Section 01
예외 처리

1. 예외 처리

예외와 예외 처리

- 예외

→프로그램의 정상적인 흐름을 방해하는 원치 않는 이벤트

→비정상적인 상태인 예외가 발생하면 프로그램의 실행이 종료되고 시스템 생성 오류 메시지가 나타냄

→[예] 존재하지 않는 파일 열기, 네트워크 연결 실패, 잘못된 데이터 입력 등

✓이러한 문제를 처리할 수 있도록 예외를 발생시킴

예외 발생 예시

```
public class Example01 {
   public static void main(String[] args) {
     int a = 0;
     int b = 5/a; // 0□□ □□□ □□ □□
   }
}
```

실행 결과

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
   at Example01.main(Example01.java:6)
```

예외의 이름과 설명

1. 예외 처리

■ 예외와 예외 처리

- 예외 처리

→애플리케이션의 정상적인 흐름을 유지할 수 있도록 런타임 오류를 처리하는 강력한 방법 중 하나

→ [예] 시스템에서 생성된 오류 메시지

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at Example01.main(Example01.java:6)
```

클래스명 메서드명 파일명 행 번호

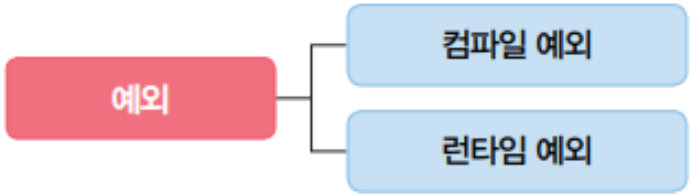
[그림 11-2] 시스템에서 생성된 오류 메시지

✓시스템에서 생성된 오류 메시지는 사용자가 무엇이 잘못되었는지 이해할 수 없으므로 간단한 메시지로 알리기 위해 예외를 처리함

Section 02
예외의 유형과 클래스

2. 예외의 유형과 클래스

■ 예외의 유형



[그림 11-3] 예외의 유형

- 컴파일 예외(코드에 예외처리 강제)

  →컴파일러가 컴파일 시간에 확인하는 검사형 예외(checked exception)

  →메서드 내부에 확인된 예외가 있는 경우 메서드는 예외를 처리하거나 throw 키워드를 사용하여 예외를 처리할 수 있음

- 런타임 예외

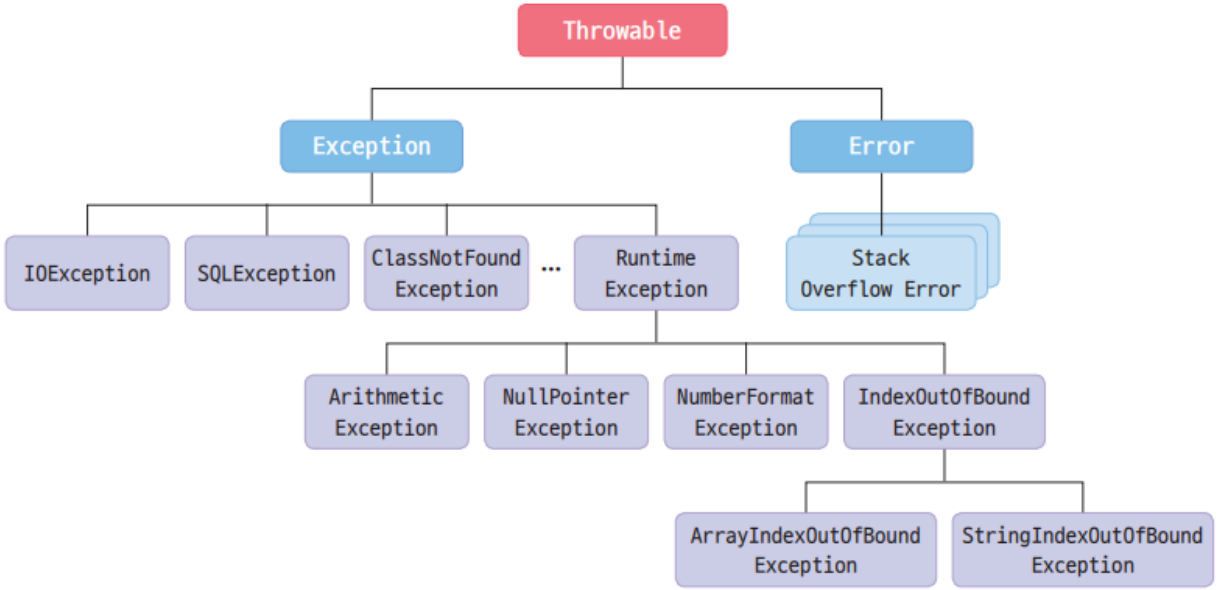
  →런타임에 발생하는 비검사형 예외(unchecked exception)

