* Java.

<ArrayList vs Vector>

ArrayList 와 Vector는 기능적으로 거의 같음.

Vector는 동기화 기능을 가지고 있기 때문에 여러 thread가 동시에 접근하지 못함.

<JVM>

· Java Compiler : Java 소스파일을 컴파일. byte code로 변환.

· Class loader :

$$\begin{tikzcd}
\text{Bootstrap Class Loader.} \arrow[d, bend left] \\
\text{Extension Class Loader.} \arrow[u, bend left, "위임"] \arrow[d, bend left] \\
\text{System Class Loader.} \arrow[u, bend left, "위임"] \arrow[d, bend left] \\
\text{User-Defined Class Loader.} \arrow[u, bend left, "위임"]
\end{tikzcd}$$

~~User-Defined Class Loader~~.

$JAVA-HOME/jre/lib/rt.jar의 class load : JVM이 구동될때 필요한 class

$JAVA-HOME/jre/lib/ext 경로의 java 확장 class load.

$CLASSPATH의 class 즉 사용자가 만든 class 로드

→ 여기까지 찾지 못하면 ClassNotFoundException.

Visible principle → 상위클래스로더는 하위클래스로더가 로드한 클래스 알수 X.

Uniquess principle → ~~하~~상위클래스로더가 로드한 클래스를 상위클래스로더가 다시로드X.

Delegation / Visibility / Uniqueness

· Compile time이 아닌 Runtime에 Class를 로딩.

· class 로더는 Class가 요청될 때 파일로부터 읽어 메모리로 로딩.

· 자바 클래스들은 모든 클래스가 메모리에 올라가지 않고 클래스로더에 의해 필요할 때 올라가게 됨.

· 장단점

- 일부 클래스가 ~~는~~ 변경되어도 전체 어플리케이션을 다시 컴파일X
- 변경 사항이 발생하여도 비교적 적은 작업만으로 처리 → 유연
- 실행시 연결부분에 대한 판단을 해야하므로 속도 측면에는 불리

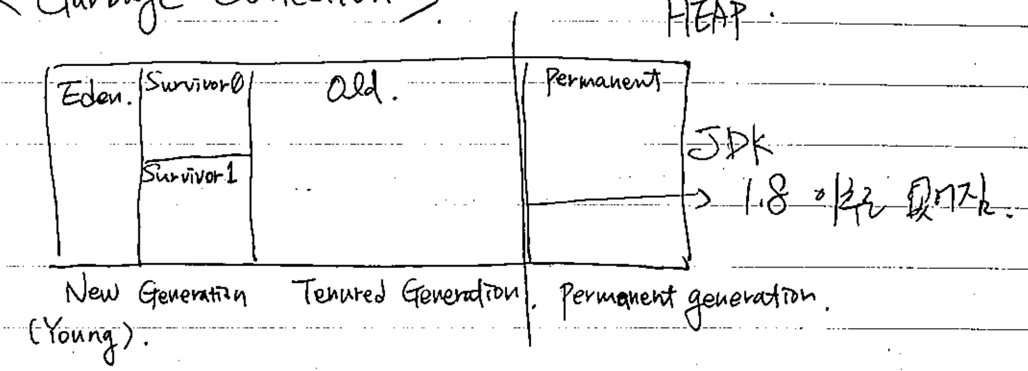
cf.) new vs newInstance

new ↳ Compile-time

newInstance ↳ Runtime

→ 대부분의 객체는 빠른 시간안에 unreachable 상태가 된다.

<Garbage Collection>



· New (Young) Generation : 새롭게 생성된 객체가 위치. ~~New를 이용하여 객체가~~ 생성되면 Eden 영역에 위치하며, ~~GC가 한번 발생한 후 살아남은 객체는~~ ~~Survivor 영역으로 이동.~~ ~~→ Minor GC~~.

