**JDK** stands for **Java Development Kit** or sometimes it is also referred as **Java Standard Edition Development Kit**. JDK is a development environment to develop wide range of applications such as desktop applications, web applications or mobile applications using Java programming language.

**JRE**– Java Runtime Environment, is also part of JDK. JRE provides the minimum runtime requirements for executing a java application. It consists of Java Virtual Machine(JVM) executables, core classes and some supporting files.

**main** is a method name. Method name must be in lowercase. **main** method is a special method because execution of a java program starts from **main** method. This method takes one argument of type String array. Remember **main** is not a keyword.

**String** is a final class from *java.lang* package.

**System** is also a final class from *java.lang* package. **out** is a static member of **System** class of type *PrintStream*. **println** is a method of *PrintStream* class.

**Post Increment Operator:**

**package** incrementConcept;

**public** **class** PostIncrement {

**public** **static** **void** main(String[] args) {

**int** i=0;

System.***out***.println(i++);

}

}

If you are thinking that output will be 1 then you are wrong. If you run this program output will be 0 but not 1. Because the operator used is post increment operator.

**According to definition of post increment operator, first, value of the variable is used and then incremented** i.e. first, value of i(0) is printed and then i is incremented to 1. So, here usage value(used value) of i is 0 and storage value(value stored in the memory) is 1.

Another Program:

**package** incrementConcept;

**public** **class** PostIncrement {

**public** **static** **void** main(String[] args) {

**int** i;

**for**(i=0;i<10;i++) {

System.***out***.println(i++);

}

}

}

Output:

0

2

4

6

8

**Pre Increment Operator:**

**package** incrementConcept;

**public** **class** PreIncrement {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

**int** i=0;

System.***out***.println(++i);

}

}

Here, output will be 1, The operator used is pre increment operator**. When you use pre increment operator, first, value is incremented and then used**. In the above program, first, value of i is incremented to 1 then it is used. So, usage value and storage value both are same.

**Another Example for Pre increment Operator:**

**package** incrementConcept;

**public** **class** PreIncrement {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

**int** i=0;

**for**(i=0;i<10;++i)

{

System.***out***.println(++i);

}

}

}

Output:

1

3

5

7

9

**Post Decrement Operator:**

**package** incrementConcept;

**public** **class** PostDecrement {

**public** **static** **void** main(String[] args) {

**int** i = 0;

System.***out***.println(i--);

}

}

Here, the operator used is post decrement operator. It operates in the same manner as post increment operator but here the value is decremented. The output of this program will be 0 not -1, because first, value is used and then decremented. So, here usage value is 0 and storage value is -1.

Pre increment Operator:

**package** incrementConcept;

**public** **class** PreDecrement {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

**int** i = 0;

System.***out***.println(--i);

}

}

Here, the operator used is pre decrement operator. It is also operates in the same manner as pre increment operator but here the value is decremented. If you run this program, output will be -1. That means first, value is decremented and then used. So, usage value is -1 and storage value is also -1.

Can we use local variables before they are initializes?

**package** variables;

**public** **class** GlobalAndLocalVariables {

**static** **int** *globalVariable*;

**static** **void** methodOne() {

**int** localToMethodOne=0;

System.***out***.println(*globalVariable*);

System.***out***.println(localToMethodOne);

}

**public** **static** **void** main(String[] args) {

**int** localToMain=0;

System.***out***.println(localToMain);

System.***out***.println(*globalVariable*);

}

}

In the above program, ‘globalVariable’ declared in the Line 3 is a global variable. **Global Variable** has to be declared anywhere in the class body but not inside any method or block. If a variable is declared as global, it can be used anywhere in the class. For example, see the above program, ‘globalVariable’ declared in the Line 3 is used inside the methodOne() (Line 6) and also inside the main() method (Line 12). So, Global variables are available for all methods and blocks of that class.

If the variable is declared inside a method or block, it is called **local variable**. Local variable is available only to method or block in which it is declared. For example, in the above program, ‘localToMethodOne’ is a local variable of methodOne() and it is accessible only in methodOne()and not available outside the methodOne(). If you use outside methodOne(), you will get compile time error. Variable ‘localToMain’ declared is also local variable. It is available only inside main() method.

