

■ 오류의 종류

● 에러(Error)

- 하드웨어의 잘못된 동작 또는 고장으로 인한 오류
- 에러가 발생되면 프로그램 종료
- 정상 실행 상태로 돌아갈 수 없음

● 예외(Exception)

- 사용자의 잘못된 조작 또는 개발자의 잘못된 코딩으로 인한 오류
- 예외가 발생되면 프로그램 종료
- 예외 처리 추가하면 정상 실행 상태로 돌아갈 수 있음

컴파일 오류 (Compile Error) – 컴파일 불가

런타임 오류 (Runtime Error) – 실행 중 오류

논리 오류 (Logical Error) – 버그 (흐름상 잘못된 코딩)

■ 오류의 종류

```
public class ErrorExam {  
    public static void main(String[] args) {  
        int a = 1.2; // compile error  
  
        // runtime error  
        System.out.println(4/0); // Arithmetic  
        System.out.println(new String().charAt(1)); // IndexOutOfBounds  
        String str = null;  
        System.out.println(str.equals("")); // NullPointerException  
  
        int[] arrs = new int[-1]; // NegativeArraySize  
  
        String s = "Exception";  
        int count = s.indexOf("a");  
        int[] arrays = new int[count]; // NegativeArraySize  
        System.out.println(arrays);  
    }  
}
```

■ if문을 이용한 예외처리

- 나이를 입력하라고 했는데, 0보다 작은 값이 입력되었다.
- 나눗셈을 위한 두 개의 정수를 입력 받는데, 제수(나누는 수)로 0이 입력되었다.
- 주민등록번호 13자리만 입력하라고 했더니, 중간에 -를 포함하여 14자리를 입력하였다.

이렇듯 프로그램의 실행 도중에 발생하는 문제의 상황을 가리켜 **예외**라 한다.

예외는 컴파일 오류와 같은 문법의 오류와는 의미가 다르다.

```
System.out.print("피제수 입력 : ");  
int num1=keyboard.nextInt();  
  
System.out.print("제수 입력 : ");  
int num2=keyboard.nextInt();
```

```
if(num2==0)  
{  
    System.out.println("제수는 0이 될 수 없습니다.");  
    i-=1;  
    continue;  
}
```

이것이 지금까지 우리가 사용해온 예외의 처리 방식이다. 이는 **if문이 프로그램의 주 흐름인지, 아니면 예외의 처리인지** 구분이 되지 안된다는 단점이 있다.

■ if문을 이용한 예외처리 (1 / 2)

