Introduction to JVM

Rajeev Gupta

Topics

- JVM, JRE, JDK
- basic arch of JVM
- Class loader sub system
  - loading ,linking, initialization
- Types of class loader
  - boot strap class Loader
  - extension class Loader
  - application class Loader
- How class loader works?
- Need of customized class Loader
- Pseudo code to define customized class Loader
- Java Memory management
- Various memory area of JVM
  - Method area
  - heap area
  - stack area
  - PC register area
  - Native method area

- Garbage collector
  - Issue of memory leak, soft link
  - Running Java visual VM
  - Mark and sweep algorithm
  - Generational collection
  - Analysis heap dump
- Turning the JVM
  - Various parameter for turning JVM
  - Type of GC: Serial, Parallel and mostly concurrent
- Execution engine

Topics

- **JVM, JRE, JDK**
- basic arch of JVM
- Class loader sub system
  - loading ,linking, initialization
- Types of class loader
  - boot strap class Loader
  - extension class Loader
  - application class Loader
- How class loader works?
- Need of customized class Loader
- Pseudo code to define customized classLoader
- Java Memory management
- Various memory area of JVM
  - Method area
  - heap area
  - stack area
  - PC register area
  - Native method area

- Garbage collector
  - Issue of memory leak, soft link
  - Running Java visual VM
  - Mark and sweep algorithm
  - Generational collection
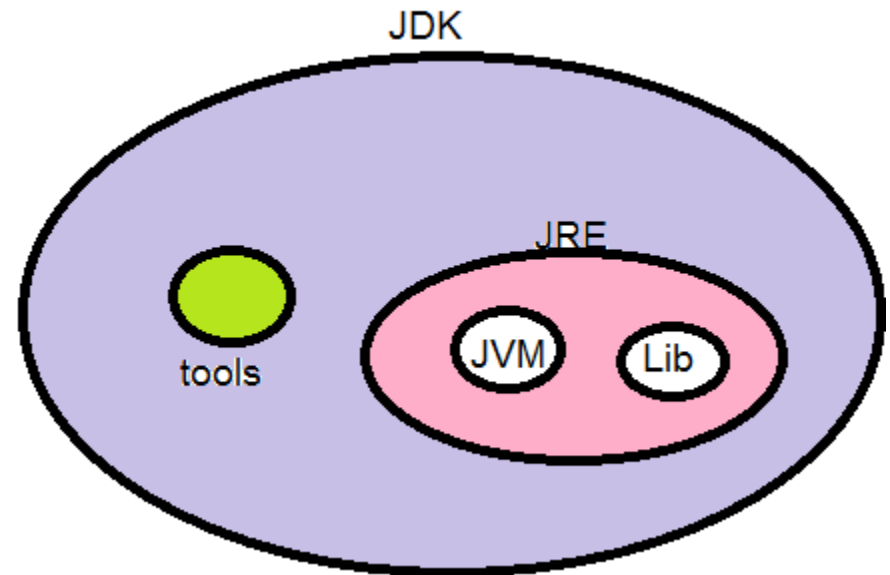 - Analysis heap dump
- Execution engine

Differentiate JVM JRE JDK

- **Java Virtual Machine (JVM)**
  - Is an abstract computing machine.

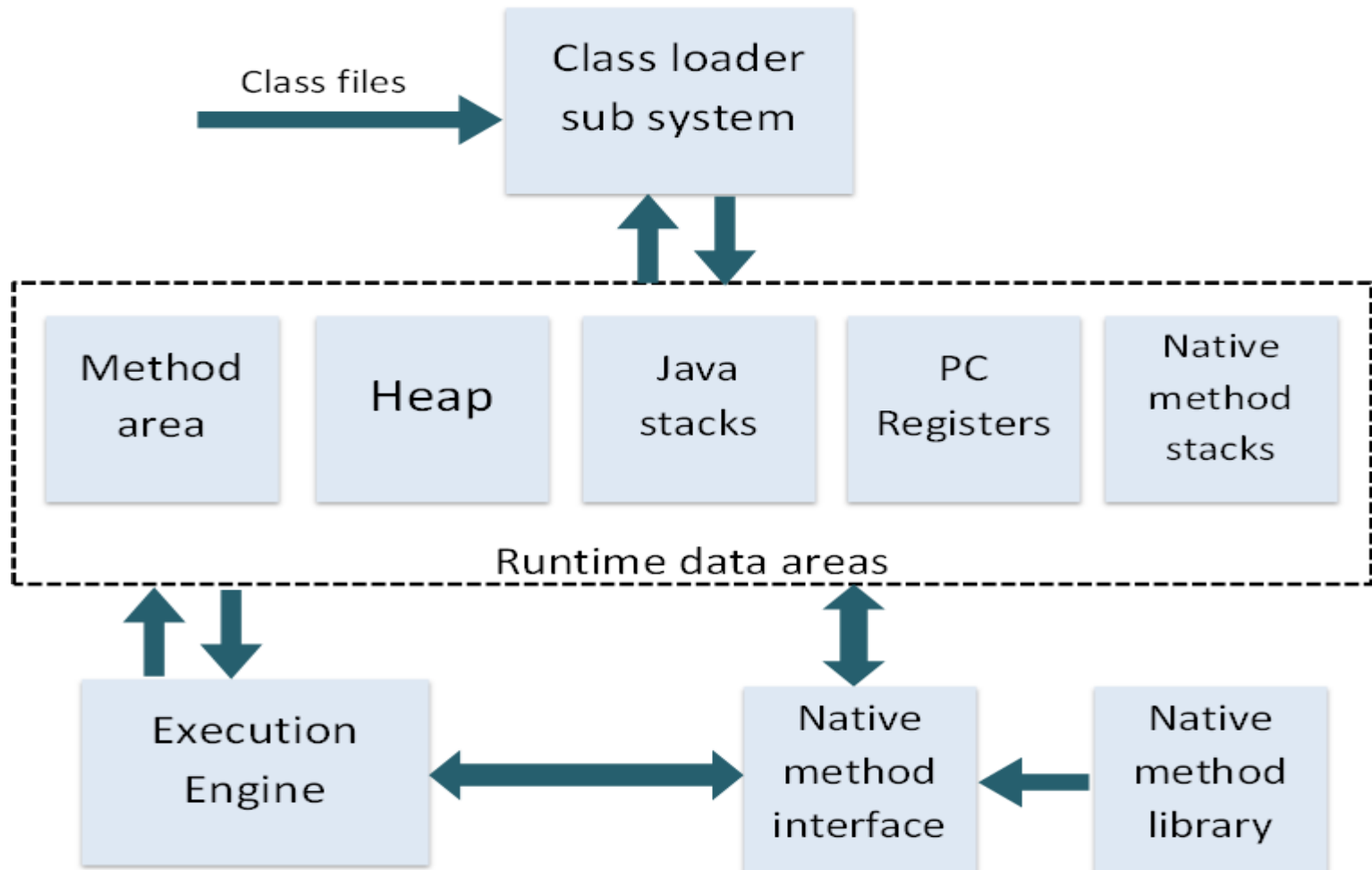- **Java Runtime Environment (JRE)**
  - Is an

Topics

- JVM, JRE, JDK
- **basic arch of JVM**
- Class loader sub system
  - loading ,linking, initialization
- Types of class loader
  - boot strap class Loader
  - extension class Loader
  - application class Loader
- ☐ How class loader works?
- ☐ Need of customized classLoader
- ☐ Pseudo code to define customized classLoader
- ☐ Java Memory management
- ☐ Various memory area of JVM
  - ☐ Method area
  - ☐ heap area
  - ☐ stack area
  - ☐ PC register area
  - ☐ Native method area

- ☐ Garbage collector
  - ☐ Issue of memory leak, soft link
  - ☐ Running Java visual VM
  - ☐ Mark and sweep algorithm
  - ☐ Generational collection
  - ☐ Analysis heap dump
- ☐ Execution engine

basic arch of JVM

Class files →

Class loader sub system

Runtime data areas

| Method area | Heap | Java stacks | PC Registers | Native method stacks |

Execution Engine

Native method interface

Native method library

3 Main component of JVM?

- class loader
- memory area
- execution engine

JVM primary responsibility to load and run .class files
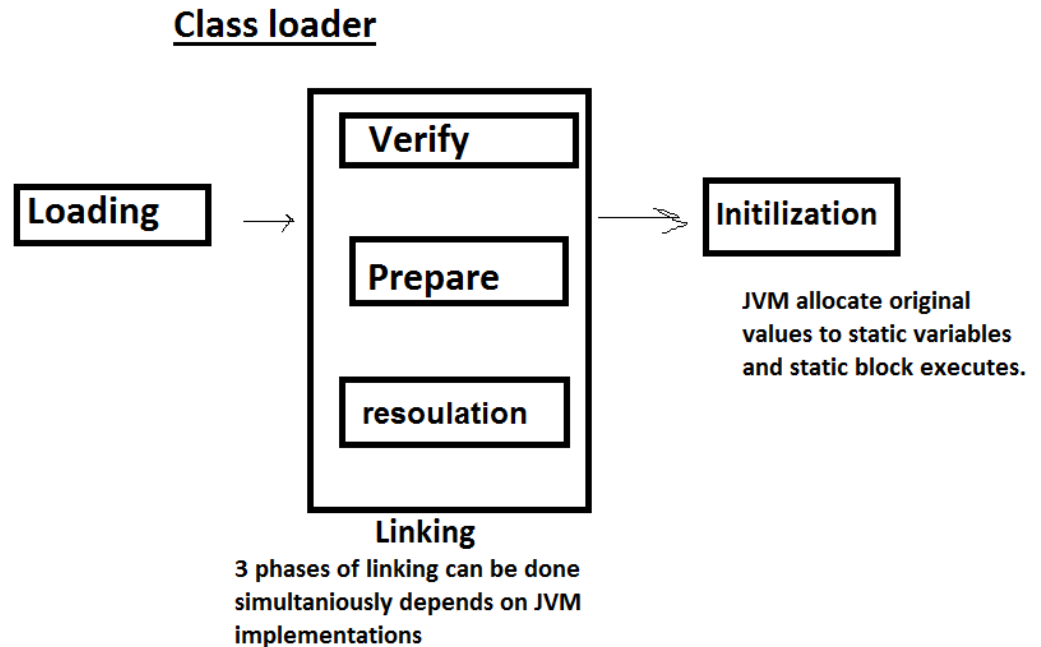
Topics

