삼육대학교
SAHMYOOK UNIVERSITY

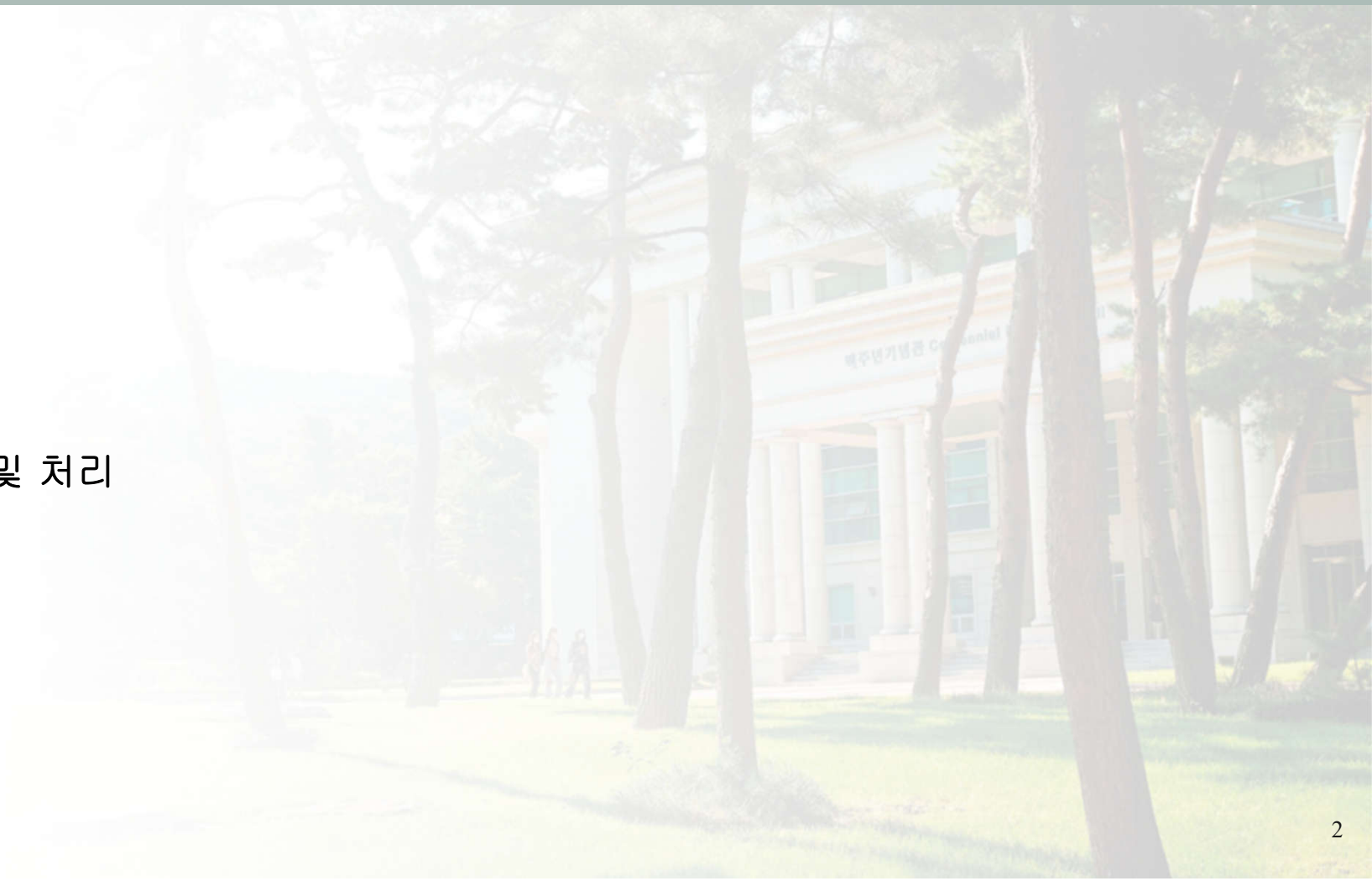
# 예외 처리

## 예외 처리

김 병 국 교수

# Contents





# 1. 예외(Exception) [1/4]

❖ 예외

- ✓ **비정상적**인 이벤트(events, 사건)
- ✓ 프로그램의 정상적인 명령 흐름을 **방해하는 이벤트**
- ✓ 명령처리 중 오버플로우(overflow), 언더플로우(underflow), 0나누기, 비정상적 메모리 접근, 배열 색인 범위 초과 등 정상적인 동작이 불가할 때 발생하는 이벤트
- ✓ 일부 예외 이벤트 발생은 프로세스의 종료로 이어짐(운영체제 보호 차원)
- ✓ 예외 이벤트에 대한 처리: 예외처리(exception handling) 또는 오류처리(trouble shooting)

# 1. 예외(Exception) [2/4]

❖ 예외 vs. 오류(error)

✓ 에러

› 동작중인 프로그램에 이상이 발생

› 정상동작을 위한 복구가 불가한 상황

› 보통, 디버깅에 어려움이 존재

✓ 예외

