

```
for object to mirror...
mirror_mod.mirror_object = ...

operation == "MIRROR_X":
    mirror_mod.use_x = True
    mirror_mod.use_y = False
    mirror_mod.use_z = False
operation == "MIRROR_Y":
    mirror_mod.use_x = False
    mirror_mod.use_y = True
    mirror_mod.use_z = False
operation == "MIRROR_Z":
    mirror_mod.use_x = False
    mirror_mod.use_y = False
    mirror_mod.use_z = True

#selection at the end -add
mirror_ob.select= 1
modifier_ob.select=1
context.scene.objects.active = ...
("Selected" + str(modifier_ob.name))
mirror_ob.select = 0
= bpy.context.selected_objects[0]
data.objects[one.name].select = 1

print("please select exactly one object")

-- OPERATOR CLASSES --

bpy.types.Operator):
    """X mirror to the selected object.mirror_mirror_x"""
    mirror X"
```

# Java 기초

예외처리

# 예외처리

---

- 예외처리의 정의와 목적

에러(error) – 프로그램 코드에 의해서 수습될 수 없는 심각한 오류

예외(exception) – 프로그램 코드에 의해서 수습될 수 있는 다소 미약한 오류

- 에러(error)는 어쩔 수 없지만, 예외(exception)는 처리해야 한다.

- 예외처리의 정의와 목적

예외처리란?

정의 - 프로그램 실행 시 발생할 수 있는 예외의 발생에 대비한 코드를 작성하는 것

목적 - 프로그램의 비정상 종료를 막고, 정상적인 실행상태를 유지하는 것

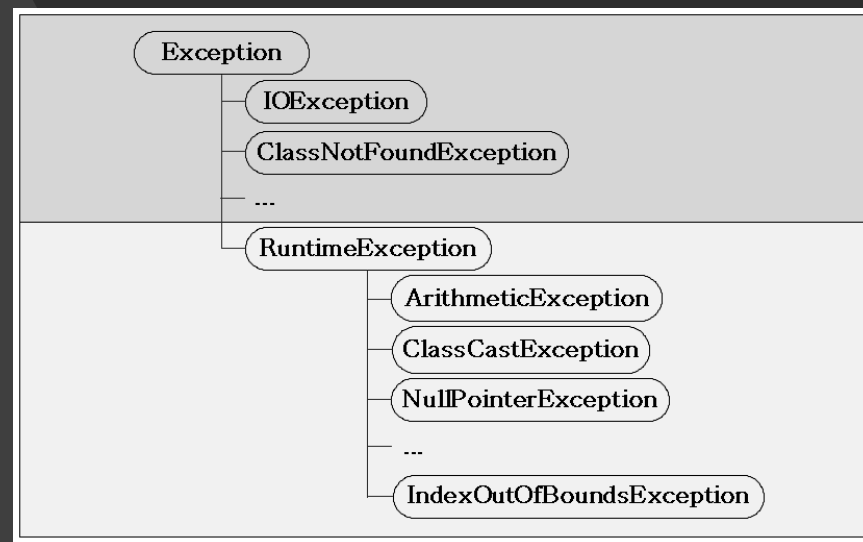
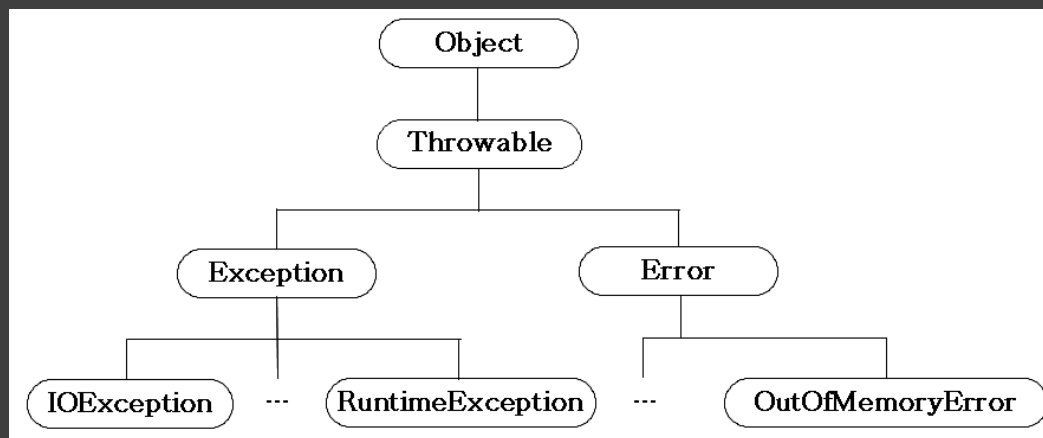
# 예외처리

- 예외의 종류

- 예외는 크게 두 부류로 나뉜다.

