Chapter 12

Static Members
& their execution *control flow*

- In this chapter, You will learn
  - Need of static variable, static block, static method
  - Storing, modifying and using data common for all objects
  - Executing logic commonly for all objects
  - Executing logic only once at the time of class loading
  - JVM activities at the time of class loading.
  - Static members execution flow

- By the end of this chapter- you will be in a position to tell right answer yourself without using Computer and Java Software.

Interview Questions

By the end of this chapter you answer all below interview questions

1. What members are called Static members?
2. Types of Static members?
   a. Static variables
   b. Static blocks
   c. Static methods
   d. Main method
3. When do all these members get memory location and by whom?
4. Static Variable
   a. When a variable can be called as Static variable?
   b. Does static variable executed by JVM by default?
   c. Can developer execute static variable?
   d. How many static variables can we define in a class?
   e. What is the order of execution of all static variables?
   f. When, where, how and by whom memory location is provided?
   g. What is the lifetime and scope of static variable?
   h. Can we declare local variables or parameters as static?
   i. JVM Architecture with static variables
   j. Duplicate Variables
   k. Can we create a local variable with parameter name?
   l. How Static variable can be differentiated from local variable when both have same name? Local preference/Shadowing.
   m. What is the order of execution of static variables and main method?
   n. How JVM execute the static variable before main method even if it is defined after main method?
   o. Identification and execution phases
5. Static method
   a. When a method is called as static method?
   b. Does JVM execute static methods automatically?
   c. Can developer execute static methods?
   d. When static methods are executed, what is the order of execution?
   e. Where static method's logic is stored and where it is executed?
   f. Variable initialization with its own name or parameter
   g. Initializing static variable with parameter
   h. If we modify static variable in one class is that modification affected to all classes of the project in side that JVM?
   i. Modularity and advantages of modularity

Static Members and their control flow

What members are called static members?
The Class level members which have static keyword in their definition are called static members.

Types of Static Members
Java supports four types of static members
1. Static Variables.
2. Static Blocks
3. Static Methods.
4. Main Method.

When do all these members get memory location and by whom?
All static members are identified and get memory location at the time of class loading by default by JVM in method area.

Static Variable
A class level variable which has static keyword in its creation statement is called static variable.

Does JVM execute static variables by default?
Yes, it executes static variables - means provides memory location- at the time of class loading.

How many static variables can we define in a class?
Multiple static variables we can define, but every variable name should be different

What is the order of execution of all static variables?
In the order they are defined from top to bottom.

When, where, how and by whom memory location is provided to static variable?
When class is loaded JVM provides *individual single copy* of memory location to each static variable in method area only once in a class life time.

What is the lifetime and scope of static variable?
Static variable get life as soon as class is loaded into JVM and is available till class is removed from JVM (or) JVM is shutdown. Its scope is class scope means it is accessible throughout the class directly by its name, and outside class by its class name provided it is non-private.
For example

```
class Example{
        static int a = 10;
        static void m1(){
                System.out.println( a );
        }
}
```

```
class Sample{

        static void m2(){
                System.out.println(Example.a);
        }
}
```

Can we declare local variables or parameters as static?

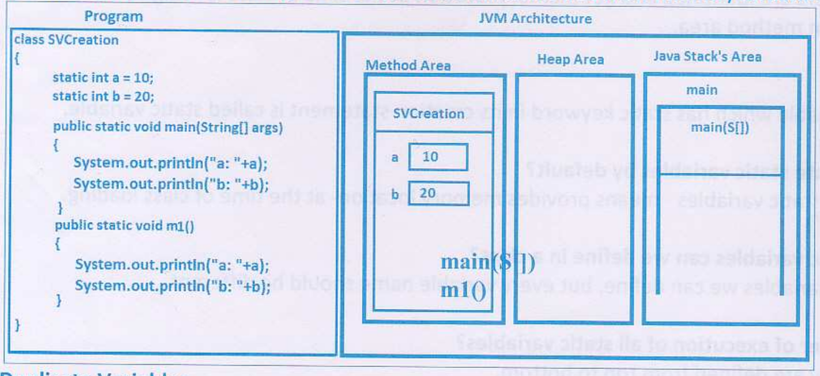
No, it is not possible. It leads to CE: "*illegal start of expression*". Static keyword is not allowed inside a block "{ }", because as being a static variable it must get memory at the time of class loading which is not possible to provide memory to local variable at the time of class loading.

