KTF 모바일 표준플랫폼 상용화 프로젝트

Java의 기초 문법 정리

개정일자 : 2003.9. 4

㈜KTF

목차

1. 개요

1.1 문서의 개요

본 문서는 KTF-WIPI 개발시 필요한 자바 기초 개념을 정리한 것으로

현직 개발자들을 대상으로 제작된 문서다.

자바 개발자들이 모르고 지나칠 수 있는 부분들을 위주로 적다보니 처음 자바를 접하는 사람에게는 다소 난해한 부분이 있을수도 있다.

본 문서는 계속된 개정과정을 거칠 예정이다.

2. Java Architecture

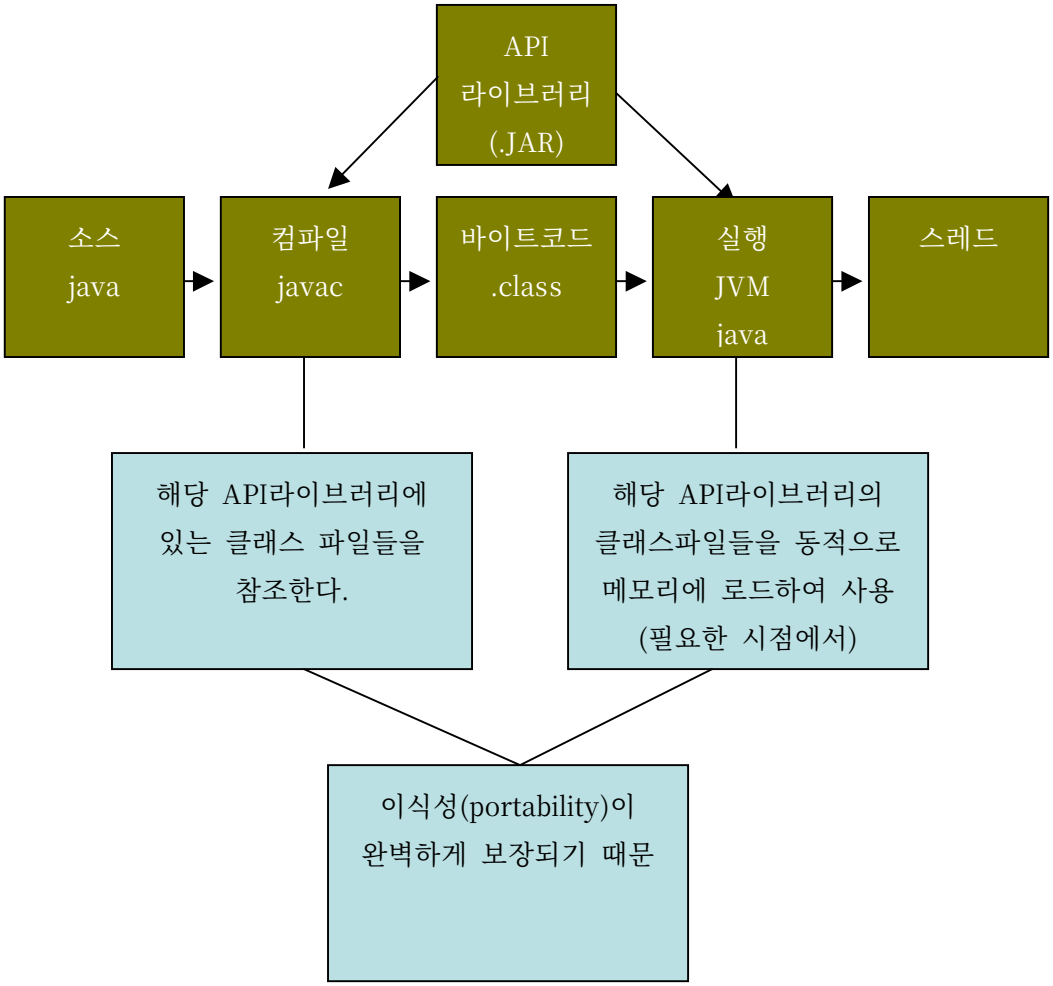
완벽한 자바의 구조는 실제로 4 개의 컴포넌트의 조합으로 이루어진다.

(1) 자바프로그래밍 언어

(2) 자바 클래스 파일 포맷

(3) 자바 애플리케이션 프로그래밍 인터페이스(APIs)

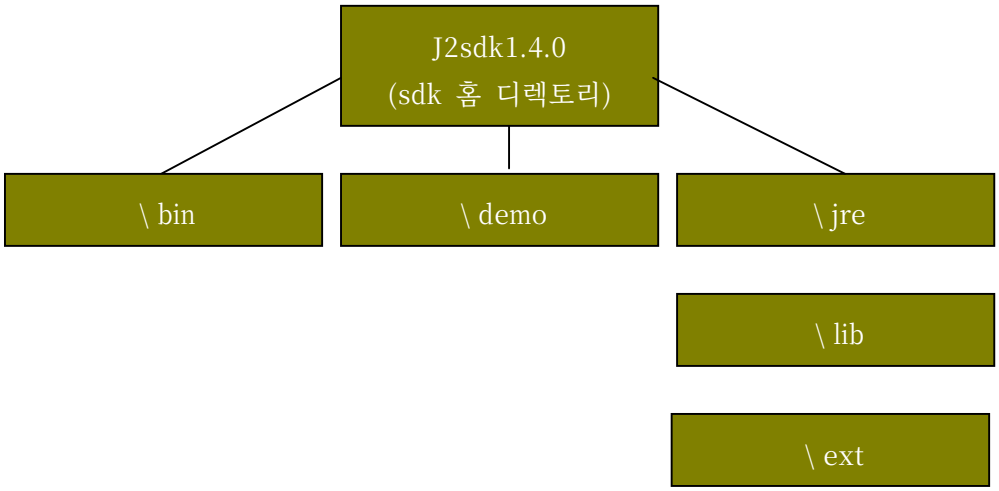
(4) 자바 가상 머신(Java Virtual Machine, JVM)]



2.1 자바 2SDK(Software Development Kit)

툴소프트웨어(javac, java) + API 라이브러리

종전에는 JDK(Java Development kit)라 하였음



\ bin : 컴파일러나 JVM 을 포함하여 각 종 유틸리티 프로그램

\ demo : 각종 데모 프로그램들이 있다

\ jre \ lib : 자바프로그램을 컴파일 또는 실행할 때 필요한 API 라이브러리들이 .jar 파일로 들어있다. 이 중에서, rt.jar[1]가 자바의 핵심 라이브러리로서, 대부 분의 클래스 파일들이 들어있다.

\ jre \ lib \ ext : 우리가 개발한 클래스 파일들을 .jar 파일 형태의 라이브러리로 만든 후 이 디렉토리에 두면, 자바 컴파일러나 JVM 이 자동으로 참조한다.

[1] Jar 은 Java Archive 로서 자바 압축파일 형식이며, 라이브러리를 만들 때 사용

2.2 JRE(Java Runtime Environment) 자바실행환경

JVM 과 코어 클래스[2](API 라이브러리)들의 조합을 자바 2DSK 에서 발췌하여 Java Platform 이 형성된 것

SDK 가 필요없는 사용자 컴퓨터에 자바 실행환경만을 설치하기 위한 것이다.

어떠한 운영체제에서도 사용 될 수 있다.

JVM 이 시작될 때 JRE 도 로드된다.

[2] 코어 클래스 : 원래 core 의 의미를 그대로 유지하려면 매우 중요한 "핵심" 사항입니다. 그러나, 코어 자바에서는 기본사항만 다루었고 코어 클래스는 중요한 것만 다루어서,코어 자바는 기본 자바로 코어 클래스는 핵심 클래스로 이해하면 문안할 듯

2.3 자바 가상 머신(Java Virtual Machine, JVM)

자바의 엔진을 Java Virtual Machine(JVM)이라고 한다.

자바 가상 머신은 컴파일된 자바 프로그램을 해석하는 추상적인 컴퓨팅 머신이다.

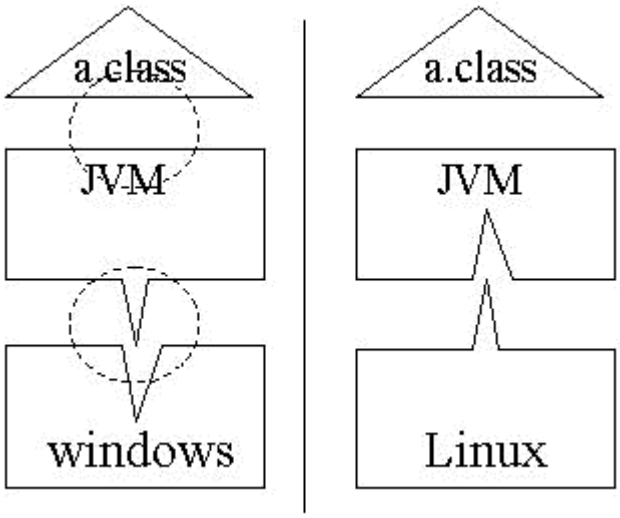
자바언어의 이식성(portability)의 핵심은 자바 컴파일러가 바이트코드 프로그램이라고 불리는 형태로 컴파일하게 되어 있다.

이 바이트코드 class 파일은 그 자체가 실행파일이 아니다. 따라서 그냥 수행될 수 없으며, 이 class 파일을 읽어서 해석한 후, 실행해 줄 무언가가 필요한데, 그것이 바로 JVM 이다

운영체제에서 직접 실행되지 않고 바이트코드를 해석하는 JVM 을 통해 실행되기 때문에 자바는 해석 언어(interpreted language)라고도 불리기도 한다.

컴파일 후 바로 실행할 수 있는 언어,특정 하드웨어에 최적화된 언어,또는 기계어 형태로 직접 실행하는 언어와 비교할 때 기계어로 바꾸면서 실행하는 자바가 성능이 떨어질 수도 있다.그러나,노키아등에서 실행되는 자바를 본다면 꼭 그렇지도 않다.이건 필자 견해다.

이런 문제 해결을 위해 다른 언어로 작성된 APP 와 결합하기 위한 API 가 JNI 라고 불리며 자바 코드 이외의 언어를 호출하여 사용할 수 있다.그러나,플랫폼의 독립성을 상실하므로 지양하는 API 다.



2.4 JVM 동작

로딩(loading),링크(link),초기화(initializing)

JVM 이 자바 바이트코드 해석하기 위해서 필요한 클래스와 인터페이스에서 다음의 3 단계를 수행해야 한다.