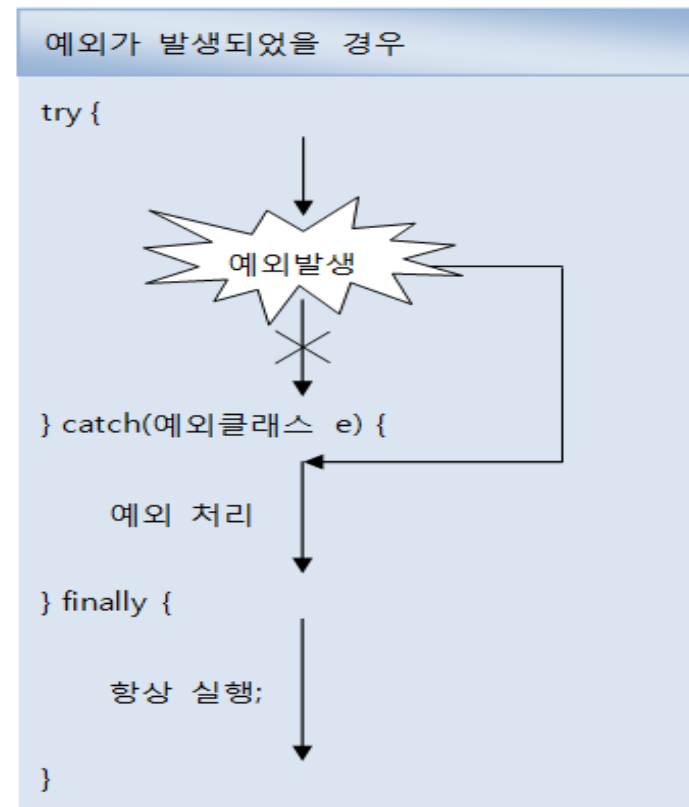
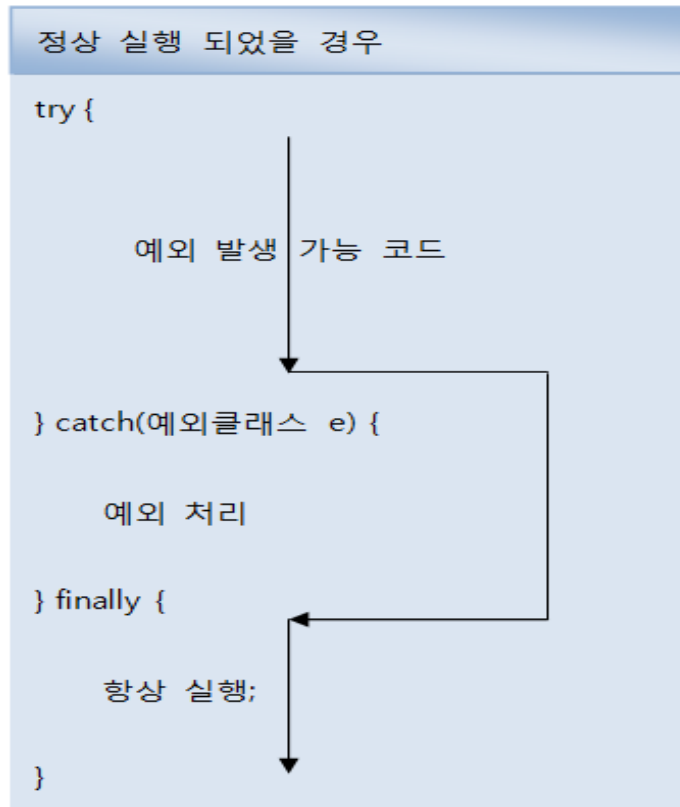
```
public class ExceptionExam1 {  
    public static void main(String[] args) {  
        String numStr = " 123";  
        int num = Integer.parseInt(numStr); // NumberFormatException  
  
        Object obj = new String("a");  
  
        int a = (Integer) obj; // ClassCastException  
    }  
}
```

■ if문을 이용한 예외처리 (2 / 2)

```
public class ExceptionExam1 {  
    public static void main(String[] args) {  
        String numStr = " 123";  
  
        Pattern p = Pattern.compile("[0-9]*$");  
        Matcher m = p.matcher(numStr);  
        boolean isNumber = m.matches();  
        if(isNumber) {  
            int num = Integer.parseInt(numStr);  
        }  
  
        Object obj = new String("a");  
        if(obj instanceof Integer) {  
            int a = (Integer) obj;  
        } else if(obj instanceof String) {  
            String a = (String) obj;  
        }  
    }  
}
```

■ 예외 처리 방법

- 예외 발생시 프로그램 종료 막고, 정상 실행 유지할 수 있도록 처리
 - Exception (일반 예외) : 반드시 작성해야 컴파일 가능
 - Runtime Exception (실행 예외) : 개발자 경험 의해 작성 (컴파일시 체크 X)
- try – catch – finally 블록 이용해 예외 처리 코드 작성



■ try-catch-finally를 이용한 예외처리

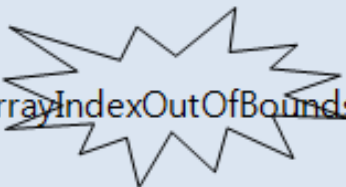
```
public class ExceptionExam2 {  
    public static void main(String[] args) {  
        String numStr = " 123";  
        try {  
            int num = Integer.parseInt(numStr);  
        } catch(NumberFormatException e) {  
            e.printStackTrace();  
        } finally {  
            System.out.println("항상 실행 1");  
        }  
    }  
}
```