For Example:

```
class Example{
        static int a = 10;
        public static void main(String[] args){
                static int b = 20; ╳ CE: illegal start of expression
        }
}
```

JVM Architecture with Static variables

Below diagram shows creating a class with static variables and their memory structure.



```
class SVCreation
{
        static int a = 10;
        static int b = 20;
        public static void main(String[] args)
        {
            System.out.println("a: "+a);
            System.out.println("b: "+b);
        }
        public static void m1()
        {
            System.out.println("a: "+a);
            System.out.println("b: "+b);
        }
}
```

Duplicate Variables

A variable that is created with existed variable name in the *same scope* is called duplicated variable. It leads to compile time error "**variable is already defined**". Even if we change data type or modifier or its assigned value we cannot create another variable with same name.

But it is possible to *create multiple variables* with *same name* in *different scopes*, for example the variable created in class scope can be created in method scope. *Check below program*

```
class Example
{
        static int a = 20;
        //int a = 10; //CE: a is already defined in Example

        public static void main(String[] args)
        {
                //it is allowed to define "a" variable in this method
                int a = 20;

                //creating local variable
                int p = 10;
                //double p = 30; //CE: a is already defined in Example
        }
        static void m1()
        {
                int p = 30;
        }
}
```

Q) Can we create local variable with parameter name?

A) No, because both parameter and local variables are created in the same method scope, so local variable is considered as duplicate variable

```
class Example{
        static void m1(int a){
                int a = 20; ╳ CE: a is already defined
                int b = 30; ✓
        }
}
```

Shadowing
Creating a local variable *or* parameter with same static or non-static variable name is called shadowing. It means local variable is a shadow of class level variable.

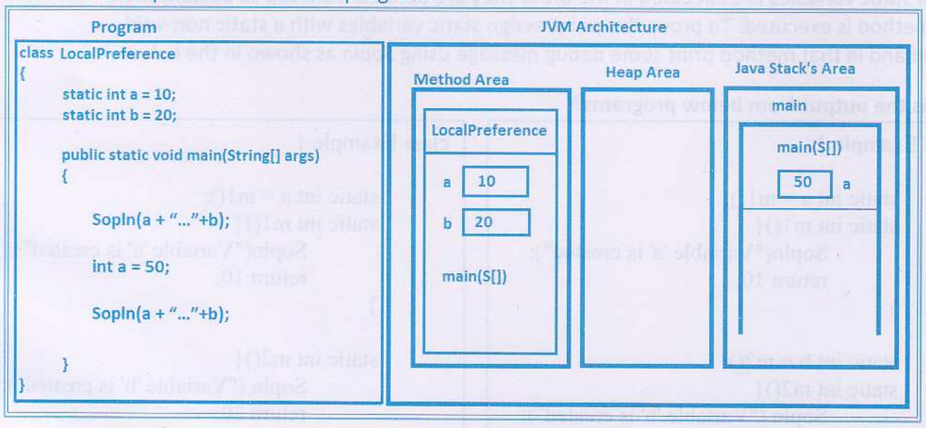
In this case when you access a variable in the method you will get local variable's value, but not from class level variable.

Local preference
How compiler and JVM search for variable definition?
When we call a variable, compiler and JVM search for its creation statement in the current method. If it is not found in the current method, compiler and JVM next search for its creation statement at class level. If that variable creation statement is not found at class level also, compiler throws CE: **cannot find symbol**.

This phenomenon is called local preference.

Hence, if a variable is created in both method and also in class with same name, compiler and JVM access that variable from method, because always local variable has first priority.

What is the output from the below program?



Program

```
class LocalPreference
{
        static int a = 10;
        static int b = 20;

        public static void main(String[] args)
        {

            Sopln(a + "..."+b);
            int a = 50;

            Sopln(a + "..."+b);

        }
}
```

How static variable can be differenced from local variable or parameter when both have same name?
We should use ***class name*** to differentiate static variable from local variable.

As you observed in the above diagram static variable get memory location with respect to class, and local variables get memory location with respect to method. So to differentiate static variables from local variable we should use class name, as shown in the below diagram.

