**객체지향 프로그래밍** | 최윤정 교수님 | Summary 이서현 2276219  
[Chapter 01] week 1: Java 프로그래밍 주요개념

프로그래밍 언어란? / 컴파일 / 가상머신과 플랫폼 / Java의 특징과 응용분야

**컴퓨터 (**메인프레임, PC, 태블릿, 스마트폰, 원칩 컴퓨터) & 소프트웨어 혹은 앱

**프로그래밍 언어의 역할은?** 사람과 컴퓨터의 대화를 가능하게 하는 역할. 사람의 언어와 기계어 사이에서 번역자와 같은 역할. 고급 언어(기계어, 어셈블리어)와 저급 언어로 구분한다. (사람이 소스 코드 작성 🡪 컴파일러가 기계어로 변환 후 실행)

프로그래밍 언어란?

* 프로그램 작성 언어
* 기계어
* 0, 1 이진수로 구성된 언어
* 컴퓨터의 CPU는 기계어만 이해하고 처리할 수 있다
* 어셈블리어
* 기계 어 명령을 ADD, SUM, MOVE 같은 표현하기 쉬운 상징적인 단어인 니모닉 기호로 일대일 대응시킨 언어
* 고급언어
* 사람이 이해하기 쉽고, 복잡한 작업, 자료 구조, 알고리즘을 표현하기 위해 고안된 언어
* Pascal, Basic, C/C++, Java, C#
* 절차 지향 언어와 객체 지향 언어
* 플랫폼과 클래스의 모듈화

컴파일

소스 : 프로그래밍 언어로 작성된 텍스트 파일

컴파일: 소스 파일을 컴퓨터가 이해할 수 있는 기계어로 만드는 과정

소스 파일 확장자와 컴파일 된 파일의 확장자

윈도우 기준 (맥은 리눅스처럼 실행권한이 있음)

C: a.c 🡪 a.obj 🡪 .exe

자바: a.java 🡪 a.class

C++ : a.cpp 🡪 a.obj 🡪 .exe

자바 프로그램 개발 순서

소스 작성부터 실행까지

Java 소스 파일 작성

컴파일러 (javac.exe)로 바이트 코드 파일 (.class) 생성

JVM 구동 명령어 (java.exe)로 실행

자바 소스 파일 (\*.java) 🡪 [컴파일러 javac.exe] 바이트 코드 파일 (\*.class) 🡪 [자바 가상 기계 java.exe] 🡪 바이트 코드를 기계어 변환 => 실행된다.

클래스와 메소드는 블록.

Java 그리고 변화

대표적 자바 개발 툴: 이클립스

IntelliJ

Java 그리고 변화

안드로이드

가상머신 특성상 하드웨어의 영향을 받지 않고 어플리케이션 개발 가능

자바 가상머신과 플랫폼

안드로이드에서 자바의 가상머신 기술을 채택하여 앱 개발에 사용

장점: 하드웨어나 운영체제를 가리지 않고 어디서나 동일한 프로그램을 실행

단점: 최적화된 바이너리에 비해 속도가 느릴 수 있고, 각 플랫폼에서 동작하는 가상머신의 개발이 필요

플랫폼 종속성

플랫폼=하드웨어플랫폼+운영체제플랫폼

프로그램의 플랫폼 호환성 없는 이유

기계어가 CPU마다 다름

운영체제마다 API 다름

운영체제마다 실행파일 형식 다름

인텔 CPU 가진 리눅스 환경에서 개발시 인텔CPU+리눅스 에서만 실행됨

자바의 플랫폼 독립성, WORA

자바 응용 프로그램🡪바이트 코드

자바 가상 기계가 설치된 모든 컴퓨터에서 실행됨

자바 가상머신과 플랫폼

자바 프로그램이 실행되는 특정 환경

다양한 환경의 IT시스템을 구축해야 하는 개발자와 사용자의 운영 환경에 따라 세분화

자바는 가상머신과 API로 구성된 형태

자바 플랫폼 – Java SE, Java ME, Java EE…

EE는 기업 등 대규모 시스템 개발에 적합한 요소로 구성, 대표적인 웹 개발 기술인 서블릿과 JSP를 포함

자바 플랫폼

JDK : 자바 프로그램 개발에 필요한 구성 요소  
JRE : 자바 프로그램 실행에 필요한 구성 요소  
API : 자바 응용 프로그램 개발에 사용하는 라이브러리

