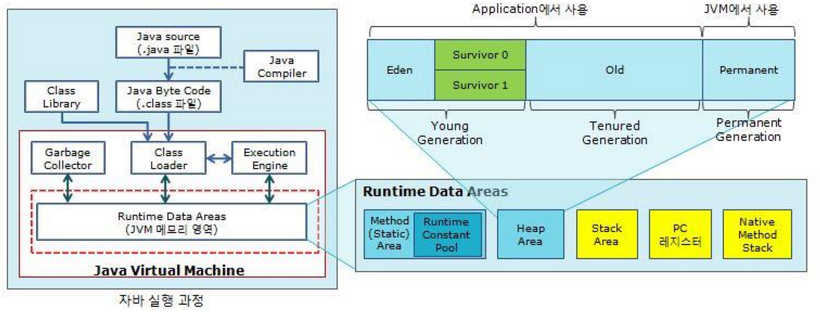
**DEV #4 – 2020.04.16**

>> JVM 메모리 구조를 조사하고 설명하세요.

.java 파일 은 java compiler 에 의해 .class 파일로 컴파일이 된다 이후 class loader 로 .class 파일을 로드하여 excution engine 에 의해 runtiome data area 에 배치된 코드드를 명령어 단위로 실행시킨다.

Runtime data area 중 heap area 는 Garbage Collector 로 인하여 메모리 관리를 한다

data 를 eden(young generation) 에 저장하여 할당한 메모리가 부족할 경우 GC를 이용하여 null 을 반환하거나 사용하지 않는 객체들의 할당을 해제하여 주고 survivor 0과1에 순차적으로 메모리를 정리하는 것으로 메모리 관리에 들어간다(minor GC). Survivor 마저 할당할 메모리가 부족하다면 old generatrion영역에 GC를 해준다.old 영역까지 할당할 메모리가 부족하게 되면 full GC를 실행한다. 하지만 GC의 경우 stop the world 를 발생하기 때문에 메모리 관리시 stop the world 의 길이가 긴 full GC 가 실행을 최소한으로 하기위한 알고리즘을 사용하여야 한다.

Permarnent generation 은 클래스 로더에 의해서 클래스,메소드 등에 대한 메타 정보가 저장되는 영역이다.

Jdk8 버전 부터는 permarnent generation 과 method area 가 통합되어 mata space 로 변경됨

기종 permarnent 의 경우 jvm에서 메모리 일정부분을 강제로 할당하여 사용되지만 metaspace 의 경우 native memory 영역으로 os가 자동으로 크기를 조절해 주어 permarnent 보다 비교적 큰 메모리 영역을 사용할 수 있게 되었음

-------------------------------jdk 7

<---------- Java Heap -----------><--- Native Memory --->

+------+----+----+-----+-----------+--------+--------------+

| Eden | S0 | S1 | Old | Permanent | C Heap | Thread Stack |

+------+----+----+-----+-----------+--------+--------------+

<--------->

Permanent Heap

S0: Survivor 0

S1: Survivor 1

------------------------------------------------------jdk 8 이후

<----- Java Heap -----><------- Native Memory ---------->

+------+----+----+-----+-----------+--------+--------------+

| Eden | S0 | S1 | Old | Metaspace | C Heap | Thread Stack |

+------+----+----+-----+-----------+--------+--------------+

Stack area는 jvm에서 읽어들인 클래스와 상수 메서드 static 변수 등을 보관하는 area이다

native method stack 은 c,c++ 같이 java가 아닌 코드로 짜여진 파일들을 java에 맞게 변환하여 이식해주는 기능을 가지고 있다

pc resister 현재 실행중인 jvm 주소를 가지고 있는 resister

>> “openjdk 1.8” 이 설치된 centos 7 구동용 Dockerfile 및 docker image를 생성하세요

* openjdk 설치는 yum을 사용하고, 검색방법/설치방법을 익히세요. (yum cheat sheet)
* yum 사용하기 전 ( yum update && yum upgrade )를 먼저 실행합니다.
* “/javawork” 디렉토리를 workdir로 설정합니다.
* docker image 명은 “centos7\_openjdk8”로 지정합니다.

**Dockerfile**

FROM centos:7  
  
RUN //bin//bash   
  
RUN yum update -y && \  
yum upgrade -y && \  
yum install -y java-1.8.0-openjdk-devel.x86\_64   
 WORKDIR /javawork

docker build --tag centos\_7\_openjdk8:1 . // .의 위치에 존재하는 dockerfile 을 빌드하여 centos\_7\_openjdk8 이라는 tag를 가진 image를 생성함

>> 빌드된 이미지 (centos7\_openjdk8)로 컨테이너 구동 옵션 추가

* 컨테이너 workdir로 설정된 경로(/javawork)를 host pc 임의 경로와 연결하세요.
* docker run -it --name test -v /임의경로/javawork centos7\_openjdk8:1

>> “Hello World”를 출력하는 코드를 작성하고, compile 후 실행하세요.

Echo "public class HelloWorld {" > HelloWorld.java

Echo "public class HelloWorld {" >> HelloWorld.java

Echo " System.out.println(\"Hello, World\");" >> HelloWorld.java

Echo " }" >> HelloWorld.java

Echo "}" >> HelloWorld.java

javac HelloWorld.java

java HelloWorld

**class path 지정**

java -cp “.:bin” packageaname.classname // 현재 디렉토리와 bin 에서 class path를 지정하고 package 경로 를 검색하여 classname 인 class 를 실행함

ex) java -cp “.:bin” com.gachidata.demo.HelloWorld // 현재 디렉토리와 bin 에서 com/gachidata/demo/Hellowrld 라는 class 파일을 실행함

>> “public static void main(String[] args)” 의 각 키워드를 설명하세요.

Public // public 은 접근제한자로써 해당 클래스의 모든 접근을 허용함

static // 인스턴스를 생성할때 인스턴스들 같에 공통적인 값이 유지되어야 하는 경우 사용하는 메서드 static 은 임의의 메모리 주소를 할당받음

void // 반환되는 값이 없음

String[] args // String 배열의 args 라는 변수명을 가진 값을 넣을 수 있음

>> classpath 의 개념을 조사하고 설명하세요.

Classpath 란 자바 가상머신이 프로그램을 실행할때 class 파일을 찾는데 기준이 된는 경로를 의미함

classpath를 지정하지 않으면 jvm이 위치한 디렉토리에서만 클래스를 찾게됨