

예외처리 (Exception)

▶ 프로그램 오류

프로그램 수행 시 치명적 상황이 발생하여 비정상 종료 상황이 발생한 것, 프로그램 에러라고도 함

✓ 오류의 종류

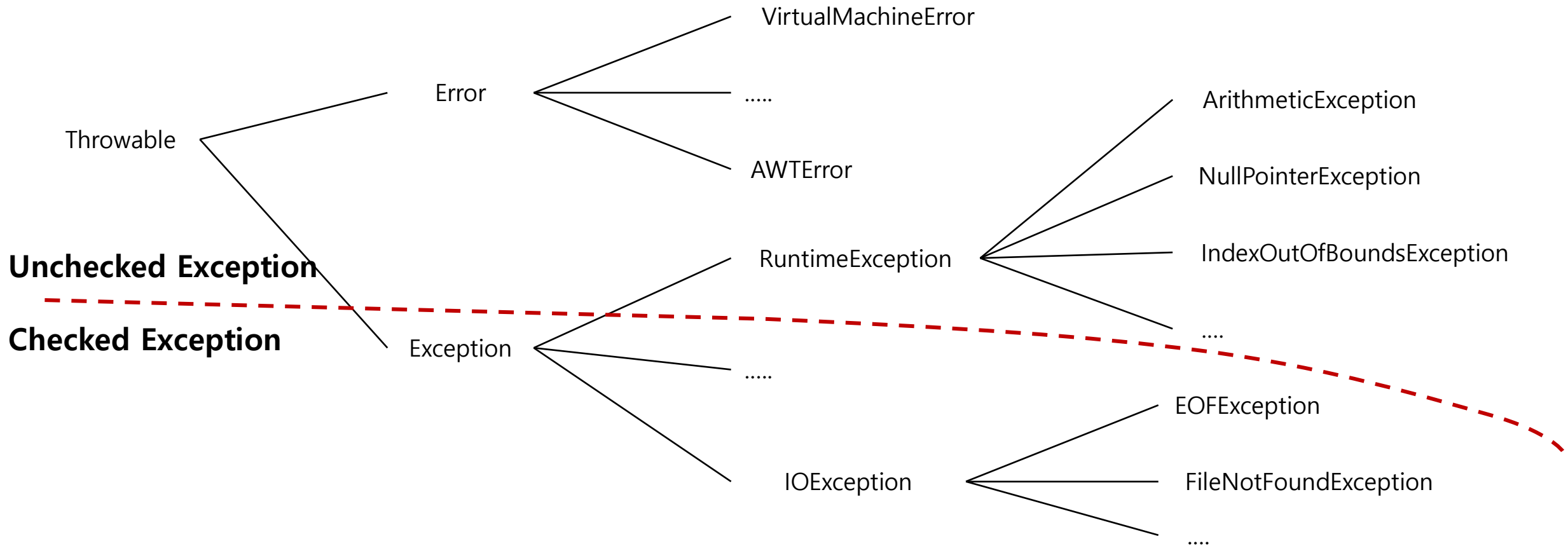
1. 컴파일 에러 : 프로그램의 실행을 막는 소스 상의 문법 에러, 소스 구문을 수정하여 해결
2. 런타임 에러 : 입력 값이 틀렸거나, 배열의 인덱스 범위를 벗어났거나, 계산식의 오류 등
주로 if문 사용으로 에러 처리
3. 시스템 에러 : 컴퓨터 오작동으로 인한 에러, 소스 구문으로 해결 불가

✓ 오류 해결 방법

소스 수정으로 해결 가능한 에러를 예외(Exception)라고 하는데
이러한 예외 상황(예측 가능한 에러) 구문을 처리 하는 방법인 예외처리를 통해 해결

▶ 예외 클래스 계층 구조

Exception과 Error 클래스 모두 Object 클래스의 자손이며 모든 예외의 최고 조상은 Exception 클래스
반드시 예외 처리해야 하는 Checked Exception과 해주지 않아도 되는 Unchecked Exception으로 나뉨