- JVM, JRE, JDK
- basic arch of JVM
- **Class loader sub system**
  - **loading ,linking, initialization**
- Types of class loader
  - boot strap class Loader
  - extension class Loader
  - application class Loader

- How class loader works?
- Need of customized classLoader
- Pseudo code to define customized classLoader
- Various memory area of JVM
  - Method area
  - heap area
  - stack area
  - PC register area
  - Native method area
- Program ot display heap memory statistics
- How to set max and min heap size
- Execution engine

Class loader subsystem

- Class loader is mainly responsibly for 3 activity:
- loading
- linking
  - verification
  - perpetration
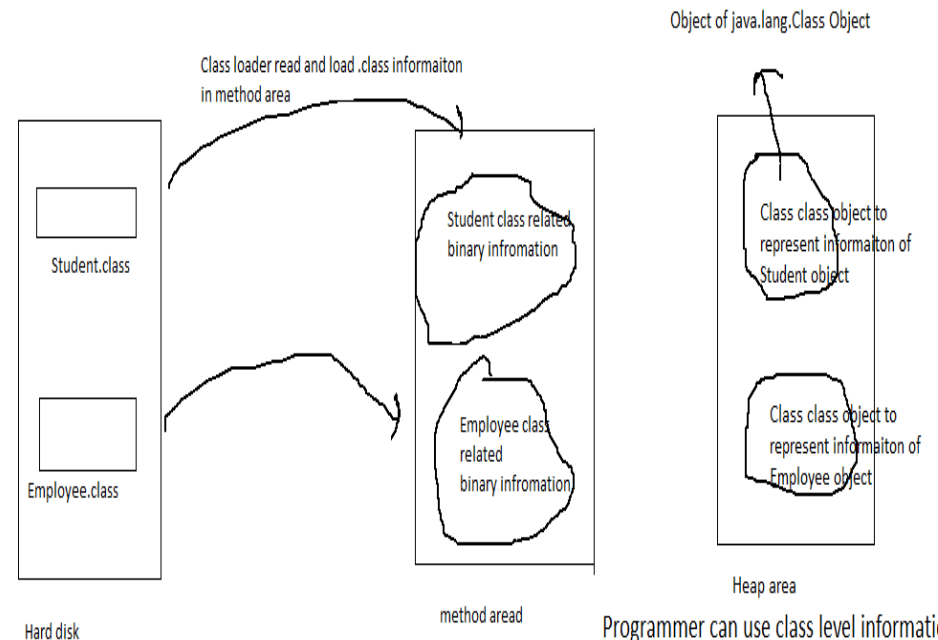 - resolution
- Initialization

Class loader

Loading → Verify, Prepare, resoulation (Linking)

3 phases of linking can be done simultaniously depends on JVM implementations

Initilization

JVM allocate original values to static variables and static block executes.

Class Loading process

During loading phase class loader
read .class file from HD and dumped
method area

```
Student.class --> Stored in method
(HD)                    area of JVM

What information is stored in method area about
a .class file?

1> fully qualified class name
2> fully qualified info about immediate parent
3> whether this class is interface, class or enum
4> method/ constructors/variable information
5> modifer information
6> constant pool information
```
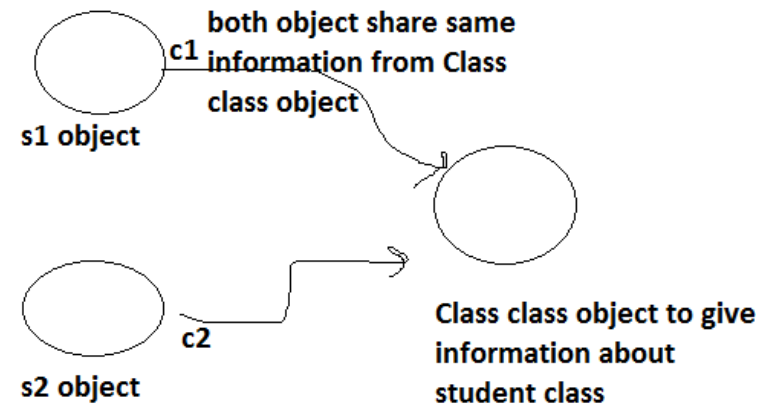
Class loader read and load .class informaiton
in method area

Object of java.lang.Class Object

Student.class

Employee.class

Hard disk

Student class related
binary infromation

Employee class
related
binary infromation

method aread

Class class object to
represent infromaiton of
Student object

Class class object to
represent infromaiton of
Employee object

Heap area

Programmer can use class level information from
this class object using Java reflection!

After loading class information in method area , JVM
create an Class class information binary information and
stored in Heap area

Class Loading process

• For every loaded .class file only one class object will be created, even though we are using class multiple times in our applications

```
class Demo{
        psvm(..........){
                Student s1=new Student();
                Student s2=new Student();

                Class c1=s1.getClass();
                Class c2=s1.getClass();

                Sysout(c1.hashCode());
                Sysout(c2.hashCode());
                Sysout(c1==c2);//true

        }
}
```


s1 object

c1

both object share same information from Class class object

s2 object

c2

Class class object to give information about student class

Getting class information using java reflection

- Programmer can use Class class information using java re

```
Ex:
class Student{
        private String name;
        private int rollNumber;
        //..........|
}

....
Class c=Class.forName("Student");
//hay jvm load class Student and give me handler to Class class object

//gettting method information

//java.lang.reflect
Method[]m=c.getDelaredMethod();
for(Method m1:m){
        Sysout(m1);
}

//gettting Fields information
Field[]m=c.getDelaredMethod();
for(Field m1:m){
        Sysout(m1);
}
```

Linking phase

Linking consist of 3 activities:

1. verification
2. perperation
3. resoulation

Verification phase

=> it is the process of ensuring that binary representation of a class
is structually correct or not?

=> that is JVM will check wheter .class file generated by valid compiler or not
ie whether .class file is properly formatted or not?

=> Internally byte code verifier which is part of class loader sub system is
responsible for this activity

ByteCode Verifier is component of classloader subsystem that verify the .class code
        => if verification fail then we will get RE: java.lang.VerityError

This is how java is secure!

Perpetration and resolution phase

2. perperation
=============
        In this phase JVM will allocate memory for the class level static variables and
        assign default values (not original values)
        eg static int i will be assigned with 0

        Note: original values are assigned in initialization phase

3. resoulation
===============
        it is the process of replacing symbolic reference used by the loaded type with original references
        Symbolic ref are resouved into direct ref by searching through method aread to locate referenced entity.

        Programmer friendly symboles are translated to machine friendly symboles in method area.

Topics

- JVM, JRE, JDK
- basic arch of JVM
- Class loader sub system
  - loading ,linking, initialization
- **Types of class loader**
  - **boot strap class Loader**
  - **extension class Loader**
  - **application class Loader**
- How class loader works?
- Need of customized classLoader
- Pseudo code to define customized classLoader
- Java Memory management
- Various memory area of JVM
  - Method area
  - heap area
  - stack area
  - PC register area
  - Native method area

- Garbage collector
  - Issue of memory leak, soft link
  - Running Java visual VM
  - Mark and sweep algorithm
  - Generational collection
  - Analysis heap dump
- Execution engine

Type of class loaders

Every class loader subsystem contain 3 class loaders:

boot strap classLoader
          |
 extension classLoader
          |
 applicatin classLoader

=> boot strap classLoader/ Primordial class loader
=> extension classLoader
=> applicatin classLoader/ System class loader

Boot strap class loader

```
boot strap classLoader/ Primordial class loader
--------------------------------------------
=> This class loader is written in native languages

=> responsible for loading core java API classes such (String, StringBuilder etc)
 as class files present in rt.jar  (jdk/jre7/lib/rt.jar)

=> This location is called bootstrap class loader path. ie. Bootstrap class loader is responsible for
loading classes from bootstrap class path.

=> bootstrap class loader is by default available with JVM
```

Extension class loader

extension classLoader

=> it is child of boot strap classLoader. this class loader is responsible to load classes
from extension classLoader.   (jdk/jre7/lib/ext)

```
jdk
|_jre_
      |_lib_
            |_ext
```

This class loader is implemented in java and the corrosponding class .class file name is
sun.misc.Lanucher$ExtClassLoader.class

