

예외처리 (Exception)

■ 경고(warning)

- 변수 선언 사용하지 않는 경우, 안내멘트
- 프로그램상에 문제는 없음.

■ 오류(error)

● 구문오류

- { { }
- Int int;
- **컴파일러**가 탐지..해서 바이트 코드를 생성하지 않는다.

● 실행오류

- 컴파일 문제 , 바이트코드 생성...
- 실행중에 오류가 발생하는 경우
- $1 / 2$, $2 / 0 \Rightarrow ?$ Divide by zero
- 프로그램이 중간 강제 종료 , **프로그램 코드에서 작성**

● 논리오류

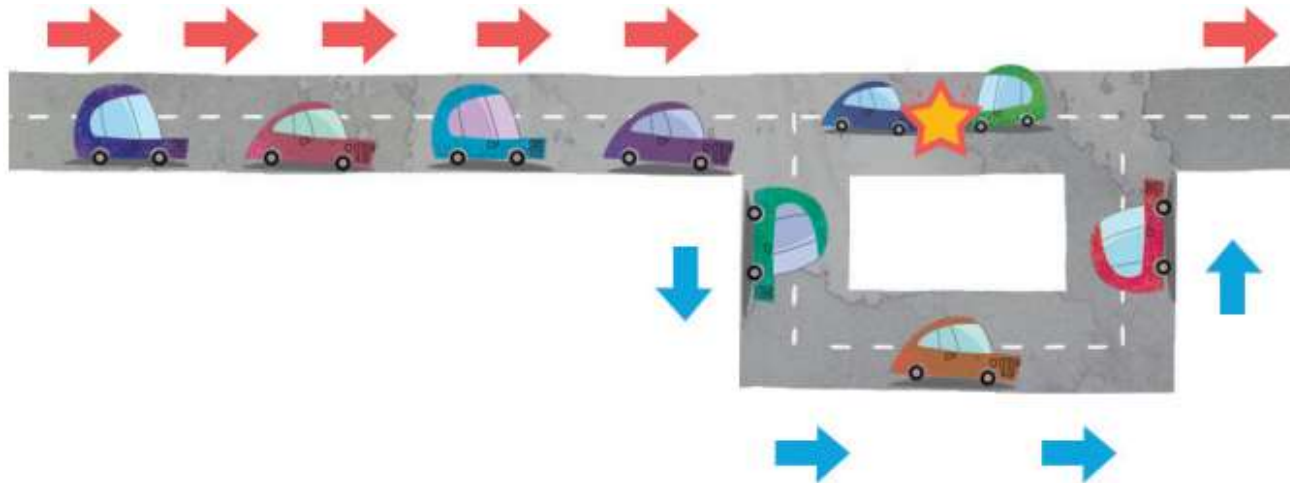
- 프로그램의 구문 없음. 바이트 코드생성

```
for(int i=0; i <= 10 ; i++) ;  
    sum += i;
```
- **프로그래머가 수정..**

