

예외처리

- 1절. 예외와 예외 클래스
- 2절. 실행 예외
- 3절. 예외 처리 코드
- 4절. 예외 종류에 따른 처리 코드
- 5절. 자동 리소스 닫기
- 6절. 예외 처리 떠넘기기
- 7절. 사용자 정의 예외와 예외 발생

1절. 예외와 예외 클래스

■ 오류의 종류

■ 에러(Error)

- 하드웨어의 잘못된 동작 또는 고장으로 인한 오류
- 수습할 수 없는 심각한 문제를 의미
- 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

1절. 예외와 예외 클래스

■ 예외의 종류

■ 일반(컴파일 체크) 예외(Exception)

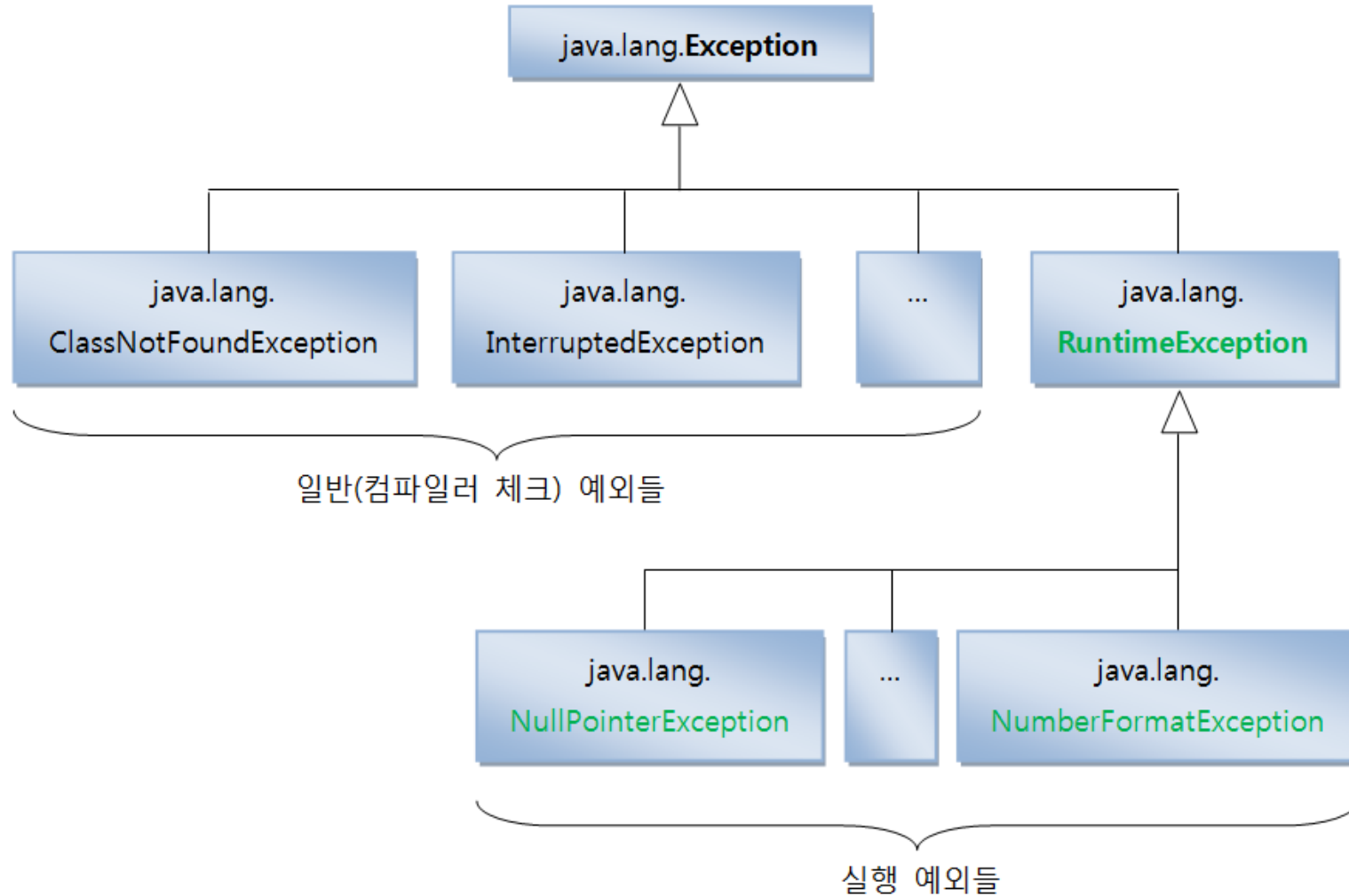
- 예외 처리 코드 없으면 컴파일 오류 발생

