Avishek

9

* Garbage Collection (181 - 182)
* Internationlization (183 - 185)
* ENUM (186 - 188)
* Development (189 - 191)
* Assertions (192 - 195)
* JVM Architecture (196 - 203)

#181 finalization

11/May/23

* Finalization

→ Just before destroying an object GC calls finalize() method to perform clean-up activities.

→ Once finalize() method complete automatically GC destroyes the object.

→ finalize() present in Object class.

> protected void finalize() throws throwable

→ we can override finalize() in our class to define our own clean-up activities.

Case-I

```
class Test {
    p s v m (s [] a){
        Test s = new Test();
        String s = new S("durga");
        s = null;
        System.gc();
        Sopln("End of main");
    }
    p void finalize() {
        Sopln("finalize method called");
    }
}
```

s → durga

{main → GC

End | final
final | End

→ Just before destroying an object GC calls finalize() method on the object whic is eligible for GC. then the corresponding class finalize() method will be executed. for eg. if String obj. eligible for GC then string class finalize method will be executed byt not Test class finalize method.

Case - II

```
class Test {
    p s v m (S[] a){
        Test t = new Test();
        t.finalize(); t.finalize()
        t = null;
        System.gc();
        Sopln("End of main"); }
    p void finalize() {
        Sopln("Finalize method"); }
}
```

o/p:
Final
Final
End
Final

→ Based on our requirement we can call finalize() method explicitly then it will be executed just like a normal method call & object won't be destroyed.

Note:- If we are calling finalize() method then it is like normal method call. & obj. won't be destroyed but if GC calls finalize() method then object will be destroyed.

Note:- init(), service() & destroy() method are considered as lifecycle method of Servlet.

→ Just before destroying servlet object web container calls destroy method to perform clean-up activities ~~then~~. but based on our requr. we can call destroy() method from init() & service method then it will be executed like normal meth

#182 Finalization

11/May/23

Case-III

```
>Class FinalizeDemo {
    Static FinalizeDemo s;
    p s v m(S[] a)throws IE {
      FD f = new FD();
      Sopln(f.hashCode()); 100
      f = null;                          100
      System.gc();                       finalize
      Thread.sleep(5000);                100
      Sopln(s.hashCode()); 100           End
      s = null;
      System.gc();
      Thread.sleep(10000);
      Sopln("End of main"); }
    public void finalize() {
      Sopln("finalize method");
      s = this;                  }
}
```

→ Eventhough object eligible for GC multiple times but GC calls finalize() method only once.

Case-IV

```
Class Test {
    p s v m(S[] a){
      for (int i=0; i<10    ; i++){
        Test t = new Test();
        t.null);                  }
}
```

```
static int count = 0;
public void finalize(){
    Sopln("Finalize m" + ++count);
}
```

→ We can't expect exact bevibour of GC. it is varied from JVM to JVM hence for following questions we can't provide exact answers

1) When exactly JVM runs GC?
2) In which order GC identifies eligible objects.
3) In which order GC destroyes eligible objs.
4) Wether GC destroyes all eligible objs or nat.
5) What is algorithm followed by GC. etc.

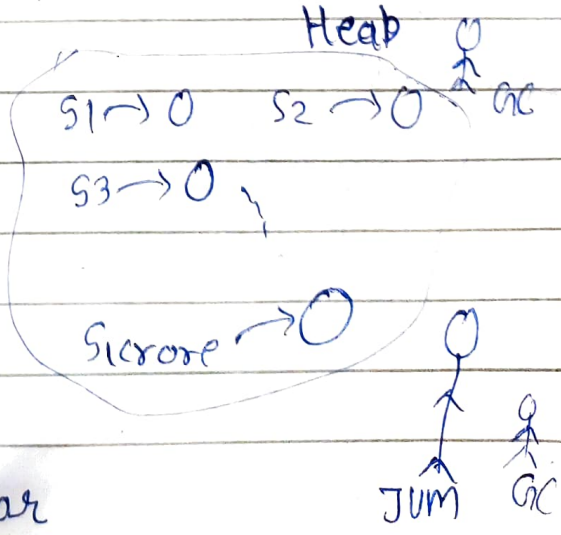
Note :-

→ Whenever programes runs with low memory then JVM runs GC but we can't expect exactly atwhat time.

→ Most of the GC follow standard algo. mark & sweep algo. It does nat every GC follow the same algo.

Case - V

```
Student s1 = new S();
Student s2 = new S();
s1 = s2;
Student s3 = new S();
```



RE: Out Of Memory Error

* Memory leaks :-

The objects which are not used in our program & which are not eligible for GC such type of useless objects are called Memory leaks.

→ In our program if memory leaks present then the program will be terminated by raising OutOfMemory Error.

→ hence if object no longer required it is highly recommended to make that object eligible for GC.

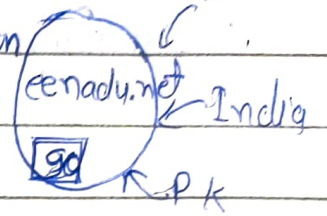
* various third party memory management tools to identify memory leaks.

HP OVO, HP J Meter, JProbe, Patral, IBM Tivoli

#183 Internationalization [I18N]

12/05/23

→ The process of designing web-application in such a way that which provide support for various country & varius languages & various currencies automically without performing any chang in application, is called Internationalization (I18N)



→ We can implement I18N by using these 3 classes
① Locale ② NumberFormat ③ DateFormat

#184 Locale class

12/May/23

→ A locale object represents a geo-graphic location (country), or language or both.
eg:- We can create a locale obj. to ~~represent~~ India.
→ Locale class present in java.util pkg.
→ It is a final class & it is direct class of object.
→ It implements serializable & clonable interfaces

* Constructors

① Locale l = new Locale(String language);
② Locale l = new Locale(str lang, str country);
eg locale l = new Locale("pa", "IN");
Punjabi ↳ India

→ Locale class already define some constants to represent some standard locales.
→ locale.US, locale.ITALY.

* methods

① Locale.getDefault();
② Locale.setDefault(Local l);
③ s getLanguage();
④ s getDisplayLanguage();
⑤ s getCountry()
⑥ s getDisplayCountry()
⑦ String[] getISOlanguages()
⑧ String[] getISOcountries()
⑨ local[] getAvailablelocales()

* Number Format

→ Various location follow various styles to represent a Java number eg.

double d = 123456.789;

IN : 1,23,456.789
US : 123,456.789
ITALY : 123.456,789

→ We can use numberformat class to format a Java no. A/c to a particular locale.
→ It is present in java.text pkg.
→ It is abstract class.
⇒ ✗ NumberFormat nf = new NumberFormat();
↳ bcz it is an abstract class.

* Getting NumberFormat Object for Default locale.

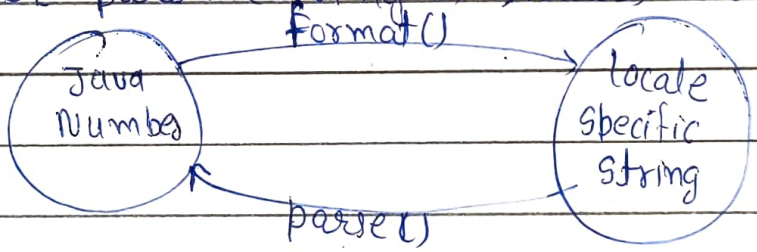
=> NF nf = NF.getInstance();
> NF nf = NF.getCurrencyInstance();
> NF nf = NF.getPercentInstance();
> NF nf = NF.getNumberInstance();

* Getting NF ~~for~~ object for specific locale.

NF nf = NF.getInstance(locale l);

→ In all above method just pass specific locale as argu
→ Once we got NF object we can call format() & parse() method on that object.

> p String format(long l);
p String format(double d);
> p Number parse(String s) throws ParseException.



Q WAP to display a java no. in itely specific form.

> ~~locale.ITALY~~ double d = 123456.789;
> NF nf = NF.getInstance(locale.ITALY);
> Str s = nf.format(d);
> Sop ln("Itely sp.":+s);

Q WAP to display a java no. in UK, US & India currency form.

#185 NumberFormat class

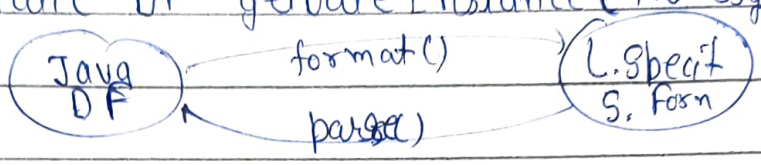
12|May|23

* Setting maximum & min. fraction & Integer digits.
→ NF cdp v setMaximumFractionDigits(int n)
> p v setMinimumFractionDigits(int n)
> p v setMaximumIntegerDigits(int n)
> p v setMinimumIntegerDigits(int n)

* DateFormat
⇒ Various location follows various styles to represent date. eg
> IN: DD-MM-YYYY US: MM-DD-YYYY
→ we can use DateFormat to format java date a/c to particular locale.
→ It is present in java.text package
→ It is an abstract class.
→ DateFormat df=new DF();
* Getting DF for Default locale
> DF.getInstance()
> DF.getDateInstance()
> DF.getDateInstance(int style)
DF.FULL → 0 → Wed ens 10th Sept 2023
DF.LONG → 1 → 10th Sept 2023
DF.MEDIUM → 2 → 10th Sep 2023
DF.SHORT → 3 → 10/09/23
→ The default style is medium.

* Getting DF for specific locale.

> p static DF getDateInstance(int style, Locale l)



> p String format(Date d);

> p Date parse(String s) throws ParseException.

Q: WAP to display current System date in all possible df of US form.

> Sopln("Full form" :"+ DF.getDateInstance(0, LocaleUS).format(new Date())); (0 → change & get)

Q: WAP to display current System date in UK, US, ITALY.

* Getting DF object to Display both Date & Time.

> ps DF getDateTimeInstance()

> p s DF getDateTimeInstance(int datestyle, int timestyle)

> p s DF getDateTimeInstance(int datestyle, int timestyle, Locale l) 0 to 3

#186 ENUM (Enumeration)

12/May/23

* ENUM

→ If we want to represent a group of named constants then we should go for ENUM.