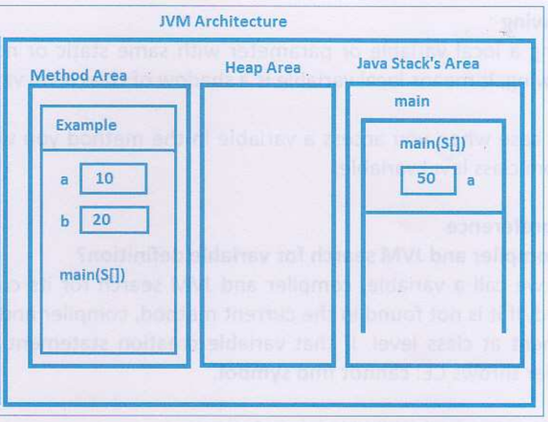
Program

```
class Example
{
        static int a = 10;
        static int b = 20;
        public static void main(String[] args)
        {
            System.out.println(a);
            System.out.println(b);

            int a = 50;

            System.out.println(a);

            System.out.println(Example.a);
            System.out.println(b);
            System.out.println(Example.b);
        }
}
```



Note: We can access static variables with class name even though there is no local variable by static variable name. But it is optional.

Q) What is the order of execution of static variables and main method?
First all static variables are executed in the order they are defined from top to bottom then main method is executed. To prove this point assign static variables with a static non-void method and in that method print some debug message using *Sopln* as shown in the below

What is the output from below programs?

```
class Example {

        static int a = m1();
        static int m1(){
                Sopln("Variable 'a' is created");
                return 10;
        }

        static int b = m2();
        static int m2(){
                Sopln ("Variable 'b' is created");
                return 20;
        }

        public static void main(String[] args) {

                System.out.println("main");
                System.out.println("a: "+ a);
                System.out.println("b: "+ b);
        }
}
```

```
class Example {

        static int a = m1();
        static int m1(){
                Sopln("Variable 'a' is created");
                return 10;
        }

        static int m2(){
                Sopln ("Variable 'b' is created");
                return 20;
        }

        public static void main(String[] args) {

                System.out.println("main");
                System.out.println("a: "+ a);
                System.out.println("b: "+ b);
        }
        static int b = m2();
}
```

In the above program, how JVM can find variable *b* that is defined after main method?
Because of *Identification phase*, actually JVM does not execute class members directly. It executes class members in two phases

They are:

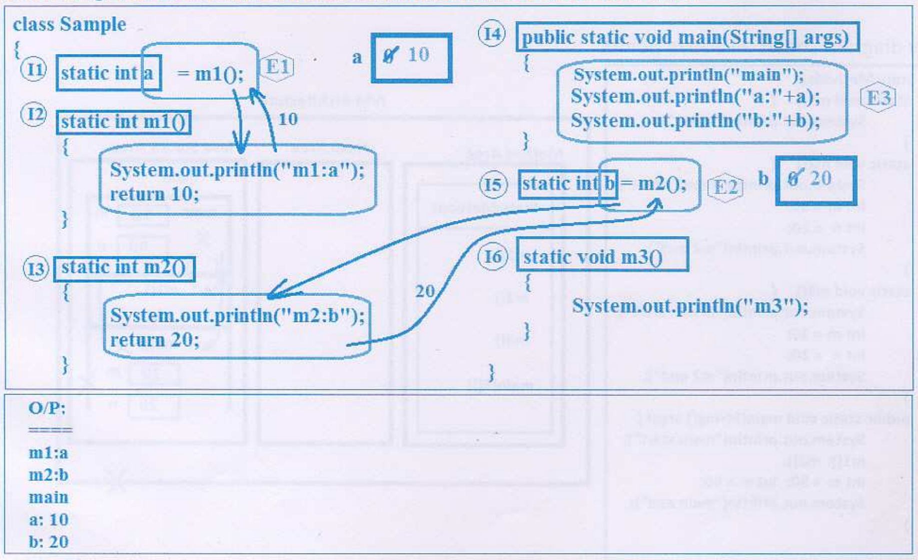
1. ***Identification phase***
2. ***Execution phase***

- First JVM identifies complete class static members at the time of class loading from top to bottom. After identification then it starts the static member's execution according to their priority from top to bottom.

- Identifying a variable means,
    - Creating its memory with default value based on its datatype
- Identifying a method means,
    - Remembering its prototype.

- Executing a variable means,
    - Storing its assigned value if any.
- Executing a method means,
    - Executing its logic if it is called.

