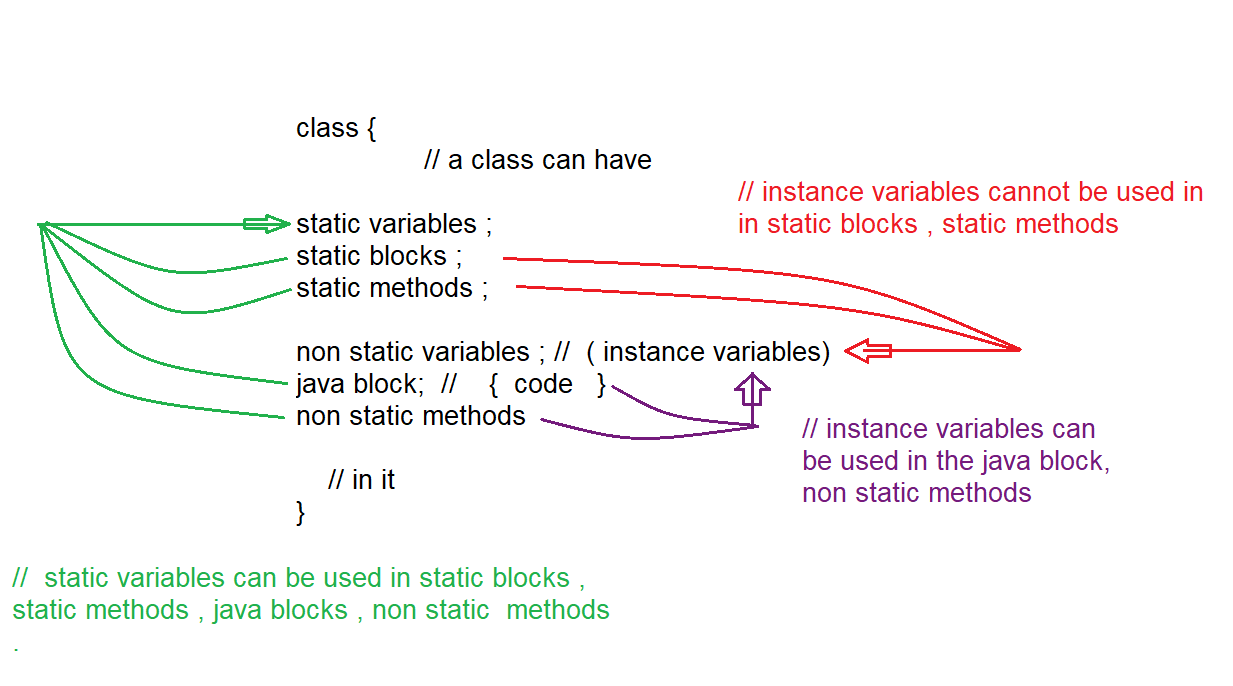
Static keyword :



Note : main() is not the first one to execute in the program , jvm will decide whether to execute the program or not , if program has main() it is executed .

Order of execution of java program

1. Static variable
2. Static block
3. Static method ( main method is also static )
4. Non static variables
5. non static block
6. constructor
7. non static method

jvm evaluates the program in the above order.

but you can invoke them in main() in different order of your choice .

Eg: Order\_Of\_Execution

Note : when object is created 3 things will happen

1. memory for instance variables is allocated
2. java block is executed
3. constructor is executed .

