Chapter 10. 라이브러리와 모듈

라이브러리(library) 는 프로그램 개발 시 활용할 수 있는 클래스와 인터페이스들을 모아놓은 것을 말한다. 일반적으로 JAR(Java ARchive)압축 파일(~.jar) 형태로 존재한다. JAR 파일에는 클래스와 인터페이스의 바이트코드 파일(~.class)들이 압축되어 있다. 특정 클래스와 인터페이스가 여러 응용프로그램을 개발할 때 공통으로 자주 사용된다면 JAR 파일로 압축해서 라이브러리로 관리하는 것이 좋다.

Java 9부터 지원하는 *모듈(module)* 은 패키지 관리 기능까지 포함된 라이브러리다. 일반 라이브러리는 내부에 포함된 모든 패키지에 외부 프로그램에서의 접근이 가능하지만 모듈은 일부 패키지를 은닉하여 접근할 수 없게 끔 할 수 있다. 또한 자신이 실행할 때 필요로 하는 의존 모듈을 모듈 기술자(module-info.java)에 기술할 수 있기 때문에 모듈 간의 의존 관계를 쉽게 파악할 수 있다.

Chapter 11. 예외처리

11.1 예외와 예외 클래스

예외(exception)란 잘못된 사용 또는 코딩으로 인한 오류로, 예외가 발생하면 프로그램은 곧바로 종료된다는 점에서 에러(error)와 동일하지만 예외 처리를 통해 계속 실행 상태를 유지할 수 있다.

- 일반 예외(Exception)
  - 컴파일러가 예외 처리 코드 여부를 검사하는 예외를 말한다.
- 실행 예외(Runtime Exception)
  - 컴파일러가 예외 처리 코드 여부를 검사하지 않는 예외를 말한다.

11.2 예외 처리 코드

예외가 발생했을 때 프로그램의 갑작스러운 종료를 막고 정상 실행을 유지할 수 있도록 처리하는 코드를 **예외 처리 코드**라고 한다. 예외 처리 코드는 `try-catch-finally` 블록으로 구성된다. 해당 블록은 생성자 내부와 메소드 내부에서 작성된다.
`try` 블록에서 작성한 코드가 예외 없이 정상 실행되면 `catch` 블록은 실행되지 않고 `finally` 블록이 실행된다. 그러나 `try` 블록에서 예외가 발생하면 `catch` 블록이 실행되고 연이어 `finally` 블록이 실행된다. 예외 발생 여부와 상관없이, 심지어 `return` 문을 사용하더라도 `finally` 블록은 항상 실행된다. 이는 옵션으로 생략 가능하다.

ExceptionHandlingExample2.java

```
package ch11.sec02.exam01;

public class ExceptionHandlingExample2 {
    public static void printLength(String data) {
        try {
            int result = data.length();
            System.out.println("문자 수: " + result);
        } catch(NullPointerException e) {
            System.out.println(e.getMessage()); //①
            //System.out.println(e.toString()); //②
            //e.printStackTrace(); //③
        } finally {
```

```
            System.out.println("[마무리 실행]\n");
        }
    }

    public static void main(String[] args) {
        System.out.println("[프로그램 시작]\n");
        printLength("ThisIsJava");
        printLength(null);
        System.out.println("[프로그램 종료]");
    }
}
```

결과

```
[프로그램 시작]

문자 수: 10
[마무리 실행]

Cannot invoke "String.length()" because "data" is null
[마무리 실행]

[프로그램 종료]
```

① `e.getMessage()` : 예외가 발생한 이유만 리턴
② `e.toString()` : 이유와 예외의 종류도 리턴
③ `e.printStackTrace()` : 예외가 어디서 발생했는지 추적한 내용까지 출력

11.3 예외 종류에 따른 처리

`try` 블록에는 다양한 종류의 예외가 발생할 수 있는데, 다중 `catch`를 사용하면 발생하는 예외에 따라 예외 처리 코드를 다르게 작성할 수 있다. `catch` 블록은 여러 개라 할지라도 단 하나만 실행된다.

처리해야 할 예외 클래스들이 상속 관계에 있을 때는 하위 클래스 `catch` 블록을 먼저 작성하고 상위 클래스 `catch` 블록을 나중에 작성해야 한다.