**Default Values of Global Variables:**

If you don’t initialize global variables, they take default values of declared type. For example, If global variable is int type and it is not initialized explicitly, it will take default value of int type i.e 0. Below is the list of some data types and their default values.

|  |  |
| --- | --- |
| DataType | DefaultValue |
| Int | 0 |
| Boolean | false |
| Byte | 0 |
| Short | 0 |
| Long | 0 |
| Float | 0.0 |
| Double | 0.0 |
| All derived Data types | null |

**Important Note : If the local variables are not initialized explicitly, they don’t take default values. They remain uninitialized until you initialize them explicitly.**

Now, come to our question, **can we use local variables before they are initialized?**. Consider following program.

**package** variables;

**public** **class** LocalVariableBehaviour {

**public** **static** **void** main(String[] args) {

**int** i;

System.***out***.println(i);

i=10;

}

}

If you try to compile above program, you will get a **compile time error :** The local variable i may not have been initialized**.** Because, any variable, global or local, should have some value before they are used. If you don’t initialize global variables explicitly, they take default values. But, If you don’t initialize local variables explicitly, they don’t take default values. They remain uninitialized until you initialize them explicitly. Therefore, local variables will not be having any value until they are initialized explicitly. Therefore, when you use local variables before they are initialized, you get compile time error. That’s why we can’t use local variables before they are initialized.

In The above program, local variable i is used(Line 5) before it is initialized(Line 6).

To make the above program error free, put i=10 before System.out.println(i).

**SIB – Static Initialization Block, Static Variables And Static Methods**

Static variables, Static Initialization Block and Static Methods – these all are static components or static members of a class. These static members are stored inside the Class Memory. To access static members, you need not to create objects. Directly you can access them with class name.

Static Initialization Block is used to initialize only static variables. It is a block without a name. It contains set of statements enclosed within { }. The syntax of SIB looks like this,

1. static
2. {
3. //Set Of Statements
4. }

Consider the following program.

1. class StaticComponents
2. {
3. static **int** staticVariable;
4. static
5. {
6. System.out.println("StaticComponents SIB");
7. staticVariable = 10;
8. }
9. static void **staticMethod**()
10. {
11. System.out.println("From StaticMethod");
12. System.***out***.println("Printing staticVariable from Static Method: " + *staticVariable*);
13. }
14. }
15. public class MainClass
16. {
17. static
18. {
19. System.out.println("MainClass SIB");
20. }
21. public static void **main**(**String**[] args)
22. {
23. //Static Members directly accessed with Class Name
24. StaticComponents.staticVariable = 20;
25. StaticComponents.staticMethod();
26. }
27. }

Output:

Main class static initialization

static components SIB

From Static Method

Printing staticVariable from Static Method: 20

Let us discuss execution of above program step by step.

**Step 1:**

When you trigger >java MainClass, java command divides allocated memory into two parts – Stack and Heap. First, java command enters stack memory for execution. First, it checks whether **MainClass** is loaded into heap memory or not. If it is not loaded, loading operation of MainClass starts. Randomly some memory space is allocated to MainClass. It is called **Class memory**. All static members are loaded into this class memory. There is only one satic member in MainClass – main() method. It is loaded into class memory of MainClass.

**Step 2:**

After loading all static members, SIB – Static initialization Blocks are executed. Remember, **SIBs are not stored in the heap memory. They just come to stack, execute their tasks and leaves the memory**. So, after loading main() method, SIB of MainClass enters stack for execution. There is only one statement (Line 22) in SIB. it is executed. It prints “MainClass SIB” on console. After executing this statement, SIB leaves the stack memory.

**Step 3:**

Now, java command calls main() method for execution. main() method enters the stack. First statement (Line 28) is executed first. First, It checks whether class StaticComponents is loaded into memory. If it is not loaded, loading operation of StaticComponents takes place. Randomly, some memory is allocated to Class StaticComponents, then all static members of StaticComponents – ‘staticVariable’ and ‘staticMethod()’ are loaded into that class memory. ‘staticVariable’ is a global variable. So, first it is initialized with default value i.e 0.