예외

■ 의미

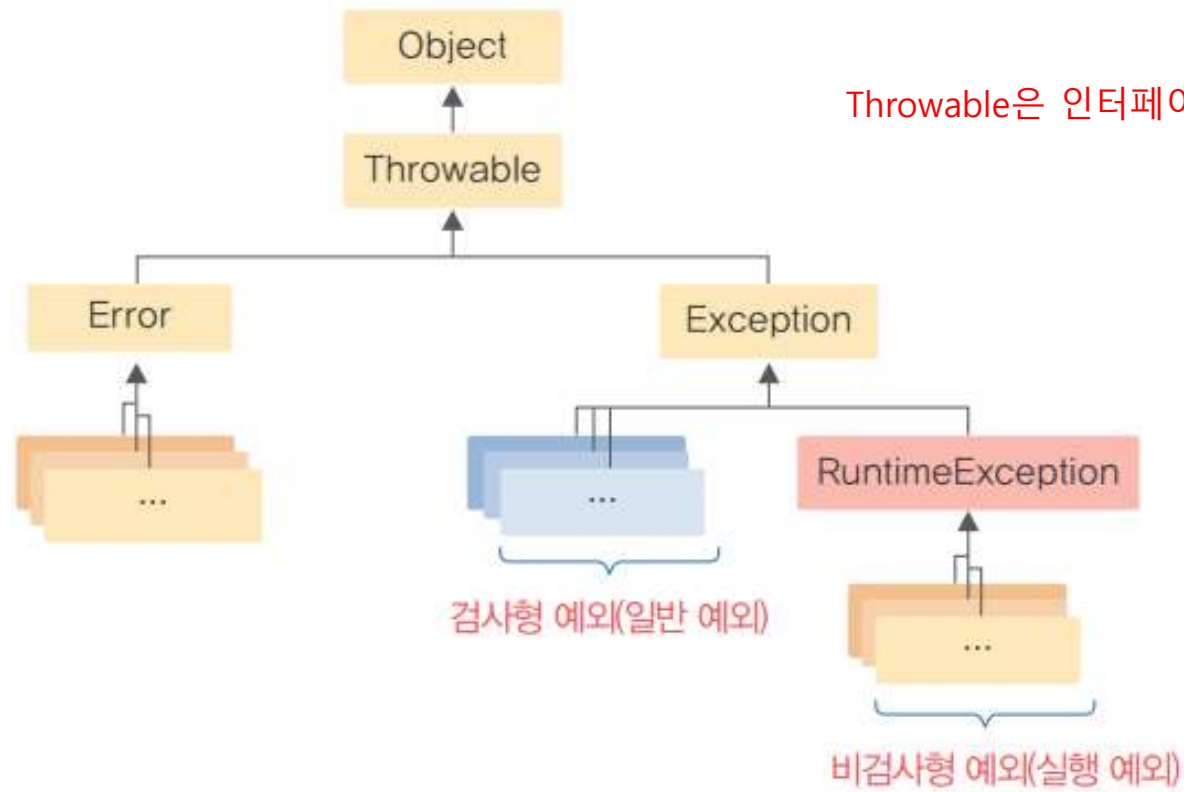
- 에러(error) : 개발자가 해결할 수 없는 치명적인 오류
- 예외(exception) : 개발자가 해결할 수 있는 오류
- 예외가 발생하면 비정상적인 종료를 막고, 프로그램을 계속 진행할 수 있도록 우회 경로를 제공하는 것이 바람직



예외

■ 종류

- 일반 예외와 실행 예외



Throwable은 인터페이스가 아니라 클래스

예외

■ 실행 예외

- 예외가 발생하면 JVM은 해당하는 실행 예외 객체를 생성
- 실행 예외는 컴파일러가 예외 처리 여부를 확인하지 않음. 따라서 개발자가 예외 처리 코드의 추가 여부를 결정
- 대표적인 실행 예외

실행 예외	발생 이유
ArithmeticException	0으로 나누기와 같은 부적절한 산술 연산을 수행할 때 발생한다.
IllegalArgumentException	메서드에 부적절한 인수를 전달할 때 발생한다.
IndexOutOfBoundsException	배열, 벡터 등에서 범위를 벗어난 인덱스를 사용할 때 발생한다.
NoSuchElementException	요구한 원소가 없을 때 발생한다.
NullPointerException	null 값을 가진 참조 변수에 접근할 때 발생한다.
NumberFormatException	숫자로 바꿀 수 없는 문자열을 숫자로 변환하려 할 때 발생한다.

```
5 public class UnChecked1Demo {  
6     public static void main(String[] args) {  
7         String s = "Time is money";  
8         StringTokenizer st = new StringTokenizer(s);  
9  
10        while (st.hasMoreTokens()) {  
11            System.out.print(st.nextToken() + "+");  
12        }  
13        System.out.print(st.nextToken());  
14    }  
15 }
```

