Chapter 08 자바 API 패키지, 예외 처리, 모듈

8.1 패키지란?

패키지의 개념

- 패키지: 관련 있는 클래스들을 하나로 묶은 것. lang(<- String), util(<-ArrayList), awt, javax, swing, net, io, sql

Package of JAVA	Description
내장 패키지	자바에서 기본적으로 제공하는 패키지
사용자 정의 패키지	사용자가 정의하는 패키지

왜 패키지가 필요할까?

1. 서로 관련된 클래스들을 하나의 단위로 모아 쉽게 유지 관리할 수 있다.
2. 패키지 안에 있는 클래스들은 패키지 안에서만 사용할 수 있도록 선언하여 캡슐화를 구현할 수 있다.
3. 패키지가 다르면 동일한 클래스의 이름을 사용할 수 있다.(이름 공간)

8.2 패키지 선언하기

- 패키지 선언문은 반드시 소스 파일의 첫 번째 문장이어야 한다.
- 여러 개의 소스 파일에 동일한 패키지 선언문을 넣을 수 있다.
- 패키지는 계층 구조를 가질 수 있다. 패키지 안에 패키지를 넣을 수 있다.

```
package graphics; // Circle Class를 graphics package에 속하게 하는 코드

public class Circle {
    double radius;
}
```

```
package pkg1.pkg2; // pkg1 안에 pkg2를 넣을 수 있다.
```

이클립스에서 패키지 만들기

소스 파일의 위치

클래스 파일의 위치

8.3 패키지 사용하기

```
graphics.Rectangle myRect = new graphics.Rectangle(); // 1. 완전한 이름으로 참조
- 패키지 이름을 클래스 앞에 적는다.
```

```
import graphics.Rectangle; // 2. 패키지 안에서 우리가 원하는 클래스만을 포함한다.
Rectangle myRect = new Rectangle();

import graphics.*; // 3. 패키지 안의 모든 클래스를 포함한다.
Rectangle myRect = new Rectangle();
```

계층 구조의 패키지 포함하기

```
import java.awt.*; // java.awt 패키지의 모든 클래스와 인터페이스를 가져온다. 하지만 하위 패키지의 클래스는 가져오지 않는다.
import java.awt.font.*;
```

정적 import 문장

- 일반적으로 클래스 안에 정의된 정적 상수나 정적 메소드를 사용하는 경우에 클래스 이름을 앞에 적어야 한다.
- 하지만 정적 import 문장을 사용하면 클래스의 이름을 생략할 수 있다.

```
double r = Math.cos(Math.PI * theta);

import static java.lang.Math.*;
double r = cos(PI * theta);
```

예제 8-1

```

```

8.4 클래스 파일은 언제 로드될까?

언제 클래스 파일이 로드되는가?

- Lazy Class Loading(지연 클래스 로드): 요청 시 클래스 파일은 JVM에 로드된다. -> JVM의 메모리를 절약할 수 있다.
- 시작 시 애플리케이션 코드를 구성하는 파일은 로드된다.

자바 가상 머신이 클래스를 찾는 순서

1. 부트스트랩 클래스: 자바 플랫폼을 구성하는 핵심적인 클래스
2. 확장 클래스: 자바 확장 메커니즘을 사용하는 클래스
3. 사용자 클래스: 확장 메커니즘을 활용하지 않는 개발자 및 타사에 의해 정의된 클래스

CLASSPATH

- 사용자가 직접 작성하였거나 외부에서 다운로드 받은 클래스를 찾기 위하여 가상 머신이 살펴보는 디렉토리들을 모아 둔 경로

클래스 경로를 지정하는 3가지 방법

1. 현재 디렉토리
2. CLASSPATH에 설정된 환경 변수 설정
3. -classpath 옵션

JAR 압축 파일

이클립스를 사용하는 경우

JAR 형태의 라이브러리 사용하기

8.5 자바 API 패키지

8.6 Object 클래스

- java.lang 패키지에 들어 있으며, 자바 클래스 계층 구조에서 맨 위에 위치하는 클래스

Object 클래스 안에 정의된 메소드

이름	기능
public boolean equals(Object obj)	obj가 이 객체와 같은지 검사한다.
public String toString()	객체의 문자열 표현을 반환한다.
protected Object clone()	객체 자신의 복사본을 생성하여 반환한다.
public int hashCode()	객체에 대한 해쉬 코드를 반환한다.
protected void finalize()	가비지 컬렉터에 의하여 호출된다.
public final Class getClass()	객체의 클래스 정보를 반환한다.

getClass() 메소드

- 객체가 어떤 클래스로 생성되었는지에 대한 정보를 반환한다.
- reflection(리플랙션): 객체가 자신을 만든 클래스에 대하여 물어보는 것

toString() 클래스

- 객체의 상태를 몇 줄의 문자열로 요약하여 반환하는 함수
- 일반적으로 재정의하여야 하는 메소드

equals() 메소드

- 동등 비교
- '==' : 참조값 비교
- equals() : 내용 비교

8.7 랩퍼 클래스

- 기초 자료형을 객체로 포장하는 클래스

오토박싱

- 랩퍼 객체와 기초 자료형 사이의 변환을 자동으로 해주는 기능

8.8 String 클래스

String 클래스

객체 생성

문자열 객체를 생성하는 두 가지 방법

```
String s1 = new String("java");

String s2 = "java";
```

문자열의 기초 연산들

- length()
- charAt()
- concat()

문자열 비교하기

문자열 안에서 단어 찾기

문자열을 단어로 분리할 때

StringBuffer 클래스

8.9 기타 유용한 클래스

8.10 예외 처리

- 예외 : 배열의 인덱스가 배열의 한계를 넘거나 디스크에서 하드웨어 에러가 발생하는 등의 오류
- 일반적으로 오류가 발생하면 프로그램이 종료된다.
- 하지만 오류 처리를 통해 프로그램에서 오류를 감지하여 오류를 처리한 후에 계속 처리할 수 있다.