Application class loader

```
applicatin classLoader/ System class loader
-------------------------------------------

It is the child of extension classLoader.
This class loader is responsible to load classes
from application class apth.

It internally use environment variable class path
Application class loader is implemented in java and corrosponding class file name is
sun.misc.Lanucher$AppClassLoader.class

  Ex:

  .........
  Sysout(String.class.getClassLoader())//null as it is written in native languages

  // bundle Foo.class in a jar file and drop in  (jdk/jre7/lib/ext) folder

  // jar -cvf demo.jar Foo.class

  Sysout(Foo.class.getClassLoader())//sun.misc.Lanucher$ExtClassLoader.class

  Sysout(Test.class.getClassLoader())//sun.misc.Lanucher$AppClassLoader.class
```

Topics

- JVM, JRE, JDK
- basic arch of JVM
- Class loader sub system
  - loading ,linking, initialization
- Types of class loader
  - boot strap class Loader
  - extension class Loader
  - application class Loader
- ☐ **How class loader works?**
- ☐ Need of customized classLoader
- ☐ Pseudo code to define customized classLoader
- ☐ Java Memory management
- ☐ Various memory area of JVM
  - ☐ Method area
  - ☐ heap area
  - ☐ stack area
  - ☐ PC register area
  - ☐ Native method area

- ☐ Garbage collector
  - ☐ Issue of memory leak, soft link
  - ☐ Running Java visual VM
  - ☐ Mark and sweep algorithm
  - ☐ Generational collection
  - ☐ Analysis heap dump
- ☐ Execution engine

Delegation hierarchy, How Class loader works?

How Java classLoader works?

delegation hierarchy principle

searched in

request to load a class file

JVM

class loader sub system

Bootstrap class loader

bootstrap class path
(jdk/jre7/lib/rt.jar)

delegation

Ext class loader

ext class loader path
(jdk/jre7/lib/ext/*.jar)

forward request to application class loader

Application class loader

application class path environment variable
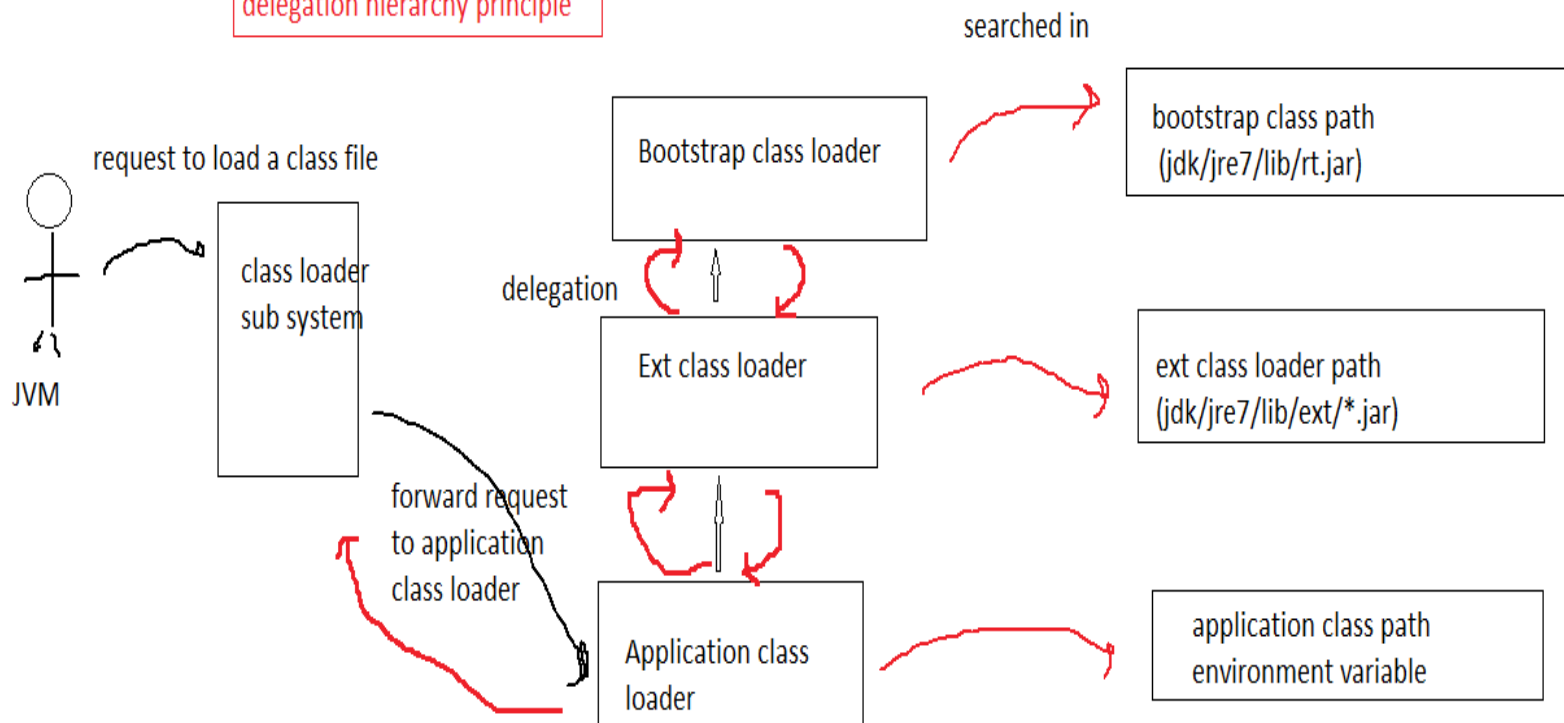
HOw class loader works?
========================

1> class loader follows delegation hierarchy principle

2> wheneverJVm comes across a particular class, first it will check
wheterh the corrospondng class is already loaded or not?

3> If it is already loaded in memory area then JVM will use that loaded class

4> If it is not already loaded then JVM requests class loader sub system to load
that particular class, then class loader sub system handover
that request to applicatin class loader

5> Application class loader delegates request to extension class loader and
ext class loader in turn delegates to boot strap class loader

6> boot strap class loader searches in boot strap class path (JDK/JRE/lib).
If the specified class is available then it will be loaded.
Otherwise bootstrap class loader delegates the request to extension class loader

7> Extension class loader will searches in extension class path(JDK/JRE/Lib/ext).
If the specified class is available then it will be loaded.
otherwise it delegates the request to application class

8> Application class loader will search in application class path.
If the specified class is already available then it will be loaded otherwise
 we will get RE: ClassNotFoundExecption

Topics

- JVM, JRE, JDK
- basic arch of JVM
- Class loader sub system
  - loading ,linking, initialization
- Types of class loader
  - boot strap class Loader
  - extension class Loader
  - application class Loader
- ☐ How class loader works?
- ☐ **Need of customized classLoader**
  - ☐ **Pseudo code to define customized classLoader**
- ☐ Java Memory management
- ☐ Various memory area of JVM
  - ☐ Method area
  - ☐ heap area
  - ☐ stack area
  - ☐ PC register area
  - ☐ Native method area

- ☐ Garbage collector
  - ☐ Issue of memory leak, soft link
  - ☐ Running Java visual VM
  - ☐ Mark and sweep algorithm
  - ☐ Generational collection
  - ☐ Analysis heap dump
- ☐ Execution engine

Need of custom class loader?

Default class loader load class files once...

```
Student s=new Student();
...
...
Student s=new Student();

...
..
Student s=new Student();
...
Student s=new Student();
..
Student s=new Student();
```

Customized class loader
check for updates.
if update available then
load updated varsion
otherwise use
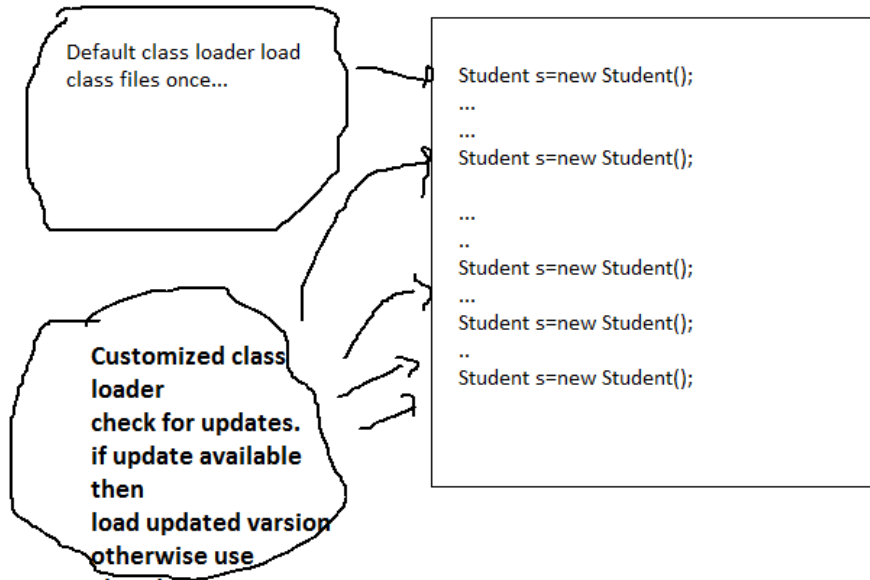
```
Need of customized classLoader?
=================================
If our programming req is not satisified with predefine class loader,
 we can define custom class loader

Ex: Default Class loader will load class files only once, what if
class def is modified afterward,
class loader dont do that update....

Our programm may miss updated information...  we need customized class loader....
```