Eden 영역이 가득차게 되면 GC가 발생. (minor gc).
살아남은 객체는 survivor∅영역으로 이동.
다음 minor gc 때 eden에서 살아남은 객체는 survivor1 영역으로 이동.
원래 survivor∅에 있던 참조 객체들도 survivor1로 이동. (survivor∅는 clear).
(→ 반드시 하나의 survivor 공간은 clear 상태).

· Old Generation : ~~영향받은~~ 특정 age threshold를 넘은 객체들은 old generation으로 이동. old generation이 가득차면 major gc가 발생하여 garbage를 회수하고 compact 하여 memory를 확보.
→ 시간이 오래 걸리고 실행중 process가 중지된다. (stop-the-world) → 프로그램 실행중 메모리를 바꿀수 없으므로
GC 실행 스레드 이외의 스레드는 모두 멈춘다.

· GC의 종류 : Serial GC - 적은 메모리와 CPU일때 유리
Parallel GC - 많은 자원일때 유리

· GC 튜닝 JVM 옵션
- Xms : JVM 시작시 heap 크기
- Xmx : 최대 heap 크기.
- XX: ~~NewRatio~~ NewRatio : New와 Old 비율
- XX: ~~Eden~~ SurvivorRatio : Eden / Survivor 비율
- XX:+UseSerialGC
- XX:+UseParallelGC
- XX:ParallelGCThreads = ~~@~~ value

< Generic >　　　↳ 클래스의 인스턴스화가 이루어질때.

• 타입을 파라미터화 하여 컴파일시 구체적인 타입이 결정되게 하는 것.

→ since Java 1.5 .

→ 컴파일시 강한 타입체크 가능: 예기치 못한 런타임 에러 방지.

→ 형변환에 의한 성능 저하 방지.

• 배열의 생성에는 왜 generic을 사용할 수 없는가?

→ ~~new 연산자로 배열을 동적으로 생성하기 때문~~

< Java의 특징 >

1. 이식성이 높다. → JVM. 단 JVM은 해당 플랫폼에 맞는걸 써야함.
2. 객체 지향 언어
3. Garbage collector.
4. 클래스 동적로딩.

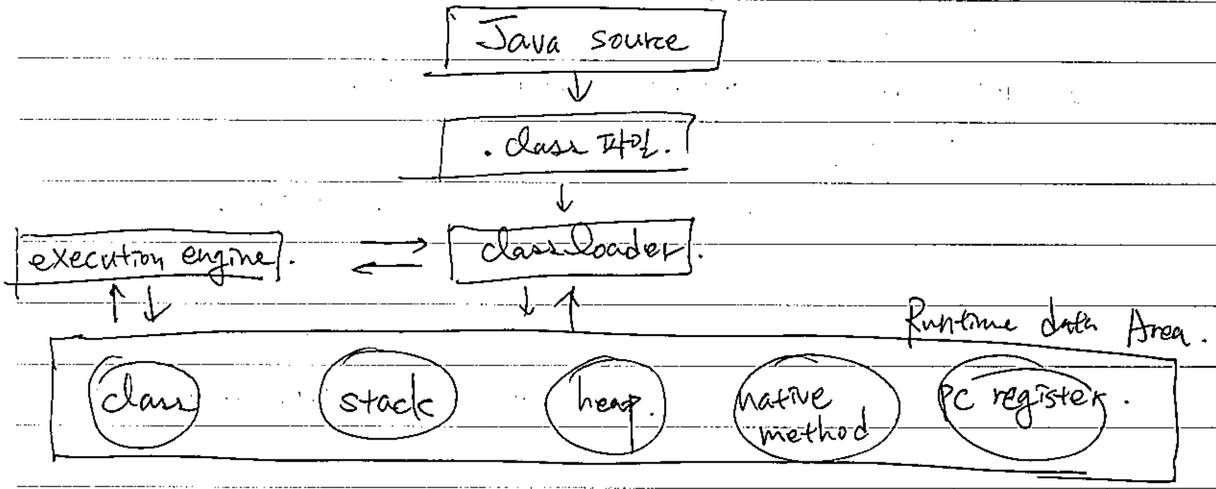
< Java 1.4 → 1.5 >

1. ~~Java~~ Generics 사용 가능.
2. Autoboxing / Unboxing 가능.
   Primitive 타입 ⟷ Wrapper 타입 변경해주지않게 value를 넣고 빼는것이 가능.
3. for each 문 향상
4. StringBuilder 추가.