**Step 4 :**

After loading all static members of StaticComponents, SIB blocks are executed. So, SIB of class StaticComponents enters the stack for execution. First Statement (Line 7) is executed. It prints “StaticComponents SIB” on the console. In the second statement, value 10 is assigned to ‘staticVariable’. There are no other statements left for execution, so it leaves stack memory.

**Step 5 :**

Now control comes back to main() method. The remaining part of first statement i.e value 20 is assigned to ‘staticVariable’ of class StaticComponents, is executed. In the second statement (Line 29), it calls staticMethod() of class StaticComponents for execution.

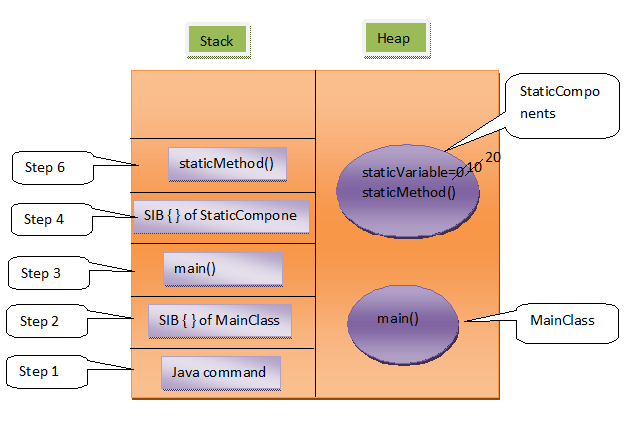
**Step 6:**

staticMethod() of StaticComponents enters stack for execution.  First statement (Line 13) is executed first. It prints “From staticMethod” on the console. In the second statement (Line 14), it prints the value of staticVariable i.e 20 on the console. There are no statements left. so, it leaves the stack.

**Step 7:**

Again, control comes back to main() method. There are no other statements left in main() method. so, it also leaves stack. java command also leaves the stack.

Diagramatic representation of memory allocation of above program looks like this.



**Non Static Members and their Memory Management in Java:**

**Key Points:**

Class : Class is the model/template/blueprint for the objects to be created of its type.

Object : It is an instance of a class. It is the real-time copy of class.

If you don’t understand with the definitions, read out this example. A class is like a blue print of a house. With this blueprint, you can build any number of houses. Each house build with this blueprint is an object or an instance of that blue print.

Non-Static variables and Non-Static methods are non-static components of a class. These are also called instance components of a class. Non-static components are stored inside the object memory. Each object will have their own copy of non-static components. But,  static components are common to all objects of that class.

**Facts about Non Static Members:**

* 1. **you can’t refer a non-static members through a class name. Because, non-static members are stored inside the object memory. You have to refer them through objects only.**
  2. **directly** **you can’t use non-static member inside a static method. Because, non-static members are stored inside the object memory. You have to create objects to use them. You have to refer them through objects only.**
  3. **All static members are loaded into the object memory.**
  4. **You can refer a static member of a class through object of that class like in Line 32. Whenever you refer a static member through a object, compiller replaces object name with its class name like a1.staticVariable is treated as A.staticVariable by the compiler.**
  5. **changes made to static components through one object is reflected in another object also. Because, the same copy of static components is available to all the objects of that class.**

1. class A
2. {
3. **int** nonStaticVariable;
4. static **int** staticVariable;
5. static void **staticMethod**()
6. {
7. System.out.println(staticVariable);
8. // System.out.println(nonStaticVariable);
9. }
10. void **nonStaticMethod**()
11. {
12. System.out.println(staticVariable);
13. System.out.println(nonStaticVariable);
14. }
15. }
16. class MainClass
17. {
18. public static void **main**(**String**[] args)
19. {
20. A.staticVariable = 10;
21. // A.nonStaticVariable = 10;
22. A.staticMethod();
23. // A.nonStaticMethod();
24. A a1 = **new** **A**();
25. A a2 = **new** **A**();
26. System.out.println(a1.nonStaticVariable);
27. System.out.println(a1.staticVariable);
28. a1.nonStaticMethod();
29. a1.staticMethod();
30. System.out.println(a2.staticVariable);
31. a1.staticVariable = 20;
32. System.out.println(a2.staticVariable);
33. }
34. }

**Output :**

10  
0  
10  
10  
0  
10  
10  
20

Let’s discuss memory allocation of above example step by step.

