Chapter 10. Exception Handling(예외 처리)

10.1 예외와 예외 클래스

- **예외** : 사용자의 잘못된 조작 또는 개발자의 잘못된 코딩으로 인해 발생하는 오류
  - **일반 예외** : 컴파일러 체크 예외, RunTimeException 상속X
  - **실행 예외** : 컴파일하는 과정에서 예외 처리 코드를 검사하지 않는 예외, RunTimeException 상속O

10.2 실행 예외

10.2.1 NullPointerException

: 객체 참조가 없는 상태, 즉 null 값을 갖는 객체를 접근했을 때 발생

```
package NulllPointerException;

public class NullPointerExceptionExample {
    public static void main(String[] args) {
        String data = null;

        // data 변수는 null 값을 가지고 있기 때문에
        // String 객체를 참조하고 있지 않다.
        System.out.println(data.toString());
    }
}
```

실행 결과(NullPointerException 발생)

```
Exception in thread "main" java.lang.NullPointerException at
NulllPointerException.NullPointerExceptionExample.main(NullPointerExceptionExample.java:6
)
```

10.2.2 ArrayIndexOutOfBoundsException

: 배열에서 인덱스 범위를 초과하여 사용할 경우 발생

```
package array_index_out_of_bounds_exception;
```

```
public class ArrayIndexOutOfBoundsExceptionExample {
    public static void main(String[] args) {
        // 실행 매개값인 args를 2개를 선언하지 않으면
        // 예외 발생하므로 조건물으로 예외 처리
        if(args.length == 2) {
            String data1 = args[0];
            String data2 = args[1];

            System.out.println("args[0]: " + args[0]);
            System.out.println("args[1]: " + args[1]);
        } else {
            System.out.println("[실행 방법]");
            System.out.print("java ArrayIndexOutOfBoundsExceptionExample");
            System.out.print(" 값1 값2");
        }

    }
}
```

실행 결과

```
"C:\Program Files\Java\jdk-10.0.1\bin\java.exe" "-javaagent:C:\Program
Files\JetBrains\IntelliJ IDEA 2018.3\lib\idea_rt.jar=58521:C:\Program
Files\JetBrains\IntelliJ IDEA 2018.3\bin" -Dfile.encoding=UTF-8 -classpath
C:\Users\lenovo\Desktop\Git\TIL\Java\ch10_Exception_Handling\out\production\ch10_Exceptio
n_Handling array_index_out_of_bounds_exception.ArrayIndexOutOfBoundsExceptionExample
[실행 방법]
java ArrayIndexOutOfBoundsExceptionExample 값1 값2
```

10.2.3 NumberFormatException

: 문자열을 숫자로 변경하였을 때 발생하는 예외

```
package number_format_exception;

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

        int value1 = Integer.parseInt(data1);

        // 숫자로 변환될 수 없는 문자가 포함되어 있으므로 예외 발생
        int value2 = Integer.parseInt(data2);

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

```
    }
}
```

실행 결과

```
Exception in thread "main" java.lang.NumberFormatException: For input string: "a100"
    at
java.base/java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)
    at java.base/java.lang.Integer.parseInt(Integer.java:652)
    at java.base/java.lang.Integer.parseInt(Integer.java:770)
    at
number_format_exception.NumberFormatExceptionExample.main(NumberFormatExceptionExample.ja
va:9)
```

10.2.4 ClassCastException

: 타입 변환(Cating)은 상위 클래스와 하위 클래스 간에 발생하고 구현 클래스와 인터페이스 간에 발생한다. 하지만 이러한 관계가 아닌 클래스를 다른 클래스로 강제 타입 변환시 예외가 발생한다.