Below diagram shows static variable and main method execution order



```
class Sample
{
  static int a = m1();
  static int m1()
  {
      System.out.println("m1:a");
      return 10;
  }
  static int m2()
  {
      System.out.println("m2:b");
      return 20;
  }
                    public static void main(String[] args)
                    {
                        System.out.println("main");
                        System.out.println("a:"+a);
                        System.out.println("b:"+b);
                    }
                    static int b = m2();
                    static void m3()
                    {
                        System.out.println("m3");
                    }
}
```

```
O/P:
====
m1:a
m2:b
main
a: 10
b: 20
```

Note: m3() method is identified but it is not executed as it is not called.

Static Method

A method which has static keyword in its definition is called static method.
Ex:

```
static void m1() {
  System.out.println("m1");
}
```

Does JVM execute static methods by default like static variables?
No, JVM does not execute static methods by itself. They are executed only if they are called explicitly by developer either *from main method*, or *from static variable* as its assignment statement or *from static block*.

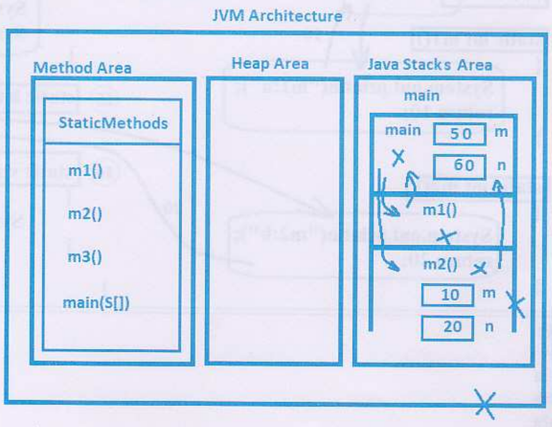
What is the order of execution of static methods?
In the order they are called, not in the order they are defined.

Where static methods logic is stored and where they are executed?
All static methods' logic is stored in method area and that logic executed in java stacks area in main thread by creating separate stack frame.

JVM architecture with Static method execution flow
When a static method is called from main method, JVM creates stack frame in main thread, loads that method complete logic and executes line by line of that method logic. The stack frame is destroyed immediately after method execution is completed.

Below diagram shows all above points

```
class StaticMethods{
        static void m1()    {
                System.out.println("In m1");
        }
        static void m2()    {
                System.out.println("In m2, start");
                int m  = 10;
                int n   = 20;
                System.out.println("m2 end");
        }
        static void m3()    {
                System.out.println("In m3, start");
                int m  = 10;
                int n   = 20;
                System.out.println("m2 end");
        }
        public static void main(String[] args) {
                System.out.println("main start");
                m1(); m2();
                int m  = 50;  int n   = 60;
                System.out.println("main end");
        }
}
```



Variable initialization with same variable

We can initialize a variable with same variable name, this assignment is valid. In this case the variable value is replaced with same value.

Ex:
```
int a = 10;
a = a ; ✓
```

Assigning a static variable with a local variable or parameter

If a local variable or parameter is created with static variable name we must use class name in assigning static variable with that local variable or parameter. Else the modification is stored in that local variable or parameter not in static variable.

Check below programs

In the below program is static variable value is modified?

```
class Example
{
        static int a = 10;
        public static void main(String[] args)
        {
                int a = 20;

                a = a ;

                System.out.println(a);
                System.out.println(Example.a);
        }
}
```

To intialize static varaible with same name local variable we must refer left hand side variable with class name explicitly as shown below

```
class Example
{
        static int a = 10;
        public static void main(String[] args)
        {
                int a = 50;

                Example.a = a;

                System.out.println(a);
                System.out.println(Example.a);
        }
}
```

Local preference with parameters

Since parameters are also treated as local variables, we must refer static variable with class name if parameter is declared with same static variable name.

Find out, is static variable modified in below programs

```
class Example{
        static int a;

        static void m1(int x){
                a = x;
                System.out.println(a);
        }

        public static void main(String[] args){
                System.out.println(a);
                m1(50);
                System.out.println(a);
        }
}
```

```
class Example{
        static int a;

        static void m1(int a){
                a = a;
                System.out.println(a);
        }

        public static void main(String[] args){
                System.out.println(a);
                m1(50);
                System.out.println(a);
        }
}
```

```
class Example{
        static int a;
        static void m1(int a){
                Example.a = a;
                System.out.println(a);
        }
        public static void main(String[] args){
                System.out.println(a);
                m1(50);
                System.out.println(a);
        }
}
```

```
class Example{
        static int a;
        static void m1(int a){
                a = Example.a;
                System.out.println(a);
        }
        public static void main(String[] args){
                System.out.println(a);
                m1(50);
                System.out.println(a);
        }
}
```