**Step 1 :**

When you trigger >java MainClass, java command divides allocated memory into two parts – stack and heap. First java command enters stack for execution. First it loads class **MainClass**into heap memory. Randomly some memory is allocated to MainClass. All static members are loaded into this class memory. There is only one static member in MainClass i.e main() method. It is loaded into class memory. After loading static members, SIBs are executed. But there is no SIBs in MainClass. So, directly java command calls main() method for execution.

**Step 2 :**

main() method enters stack for execution. First statement (Line 23) refers to class A. First it checks whether class A is loaded into heap memory or not. If it is not loaded, it loads class A into heap memory. Randomly some memory is allocated to class A. All static members of class A , ‘staticVariable’ and ‘staticMethod()’ , are loaded into this memory. ‘staticVariable’ is first initialized with default value 0. No SIBs in Class A. So, after loading static members, main() method assigns value 10 to ‘staticVariable’ of class A.

Second statement (Line 24) of main() method is commented. **Because, you can’t refer a non-static members through a class name. Because, non-static members are stored inside the object memory. You have to refer them through objects only.**

**Step 3 :**

In Line 25, it calls staticMethod() of class A. staticMethod() comes to stack for execution. First statement(Line 8) prints value of ‘staticVariable’ i. e 10 on the console.

Second statement(Line 9) is commented. Because, **directly** **you can’t use non-static member inside a static method. Because, non-static members are stored inside the object memory. You have to create objects to use them. You have to refer them through objects only.**

No statements left in staticMethod(). So, it leaves the stack memory.

**Step 4 :**

Control comes back to main() method. The next statement (Line 26) is also commented. **Because, You can’t refer non-static member through a class name.** In the next statement (Line 28), an object of class A type is created. Randomly, some memory is allocated to object. **All non-static members, ‘nonStaticVariable’ and ‘nonStaticMethod()’,  of class A are loaded into this object memory.** ‘nonStaticVariable’ is a global variable, so it is first initialized with default value 0. A reference variable of type class A  **‘a1’** is created in main() method. It points to this newly created object.

In the same manner, object ‘a2’ is also created (Line 29). In the next statement (Line 31), value of ‘nonStaticVariable’ of ‘a1’ i.e 0 is printed. In the next statement (Line 32), value of ‘staticVariable’ of class A i.e 10 is printed.

**You can refer a static member of a class through object of that class like in Line 32. Whenever you refer a static member through a object, compiller replaces object name with its class name like a1.staticVariable is treated as A.staticVariable by the compiler.**

In the next statement (Line 33), it calls ‘nonStaticMethod()’ of a1.

**Step 5 :**

‘nonStaticMethod()’ of a1 comes to the stack for execution. First statement (Line 14) prints value of  ‘staticVariable’ of class A i.e 10 on the console. Second statement (Line 15) prints the value of ‘nonStaticVariable’ of a1 i.e 0. There are no other statements left in ‘nonStaticMethod()’ , so it leaves the stack.

**Step 6 :**

Control comes back to Line 34 of main() method. It calls staticMethod() of class A. ‘staticMethod()’ enters the stack for execution. First statment (Line 8) prints value of  ‘staticVariable’  i.e 10 on the console. It leaves the memory after executing this statement.