예외 발생 예제

```
public class DivideByAero {
    public static void main(String [] args) {
        int a = 10 / 0;
        int b = 10;
        System.out.println(a);

        System.out.println(b);
    }
}
```

```
/*
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at DivideByAero.main(DivideByAero.java:3)
 */
```

- 예외가 발생하는 즉시 프로그램이 종료되며 예외가 발생하는 지점 이후의 문장들이 실행되지 않는다.

try-catch 구조

- JAVA에서는 try-catch 구조를 사용하여 예외를 처리한다.
- 예외마다 하나의 catch 블록을 사용해야 한다.

```
try {
    // 예외가 발생할 수 있는 코드
} catch (예외 클래스 변수) {
    // 예외를 처리하는 코드
}
finally {
    // try 블록이 끝나면 무조건 실행된다.(생략 가능))
}
```

0으로 나누는 예외 처리

```
import java.util.Scanner;

public class DiviedbyZeroOK {
    public static void main(String [] args) {
        try {
            int a = 10 / 0;
            int b = 10 / 2;
        } catch (ArithmeticException e) {
            System.out.println("예외 발생");
        }
        finally {
            System.out.println("계속 진행");
        }
    }
}
```

예외의 종류

- try-catch 구조를 사용하여 예외를 처리하려면 클래스 이름을 catch 블록에 적어야 하기 때문에 각종 예외를 나타내는 클래스의 이름을 알아야 한다.

Object

Object			
	Throwable	Error	
	Exception	Runtime Exception	Arithmetic Exception
			NullPointer Exception
			ArrayIndex OutOfBounds Exception

- Error : 하드웨어의 오류로 자바 가상 기계 안에서 치명적인 오류가 발생하면 생성된다.
- 애플리케이션은 이러한 오류를 보고할 수 있지만 예측하거나 복구할 수 없다. 컴파일러가 체크하지 않는다.
- RuntimeException : 주로 프로그래밍 버그나 논리 오류에서 기인하는 오류
- 애플리케이션은 이러한 예외를 잡아서 처리할 수 있지만 보다 합리적인 방법은 예외를 일으킨 버그를 잡는 것이다. RuntimeException도 예외 처리의 주된 대상이 아니다. 따라서 컴파일러가 체크하지 않는다.
- unchecked exception(비체크 예외) : Error와 RuntimeException
- checked exception(체크 예외) : 충분히 예견될 수 있고 회복할 수 있는 예외로 프로그램에서 반드시 처리해야 하는 예외

분류	Exception	Description
Checked-Exception	ClassNotFoundException	클래스가 발견되지 않았을 때
	IOException	입출력 오류
	illegalAccessException	클래스의 전근이 금지되었을 때
	NoSuchMethodException	메소드가 발견되지 않았을 때
	NoSuchFieldException	필드가 발견되지 않았을 때
	InterruptedException	스레드가 다른 스레드에 의하여 중단되었을 때
	FileNotFoundException	파일을 찾지 못했을 때

예제 8-2

```
public class ArrayError {
    public static void main(String [] args) {
        int array [] = {10, 20, 30, 40, 50};
        int i = 0;

        try {
            for (i = 0; i <= array.length; i++)
                System.out.println(array[i] + " ");
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("인덱스 " + i + "는 사용할 수 없네요!");
        }
    }
}
```

예외 8-3

```
public class ExceptionTest3 {
    public static void main(String [] args) {
        try {
            int num = Integer.parseInt("ABC");
            System.out.println(num);
        } catch (NumberFormatException e) {
            System.out.println("NumberFormatException is arising");
        }
    }
}
```

Try-With-Resource(Java 7)

```
import java.io.*;

public class TryTest {
    public static void main(String [] args) {
        try (FileReader fr = new FileReader("hello.rtf")) {
            char [] a = new char[50];
            fr.read(a);
            for (char c : a)
                System.out.println(c);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

예외 떠넘기기

- 발생하는 모든 예외를 그 자리에서 처리하는 것은 상당한 양의 코드가 필요하고 반드시 사위 메소드가 그 예외를 처리하도록 해야 하는 경우가 있다.
- 자신을 호출한 상위 메소드로 예외를 전달하는 경우도 있다.

```
import java.io.*;

public class ExceptionTest {
    public static void main(String [] args) throws IOException {
        FileReader fr = new FileReader("hello.rtf");
        char [] a = new char[50];
        fr.read(a);
        for (char c : a)
            System.out.println(c);
    }
}
```

8.11 모듈(Java 9)

- Module : 여러 가지 자바 패키지들을 하나의 단위(모듈)에 포장할 수 있는 메커니즘

직쏘 프로젝트

- 직쏘 프로젝트의 목표 : 개발자가 라이브러리와 대규모 애플리케이션을 쉽게 구성하고 유지 관리이 할 수 있도록 하는 것

모듈화의 장점

1. 자신이 필요한 모듈만 골라서 실행 파일로 묶을 수 있다.
2. 외부로 노출되는 패키지와 노출되지 않는 패키지를 지정할 수 있다.

모듈의 정의

- 모듈 : 패키지의 모임

자바 모듈 컴파일

모듈 실행

독립 실행형 애플리케이션으로 자바 모듈 패키징

독립 실행형 애플리케이션 실행