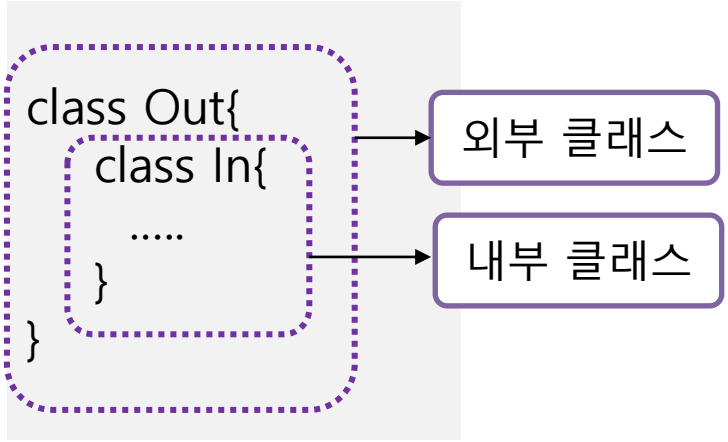
11장. 중첩 클래스 및 예외 처리

Exception

내부 클래스

내부 클래스 정의와 유형

- ✓ 클래스 내부에 선언한 클래스(inner class), 중첩 클래스라고도 한다.
- ✓ 내부에 클래스를 선언한 이유는 이 클래스와 외부 클래스가 밀접한 관련이 있거나 다른 클래스와 협력할 일이 없는 경우에 사용한다.



분류	선언 위치	설명

인스턴스 멤버 클래스	class A{
	class B{...}
	}
		A 객체를 생성해야만 사용할 수 있는 B 내부 클래스

정적 멤버 클래스	class A{
	static class B{...}
	}
		A 클래스로 바로 접근할 수 있는 B내부 클래스

로컬 클래스	class A{
	void method(){
	class B{...}
	}
	}
		method()를 실행할 때만 사용할 수 있는 B 중첩 클래스

내부 클래스

```
package innerclass;

class A{
    A(){System.out.println("A 객체가 생성됨");}

    // 인스턴스 멤버 클래스
    class B{
        B(){System.out.println("B 객체가 생성됨");}
        int field1;
        //static int field2;
        void method1() {}
        //static void method2();
    }

    //정적 멤버 클래스
    static class C{
        C(){System.out.println("C 객체가 생성됨");}
        int field1;
        static int field2;
        void method1() {}
        static void method2() {}
    }

    //local 클래스 - 매소드 내에서 선언된 클래스
    //static 사용 불가
    void method() {
        class D{
            D(){System.out.println("D 객체가 생성됨");}
            int field1;
            //static int field2;
            void method1() {}
            //static void method2() {}
        }
        D d = new D();
        d.field1 = 3;
        d.method1();
    }
}
```

내부 클래스

```
public class NestedClassTest {
    public static void main(String[] args) {

        A a = new A();

        //인스턴스 멤버 클래스 객체 생성
        A.B b = a.new B();
        b.field1 = 4;
        b.method1();

        //정적 멤버 클래스 객체 생성
        A.C c = new A.C();
        c.field1 = 5;
        c.method1();
        A.C.field2 = 6;
        A.C.method2();

        //로컬 클래스 객체 생성을 위한 메소드 호출
        a.method();
    }
}
```

```
A 객체가 생성됨
B 객체가 생성됨
C 객체가 생성됨
D 객체가 생성됨
```

내부 클래스

인스턴스 멤버 클래스

```
package innerclass;

class Student{

    String name;

    Student(String name) {
        this.name = name;
    }

    class Score{
        int eng;
        int math;

        void showInfo() {
            System.out.println("이름 : " + name);
            System.out.println("영어 점수: " + eng);
            System.out.println("수학 점수: " + math);
        }
    }
}
```

내부 클래스

```
public class StudentTest {
    public static void main(String[] args) {
        //외부 클래스 객체 생성
        Student jang = new Student("장그래");

        //내부 클래스 객체 생성
        Student.Score score = jang.new Score();

        score.eng = 88;
        score.math = 73;
        score.showInfo();
    }
}
```