  →예외를 처리할 지 말지가 전적으로 프로그래머에게 달려 있음

2. 예외의 유형과 클래스

예외 클래스

- 예외의 모든 유형은 예외 클래스의 계층 구조 맨 위에 있는 Throwable 클래스의 하위 클래스
- 예외 클래스는 프로그램이 오류가 발생하기 전에 잡아내야 하는 예외적인 조건을 위한 클래스이므로 사용자별 예외 클래스를 생성하도록 확장됨



[그림 11-4] 예외 클래스의 계층 구조

2. 예외의 유형과 클래스

예외 클래스

[표 11-1] 주요 런타임 예외 클래스

클래스	설명
RuntimeException	실행 시간에 예외가 발생한다.
ArithmeticException	0으로 나누는 등의 산술적인 예외가 발생한다.
NullPointerException	실제 값이 아닌 null을 가지고 있는 객체/변수를 호출할 때 발생한다.
NumberFormatException	문자열을 숫자로 변환할 때 발생한다.
IndexOutOfBoundException	인덱스가 배열이나 스트링의 범위를 벗어날 때 발생한다.
ArrayIndexOutOfBoundException	인덱스가 배열의 크기보다 크거나 음수일 때 발생한다.
StringIndexOutOfBoundException	문자열 인덱스의 범위를 벗어날 때 발생한다.

[표 11-2] 주요 컴파일 예외 클래스

클래스	설명
IOException	입출력과 관련된 예외가 발생한다.
ClassNotFoundException	jar 또는 class 파일이 없을 때 발생한다.
SQLException	JDBC에서 쿼리를 실행하거나 생성할 때 발생한다.
InterruptedException	인터럽트되었을 때 발생한다.

2. 예외의 유형과 클래스

예외 클래스

[표 11-3] Exception 클래스의 주요 메서드

메서드	설명
getMessage()	발생한 예외에 대해 요약된 메시지를 반환한다.
getCause()	발생한 예외의 원인을 Throwable 객체 형태로 반환한다.
toString()	예외 메시지에 추가된 문자열의 이름이 포함된 문자열을 반환한다.
printStackTrace()	시스템 스택 추적을 오류 출력 스트림으로 출력한다.
getStackTrace()	스택 추적 요소가 포함된 배열을 반환한다.
fillinStackTrace()	Throwable 객체를 반환한다.

Section 03
예외 처리 방법

3. 예외 처리 방법

예외 처리를 위한 키워드

- 자바에서 예외를 처리하는 데 사용하는 키워드

→try, catch, finally, throw, throws

[표 11-4] 예외 처리를 위한 키워드

키워드	설명
try	예외 코드를 배치할 블록을 지정하기 위해 사용한다. try 블록을 단독으로 사용할 수 없고 catch 또는 finally가 뒤따라야 한다.
catch	catch 블록만 단독으로 사용할 수 없고 앞에 try 블록이 있어야 하며, finally 블록이 뒤따를 수도 있다.
finally	프로그램에서 꼭 필요한 코드를 실행하는 데 사용되며 예외 처리 여부에 관계없이 실행된다.
throw	예외를 고의로 발생시킨다.
throws	자신을 호출한 상위 메서드로 오류를 던지는 역할을 한다.