> enum Month { JAN, FEB, MAR, ---, DEC; } → optional

→ The main objective of ENUM is to define our own data types. (Enumerated Data types)

→ It introduced in 1.5 v.

→ When compared with old languages enum Java enum is more powerful.

* Internal implemention of ENUM

> enum Beer { KF, RC; }

class Beer {
p. s final Beer KF = new B();
p. s final Beer RC = new B();
}

→ Every Enum is internally implemented by using class concept.

→ Every Enum is always ~~constant~~ public static final

→ Every Enum constant represent an object of type Enum.

> Enum Beer { KF, KO, RC, FO; }

class Test {
p s v m (S[] a) {
Beer b = Beer.RC;
Sop ln(b); o/p: RC
}
}

Note:- Inside Enum toString() internally implemented to return name of the constant.

→ We can declare enum either within the class or outside of class but not inside a method. If we try to declare inside method then we will get CE: enum Must not be local.

```
enum X { }
class Y { }
```

```
class X {
    enum X { }
}
```

```
class X {
    p v m1() {
        enum X { } }
}
```

CE:

→ If we declare enum outside of the class the applicable modifiers are Public, <default>, strictfp.

→ If we declare enum inside a class the applicable modifiers are public, <default>, strictfp, private, protected static;

* Enum vs Switch

→ Untill 1.4 v the allowed argument types for the Switch are b, s, i, c, & but 1.5 v onward corresponding wrapper classes & enum types are allowed.

from 1.7 v onwaard String type also allowed.

1.4 v	1.5 v	1.7v
byte	→ Byte	
Short	→ Short	String
Char	→ Character	
int	→ Integer	
	⊕	
	enum	

```
> enum Beer{
      KO, KO; RC, FO'; {
Class Test {
      p s v m(S[]a){
        Beer b = Beer.KF';
        Switch (b){
          case KF: Sop1n("gt is children brand");
                   break;
          Case RC: Sop1n("Nat kick");
                   breaka
        }}
}
```

→ Every case lebel shoud be valic enum constant otherwise we will get CE: Unqualifid enumerat.

```
Switch (b) {
    Case KF;
    Case KALYANI:
}
```

#187 enum Vs inheritance

12/May/23

* enum Vs inheritance

→ Every enum is directly child class of java.lang.Enum & hence our enum can't extends any other Enum (Bcz java won't provide support for multiple inheritance.

→ Every enum is final implicitly & hence for our enum we can't create child enum. hence we can't

→ Inheritence is not applicable for enum explicitly.

> enum X { }
> enum Y extends X { } ✗ CE:

> class X { }
> enum Y extends X { } ✗ CE:

enum X extends J.l.Enum { } ✗ CE:-

enum X { }
class Y extends X { } ✗ CE1: CE2:

> interface X { }
> enum Y implements X { }

→ Enum can implement any no. of interfaces.

* Java.lang.Enum

→ Every Enum is direct child class of java.lang.Enum

→ This class acts as base class for all javaa Enums.

→ It is an abstract class & direct child of object.

→ It implements Serializable & Comparable interfaces.

* Values()

> Enum Beer {
KF, KO, RC, FO; }

> class Test {
psvm (S[]a){
Beer[] b = Beer.values();
for (Beer b1 : b){
Sopln (b1); } // KF / KO / RC / FO
}
}

→ Every enum implicitly contain values() method to listout all values present inside enum.

Beer[] b = Beer.values();

Note: Values() method not present in java.lang.enum & object classes. enum keyword implicitly provides this method.

* ordinal()

→ Inside enum order of constant is important & we can represent this order by using ordinal value.

→ We can find ordinal value of enum constant by using ordinal() method.

> public final int ordinal();

→ Ordinal value is zero based.

KF --- 0
KO --- 1
RC --- 2
FO --- 3

* Speacility of java Enum.

→ In old languages enum we can take only constants. but in java enum in addition to constants we can take methods, constructors normal variables etc.

→ Java enum is more powerful than old lang. enum.

→ Even in java enum we can declare main method & we can run enum class directly from cmd.

```
> enum Fish {
      STAR, GUPPY, GOLD();
      p s v m (S[]a){
          Sop ln ("ENUM main");}
  }
```

Java Fish.java ↵
Java Fish ↵
o/p: ENUM main.

→ In addition to constants if we are taking any extra ~~methods~~ members like method then list of constant should be in the first line & should ends with ";"

```
> enum Fish {          mandatory
      STAR, GUPPY();
      public void m1(){}
  }
```

```
> enum Fish {
      STAR, GUPPY
      p v m1(){} }
  }
```

```
> enum Fish {
      p v m1(){}
      STAR, GUPPY;
  }
```

```
> enum Fish {
      pa v m1(){}
  }
```

```
> enum Fish
      ;
      p v m1(){}
  }
```

#188 enum Vs Constructor

12|May|23

→ Inside enum if we are taking any extra member like method compulsary the first line should contains list of constants at least ';'.

→ An empty enum is valid java syntax.

> enum Fish { } ✓

* enum Vs Constructor

```
> enum Beer {
static KF, KO, RC, FO;
    Beer() {
        Sopln("Constructor"); }
}
> class Test {
    p s v m(S[] a){
        Beer b = Beer.RC;
        Sopln("Hello"); }
}
```

Java Test.java

Beer.class Test.class

Java Test ↵

Hello

Java Test ↵

op: Const. Const. Const. Const. Hello

→ An enum can contain constructor.

→ Enum constructor will be executed seperately for every enum constant at the time of enum class loading automatically

> Beer b = new Beer(); CE: can't be instantinted.

→ we can't create enum object directly & hence we can't envoke enum constructor directly

```
enum Beer {
    KF(70), KO(80), RC(90), FO;
    int price;
    Beer(int price){
        this.price = price; }
    Beer() {
        this.price = 65; }
    public int getPrice() {
        return price; }
}
class Test {
    p s v m(S[]a){
        Beer[] b = Beer.value();
        for (Beer b1:b) {
            Sopln(b1 + --- + b1.getPrice()); }
    }
}
```

note :-

KF ⇒ p static final Beer KF = new Beer();
KF(70) ⇒ p static final Beer KF = new Beer(70);

→ Inside enum we can declare ~~enum~~ methods but should be concrete method only & we can't declare abstract methods.

C-I

→ Beer.KF.equals(Beer.RC)

→ Beer.KF.hashCode() > Beer.RC.hashCode()

X> Beer.KF < Beer.RC

→ Beer.KF.ordinal() < Beer.RC.ordinal()

→ Every enum constant represents object of type enum hence whatever methods we can apply on normal java objects, can be applicable on enum constant also.

C-II

→ If we want to use any class or interface directly from outside package then the required import is normal import.

→ If we want to ~~use~~ access static members without class name then the required import is static import.

```
import static java.lang.Math.sqrt;
import java.util.AL;
class Test{
    p s v m(S[] a){
        AL l = new AL();
        Sopln(sqrt(4)); }
}
```

```
package pack1;
public enum Fish {
    STAR, GUPPY; }
```

```
import pack1.*;
import pack1.Fish;
package pack2;
public class Test1{
    p s v m(S[] a){
        Fish f = Fish.GUPPY;
        Sopln(f); } }
```

```
import static pack1.Fish.*;
import static pack1.Fish.STAR;
```

```
> package pack3;
> public class Test2 {
      p s v m(S[]a){
        Sopln(STAR); }
  }
```

```
> package pack4;                    → import pack1.Fish;
> p class Test3 {                   → import static pack1.Fish.Guppy;
      p s v m(S[]a){
        Fish f = Fish.STAR;
        Sopln(GUPPY;
  }
```

Case-III

```
> enum Color
      BLUE, RED, GREEN;
      p v info() {
        Sopln("Universal col"); }
  }
> class Test {
      p s v m(S[] args) {
        Color[] c = Color.values();
        for (Color c1 : c) {
            c1.info(); }
  }
}
```

o/p:
UC
UC
UC

```
> enum Color {
      BLUE, RED
            { p v info() {
                  Soplm ("Dangerous color"); }
            }, GREEN;
      p v info () {
            Soplm ("Universal color"); }
}
> class Test {
      p s v m (S[] a) {
        Color[] c = Color.values();
        for(Color C1 : c) {
            C1.info();      }
      }
}
```

O/p.

```
U. C
D C
U C
```

* enum Vs Enum Vs Enumeration

enum	Enum	Enumeration
↳ keyword	↳ class	↳ interface {iterator type.
→ used to define a grp. of named constants.	→ Every enum in java is direct child of Enum class.	→ We use it to get object one by one from the collection.

#189. Development

12/May/23

* Javac

javac [options] Test.java ↵

Javac [options] A.java B.java C.java ↵

Javac [options] *.java

↳ -version, -d, -source, -verbose, -cp, ---

→ We can use javac command to compile single or group of java source files.

* java

java [options] Test A B C ↵
cmd-arg.

↳ -version, -D, -cp/-classpath, -ea, ----

→ We can use java command to run a single class file.

Note:- We can compile any no. of source file at a time but we can run only one .class file at a time.

* classpath :

→ classpath describes the location where required .class file are available.

→ Java compiler & JVM will use class path to locate required .class file.

```
> class Test{
      p s v m(S[] a){
        Sop1n("ClassPath Demo");}
  }
```

c:\durga_classes > Javac Test.java ↵

c:\durga_classes > Javac Test ↵

o/p: classPath Demo.

C:\> Java Test ↵;

RE: NoClassDefFoundError : Test

→ By default JVM will always searches in current working directory for required .class file.

→ If we set class path explicetly then JVM will search in our specified class path location & JVM won't search in cmd.

① Environment var classpath → Permanent

② set classpath = C:\durga_classes → Temprory

③ Java -cp C:\durga_classes Test ↵ → Temp.

→ Setting pat at cmd leved is recommended bcz dependent classes are varied from cmd to cmd.

> C:\> Java -cp C:\durga_classes Test ↵

o/p: cp demo.

> D:\> Java -cp C:\durg_classes Test ↵

o/p: cp demo.

→ Once we set the class path we can run our program from any location.

→ Once we set class path JVM won't search in current working directory & it will always search in the specified class path location only.

C:\durga_classes > Java -cp .; F: Test

o/p. cp demo.

