To write instructions we need a programming language

Programs are set of instructions

Set of programs make up a software

Oops:

In oops while solving a problem

1)objects should be marked

2)every object we mark should have 2parts

1)HAS part /fields/attributes (store them as variables)

2)DOES part / behaviour (represent them as methods)

3) to represent object we need blueprint of object.

4) we use “new” keyword to create object of blueprint class.

What it has part and does part represents

Has part – what it can hold

Does part – what can do

Eg: student

Email , rollno , gender, age(variables/ identifiers)

Playing, sleep, run(methods)

To represent blue-print we use reserve keyword called class

// globally used conventions

// pascal convention for class declaration

Variables are written in camel case

Example : firstName , lastName

Methods also will be written in camel case

Example: toUpper() , toLower().

Class Student { // blueprint of student object

String email;

int roll.no;

int age;

char gender;

void playing() // while playing nothing happens to me so marked as void

void sleep() // while sleeping nothing will happen during sleep so void

To create an object we use new keyword

Syntax:

ClassName variable = new Classname();

new -> it is a signal to jvm to create a some space for object in heap area.

we inform the class name, jvm will create the hash code to user

User collects That hash code is stored in the reference variable

// all the above will be done when bytecode is given to jvm.

What is object?

Physical existence of an element we say object

Eg: car, bike, airplane etc.

Realtime Eg: Bookmyshow

Person, ticket, cinema hall ,seat, etc.

All the above will be virtually available in the application

Types of variables

Based on type of values assigned to a variable, they are divided into two types

1. Reference variable

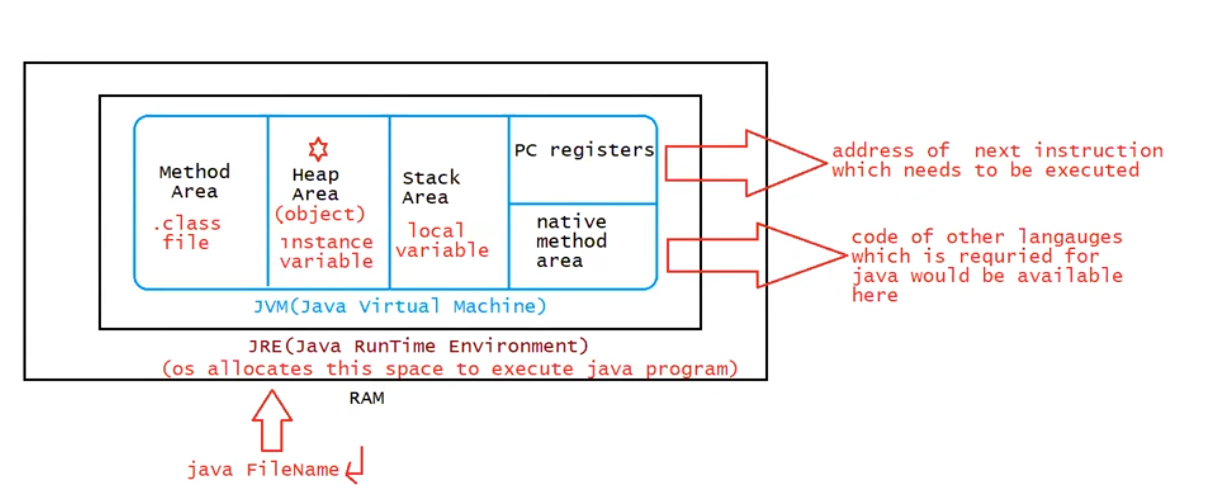
reference variables are used to assign objects

ClassName variable = new Classname();

1. Primitive variable

Primitive variables are assigned to represent primitive values

int a = 10

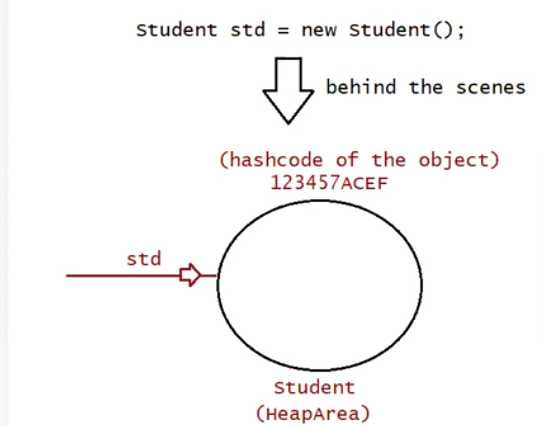


instance variables:

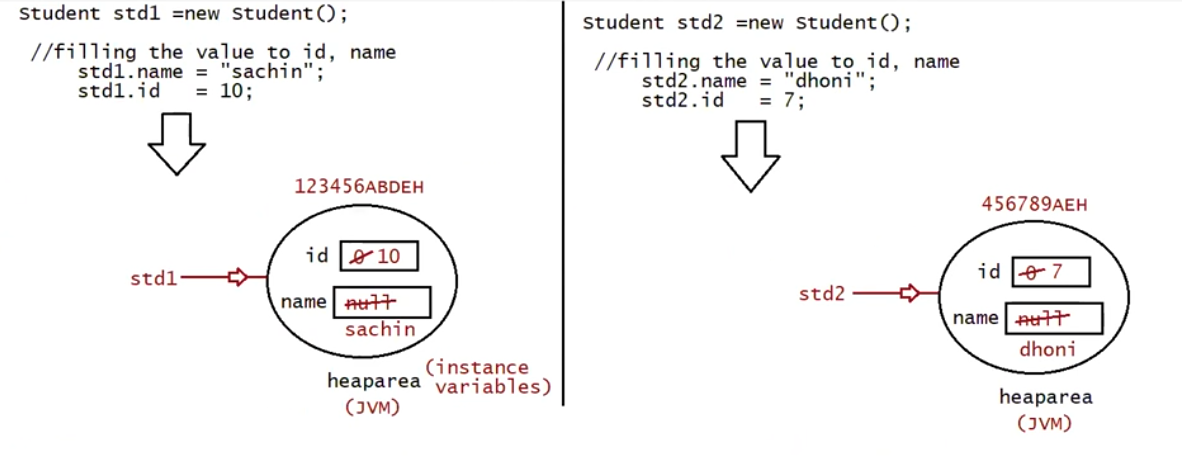
variable is declared inside the class and outside the method

or

\*imp def if the value of the variable changes from object to object then it is called instance variable



Eg: Student obj = new Student(); // id = 10 name = pavan Student obj1 = new Student(); // id = 7 name = kalyan



When object is created, jvm will create a memory in the heap area, and default values will be assigned based on the datatype.

Note: The scope of instance variable is available only until the reference variable is pointing to the object

Accessing Instance Variables

We can access the instance variable using two ways.

1. Using the "this" keyword.

Eg: Instance\_Variable\_This\_Keyword\_3

1. Using an object of the class.

Eg: Instance\_Variable\_1

Instance\_Variable\_2

If object references becomes null we can’t access “instant variables”

* For every object a separate copy of instance variables is created
* Instance variables are created at a time of object creation and destroyed at a time of object destruction.so scope of instance variables is exactly same as scope of object.
* Instance variables can be accessed directly from instance area. but direct access is not possible in static area you should create obj to and use that variables with the obj.
* Instance variables are inside class and outside of any method, block, constructor
* Instance variables are stored in the heap area as a part of object.
* Instance variable doesn't get memory at compile time.
* It gets memory at runtime when an object or instance is created. That is why it is known as an instance variable.
* The instance variables cannot be marked as static.
* instance variables can be marked by the access specifiers (or) modifiers
* Instance variables cannot be marked by the abstract keyword(abstract keyword only used for the methods).
* Instance variables cannot be marked by the strictf keyword.
* Instance variables can be declared as final but not static.

Local variable

* Variables created inside a method are called local variables
* During the execution of the method memory for local variables are created in stack (push), and after execution they are taken out (pop)
* Jvm do not provide default values for local variables, they should be initialized during the declaration.
* If not initialized by programmer and used in program results in C.E

Keypoints on local variable:

* Sometimes to meet the temporary requirements of the programmer we declare variables inside the method (or) block (or) constructor such variables are called as temporary variables (or) automatic variables (or) stack variables (or) local variables.
* Temporary variables are stored in the stack
* Automatic variables are created during the execution of a block and destroyed after the block execution.
* Stack variables are stored inside the stack.
* Access modifiers cannot be used for local variables.
* A local variable cannot be defined with "static" keyword.