```
package class_cast_exception;

public class ClassCastExceptionExample {
    public static void main(String[] args) {
        Dog dog = new Dog();
        // Animal animal = dog 로 타입 변환 후
        // change 메소드 내에서 Dog dog = (Dog) animal 로 다시 타입 변환
        changeDog(dog);

        Cat cat = new Cat();
        // Animal animal = cat 으로 타입 변환 후
        // change 메소드 내에서 Dog dog = (Dog) animal 변환 시
        // animal이 Cat을 가리키고 있기 때문에 Cat은 dog의 부모 클래스가 아니므로
        // 예외가 발생한다.
        changeDog(cat);
    }

    public static void changeDog(Animal animal) {
        // if(animal instanceof Dog) {
        Dog dog = (Dog) animal;
        //}
    }
}

class Animal{}
class Dog extends Animal{}
class Cat extends Animal{}
```

실행 결과

```
Exception in thread "main" java.lang.ClassCastException: class_cast_exception.Cat cannot
be cast to class_cast_exception.Dog
    at
class_cast_exception.ClassCastExceptionExample.changeDog(ClassCastExceptionExample.java:1
4)
    at
class_cast_exception.ClassCastExceptionExample.main(ClassCastExceptionExample.java:9)
```

10.3 예외 처리 코드

: 프로그램에서 예외가 발생했을 경우 프로그램의 갑작스러운 종료를 막고, 정상 실행을 유지할 수 있도록 처리하는 코드

- **try-catch-finally 블록** : 생성자 내부와 메소드 내부에서 작성되어 일반 예외와 실행 예외가 발생할 경우 예외 처리를 할 수 있도록 해준다.
  - **try 블록** : 예외 발생 가능 코드가 위치한다.
  - **catch 블록** : 예외 처리 코드가 위치한다.
  - **finally 블록** : 예외 발생 여부와 상관없이 항상 실행할 코드 위치

- **일반 예외 처리 예제**

```
package try_catch_finally;

public class TryCatchFinallyExample {
    public static void main(String[] args) {
        try {
            // java.lang.String2 라는 클래스가 존재하지 않기 때문에
            // catch 에서 ClassNotFoundException 예외를 처리해준다.
            Class clazz = Class.forName("java.lang.String2");
        } catch (ClassNotFoundException e) {
            System.out.println("클래스가 존재하지 않습니다.");
        }
    }
}
```

실행 결과

```
클래스가 존재하지 않습니다.
```

- **실행 예외 처리 예제**

```
package try_catch_finally;
```

```
public class TryCatchFinallyRuntimeExceptionExample {
    public static void main(String[] args) {
        String data1 = null;
        String data2 = null;

        try {
            data1 = args[0];
            // 실행 매개값이 부족하기 때문에 예외가 발생
            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;
        }

        // 실행 매개 값으로 a, 3 을 추가

        try {
            // a를 int로 변환할 수 없기 때문에 예외 발생
            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.");
        }
    }
}
```

실행 매개 값 추가하기 전 실행 결과

```
실행 매개값의 수가 부족합니다.
[실행 방법]
java TryCatchFinallyRuntimeExceptionExample  num1  num2
```

실행 매개 값(a, 3) 추가 후 실행 결과

```
숫자로 변환할 수 없습니다.
다시 실행하세요.
```

실행 매개 값(5, 3) 수정 후 실행 결과

```
5+3=8
다시 실행하세요.
```

> **finally**는 예외가 발생하든 발생하지 않든 무조건 실행된다.

10.4 예외 종류에 따른 처리 코드

10.4.1 다중 catch

: 발생되는 예외별로 예외 처리 코드를 다르게 다중 catch 블록을 작성한다.

```
package multiple_catch;

public class CatchByExceptionKindExample {
    public static void main(String[] args) {
        try {
            String data1 = args[0];
            String data2 = args[1]; // 실행 매개값 부족 예외 발생

            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.");
            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.");
        }
    }
}
```

실행 결과

```
실행 매개값의 수가 부족합니다.
[실행 방법]
java CatchByExceptionKindExample  num1  num2
다시 실행하세요.
```

10.4.2 catch 순서

: 다중 catch 블록을 작성할 때 주의할 점은 **상위 예외 클래스가 하위 예외 클래스보다 아래쪽에 위치해야 한다.

- **예시**