C: → pack1 → pack2 → kareena.class

```
package pack1.pack2;
p c kareena {
    p v m1() {
        Sopln("Hello S. Hello!");
    }
}
```

D: → pack3 → pack4 → saif.class

```
package pack3.pack4;
import pack1.pack2.kareena;
p c saif {
    p v m2() {
    kareena k = new k();
    k.m1();
    Sopln("Not possible"); } }
```

E:

```
import pack3.pack4.saif;
p c Durga {
    p s v m(S[]a) {
     saif s = new S();
     s.m2();
     Sopln("Hello kareena"); }
}
```

C:\> javac -d . kareena.java ↵

C:\> javac -d . saif.javav ↵ CE:

C:\> javac -d . -cp C:\ saif.java ↵

F:\> javac Durga.java ↵ CE:

F:\> javac -cp D:\ Durga.java ✓

→ Compiler will only check for one level of depend.

E:\> java Durga ↵ RE: NCDFE:

E:\> java -cp .; D:\ Durga ↵ RE: NCDFE

→ JVM will check for all level of dependency.

E:\> java -cp .; D:\;C:\ Durga ↵ ✓

F:\> java -cp .; D:\;C:\ Durga ↵ ✓

#190 Jar Files

13|May|23

→ If any location created bcz of package statem that location should be resolved by using import statement. & base package location we have to update in class path.

→ Compiler will check only one level dependency whereas JVM will check all levels of dependency

→ In class path the order of location is importan & JVM will always consider from left to ~~wr~~right untill required match available.

* Jar Files

→ All third party software plugins are by defaul available in the jar file only.

→ All .class file we have to group in a zip file & we have to make that zip ~~fi~~ available ~~to~~ in the class. this zip file jar.

→ To develope a servlet all required dependent classes are available in servlet-api.jar. we have to place this jar file ain class ~~p~~path ~~ite~~ to compile a servlet program.

→ To run a JDBC program all dependent classes are available in ojdbc14.jar.

→ To use Log4J in our application dependent class are available Log4J.jar.

* Various command

① To create a jar file (zip file)

> jar -cvf durgacalc.jar Test.class

> jar -cvf durgacalc.jar A.class B.class C.class

> jar -cvf durgacalc.jar *.class

> jar -cvf durgacalc.jar *.*

② To extract a jar file (Unzip file)

> jar -xvf durgacalc.jar

③ To display table of contents:

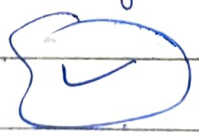
> jar -tvf durgacalc.jar

⇒ Service provider Role

```
> p class DurgaColorfulCalc {
      public static v add(int x, int y){
          Sop(x*y);
      }
      p s v multiply(int x, int y){
          Sop(2*x*y);
      }
  }
```

> Javac DurgaColorfulCalc.java ↵

> jar -cuf durgacalc.jar DurgaColorfulCalc.class



durgacalc.jar

⇒ Client's Role:-

> He downloaded jar file & he placed in D: of client machine.

```
class calc {
    p s v m (S[] a){
        DurgaColorfulCalc.add(10,20);
                          .multiply(10,20); }
}
```

C:\durga_classes> javac calc.java ✗ → CE:

C:\durga-classes> javac -cp D: calc.java ✗ → CE:

C:\durga-classes> javac -cp D:\durgacalc.jar calc.java

C:\durga-classes> java calc ↵ RE: NCDFE: DurgaColorful

C:\durga-classes> java -cp D: calc ↵ RE: NCDFE: Calc.

C:\durga-classes> java -cp .;D: calc ↵ RE:

C:\durga_classes> java -cp .;D:\durgacalc.jar calc ↵

o/p :- 200, 400

Note: To place .class file in class path just location is enough but to make jar file availa-ble in class path location is not enough compulsary we have to include name of the jar file also.

* Shortcut way to place jar file in classpath.

→ If we ~~jar~~ place jar fille in → JDK ↦ JRE ↦ lib ↦ ext ↦ *.jar

below location then all classes & interface present in jar file by default available to java compiler & JVM.

* Properties

```
> class Test {
      p s v m(S[]a){
          Properties p = System.getProperties();
          p.list(System.out); } }
```

→ For every system some persistant information will be maintained in the form of system properties these include name of OS, java version, JVM vendor, user country etc.

(space not allowed) → Pname → Pvalue

```
> java -Ddurga = ocjp Test
```

To set system property

→ we can set system property explicitly from cmd prompt by using '-D' option.

→ The main adv. of setting system property is, we can customize bevibour of java program.

```
> class Test{
      p s v m(S[]a){
          String course = System.getProperty("course");
          if (course.equals("scjp")) {
              Sopln("scjp");
          }else {
              Sopln("other"); }
      }
}
```

```
> java -Dcourse = scjp Test  o/p: scjp
> java -Dcourse = scwcd Test  o/p: other.
```

#191 Jar Vs War Vs Ear || JDK VS JRE VS JVM

13/May/23

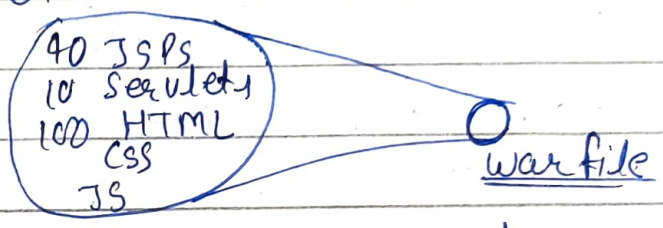
* Jar Vs War Vs Var

① Jar (Java Archieve)

→ It contains a group of .class files.

② War (Web Archieve)

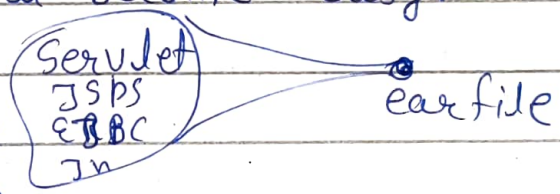
→ A war file represents one web application. which contain servlet, JSPs, HTML pages, CSS, JS etc.



→ The main adv. of maintaining web application as a war file is project deployment, project delevery & project transportation will become easy.

③ ear (enterprise Archieve)

→ An ear file represents one enterprise application which contains servlets, JSPs, EJBS, JMS components etc.



Note :- In general ear files represents a group of war & jar files.

* Webapplication Vs Enterprise Application

→ A webapplication can be developed by only web related technologies like servlets, JSPs, HTML, CSS, JS, XML etc.. eg :- online lib. management System.

→ An Enterprise application can be developed by any technology from Java EE like servlets, JSPs, EJBs, JMS component etc. eg :- Banking Project.

Note :- JEE compatible is enterprise application.

* Web Server Vs Application Server

→ Web server provides environment to run web -application.

→ Web server provides supports for web related technologies like servelets, JSPs, HTML, CSS etc.
eg :- Tomcat.

→ Application server provide environment to run enterprise applications.

→ Application server can provide support for any technology from JEE. like servelets, JSPs, EJB, JMS comp. etc. eg - Weblogic,

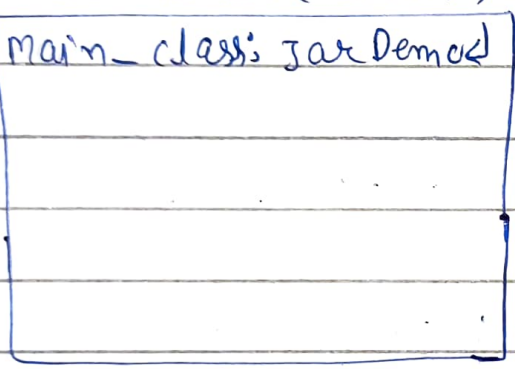
Note :- Every App. server contains inbuilt web -server to provide support for web related technologies.

→ J2EE compatible server is application server.

* How to create executable Jar File.

```
> import java.awt.*;
> import java.awt.event.*;
> public class JarDemo {
      p s v m (S[] args) {
          Frame f = new Frame();
          f.addWindowListner(new WindowAdapter()
          { public void windowClosing(WindowEvent e)
              { for int i=1; i<=10; i++){
                  Sop("I am closing window!"+i);
```

```
                System.exit(0); }
});
f.add(new Label("I can create Exec. Jar File!!!"));
f.setSize(500,500);
f.setVisible(true);
}}
```



JavaC JarDemo.Java

JarDemo.class JarDemo$1.class

manifest.mf

```
> jar -cvfm demo8.jar JarDemo.class JarDemo$1.class
                manifest.mf
> java -jar demo8.jar
```

* How many ways to run a java program?

① java to run .class file.

```
java JarDemo
```

② java to run .jar file

```
java -jar demo8.jar
```

③ By double clicking a jar File:

④ By double clicking a batch file:

→ A batch file contains a group of commands whenever we are double clicking a batch file then all cmds will be executed one by one in the sequence.

* Diff. b/w path & classpath.

> classpath : It describe location where required .class file is available.

→ Java compiler & JVM will use classpath to locate required class files. If we are not setting class path then our program may not compile or may not Run.

> Path : It describe the location where required binary executable are available.

→ If we are not setting path then javac & java command won't work.

> set path = C:\Program Files\Java\jdk1.8.0_111\bin

* Diff b/w JDK vs JRE vs JVM

> JDK (Java Development Kit)

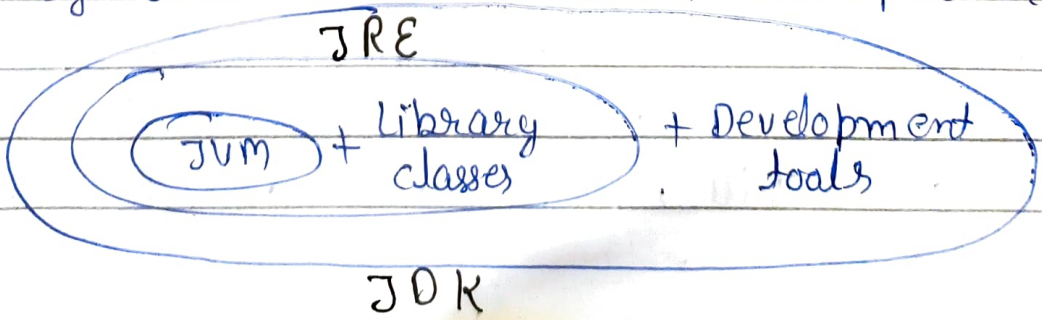
→ JDK provides environment to develope & Run java applications.