■ 실행 예외(RuntimeException)

- 예외 처리 코드를 생략하더라도 컴파일이 되는 예외
- 경험 따라 예외 처리 코드 작성 필요

1절. 예외와 예외 클래스

■ 예외 클래스



2절. 실행 예외(RuntimeException) 종류

- NullPointerException

- 객체 참조가 없는 상태

- null 값 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);  
}
```

2절. 실행 예외(RuntimeException) 종류

- NumberFormatException
 - 문자열을 숫자로 변화하는 경우가 많다.

반환타입	메소드명(매개 변수)
int	Integer.parseInt(String s)
double	Double.parseDouble(String s)

- 숫자로 변환될 수 없는 문자가 포함되어 있을 경우 발생

```
String data1 = "100";  
String data2 = "a100";
```

```
int value1 = Integer.parseInt(data1);  
int value2 = Integer.parseInt(data2);
```

2절. 실행 예외(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;
```

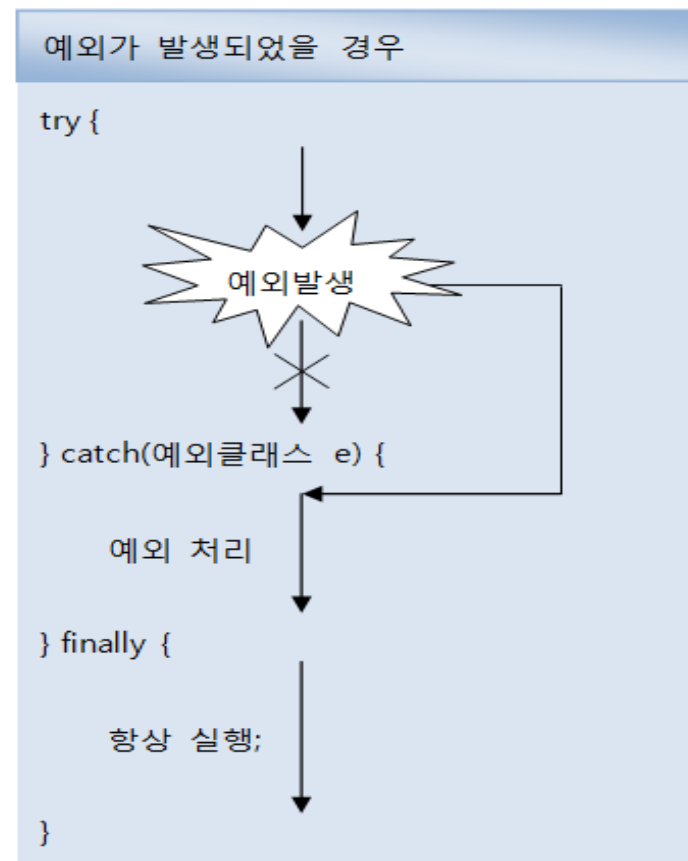
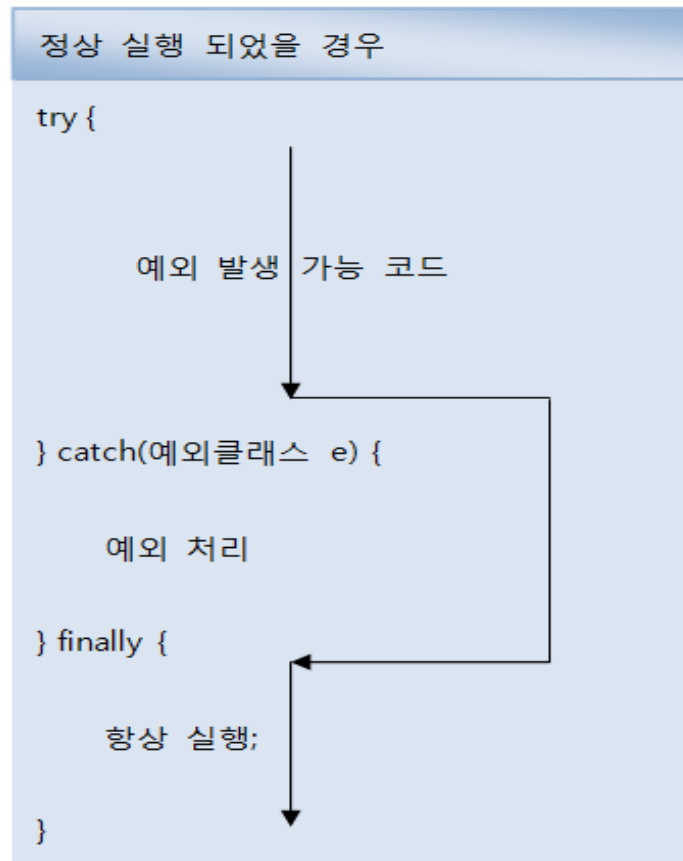
■ 예외 발생 코드