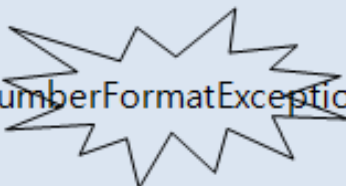
```
        Object obj = new String("a");  
        try {  
            int a = (Integer) obj;  
        } catch(ClassCastException e) {  
            e.printStackTrace();  
        } finally {  
            System.out.println("항상 실행 2");  
        }  
    }  
}
```

■ 다중 예외 처리 방법

● 예외 별로 처리 코드 다르게 구현

```
try {
```

 **ArrayIndexOutOfBoundsException** 발생

 **NumberFormatException** 발생

```
} catch(ArrayIndexOutOfBoundsException e) {
```

예외 처리 1

```
} catch(NumberFormatException e) {
```

예외 처리 2

```
}
```



■ 다중 예외 처리 방법

- 상위 예외 클래스는 가장 마지막에 처리

try {

ArrayIndexOutOfBoundsException 발생

NumberFormatException 발생

} catch(**Exception** e) {

예외 처리 1

} catch(~~ArrayIndexOutOfBoundsException~~ e) {

예외 처리 2

}

try {

ArrayIndexOutOfBoundsException 발생

다른 Exception 발생

} catch(**ArrayIndexOutOfBoundsException** e) {

예외 처리 1

} catch(**Exception** e) {

예외 처리 2

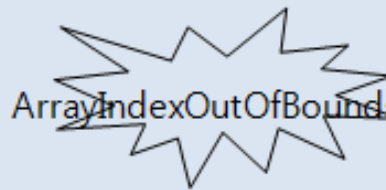
}

■ 다중 예외 처리 방법

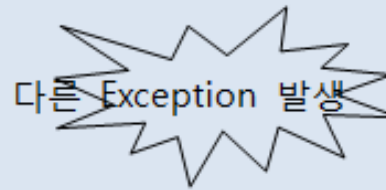
● multi catch

- 자바 1.7 이상부터 사용 가능
- 하나의 catch 블록에 | (파이프 라인)을 사용하여 여러 개 처리 가능

```
try {
```



ArrayIndexOutOfBoundsException 또는 NumberFormatException 발생



다른 Exception 발생

```
} catch(ArrayIndexOutOfBoundsException | NumberFormatException e) {
```

예외 처리 1

```
} catch(Exception e) {
```

예외 처리 2

```
}
```

■ 예외 던지기 (throws)

● 메소드의 선언부에 작성

```
리턴타입 메소드명(매개변수,...) throws 예외클래스 1, 예외클래스 2, ... {  
}
```

● 메소드를 호출한 곳으로 던지기는 역할

```
public void method1() {  
    try {  
        method2();  
    } catch(ClassNotFoundException e) {  
        //예외 처리 코드  
        System.out.println("클래스가 존재하지 않습니다.");  
    }  
}  
  
public void method2() throws ClassNotFoundException {  
    Class clazz = Class.forName("java.lang.String2");  
}
```

호출한 곳에서 예외 처리

■ 예외 떠넘기기 - 1 (throws)

```
public class ExceptionExam3 {  
    public static void main(String[] args)  
        throws NumberFormatException, ClassCastException {  
        String numStr = " 123";  
        int num = Integer.parseInt(numStr);  
  
        Object obj = new String("a");  
        int a = (Integer) obj;  
    }  
}
```

■ 예외 떠넘기기 - 2 (throws)

```
public class ExceptionExam4 {  
    public static void main(String[] args) {  
        ExceptionExam4 ee =  
            new ExceptionExam4();  
  
        try {  
            ee.check();  
        } catch(NumberFormatException e) {  
            e.printStackTrace();  
        } catch(ClassCastException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

```
    public void check()  
        throws NumberFormatException,  
               ClassCastException {  
        String numStr = " 123";  
        int num = Integer.parseInt(numStr);  
  
        Object obj = new String("a");  
        int a = (Integer) obj;  
    }  
}
```

■ 의도적인 예외 발생

1. 먼저, 연산자 `new`를 이용해서 발생시키려는 예외 클래스의 객체를 만든 다음

```
Exception e = new Exception("고의로 발생시켰음");
```

2. 키워드 `throw`를 이용해서 예외를 발생시킨다.