Pseudo code for custom class loader?

```
Pseudo code to define customized classLoader?
=============================================
 => In order to write custom class loader
        We have to write a class that must extends ClassLoader class.

  => In that class we have to overwrite loadClass() method


 //Pseudo code to define customized classLoader
 ----------------------------------------
 public class CustomizedClassLoader extends ClassLoader{
        public Class loadClass(String cname)throws ClassNotFoundExcetion{
                check wheterh updated version is available or not?

                if  (updated version is avaliable then load updated version and return
                    corropsonding class Class object)
                otherwise
                        return Class Object of already loaded class
        }

 }
```

Topics

- JVM, JRE, JDK
- basic arch of JVM
- Class loader sub system
  - loading ,linking, initialization
- Types of class loader
  - boot strap class Loader
  - extension class Loader
  - application class Loader
- How class loader works?
- Need of customized classLoader
- Pseudo code to define customized classLoader
- **Java Memory management**
- **Various memory area of JVM**
  - **Method area**
  - **heap area**
  - **stack area**
  - **PC register area**
  - **Native method area**

- Garbage collector
  - Issue of memory leak, soft link
  - Running Java visual VM
  - Mark and sweep algorithm
  - Generational collection
  - Analysis heap dump
- Execution engine

Various memory area of JVM

```
Various memory area of JVM
    1. Method area           ------ Per JVM
    2. heap area             ------ Per JVM
    3. stack area            ------ Per thread
    4. PC register area      ------ Per thread
    5. Native method area    ------ Per thread
```

Heap area

```
heap area:
=========
=> All objects with instance varaible stored in heap, all thread access same heap
so heap in not thread safe
=> Heap area is per JVM
=> Heap area is created at JVM startup
=> Every array is object in java, hence stored in heap area
=> can be accessed by multiple threads, hence data stored in heap memory is not thread safe
=> need not to be continous


Heap memory statatics?
======================
Program to display heap memory stats...display heap memroy, fre memory, used memory


Java      RunTime class (Singleton class java.lang.*)
App         ------->       JVM

Using Runtime class (Singleton class)    we can get infromation
RunTime  r=new Runtime ();
double mb=1024*1024;
r.maxMemory()/mb;
r.freeMemory()/mb;
r.initalMemory()/mb;


consumed memory=inital memory-consumed memory
```

Heap memory area
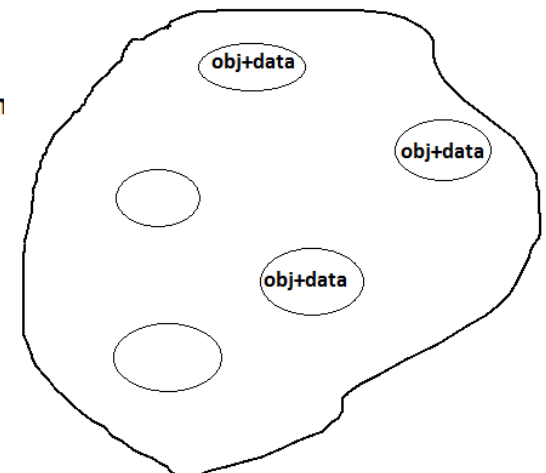
```
How to set max/min heap size?
-----------------------------
heap memory is finite memroy, we can adjust as per our requirments

java -Xmx512M Demo  //will set max heap memory to 512M (default 256Mb)
java -Xms64M Demo   //will set min heap memory to 64M (default 16Mb)

in one go:
java -Xmx512M -Xms64M Demo
```

Stack area

stack area
==========

=> for every thread jvm create one stack area
=> thread safe
=> each method call and local variable stored in stack area
After completing all methods of Foo class RuntimeStack is destroyed.

for every thead jvm create an sepeate stack, at the time of thread creation, each and every method call performed by that thread will be stored in the stack including local variable also. After completing an method corrosponding entry (stack frame) form the stack will be removed.After completing all methods call stack will be empty and that empty stack will be destroyed by the JVM, just before termaniting the thread

Ex.

Ex:
```
class Foo{
    psmv(....){
        m1();
    }

    psv m1(){
        m2();
    }

    psv m2(){
    }
}
```

main thread

thread t1

thread t2

m2()

m1()

main()

m2()

m1()

m2()

m1()

activation record
or stack frame
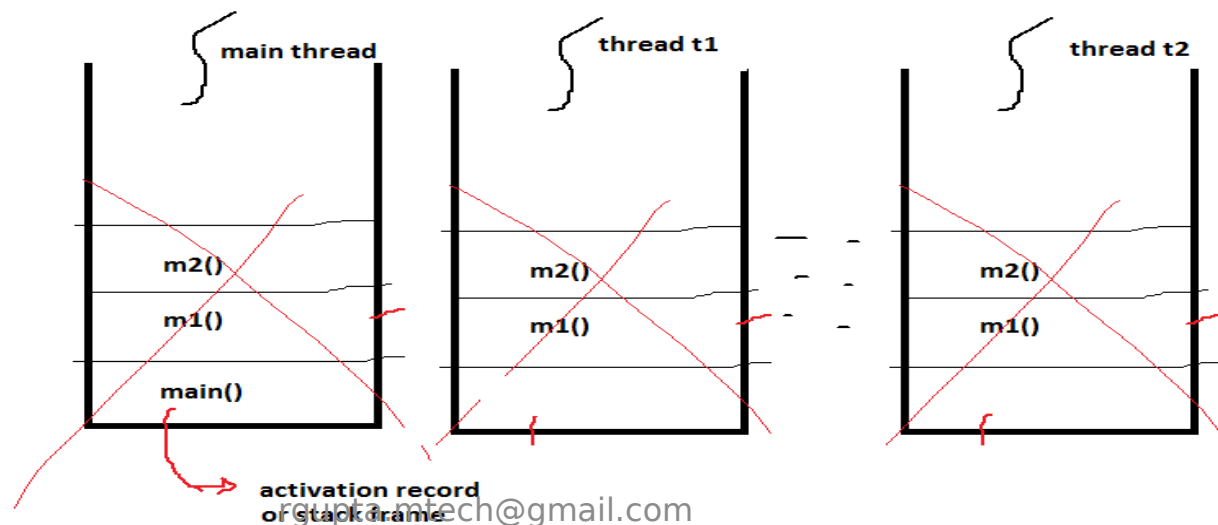
Internal of stack frame?

Each stack frame contain 3 parts:
1. local variable array
         contain array of all local varaible passed in method
2. operand stack
3. Frame data


```
public void foo(int a, float b, boolean f, Bar bar){
}
```

Internal of stack frame?
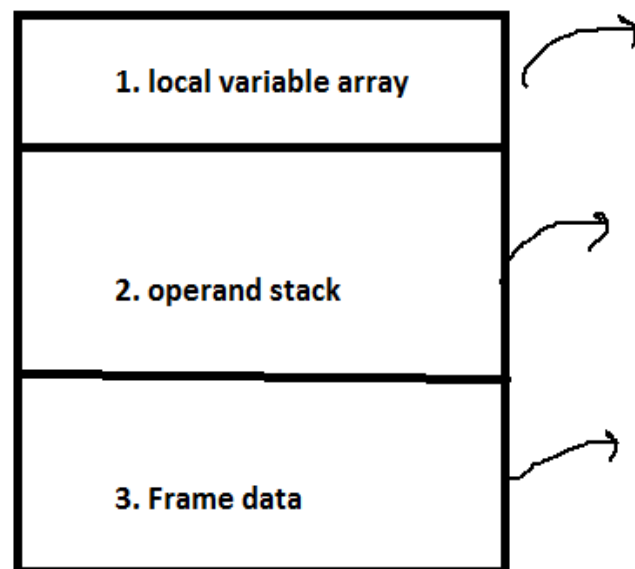
local variable array: store method arguments
  public void foo(int a, float b, boolean f, Bar bar){
  }   4 byte
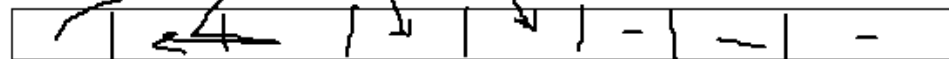
byte etc promoted to int
and occupy 1 slot
boolean jvm dependent

1. local variable array
2. operand stack
3. Frame data

Stack frame

aka of erable roughf work..
.operand stack is workspack to performing and store intermediate result of processing
used by JVM
some instruction push/pop/operation values from operand stack

Threre are many symbols used in method,stored in frame data.

frame data contain pool of all constant symbol used in method.
it also cotain reference to all exception table which provide
corrosponding catch block information in case of exception.

m(....){

frame data

}
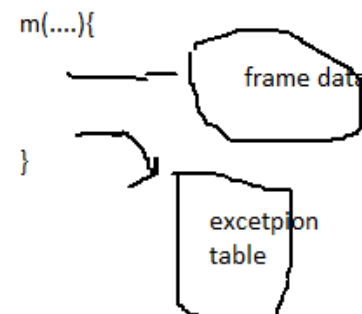
excetpion
table

Operand stack

understanding operand stack frame?

