Interface and it’s declaration

- Interface is nothing but the pure abstract class which contains only abstract method.
- Interface are always abstract whether we mentionit or not.
- Declaration :

In order to declare an interface the keyword “interface” is used. e.g.

```
public interface MyInterface{ }
```

Interface method properties

- **Interface methods are always “public” and “abstract” , so mentioning it as “public” and “abstract” is redundant.**
- **public abstract interface** Test {
- **public abstract void** go();//public and abstract is redundant
- **abstract void** go2();//still don't need to use abstract
- **void** go3();//whether we mention or not...it is always public and abstract
- **private abstract void** go4(); //it will not compile}

- **The return type of interface methods can be anything, just like normal methods, but same should be in overridden method and body of the method should be always empty.**
-
- **public abstract interface** Test {
- String go();
- **int** first();
- **boolean** second();}

Interface method properties continues

- **No access modifier apart from public can be used with interface methods (using "public " is redundant) and no non-access modifier apart from abstract can be used with interface methods (using "abstract" is also redundant).**
- **We are not allowed to use "final" and "static" as it will restrict the method to overridden.**
- **public abstract interface** Test {
- String go();
- **int** first();
- **static boolean** second(); //only public and abstract is permitted}

- **The variables of the interface must be always "public", "static", and "final", hence they are always constant.**
- **public abstract interface** Test {
- String go();
- **public static final float** *PI* = 3.14f;}

Super keyword and it’s use

- Super keyword is used to call the “superclass version” of overridden method from subclass.
- **public void sound () //in animal class**
- {
- System.*out*.println(*"generic sound"*);
- }
- **public void sound() // in dog class**
- {

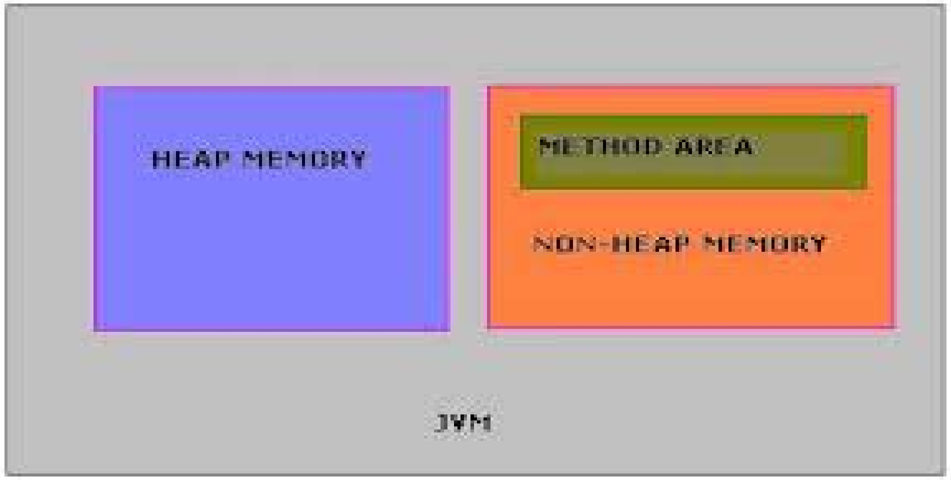
System.*out*.println(*"woof woof!!!"*);

- **super.sound();**
- }

Memories in java

- In java there are two areas of memory which we should care about:

1) Stack
2) Heap



- Heap:

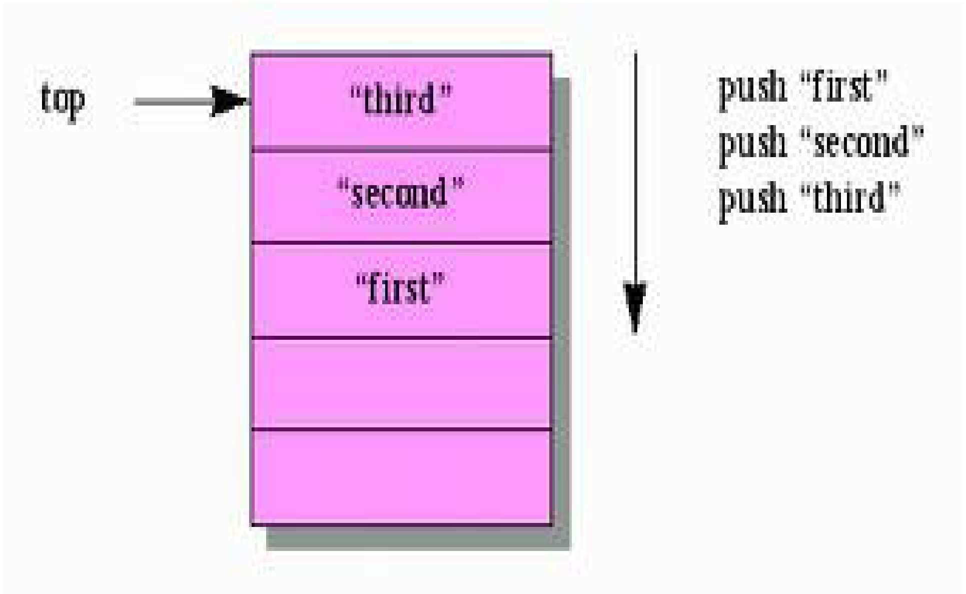
It is that area of memory in which object and instance variable lives.

- Stack:

It is that area of memory in which methods and local variable lives.

Stack Mechanism

- Whenever a method is called or executed, it goes on top of the stack in a stack frame. The stack frame holds the method and it’s local variable.
- Once the method hits it’s closing curly brackets, then the stack frame is popped off and the lower stack frame is executed.



Constructors, it’s declaration and use

- Constructors are nothing but the code which gets executed whenever a class is instantiated.
- Declaration of constructors:

1)Constructor must use the name of Class only and not any other name.

2)Constructor must not contain any return type, otherwise it will become a method.

3) The main use of constructors is to intialize the instance variables.

- **public Dog() //in Dog class**
- {
- System.*out.println("I am constructor");*
- }

Default constructor and constructor overloading

- The default constructor is a no-argument constructors. E.g.

public Dog(){} //constructor for Dog class

- The compiler provides a default constructor only when no constructor is provided by the programmer.
- Constructor Overloading:

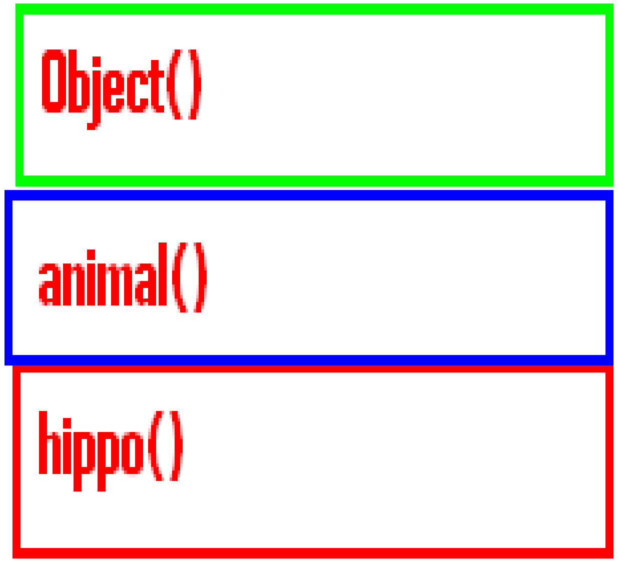
Constructors can be overloaded like methods, on the basis of different argument list or their order

- **public Dog()**
- {System.*out.println("I am constructor");*}
- **public Dog (int x)**
- {System.*out.println("in int x constructor");*}
- **public Dog(int x, String y)**
- {System.*out.println("in intx and string y constructor");*}
- **public Dog(String y, int x)**
- {System.*out.println("in String y and int x constructor");*}

Constructor chaining

- Constructor chaining is basically a process in which whenever constructor of sub-class is called, it will call the constructor of all superclasses until constructor of class Object is called.