▶ 예외처리(Exception)

✓ RuntimeException 클래스

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

✓ RuntimeException 후손 클래스

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

▶ 예외처리(Exception)

✓ Exception 확인하기

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

✓ 예시

java.io.BufferedReader의 readLine() 메소드

readLine

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

▶ 예외처리 방법

1. Exception이 발생한 곳에서 직접 처리

try~catch문을 이용하여 예외처리

try : exception 발생할 가능성이 있는 코드를 안에 기술

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

중간에 return문을 만나도 finally구문은 실행되지만

System.exit();를 만나면 무조건 프로그램 종료

주로 java.io나 java.sql 패키지의 메소드 처리 시 이용

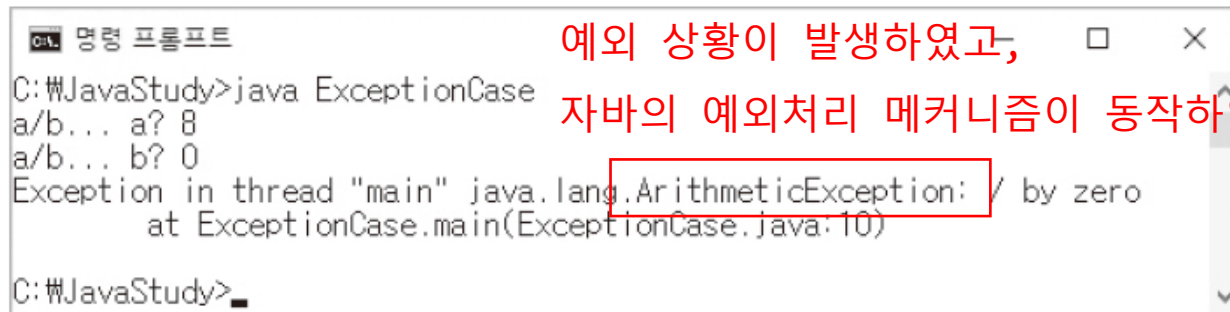
2. Exception 처리를 호출한 메소드에게 위임

메소드 선언 시 throws ExceptionName문을 추가하여 호출한 상위 메소드에게 처리 위임

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

예외 상황의 예

```
public static void main(String[] args) {  
    Scanner kb = new Scanner(System.in);  
  
    System.out.print("a/b...a? ");  
    int n1 = kb.nextInt();    // int형 정수 입력  
  
    System.out.print("a/b...b? ");  
    int n2 = kb.nextInt();    // int형 정수 입력  
  
    System.out.printf("%d / %d = %d \n", n1, n2, n1 / n2);  
    System.out.println("Good bye~~!");  
}
```



The screenshot shows a Windows Command Prompt window titled "명령 프롬프트". The command prompt shows the execution of a Java program. The user enters '8' for 'a' and '0' for 'b'. The program attempts to calculate '8 / 0', which results in an 'ArithmeticException: / by zero'. The exception message is displayed in the command prompt, with 'ArithmeticException:' highlighted by a red box. The command prompt also shows the file path 'C:\JavaStudy' and the command 'java ExceptionCase'.

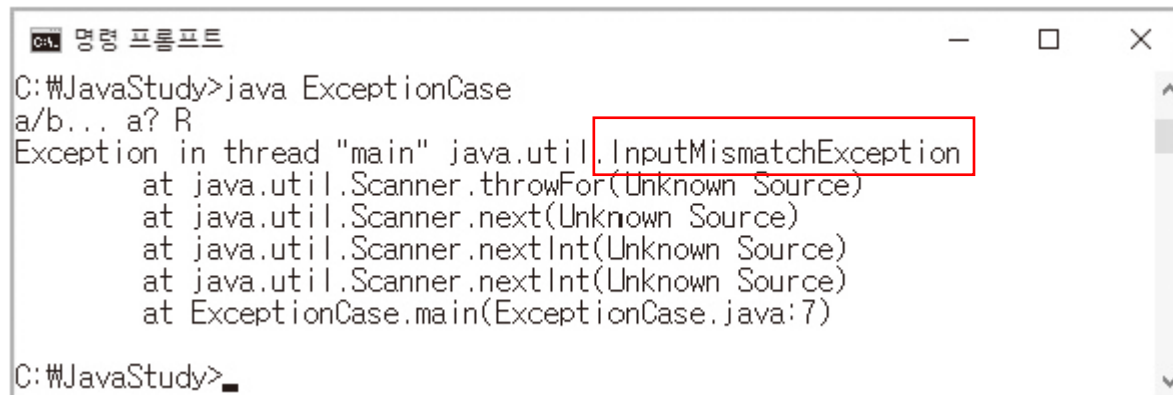
예외 상황이 발생하였고,
자바의 예외처리 메커니즘이 동작하였다.