Eg: Local\_Variable\_Eg1

//go through the code

Eg: Local\_Variable\_Eg2

// go through the code

Eg: Local\_Variable\_Eg\_3

// go through the code

Eg: Local\_Variable\_Eg4

Since local variable do not have default values Local it must be initialized before using it, if declared and not initialized and used no problem.

If declared not initialized and tried to use it will lead to compile time error.

Eg: Local\_Variable\_Eg5

// same as above go through the code.

final Local Variable

final is the only allowed access modifier for local variables.

final local variable is not required to be initialized during declaration.

final local variable allows compiler to generate an optimized code.

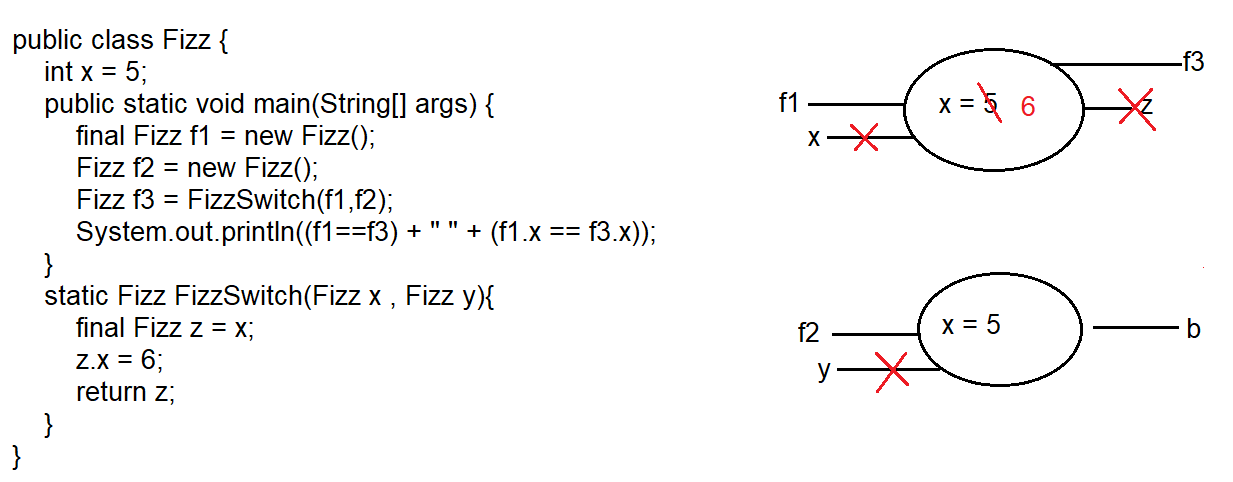
final local variable can be used by anonymous inner class or in anonymous methods.

Eg: Local\_Final\_Variable\_2

Eg: Variable\_Declaration

// go through the code

Eg: Fizz



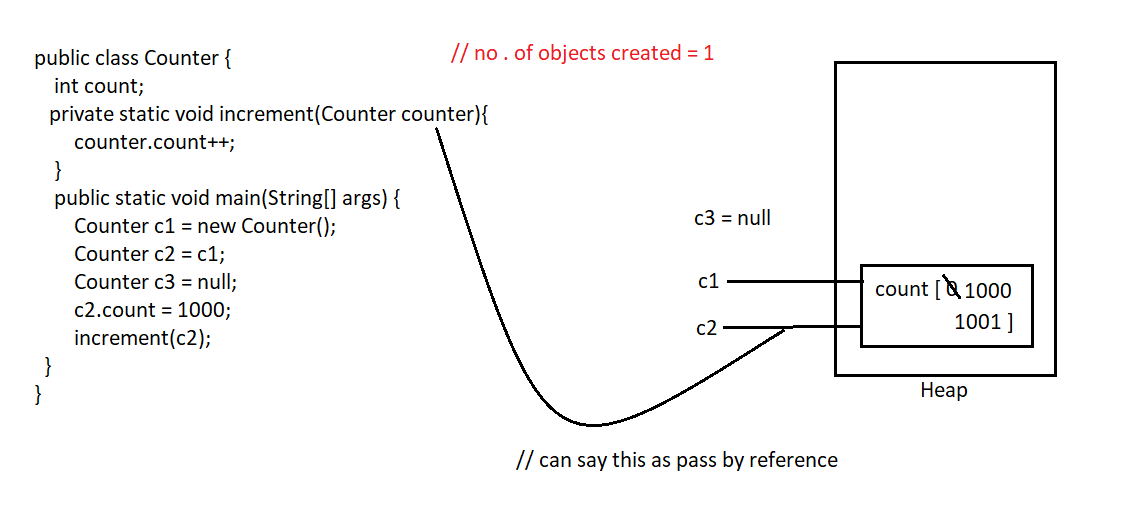
Explanation:

1. First control comes to the main method, final object f1 is created, if anything is made final changes can’t be made on the object.
2. After that object f2 is created.
3. Static methods can be called directly with name as above, here static method FizzSwitch() has class return type Fizz
4. f1 , f2 reference variables are passed as arguments to FizzSwitch(), means parameter x is refers to object f1, and y to object f2 .
5. then x is assigned to z.
6. with the help of the reference variable z value to x changed from 5 to 6 .
7. z is a reference variable storing address of the object . the static method is returning z and it is stored in f3 .
8. After the execution of the static block , local variables x , y , z are deleted ( local variables have scope only within the block .
9. Since f1 and f3 are pointing to same object their reference address is same , so it evaluates to true .
10. Value of x for f1 and f3 are same , so it is also true .

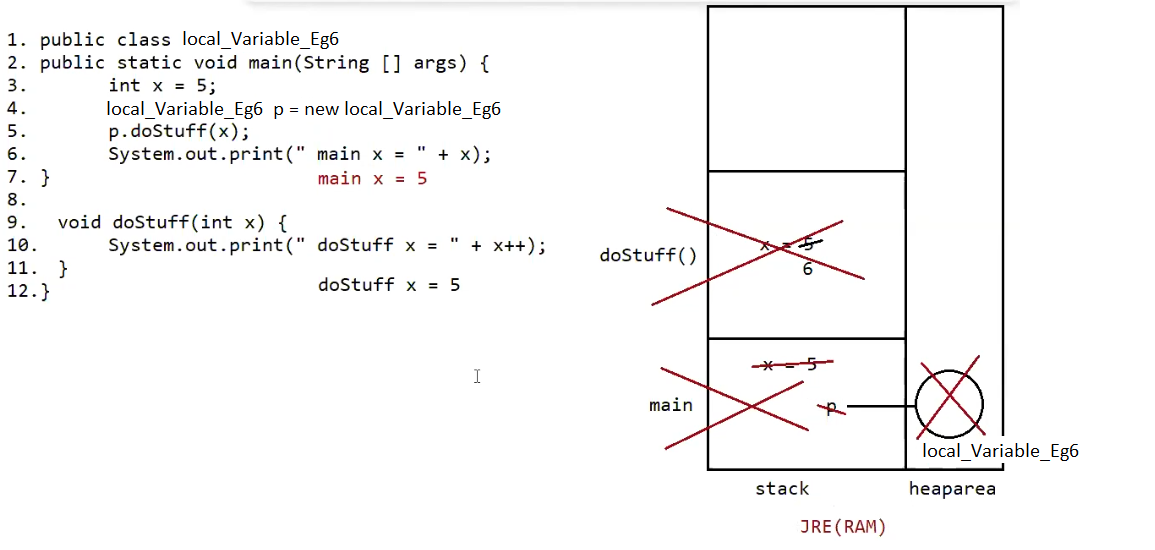
Eg: ReturnType

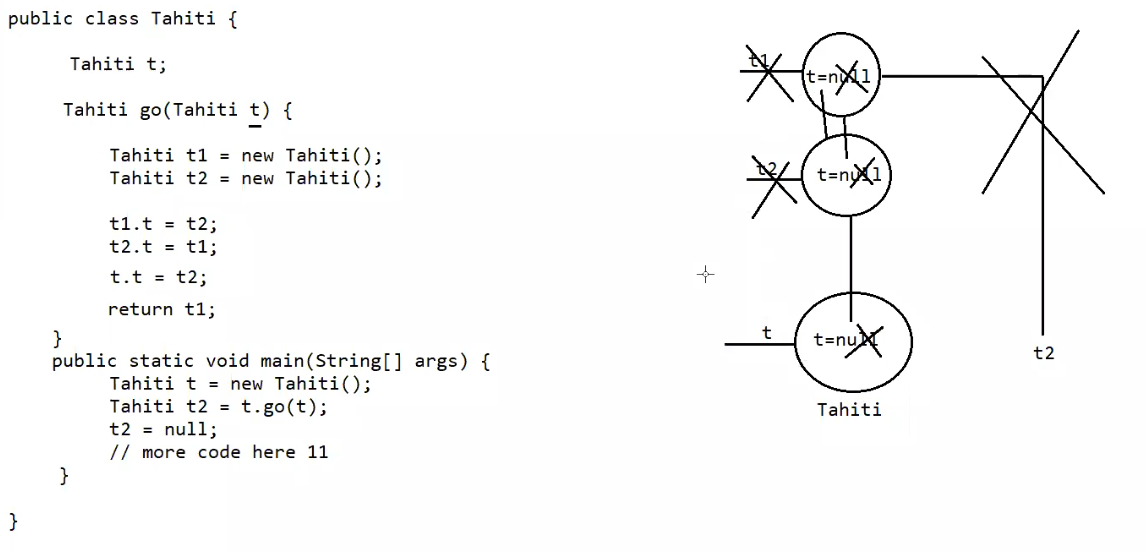
// go through the code

Eg: Counter



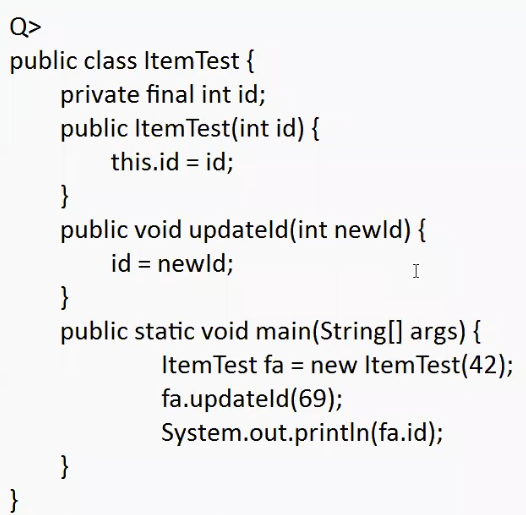
Eg: Local\_Variable\_Eg6