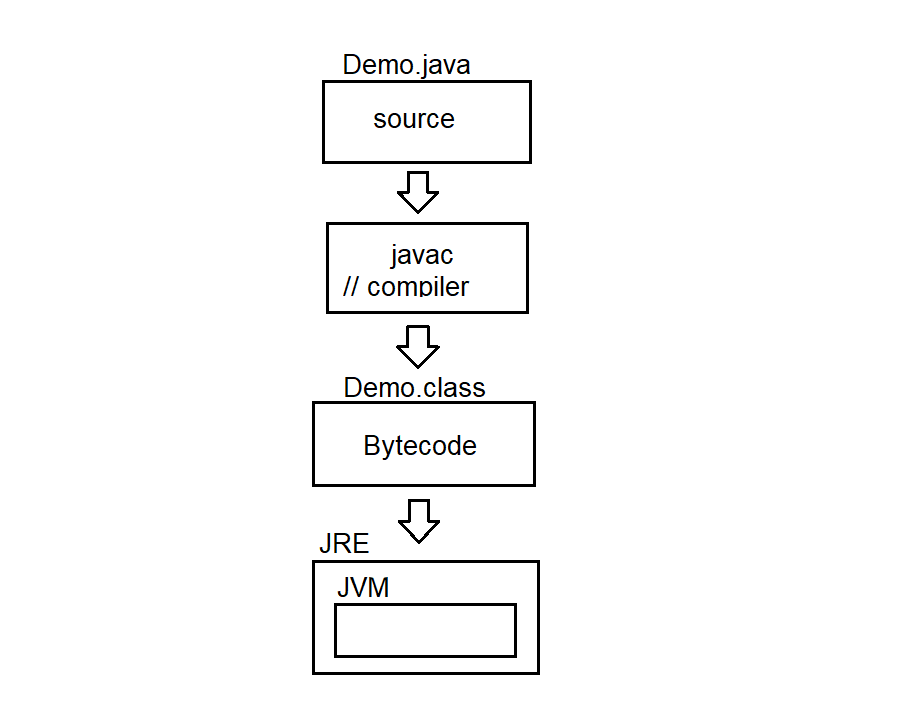
Note : static block will have no name .

Syntax : static {

// Code

}

Program is executed as follows :



JRE (java runtime environment ) internally has JVM ( java virtual machine )

Byte code is given to JRE

JRE gives the byte code to the JVM

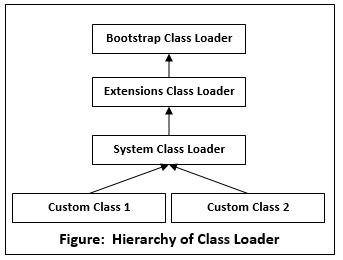
JVM takes that bytecode , and process it internally

Below diagram shows the JVM ARCHITERCTURE ( internal working of JVM )

0-

Java class loader :

Java class loader is an abstract class present in java.lang package . It is used to load classes at runtime.

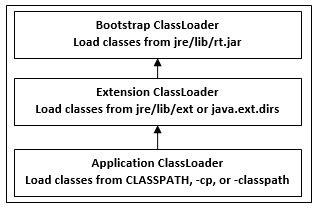


Application class loader is also known as system class loader.

Java class loader works on 3 principles

1. delegation principle : It forwards class loading request to parent class loader. Delegation principle only loads the class if the parent does not find the class (or) loads the class.
2. Visibility principle : It allows the child class loader to see all the classes loaded by the parent class loader . But parent class loader cannot see the classes that are loaded by the child class loader.
3. Uniqueness principle : It allows to load the class once . It is achieved by delegation principle . It ensures that class loaded by the parent is not reloaded by the child class loader.

Types of class loader :

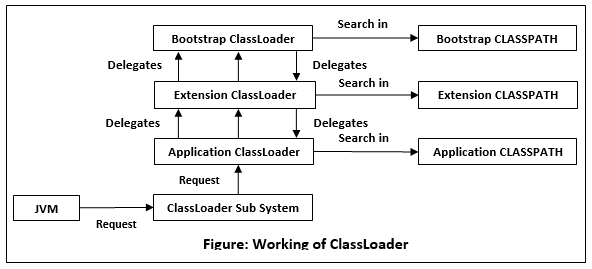


Bootstrap Classloader : It loads the predefined java classes / api classes (or) standard jdk classes from rt.jar . It is the parent of all the class loaders . it does have any parent.

Extension Classloader : If the loading of the class request is unsuccessful . It loads classes from jre/lib/ext.dirs or from any other directories . generally extension class loader loads user defined classes. It delegates class loading request to its parent if the class loading is unsuccessful. It is a child of Bootstrap class loader. And parent of system class loader (or) application class loader.

Application Classloader : loads classes from environment class variables . it can be set while invoking the program using classpath command line arguments .

Note : In simple terms bootstrap class loader loads predefined classes and extension, application class loader loads user defined classes.