What is the output from the below program?

```
class Example
{
        static int a;
        static int b;

        static void m1(){
                a = 10;
                b = 20;
        }

        static void m2(int x , int y){
                a = x;
                b = y;
        }

        static void m3(int a , int b){
                a = a;
                b = b;
        }

        static void m4(int a , int b){
                Example.a = a;
                Example.b = b;
        }

        public static void main(String[] args)
        {
                System.out.println(a+"..."+b);

                System.out.println();

                m1();
                System.out.println(a+"..."+b);

                System.out.println();

                m2(30, 40);
                System.out.println(a+"..."+b);

                System.out.println();

                m3(50, 60);
                System.out.println(a+"..."+b);

                System.out.println();

                m4(70, 80);
                System.out.println(a+"..."+b);

        }
}
```

Modularity

Modularity means dividing big task into small pieces. In Java we can achieve modularity with methods and also with classes.

In the below program we have written all arithmetic operations in single method, so it is not possible to execute exactly one of the arithmetic operations, all 4 operations will be executed, hence it is wrong design.

Dividing four tasks into four methods to execute each task individually is called modularity.

```
class AO                No Modularity      ✗
{
        static void ao(int a , int b)
        {
                //addition
                System.out.println(a + b);

                //substraction
                System.out.println(a - b);

                //multiplication
                System.out.println(a * b);

                //division
                System.out.println(a / b);
        }
}
```

```
class AO                With Modularity    ✓
{
    static void add(int a , int b)
    {
        //addition
        System.out.println(a + b);
    }
    static void sub(int a , int b)
    {
        //substraction
        System.out.println(a - b);
    }
    static void mul(int a , int b)
    {
        //multiplication
        System.out.println(a * b);
    }
    static void div(int a , int b)
    {
        //division
        System.out.println(a / b);
    }
}
```

Advantages in modularity

We can achieve *centralized code* change and *code reusability*.

For example if we have some logic to be executed in multiple methods, placing that logic in all methods is *not feasible* because for every small change in the logic we must do this modification in all places. This considered as *code redundancy* and there is no centralized code change.

So to solve above two problems that logic should not be placed in every method rather define this logic in a separate method and call that method from every method where ever we need it. So that if we need to do any changes further those changes we required to do this change only in the common method as shown below.

```
class Example                                                                    static void add()
{                                                                                {
    static void m1()                                                                 Sopln(a + b);
    {                                                                                Sopln("Result: "+(a + b))
        ---------------        add();                                            }
        Sopln(10 + 20); => Sopln("The result after adding  10  and 20 is "+ 30 );
        ---------------                                                       }
    }
    static void m2()
    {
        ---------------       add();
        Sopln(10 + 20); => Sopln("The result after adding  10  and 20 is "+ 30 );
        ---------------
    }
    static void m3()
    {
        ---------------       add();
        Sopln(10 + 20); => Sopln("The result after adding  10  and 20 is "+ 30 );
        ---------------
    }
```

Q) What is the design change we should do in the above program to use add method from multiple classes?

We should define the above add method in another separate class. Then we should access / reuse this method with that class name or with that class object. This approach is called developing *modularity at class level.*

Below program shows above said code change.

This Addition class is called ***component***, a *reusable class.*

```
//Addition.java
public class Addition {
        public static void add(int a, int b){
                System.out.println("The addition of "+ a + " and "+ b + " is: "+ (a + b) );
        }
}
```

Below classes are reusing this addition logic

```
//A.java
class A {
        public static void main(String[] args) {
                Addition.add(10, 20);
        }
}
```

```
//B.java
class B {
        public static void main(String[] args) {
                Addition.add(50, 60);
        }
}
```

Static Blocks
Static block is a class level nameless block that contains only static keyword in its prototype.

Syntax:

```
class Example
{
        static
        {
                -------------   }  Any Java Legal
                -------------   }  statement is allowed
                -------------   }  except return and
                -------------   }  throw statement
        }
}
```

Need of static block
It is used to execute logic only at the time of class loading.
Logic like,

- Initializing static variables
- Registering native libraries
- To know classes loading order, etc...

Different ways to execute logic at the time of class loading
Actually there are two ways to execute logic at the time of class loading.