› 예외 처리를 위한 명령코드를 통해 복구 및 비정상적인 동작에 대한 대처가 가능

› 단, 에러의 일부는 예외에 포함되고 있음.

# 1. 예외(Exception) [3/4]

❖ 유형

✓ 검사 예외(checked exception)

› 컴파일 타임 예외(compile time exception)라고도 함

› 반드시 에러 처리를 해야함

› 예상되는 예외상황(문제)에 대한 대체작업을 미리 준비하게 해주는 예외

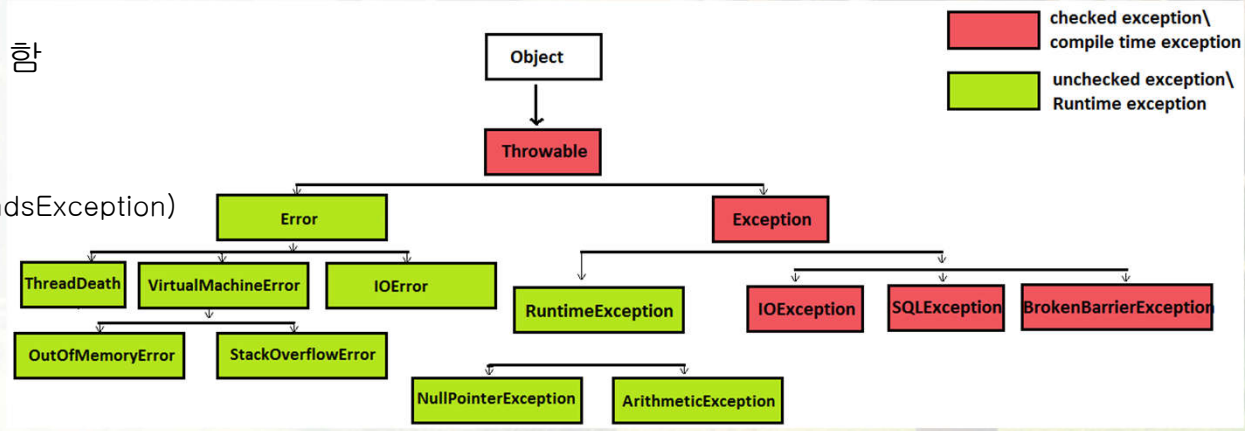
✓ 비검사 예외(unchecked exception)

› 런타임 예외(runtime exception)이라고도 함

› 동작 중에 발생할 수 있는 예외

- 예 1: 배열 인덱싱의 범위 밖(ArrayIndexOutOfBoundsException)
- 예 2: null 참조(NullPointerException)

› 에러 처리를 강제하지 않음(선택적)




※ 이미지 출처:
https://www.javamadesoeasy.com/2015/05/exception-handling-exception-hierarchy.html