내부 클래스

정적 멤버 클래스

```
class OutClass{
    int num = 10;
    static int sNum = 20;

    //정적 내부 클래스
    static class InClass{
        int inNum = 100;
        static int sInNum = 200;

        void inTest() {
            //num += 10;    //외부 클래스의 인스턴스 변수는 사용못함
            System.out.println(sNum + "(외부 클래스의 정적 변수 사용)");
            System.out.println(inNum + "(내부 클래스의 인스턴스 변수 사용)");
            System.out.println(sInNum + "(내부 클래스의 정적 변수 사용)");
        }

        static void sTest() {
            //num += 20;        //사용 못함
            //inNum += 10;      //사용 못함
            System.out.println(sNum + "(외부 클래스의 정적 변수 사용)");
            System.out.println(sInNum + "(내부 클래스의 정적 변수 사용)");
        }
    }
}
```

내부 클래스

정적(static) 멤버 클래스

```
public class StaticInnerClassTest {

    public static void main(String[] args) {
        //중첩 클래스의 객체 생성
        OutClass.InClass inClass = new OutClass.InClass();

        System.out.println("**정적 내부 클래스의 일반 메서드 호출**");
        inClass.inTest();

        System.out.println("**정적 내부 클래스의 정적 메서드 호출**");
        OutClass.InClass.sTest();

    }
}
```

```
===== 정적 내부 클래스의 일반 메서드 호출 =====
20(외부 클래스의 정적 변수 사용)
100(내부 클래스의 인스턴스 변수 사용)
200(내부 클래스의 정적 변수 사용)
===== 정적 내부 클래스의 정적 메서드 호출 =====
20(외부 클래스의 정적 변수 사용)
200(내부 클래스의 정적 변수 사용)
```

내부 클래스

지역 내부 클래스

Module java.base
Package java.lang

Interface Runnable

Runnable 인터페이스는 스레드(thread)를 만들때 사용하는 인터페이스로 반드시 run()메서드를 구현해야 한다.

```
class Outer{
    int outNum = 100;
    static int sNum = 200;

    Runnable getRunnable() {
        int num = 10;  //인터페이스 상수

        //메서드의 내부 클래스로 사용
        class MyRunnable implements Runnable{
            int localNum = 20;

            @Override
            public void run() {
                //num = 20;
                System.out.println(outNum + "(외부클래스의 인스턴스 변수)");
                System.out.println(sNum + "(외부클래스의 정적 변수)");
                System.out.println(localNum + "(내부클래스의 멤버 변수)");
            }
        }
        //MyRunnable myRun = new MyRunnable();
        //return myRun;
        return new MyRunnable();
    }
}
```

내부 클래스

```
public class LocalInnerTest {

    public static void main(String[] args) {
        Outer outer = new Outer();
        outer.getRunnable().run();

        System.out.println("*** 인터페이스형으로 변환 ***");
        Runnable runner = outer.getRunnable();
        runner.run();
    }
}
```

```
100(외부 클래스의 인스턴스 변수)
200(외부 클래스의 정적 변수)
20(내부 클래스의 멤버 변수)
=== 인터페이스형으로 형 변환 ===
100(외부 클래스의 인스턴스 변수)
200(외부 클래스의 정적 변수)
20(내부 클래스의 멤버 변수)
```

익명 내부 클래스

익명(Annoymous) 내부 클래스

- 클래스 이름을 사용하지 않는 클래스가 있고, 이런 클래스를 **익명 클래스**라고 부른다.
- 사용할 땐 중괄호 블록 뒤에 세미콜론(;)을 먼저 붙여야 한다.

```
public class Outer2 {
    int outNum = 100;
    static int sNum = 200;

    Runnable runner = new Runnable() { //익명 구현 클래스
        int num = 10;

        @Override
        public void run() {
            System.out.println(outNum + "(외부 클래스의 인스턴스 변수)");
            System.out.println(sNum + "(외부 클래스의 정적 변수)");
            System.out.println(num + "(익명 클래스의 멤버 변수)");
        }
    };  //세미콜론을 붙인다.
}
```

익명 내부 클래스

```
public class LocalInnerTest2 {

    public static void main(String[] args) {
        Outer2 out = new Outer2();
        out.runner.run();
    }
}
```

```
100(외부 클래스의 인스턴스 변수)
200(외부 클래스의 정적 변수)
10(익명 클래스의 멤버 변수)
```

익명 객체

익명 자식 객체 생성

자식 클래스를 재사용하지 않고 오로지 특정 위치에서 사용할 경우라면 자식 클래스를 명시적으로 선언하지 않고, 익명 자식 객체를 사용하는것이 좋음

```
부모클래스[필드] = new 부모클래스(매개값, ..){
    //필드
    //메서드
};
```