RuntimeException 클래스들 – 프로그래머의 실수로 발생하는 예외 => 예외처리 필수

Exception 클래스들 – 사용자의 실수와 같은 외적인 요인에 의해 발생하는 예외 => 예외처리 선택



# 예외처리

---

- try~catch

```
Try{  
    //에러가 발생할 문장  
}catch (Exception1 e1) {  
    //Exception1에 해당하는 에러 발생 시 조건 수행  
}catch (Exception2 e2) {  
    //Exception2에 해당하는 에러 발생 시 조건 수행  
    .....  
  
}catch (ExceptionN eN) {  
    //ExceptionN에 해당하는 에러 발생 시 조건 수행  
}catch (Exception e) {  
    //Exception에 해당하는 에러 발생 시 조건 수행  
}
```

# 예외처리

---

- try~catch 예제

```
public class TryCatchException {  
    public static void main(String[] args) {  
        try{  
            Scanner scan = new Scanner(System.in);  
            String n = scan.next();  
            System.out.println(n);  
        }catch(NullPointerException e1){  
            e1.printStackTrace();  
        }catch(ArrayIndexOutOfBoundsException e2){  
            e2.printStackTrace();  
        }catch(Exception e){  
            e.printStackTrace();  
        }  
    }  
}
```

# 예외처리

- try~catch 유의점
  - Exception은 모든 예외처리 클래스들의 최상의 클래스이다.
  - try문에서 Exception이 발생하면 아래 모든 구문이 무시되고 바로 해당 Exception문으로 들어간다.
  - Exception이 발생시에 해당 조건에 해당하는 Exception이 없으면 Catch문을 무시하고 이후 바로 아래의 명령줄을 실행한다.
  - 위와 같은 상황이 발생되지 않기 위해서 맨 마지막에 catch(Exception e)를 선언한다.(Exception 클래스는 모든 예외를 처리할 수 있다)

```
package com.hb.operator;

public class ExceptionTest1 {
    public static void main(String[] args) {
        System.out.println(1);
        System.out.println(2);
        try{
            System.out.println(3);
            System.out.println(4);
        }catch(Exception e){
            System.out.println(5);
        }
        System.out.println(6);
    }
}
```

1
2
3
4
6

```
package com.hb.operator;

public class ExceptionTest2 {
    public static void main(String[] args) {
        System.out.println(1);
        System.out.println(2);
        try{
            System.out.println(3);
            System.out.println(0/0);
        }catch(Exception e){
            System.out.println(5);
        }
        System.out.println(6);
    }
}
```

1
2
3
5
6

# 예외처리

---

- finally
  - 예외의 발생여부와 관계없이 실행되어야 하는 코드를 넣는다.
  - 선택적으로 사용할 수 있으며, try-catch-finally의 순서로 구성된다.
  - 예외 발생시, try → catch → finally의 순서로 실행되고 예외 미발생시, try → finally의 순서로 실행된다.
  - try 또는 catch블럭에서 return문을 만나도 finally블럭은 수행된다

```
try {  
    // 예외가 발생할 가능성이 있는 문장들을 넣는다.  
} catch (Exception1 e1) {  
    // 예외처리를 위한 문장을 적는다.  
} finally {  
    // 예외의 발생여부에 관계없이 항상 수행되어야하는 문장들을 넣는다.  
    // finally블럭은 try-catch문의 맨 마지막에 위치해야한다.  
}
```