ExceptionHandlingExample.java

```
package ch11.sec03.exam02;

public class ExceptionHandlingExample {
    public static void main(String[] args) {
        String[] array = {"100", "1oo"};

        for(int i=0; i<=array.length; i++) {
            try {
                int value = Integer.parseInt(array[i]);
                System.out.println("array[" + i + "]: " + value);
```

```
            } catch(ArrayIndexOutOfBoundsException e) {
                System.out.println("배열 인덱스가 초과됨: " + e.getMessage());
            } catch(Exception e) {      // 상위 예외 클래스는 아래쪽에 작성
                System.out.println("실행에 문제가 있습니다.");
            }
        }
    }
}
```

두 개 이상의 예외를 하나의 `catch` 블록으로 동일하게 예외 처리하고 싶을 땐 예외 클래스를 `|` 기호로 연결하면 된다.

11.4 리소스 자동 닫기

리소스(resource) 란 데이터를 제공하는 객체를 말한다. 리소스를 사용하다 예외가 발생될 경우에도 안전하게 닫는 것이 중요한데, 그렇지 않으면 리소스가 불안정한 상태로 남아있게 되기 때문이다. `try-with-resources` 블록을 사용하면 예외 발생 여부와 상관없이 리소스를 자동으로 닫아준다. `try` 괄호에 리소스를 여는 코드를 작성하면 `try` 블록이 정상적으로 실행을 완료했거나 도중에 예외가 발생하면 자동으로 리소스의 `close()` 메소드가 호출된다. `try-with-resources` 블록을 사용하기 위해서는 `java.lang.AutoCloseable` 인터페이스를 구현해서 `AutoCloseable` 인터페이스의 `close()` 메소드를 재정의해야 한다.

MyResource.java

```
package ch11.sec04;

public class MyResource implements AutoCloseable {
    private String name;

    public MyResource(String name) {
        this.name = name;
        System.out.println("[MyResource(" + name + ") 열기]");
    }

    public String read1() {
        System.out.println("[MyResource(" + name + ") 읽기]");
        return "100";
    }

    public String read2() {
        System.out.println("[MyResource(" + name + ") 읽기]");
        return "abc";
    }

    @Override
    public void close() throws Exception {
        System.out.println("[MyResource(" + name + ") 닫기]");
    }
}
```

TryWithResourceExample.java

```
package ch11.sec04;

public class TryWithResourceExample {
    public static void main(String[] args) {
        try (MyResource res = new MyResource("A")) {
            String data = res.read1();
            int value = Integer.parseInt(data);
        } catch(Exception e) {
            System.out.println("예외 처리: " + e.getMessage());
        }

        System.out.println();

        try (MyResource res = new MyResource("A")) {
            String data = res.read2();
            //NumberFormatException 발생
            int value = Integer.parseInt(data);
        } catch(Exception e) {
            System.out.println("예외 처리: " + e.getMessage());
        }

        System.out.println();

        /*try (
            MyResource res1 = new MyResource("A");
            MyResource res2 = new MyResource("B")
        ) {
            String data1 = res1.read1();
            String data2 = res2.read1();
        } catch(Exception e) {
            System.out.println("예외 처리: " + e.getMessage());
        }*/

        MyResource res1 = new MyResource("A");
        MyResource res2 = new MyResource("B");
        try (res1; res2) {
            String data1 = res1.read1();
            String data2 = res2.read1();
        } catch(Exception e) {
            System.out.println("예외 처리: " + e.getMessage());
        }
    }
}
```

11.5 예외 떠넘기기

메소드 내부에서 예외가 발생할 때 `try-catch` 블록으로 예외를 처리하는 것이 기본이지만, 메소드를 호출한 곳으로 에외를 떠넘길 수도 있다. 이때 사용하는 키워드가 `throws`다. `throws` 키워드가 붙어 있는 메소드에서

해당 예외를 처리하지 않고 떠넘겼기 때문에 이 메소드를 호출하는 곳에서 예외를 받아 처리해야 한다.