예외 상황의 또 다른 예

```
public static void main(String[] args) {  
    Scanner kb = new Scanner(System.in);  
  
    System.out.print("a/b...a? ");  
    int n1 = kb.nextInt();    // int형 정수 입력  
  
    System.out.print("a/b...b? ");  
    int n2 = kb.nextInt();    // int형 정수 입력  
  
    System.out.printf("%d / %d = %d \n", n1, n2, n1 / n2);  
    System.out.println("Good bye~~!");  
}
```

예외에 대한 처리 방법은 프로그래머가 결정할 수 있다.

자바의 기본 예외처리 메커니즘은
문제가 발생한 지점에 대한 정보 출력과
프로그램 종료이다!



```
명령 프롬프트
C:\JavaStudy>java ExceptionCase
a/b... a? R
Exception in thread "main" java.util.InputMismatchException
    at java.util.Scanner.throwFor(Unknown Source)
    at java.util.Scanner.next(Unknown Source)
    at java.util.Scanner.nextInt(Unknown Source)
    at java.util.Scanner.nextInt(Unknown Source)
    at ExceptionCase.main(ExceptionCase.java:7)
C:\JavaStudy>
```


예외 상황을 알리기 위한 클래스

`java.lang.ArithmeticException`

→ 수학 연산에서의 오류 상황을 의미하는 예외 클래스

`java.util.InputMismatchException`

→ 클래스 Scanner를 통한 값의 입력에서의 오류 상황을 의미하는 예외 클래스

예외의 처리를 위한 try ~ catch

```
try {  
    ...관찰 영역...  
}  
  
catch(ArithmeticException e) {  
    ...처리 영역...  
}
```

예외의 처리를 위한 코드를 별도로 구분하기 위해 디자인된 예외처리 메커니즘이 try ~ catch 이다.

try ~ catch의 예

```
public static void main(String[] args) {  
    Scanner kb = new Scanner(System.in);  
  
    try {  
        System.out.print("a/b...a? ");  
        int n1 = kb.nextInt();  
        System.out.print("a/b...b? ");  
        int n2 = kb.nextInt();  
        System.out.printf("%d / %d = %d \n", n1, n2, n1 / n2);    // 예외 발생 지점  
    }  
    catch(ArithmeticException e) {  
        System.out.println(e.getMessage());  
    }  
  
    System.out.println("Good bye~~!");  
}
```

C:\ 명령 프롬프트

```
C:\JavaStudy>java ExceptionCase2  
a/b... a? 2  
a/b... b? 0  
/ by zero  
Good bye~~!  
C:\JavaStudy>_
```

예외 발생 이후의 실행 흐름

```
try {  
    1. ...  
    2. 예외 발생 지점  
    3. ...  
}  
catch(Exception e) {  
    ...  
}
```

4. 예외 처리 이후 실행 지점

try로 감싸야 할 영역의 결정

```
public static void main(String[] args) {  
    Scanner kb = new Scanner(System.in);
```

```
    try {
```

