예외 처리

514760-1
2016년 가을학기
12/08/2016
박경신

예외와 예외 클래스

- 오류의 종류
  - 에러(Error)
    - 하드웨어의 잘못된 동작 또는 고장으로 인한 오류
    - 에러가 발생되면 JVM실행에 문제가 있으므로 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

예외와 예외 클래스

- 예외의 종류
  - 일반 예외(컴파일 체크 Exception)
    - 컴파일하는 과정에서 예외 처리 코드가 필요한지 검사
    - 예외 처리 코드 없으면 컴파일 오류 발생
  - **실행 예외(RuntimeException)**
    - 예외 처리 코드를 생략하더라도 컴파일이 되는 예외
    - '경험'따라 예외 처리 코드 작성 필요

예외 클래스

- 예외 클래스
  - **Java는 예외를 클래스로 관리**
  - JVM이 프로그램을 실행하는 도중에 예외가 발생하면 해당 예외 클래스로 객체를 생성
    - 예외 처리코드에서 예외 객체를 이용

실행 예외(RuntimeException)

- **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);
}
```

실행시 매개값을 주지 않을 경우 예외 발생

실행 예외(RuntimeException)

- **NumberFormatException**
  - 숫자로 변환될 수 없는 문자가 포함되어 있는 문자열을 숫자로 변경할 경우
    - Integer.parseInt(String s)
    - Double.parseDouble(String s)

```
public class NumberFormatExceptionExample {
    public static void main(String[] args) {
        String data1 = "100";
        String data2 = "a100";

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

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

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;

실행 예외(RuntimeException)

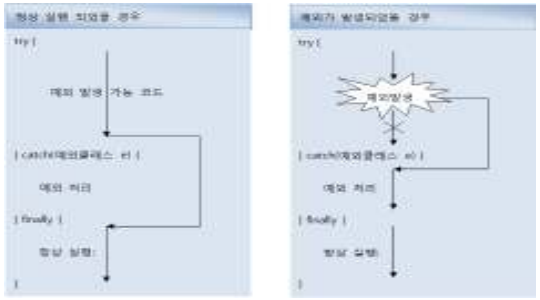
- **ClassCastException**
  - 타입 변환이 되지 않을 경우 발생

```
public class NumberFormatExceptionExample {
    public static void main(String[] args) {
        Dog dog = new Dog();
        changeDog(dog);
        Cat cat = new Cat();
        changeDog(cat);
    }
    public static void changeDog(Animal animal) {
        //if(animal instanceof Dog) {
            Dog dog = (Dog) animal;     //ClassCastException 발생 가능
        //}
    }
}
class Animal {}
class Dog extends Animal {}

class Cat extends Animal {}
```

예외처리 코드 (try-catch-finally)

- 예외처리 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



예외 처리 코드(try-catch-finally)

```
public class TryCatchFinallyRuntimeExceptionExample {
    public static void main(String[] args) {
        String data1 = null;
        String data2 = null;
        try {                                          실행시 매개값을 주지 않을 경우 예외 발생
            data1 = args[0];
            data2 = args[1];
        } catch(ArrayIndexOutOfBoundsException e) {
            System.out.println("실행 매개값의 수가 부족합니다.");
            System.out.println("[실행 방법]");
            System.out.println("java TryCatchFinallyRuntimeExceptionExample num1 num2");
            return;
        }
        try {
            int value1 = Integer.parseInt(data1);   실행시 매개값을 잘못 주었을 경우 예외 발생
            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.");
        }
    }
}
```

예외 종류에 따른 처리 코드

- **다중 catch**
  - 하나의 try 블록 내부에서 다양한 종류의 예외 발생시
  - 각 예외 별로 예외 처리 코드(catch 블록) 다르게 구현
  - 단 하나의 catch 블록만 실행

```
try {
    ArrayIndexOutOfBoundsException 발생
    NumberFormatException 발생
} catch(ArrayIndexOutOfBoundsException e) {
    예외 처리 1
} catch(NumberFormatException e) {
    예외 처리 2
}
```