```
try {
    ArrayIndexOutOfBoundsException 발생

    NumberFormatException 발생
} catch (Exception e) {     // 위의 예외 두 개 모두 이 예외에 걸린다.
    예외 처리1
} catch (ArrayIndexOutOfBoundsException e) {
    예외 처리2
}
```

수정한 코드

```
try {
    ArrayIndexOutOfBoundsException 발생

    NumberFormatException 발생
} catch (ArrayIndexOutOfBoundsException e) {    // ArrayIndex 예외 처리
    예외 처리1
} catch (Exception e) { // 나머지 예외들 처리
    예외 처리2
}
```

- **예제**

```
package catch_order;

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) {  // 하위 예외 클래스
            System.out.println("실행 매개값의 수가 부족");
        } catch (Exception e) {                       // 상위 예외 클래스
            System.out.println("실행에 문제가 있음");
        } finally {
            System.out.println("다시 실행하세요.");
        }
    }
}
```

실행 결과

```
실행 매개값의 수가 부족
다시 실행하세요.
```

10.4.3 멀티 catch

: 하나의 catch 블록에서 여러 개의 예외를 처리할 수 있는 멀티(multi) catch 기능

- **예제**

```
package multi_catch;

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) {  // 멀티 catch로 여러 개의 예외 처리
            System.out.println("실행 매개값의 수가 부족하거나 숫자로 변환할 수 없다.");
        } catch (Exception e) {
            System.out.println("알수 없는 예외 발생");
        } finally {
            System.out.println("다시 실행하세요.");
        }
    }
}
```

실행 결과

```
실행 매개값의 수가 부족하거나 숫자로 변환할 수 없다.
다시 실행하세요.
```

10.5 자동 리소스 닫기(질문)

: try-with-resources를 사용하면 예외 발생 여부와 상관없이 사용했던 리소스(데이터를 읽고 쓰는 객체) 객체(각종 입출력 스트림, 서버 소켓, 소켓, 각종 채널)의 close() 메소드를 호출해서 안전하게 리소스를 닫아준다.

- **예제**

  FileInputStream.java(**AutoCloseable 구현 클래스**)

```
package auto_resources_close;

// AutoCloseable 인터페이스의 구현 클래스
public class FileInputStream implements AutoCloseable{
    private String file;

    // 생산자
    public FileInputStream(String file) {
        this.file = file;
    }

    public void read() {
        System.out.println(file + " 을 읽습니다.");
    }

    // 파일을 닫아주는 메소드를 오버라이딩
    @Override
    public void close() throws Exception {
        System.out.println(file + " 을 닫습니다.");
    }
}
```

  TryWithResourceExample.java(**AutoCloseable 구현 클래스**)

```
package auto_resources_close;

public class TryWithResourceExample {
    public static void main(String[] args) {
        try (FileInputStream fis = new FileInputStream("file.txt")) {
            fis.read();
            throw new Exception();  // 강제적으로 예외 발생시킨다.
        } catch (Exception e) {
            System.out.println("예외 처리 코드가 실행되었습니다.");
        }
    }
}
```

  실행 결과

```
file.txt 을 읽습니다.
file.txt 을 닫습니다.
예외 처리 코드가 실행되었습니다.
```

10.6 예외 떠넘기기

- **throw 키워드** : 메소드 선언부 끝에 작성되어 메소드에서 처리하지 않은 예외를 호출한 곳으로 떠넘기는 역할을 한다.
- **예시**

```
리턴타입 메소드명(매개변수, ...) throws 예외클래스1, 예외클래스2, ... {
}

// 모든 예외를 떠넘기는 방법
리턴타입 메소드명(매개변수, ...) throws Exception{
}
```

- **예제**

```
package exception_throws;

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

    // 호출한 곳으로 예외를 떠넘긴다.
    public static void findClass() throws ClassNotFoundException {
        Class clazz = Class.forName("java.lang.String2");
    }
}
```

실행 결과