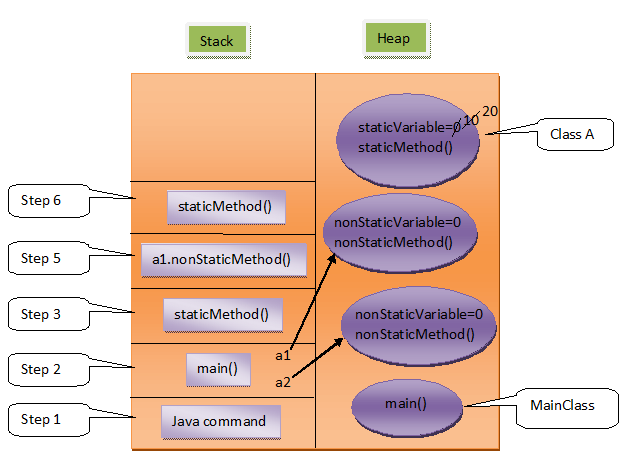
**Step 7 :**

Control comes back to the main() method. Line 36 prints value of ‘staticVariable’ i.e 10 on the console through object a2. In the next statement it changes value of ‘staticVariable’ to 20 through a1. In the next statement, again it prints the value of ‘staticVariable’ through a2. This time 20 is printed on the console.

**This means changes made to static components through one object is reflected in another object also. Because, the same copy of static components is available to all the objects of that class.**

As all statements are executed, first main() method then java command leaves the stack memory.

Diagramatic representation of memory allocation of above program looks like this,



**Constructors in Java**

**Rules of a Constructor:**

1) **Name of the constructor must be same as that of class name**. if you another name, it will give compile time error. If you give another name, it is neither a method because of no return type, nor constructor because name is different from class name.

2) **Constructors must not have a return type**. If you keep return type for the constructor, it will be treated as another method.But compiler gives a warning saying that this method has a constructor name. That means, it is legal to have method name same as constructor name or same as class name but it is not recommended.

3) **Every class should have at least one constructor**. If you don’t write constructor for your class, compiler will give default constructor. **Default constructor is always public and it has no arguments (No-Arg Constructor).**

4) **Constructor can be declared as private.** If you declare constructor as private, you can’t use it outside that class.

5) **One class can have more than one constructors. It is called Constructor Overloading.**Through constructor overloading, you can have multiple ways to create objects.

6) **Duplicate Constructors not allowed.** If you keep duplicate constructors, you will get compile time error.

7) **Multiple arguments of the constructors can’t have same name.** If the two arguments have the same name, you will get compile time error

**8) Only public, protected and private keywords are allowed before a constructor name.** If you keep any other keyword before a constructor name, it gives compile time error.

9**) First statement in a constructor must be either super() or this().** If you put any other statements you will get compile time error.If you don’t include these statements, by default compiler will keep super() calling statement. **super() – It is a calling statement to default constructor of super class. this()- it is a calling statement to constructor of the same class.**

10) Recursive constructor calling is not allowed.

11) No Cylic calling of constructors.

1)**Name of the constructor must be same as that of class name**. if you another name, it will give compile time error. If you give another name, it is neither a method because of no return type, nor constructor because name is different from class name.

1. class A
2. {
3. **A**()
4. {
5. // Constructor of Class A
6. }
7. **A1**()
8. {
9. // Compile time error, It is neither a constructor nor a method
10. }
11. }

2) **Constructors must not have a return type**. If you keep return type for the constructor, it will be treated as another method.But compiler gives a warning saying that this method has a constructor name. That means, it is legal to have method name same as constructor name or same as class name but it is not recommended.

1. class A
2. {
3. **A**()
4. {
5. // Constructor of Class A, not having any return type.
6. }
7. void **A**()
8. {
9. // constructor having a return type, It will be treated as method but with a warning.
10. }
11. }

3) Every class should have at least one constructor. If you don’t write constructor for your class, compiler will give default constructor. Default constructor is always public and it has no arguments (No-Arg Constructor).

1. class A
2. {
3. // No Constructors written
4. }

Compiler will treat the above code as,

1. class A
2. {
3. public **A**()
4. {
5. // Constructor provided by the compiler.
6. }

4) Constructor can be declared as private. If you declare constructor as private, you can’t use it outside that class.