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

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


3절. 예외 처리 코드(try-catch-finally)

- 예외 처리 코드 예외 발생시 프로그램 종료 막고, 정상 실행 유지할 수 있도록 처리
 - 일반 예외: 반드시 작성해야 컴파일 가능
 - 실행 예외: 컴파일러가 체크해주지 않으며 개발자 경험 의해 작성
- try - catch - finally 블록 이용해 예외 처리 코드 작성



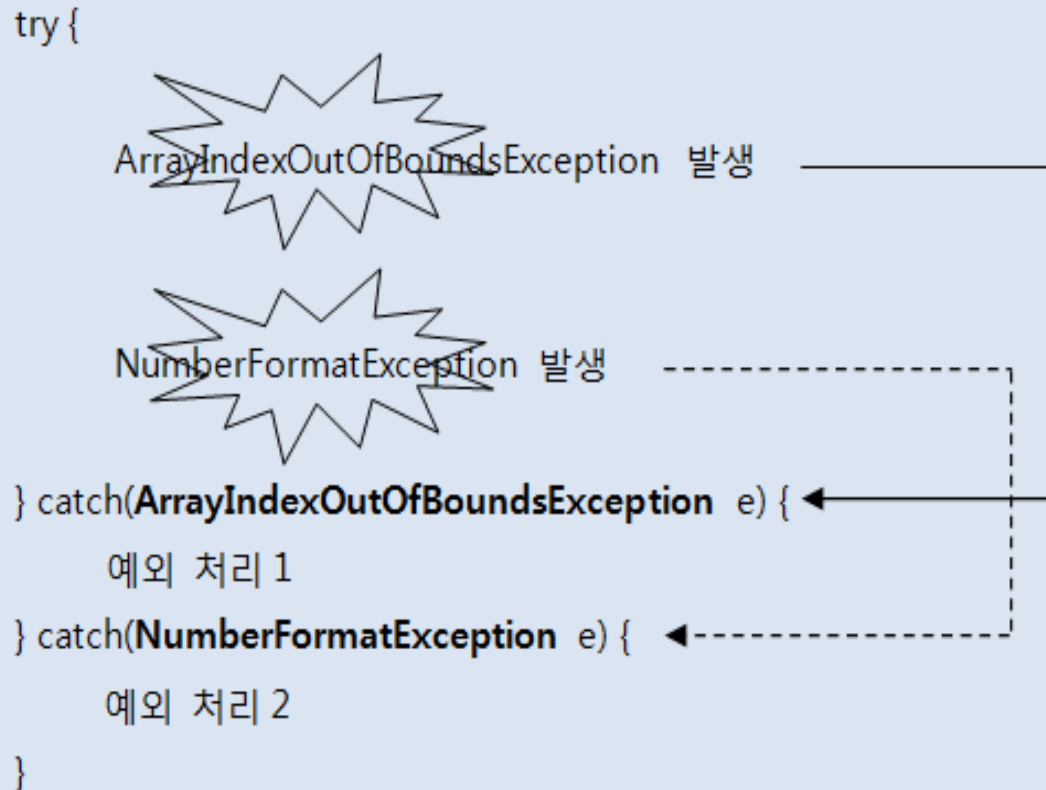
```
public static void main(String[] args) {  
    try {  
        Class clazz = Class.forName("java.lang.String2");  
    } catch(ClassNotFoundException a) {  
        System.out.println("클래스가 존재하지 않습니다.");  
    } finally {  
  
    }  
}
```

```
public static void main(String[] args) {  
    String data1 = null;  
    String data2 = null;  
    try {  
        data1 = args[0];  
        data2 = args[1];  
    } catch (ArrayIndexOutOfBoundsException e) {  
        System.out.println("실행 매개값의 수가 부족합니다.");  
    }  
}
```

```
try {  
    int value1 = Integer.parseInt(data1);  
    int value2 = Integer.parseInt(data2);  
    int result = value1 + value2;  
    System.out.println(data1 + "+" + data2 + "=" + result);  
} catch (NumberFormatException e) {  
    System.out.println("숫자로 변환할 수 없습니다.");  
} finally {  
    System.out.println("다시 실행하세요.");  
}
```