	before operation	After istore0	After istore1	After iadd	After istore2

operation

istore0
istore1
iadd
istore2

before operation

local variable array

| 100 |
| 90 |
| |

operand stack

After istore0

local variable array

| 100 |
| 90 |
| |

operand stack

| |
| 100 |

After istore1

local variable array

| 100 |
| 90 |
| |

operand stack

| 90 |
| 100 |

After iadd

local variable array

| 100 |
| 90 |
| |

operand stack

| |
| 190 |

After istore2

local variable array

| 100 |
| 90 |
| |
| 190 |

operand stack

| |
| |

Program counter register

```
PC (Program counter ) regester
-----------------------------

Internally used by JVM, for every thread an seperate flow of execution,
in each flow we need to keep information of next instruction.

from where JVM instrcution get address of current execution instruction

for 10 threads 10 PC regester is required.
```

Topics

- JVM, JRE, JDK
- basic arch of JVM
- Class loader sub system
  - loading ,linking, initialization
- Types of class loader
  - boot strap class Loader
  - extension class Loader
  - application class Loader
- ☐ How class loader works?
- ☐ Need of customized classLoader
- ☐ Pseudo code to define customized classLoader
- ☐ Java Memory management
- ☐ Various memory area of JVM
  - ☐ Method area
  - ☐ heap area
  - ☐ stack area
  - ☐ PC register area
  - ☐ Native method area

☐ **Garbage collector**
  ☐ **Issue of memory leak, soft link**
  ☐ **Running Java visual VM**
  ☐ **Mark and sweep algorithm**
  ☐ **Generational collection**
  ☐ **PemGen space problem**
  ☐ **Analysis heap dump**
☐ Execution engine

Garbage collector

- GC is used to remove objects that goes out of scope.
- We can create object by "new" but no keyword as "delete" in Java
- what GC gets started?
  - GC is application program that runs inside jvm, when we invoke " java" command it get started as demon process

Escaping reference

- Consider CustomerRecord code, it have issue of escaping reference

```java
public class CustomerRecords {
    private Map<String, Customer> records;

    public CustomerRecords() {
        this.records = new HashMap<String, Customer>();
    }

    public void addCustomer(Customer c) {
        this.records.put(c.getName(), c);
    }

    public Map<String, Customer> getCustomers() {
        return this.records;
    }
}
```

```java
CustomerRecords records = new CustomerRecords();

Map<String, Customer> myCustomers =
    records.getCustomers();

myCustomers.clear();
```

rgupta.mtech@gmail.com

How GC works?

For C/C++ programmer must write code to indicate freeing memory.
If memory leak only solution is to restart system, what if it is web server?
Eventually system crash
Finding memory leak is very difficult
Java avoid memory leak by :
       1. Memory is not acquire from OS directly as happens in C/C++,
            Memory provided by JVM and OS level memroy leak is not a possibility in java

       2. Java provide GC

How GC come to know an object is candidate of GC?

Any object on the heap which can not be reached through a reference from the stack is "eligible for GC"

stack

heap

circular ref
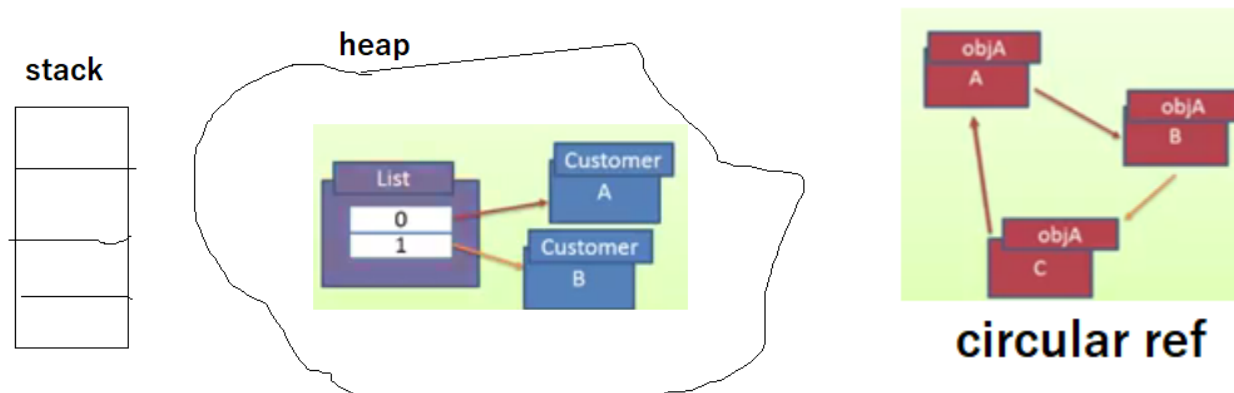
what about System.gc()

About System.gc() ?

System.gc()?

Can not say when when GC will run,even GC may not run at all after calling System.gc()
When GC run application is suspended, till GC process is not complete, therefor calling GC
is not good idea
When GC remove an object, finilize() method is called.
finilize is not reliable

```java
public class Customer {
    private String name;

    public Customer(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }

}
```

```java
Runtime runtime=Runtime.getRuntime();

long availableMemory=runtime.freeMemory();
System.out.println("available memroy:"+availableMemory/1024+" k");

//lets create a tons of garbage
for(int i=0;i<1000000;i++){
    Customer c;
    c=new Customer("raja"+i);
}
//Now check available memory
availableMemory=runtime.freeMemory();
System.out.println("available memroy:"+availableMemory/1024+" k");

System.gc();

//Now check available memory after System.gc()
availableMemory=runtime.freeMemory();
System.out.println("available memroy:"+availableMemory/1024+" k");
```

<terminated> Demo [Java Application] C:\Program Fil
available memroy:62549 k
available memroy:68557 k
available memroy:90348 k

rgupta.mtech@gmail.com

Effect of finalize() method

- Java run finalized when GC remove an object from heap, but it is not guaranteed. Don't trust on this method
- When below code is run, finalized() is called 10 time not 100 time.\
- GC don't run just because we call it, the purpose of GC is to keep Heap nice and clear himself.

```java
public class Customer {
    private String name;

    public Customer(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }

    @Override
    public void finalize() throws Throwable {
        super.finalize();
        System.out.println("this object is GC");
    }
}
```

```java
public static void main(String[] args) {
    Runtime runtime=Runtime.getRuntime();

    long availableMemory=runtime.freeMemory();
    System.out.println("available memroy:"+availableMem

    //lets create a tons of garbage
    for(int i=0;i<100;i++){
        Customer c;
        c=new Customer("raja"+i);
    }
    //Now check available memory
    availableMemory=runtime.freeMemory();
    System.out.println("available memroy:"+availableMem

    System.gc();

    //Now check available memory after System.gc()
    availableMemory=runtime.freeMemory();
    System.out.println("available memroy:"+availableMemory/1024+" k");
}
```

```
<terminated> Demo [Java Application] C:\Program F
available memroy:62524 k
available memroy:62691 k
this object is GC
this object is GC
this object is GC
this object is GC
this object is GC
this object is GC
this object is GC
this object is GC
this object is GC
this object is GC
```

Garbage Eligibility

- Java avoid memory leak by
  - Runtime on a VM (avoid so level memory leak)
  - Adopts a garbage collector strategy
- Soft link
  - an object is reference on the stack even though it is not going to be used again.

Running Java visual VM to analysis application with leak

- Application with soft leak almost consume all the heap and (40 out of 50Mb). Why some of the memory 10mb can not be used.
- Increasing heap size is not the solution.
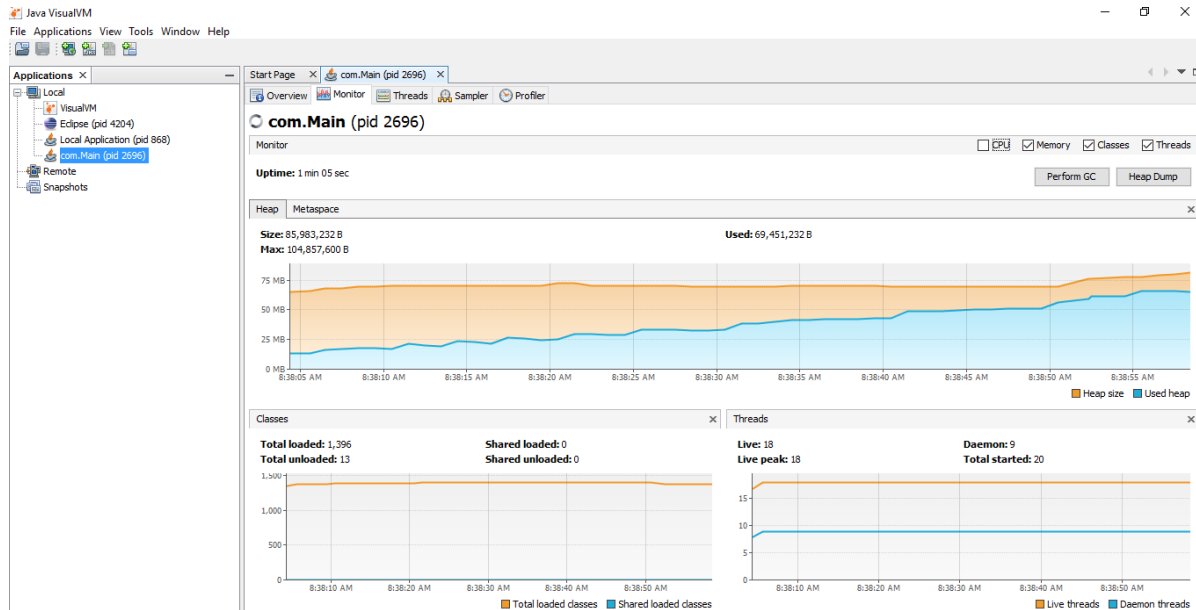
Running Java visual VM to analysis application with leak