1. **Using static variable.**

```
class Example{
        static int a = m1();

        static int m1(){
                System.out.println("SV : a");

                return 10;
        }
```

```
        public static void main(String[] args)
        {
                System.out.println("main");
                m1();
        }
}
```

2 Drawbacks in this approach

a. m1() method logic can also be executed after class loading, because it can be called from main() method, that leads to execution of m1( ) after class loading.
b. the method must be a non-void method.

2. **Using static block**
It is the Solution for the above problem - write that logic in static block to execute that logic only at the time of class loading

Who will execute static block when and where?
Static blocks are executed automatically by JVM at the time of class loading in the order they defined from top to bottom by creating separate stack frame in main thread in Java stacks area.

Then what is order of execution of SB and Main method?
Static block is always executed before main method.

What is the output from the below program?

```
class Example
{
        static
        {
                System.out.println("SB");

        }
        public static void main(String[] args)
        {
                System.out.println("main");
        }
}
```

```
class Example
{
        public static void main(String[] args)
        {
                System.out.println("main");
        }
        static
        {
                System.out.println("SB");

        }
}
```

How many static blocks can be defined in a class?
We can define *more than one static block* in a class.

Can we nest static blocks *means* Can we write a static block in another static block?
No, static keyword is not allowed inside blocks or methods

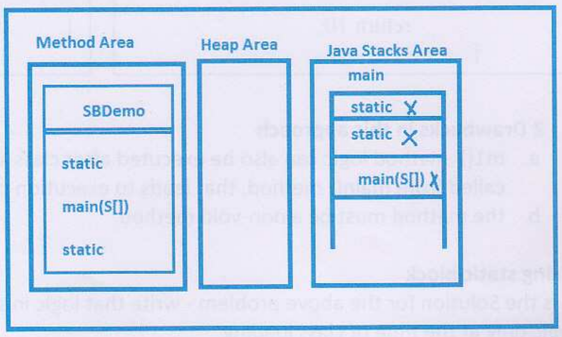
What is the order of execution of all static blocks?
All static blocks are executed in the order they defined from top to bottom.

Check below program, what is the output?

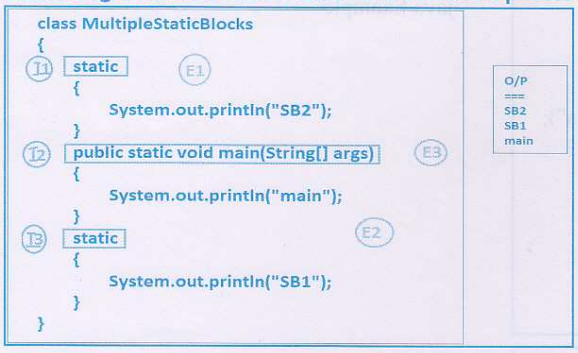
```
class SBDemo
{
        static
        {
                System.out.println("SB2");
        }

        public static void main(String[] args)
        {
                System.out.println("main");
        }

        static
        {
                System.out.println("SB1");
        }
}
```



Below diagram shows Identification and execution phases of SB and main



```
class MultipleStaticBlocks
{
    static
    {
        System.out.println("SB2");
    }
    public static void main(String[] args)
    {
        System.out.println("main");
    }
    static
    {
        System.out.println("SB1");
    }
}
```

```
O/P
===
SB2
SB1
main
```

Can we execute class logic without main method?
Yes, we can execute by using either static variables or static blocks, but from Java 7 main method is mandatory to execute a class.

Class execution procedure with static block and main method?
When a class is loaded, first JVM executes static variables and static block then after it searches from main method, if main method presents it execute main method else it terminates program execution with exception *java.lang.NoSuchMethodError*.

But from Java 7 onwards class execution starts only if it has main method, else program execution is terminated without executing static variables or static blocks by throwing an ***Error: main method is not available***.

What is the output from the below program?

```
class Example{
}
```

```
>java Example
```

```
class Example{
        static {
                System.out.println("SB");
        }
}
```

```
>java Example
```

```
class Example{
        static int a = m1();

        static int m1(){
                System.out.println("SV : a");
                return 10;
        }
}
```

```
>java Example
```

```
class Example{
        static int a = m1();

        static int m1(){
                System.out.println("SV : a");
                return 10;
        }

        static {
                System.out.println("SB");
        }
}
```

```
>java Example
```

```
class Example{
        static {
                System.out.println("SB");
        }

        static int a = m1();

        static int m1(){
                System.out.println("SV : a");
                return 10;
        }
        public static void main(String[] args){
                System.out.println("main");
        }
}
```

```
>java Example
```

Q) Can we execute main method at the time of class loading?