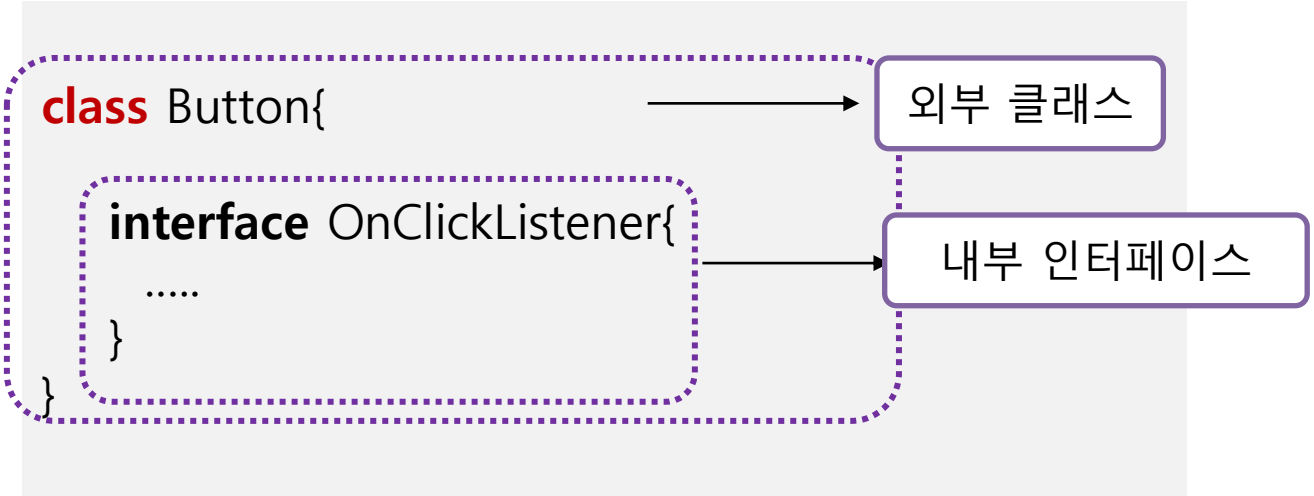
하나의 실행문이므로 끝에는 세미콜론(;)을 반드시 붙여야 한다.

내부(중첩) 인터페이스

내부 인터페이스

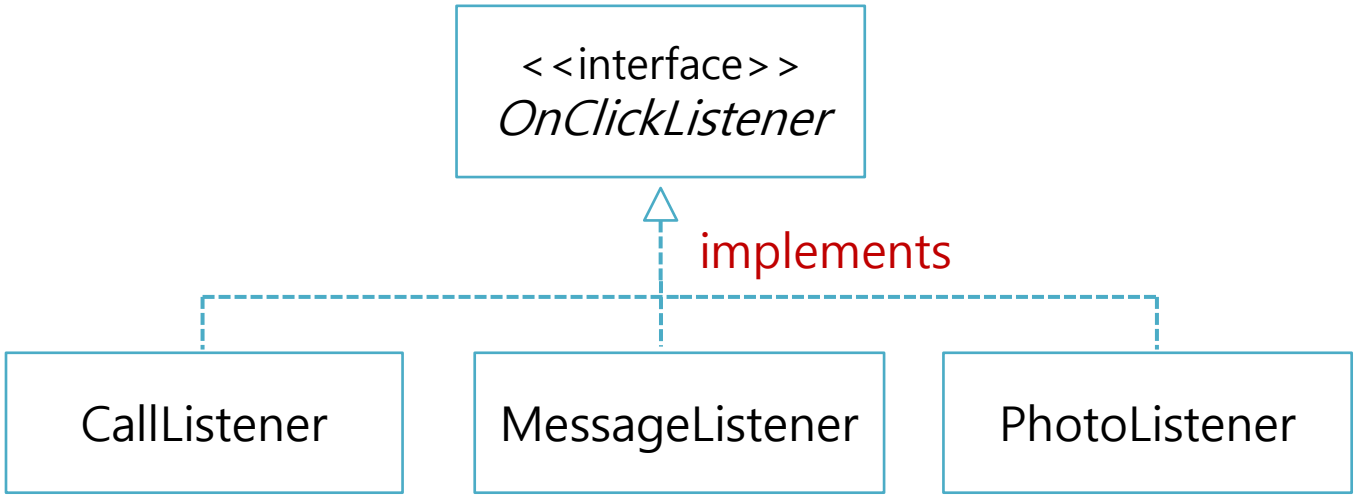
클래스의 멤버로 선언된 인터페이스를 중첩 인터페이스라 한다.