- After solving leak issue
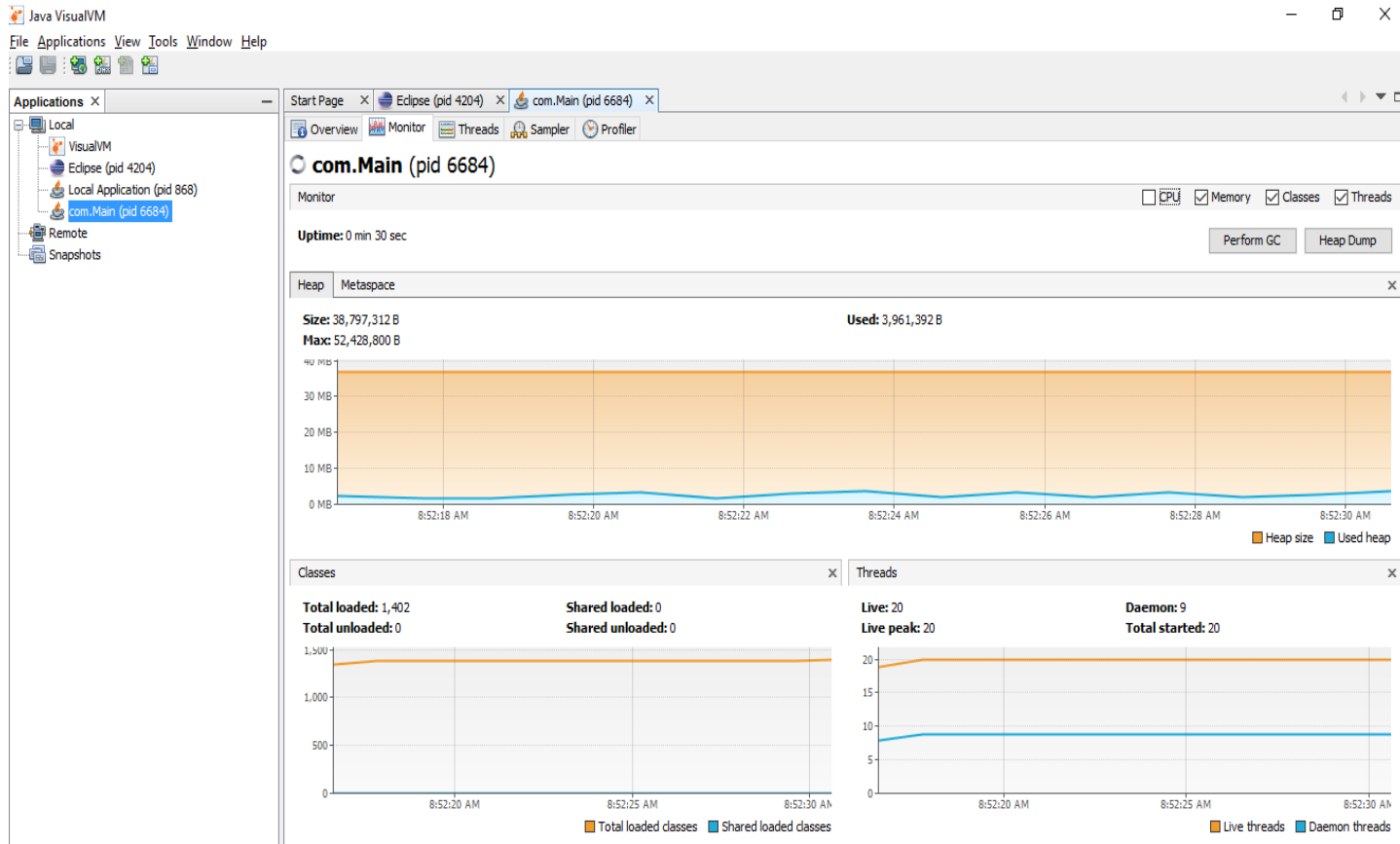
Algorithm GC used?

- **Mark and swap methodology**

Rather then looking for object for deletion, modern GC use cleaver method
GC look for object that need to be rescue and keep them

marking:
-> Program execution is first halt, all running thread suspended
       "stop the world event"
-> GC then check all live instance, simply GC check valid ref from stack.
-> Full scan of heap is done and object that is not ref by valid reference
-> memory is free up
-> finally object are kept in single continus space

Mark and swap methodology

- Step 1: full scan of heap is done

- Step 2: all the valid reference object are marked, and unused objects are swept

- Step 3: reshuffling of objects in continuous memory
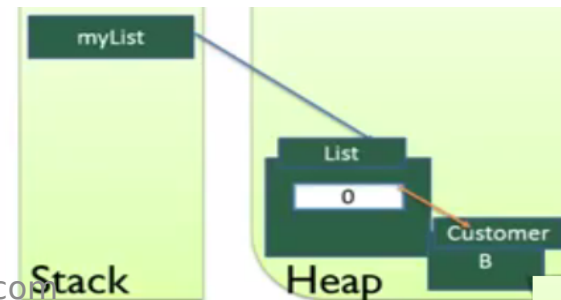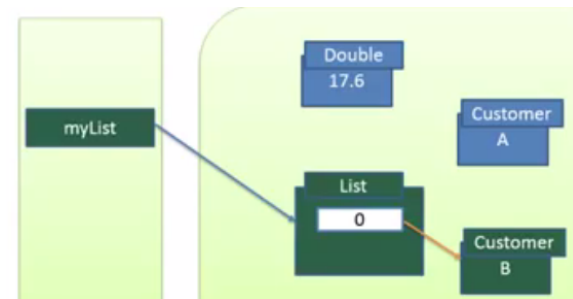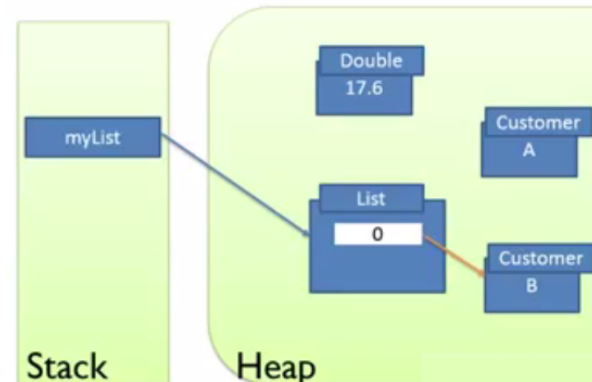
Generational GC

- Generation is way to organized heap

```
If almost everything is garbage in heap then GC have nothing to do and it is very fast

Need of Generational GC
-----------------------

If lots of object that can not be GC because garbage collection is
"stop the world process"

When GC runs application run application is just frozen for few second.
It is not acceptable.

Java used a technique to solve this issue. Java introduce Generational GC

Most object in java lives for very short span of time, if an
object servive on GC proces then it have more chances to service for ever.
```

Generational GC

new object created in young generation space, fill fast,
GC collection work on young generation and it is very quick process, fraction of second
application don't freeze.
=> GC that run on new generation is called "minor GC"

All surviving object moved to old generation

=> GC also run on old generation only if it needed, if old generation heap is filled
=> GC for old generation is called "major collection " and it is slower as compared to minor GC
   as it have bigger memory area to sweep and most of the object in this memory are required to be
   alive

=> compacting process also takes some time
=> major generation takes some seconds as compared to fraction of seconds in case of young generation

Most objects don't live for long
If an object survives it is likely to live forever

