 발생
	SQLException	데이터베이스 처리가 실패하는 경우에 발생