```
클래스가 존재하지 않습니다.
```

10.7 사용자 정의 예외와 예외 발생

- **애플리케이션 예외(Application Exception)** : 개발자가 직접 정의해서 만드는 것.(사용자 정의 예외)

10.7.1 사용자 정의 예외 클래스 선언

: 컴파일러가 체크하는 일반 예외로 선언할 수도 있고, 컴파일러가 체크하지 않는 실행 예외로 선언할 수도 있다. 일반 예외로 선언할 경우 **Exception**을 상속하면 되고, 실행 예외로 선언할 경우에는 **RuntimeException**을 상속하면 된다.

- 사용자 정의 예외 클래스 이름은 **Exception**으로 끝나는 것이 좋다.
- 대부분 생성자 선언만 사용자 정의 예외로 사용
  - ex) 예외 발생 원인을 전달하기 위한 String 타입의 매개 변수를 갖는 생성자.

- **예제**

  BalanceInsufficientException.java(**사용자 정의 예외 클래스**)

```
package application_exception;

public class BalanceInsufficientException extends Exception{
    public BalanceInsufficientException(){}

    // 상위 클래스의 생성자를 호출하여 예외 메시지를 넘겨준다.
    public BalanceInsufficientException(String message) {
        super(message);
    }
}
```

10.7.2 예외 발생시키기

: 사용자 정의 예외 또는 자바 표준 예외를 코드에서 발생시키는 방법.

- Account.java(**사용자 정의 예외 발생시키기**)

```
package exception_generate;

import application_exception.BalanceInsufficientException;

public class Account {
    private long balance;

    public Account() {}

    public long getBalance() {
        return balance;
    }

    public void deposit(int money) {
```

```
        balance += money;
    }

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

> 자신을 호출한 곳에서 예외를 처리하도록 throws 키워드로 예외를 떠넘긴다.

10.8 예외 정보 얻기

: try 블록에서 예외가 발생되면 예외 객체는 catch 블록의 매개 변수에서 참조하게 되므로 매개 변수를 이용하면 예외 객체의 정보를 알 수 있다.

- **예제**

  AccountExample.java(**사용자 정의 예외 발생시키기**)

```
package exception_getting_information;

import application_exception.BalanceInsufficientException;
import exception_generate.Account;

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) {  // 예외 메시지 얻기
            // 예외 메시지를 getMessage() 메소드의 리턴값으로 얻는다.
            String message = e.getMessage();
            System.out.println(message);
            System.out.println();

            // 예외 발생 코드를 추적해서 모두 콘솔에 출력
            e.printStackTrace();
        }
```

```
        }
    }
```

실행 결과

```
예금액: 10000
잔고부족:20000 모자람

application_exception.BalanceInsufficientException: 잔고부족:20000 모자람
    at exception_generate.Account.withdraw(Account.java:22)
    at exception_getting_information.AccountExample.main(AccountExample.java:16)
```

printStackTrace() 메소드

: 예외에 대한 정보를 가져올 수 있다.예외 발생 코드를 추적해서 모두 콘솔에 출력시킨다.

확인문제

1. 예외에 대한 설명 중 틀린 것은 무엇입니까?
    1. 예외는 사용자의 잘못된 조작, 개발자의 잘못된 코딩으로 인한 프로그램 오류를 말한다.
    2. RuntimeException의 하위 예외는 컴파일러가 예외 처리 코드를 체크하지 않는다.
    3. 예외는 try-catch 블록을 사용해서 처리된다.
    4. **자바 표준 예외만 프로그램에서 처리할 수 있다. (X, 사용자 정의 예외도 프로그램에서 처리할 수 있다.)**

2. try-catch-finally 블록에 대한 설명 중 틀린 것은 무엇입니까?
    1. try {} 블록에는 예외가 발생할 수 있는 코드를 작성한다.
    2. catch {} 블록은 try {} 블록에서 발생한 예외를 처리하는 블록이다.
    3. **try {} 블록에서 return문을 사용하면 finally {} 블록은 실행되지 않는다. (X, finally 블록은 반드시 실행된다.)**
    4. catch {} 블록은 예외의 종류별로 여러 개를 작성할 수 있다.