3. 예외 처리 방법

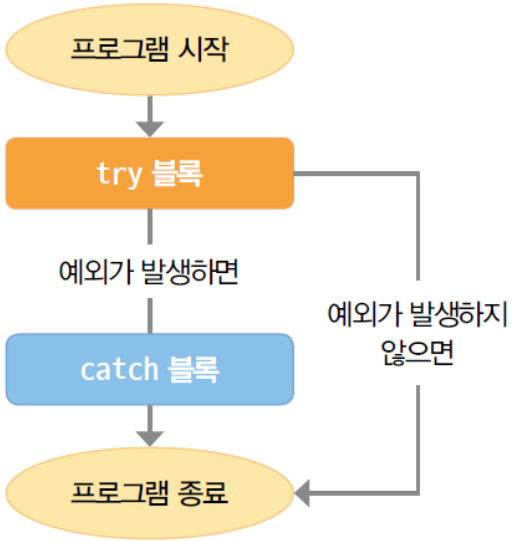
try~catch문

- 프로그램 실행 중에 예외를 일으킬 수 있는 코드를 try 블록으로 묶고 catch 블록으로 예외를 처리
- try 블록 다음에 catch 블록을 사용해야 하며, 하나의 try 블록에 여러 개의 catch 블록이 뒤따를 수도 있음

```
try {
   // try 블록의 명령문
}

catch(예외클래스 e) {
   // catch 블록의 명령문: 예외 처리
}
```

try 블록에서 예외 발생 시 실행



[그림 11-5] try~catch문의 구조

3. 예외 처리 방법

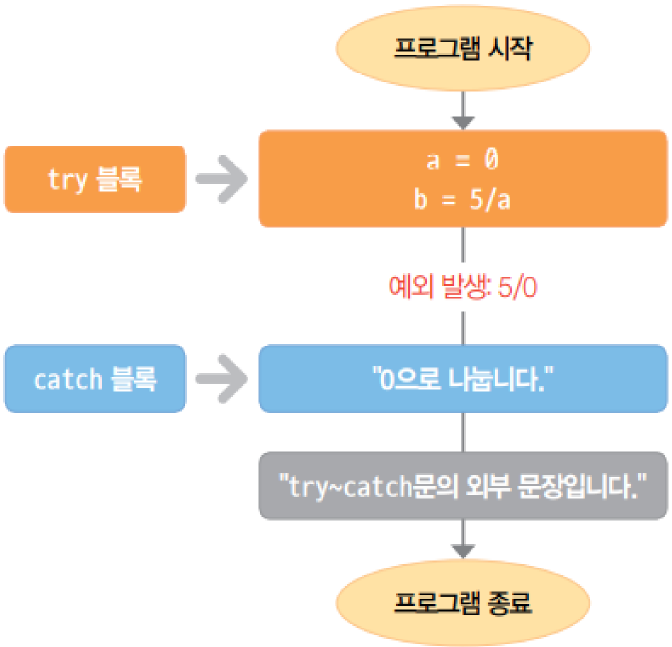
try~catch문

try~catch문 사용 예시

```
public class Example01 {
    public static void main(String[] args) {
        try {
            int a = 0;
             int b = 5/a;
        }
        catch(ArithmeticException e) {
             System.out.println("0으으 으으으 .");
        }
        System.out.println("try~catch으으 으으 으으으으 .");
    }
}
```

실행 결과

```
0으로 나눕니다.
try~catch문의 외부 문장입니다.
```



[그림 11-6] Example01 클래스 try~catch문의 구조

3. 예외 처리 방법

예제 11-1 try~catch문을 사용하여 예외 처리하기

```
01 import java.util.Scanner;
02 public class Exception01 {
03     public static void main(String[] args) {
04
05         Scanner s = new Scanner(System.in);
06         System.out.println("숫자를 입력하세요.");
07         int num1 = s.nextInt();
08
09         // try 블록
10         try {
11             int num2 = 10 / num1;
12             System.out.println(num2);
13         } // catch 블록
```

3. 예외 처리 방법

```
14            catch(Exception e) {
15                System.out.println("올바른 숫자를 입력하세요.");
16                 System.out.println(e.getMessage());
17            e.printStackTrace();
18        }
19         System.out.println("try~catch 블록의 외부 문장입니다.");
20      }
21 }
```

실행 결과

```
숫자를 입력하세요.
0
올바른 숫자를 입력하세요.
java.lang.ArithmeticException: / by zero
try~catch 블록의 외부 문장입니다.
    at com.section01.Example01.main(Example01.java:14)
```