< Java 1.7 → 1.8 >

1. 람다식.
2. 메타데이터 지원, 보완.   Permanent Generation
3. ~~Hep~~ Heap 영역에서 Perm Generation 제거

* JVM 메모리 구조.



· class 영역 : field, method, type, constant, static 변수의 정보
· stack 영역 : 지역변수, 매개변수, return 값, return 주소
· heap 영역 : new 연산자로 생성된 객체와 배열을 저장.
  Garbage collector 에 의해 관리되는 영역.

→ Permanent 영역 : 다이나믹하게 로드되는 클래스에 대한 메타데이터 및 static 변수나 상수들이 저장되던 공간.
  1.8 기준으로 사라지게 되었는데 메타데이터는 Native 메모리를 이용하게 됨 (metaspace) 개발자는 이 영역에 대한 확보 신경 X 어드레스 OS에 의해 관리

自然別曲 from.13p.

낙고

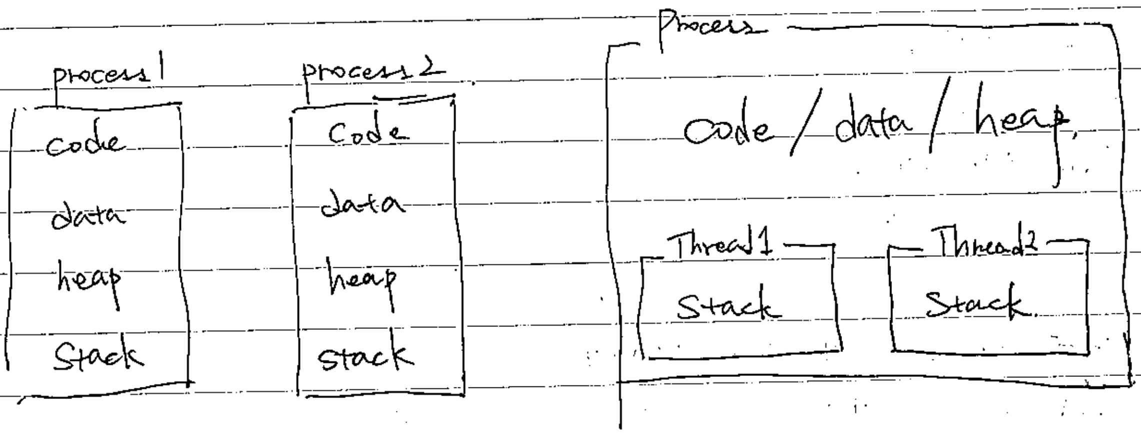
< Process Vs Thread >

* Process

· 컴퓨터에서 연속적으로 실행되고 있는 프로그램.

* Thread

· 프로세스 내에서 ~~실행되는~~ 여러 실행의 흐름.



* 메모리 영역별 용도.

· Stack 영역 : 지역변수, 매개변수, 리턴값, 리턴주소.
런타임에 그 크기를 변경할수 없다.

· Heap 영역 : 동적 데이터 영역. new, malloc() 함수 등에 의해 할당되는 영역. Runtime.

· Data 영역 : 전역변수, 정적 변수, 정적 배열, 구조체 등이 저장.

· Code 영역 : 프로세스가 실행할 코드와 매크로 상수가 기계어 형태로 저장.

morning glory

< 동기 방식 vs 비동기 방식 >.