예외

■ 일반 예외

- 컴파일러는 발생할 가능성을 발견하면 컴파일 오류를 발생
- 개발자는 예외 처리 코드를 반드시 추가
- 대표적인 일반 예외 예

일반 예외	발생 이유
ClassNotFoundException	존재하지 않는 클래스를 사용하려고 할 때 발생한다.
InterruptedException	인터럽트되었을 때 발생한다.
NoSuchFieldException	클래스가 명시한 필드를 포함하지 않을 때 발생한다.
NoSuchMethodException	클래스가 명시한 메서드를 포함하지 않을 때 발생한다.
IOException	데이터 읽기 같은 입출력 문제가 있을 때 발생한다.

```
public class CheckedDemo {  
    public static void main(String[] args) {  
        // Thread.sleep(100);  
    }  
}
```

예외 처리 방법

■ 두 가지 방법

- 예외 잡아 처리하기
 - 발생한 시점에서 우회 처리(오류) 작성
- 예외 떠넘기기
 - 다른 객체에 전달....
 - 호출한 위치로 돌려주기

예외 처리 방법

■ 예외 잡아 처리하기



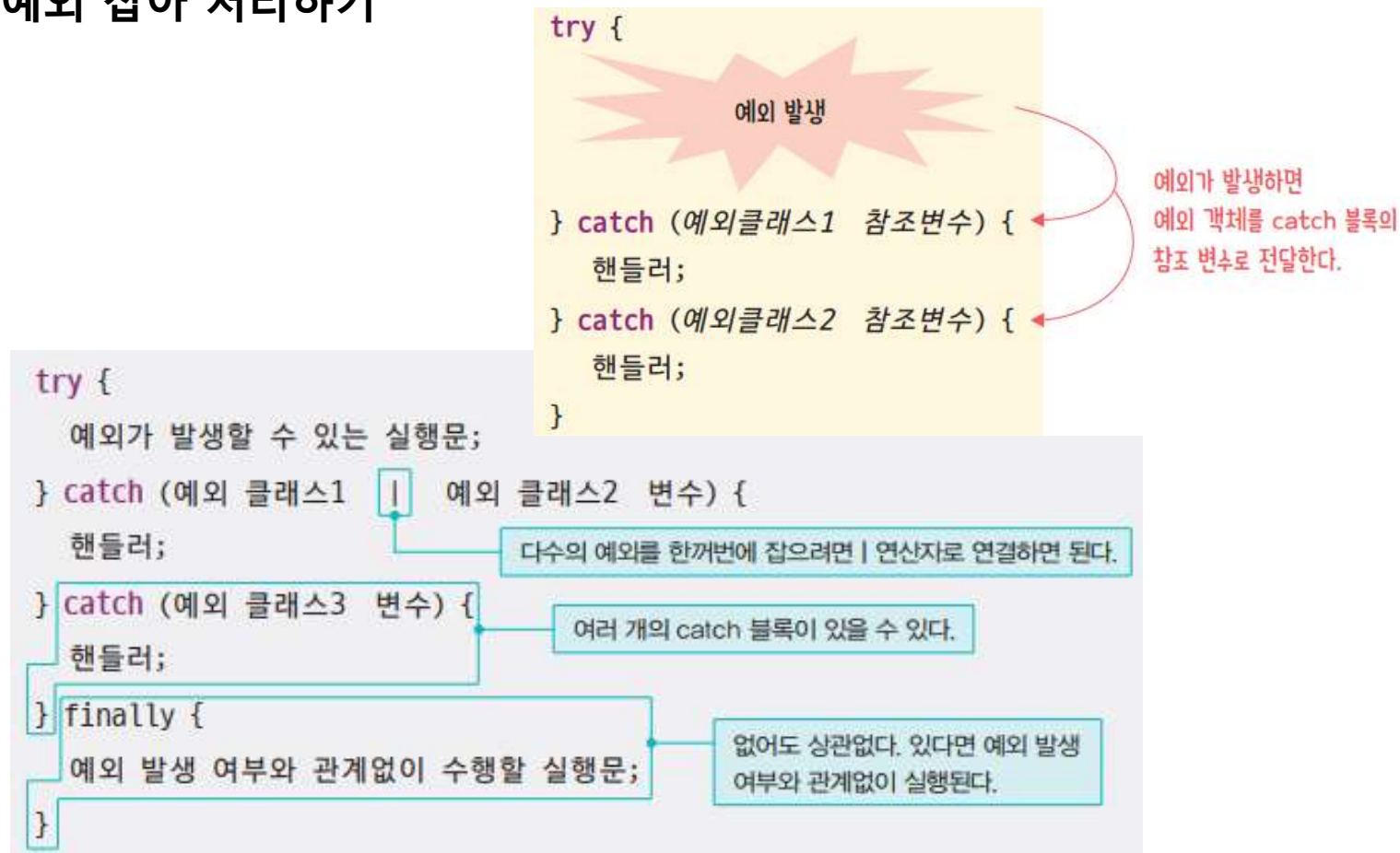
(a) 일반적인 코드



(b) try~catch 코드

예외 처리 방법

■ 예외 잡아 처리하기



- catch 블록의 순서도 중요