> JRE (Java Runtime Environment)

→ JRE provides environment to Run java appl"

> JVM (Java Virtual Machine)

→ JVM is responsible to run java program line by line. hence it is an interpreter.



> JDK = JRE + Development tools
> JRE = JVM + Library classes
Note:- On developer machine we have to install JDK whereas client machine we have to install JRE.

* Diff. b/w Java Vs Javaw Vs Javaws

① Java.

```
> class Test {
      p s v m (S[]a) {
       Sopln ("console output");
        }                      }
}
```

→ We can use java command to run a java class file where sops will be executed & corresponding output will be displayed to the console.

② Javaw (Java without console output)

→ We can use javaw cmd to run a java class file where sops will be executed but corresponding output will not be displayed to console.

→ In general we use javaw cmd to run GUI based applications

③ Javaws (Java web Start utility)

→ We can use javaws to download a java appl. from the web & to start it's execution.

→ We can use javaws cmd as follows.

→ [Javaws Jnlp-url] ↵

→ It downloads the application from the specified URL & start's execution.

→ The main adv. in this approach is every end user will get updated version. & enhancement will become easy bcz of centeralized control.

#192 Assertions (1.4V)

14/May/23

** ① Introduction
② assert as a keyword & identifier
③ Types of assert statement
④ Various possible Runtime flags
⑤ Appropriate & Inappropriate use of assertions
⑥ AssertionError.

* Introduction

→ Very common we have debugging is usage of Sop statements but problem with Sops is after fixing the bug compulsary we have to delete Sop statements otherwise these Sops will be executed at runtime for every request, which creates performance problem & disturbes server logging.

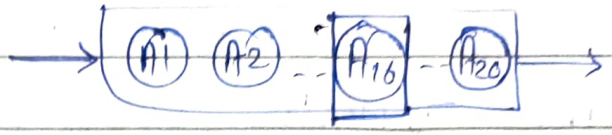
→ To overcome this problem sun people introduced assertions concept in 1.4 v.

→ The main adv. of assertions when compared with Sops is after fixing the bug we are not require to remove assert statement bcz they won't be executed by default at run time.

→ based on our requirement we can enable & disable assertion & by default assertions are disabled.

→ The main objective of assertion is debugging.

① Development (Dev)
② Test
③ production



→ Usually we can perform debugging in development & Test environment but not in production envir. hence assertion concept applicable for dev & Test environment but not for production.

* assert →1.4v as keyword & identifier :

→ assert keyword introduced in 1.4 v.

→ 1.4 v onwards we can't use assert as identifier otherwise we will ge CE:

```
class Test{
    p s v m(S[] a){
        int assert=10;
        Sop ln(assert);}
}
```

① Javac Test.java ✗
CE:

② Javac -source 1.3 Test.Java
compile fine but with warning

③ Java Test ✓ (10)

Note:→ If we are using assert as identifier & if we are trying to compile A/c to old version (1.3 or lower) the code compiles fine but with warning.

→ We can compile a java program A/c to particular version by using -source option.

#193 Simple assert || augmented assert

14/May/23

* Tybes of assert statement
① Simple version ② Augmented version.

* Simple version

> assert(b);

→ b should be boolean type.
→ If b is true then our assumption satisfy & hence rest of the program will be executed normaly
→ If b is false then our assumptions fails i.e somewhere something goes wrong & the prog. will be terminated abnormally by raise Assertion Error.

```
> class Test {
      p s v m (S[] a){
          int x = 10;
          ; ; ; ; ; ; ; ;
          ; ; ; ; ; ; ; ; ---
          assert(x ==10);
          ; ; ; ; ; ; ;
          sopln(x);       }
  }
```

Javac Test.java ✓
Java Test ↵
o/p 10

Java -ea Test ↵
RE: Assertion Error

Note:- By default assert statement won't be executed bcz assertions are disabled by default but we can enable assertions by using -ea option.

* Augmented version
→ we can augment some description with assertion error by using augmented version.

> assert (b) : e;
b → bool
e → Can be any type.

Conclusion :-

C-1

```
class Test {
    p s v m(S[]a){
        int x = 10;
        assert (x == 10) : ++x;
        Sop(x);
    }
}
```

Java -ea Test
o/p [10]

RE: AE : 11

> assert (b) : e;
↳ e will be executed iff b is false.

C-2

```
> class Test {
    p s v m(S[]a) {
        int x = 10;
        assert(x>10) : m1();
        Sop(x); }
    p s int m1() {
        return 777; }
}
```

RE: AE : 777

> assert (b) : e; → we can take method call but void return method call is not allowed.

→ Among two version, it is recommend to use augmented version.

#194 Various run time flags

14/May/23

* Various possible run time flags

① -ea | -enableassertion
→ To enable assertions in every non-System class. (our own classes)

② -da | -disable assertions
→ To disable assertions in every non-System class.

③ -esa | -enableSystemassertions
→ To enable assertions in every system class (pre-d

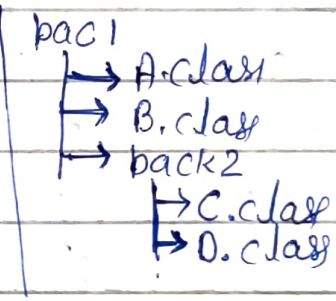
④ -dsa |
→ To disable assertions in every System class.

→ We can use above flags simultniously.

* Case Study :-

① To enable assertions only in B
java -ea: pack1.B

② To enable assertions in bot B & D
java -ea: pack1.B -ea: pack1.pack2.D



③ To enable assertions in every class of pack1
java -ea:pack1...

④ To enable assertions in every class of pack1 except B class
java -ea: pack1... -da: pack1.B

⑤ To enable assertions in every class of pack1 except pack2 class a
java -ea: pack1... -da: pack1.pack2.A

Note:- We can enable or disable assertions either class wise (or) package wise also.

* Appropriate & Inappropriate use of assertions

> p v withdraw(double amount){
    if (amount < 100)
        throw new IllegalReq.Exc();
    else
        process request
}

> public v withdraw(double amount){
    assert(amount >=100);
    process Request
}
⇒ Inappropriate way

→ It is always inappropriate to mix programming language with assert statement bcz there is no gurantee for the execution of assert statement always at Run time.

> switch (x) Should be a valid month no.
    Case1: Sopln("JAN"); break;
    !
    Case12: Sopln("Dec"); break;
    default: assert(false);
    }

→ while performing debugging in our program if there is any place where the control is not allow to reach that is the best place to use assertions.

private local outside.

> pub. v m1(int x){
  assert (x>10);

→ It is always inappropriate for validating public method arguments by using assertions bcz outside person doesn't aware that wether assertions are enabled or disabled in our system.

→ It is always appropriate for validating private method arg. by using assertions. bcz local person aware about it.

> p s v m(S[]a)
  Xassert(args[0].equals("durga")); }

→ It is always inappropriate for validating cmd line args. by using assertions bcz these are arg. to main method & it public.

#195 proper & improper use of assert

14/May/23

```
> class One {
      p s v m(S[]a){
        int assert=10;
        Sop(assert);}
  }
```

① javac -source 1.3 one.java
✗ ② javac -source 1.4 one.java
✗ ③ javac -source 1.3 Two.java
④ javac -source 1.4 Two.java

```
> class Test {
      p s v m(S[]a){
        int x=10;
        assert(x>10);}
  }
```

If ass. are not enabled?
NO o/p
If ass. are enabled:
RE: AE: true

```
> class Test {
      p s v m(S[]a){
        bool assertOn=false;
        assert(assertOn); assertOn=true;
        if(assertOn){
          Sop("assertOn");}
      }
  }
```

* Assertion Error

→ It is child class of error. It is unchecked.
→ If assert statement fails then we get AE.

→ In the case of web applications if we run java prog. in debug mode automatically assert stat. will be executed.

#196 JVM Architecture

14/May/23

① Virtual Machine
② Types of VM
   1. Hard ware Based VM
   2. Application Based VM
③ Basic Architecture of JVM
④ Class Loader SubSystem
   1. Loading
   2. Linking
   3. Initialization
⑤ Types of Class Loaders
   1. Bootstrap CL
   2. Extension CL
   3. Application CL
⑥ How class loader works
⑦ What is the need of customized CL
⑧ Pseudo code for customized CL.
⑨ Various Memory Areas of JVM
   1. Method Area
   2. Heap Area
   3. Stack Area
   4. PC Registers
   5. Native Method Stacks
⑩ Program to display heap memory statistics
⑪ How to set Max. & Min heap size?
⑫ Execution Engine
   1. Interpreter 2. JIT Compiler

(13) Java Native Interface (JNI)
(14) Complete Architecture Diagram of JVM
(15) Class File structure

* Virtual Machine

→ It is a software stimulation of machine which can perform operations like physical machine.

⇒ Types of VM

① Hardware based VM (or) System based | ② Application based (or) process based

* Hardware (or) System based VM

→ It provides several logical systems on same computer with strong isolation from each other.
→ One one physical machine we are defining 6 logical machines.
→ The main adv. of hardware based VM is hardware resources sharing & improves of utilization of hardware resources. eg :- KVM, VMware, Xen etc.