# 1. 예외(Exception) [4/4]

❖ 대표적인 원인

✓ 잘못된 메모리 장치 접근

› 예: 배열의 인덱싱, 타 영역 접근시도, 소유권 및 권한 밖의 영역의 접근 등

✓ 불가능한 연산

› 예: 0으로 나누기, 무한으로 곱하기 등

✓ 자료형 불일치 및 형변환 불가

› 예: 문자열 + 정수형, Student class → Teacher class 등

✓ 비논리적인 알고리즘

✓ 하드웨어 오동작

✓ 운영체제의 비정상 동작

✓ 잘못된 파일 접근

# 2. 함수 강제 종료

❖ 방법

✓ “return” 문 활용

› 반환값을 통해 함수를 종료(정상종료)

✓ 예외 발생

› 오류가 발생했음을 시스템에게 알림

› 시스템은 해당 스택을 정리하고, 상위 스택(부모 함수)로 이동시킴

✓ “exit()”류의 함수 호출

› 프로그램을 종료시키는 기능을 수행

✓ 스레드 종료관련 함수 호출

› 예: thread_stop(), thread_kill(), thread_abort() 등

› 스레드로 만들어진 함수를 강제로 종료시킴

# 3. 예외 처리 [1/2]

❖ 예외 처리(Exception Handling)

- ✓ 실행 중 오류 발생 시, 해당 오류에 대응을 위한 방법을 정의
- ✓ 예외가 발생했을 때 이를 처리하기 위한 코드를 처리하는 과정
- ✓ 프로그램이 예외 상황에 안정적으로 대응할 수 있도록 함
- ✓ 예외 처리 적용 코드 구성:
  - › "try" 블록 :
    - 예외 발생 가능성이 있는 명령문들의 나열
  - › "catch" 블록 :
    - 다수의 블록(Multiple Catch Blocks) 정의 가능
    - 예외가 발생했을 때, 해당 예외를 처리 하기 위한 명령문들을 나열
  - › "finally" 블록 :
    - 모든 코드 블록("try" 또는 "catch")에 대한 처리 완료 후 최종 실행되는 명령들을 나열
    - 일반적으로 자원 반납이나 종료 등의 기능을 구현

```java
public class SimpleExceptionHandling {
    public static void main(String[] args) {
        try {
            int result = 10 / 0;
        } catch (ArithmeticException e) {
            System.out.println("Attempted division by zero");
        } finally {
            System.out.println("This block always executes");
        }
    }
}
```

[Java의 예외 처리 예]

**예외 전파**

- Exception Propagation
- 하위 함수에서 처리하지 않은 예외를 상위 함수로 전달
- 예외를 호출 스택 상위에서 적절히 처리할 수 있음

# 3. 예외 처리 [2/2]

❖ 장점

✓ 신뢰성/안정성 향상

› 예상치 못한 상황에서도 프로그램이 안정적으로 동작하도록 보장

✓ 디버깅 용이

› 예외 발생 시 그 원인을 로그에 기록 또는 표시

› 프로그램의 오류를 더 쉽게 알거나 코드에서의 위치를 찾아낼 수 있음

✓ 자원 관리

› 프로그램이 비정상적으로 종료될 때 열린 파일이나 네트워크 연결과 같은 자원을 적절히 해제

› 자원 누수를 방지

# 4. 언어별 예외처리 [1/4]

❖ Java

✓ <try-statement> ::=
"try" … {"catch"…}["finally"…]

✓ "try"하단의 명령수행 중 예외발생을 허용함

✓ 예외 발생시 "catch"에 해당되는 예외종류가 실행

✓ 마지막으로 "finally"영역의 명령들이 실행됨

[Java의 예외처리 코드 예]

Main.java

```java
public class ExceptionHandlingExample {
    public static void main(String[] args) {
        try {
            int nResult = 10 / 0;
            System.out.println(nResult);
        } catch (ArithmeticException e) {
            System.out.println("Error: " + e.getMessage());
        } finally {
            System.out.println("This will always execute.");
        }
    }
}
```

Output

```
ERROR!
Error: / by zero
This will always execute.

=== Code Execution Successful ===
```

