# Chapter 10

# Variables and Types of Variables

- In this chapter, You will learn
  - Definition of variable
  - o Limitation of variable
  - Variables creation syntax
  - Types of variables
  - Local variable and its rules
  - Modifiers allowed for variables
  - 8 Types of members defined in a single class and their execution control flow
- ➤ By the end of this chapter- you will learn defining, declaring, calling variables, and its execution control flow.

# **Interview Questions** By the end of this chapter you answer all below interview questions Definition How can we create a variable? Limitation on variable Defining, declaring, initializing, reinitializing, and calling/using a variable Types of variables Method Level Parameter Local final Class Level static non-static final transient volatile Rules on local variables and Parameters? What are the modifiers allowed for a variable? Default values for variables Life-time and scope of a variable How many members we can define inside a class? iii

ariables and Types of Variables

# Variables and Types of Variables

#### Definition

Variable is a named memory location used to store data temporarily. During program execution we can modify that data.

## How can a variable be created?

A variable can be created by using Datatype. As we have two types of datatypes we can create two types of variables

- 1. Primitive variables These variables are created by using primitive datatypes.
- 2. Referenced variables These variables are created by using referenced datatypes.

The difference between primitive and referenced variables is "primitive variables stores data directly, where as referenced variables stores reference of the object, not direct values".

**Note:** As per compiler and JVM we do not have a separate name called "referenced variable". It means, the variables created by using referenced datatype are also considered as like normal variables only.

Below program shows creating primitive and referenced variables

```
//Example.java
class Example
{
    int x = 10;
    int y = 20;
}
```

Referenced variables are initialized with object reference that is created and returned by "new" keyword, as shown in the left diagram.

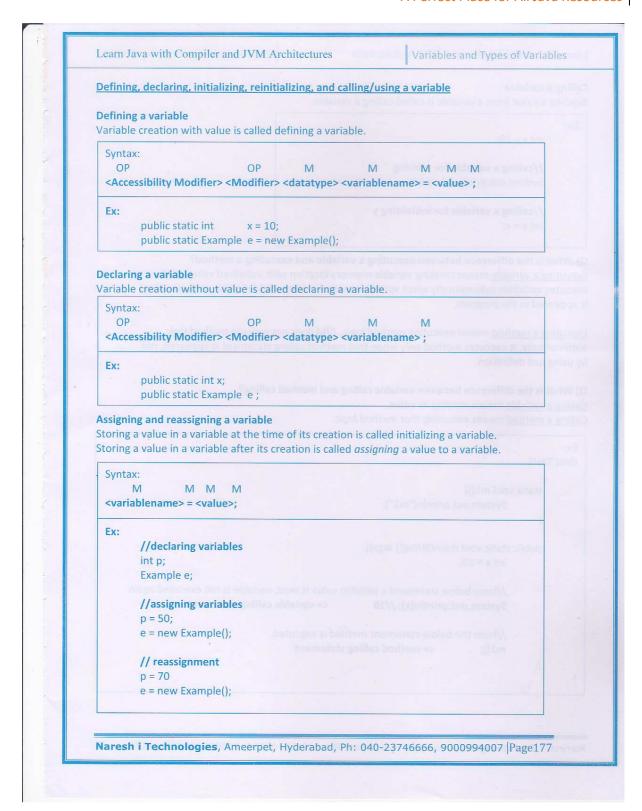
### Limitation of variable

It can only store single value at a time. If we assign new value, old value is replaced with new value. It means, always it returns latest modified value.

- If we modify primitive value previous value is replaced with new value
- If we modify referenced variable previous object's reference is replaced with new object's reference and now this referenced variable is referencing to this new object.