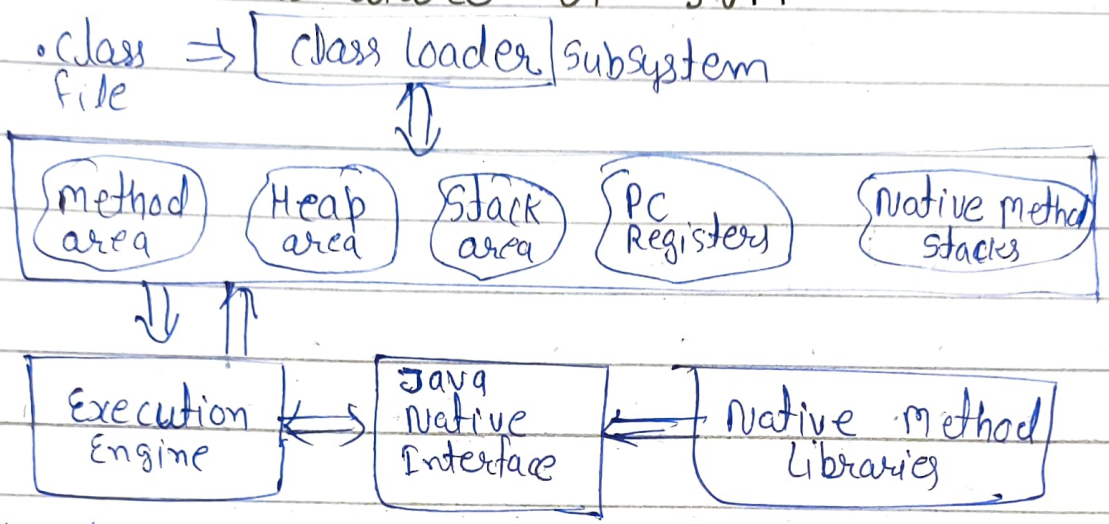
* Application (or) Process based VM.

→ These VM acts as runtime engines to run a particular programming language applications.
eg:- JVM, PVM, CLR etc.

* JVM - JRE → JDK

→ JVM is part of JRE
→ It is responsible to load & Run Java class files.

* Basic Architecture of JVM



① Loading ② Linking ③ Initialization

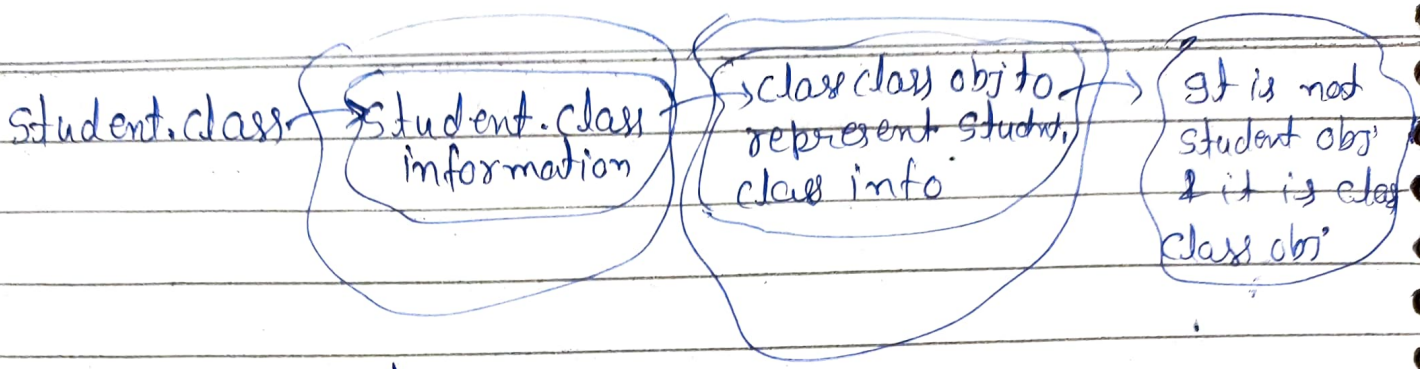
① Loading

→ Loading means reading class files & store corresponding binary data in method area.

→ For each class file JVM will store corresponding information in the method area.

(i) Fully qualified name of class.
(ii) Fully qualified name of immediate parent class.
(iii) Methods information
(iv) Variable info.
(v) Constructors info
(vi) Modifiers info.
(vii) Constant Pool infor ---- etc.

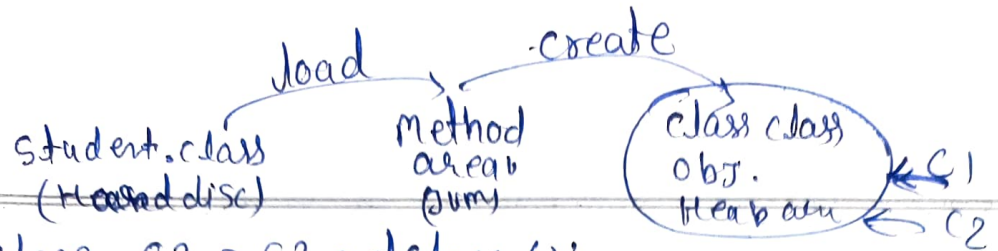
→ After loading .class file immediately JVM creates an object for that loaded class on the heap memory of type java.lang.class.



Hard-Disk Method Area(JVM) Heap Area (JVM)

→ class class obj. can be used by programmer to get class level info. like methods, var., const. info, etc.

```
> class Student {
      p str getName() {
        return null; }
      p int getRollno() {
        return 10; }
}
> class Test {
      v s v m(S[] a) {
        Class c = Class.forName("Student");
        Method [] m = c.getDeclaredMethods();
        for (Method m1 : m) {
            Sopln (m1.getName()); }
      }
}
> class Test {
      p s v m (S[]a) {
        Student S1 = new S();
        class C1 = S1.getClass();
        Student S2 = new S();
```




```
Class C2 = S2.getClass();
Sopln(C1.hashcode());
Sopln(C2.hashcode());
Sopln(C1==C2); true }
}
```

Note :- For every loaded type only one class will be created even though we are using class multi-ple times in our program.

197 Linking 14/May/23

② Linking

→ It consist of 3 activities.

(i) Verify (ii) Prepare (iii) Resolve.

(i) Verify ⓪ Verification

→ It is the process of insuring that Binary representation of a class is structraly correct or not i.e. JVM will check wether .class file is generated by valid compiler or not.

→ Internally byte code verifier is responsible for this activity.

→ Bytecode verifier is the part of class loader sub-system.

→ If verification fails then we will get RE : java.lang.VerifyError.

(ii) Preparation :~

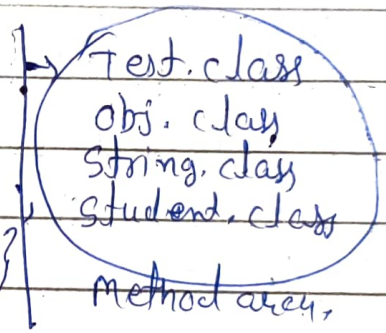
→ In this phase JVM will allocate memory for class level static var. & assign default values.

Note :- In initilization phase original value will be assigned to the static variables.

(iii) Resolution :~

→ It is the process of replacing symbolic names in our program with original memory references from method area.

```
> class Test {
      p s v m(S[] a){
          String s = new S("durga");
          String s1 = new S();
      }
}
```



→ For the above class class loader loads T, O, S, S. class.

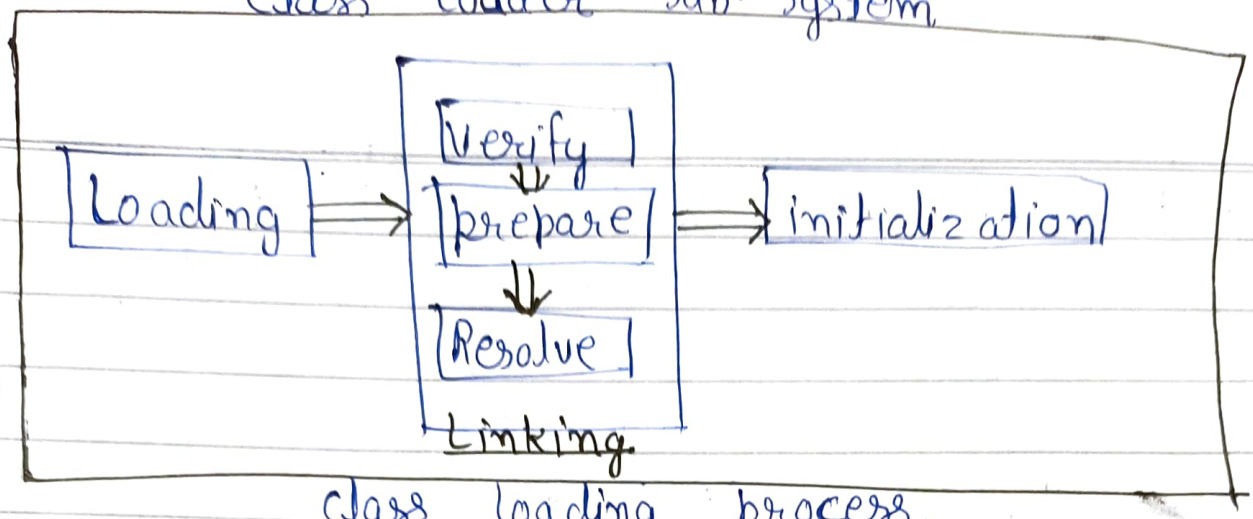
→ The names of these classes are stored in const. pool of test class.

→ In resolution phase these names are replaced with original memory level references from method area.

③ Initialization

→ In this all static variables are assigned with original values & static blocks will be executed from P to C & from Top to bottom.

Class Loader Sub-System



Class loading process.

Note:- While loading, linking & initialization if any error occurs then we will ge Runtime Error: Java.lang.LinkageError.

* Types of class loaders

→ It contains 3 types of CLs

1) Bootstrap & Primordial CL
2) Extension CL
3) Application & System CL.

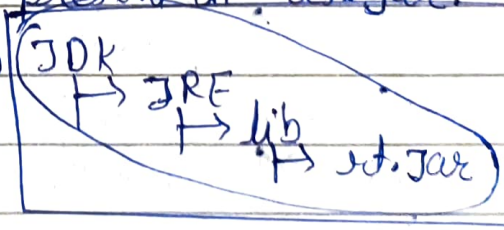
1) Bootstrap class Loader

→ Bootstrap cl is responsible to load core java API classes i.e the classes present in rt.jar.

→ This location is called Bootstrap class path.

→ Bootstrap CL is by default availabe with every JVM.

→ It is implemented in native languages like c/c++.



JDK
↳ JRE
↳ lib
↳ ext
↳ *.jar

2> Extension Class Loader

→ Extension CL is the child class of Bootstrap CL.

→ It is responsible to load classes from extension class path. (JDK/JRE/lib/ext)

→ It is implemented in java & the corresponding .class file is [sun.misc.Launcher$ExtClassLoader.class]

3> Application or System Class Loader

→ It is child class of Extension CL.