# 예외처리

---

- finally 예제

```
public class ExceptionFinalTest {  
    public static void main(String[] args) {  
        try{  
            System.out.println("try 문 내용입니다");  
            System.out.println(4/0);  
        }catch(Exception e){  
            System.out.println("catch 문 내용입니다");  
        }finally{  
            System.out.println("finally 문 내용입니다");  
        }  
    }  
}
```

```
try 문 내용입니다  
catch 문 내용입니다  
finally 문 내용입니다
```



# throw Exception

---

- Throw Exception
  - 예외를 처리하는 또 다른 방법
  - 사실은 예외를 처리하는 것이 아니라, 호출한 메서드로 전달해주는 것
  - 호출한 메서드에서 예외처리를 해야만 할 때 사용

```
void method() throws Exception1, Exception2, ... ExceptionN {  
    // 메서드의 내용  
}
```

**[참고]** 예외를 발생시키는 키워드 throw와 예외를 메서드에 선언할 때 쓰이는 throws를 잘 구별하자.

# throw Exception

---

- Throw Exception

```
public class ExceptionFinalTest {  
    public static void main(String[] args) throws Exception{  
        System.out.println("이것은 예외를 throw한 형태");  
        System.out.println(0/0);  
    }  
}
```

```
Exception in thread "main" 이것은 예외를 throw한 형태  
java.lang.ArithmeticException: / by zero  
    at com.hb.operator.ExceptionFinalTest.main(ExceptionFinalTest.java:6)
```

# throw Exception

- 예외 강제 발생시키기

1. 먼저, 연산자 new를 이용해서 발생시키려는 예외 클래스의 객체를 만든 다음

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

2. 키워드 throw를 이용해서 예외를 발생시킨다.

```
throw e;
```

```
public class TestExceptionClass {  
    public static void main(String[] args) {  
        try{  
            Exception e = new Exception("고의로 발생");  
            throw e;  
        }catch(Exception e){  
            System.out.println("에러메세지 : "+e.getMessage());  
        }  
        System.out.println("시스템 종료");  
    }  
}
```

```
에러메세지 : 고의로 발생  
시스템 종료
```

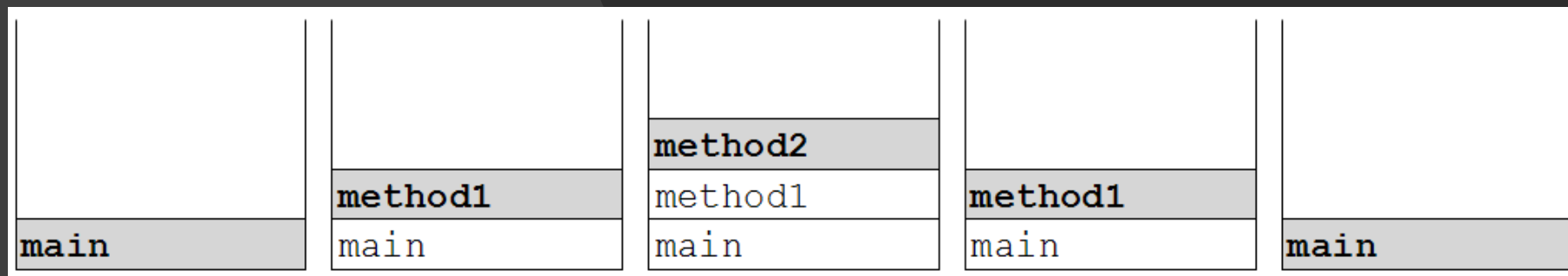
# 예외처리 예제

- 예외처리 예제 - 1

[예제8-18]/ch8/ExceptionEx18.java

```
class ExceptionEx18 {  
    public static void main(String[] args) throws Exception {  
        method1();    // 같은 클래스내의 static멤버이므로 객체생성없이 직접 호출가능.  
    }    // main메서드의 끝  
  
    static void method1() throws Exception {  
        method2();  
    }    // method1의 끝  
  
    static void method2() throws Exception {  
        throw new Exception();  
    }    // method2의 끝  
}
```

```
C:\WINDOWS\system32\CMD.exe  
C:\Wjdk1.5\work>java ExceptionEx18  
Exception in thread "main" java.lang.Exception  
    at ExceptionEx18.method2(ExceptionEx18.java:11)  
    at ExceptionEx18.method1(ExceptionEx18.java:7)  
    at ExceptionEx18.main(ExceptionEx18.java:3)
```