Naresh i Technologies, Ameerpet, Hyderabad, Ph: 040-23746666, 9000994007 | Page 175

```
Learn Java with Compiler and JVM Architectures
 Show memory structure from the below program and also output
      class Sample {
                          public static void main(String[] args) {
                                               int a = 50;
                                                System.out.println("a: "+a);
                                               a = 70;
                     System.out.println("a: "+a);
                                        Example e1 = new Example(); Example Ex
                          System.out.println("e1: "+e1); see saids not seed a said that become
                         el = new Example();
                                      System.out.println("e1: "+e1);
Output:
a: 50
a: 70
e1: Example@addbf1
e1: Example@42e816
Conclusion:
           • The value of primitive variable is mathematical data based on its data type.
           • The value of referenced variable is its class object's reference.
Naresh i Technologies, Ameerpet, Hyderabad, Ph. 040-23746666, 9000994007 Page176
```



```
Learn Java with Compiler and JVM Architectures
                                                                Variables and Types of Variables
Calling a variable
Reading a value from a variable is called calling a variable.
 Ex:
        int x = 10;
        //calling x variable for printing
        System.out.println(x); // 10
        //calling x variable for initializing y
        int y = x;
Q) What is the difference between executing a variable and executing a method?
Executing a variable means creating variable memory location with initialized value. JVM
executes variables automatically when variable creation statement (definition or declaration)
is appeared in the program.
Executing a method means executing method logic. JVM does not execute method logic
automatically. It executes method only when that method calling statement is appeared, not
by using just definition.
Q) What is the difference between variable calling and method calling?
Calling a variable means reading its value.
Calling a method means executing that method logic.
  class Test{
         static void m1(){
                System.out.println("m1");
         public static void main(String[] args){
                int x = 10;
                //from below statement x variable value is read, variable is not executed again.
                System.out.println(x); //10
                                                <= variable calling statement
                //from the below statement method is executed.
                              <= method calling statement
                m1();
Naresh i Technologies, Ameerpet, Hyderabad, Ph: 040-23746666, 9000994007 | Page 178
```

Variables and Types of Variables

#### **Types of Variables**

Based on class scopes variables are divided in two types

- 1. Local Variables, parameters
- 2. Class Level Variables
- The variables created inside a method (or) block are called local variables
- The variables created at class level are called class level variables.

#### 2 types of class level variables

Class level variables are divided into two types, based on the time they are getting memory location. They are

- Static variables
- Non-static variables

#### Definitions

The class level variable which has static keyword in its creation statement is called static variable, else it is called non-static variable.

#### Memory location of all above three variables

- Local variables get memory location when method is called and their creation statement is
  executed. They get memory with respect to method, so they are also called method
  variables. Local variables are automatically created when method is executing and are
  destroyed automatically after method execution is completed, so they are also called auto
  variables.
- Static variables get memory location when class is loaded into JVM. They get memory with respect to class name, so they are also called "class variables" also called "Fields".
- Non-static variables get memory location when object is created using new keyword. They get memory with respect to object, so they are also called "object variables or instance variables or properties or attributes" are also called "Fields".

Below diagram shows different scopes in class and all above three variables creation.

```
class Example{
    //static variables
    static int a = 10;
    static int b = 20;

//non-static variables
    int x = 30;
    int y = 40;
```

```
public static void main(String[] args)
{
     //local variables
     int p = 50;
     int q = 60;
}
```