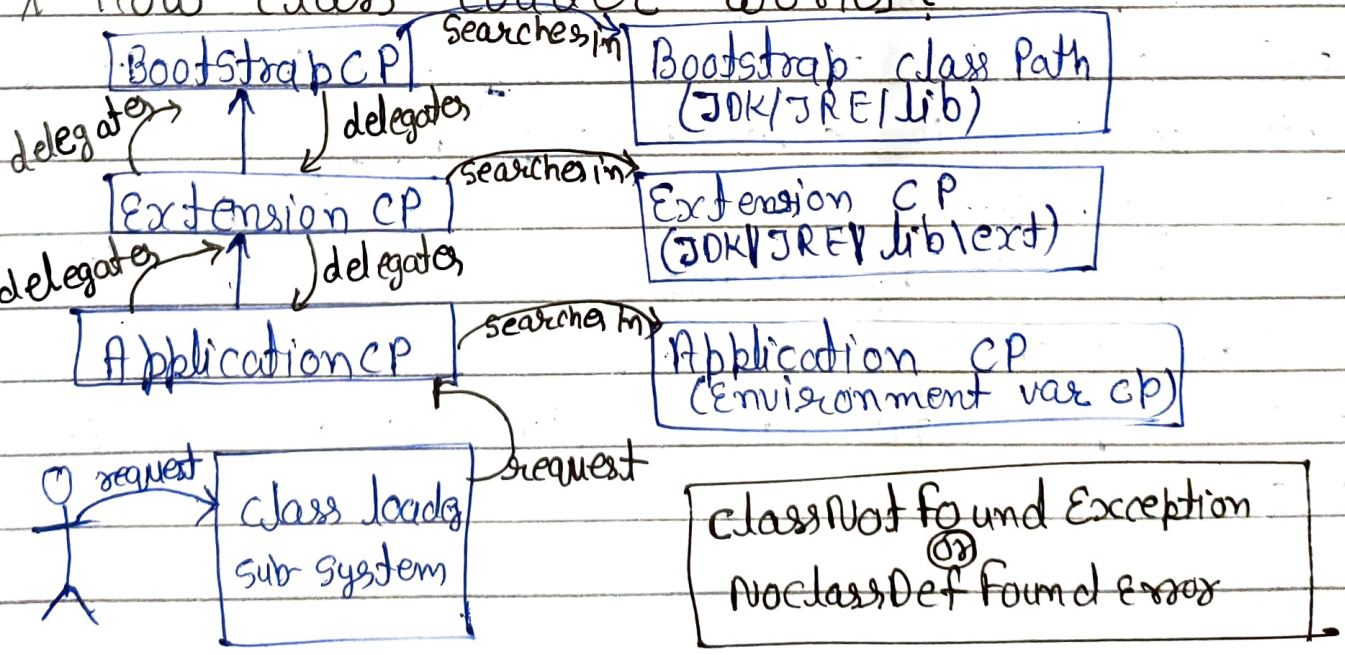
→ It is responsible to load classes from application class Path.

→ It internally uses enivorment variable class path

→ It is implemented in java & the corresponding .class file is [sun.misc.Launcher$AppClassLoader.class]

Bootstrap CL
↑
Extension CL
↑
Application CL

* How class loader works?



#198 How Class Loader Works?

14|May|23

* Class loader follows delegation hierarchi principle
→ Whenever JVM come across a particular class. First it will check wether the corresponding .class file is already loaded or not.
→ If it is already loaded in method area then JVM will consider that loaded class.
→ If it is not loaded then JVM will request class loader sub-system to load the particular class.
→ Then class loader sub-system handover request to Application CL.
→ App CL delegates request to Extension CL which inturn delegates request to Bootstrap CL.
→ Then Bootstrap CL will search in Bootstrap class path if it is available then corresponding class path will be loaded by Bootstrap CL.
→ If it is not available then Bootstrap CL delegates the request to Extension CL.
→ Extension CL will search in Ext. CP if it is available then it will be loaded other wise Ext. CL delegates request to App. CL.
→ App. CL will search in App. CP if it is available then it will be loaded otherwise we will get RE: NoClassDefFoundError (or) ClassNotFoundException

```
> class Test {
    p s v m(S[] a){
      So pln(String.class.getClassloader());
      Sopln(Test.class.getClassloader());
      Sopln(Customer.class.getClassloader()); }
}
```

o/p null

Sun.misc.Launcher$AppClassloader@1912a56

Sun.misc.Launcher$ExtClassloader@1072b90

Note :- Bootstrap CL is not java object hence we got null in first case.

→ Class L sub-sys will give the hiest priority for Bootstrap CP & then Extension CP followed by App. CP.

* Need of Customized class loader.

→ Default CL will load .class file only once even though we are using multiple time that class in our program.

→ After loading .class if it is modified outside then default .class file don't load updated version of .class file (bcz .class file already available in method area.

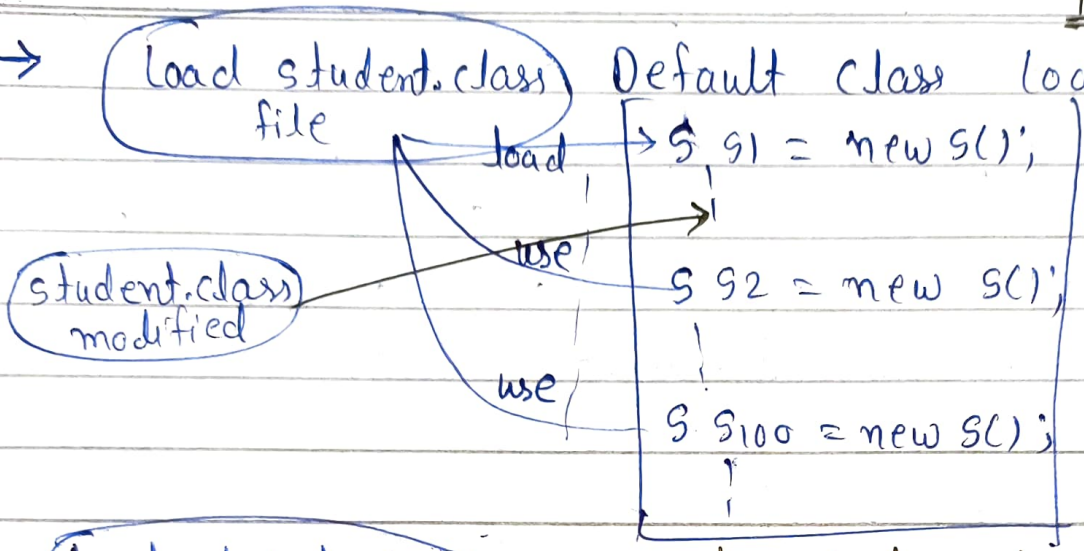
→ We can resolve this problem by defining our own class loader.

→ The main adv. of customize CL is we can control class loading mechanism based on our requirement.

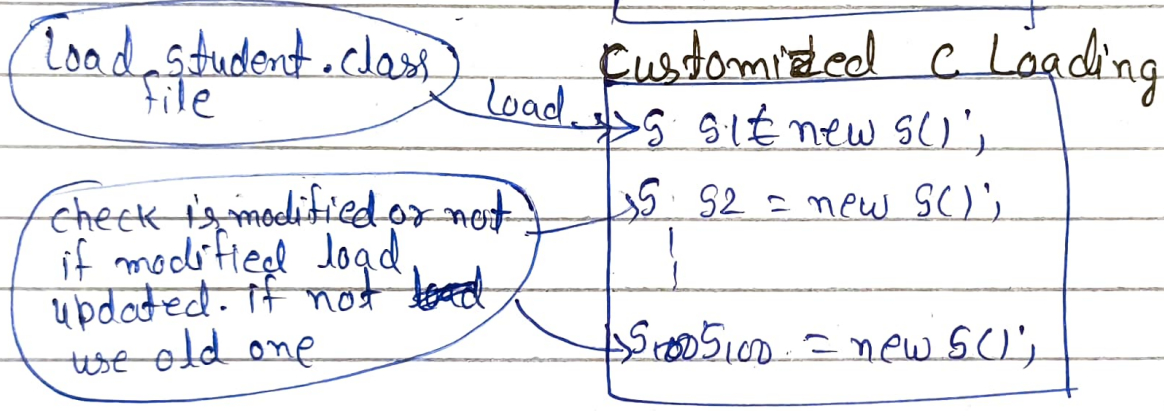
#199 Need of Customized classLoader

14|May|23

→ Default class loading



Customized C Loading



* How to define customized C L.

```
> class CustClassloader extends classloader {
      public class loadClass(Str name) throws classNotFound {
          check for updates & load updated .class file.
          & returns corresponding class      }
  }
> class client {
      p s v m (S[] a){
          Dog d1 = new Dog ();
          CustClassloader cl = new CCL();
          cl.loadclass ("Dog");
          cl.loadclass ("Dog");
      }
}
```

→ We can define our own customize class loader by extending java.lang.classloader.class.
→ While developing web @ Application servers usuly we can go for customized class loader.

Q: What is use of classloader class?

Ans:- We can use java.lang.CL class to define our own class loaders. Every class loader in java should be child class of java.lang.CL class either directly or indirectly hence this class acts as base class for all customize class loader.

Module - 2

✗ Various memory area present inside JVM.

→ Whenever JVM loads & runs java program it needs memory to store several things like bytecode, var, obj. etc.
→ Total JVM memory orginazed into 5 categories.

1) Method area
2) Heap are
3) Stack Area
4) PC Registers
5) Native method stacks

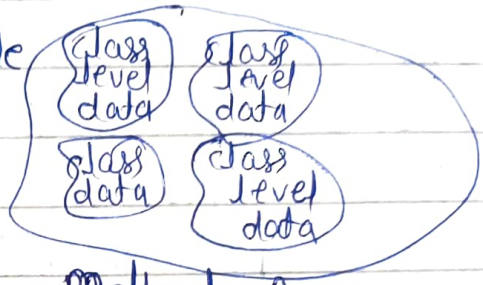
1) Method Area

→ For every JVM one method Area will be available
→ Method area will be created at the time of JVM startup.
→ Inside it class level binary data including static variables will be stored.