```
throw e;
```

[예제8-6]/ch8/ExceptionEx6.java

```
class ExceptionEx6
{
    public static void main(String args[])
    {
        try {
            Exception e = new Exception("고의로 발생시켰음.");
            throw e; // 예외를 발생시킴
            // throw new Exception("고의로 발생시켰음.");

        } catch (Exception e) {
            System.out.println("에러 메시지 : " + e.getMessage());
            e.printStackTrace();
        }
        System.out.println("프로그램이 정상 종료되었습니다.");
    }
}
```

위의 두 줄을 한 줄로
줄여 쓸 수 있다.

[실행결과]

```
에러 메시지 : 고의로 발생시켰음.
java.lang.Exception: 고의로 발생시켰음.
    at ExceptionEx6.main(ExceptionEx6.java:6)
프로그램이 정상 종료되었습니다.
```

■ 의도적인 예외 발생

```
public class ExceptionExam5 {  
    public static void main(String[] args) {  
        ExceptionExam5 ee =  
            new ExceptionExam5();  
  
        try {  
            ee.check();  
        } catch (AccessException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

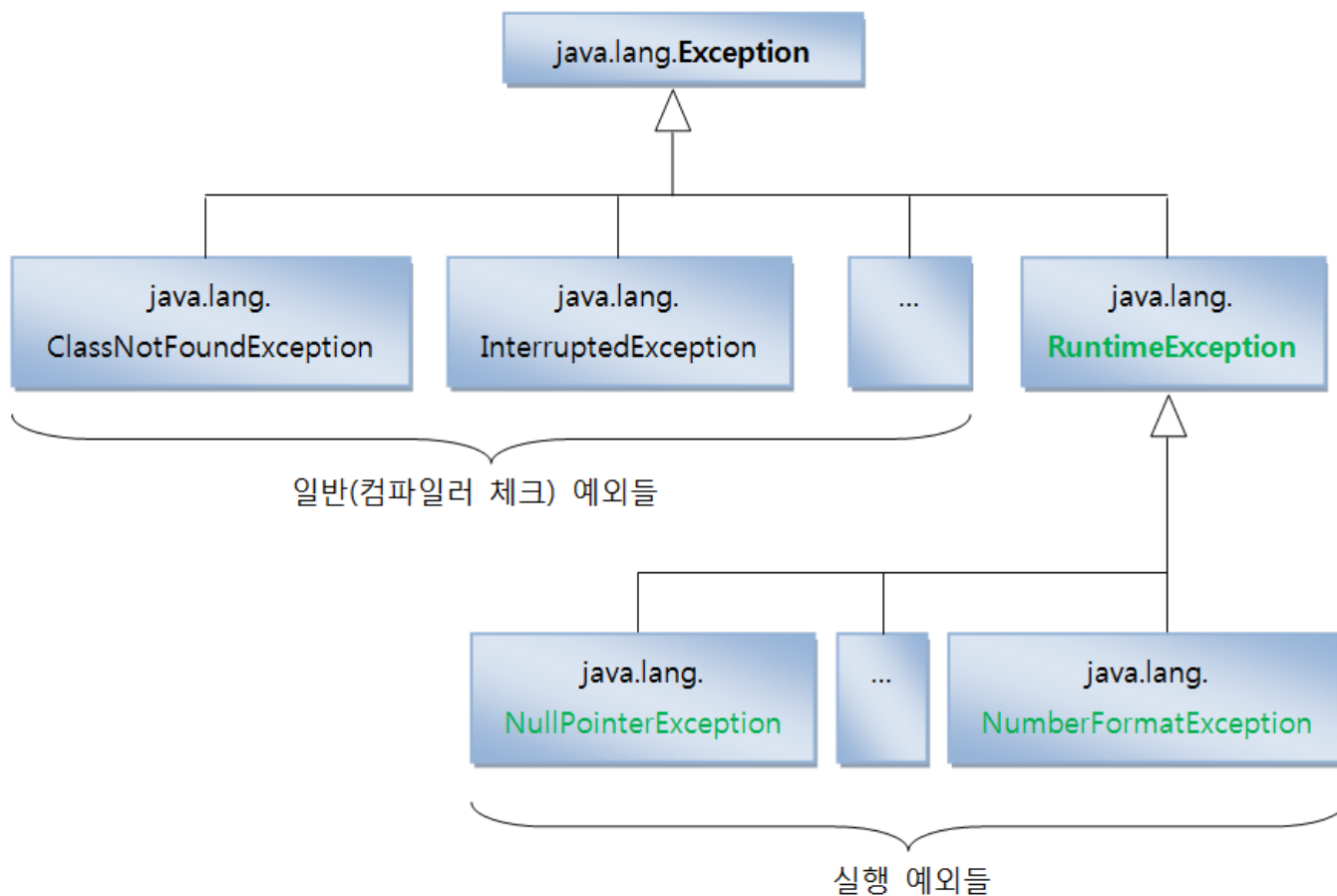
```
public void check() throws AccessException {  
    try {  
        String numStr = " 123";  
        int num = Integer.parseInt(numStr);  
  
        Object obj = new String("a");  
        int a = (Integer) obj;  
    } catch (NumberFormatException e) {  
        throw new AccessException(  
            "숫자변환 실패");  
    } catch (ClassCastException e) {  
        throw new AccessException(  
            "형변환 실패");  
    }  
}
```