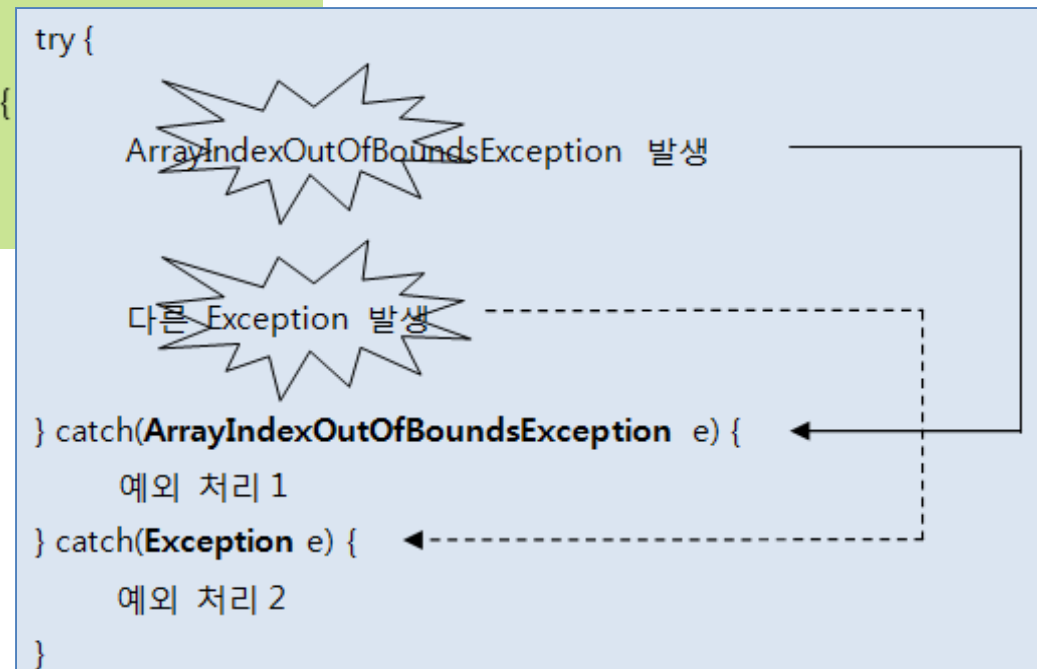
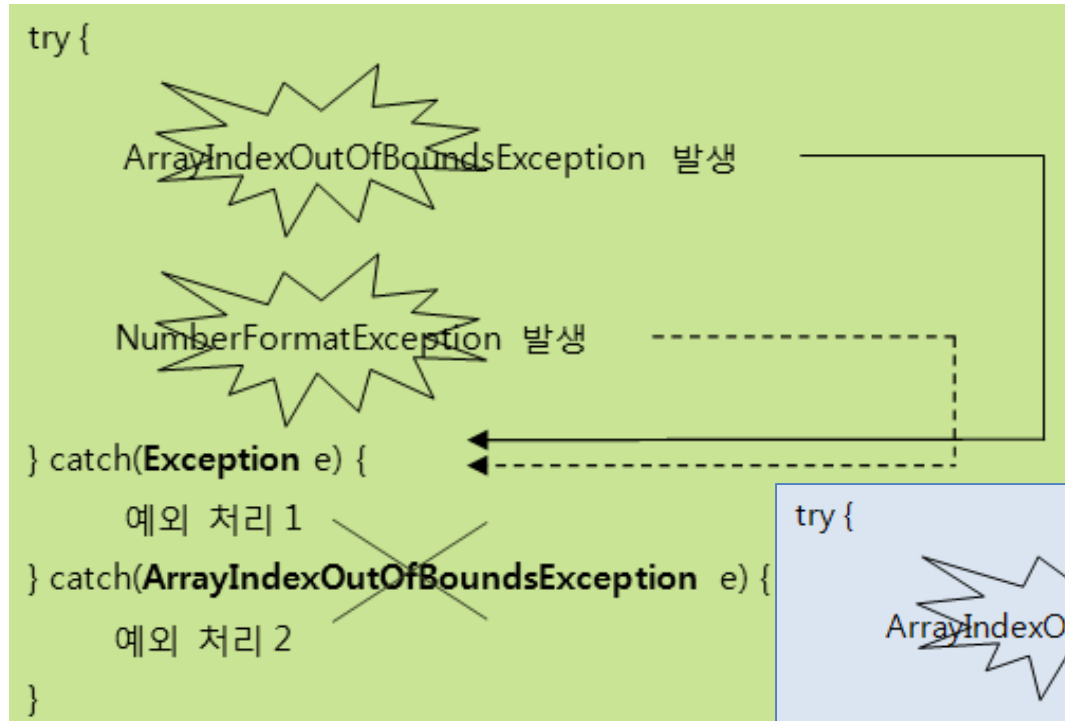
4절. 예외 종류에 따른 처리 코드