1. class A
2. {
3. private **A**()
4. {
5. // Private Constructor
6. }
7. void **methodOne**()
8. {
9. //You can use private constructor inside the class
10. A a1 = **new** **A**();
11. }
12. }
13. class MainClass
14. {
15. public static void **main**(**String**[] args)
16. {
17. //You can't use private constructor ouside the class like this
18. // A a1 = new A();
19. }
20. }

5) One class can have more than one constructors. It is called **Constructor Overloading.**Through constructor overloading, you can have multiple ways to create objects.

1. class A
2. {
3. **A**()
4. {
5. // First Constructor
6. }
7. **A**(**int** i)
8. {
9. // Second Constructor
10. }
11. **A**(**int** i, **int** j)
12. {
13. // Third Constructor
14. }
15. }

you can create the objects to the above class in three ways like below,

1. class MainClass
2. {
3. public static void **main**(**String**[] args)
4. {
5. A a1 = **new** **A**(); //Using First Constructor
6. A a2 = **new** **A**(10); // Using Second Constructor
7. A a3 = **new** **A**(10, 20); // Using Third Constructor
8. }
9. }

6)Duplicate Constructors not allowed. If you keep duplicate constructors, you will get compile time error.

1. class A
2. {
3. **A**(**int** i)
4. {
5. // Duplicate Constructor
6. }
7. **A**(**int** i)
8. {
9. // Duplicate Constructor
10. }

7) Multiple arguments of the constructors can’t have same name. If the two arguments have the same name, you will get compile time error.

1. class A
2. {
3. **A**(**int** i, **int** i)
4. {
5. // Duplicate Arguments Passed. It gives compile time error
6. }
7. }

8) Only public, protected and private keywords are allowed before a constructor name. If you keep any other keyword before a constructor name, it gives compile time error.

1. class A
2. {
3. **final** **A**()
4. {
5. **//Constructor can not be final**
6. }
7. static **A**()
8. {
9. **//Constructor can not be static**
10. }
11. abstract **A**()
12. {
13. **//Constructors can not be abstract**
14. }
15. }

9) First statement in a constructor must be either super() or this(). If you put any other statements you will get compile time error.If you don’t include these statements, by default compiler will keep super() calling statement. **super() – It is a calling statement to default constructor of super class. this()- it is a calling statement to constructor of the same class.**

**package** constructors;

**public** **class** ConstructorWithSuperThisKeyWord {

ConstructorWithSuperThisKeyWord(){

//By Default, Compile will keep super() calling statement here.

System.***out***.println(" First Constructor");

}

ConstructorWithSuperThisKeyWord(**int** i){

//Compiler will not keep any statement here

**super**();

System.***out***.println(" Second Constructor");

}

ConstructorWithSuperThisKeyWord(**int** i, **int** j){

//Compiler will not keep any statement here

**this**();

System.***out***.println(" Third Constructor");

}

ConstructorWithSuperThisKeyWord(**int** i, **int** j, **int** k)

{

**this**();

System.***out***.println("Fourth Constructor");

// super(); It will give error if you keep super() here

}

**public** **static** **void** main(String[] args) {

//ConstructorWithSuperThisKeyWord A1=new ConstructorWithSuperThisKeyWord(); --- output : first constructor

//ConstructorWithSuperThisKeyWord A2=new ConstructorWithSuperThisKeyWord(10); --- output : second constructor

//ConstructorWithSuperThisKeyWord A3=new ConstructorWithSuperThisKeyWord(10,20); --- output : first constructor , third constructor

ConstructorWithSuperThisKeyWord A4=**new** ConstructorWithSuperThisKeyWord(10,20,30); //--- output : first constructor, fourth constructor

}

}

10) Recursive constructor calling is not allowed.

1. class A
2. {
3. **A**()
4. {
5. **this**();
6. // It gives compile time error
7. }
8. }

11) No Cylic calling of constructors.

1. class A
2. {
3. **A**()
4. {
5. **this**(10);
6. // It gives compile time error
7. }
8. **A**(**int** i)
9. {
10. **this**();
11. // It gives compile time error
12. }
13. }