# Q) How many variables are created in above program?

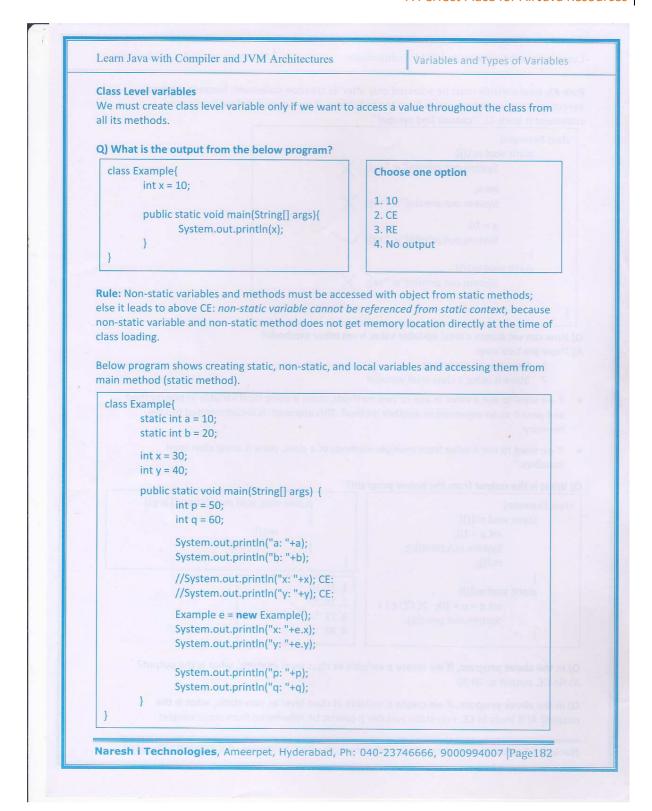
A) 5 variables. They are a, b, args, p, q.

Variables x, y are not created, because object is not created.

Naresh i Technologies, Ameerpet, Hyderabad, Ph: 040-23746666, 9000994007 Page179

```
Learn Java with Compiler and JVM Architectures
                                                              Variables and Types of Variables
Local variables and its rules
While working with local variables, we must follow below 3 rules
Rule #1: Local variable cannot be accessed from another method. Because its scope is
restricted only within its method, also we cannot guarantee variable creation, because that
method may or may not be called. It leads to CE: cannot find symbol
For example:
 class Example
        public static void main(String[] args)
               int a = 10;
               System.out.println("a: "+a);
        static void m1()
               Rule #2: Local variable should not be accessed without initialization.
It leads to CE: "variable might not have been initialized"
 class Example {
        public static void main(String[] args){
               int a = 10;
               int b;
               System.out.pritnln("a: "+a);
               //System.out.pritnln("b: "+b); X //CE: variable b might not have been initialized
               a = a + 10:
               //b = b + 10; \times //CE: variable b might not have been initialized
               System.out.pritnln("b: "+ b);
               b = b + 10;
               System.out.pritnln("a: "+ a);
               System.out.pritnln("b: "+ b);
Naresh i Technologies, Ameerpet, Hyderabad, Ph. 040-23746666, 9000994007 | Page 180
```

```
Learn Java with Compiler and JVM Architectures
Rule #3: local variable must be accessed only after its creation statement; because method
execution is sequential execution from top to bottom. If we access it before its creation
statement it leads CE: "cannot find symbol"
   class Example{
          static void m1(){
                 System.out.println("a: "+a);
                 System.out.println("a: "+a);
                 System.out.println("a: "+a);
          static void m2(){
                 System.out.println("a: "+a);
Q) How can we access a local variable value from other methods?
A) There are two ways
       1. Pass it as an argument
       2. Store it using a class level variable
• If we want to use a value in one or two methods, store it using local variable in one method
    and pass it as an argument to another method. This approach is recommended to save
    memory.
   If we want to use a value from multiple methods of a class, store it using class level
    variables.
Q) What is the output from the below program?
  class Example{
                                                       public staic void main(String[] args)
         static void m1(){
                int p = 10;
                                                              m1();
                System.out.print(p);
                m2();
                                                1.1010
         static void m2(){
                                                2.1020
                int q = p + 10; \times CE: cfs
                                                3. CE \
                System.out.print(q);
                                                4. RE
Q) In the above program, if we create p variable at class level as static, what is the output?
A) No CE, output is: 10 20
 Q) In the above program, if we create p variable at class level as non-static, what is the
 output? A) It leads to CE: non-static variable p cannot be referenced from static context
 Naresh i Technologies, Ameerpet, Hyderabad, Ph: 040-23746666, 9000994007 | Page 181
```



```
Learn Java with Compiler and JVM Architectures
                                                                  Variables and Types of Variables
Q) Can we declare local variable or parameter as static?
No, local variables can't be declared as static it leads to CE: illegal start of expression. Because
local variable should get memory location only if method is called. But if we declared as static,
it should get memory at the time of class loading, this is violating contract, so it leads to
compile time error.
For Example
      class Example{
             static int a = 10;
             static void m1(){
                     int p = 20;
                     static int q = 30; X CE: illegal start of expression
Q) The variables created inside a static method are static? => No, they are still local
Q) The variables created inside a non-static method are non-static? => No, they are still local
Final variables
The class level or local variable that has final keyword in its definition is called final variable.
Rule: once it is initialized by developer its value cannot be changed. If we try to change its
value it leads to compile time error.
Below program shows creation final variables
  class Example{
         static int a = 10;
                                       <= normal static variable
         static final int b = 20;
                                       <= final static variable
         int x = 30;
                                       <= normal non-static variable
         final int x = 40;
                                       <= final non-static variable
         public static void main(String[] args) {
                 int p = 50;
                                       <= normal local variable
                                       <= final local variable
                 final int q = 60;
                 //q = 70; CE: cannot assign a value to final variable q
                 final int r;
                 r = 70;
                 //r = 80; CE: variable r might already have been assigned
Naresh i Technologies, Ameerpet, Hyderabad, Ph: 040-23746666, 9000994007 Page183
```