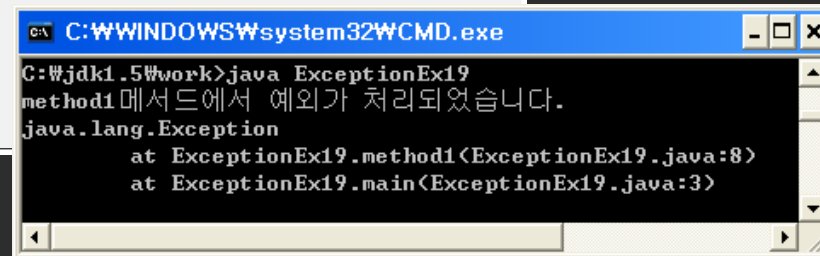
- 예외가 발생했을 때, 모두 3개의 메서드(main, method1, method2)가 호출스택에 있었으며,
- 예외가 발생한 곳은 제일 윗줄에 있는 method2()라는 것과
- main메서드가 method1()를, 그리고 method1()은 method2()를 호출했다는 것을 알 수 있다.

# 예외처리 예제

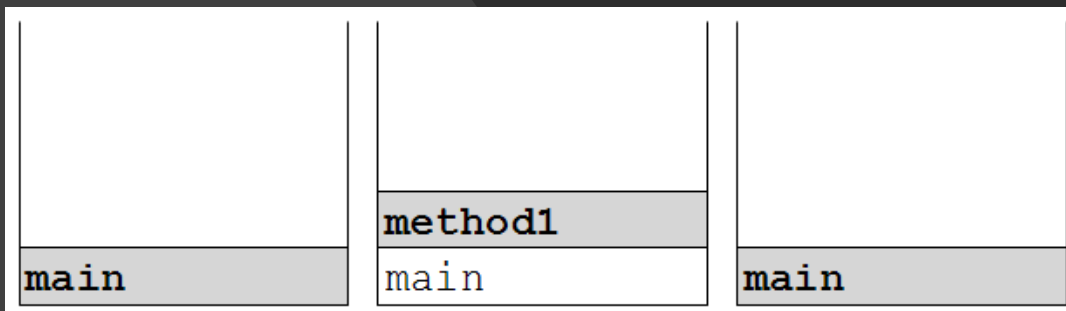
- 예외처리 예제 - 2

[예제8-19]/ch8/ExceptionEx19.java

```
class ExceptionEx19 {  
    public static void main(String[] args) {  
        method1();    // 같은 클래스내의 static멤버이므로 객체생성없이 직접 호출가능.  
    }    // main메서드의 끝  
  
    static void method1() {  
        try {  
            throw new Exception();  
        } catch (Exception e) {  
            System.out.println("method1메서드에서 예외가 처리되었습니다.");  
            e.printStackTrace();  
        }  
    }    // method1의 끝  
}
```



C:\WINDOWS\system32\cmd.exe  
C:\jdk1.5\work>java ExceptionEx19  
method1메서드에서 예외가 처리되었습니다.  
java.lang.Exception  
 at ExceptionEx19.method1(ExceptionEx19.java:8)  
 at ExceptionEx19.main(ExceptionEx19.java:3)



# 예외처리 예제

- 예외처리 예제 - 3

## [실행결과]

```
C:\jdk1.5\work>java ExceptionEx21 "test.txt"
test.txt 파일이 성공적으로 생성되었습니다.
```

```
C:\jdk1.5\work>java ExceptionEx21 ""
제목없음.txt 파일이 성공적으로 생성되었습니다.
```

```
C:\jdk1.5\work>dir *.txt
```

```
드라이브 c에 레이블이 없습니다
볼륨 일련 번호 251C-08DD
디렉터리 C:\jdk1.5\work
```

```
제목없음 TXT 0 03-01-24 0:47 제목없음.txt
TEST TXT 0 03-01-24 0:47 test.txt
```