#200 Heap Area ||

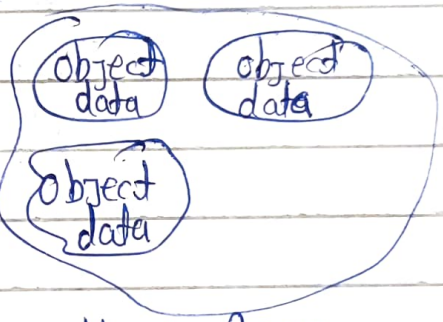
14/May/23

→ Constant pools of a class will be stored in it.
→ It can be accessed by multiple thread simultaneously. So, it is not thread safe.



2) Heap Area

→ for every JVM one Heap area is available.
→ Heap Area will be created at the time of JVM startup.
→ Objects & corresponding instance variables will be stored in heap area. Heap Area.
→ Every array in java is object only ~~or~~ hence arrays also will be stored in the heap area.
→ It can be accessed by multiple threads & hence the data stored in heap memory is not thread safe.
→ It need not be continious.



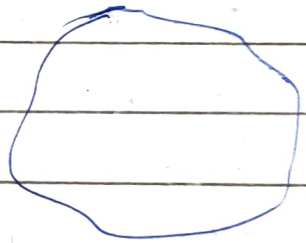
~~3) Stack Area.~~

Q: Program to display heap memory statistics?

```
Runtime r = Runtime.getRuntime();
① r.maxMemory()
② r.initialMemory()
③ r.freeMemory()
④ Consumed memory = initial - freeMemory
```

→ A java application can communicate with JVM by using Runtime object.
→ Runtime present in lang pkg & it is Singlton class.

```
> Class HeapDemo {
      p s v m (St[]a){
          Runtime r = Runtime.getRuntime();
          Sop(ln("Max Mem:" + r.maxMemory());
          Sop(ln("Initial :" + r.totalMemory());
          Sop(ln("Free :" + r.freeMemory());
          Sop(ln("Consumed :" + r.totalMemory() - r.freeMem());
```



✗ How to set min & max heap sizes?

```
> java -Xmx512m HeapDemo ↵
> java -Xms64m HeapDemo ↵
> java -Xmx512m -Xms64m HeapDemo ↵
```

→ Heap memory is finite memory but based on our requirement we can set min. & max heap size.

3) Stack memory
→ For every thread JVM will create A seperate stack at the time of thread creation.
→ Each & every method call performed by that thread will be stored in stack including local var. also.
→ After completing an method the corresponding entery from the stack will be removed.

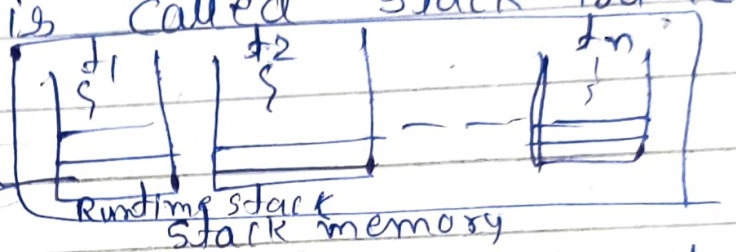
#201 Stack memory

14|May|23

→ After completing all method calls the stack will become empty & the empty stack will be destroyed by JVM just before terminating the thread.

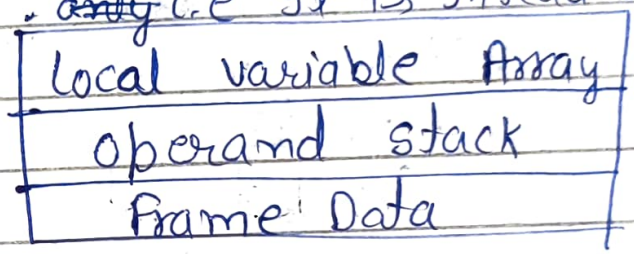
→ Each entry in stack is called stack frame or activation record.

→ The Data stored in stack is available only for the corresponding thread. i.e it is thread safe



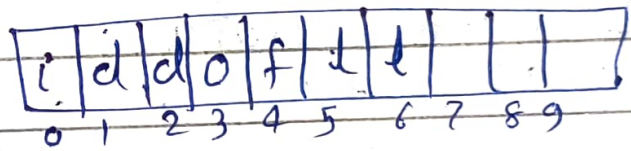
* Stack frame structure

→ Each stack frame contains 3 parts.



I) Local variable Array :-

→ It contains all parameters & local variable of the method.



→ Each slot in the array of 4 Bytes.

→ Value of types int, float & refe. occupy one entry & value of double & long occupy 2 consecutive entries in the array.

→ Byte, short & char values will be converted to int type before storing & occupy one slot.

→ The way to store bool value is varied from JVM to JVM.

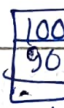
II) Operand stack.

→ JVM uses operand stack as workspace.

→ Some instructions can push the values to operand stack & some can pop values. & some inst. can perform required operations.

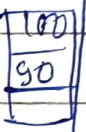
> i load=0
> i load = 1
> iadd
> istor = 2

Before start
local var. Arr.



operand stack

After iload=0
local var

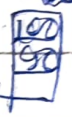


operand

After iload 1
local

operand

LVA



op.

After istore 2

III) Frame data

→ It contains all symbolic references related to that method.

→ It also contains reference to exception table.

4) PC Registers (Program Counter Registers)

→ For every thread a seperate PC Register will be created at the time of thread creation.

→ PC Register contains the adress of current executing instructions.

→ once instruction execution completes automatically PC register is incremented to hold adress of next instruction.

5) Native Method Stack

→ For every thread JVM will create a seperate Native Method Stack.

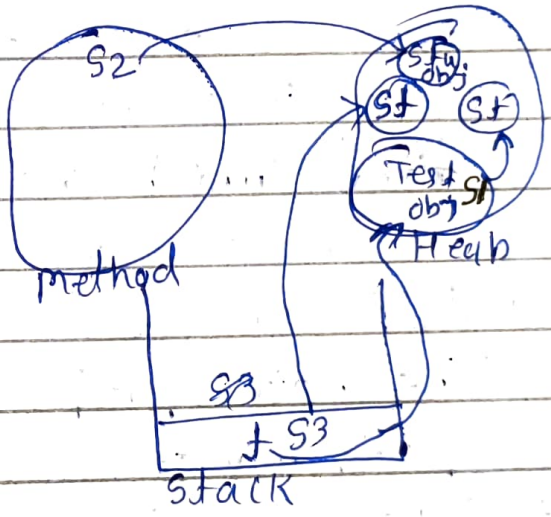
→ All native method call invoked by thread will be stored in corresponding native stack.

Note:→ Method Area, Heap Area & Stack Area are important memory area wrt programmer.

→ Method & Heap area are per JVM. where as stack, PC Register & Native M Stack are per thread

→ Static var → Method, instance var → Heap
local var → Stack area.

```
> class Test{
    Student S1 = new S();
    Static S S2 = new S();
    psvm (ST] a){
      Test t = new T();
      S S3 = new S();}
}.
```



Module-3

#202 JVM Execution Engine

14/May/23

* Execution Engine

→ It is central component of JVM.
→ Execution Engine is responsible to execute java .class files.
→ It mainly contains two component.
1) Interpreter 2) JIT Compiler.

1) Interpreter

→ It is responsible to read byte code & interpret to machine code(Native code) & execute the machine code line-by-line.
→ The problem with interpreter is it interpretes everytime even same method invoked multiple times which reduces performance of the System.
→ To overcome this problem SUN introdues JIT Compiler.

2) JIT Compiler

→ The main purpose of JIT Compiler is to improve performance.
→ Internally it maintains a seperate count for every method whenever JVM come across any method call first that method will be interpreted normally by the interpreter & JIT compiler increments corresponding count variable.
→ This process will be continued for every method once if any method count reaches threshold value then it identifies that method is repeatedly used method (Hot Spot). immediatly JIT compiler.

compiles that method & generates corresponding native code.

→ Next time if JVM come across that method call then JVM uses native call directly & executes it instead of interpreting once again. So, that performance of the system will be improved.

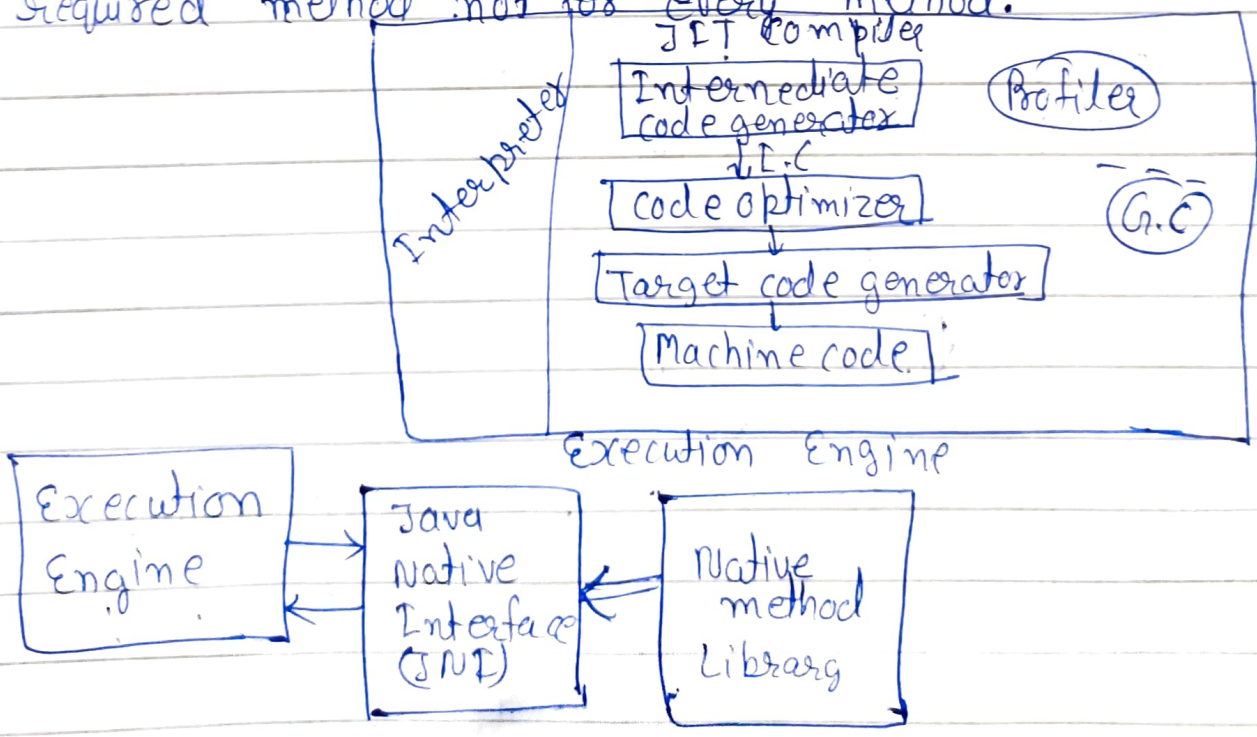
→ The threshold count varied JVM to JVM.