■ 예외 클래스의 계층구조

- 예외 클래스는 크게 두 그룹으로 나뉜다.

`RuntimeException` 클래스들 - 프로그래머의 실수로 발생하는 예외 ← 언체크 예외

`Exception` 클래스들 - 사용자의 실수와 같은 외적인 요인에 의해 발생하는 예외 ← 체크 예외



■ RuntimeException

RuntimeException을 상속하는 대표적인 예외 클래스

- ArrayIndexOutOfBoundsException
- ClassCastException
- NegativeArraySizeException
- NullPointerException

이들은 try~catch문, 또는 throws절을 반드시 필요로 하지 않기 때문에 지금까지 예외처리 없이 예제를 작성할 수 있었다!

■ RuntimeException

● NullPointerException

- 객체 참조가 없는 상태
- null 값을 갖는 참조변수로 객체 접근 연산자인 도트(.) 사용했을 때 발생

```
String data = null;  
System.out.println(data.toString());
```

● ArrayIndexOutOfBoundsException

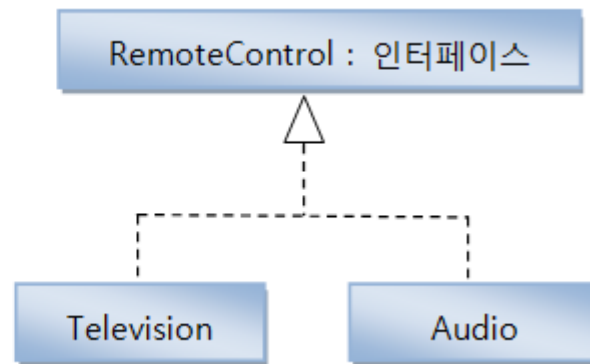
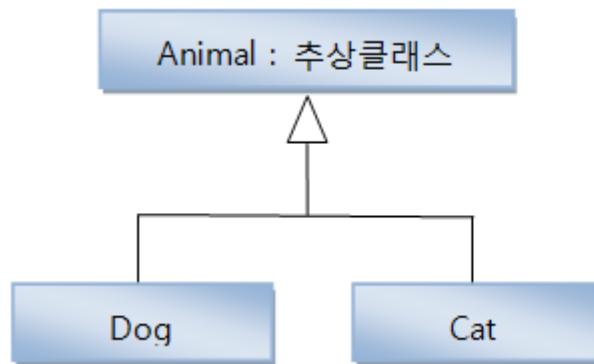
- 배열에서 인덱스 범위 초과하여 사용할 경우 발생

```
public static void main(String[] args) {  
    String data1 = args[0];  
    String data2 = args[1];  
  
    System.out.println("args[0]: " + data1);  
    System.out.println("args[1]: " + data2);  
}
```