1) First we call the constructor of Hippo class.
2) The subclass will call the Constructor of Animal class.
3) Finally the animal constructor Calls the Object constructor

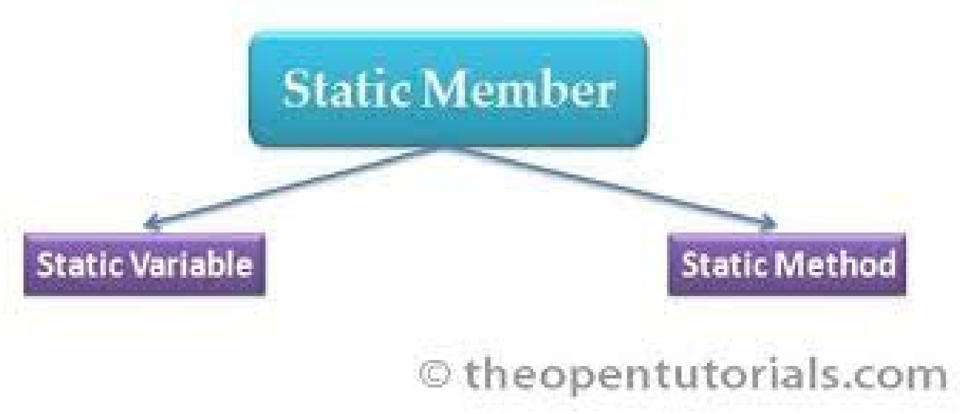


Superclass Constructors

- If we don’t call the constructor of superclass, the compiler implicitly calls the “default” constructor of super class.
- But if the constructor of super class is overloaded, it is possible to call it from child class by using the keyword “super(argument list)”.

Static components and it’s calling

- Static Components are those components which doesn’t requires instance of that class to access them. //in Test class



```
public static void go()
{ System.out.println
(“in static method”);}
```

- Calling of static components:

In order to call static components we don’t need the instance of the class, we just need name of the class and dot operator. E.g.

```
Test.go();
```

Static methods Vs Non-static methods

Static Methods

- It never uses the instance variables.
- Calling of this method doesn’t requires instantiation of class.
- These methods are called by the help name of class and dot(.) operator.

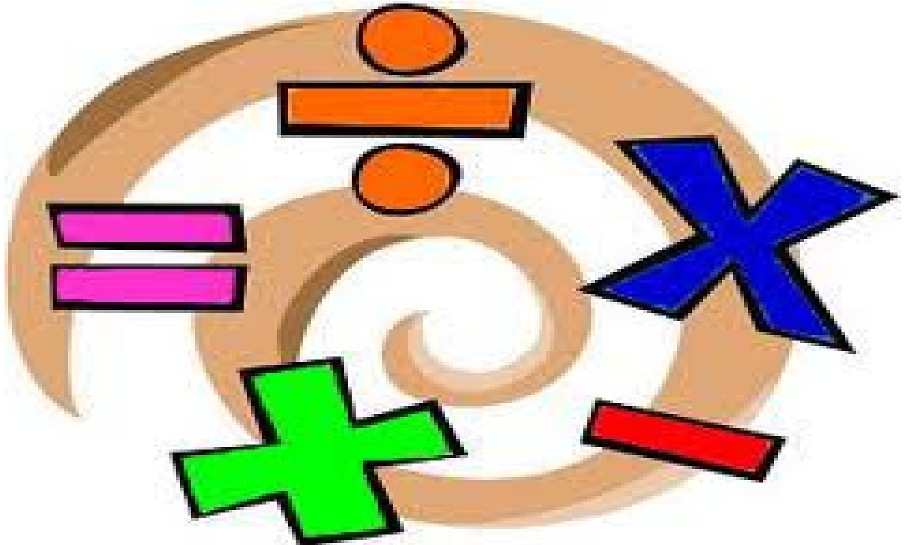
Non-static Methods

- It always uses the instance variables.
- Class needs to be instantiate to call this method.
- These methods are called by the help of instance variables.