대표적 용어

- try-catch 문
- 대다수의 언어에서 try-catch-… 의 형태를 갖고 있음

# 4. 언어별 예외처리 [2/4]

❖ Python

✓ <try-statement> ::=
"try" … {"except"… }["else"…] … ["finally"…]

✓ <try>문으로 작성된 내용은 예외 처리가 가능

✓ 예외는 <except>에서 각 등록된 부분에서 처리됨

✓ <except>가 하나라도 실행되지 않고 <try> 부분이 모두 실행되면 <else>에 해당되는 명령들이 실행됨

✓ <finally>는 최종 실행될 명령들을 정의함

› 실패/성공과 관련없이 가장 마지막에 실행됨

✓ "throw" 키워드를 이용하여 예외 생성이 가능(강제 함수종료)

main.py

```python
try:
    with open('example.txt', 'r') as file:
        content = file.read()
        print(content)
except FileNotFoundError as e:
    print(f"Error: {e}")
except IOError as e:
    print(f"Error: {e}")
else:
    print("File read successfully.")
finally:
    print("Finished file operation.")

```

[Python의 예외처리 코드 예]

# 4. 언어별 예외처리 [3/4]

❖ C++

✓ <try-statement> ::= “try” … {“catch”… }

✓ “finally” 키워드와 관련 유사기능(리소스 초기화 등)은 지원하지 않음

✓ “throw” 키워드를 이용하여 예외 생성이 가능(강제 함수종료)

main.cpp

```cpp
#include <iostream>
#include <stdexcept> // Include to use std::runtime_error

int divide(int numerator, int denominator) {
    if (denominator == 0) {
        // Throw an exception when denominator is zero
        throw std::runtime_error("Division by zero attempted");
    }
    return numerator / denominator;
}

int main() {
    int a = 10;
    int b = 0;
    try {
        int result = divide(a, b);
        std::cout << "Result: " << result << std::endl;
    } catch (const std::runtime_error& e) {
        // Handle exceptions that are of type std::runtime_error
        std::cout << "Caught a runtime error: " << e.what() << std::endl;
    } catch (...) {
        // Catch all other types of exceptions
        std::cout << "An unexpected error has occurred." << std::endl;
    }

    return 0;
}
```

Output

```
ERROR!
Caught a runtime error: Division by zero attempted


=== Code Execution Successful ===
```

# 4. 언어별 예외처리 [4/4]

❖ C#

✓ <try-statement> ::=

"try" … {"catch"…}["finally"…]

✓ 문법과 동작 형태는 Java와 동일

✓ "throw" 키워드를 통해 임의로 예외 생성이 가능

Main.cs

```
1  using System;
2
3  class Program
4  {
5      static double Divide(int numerator, int denominator)
6      {
7          if (denominator == 0)
8          {
9              throw new DivideByZeroException("Cannot divide by zero.");
10         }
11         return (double)numerator / denominator;
12     }
13
14     static void Main(string[] args)
15     {
16         try
17         {
18             int a = 10;
19             int b = 0;
20             double result = Divide(a, b);
21             Console.WriteLine("The result is " + result);
22         }
23         catch (DivideByZeroException ex)
24         {
25             Console.WriteLine("An error occurred: " + ex.Message);
26         }
27         catch (Exception ex)
28         {
29             Console.WriteLine("General error: " + ex.Message);
30         }
31         finally
32         {
33             Console.WriteLine("Execution of the finally block.");
34         }
35     }
36 }
```

Output

```
An error occurred: Cannot divide by zero.
Execution of the finally block.

=== Code Execution Successful ===
```

# 5. Java의 예외 발생 및 처리

❖ try-catch와 throws의 관계

✓ try-catch

› 메서드의 내용부(body)에서 사용되는 키워드

› 발생한 예외를 처리하기 위한 블록

› catch() 영역에 해당 예외가 있으면, 그 부분을 처리

✓ throws

› 메서드의 머리부(header)에서 사용되는 키워드

› 발생할 예외 항목들을 이후에 나열

- 예: void print(…) throws IOException {…}

› 이 메서드를 호출하는 곳에서는 이 예외 항목들에 대한 처리 구문을 필요(선택적)