입력 오류에 대한 예외의 관점에서 보았을 때 이는 하나의 작업

```
        System.out.print("a/b...a? ");  
        int n1 = kb.nextInt();  
        System.out.print("a/b...b? ");  
        int n2 = kb.nextInt();  
        System.out.printf("%d / %d = %d \n", n1, n2, n1/n2);
```

```
    }
```

```
    catch(InputMismatchException e) {  
        e.getMessage();  
    }
```

```
    System.out.println("Good bye~~!");  
}
```

둘 이상의 예외 처리를 위한 구성1

```
public static void main(String[] args) {
    Scanner kb = new Scanner(System.in);

    try {
        System.out.print("a/b...a? ");
        int n1 = kb.nextInt();
        System.out.print("a/b...b? ");
        int n2 = kb.nextInt();
        System.out.printf("%d / %d = %d \n", n1, n2, n1 / n2);
    }
    catch(ArithmeticException e) {
        e.getMessage();
    }
    catch(InputMismatchException e) {
        e.getMessage();
    }

    System.out.println("Good bye~~!");
}
```

둘 이상의 예외 처리를 위한 구성2

```
public static void main(String[] args) {  
    Scanner kb = new Scanner(System.in);  
  
    try {  
        System.out.print("a/b...a? ");  
        int n1 = kb.nextInt();  
        System.out.print("a/b...b? ");  
        int n2 = kb.nextInt();  
        System.out.printf("%d / %d = %d \n", n1, n2, n1 / n2);  
    }  
    catch(ArithmeticException | InputMismatchException e) {  
        e.getMessage();  
    }  
  
    System.out.println("Good bye~~!");  
}
```

Throwable 클래스

`java.lang.Throwable` 클래스

모든 예외 클래스의 최상위 클래스: 물론 `Throwable`도 `Object`를 상속한다.

`Throwable` 클래스의 메소드 둘

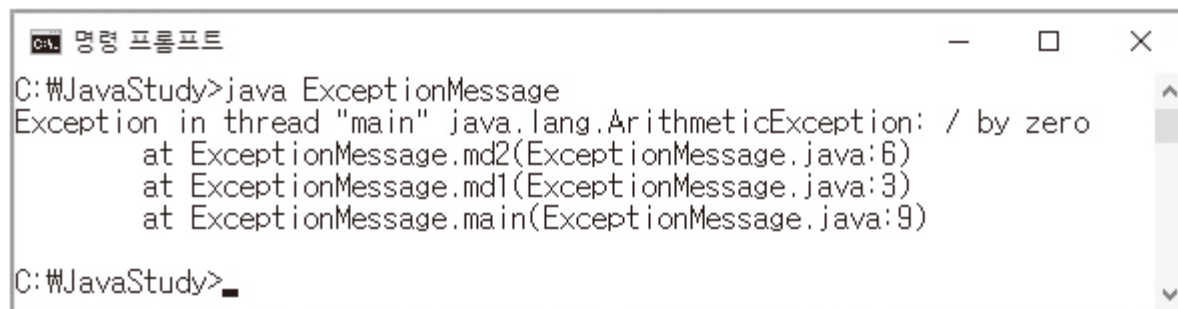
`public String getMessage()` : 예외의 원인을 담고 있는 문자열을 반환

`public void printStackTrace()` : 예외가 발생한 위치와 호출된 메소드의 정보를 출력

예외의 전달

```
class ExceptionMessage {  
    public static void md1(int n) {  
        md2(n, 0);    // 아래의 메소드 호출  
    }  
  
    public static void md2(int n1, int n2) {  
        int r = n1 / n2;    // 예외 발생 지점  
    }  
  
    public static void main(String[] args) {  
        md1(3);  
        System.out.println("Good bye~~!");  
    }  
}
```