예외 처리 방법

■ 예외 잡아 처리하기

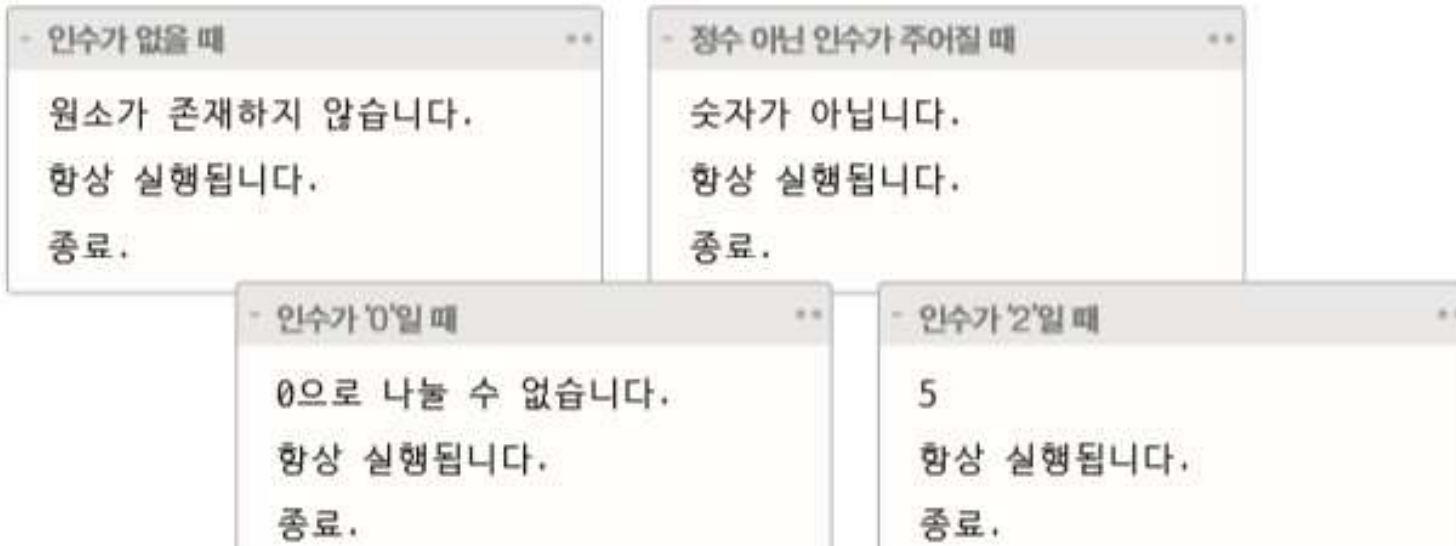
- Throwable 클래스의 주요 메서드

메서드	설명
public String getMessage()	Throwable 객체의 자세한 메시지를 반환한다.
public String toString()	Throwable 객체의 간단한 메시지를 반환한다.
public void printStackTrace()	Throwable 객체와 추적 정보를 콘솔 뷰에 출력한다.

```
3 public class TryCatch1Demo {
4     public static void main(String[] args) {
5         int[] array = { 0, 1, 2 };
6         try {
7             System.out.println("마지막 원소 => " + array[3]);
8             System.out.println("첫 번째 원소 => " + array[0]);
9         } catch (ArrayIndexOutOfBoundsException e) {
10             System.out.println("원소가 존재하지 않습니다.");
11         }
12         System.out.println("어이쿠!!!");
13     }
14 }
```