* 동기 방식 (Synchronous)

직렬적으로 작업을 처리. ~~요청을 하고~~ 요청을 하고 기다렸다가 응답을 받음

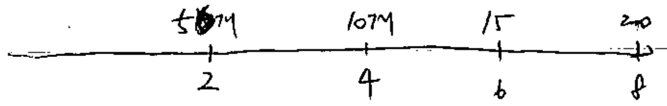
요청을 하는 주체와 응답을 받는 주체가 같음. 결과의 순서가 보장

* 비동기 방식 (Asynchronous) 결과의 순서가 보장되지 않음.

요청 후 기다리지 않고 바로 다음 작업을 처리. 요청을 하고 응답을 받는 주체가 다름.

* Processor 의 수가 같을 때 동기 vs 비동기.

작업을 처리하는데 2초. 프로세서의 수가 5개. 20개의 job.



동기 / 비동기 같다?

Requester 가 여러냐의 차이에.

* 시간당 처리량이나 처리속도의 차이는 크지 않지만

처리중 다른 작업을 할 수있느냐 없느냐, 효율성의 차이.

Snapshot의 경우 동기식으로 처리하면 다음 요청을 받을 수없다 (처리중에는).

Snapshot의 수행시간이 일정하지 않고, 대용량인 것도 있기 때문에

요청자 입장에서는 느리다고 생각될 포인트가 있음.

* OSI 7 Layer.

1. 물리계층 – 데이터를 비트로 바꾸어 전송.
2. 데이터링크 – MAC주소
3. 네트워크
4. 트랜스포트레이어
5. 세션
6. 표현 (presentation)
7. Application.

Preorder → 전위순회 : root부터 방문.
Inorder → 중위순회 : 왼쪽부터 돌음.
post order → 후위순회 : 왼, 오, 중.

< Stateless Vs Stateful >

* Stateful : 이전 동작에 따른 상태나 데이터를 저장하고 다음 수행시 해당 데이터를 사용.

* Stateless : 이전 동작의 상태 및 데이터를 저장하지않고 다음 수행시에도 처음 수행한 것과 마찬가지로 동작.

* Monitoring agent를 stateless 하게 설계하는 이유.
  1. 설계의 단순화 → stateful 한 설계를 위해서는 추가적인 요소가 필요한데 이에 대한 부하를 고려하여 간단한 구조로 설계.
  2. 비즈니스적인 요인으로 인한 모니터링 내용 변경에 따른 작업량 최소화.
  3. 운영자 관리 포인트 ↓ → 크론탭은 PM시 항상 백업후 복구. 상주 process로 stateful하게 수행하면 추가 F/U 필요.

※ 네트워크 OSI 7계층, 서브넷, 파이썬과 PYPY 차이
DI, IoC, Spring FW에서 bean scope

* ~~패키지의 패턴~~ IOC를 이용한 클래스 호출방식.

$$\begin{tikzcd}
\text{Class} \arrow[r, "\text{사용}"] & \text{Interface} \\
\text{~~Factory~~ 조립기} \arrow[u, "\text{의존성 주입}"'] \arrow[r, "\text{생성}"] & \text{Implemented Class} \arrow[u, "\text{구현}"']
\end{tikzcd}$$

FDS 조직 보기 / 회원 캘린 보기
마스터

* IOC → Inversion of Control.
  객체에 대한 제어권이 개발자로부터 컨테이너에게로 넘어감
  의존성을 줄이기 위하여 / 코드 재사용성 ↑
  객체의 생성부터 생명주기까지 모든것을 이러한 컨테이너에서 관리

* Spring Bean Scope @Scope("singleton")
- 스프링은 기본적으로 모든 bean을 singleton으로 생성.
- Singleton : Spring 컨테이너에서 한번생성 → Immutable / RO
- Prototype : ~~bean~~ 모든 요청에서 새로운 객체를 생성 → Mutable/Writable