→ Save Advance ~~JVM~~ Jit compilers will re-compile generated native code if count reaches threshold value 2nd time.

→ Internally profiler, which is part of JIT compiler is responsible to identify hot-spot.

Note:- JVM interpretes total program atleast once.

→ JIT compilation only applicable only for repeadly required method not for every method.



#203 Complete JVM architecture

14 May 23

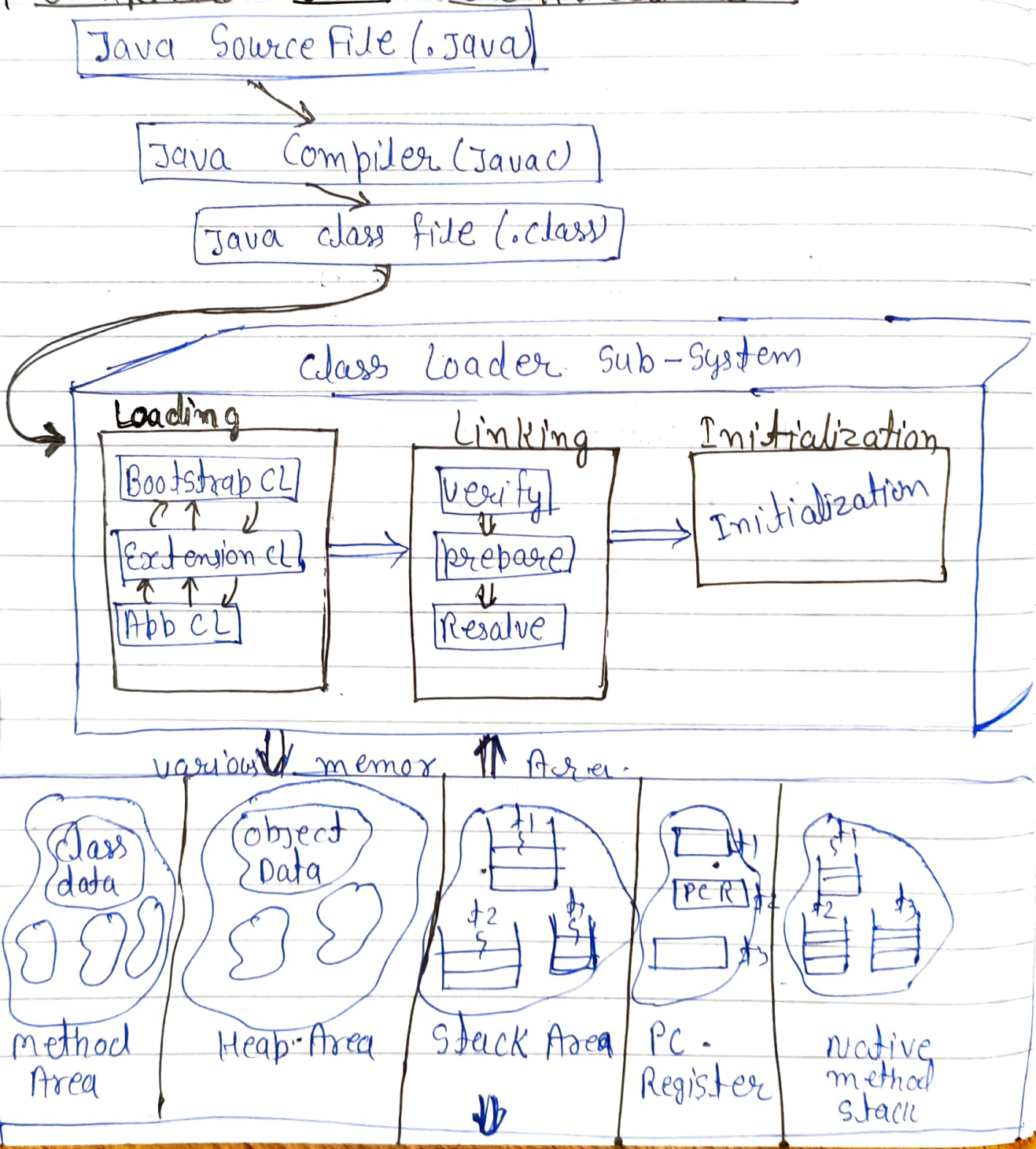
* Java Native Interface (JNI)

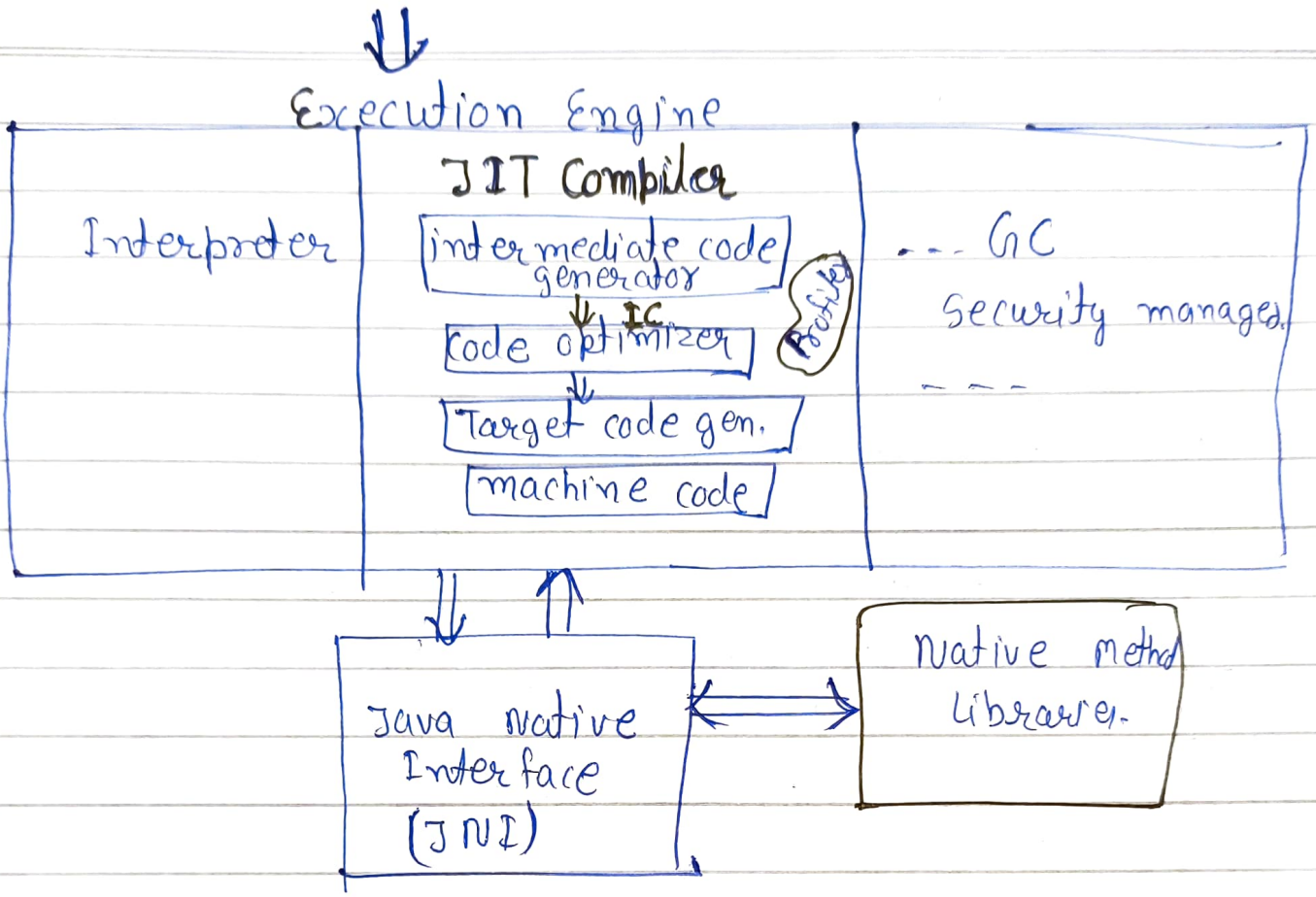
→ JNI acts as mediator for Java method calls & corresponding native libraries.

→ Native method Library holds native lib. info.

→

* Complete JVM architecture





* Class File Structure.

> class File {

magic_number ;
minor_version ;
major_version ;
constant_pool_count ;
constant_pool [] ;
access_flags ;
this_class ;
super_class ;
interface_count ;
interface [] ;
fields_count ;
fields [] ;
methods_count ;
methods [] ;
attributes_count ;
attributes [] ;

}

0XCAFEBABE.
↳ Magic number.

⇒ Magic no. :- The first 4 byte of class file is magic no. this is pre-defined value used by JVM to identify .class file is generated by valid JVM

⇒ Minor & Major version :-
→ Major & minor version represents .class file version. 1.5V(49.0), 1.6V(50.0), 1.7V(51.0)
→ JVM will use these versions to identify which version of compiler generates the current .class file

M.m
Major ↓ minor

⇒ Constant pool count :- It represent no. of constant present in constant pool.
⇒ access_flag :- about modifiers.
⇒ this_class :- fully qualified name of the class
⇒ Super_class :- fully qualified name of immidiate super

> Javap -verbose Test.class
↳ To get all class level info.