예외 처리 방법

■ 예외 잡아 처리하기



● 예제 : [sec](#)

```
3 public class TryCatch2Demo {
4     public static void main(String[] args) {
5         int dividend = 10;
6         try {
7             int divisor = Integer.parseInt(args[0]);
8             System.out.println(dividend / divisor);
9         } catch (ArrayIndexOutOfBoundsException e) {
10             System.out.println("원소가 존재하지 않습니다.");
11         } catch (NumberFormatException e) {
12             System.out.println("숫자가 아닙니다.");
13         } catch (ArithmeticException e) {
14             System.out.println("0으로 나눌 수 없습니다.");
15         } finally {
16             System.out.println("항상 실행됩니다.");
17         }
18         System.out.println("종료.");
19     }
20 }
```

예외 처리 방법

■ 예외 잡아 처리하기

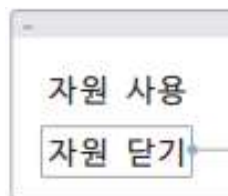
● try~with~resource 문

- try 블록에서 파일 등과 같은 리소스를 사용한다면 try 블록을 실행한 후 자원 반환 필요
- 리소스를 관리하는 코드를 추가하면 가독성도 떨어지고, 개발자도 번거롭다.
- JDK 7부터는 예외 발생 여부와 상관없이 사용한 리소스를 자동 반납하는 수단 제공. 단, 리소스는 AutoCloseable의 구현 객체

```
try (리소스) {  
} catch ( ... ) {  
}
```

- JDK 7과 8에서는 try()의 괄호 내부에서 자원 선언 필요. JDK 9부터는 try 블록 이전에 자원 선언 가능. 단, 선언된 자원 변수는 사실상 final이어야 함

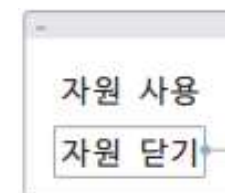
● 예제 : [sec02/TryCatch4Demo](#)



```

3 public class TryCatch4Demo {
4     public static void main(String[] args) {
5         Reso reso = new Reso();
6
7         try (reso) {
8             reso.show();
9         } catch (Exception e) {
10             System.out.println("예외 처리");
11         }
12     }
13 }
14
15 class Reso implements AutoCloseable {
16     void show() {
17         System.out.println("자원 사용");
18     }
19
20     public void close() throws Exception {
21         System.out.println("자원 닫기");
22     }
23 }
24

```



예외 처리 방법

■ 예외 떠넘기기

- 메서드에서 발생한 예외를 내부에서 처리하기가 부담스러울 때는 throws 키워드를 사용해 예외를 상위 코드 블록으로 양도 가능



예외 처리 방법

■ 예외 떠넘기기

● 사용 방법

```
public void write(String filename)
    throws IOException, ReflectiveOperationException {
    // 파일 쓰기과 관련된 실행문 ...
}
```

throws는 예외를 다른 메서드로 떠넘기는 키워드이다.

예외를 1개 이상 선언할 수 있다.

● 자바 API 문서

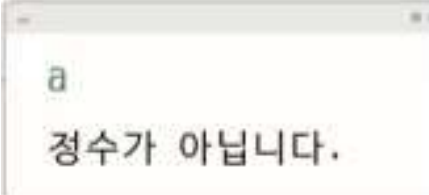
- 많은 메서드가 예외를 발생시키고 상위 코드로 예외 처리를 떠넘긴다.
- 예를 들면,

```
public static void sleep(long millis, int nanos) throws InterruptedException
```

```

5 public class ThrowsDemo {
6     public static void main(String[] args) {
7         Scanner in = new Scanner(System.in);
8         try {
9             square(in.nextLine());
10        } catch (NumberFormatException e) {
11            System.out.println("정수가 아닙니다.");
12        }
13    }
14
15    private static void square(String s)
16        throws NumberFormatException {
17        int n = Integer.parseInt(s);
18        System.out.println(n * n);
19    }
20 }

```



a

정수가 아닙니다.