(1) 로딩(loading)

JVM 이 클래스를 로드할 때, 클래스나 인터페이스의 바이너리 표현을 찾고, 그 바이너리 표현(일반적으로 자바 컴파일러에 의해 생성된 클래스 파일)으로부터 Class 객체를 생성한다. Class 객체는 클래스나 인터페이스의 런타임 상태를 캡슐화한다.

(2) 링크(link)

링크는 로드된 클래스나 인터페이스를 가져와 JVM 의 런타임과 결합하여 실행을 준비하는 과정이다.

(3) 초기화(initializing)

초기화는 JVM 이 클래스나 인터페이스 초기화 메소드를 호출할 때 이루어진다.

2.5 런타임(runtime)실행 환경으로서의 JVM 의 런타임데이터 영역

2.5.1 heap

heap 은 일시적이고 동적인 메모리 할당을 위해 사용되는 자유 메모리 영역

heap 은 클래스와 배열 객체에 메모리를 제공하는 런타임 데이터 영역

JVM 이 시작할 때 heap 은 생성되고,클래스나 배열에 할당하게 된다.

Heap 메모리는 객체나 배열에 대한 참조가 더 이상 존재하지 않을 때 가비지 콜렉터에 의해 회수된다.

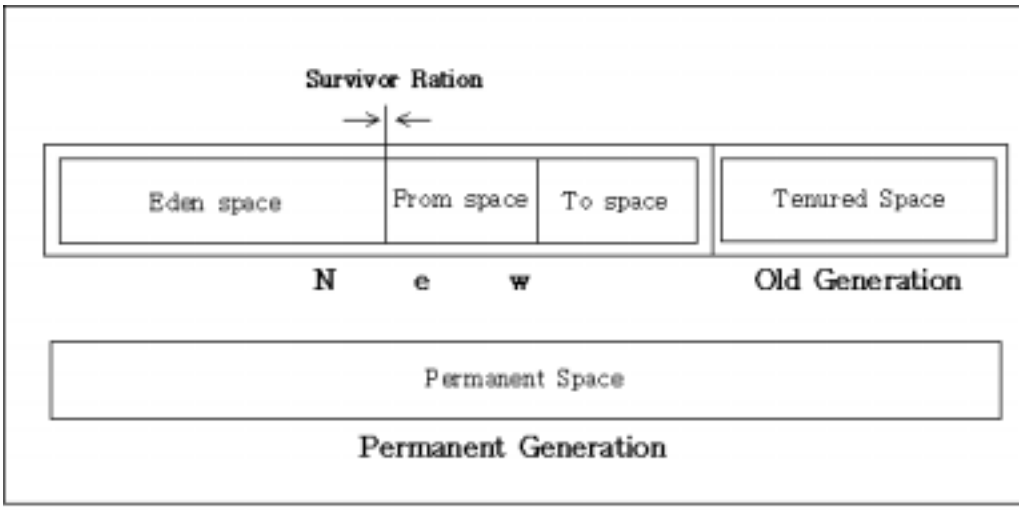
Heap 메모리가 부족하면 OutofMemoryError 예외가 전달된다.(필자는 LGT 의 C9 과 삼성 TMobile 어플리케이션 만들 때 지겹도록 이 예외를 보았죠)

힙의 세 영역은 다음과 같다

(1) Permanent space: JVM 클래스와 메소드 개체를 위해 쓰인다.

(2) Old object space: 만들어진지 좀 된 개체들을 위해 쓰인다.

(3) New(young) object space: 새로 생성된 개체들을 위해 쓰인다.



Heap layout

New object space 는 세 부분으로 다시 나누어진. 모든 새로 생성된 개체들이 가는 Eden, 그리고 그 개체들이 나이들게(Old) 되기 전에 가는 Survivor space(From, To) 1 과 2 가 있다.

2.5.1.1 OutOfMemory Error 및 해결방법

OutOfMemory Error 해결방법으로는 jdk1.4 에서 -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+PrintHeapAtGC 옵션을 사용한 GC 한 상태의 Heap 메모리 정보출력 한다. GC 정보를 통하여 New, Old, Perm 등의 영역중 실제 어느 부분이 부족하여 OutOfMemory 가 발생하는지 찾은후 부족한 영역의 충분한 size 를조절해 주는 방법으로 해결할 수 있다.

메모리 할당을 해주어도 사용률이 100%가 나온다면 프로그램 누수일수 있으니 프로그램을 점검해 봐야 할 것이다.

2.5.2 Stack

자바 스택 프래임은 메소드 호출 상태를 저장하는데 사용

스택 프래임은 데이터와 부분적인 결과를 저장하는데 사용되며, 메소드의 실행 환경,메소드 호출에 사용된 로컬 변수들,메소드의 피연산자 스택(operand stack)[3]을 포함

스택영역이 부족하면 StackOverflowError 예외가 발생한다.

[3] 피연산자 스택(operand stack) 파마미터를 저장하고 대부분의 바이트코드 구조에서 값을 반환한다.

2.5.3 method area

모든 JVM 쓰레드가 공유하는 일반적인 저장 영역.런타임 상수풀,메소드 데이터,필드 데이터,메소드와 생성자의 바이트코드 등과 같은 것들을 저장하는데 사용된다.

2.5.4 Register

프로세서(cpu)에 있으며 가장 처리속도가 빠르다.그러나, 개수가 정해져 있어서 컴파일러가 필요할 때 할당한다.우리가 직접제어하지는 않으며 프로그램에서도 전혀 언급하지 않는다.컴퓨터 시스템의 레지스터와 유사하다.레지스터는 바이트코드 실행시 업데이트 된다.

2.5.5 runtime constant pool(런타임 상수 풀)

숫자 문자형과 필드의 상수를 포함하는 상수들을 가지고 있다

각 런타임 상수 풀의 메모리는 메소드 영역으로부터 할당된다.클래스 파일 생성시 생성됨

2.5.6 비메모리영역

데이터가 프로그램 밖에 저장되어 있으면 프로그램이 실행되지 않아도 존재하며 프로그램의 제어를 받지 않는다.

Ex) stream object

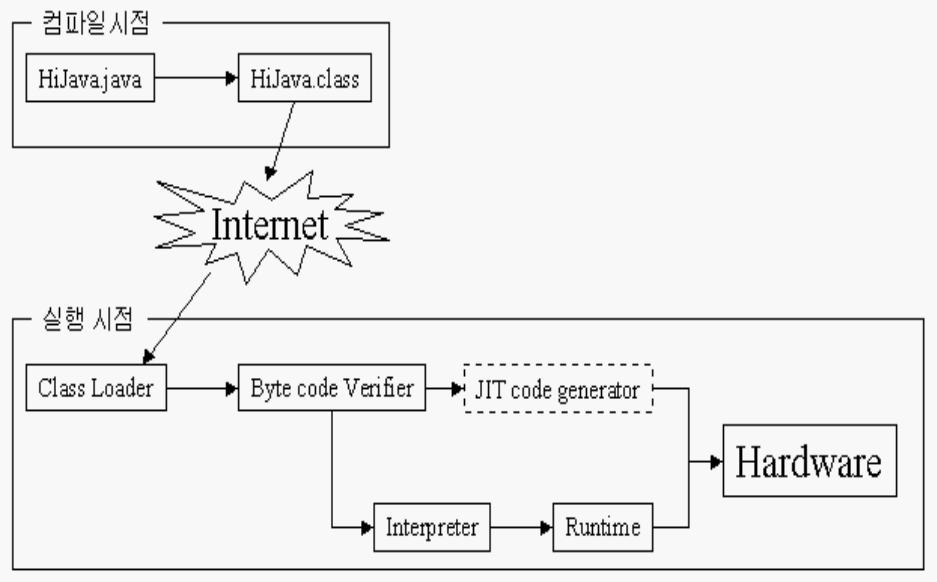
Store(영속객체)

2.6 Java 의 장점

2.6.7 Scalability

Java 를 할 줄 알면 어느 곳에나 적용 가능한 프로그램을 개발 할 수 있다는 말이다. 그 뿐만 아니라, Java 로 개발된 시스템은 다른 언어로 개발된 시스템보다도 훨씬 쉽게 확장이 가능하다

2.6.8 Security



network 을 통해 온 것이면 철저한 byte code verifier 단계를 거치는데, 이 단계에서는 주로 읽은 class 파일의 security 를 check 한다. 만약 security 에 어긋나는 code 가 숨어 있으면 error 가 발생한다.

2.6.9 Universality

'Write Once, Run Anywhere'

플랫폼의 독립성.O/S 에 종속되지 않는다

2.6.10 Modularity

자바는 완전한 객체지향언어.자바의 모든 것은 클래스로부터 시작합니다.

Java 를 이용해 재사용 가능한 object 를 만들 수 있다. 이 object 는 향후 다른 프로그램내에서 그냥 재사용 가능하다. 강력한 Java 의 재사용성은 Java 가 가지고 있는 장점 중에서도 가장 큰 장점

*.객체화,인스턴스화 : 클래스로 만들어진 것

2.6.11 Thread

2.6.11.1 Proccess

프로세스는 운영체제가 관리하는 작업의 단위입니다. 프로그램을 실행시킨다는 것은 프로세스를 만들어서 사용한다는 뜻입니다.

운영체제는 이 프로세스를 관리하여 여러 프로세스들을 관리하는 방식에 따라 선점형 멀티태스킹과 비선점형 멀티태스킹 방식으로 구분합니다.

2.6.11.2 Thread

혼자서 자신의 고유의 일을 할 수 있는 프로그램 안의 작은 프로세스가 바로 thread

자바는 쓰레드를 java.lang.Thread 의 객체로 표현합니다. Thread 의 run() 메소드를 벗어나면 스레드는 종료되는 것이며,스레드 객체를 다시 사용할 수 없습니다.

2.6.11.3 java.lang.Runnable

스레드는 run()메소드에 있는 코드를 실행하는데 run()메소드는 사실 Runnable 인터페이스에서 온 것입니다. 그러므로 스레드를 만들 때 Runnable 인터페이스를 구현한 객체를 따로 건네줄 수도 있습니다. 이는 스레드가 꼭 Thread 클래스를 상속받아야만 하는 것은 아니며 좀 더 유연한 디자인이 가능함을 보여줌

자바에서는 멀티쓰레드를 지원하고 새로운 쓰레드를 만들고 사용하기가 매우쉽다.

따라서 thread 들은 생성될 때 process 만큼 system 에 부하를 주지는 않는다.
또한 code 나 data 를 공유하기도 쉽다.