3. throws에 대한 설명으로 틀린 것은 무엇입니까?
    1. 생성자나 메소드의 선언 끝 부분에 사용되어 내부에서 발생된 예외를 떠넘긴다.
    2. throws 뒤에는 떠넘겨야 할 예외를 쉼표(,)로 구분해서 기술한다.
    3. 모든 예외를 떠넘기기 위해 간단하게 throws Exception으로 작성할 수 있다.
    4. **새로운 예외를 발생시키기 위해 사용된다.(X)**

4. throw에 대한 설명으로 틀린 것은 무엇입니까?
    1. 예외를 최초로 발생시키는 코드이다.
    2. **예외를 호출한 곳으로 떠넘기기 위해 메소드 선언부에 작성된다.(X)**

3. throw로 발생된 예외는 일반적으로 생성자나 메소드 선언부에 throws로 떠넘겨진다.
4. throw 키워드 뒤에는 예외 객체 생성 코드가 온다.

5. 다음과 같은 메소드가 있을 때 예외를 잘못 처리한 것은 무엇입니까?

```
public void method1() throws NumberFormatException, ClassNotFoundException {...}
```

1. try {method1();} catch (Exception e) {}
2. void method2() throws Exception {method1();}
3. **try {method1();} catch (Exception e) {} catch (ClassNotFoundException e) {} (X, 하위 예외 클래스를 상위 클래스보다 더 앞에 선언해야 한다.)**
4. try {method1();} catch (ClassNotFoundException e) {} catch (NumberFormatException e) {}

6. 다음 코드가 실행되었을 때 출력 결과는 무엇입니까?

TryCatchFinallyExample.java

```
public class TryCatchFinallyExample {
    public static void main(String[] args) {
        String[] strArray = {"10", "2a"};
        int value = 0;
        for(int i=0; i<=2; i++) {
            try {
                value = Integer.parseInt(strArray[i]);
            } catch (ArrayIndexOutOfBoundsException e) {
                System.out.println("인덱스를 초과했음");
            } catch (NumberFormatException e) {
                System.out.println("숫자로 변환할 수 없음");
            } finally {
                System.out.println(value);
            }
        }
    }
}
```

실행 결과

```
10
숫자로 변환할 수 없음
10
인덱스를 초과했음
10
```

7. 로그인 기능을 Member 클래스의 login() 메소드에서 구현하려고 합니다. 존재하지 않는 ID를 입력했을 경우 NotExistIDException을 발생시키고, 잘못된 패스워드를 입력했을 경우 WrongPasswordException을 발생시키려고 합니다. LoginExample의 실행 결과를 보고 빈칸을 채워보세요.

NotExistIDException.java(사용자 정의 예외 클래스)

```
class NotExistIDException extends Exception {
    public NotExistIDException() {}
    public NotExistIDException(String message) {
        // 상위 클래스인 Exception의 생산자에게 예외 메시지를 전달
        super("아이디가 존재하지 않습니다.");
    }
}
```

WrongPasswordException.java(사용자 정의 예외 클래스)

```
class WrongPasswordException extends Exception {
    public WrongPasswordException() {}
    public WrongPasswordException(String message) {
        // 상위 클래스인 Exception의 생산자에게 예외 메시지를 전달
        super("패스워드가 틀립니다.");
    }
}
```

LoginExample.java(예외 발생 및 떠넘기기)

```
class LoginExample {
    public static void main(String[] args) {
        try {
            login("white", "12345");
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }

        try {
            login("blue", "54321");
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }

    public static void login(String id, String password) throws NotExistIDException,
WrongPasswordException {

        // id가 "blue"가 아니라면 NotExistIDException 발생시킴
        if(!id.equals("blue")) {
            throw new NotExistIDException("blue");
        }

        // password가 "12345"가 아니라면 WrongPasswordException 발생시킴
        if(!password.equals("12345")) {
```

```
                throw new WrongPasswordException("12345");
            }
        }
    }
```