3. 예외 처리 방법

다중 catch문

- 하나의 try 블록 다음에 여러 개의 catch 블록이 오는 것

  →try 블록에서 예외가 발생하면 첫 번째 catch 블록으로 예외가 전달됨

  →예외의 유형이 첫 번째 catch 블록과 일치하면 예외를 처리함

  →일치하지 않으면 다음 catch 블록으로 전달

- 예외의 유형이 확실하지 않을 때 유용

3. 예외 처리 방법

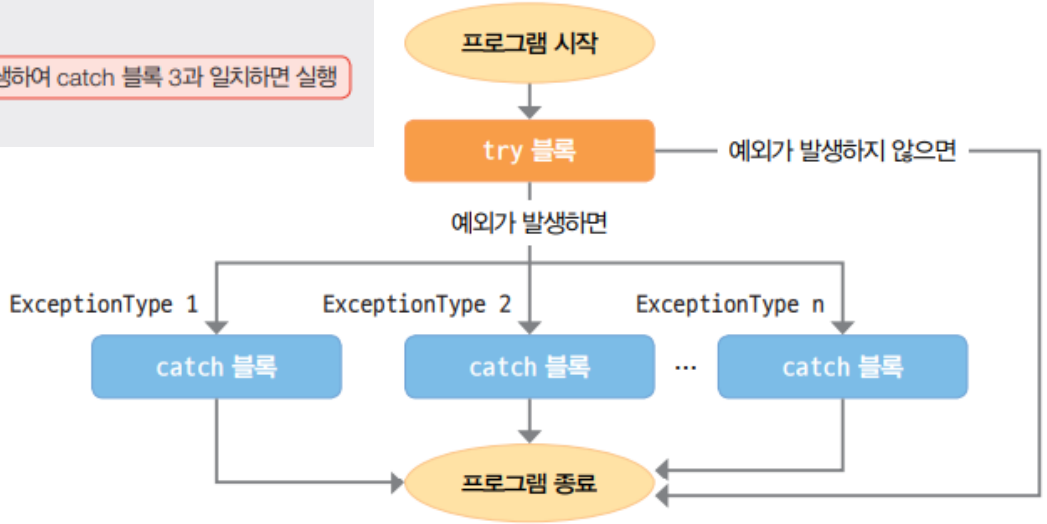
다중 catch문

```
try {
   // try 블록의 명령문
}
catch(예외클래스1 e) {
   // catch 블록 1의 명령문
}
catch(예외클래스2 e) {
   // catch 블록 2의 명령문
}
catch(예외클래스3 e) {
   // catch 블록 3의 명령문
}
```