Main.java

```java
public class ExceptionDemo {

    public static void checkValue(int value) throws IllegalArgumentException,
        ArithmeticException
    {
        if (value < 0) {
            throw new IllegalArgumentException("It's less than 0.");
        }
        if (value == 0) {
            throw new ArithmeticException("It tries to divide by zero.");
        }

        System.out.println("OK: " + value);
    }

    public static void main(String[] args) {
        try {
            checkValue(-1);
        } catch (IllegalArgumentException e) {
            System.out.println("ILLegalArgument Error: " + e.getMessage());
        } catch (ArithmeticException e) {
            System.out.println("Arithmetic Error: " + e.getMessage());
        }

        try {
            checkValue(0);
        } catch (IllegalArgumentException e) {
            System.out.println("IllegalArgument Error: " + e.getMessage());
        } catch (ArithmeticException e) {
            System.out.println("Arithmetic Error: " + e.getMessage());
        }

        try {
            checkValue(10);
        } catch (IllegalArgumentException e) {
            System.out.println("IllegalArgument Error: " + e.getMessage());
        } catch (ArithmeticException e) {
            System.out.println("Arithmetic Error: " + e.getMessage());
        }
    }
}
```

# 6. 예외 방지 [1/12]

❖ 예외 방지

✓ 프로그램 동작의 안정성/신뢰성 확보를 위한 고려사항

✓ 예외가 발생하지 않도록 명령문을 작성

✓ 예외가 발생하기전 능동적으로 점검

❖ 방지 예

✓ 객체 접근 시 Null 값 점검

✓ 파일 및 네트워크 자원 가용성 확인

# 6. 예외 방지 [2/12]

❖ 점검 사항

1. 입력 검증(Input Validation)
2. 기본값 사용(Use of Default Values)
3. 조건부 오류 처리(Error Handling with Conditionals)
4. 실패 방지 코딩 관행(Fail-safe Coding Practices)
5. 자원 및 오류 검사(Resource and Error Checks)
6. 적절한 메모리 관리(Proper Memory Management)
7. 견고한 라이브러리 함수 사용(Use of Robust Library Functions)
8. 철저한 테스트(Comprehensive Testing)
9. 적절한 옵션값 처리(Handling Optional Values Properly)
10. 동시성 관리(Concurrency Management)

# 6. 예외 방지 [3/12]

❖ 입력 검증(Input Validation)

✓ 모든 사용자 입력은 처리하기 전에 검증되어야 함

✓ 예상치 못한 형식(formats), 유형(types), 값(values) 등을 검사

✓ 유효하지 않은 데이터 입력에 따른 처리 중 발생되는 예외를 방지

✓ 예:

› 숫자 입력을 처리하기 전에 실제 숫자인지, 예상 범위 내에 있는지 등을 확인

[입력 검증 예]

main.py

```python
def validate_input(user_input):
    try:
        value = int(user_input)
        if 1 <= value <= 100:
            print("Valid input. Processing...")

            return True
        else:
            print("Input is out of the allowed range (1-100).")
            return False
    except ValueError:
        print("Invalid input: Please enter a valid integer.")
        return False

user_input = input("Enter an integer between 1 and 100: ")
validate_input(user_input)
```

Output

```
Enter an integer between 1 and 100: 0
Input is out of the allowed range (1-100).

=== Code Execution Successful ===
```

# 6. 예외 방지 [4/12]

❖ 기본값 사용(Use of Default Values)

✓ 변수나 매개변수에 기본값을 할당

✓ null 포인터 예외 등의 오류를 방지

✓ 예:

› 리스트(list)는 null이 아닌 비어 있는 값(empty)로 초기화

› 배열은 0값 등으로 초기화

› 문자열의 경우 빈 문자열로 초기화

› 함수 매개변수에 기본값을 설정 등

[기본값 사용 예]

Main.java

```
1  public class CDefaultValueExample {
2      public static void printLength(String str) {
3          if (str == null) {
4              str = "";
5          }
6          System.out.println("String length: " + str.length());
7      }
8
9      public static void main(String[] args) {
10         printLength(null);
11         printLength("Hello, world!");
12     }
13 }
```

