6.28 - (목요)

소프트웨어(S/W)

1. 시스템 S/W
- os
- driver

2. 어플리케이션 S/W
- 게임이나 앱, 이메일 등등 종류

=스탠드얼론(Standalone)
네트워크 x , 단독 실행

=클라이언트서버(ClientServer)
유지 모수가 위험

네트워크 o , 단독 실행 x / 서버 필요

.자체 정식 규격 : 권장 Client 생성가 필요

-웹 통신 규격 : 범용라우팅 사용, Http&Https

web browser	요청	web server	생성	application
	응답		반납	http 프로토콜 규칙에 맞게 응답

local **remote**

viscode

(간접 기계어가 아님)

java -> **bytecode** -> **jvm** -> **각 os**

* .java 컴파일 (jdk) * .class

JDK 경로 설정 > 환경 변수 JAVA_HOME (시스템 변수)
PATH

'컴파일과 인터프리터

hello.c	컴파일	hello.exe	실행	windows11
	gcc -o hello.exe hello.c			<기계어를 나열한 형식이 다르다.>
리눅스용 c 컴파일러				
		기계어 window+intel cpu		
		hello linux + intel cpu		리눅스

각 os에 맞는 c 컴파일러 // 실행속도 빠르다, 컴파일할때 명령문을 검사하기 때문에 실행 전에 문법오류 잡 수 있다.

같은 소스파일 가지고 -> 각 os 에 맞는 c 컴파일러
운영체제 상관 없이 되는 것 스탠다드 API -> printf (os 상관없이 실행가능)

인터프리터 방식 프로그래밍 **node.js** D

소스파일
소스코드 노출을 원한 해라는 뜻임

hello.js	->읽는다->	자바스크립트 인터프리터		->실행->		window11
자바스크립트 소스	(웹도)	(자바스크립트 플랫아이)				intel cpu
		javascript player runtime		실행할 때마다 명령문을 검사		

소스만있으면 컴파일 파로 안함
소스파일을 읽어서 실행 -> 실행 속도가 느리다
명령어를 실행하기 전에는 명령어에 오류가 있는지 알수가 없다.

web browser

html + css	renderer

	조립기	
javascript	engine	node.js
	runtime	v8 + js engine
	player	
	interpriter	

자바 -> 하이브리드 *Hybird 프로그래밍
개발프레임즈 "write once, run anywhere"

컴파일 방식		인터프리터 방식(컴파일 +인터프리터 방식)			
.java	=컴파일	**.class**	인터프리터 방식	bytecode player	윈도우
소스파일		bytecode		==JAVA RUNTIME	
-문법 오류 검사		가상 기계어로 변환		bytecode player	mac os
-명령문 최적화		소스 코드 노출을 막을 수 있음			
실행 명령수 ▼		프로그램 자산을 모호할 수 있음.		bytecode player	리눅스
실행 속도 ▲					

운영체제마다 해당 운영체제에 맞는 컴파일러를 사용해야하기 때문에 복잡

'Bytecode 실행 원리 구조

.class				
	Load>	JVM	실행>	OS
	명령문 읽어오다)			
	=진짜 기계어로 변환		진짜 기계어 실행>	
			Just-in-Time(JIT)	
			JIT 컴파일	

자바는 JIT 컴파일을 중복시 더 속도 상승 가능.
인도로이도는 .Class 전체를 Bytecode를 기계어로 변환 (설치 실행 전) Just-in-Time(JIt) 실행하기 전 정보만 부분 부분을 RAM에 저장(필요할때마다 실행되기때문에 빠름)
Ahead of time(AOT) 설치하면서 기계어로 다 바꾸기때문에 ROM에 저장(시간이 오래걸림/ 설치 후에는 빠름 ART

'CPU-OS-JDK

APP	**APP**		**APP**
Intel CPU 기계어	ARM CPU 기계어	상용 속도가 비교적 느림	Intel CPU 기계어
MAC OS	**MAC OS**		**Intel -> ARM Emulater**
			Rosetta
INTEL CPU	**M1 / M2 (CPU- ARM 계열)**		**MAC OS**
CISC 계열 CPU	RISC 계열 CPU		**M1 / M2 (CPU- ARM 계열)**
			RISC 계열 CPU

JDK 종류

		OpenJDK		오픈소스
	add			
	oracle	IBM	Amazon	azul
	유료로 추가			
	graalvm			

'자바 문법 : 기본 구조

class				hello	{	
				클래스명		프로그램 진입점(콘솔)
public	static	void		main		
		리턴값 유형		명령어 실행을 시작하는 지점(단 1개) : entry point	(string[] args)	args
명령어형	명령어형			메서드명		

*자바 문법 : 소스파일 - 클래스블록 - .class 파일

Exam01.java 컴파일 -> 하나의 클래스 파일에 여러개의 클래스블록을 만들 수 있다. 하나의 클래스 당 하나의 블록
클래스 블록 이름으로 파일이 만들어짐

class A {}	->	A	.class
class B {}	->	B	.class
class C {}	->	C	.class

		자지 모수가 쉽도록	클래스명 = 소스파일명
			1클래스 1소스파일
		A.java	class A {}
		B.java	class B {}
명령 프롬프트(명령창) = 콘솔		C.java	class C {}
command Line Interface (CLI)			

Exam02.java

public	class	D	{}
public(공개)로 설정된 클래스 블록인 경우 파일명은 클래스 블록 이름과 같아야 한다			
public	class	F	{}

X(파블릭은 한개만 존재해야함) -> 파일명은 파블릭과 동일해야함