Java 의 특징

객체 지향 언어 🡪 Objected Oriented Programming

부품 객체를 먼저 만들고, 이것들을 조합해 전체 프로그램을 완성하는 기법

자바는 처음부터 OOP 개발용 언어로 설계되었다

캡슐화, 상속, 다형성 기능을 완벽하게 지원한다

완성된 프로그램은 부품 객체들의 연결로 이루어져있다. 객체간의 관계는 상속/소유/사용, 부품들은 언제든지 교체가 가능하다

Java의 특징

함수적 스타일 코딩 지원 (람다식 Lambda Expressions) 코드 간결하게 작성기능, 컬렉션 요소를 필터링, 매핑, 그룹핑, 집계 처리시 주로 사용한다

메모리를 자동으로 관리.

사용하지 않는 객체를 자동으로 메모리에서 제거한다. 쓰레기 수집(garbage collection 가비지 컬렉션[\*], GC)은 메모리 관리 기법 중의 하나로, 프로그램이 동적으로 할당했던 메모리 영역 중에서 필요없게 된 영역을 해제하는 기능

핵심 기능 코드에 집중할 수 있도록 하는 기능이다. 다양한 어플리케이션을 개발할 수 있다.

Java의 특징

멀티 스레드 쉽게 구현 가능 (동시에 여러 작업 or 대용량 작업 빨리 처리)

동적 로딩 지원 (미리 객체를 만들어 놓지 않고 필요한 시점에 동적 로딩해 객체 생성) (유지 보수시 특정 객체만 쉽게 수정 및 교체해 사용

막강한 오픈 소스 라이브러리 풍부

자바 프로그래밍에서 기억할 것

* 플랫폼 독립성 (자바 프로그램은 플랫폼에 상관없이 어디서든지 실행된다)
* 클래스로 캡슐화 (클래스 내에 모든 변수(필수), 함수(메소드) 구현해야 함.
* 클래스 안에서 새로운 클래스 (내부 클래스) 작성 가능.
* 소스(.java)와 클래스(.class) 파일  
  소스 파일의 이름과 public으로 선언된 클래스 이름은 같아야 한다  
  Hello.java의 코드헤더는 public class Hello {…}이다

자바 프로그래밍 기억할 것

실행 코드 배포

실행 코드 : 한 개의 class 파일 또는 다수의 class 파일로 구성

여러 폴더에 걸쳐 다수의 클래스 파일로 구성된 경우, jar 파일 형태로 배포 가능

main()메소드 자바 응용프로그램의 실행은 main()메소드에서 시작

하나의 클래스 파일에 하나이상의 main()메소드 있을수없다

각 클래스 파일이 main()메소드를 포함하는 것은 상관없음!!

패키지

관련된 여러 클래지를 패키지로 묶어 관리

패키지는 폴더 개념

Java의 장단점과 활용 분야

자바의 장점: • 간결하면서도 강력한 객체지향 언어이다.

 • 개방형 구조이므로 여러 기술과 융합이 가능하다.

 • 플랫폼에 독립적이므로, 여러 운영체제나 하드웨어에서도 동일하게 실행한다.

 • 많은 오픈소스 프레임워크로 생산성을 증가하고 유지보수 비용을 절감할 수 있다

자바의 단점

 • 처리 속도가 중요한 애플리케이션에는 적합하지 않다.

 • 하드웨어에 연결하여 제어할 수 없기 때문에(가상머신 이용) 직접 하드웨어를 정밀하게 조정해야 하는

프로그램 개발에는 적합하지 않다. ---> 이럴 땐 역시 C를 씁니다.