Example of static class - Math class

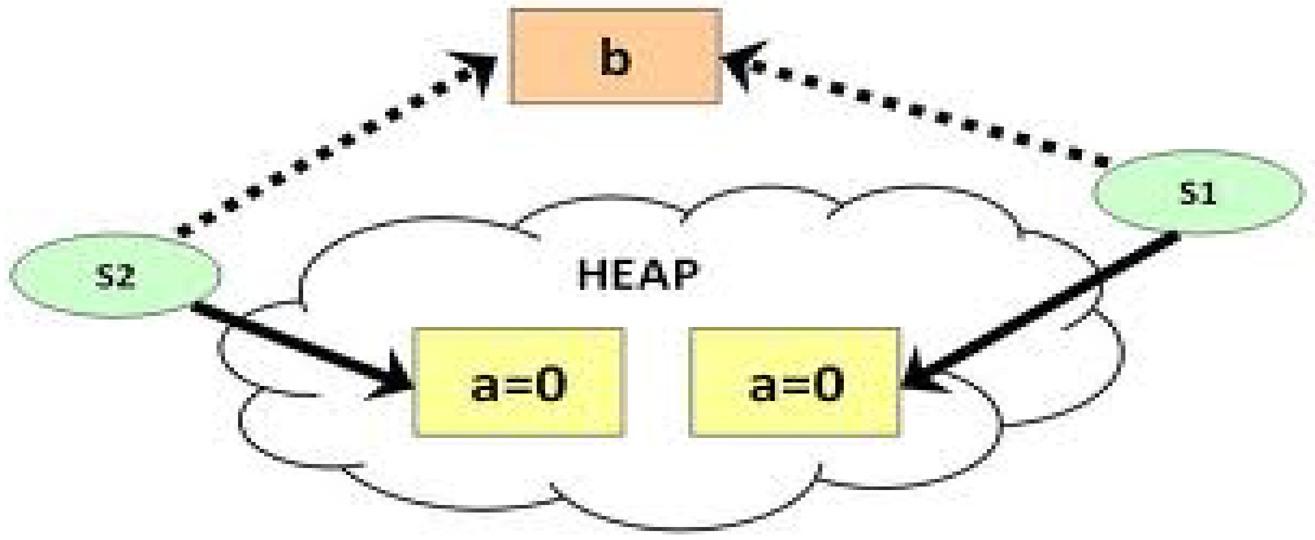
- The class Math contains methods for performing basic numeric operations such as the elementary exponential, logarithm, square root, and trigonometric functions.
- All the methods of Math class are STATIC, therefore there is no need to instantiate the math class for using it's methods or variables.

e.g. Math.random()



Static variables and their initialization

- Static variables are those variables which are shared by all objects of a class.
- In other word, if an object changes the static variable then it will affect the same static variable of other objects also.



```
public class Test {
public static void main(String[] args)
{

new PrivateConst();
new PrivateConst();
new PrivateConst();
new PrivateConst();
}}

public  class PrivateConst {

static int staticVar =0;
int nonStaticVar=0;
public PrivateConst()
{
staticVar++;
nonStaticVar++;
System.out.println("static var value--> "+
staticVar +
"non-static var value--> "+ nonStaticVar);
}}
```

Final keyword and it’s application

- Final keyword is used to make anything ‘frozen’, means if a component is made final, then it cannot be changed in future.
- Application:

1. IF final keyword is used with class, then it cannot be extended in future.
2. If final keyword is used with methods, then it cannot be override in future.
3. If final keyword is used with variable , then it’s value cannot be changed in future, means it will be constant