인터페이스를 클래스 내부에 선언하는 이유는 해당 클래스와 긴밀한 관계를 맺는 구현 클래스를 만들기 위함이다.



내부(중첩) 인터페이스

버튼을 클릭했을때 이벤트를 처리하는 객체 만들기



내부(중첩) 인터페이스

내부 인터페이스 사용 예제

```
package innerinterface;

public class Button {

    private OnClickListener listener; //인터페이스형 멤버 변수(필드)

    interface OnClickListener{ //내부 인터페이스
        public void onClick();
    }

    public void setListener(OnClickListener listener) {
        //OnClickListener 객체를 매개변수로 전달 받음
        this.listener = listener;
    }

    public void touch() {
        listener.onClick();
    }
}
```

내부(중첩) 인터페이스

내부 인터페이스 – 구현 클래스 만들기

```
public class CallListener implements Button.OnClickListener{
    //Button 클래스의 OnClickListener에 접근 -> 구현 클래스 만들기
    @Override
    public void onClick() {
        System.out.println("전화를 겁니다.");
    }

}
```

```
public class MessageListener implements Button.OnClickListener{
    //Button 클래스의 OnClickListener에 접근
    @Override
    public void onClick() {
        System.out.println("문자를 보냅니다.");
    }
}
```

내부(중첩) 인터페이스

내부 인터페이스 테스트 – 익명 객체로 구현하기

```
public class ButtonTest {
    public static void main(String[] args) {
        Button button = new Button();
        //CallListener 객체를 매개변수로 전달
        button.setListener(new CallListener());
        button.touch();

        button.setListener(new MessageListener());
        button.touch();

        //익명 객체 구현 (구현 클래스 만들지 않음) - 사진찍기
        button.setListener(new Button.OnClickListener(){

            @Override
            public void onClick() {
                System.out.println("사진을 찍습니다.");
            }
        });
        button.touch();
```

```
전화를 겁니다.
문자를 보냅니다.
사진을 찍습니다.
```