Output

```
java -cp /tmp/MsHZqF1GuQ/CDefaultValueExample
String length: 0
String length: 13

=== Code Execution Successful ===
```

# 6. 예외 방지 [5/12]

❖ 조건부 오류 처리(Error Handling with Conditionals)

✓ 예외를 발생시킬 수 있는 연산을 수행하기 전에 조건 검사를 사용하여 연산이 안전한지 확인

✓ 예시:

› 변수로 나누기 전에 그 변수가 0이 아닌지 확인

› 자연수 데이터(양의 정수)의 뺄셈일 경우, 후자의 값이 더 작은 값인지 확인

› 배열의 경우 색인(index)이 요소(elements)의 개수보다 큰지 확인

[조건부 오류처리 예]

Main.java

```
1  public class CArrayAccess {
2      public static void main(String[] args) {
3          int[] numbers = {1, 2, 3, 4, 5};
4          int index = 5;
5
6          if (index < numbers.length) {
7              System.out.println("Element at "
8              + index + " is " + numbers[index]);
9          } else {
10             System.out.println("Index out of bounds");
11         }
12     }
13 }
```

Output

```
java -cp /tmp/7gkmRN4bJI/CArrayAccess
Index out of bounds

=== Code Execution Successful ===
```

# 6. 예외 방지 [6/12]

❖ 실패 방지 코딩 관행

(Fail-safe Coding Practices)

✓ 잠재적 오류를 예상하고 잘 대처할 코딩 스타일

✓ 안전한 API와 내부 오류를 잘 처리하는 라이브러리를 사용

✓ 예:

› 문자열을 정수로 변환

• C#의 int.TryParse 같은 안전한 파싱 메소드를 사용

› 이 메소드는 변환이 실패해도 예외를 발생하지 않음

› 결과에 대해 성공(true) 또는 실패(false)를 반환

[실패 방지 코드 이용 예]

Main.cs

```
1  using System;
2
3  public class CTryParseExample
4  {
5      public static void Main()
6      {
7          string[] values = { null, "160519", "9432.0", "16,667",
8                              "   -322   ", "+4302", "(100);", "01FA" };
9          foreach (var value in values)
10         {
11             int number;
12
13             bool success = int.TryParse(value, out number);
14             if (success)
15             {
16                 Console.WriteLine($"Converted '{value}' to {number}.");
17             }
18             else
19             {
20                 Console.WriteLine($"Conversion of '{value ?? "<null>"}' failed.");
21             }
22         }
23     }
24 }
```

Output

```
mono /tmp/vXdBH8IVNo.exe
Conversion of '<null>' failed.
Converted '160519' to 160519.
Conversion of '9432.0' failed.
Conversion of '16,667' failed.
Converted '   -322   ' to -322.
Converted '+4302' to 4302.
Conversion of '(100);' failed.
Conversion of '01FA' failed.

=== Code Execution Successful ===
```

# 6. 예외 방지 [7/12]

❖ 자원 및 오류 검사

(Resource and Error Checks)

✓ 사용하기 전에 자원의 가용성과 상태(availability and state)를 항상 확인

✓ 예:

› 파일 읽기 전 파일 존재 여부 검사, 데이터베이스 쿼리 전 데이터베이스 연결 확인 등

[자원 및 오류 검사 예]

Main.java

```java
import java.io.File;
import java.io.FileReader;
import java.io.IOException;

public class CFileChecker {

    public static void main(String[] args) {
        File file = new File("example.txt");

        if (!file.exists()) {
            System.out.println("파일이 존재하지 않습니다.");
        } else {
            try {
                FileReader fr = new FileReader(file);
                System.out.println("파일을 성공적으로 열었습니다.");
                fr.close();
            } catch (IOException e) {
                System.out.println("파일을 읽기오류 발생: " + e.getMessage());
            }
        }
    }
}
```

```
andrew@goblin:~/PL/JavaException$ ls
CFileChecker.java  ExceptionDemo.class  ExceptionDemo.java
andrew@goblin:~/PL/JavaException$ javac CFileChecker.java
andrew@goblin:~/PL/JavaException$ java CFileChecker
파일이 존재하지 않습니다.
andrew@goblin:~/PL/JavaException$
```