■ RuntimeException

● ClassCastException

- 타입 변환이 되지 않을 경우 발생



- 정상 코드

```
Animal animal = new Dog();
Dog dog = (Dog) animal;
```

```
RemoteControl rc = new Television();
Television tv = (Television) rc;
```

- 예외 발생 코드

```
Animal animal = new Dog();
Cat cat = (Cat) animal;
```

```
RemoteControl rc = new Television();
Audio audio = (Audio) rc;
```

■ Check / UnCheck Exception

```
public class ExceptionExam6 {  
    public static void main(String[] args) {  
        File file = new File("c:/test.txt");  
        // check exception  
        try {  
            file.createNewFile();  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
  
        // unchecked exception  
        String str = null;  
        str.length();  
    }  
}
```

■ 사용자 정의 예외 클래스

- 자바 표준 API에서 제공하지 않는 예외를 사용하려는 경우
- 애플리케이션 서비스와 관련된 예외
ex) 잔고 부족 예외, 계좌 이체 실패 예외, 회원 가입 실패 예외....
- 선언 방법

```
public class XXXException extends [ Exception | RuntimeException ] {  
    public XXXException() { }  
    public XXXException(String message) { super(message); }  
}
```

■ 사용자 정의 예외 클래스 작성 - 1

```
public class CheckException extends Exception {  
    public CheckException(String message) {  
        super(message);  
    }  
}
```

```
public class UnCheckException extends RuntimeException {  
    public UnCheckException(String message) {  
        super(message);  
    }  
}
```

■ 예외 정보 출력

● getMessage()

- 예외 발생시킬 때 생성자 매개값으로 사용한 메시지 리턴

```
throw new XXXException("예외 메시지");
```

- 원인 세분화하기 위해 예외 코드 포함 (예: 데이터베이스 예외 코드)

- catch() 절에서 활용

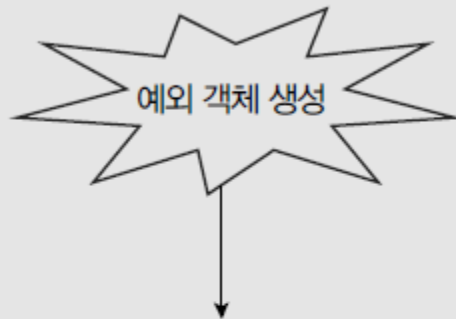
```
} catch(Exception e) {  
    String message = e.getMessage();  
}
```

■ 예외 정보 출력

● printStackTrace()

- 예외 발생 코드 추적한 내용을 모두 콘솔에 출력
- 프로그램 테스트하면서 오류 찾을 때 유용하게 활용

```
try {
```



```
} catch(예외클래스 e) {  
    //예외가 가지고 있는 Message 얻기  
    String message = e.getMessage();  
  
    //예외의 발생 경로를 추적  
    e.printStackTrace();  
}
```


■ 예외를 처리하지 않는 경우

```
public static void main(String[] args) throws AgeInputException
{
    System.out.print("나이를 입력하세요 : ");
    int age=readAge();
    System.out.println("당신은 "+age+"세입니다.");
}

public static int readAge() throws AgeInputException
{
    Scanner keyboard=new Scanner(System.in);
    int age=keyboard.nextInt();
    if(age<0)
    {
        AgeInputException excpt=new AgeInputException();
        throw excpt;
    }
    return age;
}
```

예외가 발생은 되었는데, 처리하지 않으면, 계속해서 반환이 되어 main 메소드를 호출한 가상머신에게 전달이 된다.

나이를 입력하세요 : -2

Exception in thread "main" AgeInputException : 유효하지 않은 나이가 입력되었습니다.

at ThrowsFromMain.readAge(ThrowsFromMain.java : 26)

at ThrowsFromMain.main(ThrowsFromMain.java : 16)

} 메소드의 호출관계(예외의 전달 흐름)을 보여주는 printStackTrace 메소드의 호출결과

- 가상머신의 예외처리 1 getMessage 메소드를 호출한다.
- 가상머신의 예외처리 2 예외상황이 발생해서 전달되는 과정을 출력해준다.
- 가상머신의 예외처리 3 프로그램을 종료한다

가상머신의 예외처리 방식