| young | old |

| Stack | Heap |
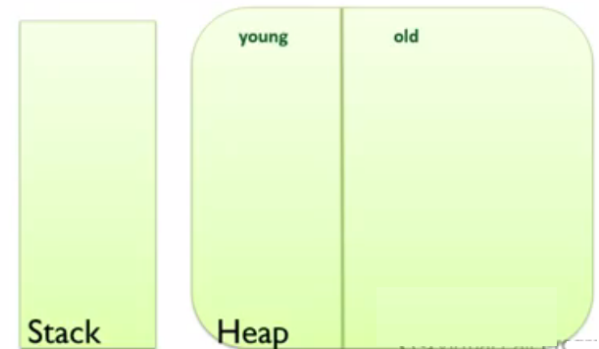
Generational GC

- Object that survive first go process moved to old generation

Study generation of GC

- To study generation of GC we need to install a plug-in in java VisualVM tool that is called Visual GC
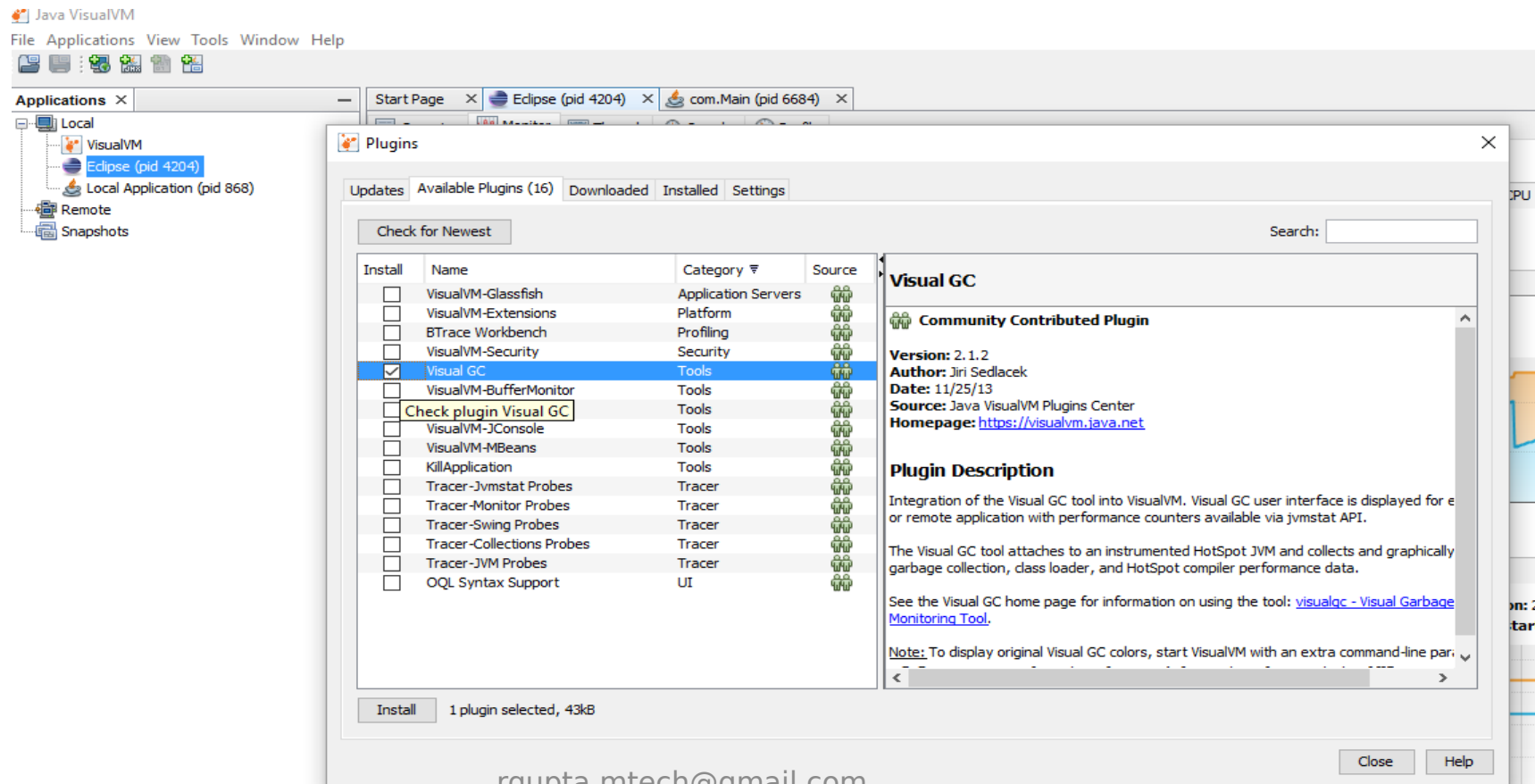
Study generation of GC
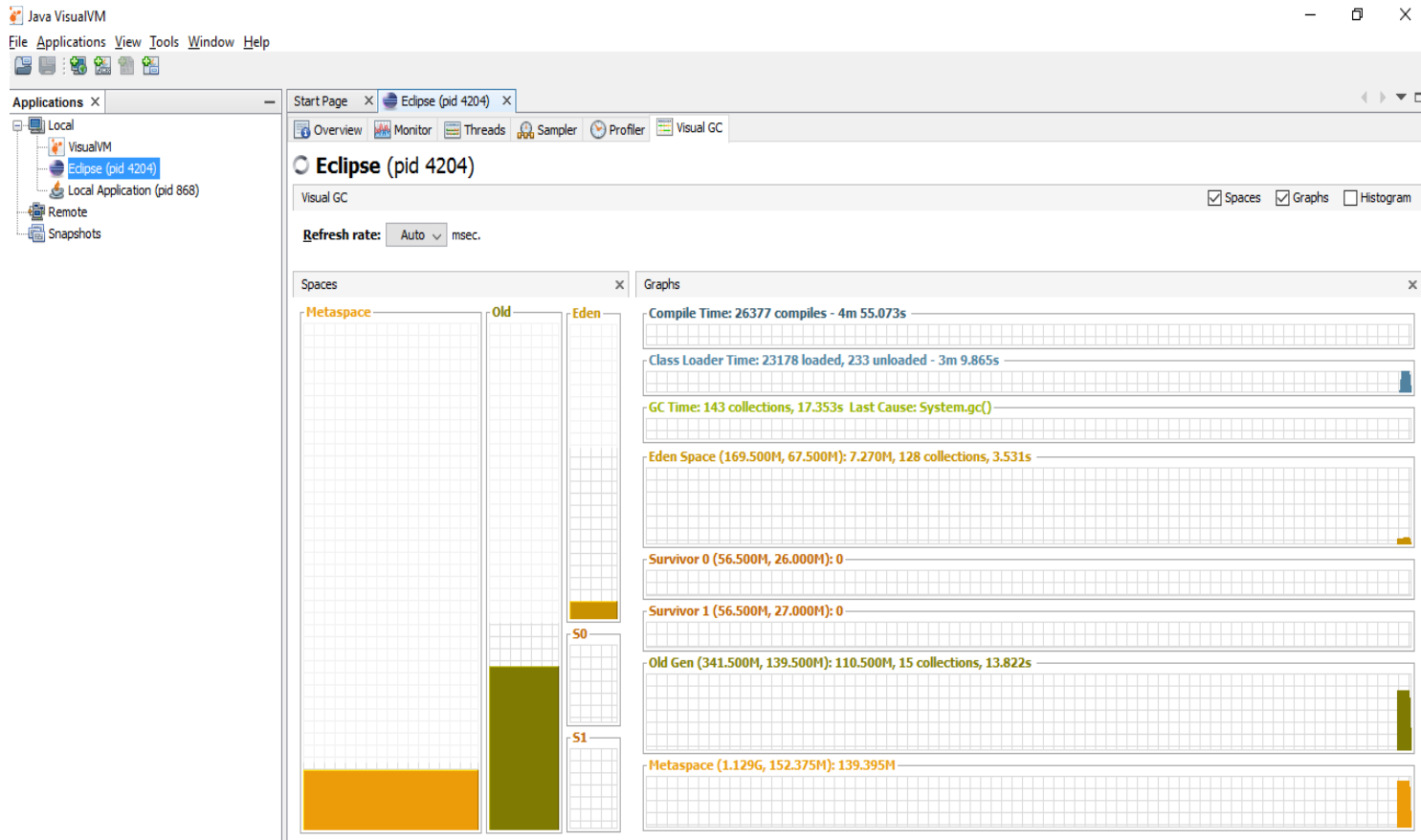
Study generation of GC

new generation is divided into 3 area:
eden, s0 and s1 (serviver space)

when eden is full, GC process happens
object that service moved to s0, next time GC happen any
object are moved to s1. Compacking process can put all
object in s1 and s0 become empty.

Why 10Mb is not used in last example. As some memory
is reserved for s0 and s1.

After an object servive 8 GC porcess as explain above then
it moved to old generation.

Study generation of GC for broken code

- Application is just going to crash ⬜ Memory leak
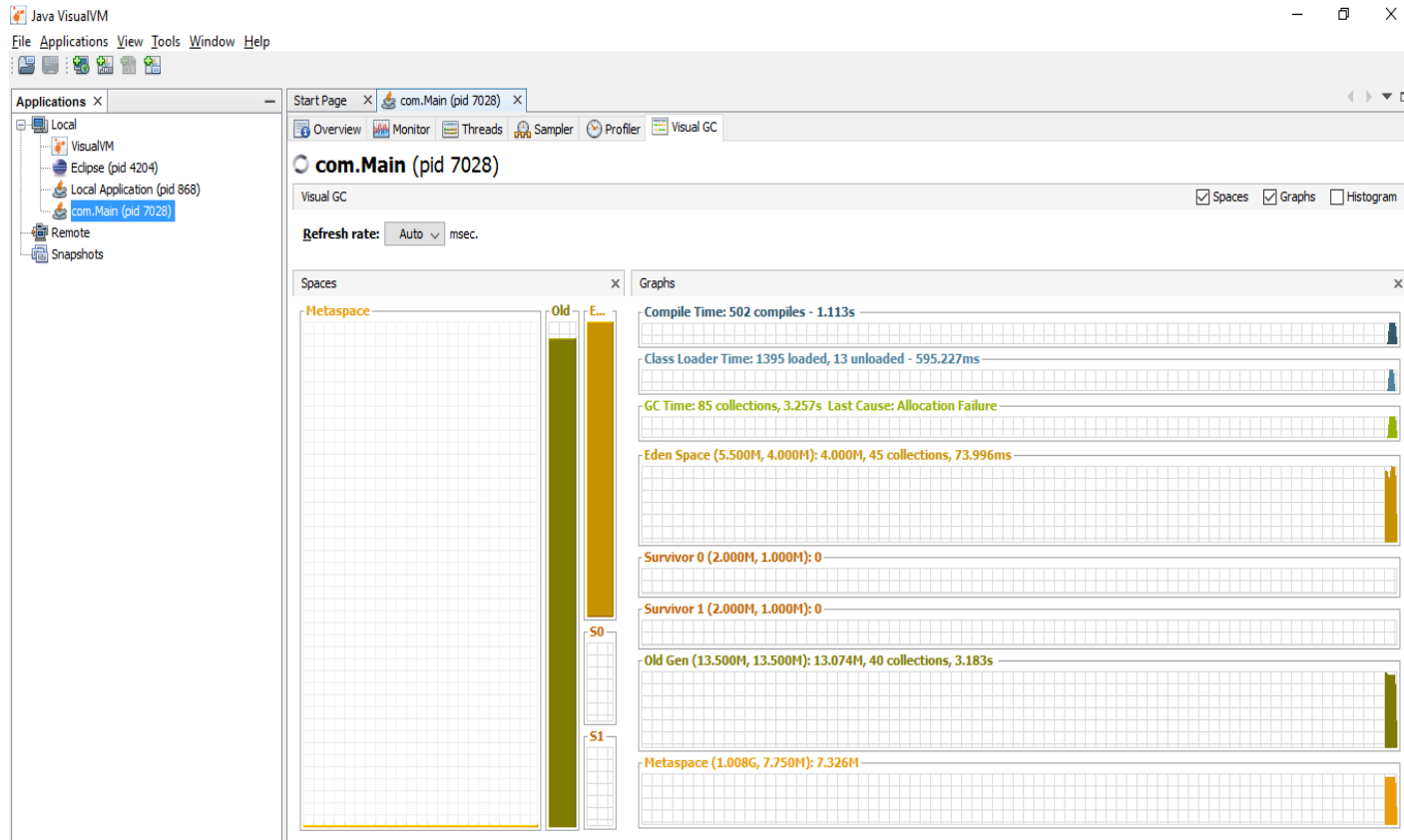
Heap dump analysis

