Oops

1. Encapsulation => promotes privacy (or) security in general you will find data hiding and data binding
2. Inheritance => promotes code reuse ability
3. Polymorphism => code flexibility
4. Abstraction => hiding implementation details only features are visible

Encapsulation:

1. Providing security to data members (or) most important component of the object (data members)

* We are providing the security by limiting the access (giving private access modifier to the data members) since private modifier is used, outside the class direct access to the data members is not allowed.
* So access is possible by creating setters and getters methods in the same class where the variables are declared as private.

Setter:

if a method is doing some activity of receiving data from outside and assigning it to its data members we can say it as setter()

Recommended syntax for setter() :

* It should not have return type
* Its name should be preceded with “set” followed by variable name in the camel case

(example: setVariable() )

* It should accept the parameters with same type as private variable

Recommended syntax for Parameters

* Give the same name as private variables (so that the user can have more understanding that method is accepting parameters for particular variable)
* use this keyword to differentiate between local and instance private data members as they are declared with same name
* if this keyword is not used it will lead to shadowing problem means, within the getter or setter whenever we are using local variable and class variable of same name, the jvm doesn’t understand which value to assign (or) return, so it returns empty or null.

getter:

if someone wants to access the data present in the private variable they can use getter() method

Recommended syntax for getter() :

* It should have the return type as that of the variable
* Its name should be preceded with “get” followed by variable name in camel case

( example : getVariable() )

* It should not accept any parameters
* (setters and getters methods has some extra logic that is used just to limit the user by giving unnecessary values to the data members)
* To use private data members, we should create object in user class and with that

Object we call getters() and setters() of the data members

* Since direct access is not possible to data members it can be said as data hiding. (by this we can say providing security (or) data hinding is same.
* Data binding means data member and its getter() and its setter() methods are binded to work together, we can say that as data binding .

1. Encapsulation prevent direct access to the data members and provide controlled access through getters() and setters()
2. Encapsulation promotes data binding

Note: a class which has all the data members as private technically we can call it as bean. irrespective whether that data members have getters and setters are not.

Eg: Encapsulation

Note : universal recommendation For boolean data type: The setter is same as others , but for getter name should not start with get ,name should start with “is”

private boolean current;

// getter for boolean current

public boolean isCurrent(){

return current;

}

// setter for boolean current

public void setCurrent(final boolean current){

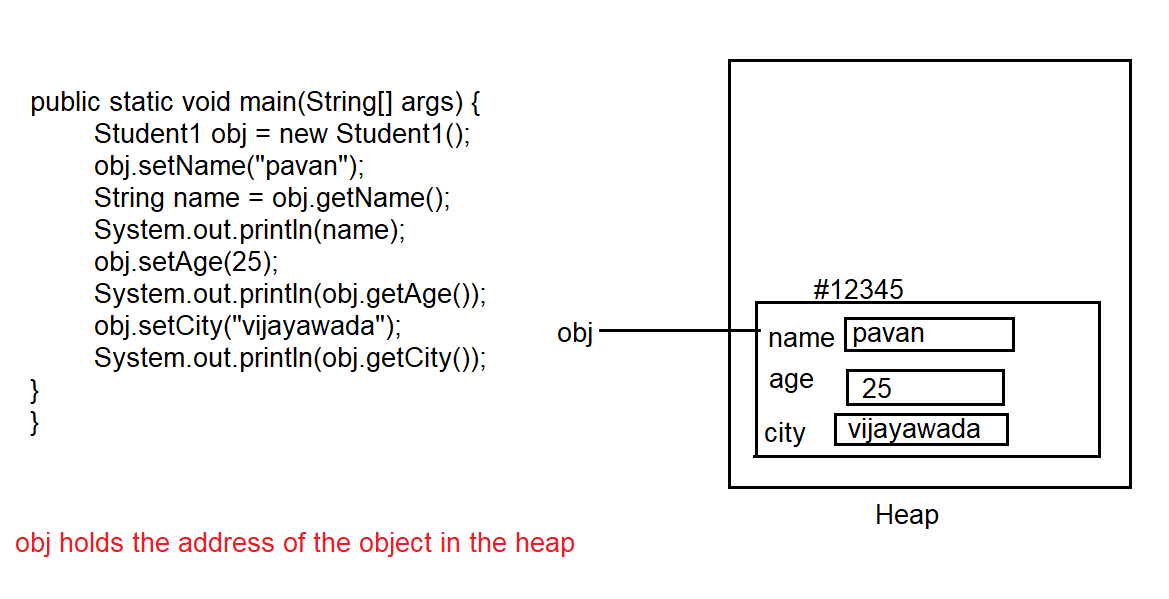
this.current = current;

}

this keyword refers to the current class running object

Eg: Encapsulation

// go through the program



// the above diagram represents the main method of eg: Encapsulation go through the complete code

Explanation :

1. When program is executed the control comes to the main method. object is created for the class Student, memory for instance variables is allocated, jvm gives default values to that variables on the basis of their data types until the values are assigned to the instance variables
2. The obj stores the address of the object in the Heap, now with that reference variable obj method setName() is called.
3. Control goes to the setName() and argument passed is assigned to it.
4. this keyword in the setName() methods takes the address of the object from obj reference variable. this.name (internally it is taken as objectaddress.name) is actually storing the local variable value in it ( instance variable in the heap)

void setName(String name){

this.name = name;

}

1. conclusion : we can create any number of objects in the program as we want . if we call setName() on that objects , current executing object address is stored in the this keyword.