try 블록에서 예외가 발생하여 catch 블록 1과 일치하면 실행

try 블록에서 예외가 발생하여 catch 블록 2와 일치하면 실행

try 블록에서 예외가 발생하여 catch 블록 3과 일치하면 실행



[그림 11-7] 다중 catch문의 구조

3. 예외 처리 방법

다중 catch문

다중 catch문 사용 예시

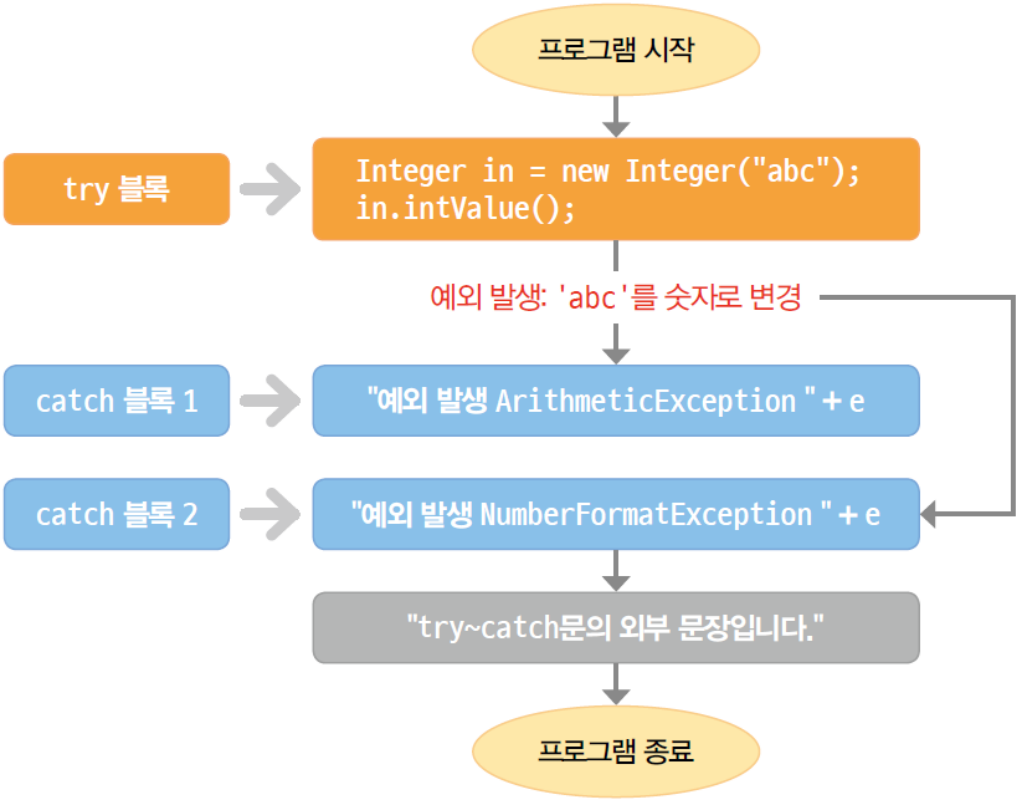
```
public class Example02 {
    public static void main(String[] args) {
        try {
            Integer in = Integer.parseInt("abc");
             in.intValue();
        }
        catch(ArithmeticException e) {
            System.out.println("예외 발생 ArithmeticException " + e);
        }
        catch(NumberFormatException e) {
            System.out.println("예외 발생   NumberFormatException " + e);
        }
        System.out.println("try~catch문의 외부 문장입니다.");
    }
}
```

실행 결과

```
예외 발생 NumberFormatException java.lang.NumberFormatException:
For input string: "abc"
try~catch문의 외부 문장입니다.
```

3. 예외 처리 방법

■ 다중 catch문



[그림 11-8] Example02 클래스의 다중 catch문 구조

6. java.text 패키지

예제 11-2 다중 catch문을 사용하여 예외 처리하기

```
01 import java.util.Scanner;
02 public class Exception02 {
03     public static void main(String[] args) {
04
05         Scanner s = new Scanner(System.in);
06         System.out.println("숫자를 입력하세요.");
07         int num = s.nextInt();
08
09         int arr[] = new int[5];
10
11         try {
12             arr[num] = 10 / num;
13             System.out.println(arr[num]);
14         }
```

3. 예외 처리 방법

```
15  catch(ArithmeticException e) {
16          System.out.println("0이 아닌 값을 입력하세요.");
17          System.out.println(e.getMessage());
18      }
19      catch(ArrayIndexOutOfBoundsException e) {
20          System.out.println("올바른 배열 인덱스를 입력하세요.");
21          System.out.println(e.getMessage());
22       }
23    }
24 }
```

실행 결과

```
숫자를 입력하세요.
0
0이 아닌 값을 입력하세요.
/ by zero
```

```
숫자를 입력하세요.
10
올바른 배열 인덱스를 입력하세요.
Index 10 out of bounds for length 5
```

3. 예외 처리 방법

try~catch~finally문

- finally 블록은 예외 발생 여부에 관계없이 실행됨

  →따라서 예외가 발생하든 발생하지 않든 실행해야 하는 명령문을 이 블록에 배치

- finally 블록은 try~catch문이 실행된 후 실행됨

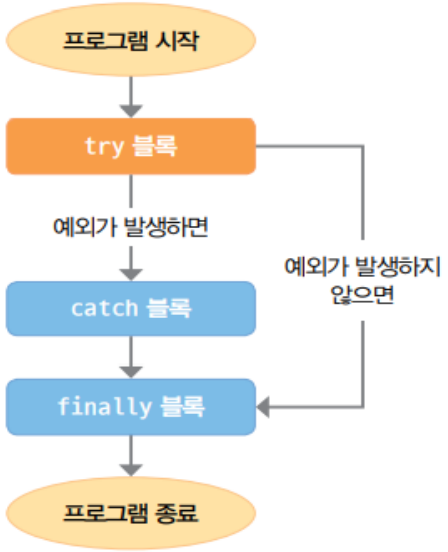
  →try 블록에서 예외가 발생하지 않으면 try 블록 실행 직후에 finally 블록이 실행됨