2.6.12 Garbage Collection

사용되지 않는 메모리에 대한 회수를 'Garbage Collection'이라고 한다. 자바에서는 프로그램이 사용하는 메모리를 JVM(Java Virtual Machine)이 모두 관리한다.

가비지 콜렉트는 프로그램이 실행되는 동안 가끔씩 레퍼런스 검사를 수행한다.

가비지 콜렉트 기능이 올바로 수행하는지 보증할 수 없지만 System.gc()를 사용하여 강제 수행 시킬 수 있습니다.

가비지 콜렉션이 안되는 객체도 있을 수 있다.

우리 프로그램의 실행 중에 메모리 부족이 생기지 않는다면 객체가 사용했던 메모리가 회수되지 않을 수도 있다. 가비지 컬렉션이 발생하지 않을 수 있기 때문이다. 그러나, 프로그램은 종료되었지만 그 전에 가비지 켈렉터가 메모리를 회수하지 않았다면 오퍼레이팅 시스템에서 회수하게 될 것이다.

2.6.13 Exception

프로그램 실행도중 치명적이지는 않으나 어떤 잘못된 상황에 맞닥뜨렸을 때 발생

call stack mechanism(자신을 호출한 method 에게 Exception 을 던지는 구조)

Error : 복구 불가능

Exception : 에러 상황

mild type 의 에러 (복구가능)

2.6.13.1 Runtime Exception

프로그래머의 잘못으로 발생. 컴파일은 되고 실행은 안됨

0 으로 나눈 경우, array 에서 범위를 넘어선 경우

2.6.13.2 compiletime Exception

프로그래머가 잘못한 것이 아니라 사용자가 잘못해서 발 생하는 exception. 주로 input, output 시에 발생 (없는 파일을 열거나 저장할려고 할 경우)

컴파일 되지 않는다. 따라서 end user 에 의해 발생할 수 있는 에러를 교정해주어야 한다.

2.7 SDK 를 이루는 요소

2.7.14 Class & Package

Class : user 가 직접 만들어 사용하는 것, Java Virtual Machine(Java VM)이 내용을 이해해 실행하는 것

Package : JDK(or J2SE)안에 포함되어 있거나, 혹은 기존에 누군가가 만들어 놓은 class package 는 class 들의 묶음

2.7.14.1 패키지의 용도

패키지는 몇개의 클래스를 묶는 것이다. 주로 3개의 목적으로 사용된다.

(1) 필요한 클래스를 간단하게 찾아낼 수 있도록 한다.

(2) 같은 이름의 클래스가 복수 존재할 경우 ,클래스명의 충돌을 막는다
패키지명을 길게 해도 프로그래머의 부담은 무거워지지 않다.

2.7.15 Java API Documentation(Java Application Programming Interface)

자바 어플리케이션 프로그래밍 인터페이스(API)는 기존의 코드들을 유사한 기능의 패키지로 분리해 놓은 것(J2SE,J2EE,J2ME)

없어지거나 변경된 API 를 일컬어 deprecation 이라고 한다.
deprecation 컴파일시에 다음과 같은 에러 메시지를 출력한다.
Note: TestDeprecation.java uses or overrides a deprecated API.
Note: Recompile with -deprecation for details.

2.7.16 Java Tutorial

예제

2.7.17 Identifier

class 명, method 명, 변수명 등, 이름을 필요로 하는 것들이 가지고 있는 이름들을 통틀어 identifier 라고 한다

2.8 A Basic Java Application (자바 유틸리티 툴)

JVM 의 대부분을 구성

모든 유틸리티 툴들의 설명은

썬 웹 사이트(htt://java.sun.com/j2se/1.3/docs/tooldocs/tools.html)에서 찾아 볼 수 있음

2.8.18 Compiler

(1) 컴파일 : 원시파일[4]로부터 클래스[5] 파일을 생성하는 것,

javac 라는 컴파일러는 클래스와 인터페이스의 정의를 읽고 바이트코드 파일로 변환한다.

javac 는 파일간의 의존관계(dependencies)를 파악해 원시 파일을 자동적으로 컴파일 한다. (원시 파일이 3 개 이상이 되면, 잘 동작하지 않는다)

(2) javac 에러나는 경우

*. 원시 파일에 잘못이 있는 것이다. 에러 메세지를 참고로 해, 원시파일을 다시 읽어, 잘못이 있으면, 올바르게 고쳐 쓴다.
*. 클래스 패스의 지정이 잘못되어 있는 것이다. 원시 파일이 참조하고 있는 클래스의 클래스 파일이 클래스 패스에 존재하고 있지 않은 경우에도, 클래스 파일은 생성되지 않는다.

2.8.18.1 javac 명령의 명령 라인 옵션

(1) ?

javac 명령의 사용법을 표시한다

(2) -classpath

javac가 원시 파일을 컴파일 할 경우에 참조해야할 클래스 패스를 지정하기 위해서 이용한다.

---

[4] 원시 파일: 우리가 읽거나 쓰거나 하는 것

[5] 클래스: Java Virtual Machine(Java VM)이 내용을 이해해 실행하는 것

(3) -d

javac에, 클래스 파일을<디렉토리명>로 지정한 디렉토리에 생성 시키기 위해서 이용한다

(4) -sourcepath <소스 패스>

javac가 원시 파일을 검색하는 패스를 지정하기 위해서 이용한다. 이 옵션을 지정하지 않는 경우는, javac는 원시 파일을 현재 디렉토리로부터 찾는다. 덧붙여 소스 패스의 지정 방법은, 클래스 패스의 지정 방법과 같다

(5) -g:<키워드(의 리스트) >

디버그 정보를 클래스 파일에 추가 하기 위한 옵션이다.

-g:lines

javac는 원시 파일중의 행 번호의 디버그 정보를 클래스 파일에 추가한다.

-g:vars

javac는 로컬 변수의 디버그 정보를 클래스 파일에 추가한다.

복수 종류의 디버그 정보를 javac에 생성 시키기 위해서는, -g:의 뒤의 키워드를“, ”로 단락짓는다.

-g:none

javac는 디버그 정보를 생성하지 않는다.

-g옵션을 지정하지 않는 경우

g:lines, source와 지정한 것과 같은 동작을 한다.

(6) –O

javac는 클래스 파일을 최적화 즉 보다 효율 좋게 실행 할 수 있도록 클래스 파일을 생성한다… 라고 하게 되어 있지다만, 현재는, -O옵션은 무효이다.

(7) -nowarn

javac는 Warning메세지를 출력 하지 않게 된다

(8) -verbose

javac가 실제로 어떠한 작업을 하고 있을까를 알 수 있다.

(9) -deprecation

javac는 원시 파일중에 존재하는 추천 되지 않는 멤버나 클래스에 대해서, 그 상세를 표시한다.

(10) -bootclasspath

이 옵션은, 부트 클래스 패스를 지정하기 위해서 이용한다. 부트 클래스 패스에는 표준 시스템 라이브러리에의 패스를 지정한다.

(11) -extdirs

extension 메카니즘의 extension 디렉토리를 지정하기 위해서 이용한다.

(12) -target

클래스 파일이 어느 버젼의 Java VM로 실행 할 수 있도록 할까를 지정한다.

지정 할 수 있는 버젼은, 1.1, 1.2, 1.3의 3 종류이다.

1.1을 지정하면, 버젼 1.1이후의 모든 Java VM로 실행 할 수 있는 클래스 파일이 생성된다. -

target 옵션을 지정하지 않았던 경우는, javac는-target에 1.1을 지정한 것과 같은 동작을 한다.

(13) -encoding

원시 파일이 어떠한 종류의 encode로 쓰여져 있을까를, javac에 알리기 위해서 이용한다.

이 옵션을 지정하지 않으면 javac는, operating system의 표준의 문자 코드를 사용하고 있는 것으로서 원시 파일을 컴파일 한다.

2.8.19 Interpreter

자바 유틸리티는 자바 애플리케이션을 실행하는 데 사용된다.클래스파일에 포함된 바이트코드를 해석

java 명령은 클래스 파일을 실행하는 툴이다.

2.8.19.1 java 명령시 에러들

Java VM은 실행에 필요한 클래스 파일을 찾아낼 수 없을 때

Exception in thread "main" java.lang.NoClassDefFoundError:

public s tatic void main(String[])라고 하는 메소드가 존재하지 않을 때

Exception in thread "main" java.lang.NoSuchMethodException: main

2.8.19.2 java 명령의 명령 라인 옵션

(1) ?

java 명령의 사용법을 표시한다

(2) “-? ”또는“-help”

java 명령은, java 명령의 간단한 사용법을 표시하고 종료한다.

(3) -classpath

Java VM의 클래스 패스를 지정할 수가 있다.

(4) -version

java 명령은 Java VM의 버젼을 표시해 종료한다.

(5) -showversion

클래스 파일의 실행전에, java 명령은 Java VM의 버젼을 표시합니다.

(6) "-verbose"또는" -verbose:class"

Java VM는 실행중에 클래스 파일을 읽어들일 때마다, 메시지를 표시한다

(7) -verbose:gc

Java VM은 가베지 콜렉션을 할 때마다, 메세지를 표시한다.

(8) -verbose:jni

Java VM은 Java Native Interface(JNI)가 사용될 때마다, 메시지를 표시한다.

(9) -X

비표준 옵션에 대한 설명이 표시된다.

(10) -jar

java 명령에 클래스 파일은 아니고 JAR 파일을 지정하면, Java VM는 그 JAR 파일로부터 직접 실행하는 클래스를 읽어들여, 실행할 수가 있다. 이 기능을 이용하기 위해서는, JAR 파일에 어느 클래스를 최초로 Java VM로 실행하면 좋은가의 정보를 추가해 둘 필요가 있다.

2.8.19.3 비표준 옵션

Sun 이외(마이크로소프트, IBM 등)의 Java VM의 설치에서는 지정 할 수 있는 보증이 없는 옵션이다. 또, 다음 버젼의 설치에서는, 이러한 옵션이 사용할 수 없게 될 가능성이 있기에 종류만 알아보고 넘어가겠다.
-Xmixed,-Xint,-Xdebug,-Xbootclasspath:,-Xbootclasspath/a:,-Xbootclasspath/p:,-Xfuture,-Xnoclassgc,-Xincgc,-Xms,-Xmx,java -Xmx256m,-Xprof,-Xrunhprof,java -Xrunhprof:help

2.8.20 자바 클래스 분해

javap 유틸리티는 클래스 파일 내부를 살펴보는 데 사용될 수 있다. 기본 명령은 non-private 와 non-static 필드,메소드.생성자,정적 초기화의 선언을 나타낸다.