- 다중 catch
 - 예외 별로 예외 처리 코드 다르게 구현



4절. 예외 종류에 따른 처리 코드

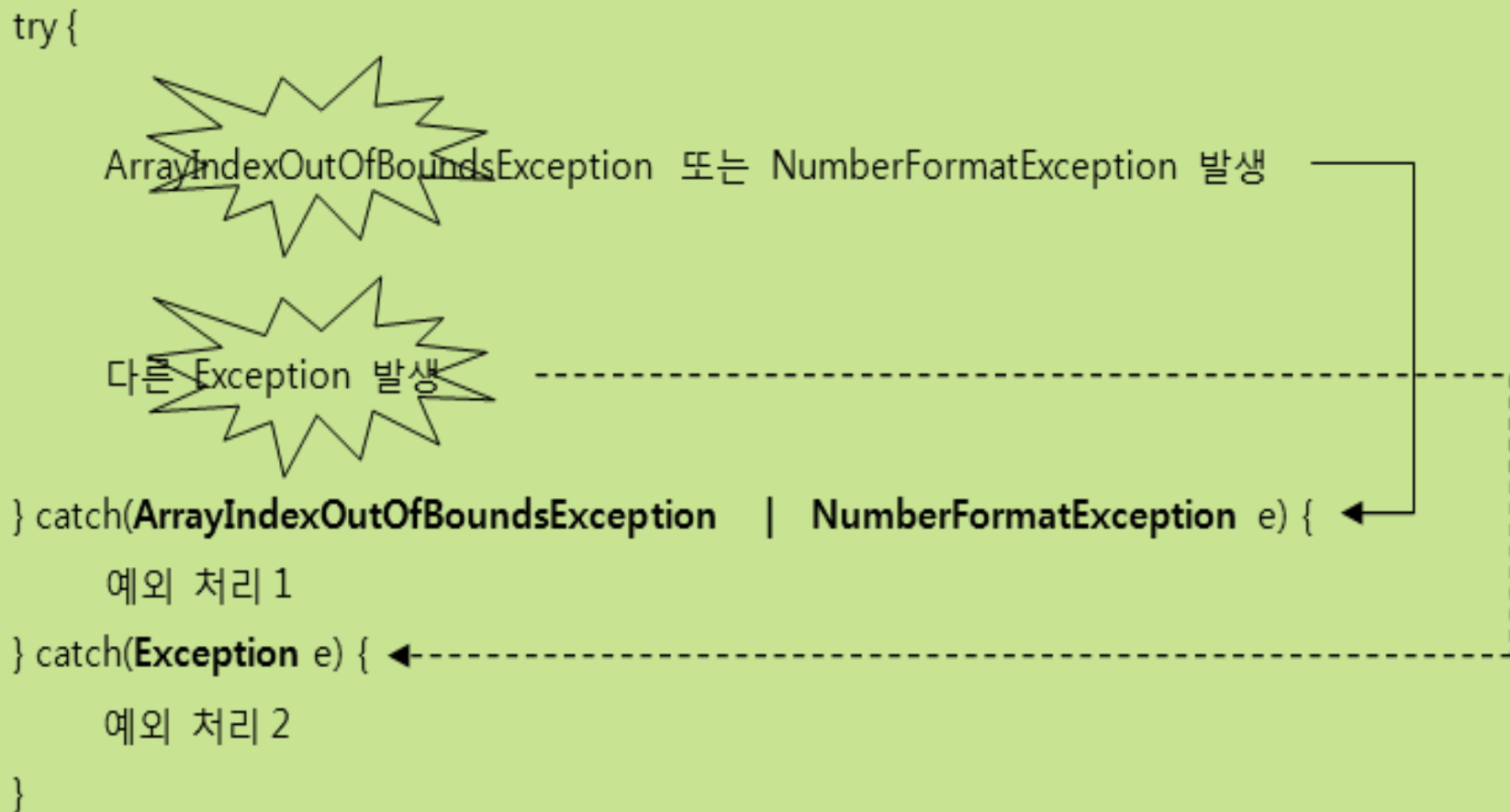
catch 순서



4절. 예외 종류에 따른 처리 코드

■ 멀티(multi) catch

- 자바 7부터는 하나의 catch 블록에서 여러 개의 예외 처리 가능
 - 동일하게 처리하고 싶은 예외를 |로 연결



6절. 예외 떠 넘기기

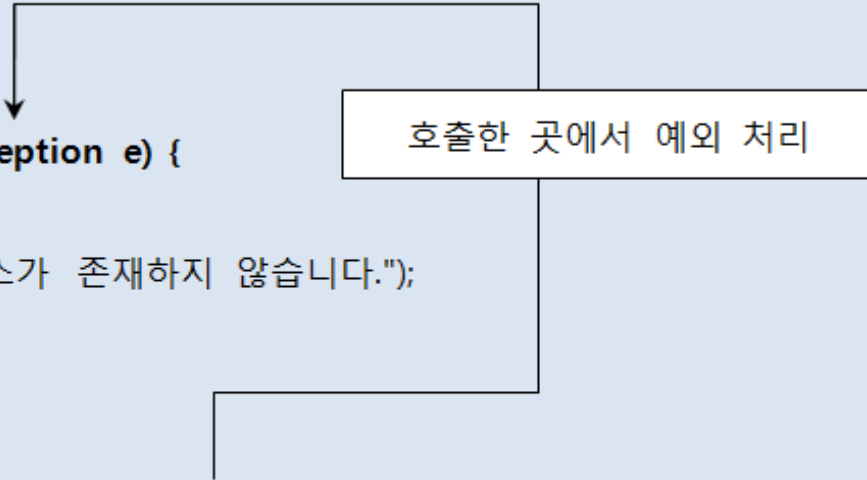
■ throws

■ 메소드 선언부 끝에 작성

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

■ 메소드에서 처리하지 않은 예외를 호출한 곳으로 떠 넘기는 역할

```
public void method1() {  
    try {  
        method2();  
    } catch(ClassNotFoundException e) {  
        //예외 처리 코드  
        System.out.println("클래스가 존재하지 않습니다.");  
    }  
}
```



호출한 곳에서 예외 처리

```
public void method2() throws ClassNotFoundException {  
    Class clazz = Class.forName("java.lang.String2");  
}
```



```
package com.java.eception;

public class MultiCatchEx {
    public void method1() throws Exception {
        throw new Exception();
    }

    public void method2() throws ClassNotFoundException{
        Class clazz = Class.forName("java.Lang.String2");
    }

    public static void main(String[] args) {
        MultiCatchEx ok = new MultiCatchEx();
        ok.method1();
    }
}
```

7절. 사용자 정의 예외와 예외 발생

- 사용자 정의 예외 클래스 선언
 - 자바 표준 API에서 제공하지 않는 예외
 - 애플리케이션 서비스와 관련된 예외
 - Ex) 잔고 부족 예외, 계좌 이체 실패 예외, 회원 가입 실패 예외...
 - 사용자 정의 예외 클래스 선언 방법

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

- 일반 예외 선언(Exception)
- 실행 예외(RuntimeException)

7절. 사용자 정의 예외와 예외 발생

- 필요한 구성요소
 - 필드, 생성자, 메소드
 - 대부분 생성자 선언만을 포함
 - 생성자(2개) 선언
 - 기본 생성자
 - 예외 발생 원인(예외 메시지)을 전달, String 타입의 매개 변수를 갖는 생성자

7절. 사용자 정의 예외와 예외 발생