에러와 예외

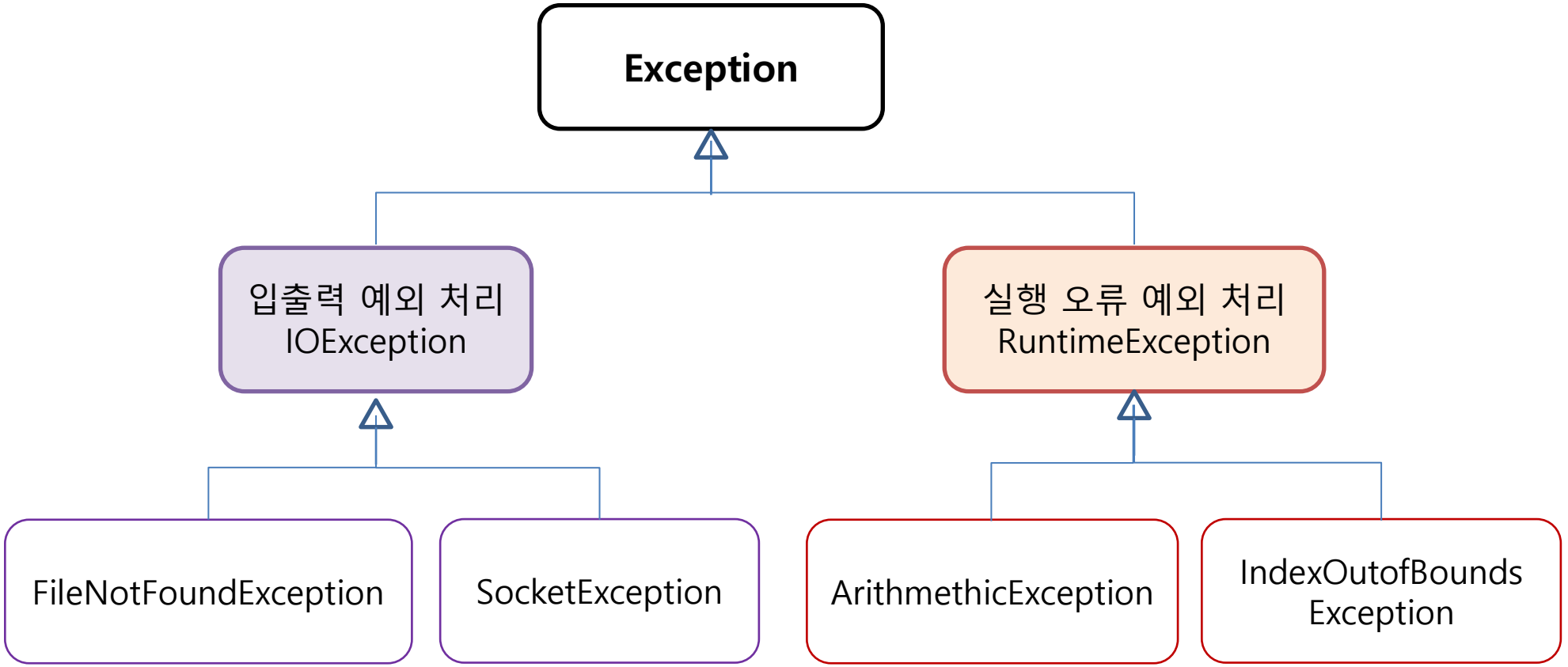
오류의 종류

1. **에러(Error)**
   - 하드웨어의 오작동 고장으로 인한 오류
   - 에러가 발생되면 프로그램이 종료되고 정상으로 돌아갈수 없음
2. **예외(Exception)**
   - 사용자의 잘못된 조작 또는 개발자의 잘못된 코딩으로 인한 오류
   - 예외가 발생되면 프로그램이 종료되고, 예외 처리를 하면 정상으로 돌아갈 수 있음

예외의 종류

1. **일반 예외(Exception)**
   - 예외 처리가 없으면 컴파일 되지 않는 예외 – 컴파일 체크
2. **실행 예외(RuntimeException)**
   - 예외 처리를 생략해도 컴파일 되는 예외
   - 개발자의 경험과 판단으로 예외 코드 작성 필요

예외 클래스의 종류


Exception
입출력 예외 처리
IOException
실행 오류 예외 처리
RuntimeException
FileNotFoundException
SocketException
ArithmethicException
IndexOutofBounds
Exception

예외 클래스의 종류

Java.lang 패키지 -> Exception Summary

Exception Summary

Exception	Description
ArithmeticException	Thrown when an exceptional arithmetic condi
ArrayIndexOutOfBoundsException	Thrown to indicate that an array has been acc
ArrayStoreException	Thrown to indicate t
ClassCastException	Thrown to indicate t
ClassNotFoundException	Thrown when an app
CloneNotSupportedException	Thrown to indicate t
EnumConstantNotPresentException	Thrown when an app
Exception	The class Exception

Module java.base
Package java.lang

Class ArithmeticException

java.lang.Object
  java.lang.Throwable
    java.lang.Exception
      java.lang.RuntimeException
        java.lang.ArithmeticException

All Implemented Interfaces:

Serializable

실행 예외

NullPointerException

```
package exception.example;

public class ExceptionExample1 {

    public static void main(String[] args) {
        String data = null;
        System.out.println(data.toString()); //NullPointerException 발생
    }
}
```

ArrayIndexOutOfBoundsException

```
public class ExceptionExample2 {

    public static void main(String[] args) {
        int[] num = new int[2];

        num[0] = 1;
        num[1] = 2;
        num[2] = 3;

        //ArrayIndexOutOfBoundsException
        System.out.println("완료");
    }
}
```

실행 예외

NumberFormatException

```
public class ExceptionExample3 {

    public static void main(String[] args) {
        String data1 = "100";
        String data2 = "a200";

        int value1 = Integer.parseInt(data1);
        int value2 = Integer.parseInt(data2); //NumberFormatException 발생

        int result = value1 + value2;
        System.out.println(data1 + "+" + data2 + "=" + result);
    }
}
```

try ~ catch문

try ~ catch문

예외처리를 하면 예외 상황을 알려 주는 메시지를 볼 수 있고, 프로그램이 비정상적으로 종료되지 않고 계속 수행되도록 만들 수 있다.

```
try{
    예외가 발생할 수 있는 코드
}catch(처리할 예외 타입 e){
    예외를 처리하는 코드
}
```

try ~ catch문

실행 예외

```
package exception.handling;

public class Exceptionhandling1 {

    public static void main(String[] args) {
        try {
            int[] num = new int[2];

            num[0] = 1;
            num[1] = 2;
            num[2] = 3;    //예외 발생

            System.out.println("1, 2, 3 저장");
        }catch(ArrayIndexOutOfBoundsException e) {
            System.out.println("배열의 범위를 벗어났습니다.");
        }
        System.out.println("Done");
     }
}
```

try ~ catch문

일반 예외 – 컴파일러 체크

```
public class Exceptionhandling3 {

    public static void main(String[] args) {

        try {
            Class cls = Class.forName("java.lang.String2");
        } catch (ClassNotFoundException e) {
            System.out.println("클래스가 존재하지 않습니다.");
        }
    }
}
```

try~catch~finally문

try~catch~finally문 사용하기

프로그램에서 외부장치와의 연동시 초기화나 마무리 작업시 주로 사용한다.