  →try 블록에서 예외가 발생하면 각 catch 블록이 실행된 후 finally 블록이 실행됨

```
try {
   // try 블록의 명령문
}

catch(Exception e) {
   // catch 블록의 명령문
}

finally {
   // finally 블록의 명령문
}
```



[그림 11-9] try~catch~finally문의 구조

3. 예외 처리 방법

try~catch~finally문

try~catch~finally문 사용 예시

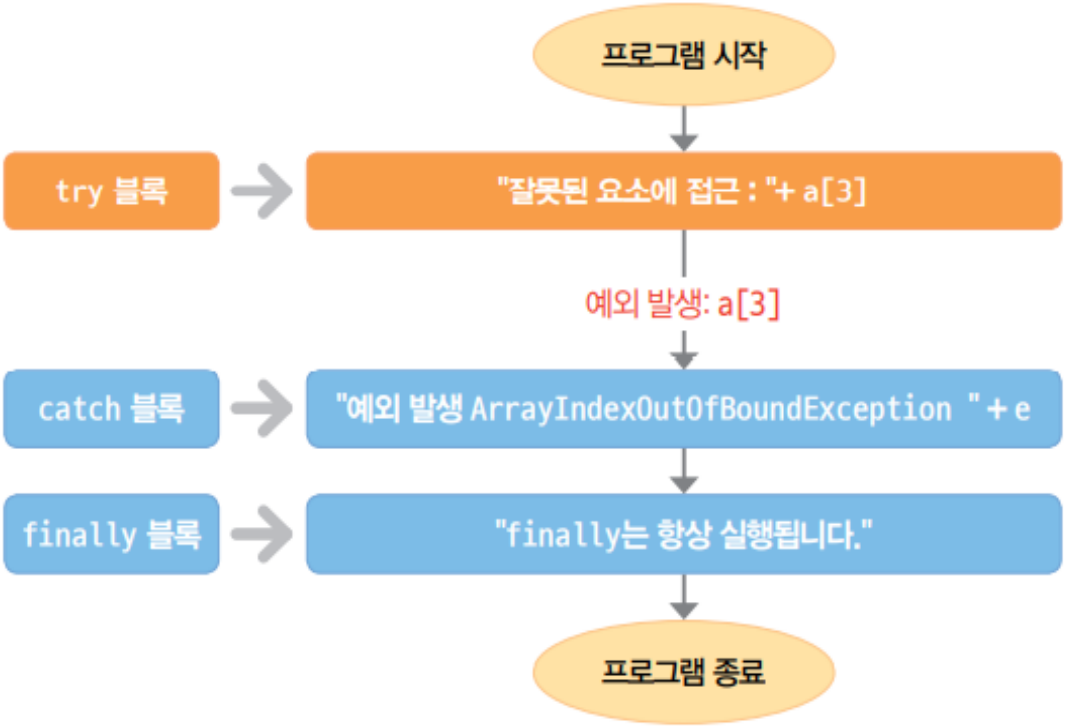
```
public class Example03 {
    public static void main(String[] args) {
        int a[] = new int[2];
        try {
            System.out.println("잘못된 요소에 접근 : "+ a[3] );
        }
        catch(Exception e) {
             System.out.println("예외 발생 ArrayIndexOutOfBoundException " + e);
        }
        finally {
            System.out.println("finally는 항상 실행됩니다.");
        }
    }
}
```

실행 결과

```
예외 발생 ArrayIndexOutOfBoundException java.lang.ArrayIndexOutOfBoundsException: Index 3
out of bounds for length 2
finally는 항상 실행됩니다.
```

3. 예외 처리 방법

try~catch~finally문



[그림 11-10] Example03 클래스 try~catch~finally문의 구조

3. 예외 처리 방법

예제 11-3 try~catch~finally문을 사용하여 예외 처리하기

```
01 public class Exception03 {
02     public static void main(String[] args) {
03
04      int[] arr = {10, 20, 30};
05
06      try {
07          for(int i = 0; i <= 3; i++) {
08              System.out.println("arr[" + i + "] : " + arr[i] );
09          }
10      }
11      catch(Exception ex) {
12           System.out.println("예외 처리입니다....");
13           System.out.println(ex.getMessage());
14     }
15     finally {
16          System.out.println("예외 발생 여부와 상관없이 실행됩니다.");
17     }
18   }
19 }
```