Javap 유틸리티는 각 메소드를 실행하는 JVM 명령의 출력정보를 제공하는데 사용

2.8.20.1 javap 의 명령의 명령 라인 옵션

(1) -b

이번 버전의 javap 와 호환되는지 확인

(2) -bootclasspath

이 명령에 이어지는 경로는 부트스트랩 클래스를 로드하기 위한 경로 지정

일반적으로 /lib/rt.jar 에 포함된 클래스를 말한다.

(3) -c

각 메소드를 실행할 때 사용되는 JVM 명령을 프린트한다. 이 정보는 각 메소드의 바이트코드가 실제로 수행하는 것이 무엇인지 말해준다.

(4) -classpath

이 명령 다음에 사용자 지정 클래스 경로가 이어지고, 이 명령은 시스템 CLASSPATH 환경변수를 오버라이드한다.

(5) -extdirs

이 명령 다음에 이어지는 디렉토리는 시스템이 인스톨된 확장 부분에서 검색하는 위치를 오버라이드한다.기본 위치는 /lib/ext 이다.

(6) -help

javap 유틸리티에 관한 정보를 프린트한다.

(7) -Jflag

런타임 시스템에 지정된 플래그를 직접 전달한다.

(8) -l

라인과 로컬변수들을 출력한다.

(9) -package

package,protected 및 public 클래스와 멤버만을 보여준다.디폴트이다.

(10) -private

모든 클래스와 멤버에 대한 정보를 보여준다.

(11) -protected

protected,public 클래스와 멤버에 대한 정보를 보여준다.

(12) -public

public 클래스와 멤버에 대한 정보를 보여준다.

(13) -s

내부 타입 신호를 프린트한다.

(14) -verbose

스텍 크기,로컬변수정보,아규먼트를 포함한 각 메소드에 대한 추가적인 정보를 프린트한다.

3. Identifiers,Keywords,Types, and Flow Control

3.1 Keywords

우리말로 예약어라고도 합니다.무언가 그 의미가 미리 정해져 있다는 소리.

기본 데이터 유형	Boolean,byte,char,short,int,long,float,double,void
변수유형	Transient,volatile
상수	True,false,null
흐름제어	Ifelse,switch,case,default,for,while,do,break,continue,return
클래스	Class,interface,extends,implements,static,abstract,final,new ,instanceof,this,super
접근제어	Public,protected,private
메소드	Synchronized,native
패키지	Package,import
예외처리	Try,catch,finally,throw,throws
사용하지않는 키워드	Goto,const

3.1.1 Interface

인터페이스(interface) : 외부에서 접근 가능한(메시지를 보낼 수 있는)멤버 메소드들의 명세를 총칭하는 것

이러한 메소드 내부의 실행 코드를 포함해서 클래스 내부에 캡슐화 되어 있는 코드들을 구현(implementation)부분이라 한다.

interface 라는 것은 일종의 abstract class 인데, 이것이 가지고 있는 method 들은

모두가 반드시 abstract method 들이어야 한다. 다시 말해서 abstract method 만 가지고 있는 abstract class 가 interface 인 것이다.

정확한 표현은 “인터페이스를 구현한다”이다.

인터페이스는 구현부분을 전혀 갖지 않으므로, 메모리도 차지하지 않고,여러 개가 결합되어도 문제가 없다.

interface 에서 선언하는 변수,필드는 무조건 static final 이 붙는다고 생각하자.즉, 자동적으로 상수가 된다.

interface 라는 것은 그 단어가 의미하듯이 '쌍방간의 약속'이라고 할 수 있다. 이 예제처럼 엔진을 사서 조립하는 쪽과 그러한 엔진을 만드는 쪽에서 서로 일종의 약속(specification 혹은 표준)을 정하는 것처럼 말이다

interface 안에 선언된

member variable → final static (안해줘도 자동으로 선언)

member method → abstract (안해줘도 자동으로 선언)

class 와의 비교

class 처럼 data type 으로 사용 가능하지만, instance 를 직접 생성할 수 없다.

interface 를 상속 받을 때는 implements 를 사용

다중 상속의 기능 구현(impelments)

특정 객체에 어떤 일을 수행하도록 요청하는 것

요청에 대한 일을 해주는 코드가 어딘가에 있어야만 그 객체가 그 일을 해줄 수 있을 것이다. 그러한 코드는 객체의 데이터와 더불어 객체 내부에 감추어져 있으며,객체 내부에 감추어진 데이터와 코드구현(implementation)부분이다.

Interface 앞에 public 을 지정할 수 있다 (인터페이스 이름과 소스파일명이 같을 때) 접근 지시자를 주지 않을 때는 “friendly”가 되어 같은 패키지에서만 사용할 수 있다.

인터페이스에 정의된 모든 메소드는 자동으로 public 이 된다. 따라서,인터페이스를 implements 하는 클래스에서는 구현할 메소드를 반드시 public 으로 지정해야 함

3.1.1.1 인터페이스를 사용하는 이유

(1) 객체를 하나 이상의 상위 타입으로 업캐스팅 할 수 있기 때문

(2) 객체를 생성하지 못하게 하면서 외부와의 인터페이스를 정의할 수 있도록 하는 것

3.1.1.2 인터페이스간의 상속

인터페이스간에도 상속이 가능하며, 이 때는 클래스처럼 extends 키워드를 사용한다.인터페이스는 구현부분을 갖지 않으므로, 인터페이스 상속은 여러 인터페이스를 결합하는 것이라 생각할 수 있다. 이 경우 상위 인터페이스를 수퍼(super)인터페이스라 한다.인터페이스는 다중상속이 가능하다.

중첩된 인터페이스가 있는 인터페이스를 implements 하면 중첩된 인터페이스는 별도로 implements 하지 않아도 된다.

3.1.2 Import

package 가 선언됐을 때 name space 를 확장해주는 개념

package 를 선언 안하면 컴파일러가 anonymous package 를 제공한다.

package 는 directory information

3.1.3 this

this 키워드는 메소드 내부에서만 사용될 수 있으며, 메소드 호출이 생긴 객체의 참조를 나타낸다.

this 키워드는 현재 객체의 참조를 명시적으로 지정할 필요가 있는 경우

자기 자신의 object 를 가리킴. 반드시 object 화 되어 있어야만 사용할 수 있다. 혹은, 자신의 constructor 를 호출할 때도 사용된다.

Super 와 마찬가지로 괄호 안에 인자가 들어갈 수 있으며, 생성자의 맨 첫줄에는 this 나 super 가 분명히 있어야 하며 없다면 super()를 컴파일러가 자동으로 넣어줌

생성자에서는 this 키워드가 다른 의미를 갖는다. 다른 생성자를 호출하라는 것,단 이 때는 메소드를 호출하는 형태로 this 키워드를 사용, this()와 같다. 항상 맨 앞에 나와야하며 생성자안에서만 사용할 수 있다.

(1) 만들어진 자기 instance 를 가리킨다.

(2) 보통 생략한다.

2 가지 경우만 제외

(1) member variable 과 local variable 의 이름이 같은 경우 구분을 위해

(2) constructor 안에서 또다른 constructor 을 호출할 때

this 는 instance 의 생성이 보장된 경우에만 사용

(static method 안에서는 this 를 사용할 수 없다. 물론 main method 안에서도 사용할 수 없다.(main 도 static 이므로))

3.1.4 static

static : 클래스가 메모리 공간으로 올라올 때 가장 먼저 실행되는 곳

static 이라고 붙은 변수와 메소드들은 전부 클래스로 접근

클래스 변수 : static 으로 선언된 변수

클래스 메소드 : static 으로 선언된 메소드

(절대 static method 는 instance 변수를 참조할 수 없다.)

인스턴스 필드 : 인스턴스 메소드와 인스턴스 변수를 함께 부를 때

클래스 필드 : 클래스 메소드와 클래스 변수를 함께 부를 때

인스턴스 변수 : 객체가 생성될 때 static 이라고 선언되지 않은 다른 변수들이 초기화 되는데 이들은 각각 초기값을 가지게 되는데 이들을 instance 변수라고 부릅니다.

자바 컴파일러는 default 생성자와 java.lang.Object 을 따로 작성하지 않았다면 자동으로 생성시켜줌

생성자 위에 존재하는 super()나 this()를 받드시 적어줘야 하는데 없을 경우 super()를 따로 작성하지 않았으면 자동으로 넣어 줌

this 키워드는 객체라는 동적인 의미를 갖는 반면에 static 은 정적인 의미를 나타냄

메소드를 static 으로 지정한다는 것은 메소드 내에서 this 키워드를 사용할 수 없다는 것을 의미한다.

static 메소드에서는 static 이 아닌 메소드를 호출할 수 없다.(반대는 가능하다.)

static 메소드는 객체를 생성하지 않고도 호출할 수 있다.

Static 메소드에서는 다른 static 메소드나 필드를 액세스할 수 있다.

어떤 사람들은 static 메소드가 전역(global)메소드의 의미를 갖는다고 해서 객체지향개념에 어긋난다고 주장한다.

3.1.5 final

변하지 않게 하는 이유는 설계와 효율성(efficiency) 관점이다.

final 은 데이터,메소드클래스 세군데 사용하고 있다

final Variables,final Methods,final Classes

3.1.5.1 Blank final

초기값을 주지 않은 필드를 blank final

사용되기 전에 반드시 초기화

3.1.5.2 final 인자

메소드의 인자를 final 로 지정할 수 있다

3.1.5.3 final Methods

상속받은 클래스에서 final 메소드의 의미를 못바꾸도록하기 위해

효율성

메소드를 final 로 지정하면, 그 메소드에 대한 모든 호출을 컴파일러가 인라인[6](inline) 호출의 형태로 바꾼다.

동적바인딩[7]이 필요없다고 컴파일러에게 알려주는 것이다.

3.1.5.4 binding

바인딩(binding) 실제 실행할 메소드 코드와 호출하는 코드를 연결시키는 것

프로그램이 실행되기 전에 즉,컴파일 할 때 바인딩 되는 것을 early 바인딩 또는 정적(static)바인딩이라 한다.

late 바인딩, 실행할 당시의 객체 타입을 기준으로 바인딩되는 것을 말한다.late 바인딩은 동적(dynamic)바인딩 또는 런타임바인딩이라고도 한다.