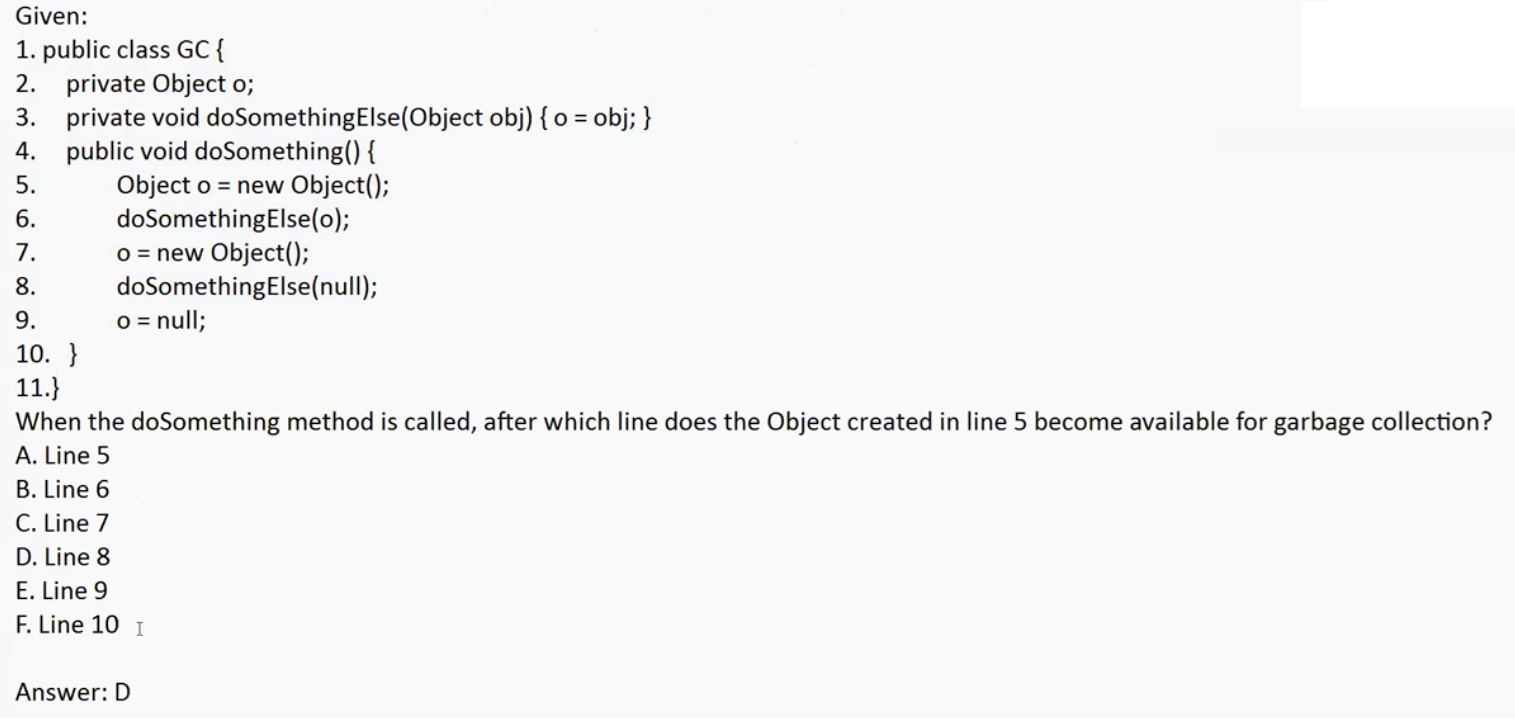


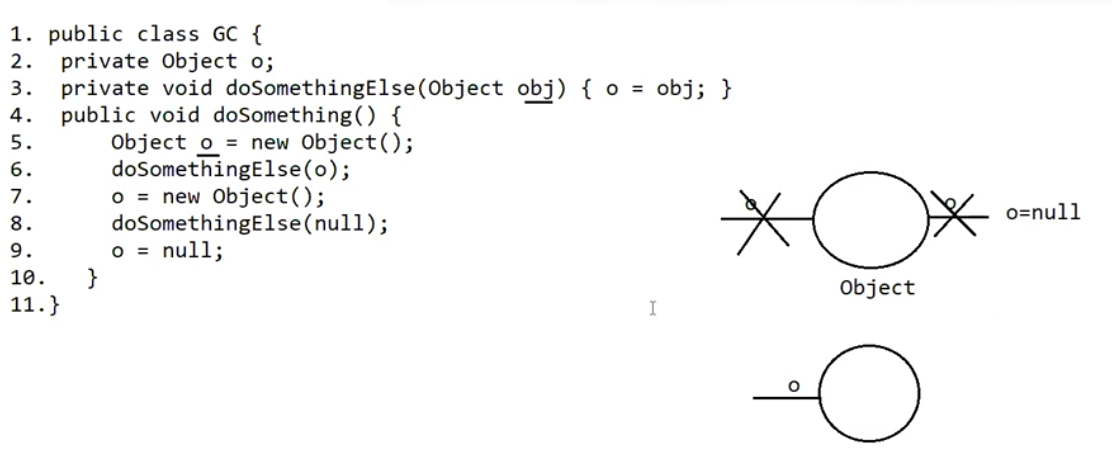
Here no objects are eligible for garbage collection. go() method reference t1 and t2 are removed after the execution of go() method. But address of t2 object is available in t1 reference variable t, similarly t2 object address is available in t1 reference variable t , so they are not subjected to garbage collection.

This problem is also called island of isolation.



Compilations fails, final variable can’t be reassigned. Compilation fails while reassigning 69 to id in method updateId.





// work on it very carefully