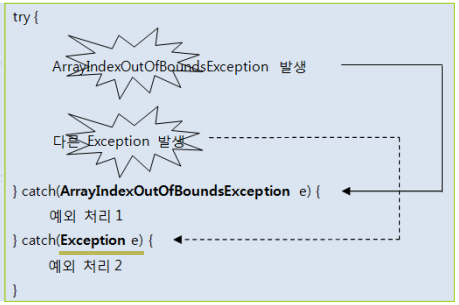
예외 종류에 따른 처리 코드

```
public class CatchByExceptionKindExample {
    public static void main(String[] args) {
        try {
            String data1 = args[0];                 실행시 매개값을 주지 않을 경우 예외 발생
            String data2 = args[1];
            int value1 = Integer.parseInt(data1);
            int value2 = Integer.parseInt(data2);   실행시 매개값을 잘못 주었을 경우 예외 발생
            int result = value1 + value2;
            System.out.println(data1 + "+" + data2 + "=" + result);
        } catch(ArrayIndexOutOfBoundsException e) {
            System.out.println("실행 매개값의 수가 부족합니다.");
            System.out.println("[실행 방법]");
            System.out.println("java CatchByExceptionKindExample num1 num2");
        } catch(NumberFormatException e) {
            System.out.println("숫자로 변환할 수 없습니다.");
        } finally {
            System.out.println("다시 실행하세요.");
        }
    }
}
```

예외 종류에 따른 처리 코드

- 하위 예외는 상위 예외를 상속
  - 하위 예외는 상위 예외 타입도 됨
- **catch** 순서 – 상위 예외가 아래에 위치해야

상위 예외 Exception이 위에 있을 경우



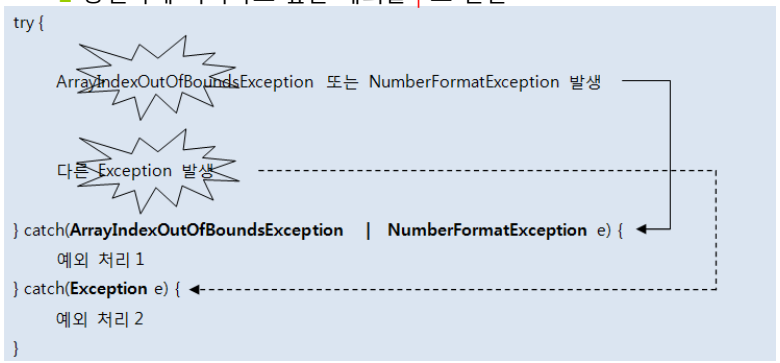
상위 예외 Exception이 아래에 있을 경우

예외 종류에 따른 처리 코드

```
public class CatchOrderExample {
    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(Exception e) {
            System.out.println("실행에 문제가 있습니다.");
        } finally {
            System.out.println("다시 실행하세요.");
        }
    }
}
```

예외 종류에 따른 처리 코드

- **멀티(multi) catch**
  - 자바 7부터는 하나의 catch 블록에서 여러 개의 예외 처리 가능
    - 동일하게 처리하고 싶은 예외를 | 로 연결



예외 종류에 따른 처리 코드

```
public class MultiCatchExample {
    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 | NumberFormatException e) {
            System.out.println("실행 매개값의 수가 부족하거나 숫자로 변환할 수 없습니다..");
        } catch(Exception e) {
            System.out.println("알수 없은 예외 발생");
        } finally {
            System.out.println("다시 실행하세요.");
        }
    }
}
```