자바의 모든 메소드 바인딩은 late 바인딩을 사용한다.단, final 메소드만이 예외이다. 자동으로 된다. 메소드를 final 로 지정하는 이유는 동적바인딩이 필요없다고 컴파일러에게 알려주는 것이다.컴파일러가 final 메소드 호출을 조금 더 효율적인 코드로 생성할 수 있다.그러나 대부분의 경우에 프로그램의 전체적인 성능은 크게 향상되지는 않는다.따라서, 성능향상보다는 설계관점에서 final 의 사용을 고려하는 것이 가장 좋은 방법이다.

3.1.6 super

부모의 object 를 의미. 혹은 부모의 constructor 를 호출할 때도 사용된다

3.1.7 return

메소드의 실행을 정상적으로 끝내고 값을 반환할 때

---

[6] 인라인(inline) 원하는 위치에 코드를 직접 끼워 넣고 실행하도록 하는 것을 인라인이라한다.

[7] 5.25.4 를 계속해서 binding 에 대해 설명해 놓았음

실행을 중단하고 그 시점의 값을 반환하는데 사용

리턴값을 결과값으로 되돌려주겠다는 의미.

원칙적으로 return void 임.void 생략

```
public static Soup access()
{
        return ps1;
}
```

Soup 객체의 참조를 리턴한다는 것을 의미

3.1.8 instanceof

Instanceof 는 연산자가 아니라 키워드.boolean 값을 리턴하기 때문에 연산자 취급을 받음.클래스로 만든 객체인지 아닌지 검사

String a = "위피 탄생 축하"

System.out.println(a instanceof String);

원래 클래스 하나에 있는 모든 String 객체들은 나름대로 효율적인 메모리 관리를 위해 중복되는 String 객체일 경우 같은 값을 참조하도록 되어 있음.

이런 참조값을 확인하고 싶다면 String 클래스의 interm()메소드를 이용하면 됩니다.

이 메소드는 JVM 내부의 String 객체들을 모아놓은 곳에서 String 을 가져오므로 공유되고 있는 String 인지 아닌지를 확인할 수 있습니다.

3.1.9 transient

I/O 관련, 어떤 클래스안에 데이터를 하나의 객체로 묶을 때, 이 객체는 포함 시키지 않을 때 사용,

3.1.10 volatile

휘발성의 변수에 적용,비동기화, default 로 선언되기 때문에 거의 쓸 일이 없다.

↔ synchronzied

3.1.11 native

플랫폼에 dependent 한 언어를 사용하고자 할 때

3.1.12 strictfp

strictfp 는 부동소수 계산의 특정한 사항들을 제어하는 데 사용된다

3.2 Variable & Constants (변수와 상수)

변수 : 변하는 수

(1) 변수는 유니코드[8]의 문자로 구성됩니다.

(2) 변수의 이름은 숫자로 시작돼서는 안됩니다.

(3) 변수는 대소문자를 구분합니다.

(4) 키워드로 지정된 것은 변수명으로 사용할 수 없습니다.

(5) 변수의 유효범위는 {}에 의해 결정된다.

(5) 변수는 가장 먼저 초기화 된다.

3.2.13 Constants

변하지 않는 수,모두 대문자.다음단어 사이에 underscore(_)를 준다.final keyword 사용

[8] 유니코드 컴퓨터가 전세계적으로 확산되면서 생긴 문제가 문자코드에 관한 문제.알파벳은 1 바이트로 충분한데 몇몇 나라는 2 바이트여만해서 혼란을 막기 위해 컴퓨터 관련 회사들이 유니코드라는 통합된 코드의 체계를 만들게 되었음.자바는 유니코드 일종인 UTF-8 을 사용

3.2.14 Local variable

method 내에서 선언된 변수를 지역변수(local, automatic, temporary, stack variable

지역변수의 생성시점은 그 method 가 수행될 때이며, method 수행이 다 끝나면 변수는 memory 에서 없어진다

또한 이 변수를 사용하는 시점에 초기화되어 있지 않으면 compile error 가 발생

3.2.15 Member variable

자동 초기화

Global (static) variable

3.3 Data type

3.3.16 Primitive Types

Call by Value : 값을 넘겨준다.

data 영역이 자동으로 생성

3.3.16.1 Integral

byte = -2 의 7 승부터 2 의 7 승 -1 까지

short = -2 의 15 승부터 2 의 15 승-1 까지

int = -2 의 31 승부터 2 의 31 승 까지(default)

long = -2 의 63 승부터 2 의 63 승 -1 까지

한 번처리하는 자바의 기본 데이터 단위는 in

즉,기본처리 단위가 4 바이트이기 때문에 byte 를 쓰나 int 를 쓰나 이득보는 것은

없음

프로그램 코드상에서도 정수에 대한 연산이 이루어졌으면 자바는 그것의 결과값을 int 형 데이터로 간주

byte,short 등은 통신을 하는 등의 데이터 양이 중요할 때 사용

만약,long 의 범위를 벗어나는 정수를 사용하고 싶다면 java.lang.math.BigDecimal 이라는 클래스를 사용

8 진수는 숫자 앞에 '0'을 덧붙입니다

16 진수는 숫자 앞에 0x 를 덧붙입니다

이렇게 하면 나중에 10 진수로 사용할 수 있음

(1) 명시적 형변환

byte a = (byte)(int 값);

(2) 암시적 형변환

long l2 = 1014L;

3.3.16.2 Floating

float = 1.4E-45 ~ 3.4028235E38 까지

double = -4.9E-324 부터 1.79769313486231 57E308 까지(default)

실수형 변수는 소수 이하의 표현도 가능하므로 정밀한 수를 담기 위해 사용

형변환 : float f = (float)(1.014*2);

*.단지,int 나 float 같은 기본형의 데이터 유형은 서로간의 상속관계가 없으므로 형변환하는게 불가능합니다.

3.3.16.3 Logical

Boolean – 1 바이트의 크기를 가짐,형변환이 되지 않음

3.3.16.4 Textual

char – 2 바이트의 크기를 가짐

정수로 형변환시 – 0~65535 까지 범위를 지닌 정수가 됨

3.3.17 Non-Primitive type || Reference Types

Call by Reference : 주소를 넘겨준다.

실제 object 를 저장할 영역을 'new'를 사용해서 만듬

[ex]String

String 이 primitive data 형처럼 생성될 경우 동일한 문자열이 있다면 같은 것을 가리키지만 new 라는 것에 의해 reference 로 생성이 되면 항상 새로운 것을 pointing 한다

3.3.18 field

데이터 멤버는 객체참조가 될 수도 있고 기본형이 될 수도 있다.

객체참조의 경우는 new 를 사용해서 실제 객체를 가리킬 수 있도록 초기화

기본형의 경우는 정의와 동시네 바로 초기화

각 객체는 자신의 데이터 멤버를 저장하기 위한 메모리를 갖는다.

따라서, 데이터 멤버는 객체간에 공유되는 것이 아니고 각 객체마다 따로 생긴다.

3.3.19 Signature (메소드 시그니처)

메소드의 이름과 인자 리스트는 각 메소드를 고유하게 식별하는 기준

3.3.20 name space (명칭 공간)

프로그램의 각 부분이 자리잡고 있는 공간에 일정한 명칭이 붙어 있는 것

3.3.21 aliasing

두개 이상의 객체 참조가 같은 객체를 참조하는 것을 앨리어싱이라 한다.

앨리어싱은 메소드의 인자로 객체참조를 전달 할 때도 생긴다.

3.3.22 Operator

{}

{}모든 연산자의 으뜸,가장 먼저 수행

3.3.22.1 unary Operator (단항부호연산자)

-음수,+양수

3.3.22.2 assignment Operator (지정연산자)

오른쪽 값을 구해서 왼쪽에 넣어라

3.3.22.3 mathematical Operator(산술연산자)

+ - / *

3.3.22.4 증감연산자

prefix(전치)

postfix(후치)

++

a++ : a=a+1;

3.3.22.5 relational Operators

boolean 값 산출

!

! : boolean 값에 대한 부정

3.3.22.6 Logical Operators and Short-Circuit Logical Operators

boolean 피연산자간의 논리적인 관계를 바탕으로 boolean 값을 산출한다

논리연산자를 사용하면 “숏 서킷(short_circuit[9])”이 적용

[9] short_circuit : 앞 수식의 결과가 전체 수식의 결과를 확정짓는 값(true or false)일때는 나머지

?:

조건 ? 조건이 true : 조건이 false

조건이 true 이면 가운데 값을 취하고 조건이 false 이면 맨 뒤에 값을 취합니다.

OP=

OP 란 바로 operand 를 뜻합니다.즉,여러가지 연산자와 '='가 붙어서 하나의 연산자를 이룬다는 뜻입니다.

==는 주로 primitive data type 간의 비교를 위해 사용한다
equals() method 는 대부분의 class 에서 값의 비교가 아닌 내용을 비교하게끔 재정의

3.3.22.7 bitwise Operators

비트연산자는 양쪽 피연산자의 대응되는 각 비트별로 부울연산을 하여 결과를 산출한다. EXCLUSIVE or XOR 하드웨어적 시스템 프로그래밍을 하기 위해 사용. 많이 사용하지 않음.TV 셋톱박스 내장 프로그램

~

~ : 비트[10]의 보수를 계산,현재 널리 쓰이고 있는 모든 컴퓨터는 0 과 1 로 나타내는 2 진수로 사용,모두 반대로 만들어 줌

^

^ 연산자는 XOR 연산을 하는데 양변의 값을 비트로 나타내었을 때 같은 위치의 비트를 비교해서 서로의 값이 다를 때 그 결과 값이 1 이 되는 연산

|

OR 연산은 두 비트 중 하나만 1 이어도 결과값이 1 이 되는 연산.

10 은 1010 이고 12 는 1100 이므로 서로 하나의 비트만 1 이어도 1 인 OR 연산의 결과값은 1110 이 됩니다.110 은 10 진수로 14 입니다.

3.3.22.8 Shift Operators

자바의 기본 정수 타입은 반드시 부호(sign)가 있는 정수 값을 가지며 제일 왼쪽의

수식을 연산하지 않는 것

[10] 비트(bit) : binary digit 의 약자로 0 과 1 로 이루어진 숫자를 뜻합니다.

1 비트에 표시한다.

이 비트를 MSB(Most Signficant Bit)라하며 양수일 때는 0,음수일 때는 1 이다.

자바에서 정수타입의 값은 부호가 있는 2 의 보수(two's complement)형태를 취한다.