실행 결과

```
arr[0] : 10
arr[1] : 20
arr[2] : 30
예외 처리입니다....
Index 3 out of bounds for length 3
예외 발생 여부와 상관없이 실행됩니다.
```

3. 예외 처리 방법

throws 키워드

- 프로그램 실행 중 메서드가 예외를 발생시킬 수 있도록 선언하려면 throw 키워드를 사용함
- 예외가 발생했을 때 발생한 메서드에서 직접 처리하지 않고 자신을 호출한 곳으로 떠넘기려는 경우에는 반드시 throws를 선언해야 함

```
메서드유형 메서드명(매개변수목록) throws 예외처리목록 {
   // 구현 코드
}
```

3. 예외 처리 방법

throws 키워드

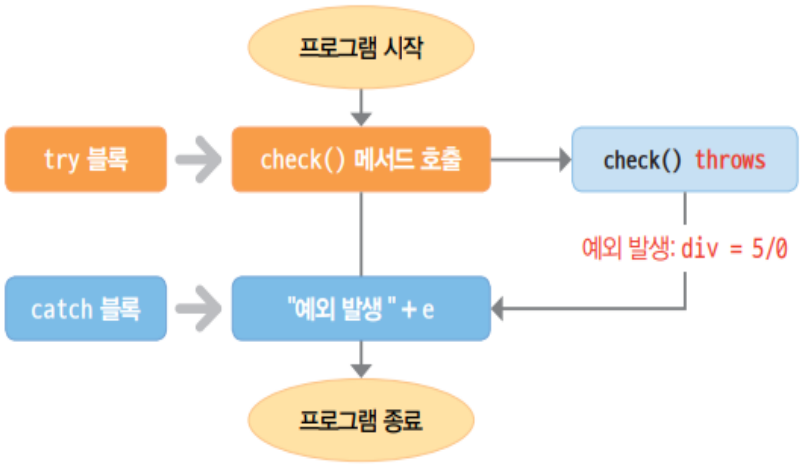
throws 사용 예시

```
public class Example04 {
   static void check() throws ArithmeticException {
       System.out.println("내부 메서드");
        int div = 5/0;
   }

   public static void main(String[] args) {
        try {
            check();
        }
        catch(ArithmeticException e) {
            System.out.println("예외 발생 " + e);
        }
   }
}
```

실행 결과

```
내부 메서드
예외 발생 java.lang.ArithmeticException: / by zero
```



[그림 11-11] Example04 클래스 throws 키워드의 처리 구조

3. 예외 처리 방법

예제 11-4 throws, throw 키워드를 사용하여 예외 처리하기

```
01 import java.util.Scanner;
02
03 public class Exception04 {
04
05 static void check(int num) throws NumberFormatException {
06     if (num < 0)
07         throw new NumberFormatException("0보다 작습니다.");
08     else
09       System.out.println(num);
10 }
11
12     public static void main(String[] args) {
13         Scanner s = new Scanner(System.in);
14         System.out.println("숫자를 입력하세요.");
15        int num = s.nextInt();
```

3. 예외 처리 방법

```
16
17      try {
18         if (num < 0)
19              throw new NumberFormatException("0보다 작습니다.");
20        else
21             System.out.println(num);
22
23        check(num);
24      } catch(NumberFormatException e) {
25          System.out.println("예외 발생 " + e);
26      }
27    }
28 }
```

실행 결과

```
숫자를 입력하세요.
-1
예외 발생 java.lang.NumberFormatException: 0보다 작습니다.
```

3. 예외 처리 방법

사용자 정의 예외

- 사용자가 새로운 예외 클래스를 만들어서 이용할 수 있는 방법
- java.lang 패키지의 Exception 예외 클래스를 상속받아 작성
- 사용자 정의 예외 클래스에 대한 생성자를 정의하고(필수는 아님)
- toString() 메서드를 재정의하여 catch 블록에서 사용자 정의 메시지를 표시할 수 있음
- 사용자 정의 예외 클래스를 만들고 throw 키워드로 예외를 발생시켜 호출함