- Heap dump analysis is the technique to understand what cause memory leak in our application.
- Click on monitor tab and Heap dump
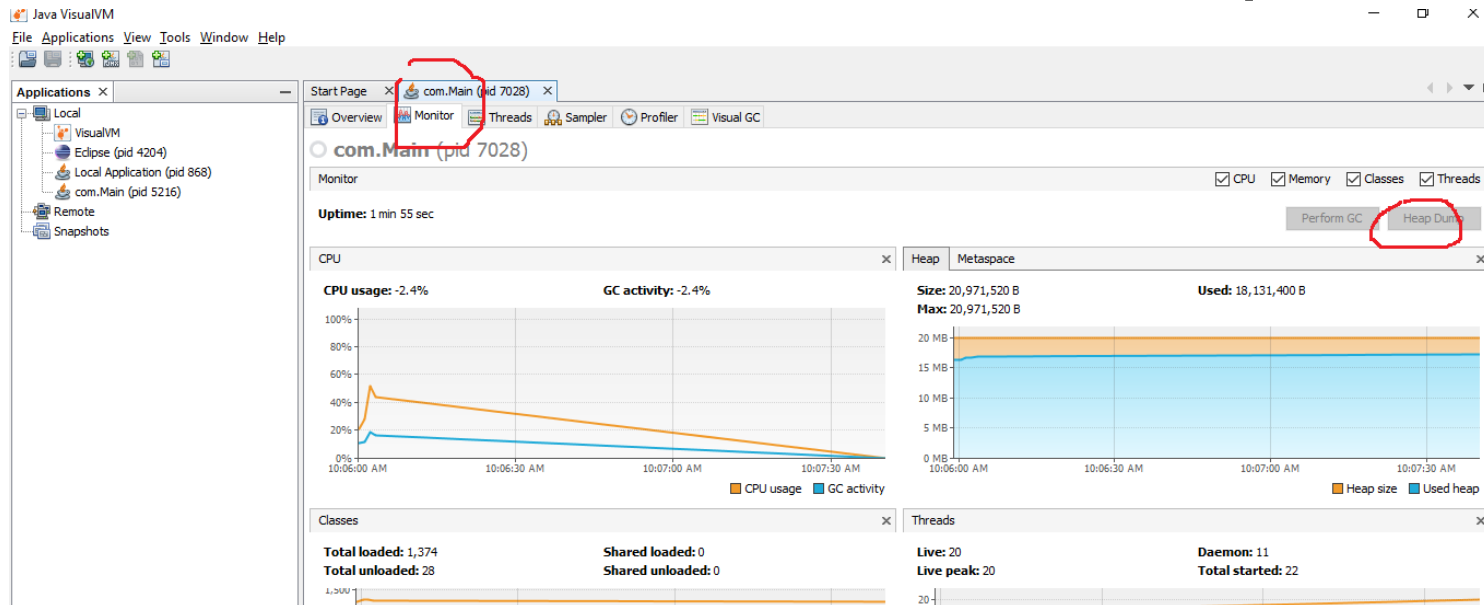
Heap dump analysis

- Copy the location of file and save it.
- Tool to analysis heap dump is called Memory analyzer (MAT), download and install it.

Memory analyzer tool

rgupta.mtech@gmail.com

Perm gen space

- Object in permgen never garbage collection
- Two type of object stored in perm gen
- 1. Internalized string
- 2. class meta data

- Problem:
- Application crashed due to perm gen error
- Solution:
  - 1. increase perm gen space
  - 2. restart application

- If we reply an application on tomcat, old meta data of application is still there on tomcat!
- On a live a server only solution is to stop and restart the server
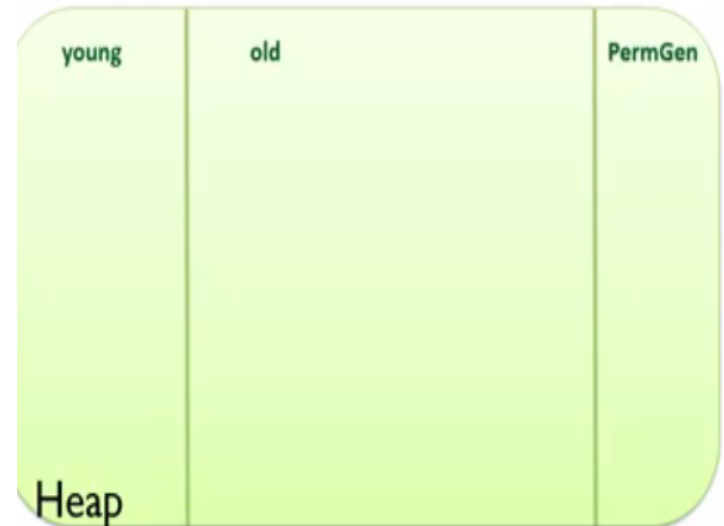
Perm gen space is now removed

In Java 7:
  ⇒ Internalised string now stored in old part of heap
  ⇒ now it is not contribute to perm gen problem

Java 8:
  ⇒ Java remove perm gen totally form architecture
  ⇒ replaced meta space with perm gen
  ⇒ if we redeploy an application all older meta data is deleted :)
  ⇒ Size of meta data can grown to whole memory

Topics

- JVM, JRE, JDK
- basic arch of JVM
- Class loader sub system
  - loading ,linking, initialization
- Types of class loader
  - boot strap class Loader
  - extension class Loader
  - application class Loader
- How class loader works?
- Need of customized class Loader
- Pseudo code to define customized class Loader
- Java Memory management
- Various memory area of JVM
  - Method area
  - heap area
  - stack area
  - PC register area
  - Native method area

- Garbage collector
  - Issue of memory leak, soft link
  - Running Java visual VM
  - Mark and sweep algorithm
  - Generational collection
  - Analysis heap dump
- **Turning the JVM**
  - **Various parameter for turning JVM**
  - **Type of GC: Serial, Parallel and mostly concurrent**
- Execution engine

Topics

- JVM, JRE, JDK
- basic arch of JVM
- Class loader sub system
  - loading ,linking, initialization
- Types of class loader
  - boot strap class Loader
  - extension class Loader
  - application class Loader
- How class loader works?
- Need of customized class Loader
- Ppseudo code to define customized class Loader
- Java Memory management
- Various memory area of JVM
  - Method area
  - heap area
  - stack area
  - PC register area
  - Native method area

- Garbage collector
  - Issue of memory leak, soft link
  - Running Java visual VM
  - Mark and sweep algorithm
  - Generational collection
  - Analysis heap dump
- **Execution engine**

Execution engine

- Is the component that is actually involve in converting the byte code to the execution code.
- Lets see how Execution engine is evolved over the time

- **Java 1.0/1.1**
  - Execution engine started as interpreter, executed one line of code one by one.
  - Java was so slow10-13 time slower then C/C++.
  - If we have while loop 100 time same instruction are going to execute 100 times.

- **Java 1.2**
  - Execution engine improved to use JIT
  - what happen with JIT?
    - Sun provided cache with execution engine. so what ever code is going to be compiled is going to be put in cache. some improvement over Java 1.1
  - Problem?
  -         limited cache

Execution engine over the year

- Java 1.3
  - JIT is replaced by hotspot JVM
  - Here with the hotspot jvm , with the help of some algorithm execution engine can determine critical path of the code ,that much code is going to be place in the cache.

- Now more and more improved hotspot JVM JRockit

- **Just-in-time Compiler (JIT)**
  - JIT is the part of the Java Virtual Machine (JVM) that is used to speed up the execution time.
  - JIT compiles parts of the byte code that have similar functionality at the same time, and hence reduces the amount of time needed for compilation.
  - Here the term "compiler" refers to a translator from the instruction set of a Java virtual machine (JVM) to the instruction set of a specific CPU