■ 예외 발생 시키기

■ 코드에서 예외 발생시키는 법

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

```
public void method() throws XXXException {  
    throw new XXXException("메시지");  
}
```

■ 호출된 곳에서 발생한 예외를 처리하도록

8절. 예외 정보 얻기

- getMessage()

- 예외 발생시킬 때 생성자 매개값으로 사용한 메시지 리턴

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

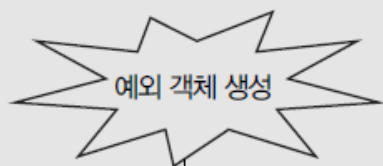
- 원인 세분화하기 위해 예외 코드 포함
- catch() 절에서 활용

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

8절. 예외 정보 얻기

- `printStackTrace()`
 - 예외 발생 코드를 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();  
    }
```

```
public class Account {
    private long balance;
    public Account() {}
    public long getBalance() {
        return balance;
    }

    public void deposit(int money) {
        balance += money;
    }

    public void withdraw(int money) throws
        BalanceInsufficientException{
        if(balance < money) {
            throw new BalanceInsufficientException("잔고부족:" +
                (money - balance) + "모자람");
        }
        balance -= money;
    }
}
```

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



```
public class AccountEx {  
  
    public static void main(String[] args) {  
        Account account = new Account();  
  
        account.deposit(10000);  
        System.out.println("예금액:"+account.getBalance());  
  
        try {  
            account.withdraw(30000);  
        } catch (BalanceInsufficientException e) {  
            System.out.println(e.getMessage());  
            e.printStackTrace();  
        }  
    }  
}
```