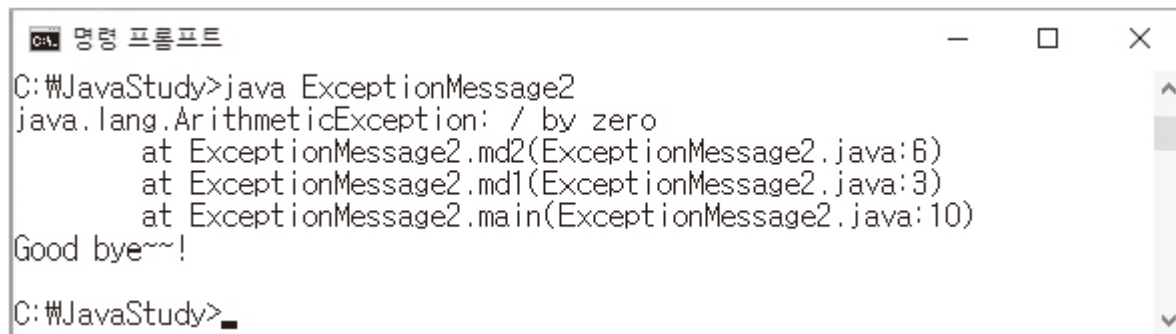
예외 발생 지점에서 예외를 처리하지 않으면 해당 메소드를 호출한 영역으로 예외가 전달된다.



```
C:\JavaStudy>java ExceptionMessage  
Exception in thread "main" java.lang.ArithmeticException: / by zero  
    at ExceptionMessage.md2(ExceptionMessage.java:6)  
    at ExceptionMessage.md1(ExceptionMessage.java:3)  
    at ExceptionMessage.main(ExceptionMessage.java:9)  
  
C:\JavaStudy>
```

printStackTrace 메소드 관련 예제

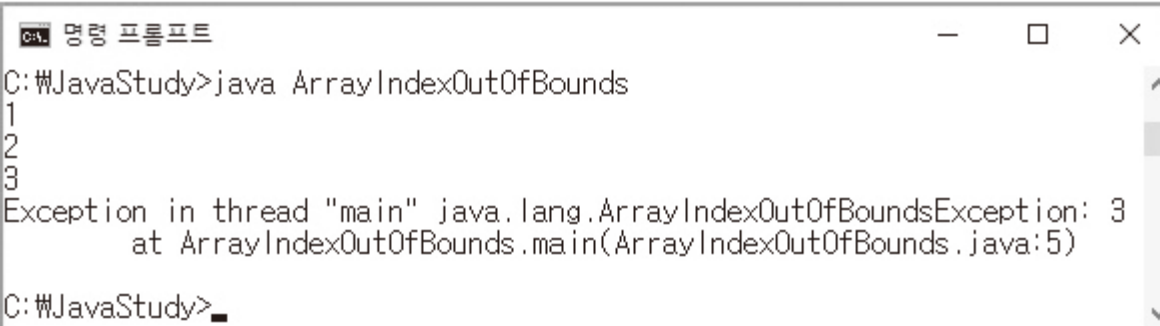
```
class ExceptionMessage2 {  
    public static void md1(int n) {  
        md2(n, 0);    // 이 지점으로 md2로부터 예외가 넘어온다.  
    }  
    public static void md2(int n1, int n2) {  
        int r = n1 / n2;    // 예외 발생 지점  
    }  
    public static void main(String[] args) {  
        try {  
            md1(3);    // 이 지점에서 md1으로부터 예외가 넘어온다.  
        }  
        catch(Throwable e) {  
            e.printStackTrace();  
        }  
  
        System.out.println("Good bye~~!");  
    }  
}
```



```
CA 명령 프롬프트
C:\WJavaStudy>java ExceptionMessage2
java.lang.ArithmeticException: / by zero
    at ExceptionMessage2.md2(ExceptionMessage2.java:6)
    at ExceptionMessage2.md1(ExceptionMessage2.java:3)
    at ExceptionMessage2.main(ExceptionMessage2.java:10)
Good bye~~!
C:\WJavaStudy>
```

ArrayIndexOutOfBoundsException

```
class ArrayIndexOutOfBounds {  
    public static void main(String[] args) {  
        int[] arr = {1, 2, 3};  
  
        for(int i = 0; i < 4; i++)  
            System.out.println(arr[i]); // 인덱스 값 3은 예외를 발생시킴  
    }  
}
```



The screenshot shows a Windows Command Prompt window titled "명령 프롬프트". The command prompt shows the execution of the command `C:\#JavaStudy>java ArrayIndexOutOfBounds`. The output displays the numbers 1, 2, and 3 on separate lines, followed by an exception message: `Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 3` and `at ArrayIndexOutOfBounds.main(ArrayIndexOutOfBounds.java:5)`. The prompt then returns to `C:\#JavaStudy>`.