3.3.22.9 Ternary Operators (삼항 연산자)

자바에서 유일한 3 항 연산자

if-else 연산자라고도 불린다.

boolean 수식? value0 : value1

3.3.22.10 Cast Operators

축소변환이라고도 불림(narrowing conversion)

데이터 값의 손실이 생길 수 있다

자바의 모든 캐스팅은 실행하는 시점에서 type 의 적합성 여부가 검사된다.따라서 객체를 올바르지 않은 type 으로 캐스팅하려고 하면 ClassCastException 예외가 발생한다.

프로그램의 오류를 자동으로 잡아주는 것이다.

이처럼 실행시점에서 타입을 검사하는 것을 런타임 타입 확인(run-time type identification:RTTI)이라 한다.

3.3.22.11 Coma Operators

자바에만 있는 것으로 for 문에서 사용

for 문의 초기화 수식과 증가식에는 여러 개의 명령문을 콤마(,)로 연이어 지정

for (int i=1 , j=i+10 ; i<5 ; i++, j=i*2)

for 문의 초기화 수식에는 변수를 여러 개 정의할 수 있지만 데이터타입은 같아야 한다.

3.3.22.12 String Concatenation With +

```
String s =  "Dr.";
String name = "Pate" + "" + "Seyour";
String title = s + name;
```

[result] Dr. Pete Seyourr

~연산자를 통해 2 의 보수화한 것	Integer.toBinaryString()를 사용해 2 진수로 표현된 스트링으로 바꾸어준 것

byte i = 104;
System.out.println(i);
i = (byte)~i;
System.out.println(i);
	byte i = 104;
	System.out.println(Integer.toBinaryString(i));
	i = (byte)~i;
	System.out.println(Integer.toBinaryString(i));

104 105	1101000(앞에 0 이 생략) 11111111111111111111111110010111

3.3.22.13 연산자 우선 순위

연산자 종류	연산자
단항연산자	+ - ++ --
산술연산자와 쉬프트연산자	* / % + - << >>
관계 연산자	> < >= <= == !=
논리연산자와 비트연산자	&& || & ^ |
지정연산자	=과 *=등과 같은 복합지정연산자

3.4 Arrays

배열은 특정타입만을 저장한다. 타입이 올바른지 컴파일 시점에 확인한다.

배열은 객체로서, 다른 객체의 참조를 저장한다.기본형 데이터 타입의 배열은 직접 그 값을 저장한다.

배열은 초기화 구문으로 자동 생성될 수 있으며, 또는 new 연산자를 이용하여 명

시적으로 생성할 수도 있다. 배열은 자동초기화 된다.

배열은 하나의 객체

첫번째 생성 방법 : int [] 배열 = {1,2,3,4,5}

두번째 생성 방법 : int[] 배열 = new int[5];

배열[0] = 1;

배열[1] = 2;

…

첨자는 항상 0 부터 시작한다는 사실

형변환 : 클래스 A 가 있고 클래스 B 가 있는데, B 는 A 를 상속한다고 가정합시다.

이 때,B 배열은 A[]배열참조변수로 참조할 수 있습니다.

물론, 이렇게 만들어진 A[]배열참조 변수는 B[] 배열참조 변수로 형변환해서 사용할 수도 있습니다.

초기값을 {}로 둘러싸면, new 를 사용했을 때와 동일하게 메모리가 할당된다.

*.shallow copy 셀로우 복사 : 객체 배열을 복사할 때는, 배열에 담긴 객체 참조가 복사된다.

3.4.23 Objectives Arrays

object 의 array 와 array object 를 혼동해서는 안 된다. object 의 array 는 array 의 element 로 object 가 들어가 있는 것을 말하고, array object 는 array 차체를 의미한다

3.4.24 Multidimensional Arrays

inr [][] 다중배열 = {{1,2},{3,4,5},……}

int [] a,b[];

int [] a 와 int[][] b 를 선언한 방식, 지양 코딩방식

3.5 Branching Statements (제어문)

for (int i = 0; i < 10; i++) - for block 안에서만 사용되는 변수를 선언

switch 조건에 올 수 있는 변수의 type 은 byte, char, int

switch 문은 선택문이라고도 한다

do-while 문

```
do {
        System.out.println(i);
         i++;
} while (i < 10);
```

do-while statement 는 while 문과 다르게 do block 을 반드시 한 번은 수행한다. 그리고 나서 조건을 check 하고 loop 를 돌면서 작업할 것인지 여부를 결정한다.

3.5.25 label

label 은 식별자(identifier[11])로서 이름이 있으며 끝에 콜론(:)을 붙인다.

label 의 적합한 위치는 반복문 바로 앞이다.

반복문이 nested(중첩) 되어 있어서 포함된 안쪽 반복문에서 포함하는 바깥쪽 반복문으로 제어를 옮길때만 라벨을 사용한다.

[11] identifier : class 명, method 명, 변수명 등, 이름을 필요로 하는 것들이 가지고 있는 이름들을 통틀어 identifier 라고 한다

4. Object Oriented Programming

4.1 Class 와 Object 와의 관계

4.1.1 Class

하나의 user 가 직접 만들어 사용하는 일종의 틀,어떤 객체를 만들어야 하는 지 기술한 설계도

4.1.2 Object

(1) 모든 것이 객체다. 객체는 우리의 “요청(request)”에 의해 자신이 맡은 일을 수행할 수 있다.

(2) 프로그램은 여러가지의 객체들로 구성되고, 각 객체들은 상호간에 메시지를 보내어 일을 수행한다.

(3) 각 객체는 다른 객체를 포함하여 자신의 메모리를 가질 수 있다. 복잡함을 단순화 시킨 객체들을 이용하여 더 복잡한 프로그램의 작성이 가능하다.

(4) 어떤 객체든 자신의 클래스 or type 이 있다. 객체는 클래스(class)의 인스턴스(instance)다.

(5) 같은 클래스에 속하는 모든 객체는 동일한 메시지를 받을 수 있다. 대체성(substitutability)

Class 를 인스턴스화 시킨 것,일반적으로 세상의 모든 것이 객체

따라서 class 는 Java 에서 사용되는 object 를 만들기 위한 정의일 뿐이다.

결국 이를 program 에서 사용하려면 object 라는 것을 만들어야 한다는 것이다. 그리고 이때 사용되는 키워드가 바로 'new'라는 것이다. new 키워드를 이용하며, 그 객체의 “참조(reference)”를 지정

객체는 참조로 조작한다.

객체를 “참조(reference)”하는 식별자(identifier)로 조작하는 것이다.

참조를 항상 초기화해야 runtime 에러발생에 안전하다.

객체는 생성한 후 사용한다.

4.1.3 Class 와 Object 와의 비교

Object	class
· instance of class (memory 에 올라가는 부분) · Object is unique. · variable + method 로 구성 (= behavior= member method = function = action)	· template for Object · source code (program text 상에서 볼수 있다) · attribute + behavior 로 구성 (= variable = member variable = field = data = information)

4.1.4 Accessing Object Members

dot 를 이용해 object 의 member 들에 접근하는 표현을 일컬어 dot notation 이라고 한다

4.2 Access Modifier

4.2.5 접근 지시자의 종류

public

private

friendly(default)

: 클래스끼리는 상호간에 접근이 가능하지만, 다른 패키지의 클래스에 접근할 수 없다는 의미

protected

: friendly 의 일종

4.2.6 접근 제어의 필요성

클래스를 사용할 때 자신들이 어떤 인터페이스를 할 수 있는지를 public 멤버를 보면 명확히 알 수 있다.

클래스 사용자에 어떤 영향을 주지 않고 라이브러리 작성자가 클래스 내부를 수정할 수 있다는 것이다.이렇게 되면 클래스를 만든 이후에도 지속적으로 발전시켜 갈 수 있는 것

인터페이스와 구현부분을 명확하게 분리해야 하며,그럼으로써 클래스를 수정했을 때 사용자가 자신들의 코드를 다시 작성하는 불상사를 미연에 방지할 수 있다.

4.3 Object-Oriented Programming Language (OOPL)의 3 大 concept

OOPL 의 3 대 concept 중 Inheritance 와 polymorphism 을 지원하지 않는 언어를 객체 기반 언어(Object Based Language)라고 함

ex) 자바 스크립트(Javascript)

4.3.7 Polymorphism

서로 다른 타입의 객체들이 같은 메시지에 대해 제각기 다른 결과를 산출하는 개념

상속관계에서 부모를 가리킬 수 있는 모든 것(다만, constructor 는 제외됨)을 자식 class 에게 주는 것. extends 라는 keyword 를 이용한다.

각각의 Object 가 동일 message 에 대해서 서로 다르게 또는 적절히 반응하는 것

- 어떤 하나의 클래스에서 상속을 받았는데 override 된 것이 있으면 override 된 것이 묶여서 반응하고 override 되지 않은 경우 parent class 가 묶여서 이것을 이용하여 계산이 이루어진다.

파생 타입의 객체를 처리할 때, 그 객체의 타입을 베이스타입으로 지정하면 유리할

때가 많다. 특정, 파생 타입에만 국한되지 않는 코드를 작성할 수 있기 때문

4.3.8 inheritance

부모 class 형태로 데이터와 코드를 묶어놓고, 내부의 데이터 중 외부에서 접근할 수 없는 형태로 구현하는 것이다.

4.3.8.1 substitution principle(대체 원리)

증분적 개발(incremental development) : 기존 코드에 영향을 주지 않으면서 새로운 코드를 만들어 갈 수 있더,새로운 결함이 나타나더라고 새 코드에서만 난다.

“is a” relationship (상속관계)라고도 불린다.

모든 파생 클래스들은 자신들의 베이스 클래스와 같은 인터페이스를 가지므로, 파생클래스는 베이스클래스와 같은 타입이 된다. 결과적으로 베이스클래스의 객체를 파생클래스의 객체로 대체 할 수 있다.같은 type 이므로 같은 interface 를 갖는다.

표현이 is kind of(~의 한 종류이다)가 가능하다면 상속관계(extends)로 잡는 것이 가장 효과적

4.3.8.2 composition

코드의 재사용(reuse)

“has a” relationship(embedded 된 관계)라고도 불린다.

기존 클래스들을 조립하여 새로운 클래스를 만들 수 있는 것

“has-a”관계로 부르며

컴퓨지션은 클래스간의 관계라기 보다는 객체간의 관계라고 할 수 있다.

즉, 개념적인 관계가 아니라 실제 객체간의 포함관계

컴포지션을 이용하면 코드의 유연성이 좋아진다.