Yesssssssss, it is possible.

Call it from static block it is also executed at the time of class loading.

What is the output from the below program?

```
class Example{
        static{
                System.out.println("SB start");
                main(new String[0]);
                System.out.println("SB end");
                System.out.println();
        }
        public static void main(String[] args){
                System.out.println("main");
        }
};
```

O/P

```
SB start
main
SB end

main
```

Main method

Why main method is public?
Because it must be called by JVM from outside of our package.

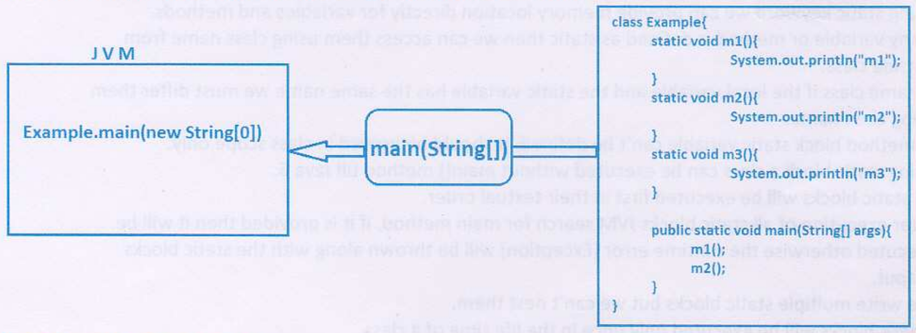
Why main method has static keyword in its definition?
main() is the initial point of class logic execution. Hence it should be identified at the time of class loading. Due to this reason it contains static keyword in its prototype. Of course it must be executed without object create.

When user defined methods should contain static keyword?
To execute the method logic without object creation, method should be defined as static method. It means if method, logic internally not using object data – non-static variables – then it is recommended to declare that method as static method.

Why JVM executes only main method why not user defined static methods? Or why main method is the initial point of class logic execution?
Main method calling statement is available in JVM software, but our static methods calling statement is not available in JVM software.

And more over JVM doesn't know your methods execution order and also it doesn't know what are the methods should be executed from your class and when.

It is the developer responsibility to inform methods and their order of execution to JVM. For this purpose there should be a common method that is known to both JVM and to developer, and that method prototype should be given by the SUN. That method is main method.



```
Example.main(new String[0])
```

```
class Example{
        static void m1(){
                        System.out.println("m1");
        }
        static void m2(){
                        System.out.println("m2");
        }
        static void m3(){
                        System.out.println("m3");
        }

        public static void main(String[] args){
                m1();
                m2();
        }
}
```

Definition of main method
main method is the mediator method between java developer and JVM to inform methods execution order.

Why main() method return type is void?
Because if we return a value, it is sent to JVM which is useless. Hence main() method return type is void.

Why its name is main?
As per the coding standards the method name should convey the operations it is doing. Since this method is intended to execute main logic of the program, that is business logic, it is named as main.

Why main() method has parameter String[]?
To read command line arguments (values) into Java application from keyboard

For example

```
>java sample 10 20
```

Why main method parameter type is String?
Because the value passing from keyboard is sent into Java application as String type value.

Why is it String[], why not it is just String?
To read more than one value

Why main() method parameter name is args?
Again it is also because of coding standard, as we are reading arguments from keyboard the name of the parameter is "args". We can change its name, but it is not recommended.

Can we call main method explicitly?
Yes it is possible, it is also a method.
Note: main( ) method should not be called from its own block or from a method that is calling from main(), it leads to exception java.lang.StackOverflowError.

Conclusions

- Using static keyword we can provide memory location directly for variables and methods.
- If any variable or method is defined as static then we can access them using class name from outside class.
- In same class if the local variable and the static variable has the same name we must differ them using class name.
- In method block static variable can't be defined, It should be proved in class scope only.
- Using static block a class can be executed without main() method till Java 6.
- All static blocks will be executed first in their textual order.
- After execution of all static blocks JVM search for main method, if it is provided then it will be executed otherwise the runtime error (Exception) will be thrown along with the static blocks output.
- We write multiple static blocks but we can't nest them.
- These blocks will be executed only once in the life time of a class