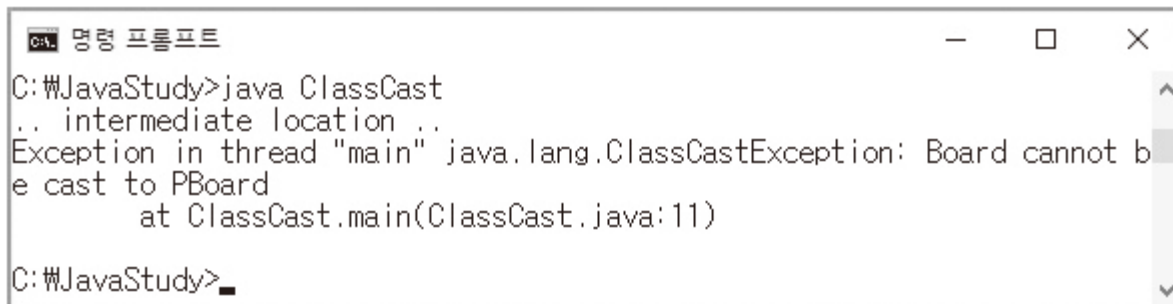
ClassCastException

```
class Board { }

class PBoard extends Board { }

class ClassCast {
    public static void main(String[] args) {
        Board pbd1 = new PBoard();
        PBoard pbd2 = (PBoard)pbd1;    // OK!

        System.out.println(".. intermediate location .. ");
        Board ebd1 = new Board();
        PBoard ebd2 = (PBoard)ebd1;    // Exception!
    }
}
```



A screenshot of a Windows Command Prompt window titled "명령 프롬프트". The window shows the execution of a Java program. The command entered is "C:\#JavaStudy>java ClassCast". The output shows the program running successfully for the first part, printing ".. intermediate location .. ", and then throwing a "java.lang.ClassCastException: Board cannot be cast to PBoard" at line 11 of ClassCast.java. The prompt "C:\#JavaStudy>" is visible at the bottom.

```
C:\#JavaStudy>java ClassCast
.. intermediate location ..
Exception in thread "main" java.lang.ClassCastException: Board cannot be
cast to PBoard
    at ClassCast.main(ClassCast.java:11)
C:\#JavaStudy>
```

NullPointerException

```
class NullPointer {  
    public static void main(String[] args) {  
        String str = null;  
        System.out.println(str);    // null 출력  
        int len = str.length();    // Exception!  
    }  
}
```



The screenshot shows a Windows Command Prompt window titled "명령 프롬프트". The command prompt shows the execution of the command `C:\JavaStudy>java NullPointer`. The output is `null` followed by an exception message: `Exception in thread "main" java.lang.NullPointerException
at NullPointer.main(NullPointer.java:5)`. The prompt then shows `C:\JavaStudy>` with a cursor.

▶ 예외처리 방법

✓ 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("프로그램 종료");  
    }  
}
```

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

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

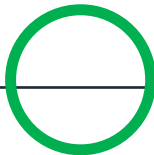
```
public void methodC() throws IOException {  
    throw new IOException();  
    //Exception 발생  
}
```

▶ Exception과 오버라이딩


오버라이딩 시 throws하는 Exception의 개수와 상관없이 같거나 후손범위여야 된다.

```
public class Parent {  
    public void method() throws IOException{  
        . . .  
    }  
}
```

```
public class Child1 extends Parent{  
    @Override  
    public void method() throws EOFException {  
        . . .  
    }  
}
```



```
public class Child2 extends Parent{  
    @Override  
    public void method() 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("사용자정의 예외발생");  
    }  
}
```

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