## [예제8-21]/ch8/ExceptionEx21.java

```
import java.io.*;

class ExceptionEx21 {
    public static void main(String[] args) {
        // command line에서 입력받은 값을 이름으로 갖는 파일을 생성한다.
        File f = createFile(args[0]);
        System.out.println( f.getName() + " 파일이 성공적으로 생성되었습니다.");
    } // main메서드의 끝

    static File createFile(String fileName) {
        try {
            if (fileName==null || fileName.equals(""))
                throw new Exception("파일이름이 유효하지 않습니다.");
        } catch (Exception e) {
            // fileName이 부적절한 경우, 파일 이름을 '제목없음.txt'로 한다.
            fileName = "제목없음.txt";
        } finally {
            File f = new File(fileName); // File클래스의 객체를 만든다.
            createNewFile(f);           // 생성된 객체를 이용해서 파일을 생성한다.
            return f;
        }
    } // createFile메서드의 끝

    static void createNewFile(File f) {
        try {
            f.createNewFile(); // 파일을 생성한다.
        } catch (Exception e) { }
    } // createNewFile메서드의 끝
} // 클래스의 끝
```

# 예외처리 예제

- 예외처리 예제 - 4

## [실행결과]

```
C:\jdk1.5\work>java ExceptionEx22 test2.txt
test2.txt파일이 성공적으로 생성되었습니다.
```

```
C:\jdk1.5\work>java ExceptionEx22 ""
파일이름이 유효하지 않습니다. 다시 입력해 주시기 바랍니다.
```

## [예제8-22]/ch8/ExceptionEx22.java

```
import java.io.*;

class ExceptionEx22 {
    public static void main(String[] args)
    {
        try {
            File f = createFile(args[0]);
            System.out.println( f.getName()+"파일이 성공적으로 생성되었습니다.");
        } catch (Exception e) {
            System.out.println(e.getMessage()+" 다시 입력해 주시기 바랍니다.");
        }
    } // main메서드의 끝

    static File createFile(String fileName) throws Exception {
        if (fileName==null || fileName.equals(""))
            throw new Exception("파일이름이 유효하지 않습니다.");
        File f = new File(fileName); // File클래스의 객체를 만든다.
        // File객체의 createNewFile메서드를 이용해서 실제 파일을 생성한다.
        f.createNewFile();
        return f; // 생성된 객체의 참조를 반환한다.
    } // createFile메서드의 끝
} // 클래스의 끝
```

# 예외 되던지기

---

- 예외를 처리한 후에 다시 예외를 생성해서 호출한 메서드로 전달하는 것
- 예외가 발생한 메서드와 호출한 메서드, 양쪽에서 예외를 처리해야 하는 경우에 사용.

```
public class ReThrowTest {  
    public static void main(String[] args) {  
        try {  
            method();  
        } catch (Exception e) {  
            System.out.println("메인 Exception");  
            System.out.println(e.getMessage());  
        }  
    }  
    private static void method() throws Exception {  
        Exception e = new Exception("method에서 선언된 Exception");  
        throw e;  
    }  
}
```

```
메인 Exception  
method에서 선언된 Exception
```



# 사용자 정의 예외 클래스 작성

---

- 기존의 예외 클래스를 상속받아서 새로운 예외 클래스를 정의할 수 있다.

```
class MyException extends Exception {  
    MyException(String msg) { // 문자열을 매개변수로 받는 생성자  
        super(msg); // 조상인 Exception 클래스의 생성자를 호출한다.  
    }  
}
```

# 사용자 정의 예외 클래스 작성

---

- 에러코드를 저장할 수 있게 ERR\_CODE와 getErrCode()를 멤버로 추가

```
class MyException extends Exception {  
    // 에러 코드 값을 저장하기 위한 필드를 추가 했다.  
    private final int ERR_CODE;  
  
    MyException(String msg, int errCode) { // 생성자.  
        super(msg);  
        ERR_CODE = errCode;  
    }  
  
    MyException(String msg) { // 생성자.  
        this(msg, 100); // ERR_CODE를 100 (기본값)으로 초기화한다.  
    }  
  
    public int getErrCode() { // 에러 코드를 얻을 수 있는 메서드도 추가했다.  
        return ERR_CODE; // 이 메서드는 주로 getMessage()와 함께 사용될 것이다.  
    }  
}
```