Note: if the method return type is void you cannot collect it in the variable, since it does not return anything

We can write common method for setters, common method cannot be used for getters because each getter have different return types.

Eg: Common\_Setter

Note: even though we can write common method for setters it is not suggestable. You can use constructor for that .

Constructor:

1. constructor should have same name as of the class name
2. it should not have return type even void also
3. whatever rules are there for method parameter, it is same for constructor also
4. can have empty constructor also.
5. constructor is automatically invoked / called the moment you create the object
6. Whatever you do in the method you can do that in the constructor also except returning something
7. Access modifiers public, private, protected are allowed with constructors.
8. We can have one than one constructor in a class

Purpose of constructor:

if there is a requirement that something in your class needs to be executed whenever you create the object, then keep that thing in the constructor.

Eg: Constructor

If there is no constructor in the program only then, compiler will include a default constructor to the program behind the scene, which has no parameters like this.

Note: by default compiler will include only constructor, which has no / 0 parameters. if you don’t include super() method , internally super() method is included in the 1st line in default & user defined constructor .

super() will call the parent class constructor ( classname() with no arguments , if parent class is present)

Note: in java for any class explicitly you have not specified a parent, the parent is object

In the above Eg: Constructor, since there is no parent the parent is object

If this() is there in 1st line you can ignore super() , similarly if super() is there in 1st line you can ignore this() both cannot be in there if one is declared in the first line of the constructor

this(), super() method should only be in the first line , they cannot be in the last or other lines of code .

public Classname(){

super();

}

this() will call constructor of same class .

Constructor overloading is supported in java, means you can declare multiple constructors in the program only with different no. of parameters (or) same no. of parameters with different types, and to call them you should declare multiple objects according to their arguments.

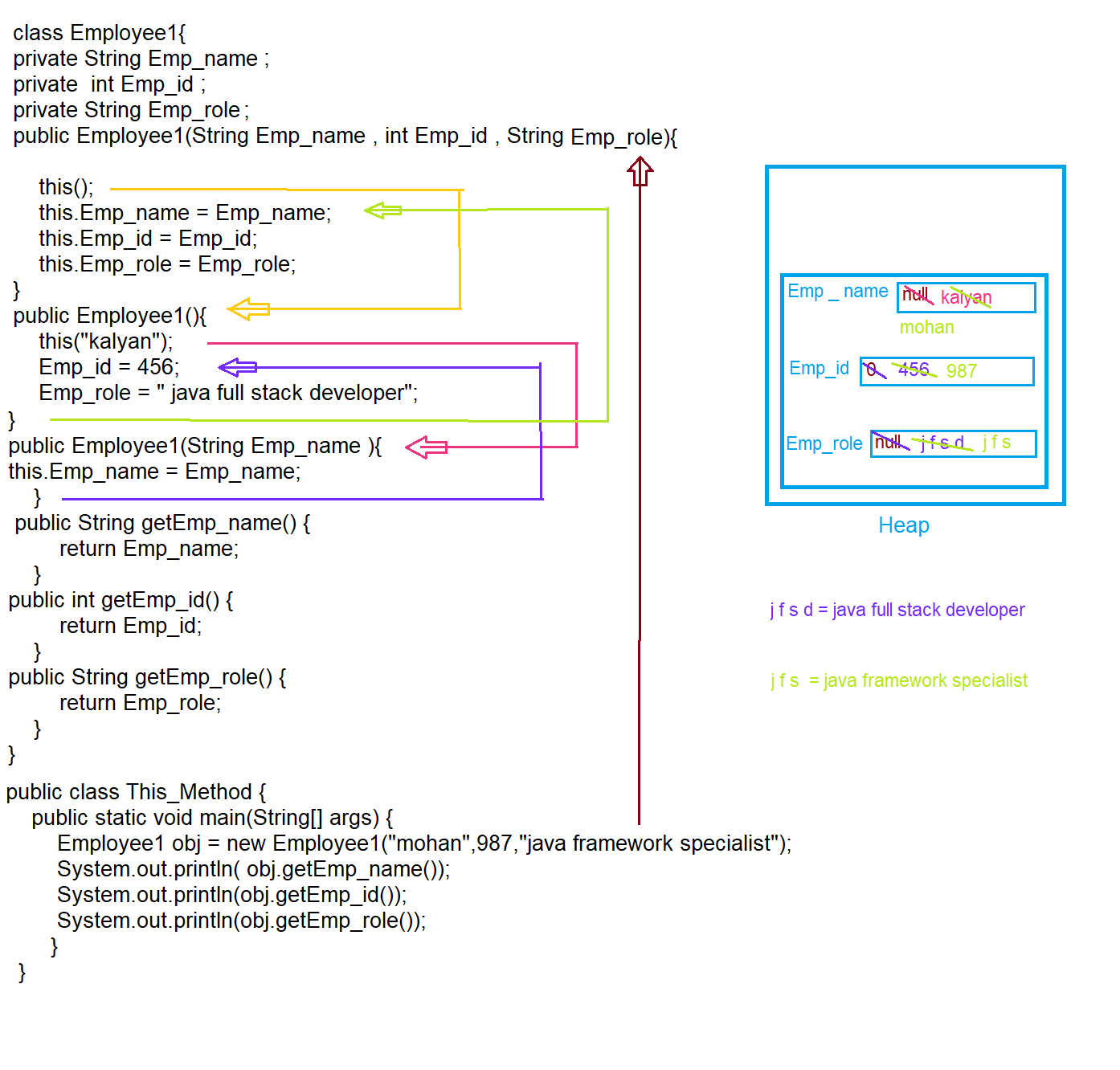
Constructor Chaining:

Calling one constructor from another constructor.

Examples

1. super() ( calls the parent class constructor)
2. this() ( calls same class constructor )

Program flow:



1. First control comes to main method, then object is created in the heap and default values are assigned to the variables according to their data types after that control goes to Employee1 constructor as shown in figure
2. Then this() is called, now this() calls the current class constructor with zero parameters as shown in figure yellow arrow
3. Here again this() calls the constructor that accepts String argument name shown in figure line red
4. now constructor accepting argument String name is executed and name is kalyan is store in the heap .
5. now control goes back to where it has came (violet line ) and remaining lines gets executed default values of Emp\_id , Emp\_role are overridden by 456 and java full stack developer
6. after completing the execution of that constructor the control goes back (green line)
7. now remaining lines in this constructor is executed and values in the heap are overriden by mohan , 987 , java framework specialist
8. now control again comes to main method and remaining code of main method is executed.

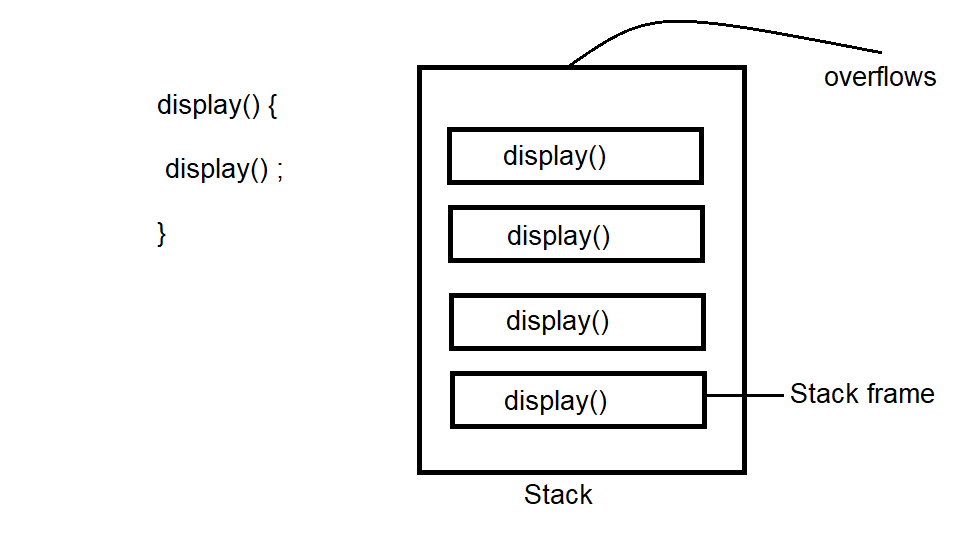
By default super() method is created in 1st line of any constructor, super() in any constructor call the parent class constructor

But this() calls current class constructor , if constructor is created and this() is declared in the first line , an empty constructor should be declared separately for this() method

If this() is declared with arguments , then constructor should be created that accepts the arguments of this()

If this() is created with zero arguments (or) some no of arguments , then that constructor should be declared that accepts same no of parameters . otherwise it leads to compile time error.

Stack Overflow example:



Here display method is brought to stack frame and executed and it is called again and again since recursion, after the complete fill up of the stack frame, it overflows as there is no free space in the stack. it can be said as stack over flow.

Note: Inside a method you cannot create a new method, but you can call another method in that method

Note: you can call another constructor in current constructor using this(), you cannot call constructor with its name in the another constructor. calling constructor with its name is not valid, it is possible only with this() .

Purpose of this() :

Inside one constructor there is a requirement to call or to execute another constructor body in that case we can use this()

|  |  |
| --- | --- |
| this | this() |
| 1. keyword | 1. method |
| 1. refers to current object | 1. refers to same class constructor |

More than one constructor with different parameters can be called as constructor overloading.