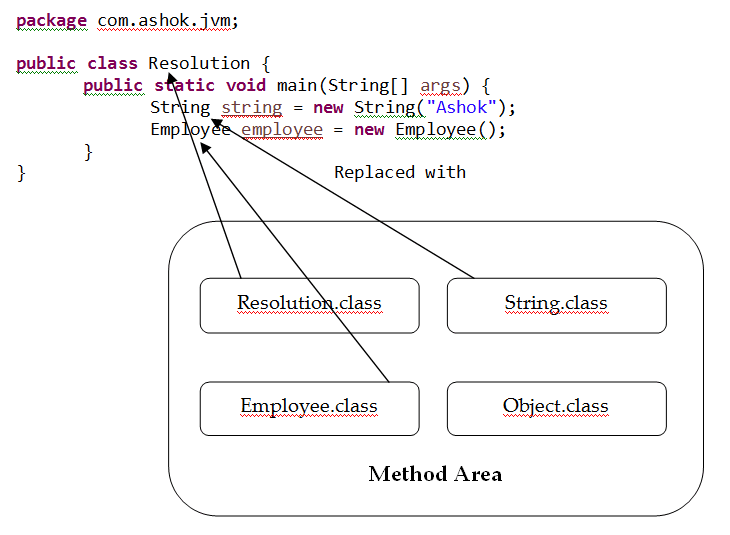


Suppose we have an application specific class demo.java . class loader subsystem request application class loader to load the class. It delegates to parent extension class loader , further it delegates to bootstrap class loader . bootstrap class loader loads the classes from rt.jar / loads predefined classes in program . if bootstrap doesn’t find the class it delegates to extension class loader which searches for jre/lib/ext and tries to locate the class there . if the extension class loader does not load it . the application class loader loads it from CLASSPATH in java .

Linking

The linking process consists of 3 steps they are

1. Verification : Here Bytecode verifier checks the byte code to ensure that .class file generated is valid or not . if not an error is thrown and the linking process will halt. (used to avoid malware and other viruses )
2. Preparation : memory to static variables is allocated and they are initialized with default values
3. Resolution : It is a process of replacing all the symbolic references used in our class with original direct references from method area



For the above class the class loader subsystem loads Resolution.class , String.class , Employee.class and Object class . in java for all the classes Object.class is parent of all the classes . for every subclass it parent class must be loaded . so Object .class is also loaded . The names of these classes are stored in constant pool of resolution class

In resolution phase these names are replaced with actual references from method area .

Initialization :

In this phase static variables are assigned with original values and static blocks are executed.

Runtime data areas :

Method area / class area : jvm will give Bytecode to class loader . The class loader will load the class and keep it in method area .The entire class data is present in the method area . class code , variable code ( static variable ,runtime constant ) , method code ( function inside class ) and constructor of a program .

Heap area :

Heap area is created at a time of jvm startup , objects and corresponding instances will be stored in heap area , array is an object so it is also stored in heap .

Note : static variables and methods are stored in heap , before java 8 they are stored in method area or class area .

Stack area :

What ever code (static block (or) static method (or) method) needs to be executed is brought to stack , frame is created and that code is brought to it and a thread executes it.

Program counter registers :

Jvm supports multiple threads at the same time . each thread has its own pc registers to hold the address of the currently executing instruction . once the execution is completed the pc register is updated with next instruction address .

Native Method stack :

Jvm contains stack that supports native methods (methods linked to native library) . These methods are written languages other than java , such as c and c++ . For every new thread a separate native stack method is created.

Execution Engine

Interpreter : jvm with the help of interpreter will convert all the code to 0’s and 1’s and then executes line by line .

Just in time compiler : it is not used all the time . if some method is called multiple times then , to avoid the conversion of same method to machine in multiple times just in time compiler is used.