Variables and Types of Variable

#### transient variables

The class level variable that has transient keyword in its definition is called transient variable.

Rule: local variable cannot be declared as transient. It leads to CE: illegal start expression

Find out CE in program if any?

```
class Example{
    static transient int x = 10;
    transient int y = 20;
    static void m1(){
        transient int z = 30;
    }
}
```

**Note:** We declare variable as transient to tell to JVM that we do not want to store variable value in a file in object serialization. Since local variable is not part object, declaring it as transient is illegal. For more details on object serialization and transient variable refer lOStreams material.

#### Volatile variable:

The class level variable that has volatile keyword in its definition is called volatile variable.

Rule: local variable cannot be declared as volatile. It leads to CE: illegal start expression

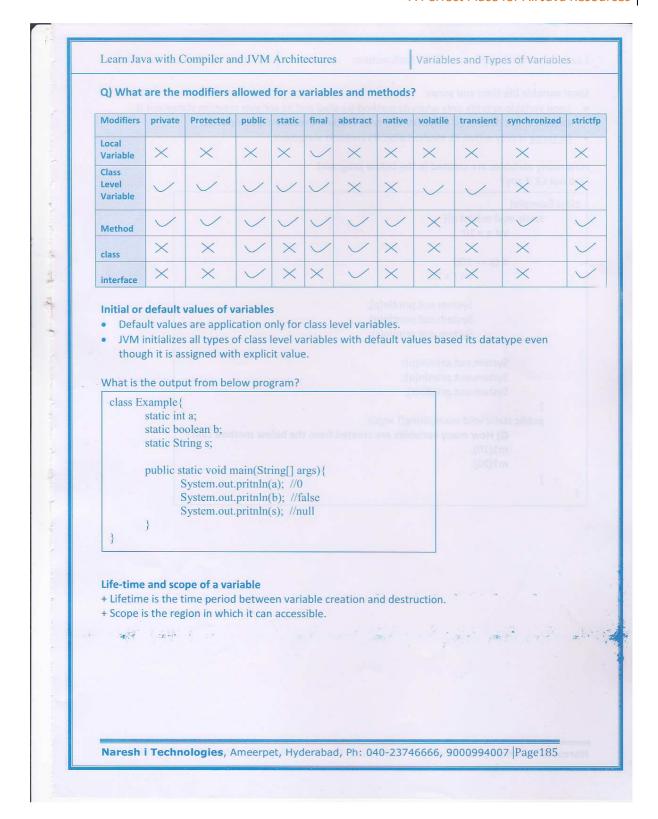
Find out CE in the below program if any?