이때 사용하는 블록이 finally인데 일단 try블록이 수행되면 어떤 경우에도 반드시 수행된다.

정상실행 되었을 경우

```
try{
    예외 발생 가능 코드
}catch(예외클래스 e){
    예외 처리
} finally {
    항상 실행
}
```

예외가 발생되었을 경우

```
try{
    예외발생
}catch(예외클래스 e){
    예외 처리
} finally {
    항상 실행
}
```

try~catch~finally문

```
public class TryCatchFinally {

    public static void main(String[] args) {
        try {
            String data1 = args[0];
            String data2 = args[1];
            int value1 = Integer.parseInt(data1);
            int value2 = Integer.parseInt(data2);

            int result = value1 + value2;
            System.out.println(data1 + "+" + data2 + "=" + result);
        }catch(ArrayIndexOutOfBoundsException e) {
            System.out.println("실행 매개값의 수가 부족합니다.");
        }catch(NumberFormatException e) {
            System.out.println("숫자로 변환할 수 없습니다.");
        }finally {
            System.out.println("다시 실행하세요.");
        }
    }
}
```

try~catch~finally문

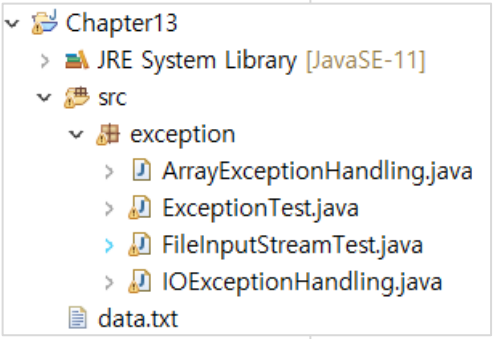
다중 try ~ catch문 사용하기

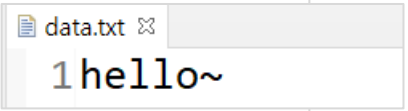
✓ 예외 상황이 여러 개라면 catch 블록을 예외 상황 수만큼 구현해야 한다

```
public class FileInputStreamTest {

    public static void main(String[] args) {
        try {
            FileInputStream fis = new FileInputStream("data.txt");
            //data.txt의 내용을 읽어서 출력하기
            int i;
            while((i=fis.read()) !=-1) {
                System.out.print((char)i);
            }
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

상위 예외클래스를 아래쪽에 위치시켜야 함





예외 처리 – throws

throws 로 예외처리 미루기(떠넘기기)

예외 처리를 해당 메서드에서 하지 않고 미룬 후, 메서드를 호출하여 사용하는 곳에서 예외를 처리하는 방법이다.

```
메서드명 throws 예외클래스1, 예외클래스2,..{
}
```

예외 처리 – throws

throws 로 예외처리 미루기(떠넘기기)

```
public class ThrowsException {

    public static void main(String[] args) {
        try {
            //호출하는 쪽에서 try ~ catch 처리
            findClass();
        } catch (ClassNotFoundException e) {
            System.out.println("클래스가 존재하지 않습니다.");
        }
    }

    public static void findClass() throws ClassNotFoundException {
        Class cls = Class.forName("java.lang.String2");
    }
}
```

예외 처리 – throws

```
class ArrayUtil {
    public void call() throws Exception {
        System.out.println("call 메서드 시작");
        int[] num = new int[2];
        num[0] = 1;
        num[1] = 2;
        //num[2] = 3;

        System.out.println("call 매서드 종료");
    }
}

public class ThrowsTest {

    public static void main(String[] args) {
        ArrayUtil util = new ArrayUtil();
        try {
            util.call();
            System.out.print("Hello");
        } catch (Exception e) {
            System.out.println("main 매서드에서 예외 처리");
        }
        System.out.println(" World");
    }
}
```