Components of jit compiler :

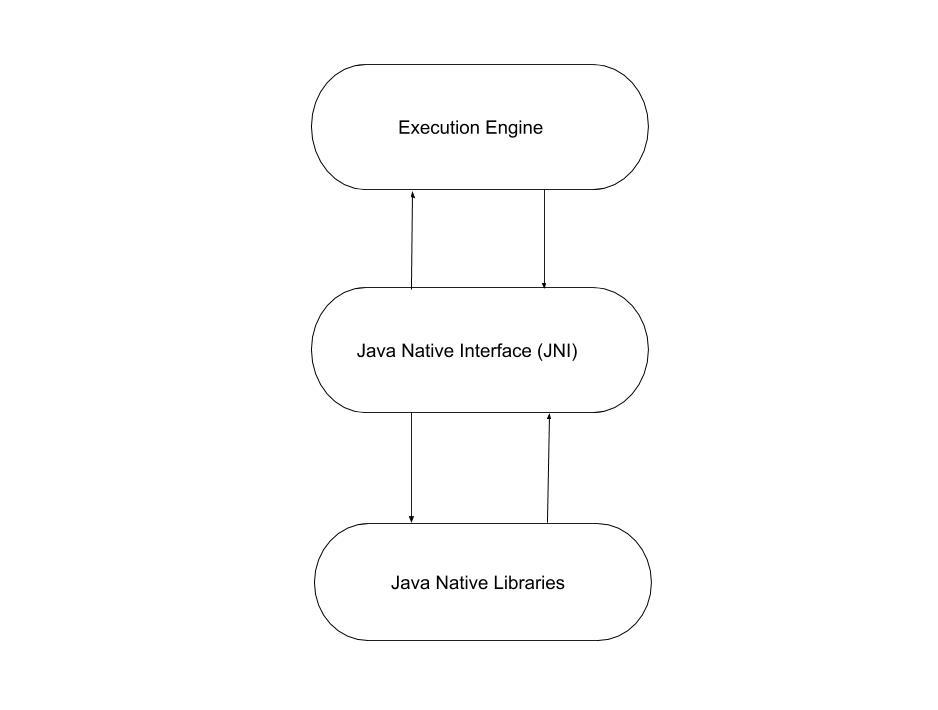
1. Intermediate code generator : generates intermediate code
2. Code optimizer : optimizes intermediate code for better performance
3. Target code generator : converts intermediate code to native machine code .

Profiler : tool of jit compiler used to find hotspots ( used to find repeatedly used methods )

Garbage collector : Removes unreferenced objects from the heap.

Java Native Interface (JNI) :

It acts as a bridge for java method calls and native libraries



// go through the jvm architecture and now check the above program Eg: Order\_Of\_Execution

Program internal working :

1. Program is compiled and byte code is generated .
2. Jvm takes the bytecode(.class file) gives it to class loader sub subsystem .
3. In the linking phase memory for static variables is allocated in the heap and default values are given .
4. And in the initialization phase static variables are assigned with original values in the heap and static blocks are executed .
5. Static block is brought into stack area and it is executed , every thing you want to execute needs to be stack only . even the static variables are assigned with original values when they are brought and executed in stack .
6. After that static method is (main method) brought to stack area and executed .

Note: Every that needs to be executed is brought to stack area and executed .

1. Now object is created memory for instance variables is allocated with default values in heap , java block is executed and constructor is executed.
2. When object is created actual internal working is , instance variables memory is allocated (order obj = new order(); new keyword creates object and order() constructor is executed ) and then constructor is executed . java block is treated as body for constructor , so first java block is executed and then constructor actual body is executed.

Note : static variables and static blocks are executed during class loading , main method is executed by jvm by default

Note : if static method and main method is in same class you can call static method directly with method\_name , class\_name.method\_name , with obj\_ref.method\_name also

Eg: Static\_Methods

If static method and main method is in different class you should call static method with class\_name.method , obj\_ref.method\_name (here object reference should be of class type in which static method is present)

Eg: Static\_Method\_1

Static variables , satic block , static method ( only main method ) is executed automatically

Eg: Automatic\_Execution

// go through the code

Static method (user defined ) , constructor , normal method is not executed automatically , it is executed if and only if we call them .

Eg: Manual\_Execution

// go through the code

Note: we can call method inside a method , but cannot create new method inside a method.

Note : non static variables (instance variables) cannot be used in static block and static method , because memory for instance variables is allocated only when object is created .

But static blocks , and static methods are executed ( at class loading) even before memory of instance variables is allocated . so instance variables are not allowed in static blocks and static methods.

Note : you can use static variables in all the static blocks , static methods , java block , constructor, non static method . since their memory is allocated at first (during class loading itself )