표현이 is part of, is composed of 가 가능하다면,한 class 를 또다른 class 의 member(member variable)로 잡는 것이 효과적이다.

상속은 컴파일 시점에서 관련 클래스가 정해지므로 컴포지션에 비해 코드의 유연성이 떨어진다.

*.용어 정리

상속되는 클래스	상속받는 클래스
부모(parents)	자식(child)
베이스(base)	파생(derived)
Super	Sub

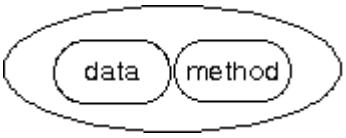
4.3.8.3 pure substitution(순수 대체)와 is-like-a 관계

상속을 받는 파생클래스에서 새로운 함수를 추가하지 않고 베이스 클래스에 있는 함수들만 오버라이딩한다면, 파생클래스는 베이스클래스와 100% 똑 같은 타입이라고 할 수 있다.따라서, 베이스 클래스의 객체는 파생 클래스의 객체로 완벽하게 대체할 수 있다.그러나, 파생타입에 새로운 인터페이스를 추가해야 할 경우도 많다. 새로운 타입은 여전히 베이스타입을 대체할 수 있지만, 100% 똑같지 않으므로 완벽한 대체는 할 수 없다.이것을 is-like-a 관계라 한다.

4.3.8.4 upcasting

파생클래스의 객체가 베이스 클래스 타입으로 변환되는 것(반대 : downcasting)

4.3.9 encapsulation

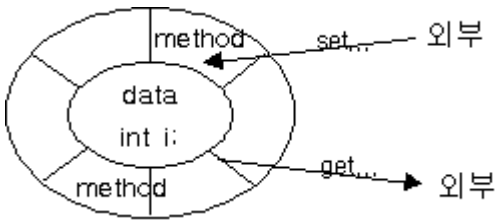


Encapsulation → data 와 그것을 handling 할 수 있는 method 를 묶은 것

⇒ Information Hiding 효과를 얻기 위하여 Encapsulation 을 함.

(중요한 data 를 보호)

· fully Encapsulation



- variable ⇒ private

- method ⇒ public

다른 데이터를 가지고 위에서 있는 데이터 값을 바꾸지 못하도록 하고 데이터를 handling 할 수 있는 method 를 정의해놓고 이를 통해서만이 접근하도록 하는 방식, 즉 int i;라고 데이터 값을 정의하고 이를 사용하기 위해서는 set.. 이나 get.. 이라는 method 를 작성하여 외부에서 만일 data 를 사용하려면 set.. 이나 get.. method 를 사용해야한다

5. Java Basic Grammar

5.1 Construct

new 를 통해서 instance 를 만들어 낼 때만 실행 되어지는 함수
default constructor : constructor 가 하나도 없을 때 컴파일러가 자동으로 생성
역할 : member 초기화
자바에서는 parent class 가 먼저 초기화된다.
Constructor 는 상속에서 제외된다.
따라서 Constructor 는 overriding 될 수 없다.(overloading 은 가능)
상속에서 제외되는 2 가지 : Constructor, private method
overload 된 또 다른 constructor 를 호출 : this( )
this( ); 는 constructor 의 맨 첫 번째 줄에 와야한다.
super( )도 마찬가지
또 다른 constructor 를 생성하면 default constructor 를 잃게 된다.
따라서 default constructor 를 만들어 줘야 한다.

5.2 abstract

한 물체를 대표하는 명사와 동사를 추출해 내는 것을 전산에서는 흔히 '추상화'라는 단어로 표현
abstract method 를 하나라도 가지고 있는 class 는 abstract class 로 정의해야 한다. 여기서 주의할 사항은 abstract class 는 instance 가 될 수 없다
interface 라는 것은 일종의 abstract class 인데, 이것이 가지고 있는 method 들은 모두가 반드시 abstract method 들이어야 한다
다시 말해서 abstract method 만 가지고 있는 abstract class 가 interface 인 것이다
interface 도 일종의 class 임을 기억하자
abstract 클래스는 private 가 될 수 없습니다.

5.3 Overriding

method 이름, return type, parameter 의 수와 type 이 일치

반드시 상속관계가 있어야 한다.

같은 method 이지만 ⇒ 선언부 일치

하는 일이 다르다 ⇒ 구현부가 다르다

상속간계에서 부모로부터 받은 메서드를 사용하지 않고 같은 이름의 메서드를 재정의 해서 사용하는 것이다. 이때, 메소드의 이름은 물론, return type, argument list 까지 모두 부모의 것과 같아야 한다.

5.3.1 closure

클로저는 호출가능한 객체로, 자신이 속해있는 유효범위내에서 생기는 정보를 가질 수 있는 것을 말한다.

5.3.2 call back

포인터 메커니즘을 자바에 넣기 위해 고려된 것을 콜백을 이용하면, 어떤 객체가 다른 객체에 대한 정보(참조와 같은)를 갖고 있어서 언젠가는 그 객체에게 콜백 할 수가 있다.

5.4 Overloading

class 내에서 같은 이름의 메소드를 여러 개 구현하는 기법. 이때 메소드 이름은 반드시 같아야 하고, return type 은 같아도 되고 달라도 된다. 다만, argument list 는 반드시 달라야 한다.

생성자에도 오버로딩이 존재한다.인자의 차이로 하나의 클래스에 생성자가 여럿 존

재할 수 있다.

5.5 Constructor

생성자, 즉 object 가 생성될 때, 반드시 수행되는 코드. 상속되지 않는다.

5.6 toString()

java.util 에 있다.

converts an object to a String

toString() method 도 역시 Object class 가 가지고 있다. 따라서 모든 class 가 소유하고 있다. 이는 자신의 instance 를 String instance 로 바꾸는 데 사용된다.
사실 화면에 출력하려면 문자(String)형태로 변환되어야 하지 않겠는가?

5.7 Inner class

JDK 1.1 에서부터 지원

event handling 때문에 나온 개념 (event 를 처리해주는 것)

outter class 의 모든 member 를 자유로이 사용할 수 있다는 장점이 inner class 를 사용하는 가장 큰 이유
outter class 의 instance 가 먼저 생성된 후, inner class 의 instance 가 생성
inner class 는 method 안에서도 정의 가능하다. 그러나 이 때 이 inner class 가 access 할 수 있는 변수는 method 안의 final 변수만 가능하다

선언과 생성을 동시에 한 것을 anonymous inner class 라고함

5.8 Wrapper Class

자바에서는 기본형만 빼고는 모두 객체다.따라서, 메소드를 호출할 때 인자로 객체를 전달해야 하는 경우가 많이 생긴다.

· primitive data type 들을 wrapping 하고 있는 class

Primitive Data Type	Wrapper Class
boolean	Boolean
byte	Byte
char	Char
short	Short
int	Integer
long	Long
float	Float
double	Double

“바깥 클래스.this”라고 표현하면 바깥 인스턴스를 직접 가르키게 됩니다.

5.9 Local class

Local 클래스는 어떤 지역에 속한 클래스로 한 지역 내에서 정의된 임시적인 성격을 가지는 클래스를 말한다. 여기에서 지역이란 메소드를 일컫는다.

```
Public class LocalClass
{
        public void testLocal(final int I)
        {
                class Test
                {
                        public void callMe()
                        {
                                System.out.println(“i=”+i);
                                System.out.println(getClass())
```

```
                    }
                }
        Test test = new Test();
        Test.callMe();
       }
        public static void main(String args[])
        {
                LocalClass local = new LocalClass();
                Local.testLocal(100);
        }
}
```

[출력결과]

i = 100

classLocalClass1$Test

Local 클래스의 가장 큰 특징은 메소드 안에 클래스가 존재한다.

TestLocalClass()메소드를 보면 final 이라고 명시된 변수는 맨 처음 대입된 값 이외에 다른 값을 넣을 수 없다.final 로 만들면 다른 곳에서 참조할 때 그 변수를 가져다가 사용하지 않고 복사해서 사용함.결국 메소드가 소멸하든 말든 Test 인스턴스 내부에서 안전하게 쓸 수 있음

Test 클래스의 정확한 이름은 LocalClass1Test

일단,메소드 안에서 생성된 클래스는 임시클래스의 성격을 지님.그러므로 메소드안에서 선언된 상수와 같은 성격을 띄므로 이 임시클래스의 이름에 “$”를 붙이고,번호를 붙이게 된다.

5.10 Anonymous class

Anonymous 란 익명을 뜻합니다.즉, 이름이 없는 “아무개”클래스입니다.

```
Public class AnonymousClass
{
        public void testAnonymousClass()
        {
```

```
                Object obj = new Object()
                {
                        public String toString();//java.lang.Object 의 메소드인데
오 Anonymous 클래스가 오버라이딩 한 것
                        {
                                System.out.println(getClass());
                                 System.out.println(AnonymousClass.this.getCla
                ss());
                                Return super.toString();
                        }
                };
                System.out.println(obj);//이 객체를 출력하면 toString()호출
        }

        public static void main(String args[])
        {
                AnonymousClass local = new AnonymousClass();
                Local.testAnonymousClass();
        }
}
```

[출력 결과]

class AnonymousClass$1
<- testAnonynousClass()메소드 안에서 임시로 만든 Anonymous 클래스의 클래스형을 출력한 것
class AnonymousClass
<- AnonymousClass.this.getClass())메소드는 내부인스턴스에서 바깥 인스턴스를 지칭하려면 바깥 인스턴스의 클래스형에 this.를 붙여서 쓴다
AnonymousClass$1@f63f9263
<-return super.toString()은 Super 클래스가 java.lang.Object 이므로 기본적인 출력을 함

5.11 Collection API

Set : without ordering, without duplication

List : ordering(순서,정렬)이 되고, duplication(복사)을 허락

대표적인 예) Vector class : dynamic array 의 개념

(1) Collection(인터페이스)는 객체모음의 최상위 Data Type 이다.

(2) 하위 level 로는 List(순서가 있는 모음), Set(중복이 없는 모음) 등이 있다.

(3) 셀내의 컴포넌트들이 다른 방식으로 정렬 가능하다.

(4) 콜렉션 내부 엘레먼트를 위해 Iterator(단방향, 제거), Enumeration(단방향), ListIterator(양방향, 추가, 변경, 제거).

(5) 키와 객체를 연결하기위해 Map 객체가 지원됨.

(6) 이외에도 BitSet, Stack, Dictionary 등이 지원됨.