실행시 매개값을 주지 않을 경우 예외 발생

실행시 매개값을 잘못 주었을 경우 예외 발생

자동 리소스 닫기

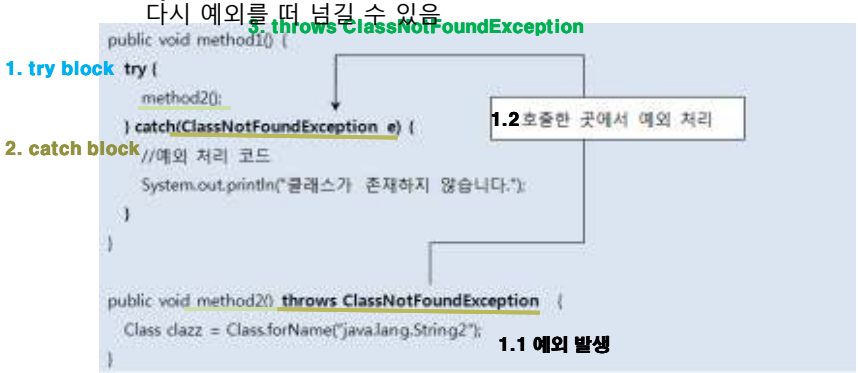
- try-with-resources
  - 예외 발생 여부와 상관 없음
  - 사용했던 리소스 객체의 close() 메소드 호출해 리소스 닫음
  - 리소스 객체
    - 각종 입출력스트림, 서버소켓, 소켓, 각종 채널
    - java.lang.AutoCloseable 인터페이스 구현하고 있어야 함

예외 떠 넘기기(throws)

- **throws**
  - 메소드 선언부 끝에 작성
  - 메소드내에서 처리하지 않은 예외를 메소드 호출한 곳(calling method)으로 떠 넘기는 역할

예외 떠 넘기기(throws)

- **throws**
  - throws 선언된 메소드를 호출하는 메소드(calling method)는
    1. 반드시 try 블록 내에서 호출
    2. catch 블록에서 떠 넘겨 받은 예외를 처리함
    3. try-catch 블록으로 예외처리를 하지 않고 throws 키워드로 자신도 다시 예외를 떠 넘길 수 있음



사용자 정의 예외와 예외 발생

- 사용자 정의 예외(user-defined exception) 클래스 선언
  - 자바 표준 API에서 제공하지 않는 예외
  - 애플리케이션 서비스와 관련된 예외, Application Exception
    - E.g. 잔고 부족 예외, 계좌 이체 실패 예외, 회원 가입 실패 예외....
  - 사용자 정의 예외 클래스 선언 방법
    1. 예외 클래스 상속
       - 일반 예외 : Exception class 상속
       - 실행 예외 : RuntimeException class 상속
    2. 생성자 정의
       - 매개변수 없는 기본 생성자, String 타입의 매개변수를 갖는 생성자

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

사용자 정의 예외와 예외 발생

- 예외 발생 시키기(throw)
  - 코드에서 예외 발생시키는 법 - 예외 객체 생성

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

  - 호출한 곳에서 발생한 예외를 처리하도록

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

사용자 정의 예외와 예외 발생

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

1. Exception class 상속

2. constrctors 생성

```
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;
    }
}
```

4. 호출한 곳에서 발생한 예외를 처리하도록 함

3. 사용자 정의 예외 발생 – 예외 객체 생성

사용자 정의 예외와 예외 발생

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

        Account account = new Account();

        //예금하기
        account.deposit(10000);
        System.out.println("예금액: " + account.getBalance());

        //출금하기
        try {
            account.withdraw(30000);
        } catch(BalanceInsufficientException e) {
            String message = e.getMessage();
            System.out.println(message);
            System.out.println();
            e.printStackTrace();
        }
    }
}
```

5. try block에서 메소드 호출

6. catch block에서 사용자정의 예외 처리

7. 예외 정보 얻기(8절)

예외 정보 얻기

- 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 얻기

- **printStackTrace()**
  - 예외 발생 코드를 추적하여 모두 콘솔에 출력

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

Problems Javadoc Declaration Console

<terminated> AccountExample [Java Application] C:₩Program Files₩Java₩jdk1.8.0_73₩bin₩javaw.exe (

```
예금액 : 10000
잔고부족:20000 모자람

BalanceInsufficientException: 잔고부족:20000 모자람
        at Account.withdraw(Account.java:14)
        at AccountExample.main(AccountExample.java:9)
```