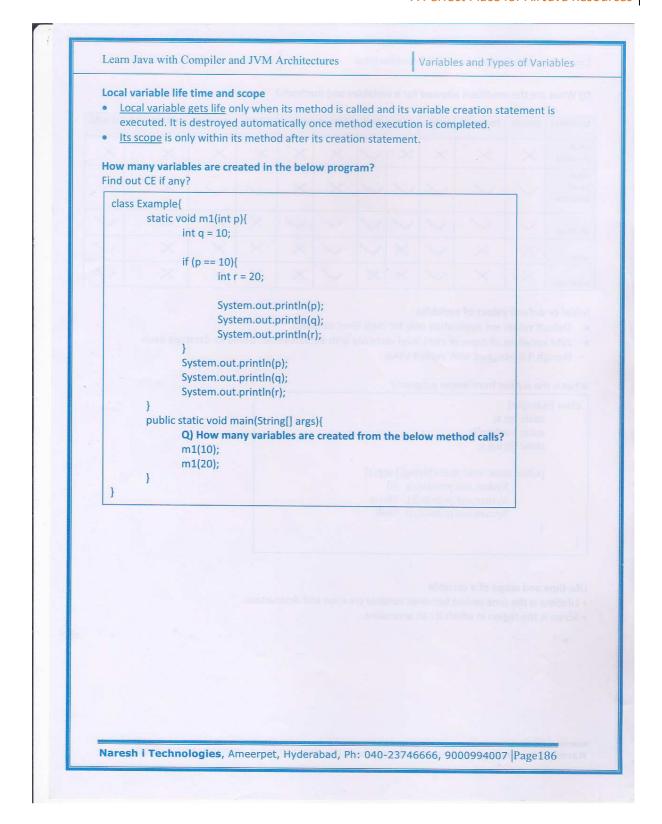
```
class Example{
    static volatile int x = 10;
    volatile int y = 20;
    static void m1(){
        volatile int z = 30;
    }
}
```

**Note:** We declare variable as volatile to tell to JVM that we do not want to modify variable value concurrently by multiple threads. If we declare variable as volatile multiple thread are allowed to change its value in sequence one after one.

Since local variable is not directly accessible by thread, declaring it as volatile is illegal. For more details on volatile and synchronized keywords refer Multithreading material.

Naresh i Technologies, Ameerpet, Hyderabad, Ph: 040-23746666, 9000994007 | Page184





Variables and Types of Variables

# Static variable life time and scope

- <u>Static variable gets life</u> when class is loaded. It is destroyed either if class is unloaded from JVM or if JVM is destroyed.
- <u>Its scope</u> is throughout class, and also in the place where class is accessible. It means where ever class is available there static variable is available provided it is public.

```
class Example{
    static int a = 10;
    static void m1(){
        System.out.pritnln(a); //10
    }
    public static void main(String[] args){
        System.out.pritnln(a); //10
    }
}
```

## Non-static variables life time and scope

- <u>Non-Static variable gets life</u> when object is created. It is destroyed when object is destroyed. object is destroyed when it is unreferenced or its referenced variable is destroyed.
- <u>Its scope</u> is the scope of object, object is available only if its referenced variable is available.

```
class Example{
    int x = 10;
    static void m1(){
        Example e1 = new Example();
        System.out.pritnln(e1.x);
    }
    public static void main(String[] args){
        Example e2 = new Example();
        System.out.pritnln(e2.x);
        System.out.pritnln(e1.x);
}
```

In the above program you have one compiler time error, can you find it out?

You cannot access *e1* variable in main() method because it is local to m1() method. So main method *e1.x* statement leads to CE: cannot find symbol symbol: variable e1

Naresh i Technologies, Ameerpet, Hyderabad, Ph: 040-23746666, 9000994007 | Page 187