자바의 태동

 1991년 그린 프로젝트(Green Project)

 선마이크로시스템즈의 제임스 고슬링(James Gosling)에 의해 시작

 가전 제품에 들어갈 소프트웨어를 위해 개발

 1995년에 자바 발표

 목적

 플랫폼 호환성 문제 해결

 기존 언어로 작성된 프로그램은 PC, 유닉스, 메인 프레임 등 플랫폼 간에 호환성 없음

 소스를 다시 컴파일하거나 프로그램을 재 작성해야 하는 단점

 플랫폼 독립적인 언어 개발

 모든 플랫폼에서 호환성을 갖는 프로그래밍 언어 필요

 네트워크, 특히 웹에 최적화된 프로그래밍 언어의 필요성 대두

 메모리 사용량이 적고 다양한 플랫폼을 가지는 가전 제품에 적용

 가전 제품 : 작은 량의 메모리를 가지는 제어 장치

 내장형 시스템 요구 충족

25

 초기 이름 : 오크(OAK)

 인터넷과 웹의 엄청난 발전에 힘입어 퍼지게 됨

 웹 브라우저 Netscape에서 실행

 2009년에 선마이크로시스템즈를 오라클에서 인수

WORA

 WORA(Write Once Run Anywhere)

 한번 작성된 코드는 모든 플랫폼에서 바로 실행

 C/C++ 등 기존 언어가 가진 플랫폼 종속성 극복

 OS, H/W에 상관없이 자바 프로그램이 동일하게 실행

 네트워크에 연결된 어느 클라이언트에서나 실행

 웹 브라우저, 분산 환경 지원

 WORA를 가능하게 하는 자바의 특징

 바이트 코드(byte code)

 자바 소스를 컴파일한 목적 코드

 CPU에 종속적이지 않은 중립적인 코드

 JVM에 의해 해석되고 실행됨

 JVM(Java Virtual Machine)

 자바 바이트 코드를 실행하는 자바 가상 기계(소프트웨어)

바이트 코드와 자바 가상 기계

 바이트 코드

 자바 가상 기계에서 실행 가능한 바이너리 코드

 바이트 코드는 컴퓨터 CPU에 의해 직접 실행되지 않음

 자바 가상 기계가 작동 중인 플랫폼에서 실행

 자바 가상 기계가 인터프리터 방식으로 바이트 코드 해석

 클래스 파일(.class)에 저장

 자바 가상 기계(JVM : Java Virtual Machine)

 동일한 자바 실행 환경 제공

 각기 다른 플랫폼에 설치

 자바 가상 기계 자체는 플랫폼에 종속적

 자바 가상 기계는 플랫폼마다 각각 작성됨

 예) 리눅스에서 작동하는 자바 가상 기계는 윈도우에서 작동하지 않음

 자바 가상 기계 개발 및 공급

 자바 개발사인 오라클, IBM 등

 자바 응용프로그램 실행

 자바 가상 기계가 응용프로그램을 구성하는 클래스 파일(.class)의 바이트 코드 실행

자바의 개발 및 실행 환경

 자바 프로그램의 개발

 여러 소스(.java)로 나누어 개발

 바이트 코드(.class)를 하나의 실행 파일(exe)로 만드는 링크 과정 없음

 실행

 main() 메소드를 가진 클래스에서 부터 실행 시작

 자바 가상 기계는 필요할 때, 클래스 파일 로딩, 적은 메모리로 실행 가능

자바 API(Application Programming Interface)란?

 JDK에 포함된 클래스 라이브러리

 주요한 기능들을 미리 구현한 클래스 라이브러리의 집합

 개발자는 API를 이용하여 쉽고 빠르게 자바 프로그램 개발

 API에서 정의한 규격에 따라 클래스 사용

 자바 패키지(package)

 서로 관련된 클래스들을 분류하여 묶어 놓은 것

 계층구조로 되어 있음

 클래스의 이름에 패키지 이름도 포함

 다른 패키지에 동일한 이름의 클래스 존재 가능

 자바 API(클래스 라이브러리)는 JDK에 패키지 형태로 제공됨

 필요한 클래스가 속한 패키지만 import하여 사용

 개발자 자신의 패키지 생성 가능

JDK : 자바로 개발할 때 필요한 툴 킷!

 JRE(Java Runtime Environment) = JVM + 표준 클래스 라이브러리 : 실행만 할 경우

 JDK(Java Development Kit) = JRE + 개발 도구

 자바 프로그램을 개발하고 실행하기 위해 반드시 설치!!