3. 예외 처리 방법

사용자 정의 예외

사용자 정의 예외 처리 예시

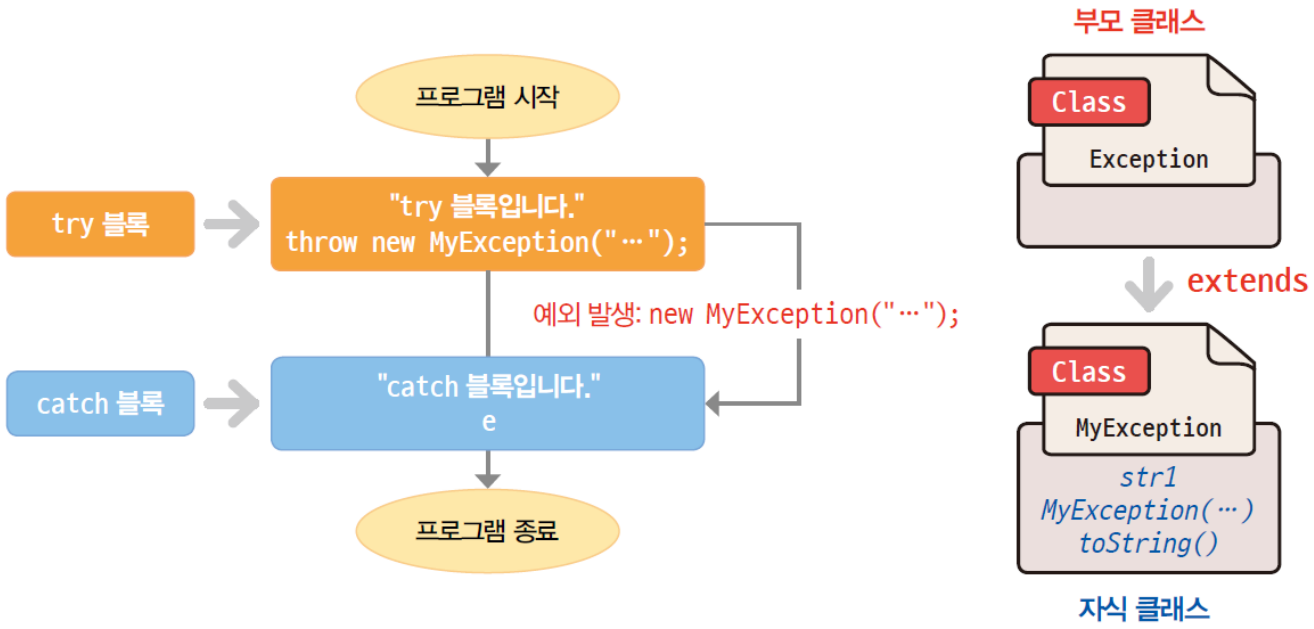
```
class MyException extends Exception {
    String str1;
    MyException(String str2) {
       str1 = str2;
   }
    public String toString() {
        return ("MyException 발생: "+ str1);
    }
}
public class Example06 {
   public static void main(String[] args) {
        try {
            System.out.println("try 블록입니다.");
            throw new MyException("MyException 클래스 호출됩니다.");
        }
        catch(MyException e) {
             System.out.println("catch 블록입니다.");
              System.out.println(e);
        }
   }
}
```

실행 결과

```
try 블록입니다.
catch 블록입니다.
MyException 발생: MyException 클래스 호출됩니다.
```

3. 예외 처리 방법

사용자 정의 예외



[그림 11-13] Example06 클래스의 사용자 정의 예외 처리 구조

3. 예외 처리 방법

예제 11-5 사용자 정의 예외 클래스 만들기

```
01 class InvalidException extends Exception {
02    public InvalidException(String s) {
03       super(s);
04    }
05 }
06 public class Exception05 {
07     void check(int weight) throws InvalidException {
08        if (weight < 100) {
09           throw new InvalidException("InvalidException 클래스 호출입니다.");
10       }
11    }
12
```

3. 예외 처리 방법

```
13 public static void main(String[] args) {
14      Exception05 obj = new Exception05();
15       try {
16            obj.check(60);
17      }
18      catch(InvalidException ex) {
19            System.out.println("예외 처리입니다. ");
20           System.out.println(ex.getMessage());
21     }
22   }
23 }
```

실행 결과

```
예외 처리입니다.
InvalidException 클래스 호출입니다.
```