# 6. 예외 방지 [8/12]

❖ 적절한 메모리 관리

(Proper Memory Management)

✓ C 및 C++과 같은 언어에서는 메모리를 명시적으로 관리

✓ 메모리 누수(memory leaks) 및 접근 위반(access violation)을 방지

✓ 예:

› 모든 malloc 또는 new 연산은 free 또는 delete와 짝을 이루어 메모리 누수를 방지

[적절한 메모리 관리 예]

```c
#include <stdio.h>
#include <stdlib.h>

int
main()
{
    char *pBuffer = NULL;

    pBuffer = (char*)malloc(100);
    sprintf(pBuffer, "Hello World");

    if(pBuffer!=NULL)
        printf("Buffer : %s\n", pBuffer);

    free(pBuffer);
    pBuffer=NULL;

    return 0;
}
```

```
andrew@goblin:~/PL/c$ gcc malloc.c  -o malloc
andrew@goblin:~/PL/c$ ./malloc
Buffer : Hello World
andrew@goblin:~/PL/c$
```

# 6. 예외 방지 [9/12]

❖ 견고한 라이브러리 함수 사용(Use of Robust Library Functions)

✓ 내부적으로 예외를 처리하는 라이브러리 함수를 사용

✓ 새 코드 또는 충분히 테스트되지 않은 코드 사용 지양

✓ 예:

› 오픈 라이브러리 사용시 stable 버전을 사용

# 6. 예외 방지 [10/12]

❖ 철저한 테스트(Comprehensive Testing)

✓ 단위 테스트(Unit Test), 통합 테스트(Integration Test), 과부하 테스트(Stress Test)를 포함한 철저한 테스트를 수행

✓ 예외를 초래할 수 있는 시나리오를 발견하고 수정

✓ 예:

› Java의 JUnit을 이용한 단위 시험

› C#의 NUnit을 이용한 단위 시험

› Python의 PyTest를 이용한 시험 등

› 각종 시험 환경을 적용

**Verification vs Validation**

- **Verification(검증)**
  - 출력물이 잘 만들어지고 있는지 평가하는 과정
  - 코드 검토, 디자인 검토, 문서 검토, 컴파일, 단위 테스트 등으로 수행됨
  - 과정에 초점을 둠

- **Validation(확인)**
  - 사용자의 요구와 기대를 만족하는지 확인
  - 시스템 테스트, 통합 테스트, 사용성 테스트, 성능 테스트 등이 해당
  - 결과물의 평가에 초점을 둠

# 6. 예외 방지 [11/12]

❖ 적절한 옵션값 처리

(Handling Optional Values Properly)

✓ Swift와 Kotlin과 같이 옵셔널 타입을 지원하는 언어에서는 이러한 타입을 사용

✓ 값의 부재를 명시적으로 처리하여 null 사용을 피함

✓ 예시:

› Kotlin의 널(null) 가능 타입을 사용하고 ?. (안전한 호출 연산자) 및 ?: (엘비스 연산자)을 사용

[옵션값 처리 예]

Kotlin

1.9.24 JVM Program arguments

```
fun main()
{
    val number: Int? = null

    val squared = number?.let { it * it }

    println(squared)

    val result = squared ?: "값이 없습니다."

    println(result)
}
```

```
null
값이 없습니다.
```

# 6. 예외 방지 [12/12]

❖ 동시성 관리(Concurrency Management)

✓ 멀티스레드 환경에서는 잠금(lock), 세마포어(semaphore), 뮤텍스(mutual-exclusion) 또는 기타 동기화 메커니즘을 사용

✓ 공유 자원(shared resources)에 대한 동시 접근을 방지 및 관리

✓ 경쟁 상태(contention States) 및 데드락(deadlock)을 방지

# Thank you

– Q & A –