5.11.3 Collection interface

Collection 이란 여러 object 를 element 로 가지고 있는 object 를 일컫는 말
Collection 의 종류 - Array,Vector, Bits, BitSet, Stack, Hashtable, LinkedList 등

5.11.4 Set interface

Set : 중복없는 객체 집합.

(1) AbstractSet

(2) HashSet

-. Hashing 을 사용하여 구현되므로 메소드의 수행시간은 일정 - 크기가 큰 집합에 유리.

-. 엘리먼트로 null 을 허용하지 않는다.

(3) BitSet

-. 비트의 배열로 구현되므로 0 부터 시작하는 인덱스로 접근할 수 있다.

Object clone().

(4) TreeSet

-. 키값을 가지고 엘리먼트를 오름차순으로 정렬한 집합.

-. 키 :

Comparable 을 implements 하고 서로 비교 가능해야 함.

Comparator 을 이용, SortedMap 을 만듬 - 비교 기준이 됨.

5.11.5 List / ListIterator interface

List : 순서가 정해져 있는 객체 집합.

ListIterator : List 에서 엘리먼트를 양방향으로 이동, 검색, 추가, 제거, 변경이 가능.

(1) AbstractList

(2) ArrayList

-. 배열형태로 구현된 리스트.

(3) Vector

-. 자바 1.1 과의 호환성을 위해 제공됨.

-. ArrayList 를 사용하는 것이 좋다.

(4) LinkedList

-. 스택, 큐, 디큐로 사용 가능한 메소드를 제공.

-. 자바 1.1 의 Stack 을 대신해서 사용.

5.11.6 Map interface

Map : 키와 값의 쌍으로 이루어져 있는 집합.

자바 1.1 의 Dictionary 대신 사용.

(1) 존재, 크기.

(2) 검색, 비교, 복사.

(3) 추가, 제거.

(4) Map.Entry 인터페이스.

-. Map 에 저장된 하나의 키-값을 가지는 객체.

(5) AbstractMap, HashMap 등.

Hashtable 클래스 :

Properties 클래스 :

변경 불가능한 자료 구조 :

기타 :

ex) Vector Class

5.12 Reflection API

(1) source code 가 아니라 binary code(.class 파일)로부터 정보를 얻어옴

(2) Java beans 를 만들 때 사용

(3) java.lang.class 가 Reflection API 의 중추적인 역할 담당

5.13 Vector

Vector 란 일종의 array 이다. 또한 Vector 에는 모든 object 를 담을 수 있다. 더군다나 Vector 는 그 크기가 자유로이 변경될 수 있다

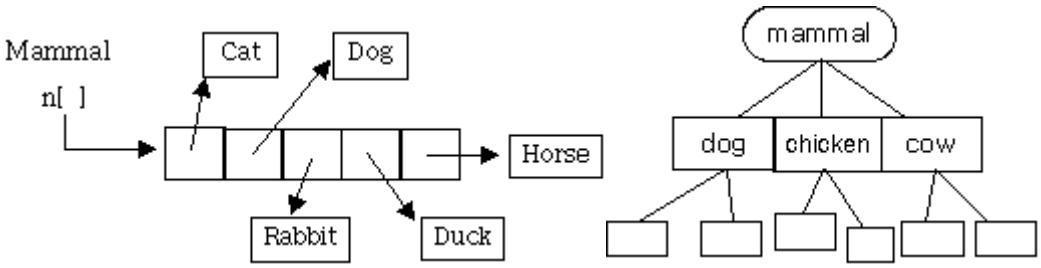
5.14 Heterogeneous Collections

super class type 으로 선언된 reference 는 자기 자신의 type 으로 만들어진 객체

또는 sub class type 으로 만들어진 객체를 가리킬 수 있다.(상속관계에서)

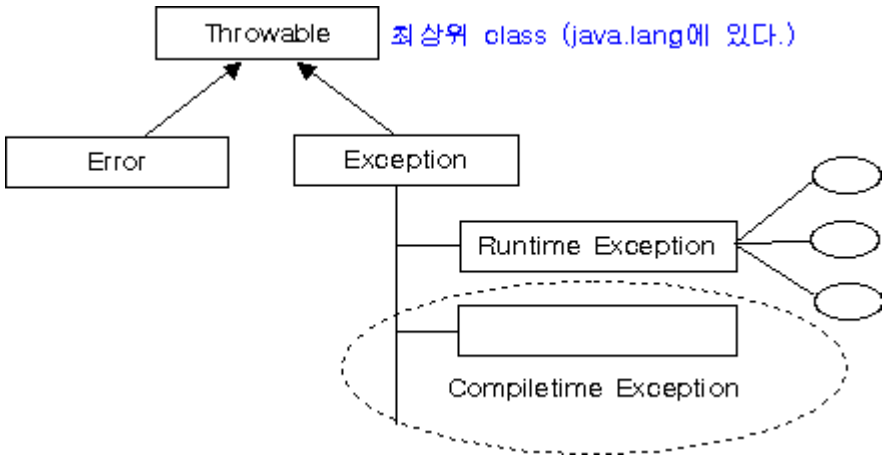
다양한 sub class 를 super class 에서 묶어서 관리

container 기법



- 하나의 배열안에서 Dog 타입, Cat 타입, Horse 타입 등 여러 type 으로 되어 있지만 하나의 Mammal 안에 배열로 만들어 놓고 이를 다시 sub class 로 만들어 놓고 이를 가릴킬 수 있다. Heterogeneous Collections 때문에 형변환에 관련된 문제도 발생할 수 있다.

5.15 Exception



Error : 복구 불가능

Exception : 에러 상황

mild type 의 에러 (복구가능)

Runtime Exception :

프로그래머의 잘못으로 발생. 컴파일은 되고 실행은 안됨

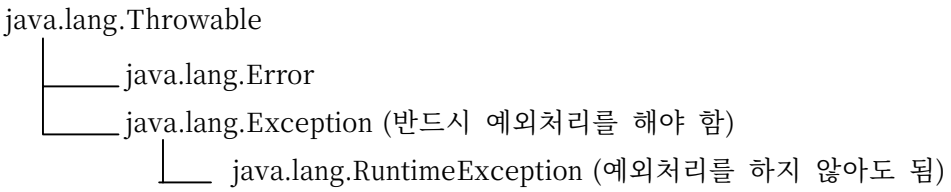
0 으로 나눈 경우, array 에서 범위를 넘어선 경우

compiletime Exception :

프로그래머가 잘못한 것이 아니라 사용자가 잘못해서 발 생하는 exception. 주로

input, output 시에 발생 (없는 파일을 열거나 저장할려고 할 경우)

컴파일 되지 않는다. 따라서 end user 에 의해 발생할 수 있는 에러를 교정해야함



Throwable 클래스는 Exception 클래스와 Error 클래스의 베이스클래스로, 자바의 모든 예외나 에러의 최상위 클래스이다.

모든 예외 클래스는 Exception 클래스의 서브클래스이므로, 어떤 예외객체도 exception 타입이기 때문에

```
try
{
        예외가 발생될 수 있는 코드
}catch 잡을 예외의 종류
{
}finally
{
        예외 발생 유무에 관계없이 실행되는 코드
```

}

catch 문은 특정 타입의 인자를 하나만 갖는 작은 메소드와 같다.
catch 로 잡은 예외보다 하위의 예외를 그 뒤에서 잡지는 못한다.이러한 경우에는 하위의 예외처리에 대한 코드를 먼저 써주어야 한다.

숫자에 관한 예외상황 발생 : java.lang.ArithmeticException
이 코드는 무조건 실행
즉,자바의 예외처리 메커니즘은 애플리케이션 실행시 주의를 요하는 조건을 처리하는데 사용할 수 있는 매우 강력하고 유연한 기능이다.

예외처리기(exception handler) : 에러의 검사와 처리를 하는 부분 - catch 키워드
예외 상태(exception condition) : 메모리의 실행을 중단시킬 수 있는 문제

메소드에 생긴 예외를 throw 하는 경우는 반드시 이와 같이 throws 키워드를 정의해 주어야 한다.메소드에서 생긴 예외를 자체처리하지 않고 호출한 곳으로 넘기기 때문

5.16 Handle or Declare

Exception 은 Handle or Declare 되어야 한다.

Handle 의 의미

try {

exception 이 발생 할 수 있는 code

} catch (발생한 Exception) {

exception 에 대한 처리 code

}

⇒ 이처럼 try block 로 감싸진 코드를 protected code 라고 함

try 구문 안에 여러 개의 catch 문이 올 수 있다.

이때 specific 한 exception 이 위에 오고, wide 한 exception 이 아래쪽에 오도록 만든다.

Declare 의 의미

선언부

~ throws OOException { ..... }

cf) throw : exception 을 발생시키는 signal (강제로 발생시킴)

exception 의 handling 과는 상관이 없다.

예) throw new OOException;

5.17 static inner class

(1) static 내부 클래스의 객체를 생성할 때 외곽 클래스의 객체는 불필요

(2) static 내부 클래스의 객체는 외곽 클래스의 객체를 액세스할 수 없다.

내부클래스의 오버라이딩은 되지 않는다.

각 내부클래스는 독자적으로 구현 상속(클래스 상속)을 받을 수 있다.따라서,외곽 클래스가 상속 받은 것과는 별개로 또다른 클래스로부터 상속을 받을 수 있다.

(1) 내부클래스는 자신의 인스턴스를 여러 개 가질 수 있다.그리고, 각 인스턴스는 외곽 클래스의 객체가 갖는 정보와는 무관하게 자신만의 정보를 갖는다.

(2) 하나의 외곽 클래스는 여러 개의 내부클래스를 가질 수 있으면, 각 내부클래스는 동일한 인터페이스나 클래스로부터 서로 다른 방법으로 상속받을 수 있다.

(3) 내부클래스의 객체는 외곽 클래스의 객체와 서로 다른 시점에서 생성할 수 있다

(4) 내부클래스는 외곽 클래스와는 독립적이다.

인터페이스나 내부클래스는 다른 oop 에는 없는 고급 개념

5.18 finally

exception 이 발생하든 안하든 전혀 영향을 받지 않고 어떤 경우에든지 반드시 수행이 되어진다.

(try 구문에 return 이나 break 가 있더라도 finally 구문은 실행됨)

- 단지 System.exit(); 로 시스템이 종료될 때, power off (전원을 꺼버릴 때), finally block 안에서 exception 이 발생할 경우 등에는 수행이 되지 않는다.

break; 하더라도 finally 구문은 실행하고 break 됨.