ThrowsExample1.java

```
package ch11.sec05;

public class ThrowsExample1 {
    public static void main(String[] args) {
        try {
            findClass();
        } catch(ClassNotFoundException e) {
            System.out.println("예외 처리: " + e.toString());
        }
    }

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

`ClassNotFoundException`을 `throws`하는 `findClass()`의 예외를 `main()`에서 호출할 때 처리하고 있다.

나열해야 할 예외 클래스가 많을 경우에는 `throws Exception` 또는 `throws Throwable` 만으로 모든 예외를 간단히 떠넘길 수도 있다. `main()` 메소드에서도 `throws` 키워드를 사용해서 예외를 떠넘길 수 있는데, 결국 JVM이 최종적으로 예외 처리를 하게 된다. JVM은 예외의 내용을 콘솔에 출력하는 것으로 예외 처리를 한다.

ThrowsExample2.java

```
package ch11.sec05;

public class ThrowsExample2 {
    public static void main(String[] args) throws Exception {
        findClass();
    }

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

결과

```
Exception in thread "main" java.lang.ClassNotFoundException: java.lang.String2
        at
java.base/jdk.internal.loader.BuiltinClassLoader.loadClass(BuiltinClassLoader.java
:641)
        at
java.base/jdk.internal.loader.ClassLoaders$AppClassLoader.loadClass(ClassLoaders.j
```

```
ava:188)
        at java.base/java.lang.ClassLoader.loadClass(ClassLoader.java:526)
        at java.base/java.lang.Class.forName0(Native Method)
        at java.base/java.lang.Class.forName(Class.java:421)
        at java.base/java.lang.Class.forName(Class.java:412)
        at ch11.sec05.ThrowsExample2.findClass(ThrowsExample2.java:9)
        at ch11.sec05.ThrowsExample2.main(ThrowsExample2.java:5)
```

11.6 사용자 정의 예외

은행의 뱅킹 프로그램에서의 잔고 부족 예외와 같이 표준 라이브러리에는 존재하지 않기 때문에 직접 예외 클래스를 정의해서 사용해야 하는 것을 *사용자 정의 예외*라고 한다.

사용자 정의 예외

컴파일러가 체크하는 **일반 예외**로 선언할 수도 있고, 컴파일러가 체크하지 않는 **실행 예외**로 선언할 수도 있다. 통상적으로 일반 예외는 `Exception`의 자식 클래스로 선언하고, 실행 예외는 `RuntimeException`의 자식 클래스로 선언한다.

사용자 정의 예외 클래스에는 기본 생성자와 예외 메시지를 입력받는 생성자를 선언해준다. 다음은 잔고 부족 예외를 사용자 정의 예외 ㅋ르래스로 선언한 것이다.

InsufficientException.java

```java
package ch11.sec06;

public class InsufficientException extends Exception {  // 일반 예외로 선언
    public InsufficientException() {
    }

    public InsufficientException(String message) {
        super(message);
    }
}   // 두 개의 생성자 선언
```

예외 발생 시키기

자바에서 제공하는 표준 예외뿐만 아니라 사용자 정의 예외를 직접 코드에서 발생시키려면 `throw` 키워드와 함께 예외 객체를 제공하면 된다. 예외의 원인에 해당하는 메시지를 제공하고 싶다면 생성자 매개값으로 전달한다. `throw`된 예외는 직접 `try-catch` 블록으로 에외를 처리할 수도 있지만 대부분은 메소드를 호출한 곳에서 예외를 처리하도록 `throws` 키워드로 예외를 떠넘긴다.

Account.java

```java
package ch11.sec06;

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 InsufficientException { // 호출한 곳으로
예외 떠넘김
        if(balance < money) {
            throw new InsufficientException("잔고 부족: "+(money-balance)+" 모자
람"); // 예외 발생
        }
        balance -= money;
    }
}
```

AccountExample.java

```
package ch11.sec06;

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 {       // 예외 처리 코드와 함께 withdraw() 메소드 호출
            account.withdraw(30000);
        } catch(InsufficientException e) {
            String message = e.getMessage();
            System.out.println(message);
        }
    }
}
```

결과

```
예금액: 10000
잔고 부족: 20000 모자람
```