***Final keyword in java***

It is used to make a variable as a constant, Restrict method overriding, Restrict inheritance. It is used at variable level, method level and class level. In java language final keyword can be used in following way.

* Final at variable level
* Final at method level
* Final at class level



**Final at variable level**

Final keyword is used to make a variable as a constant. This is similar to const in other language. A variable declared with the final keyword cannot be modified by the program after initialization. This is useful to universal constants, such as "PI".

**Final Keyword in java Example**

**public** **class** Circle {

**public** **static** **final** **double** ***PI*** = 3.14159;

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

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

}

}

**Final at method level**

It makes a method final, meaning that sub classes cannot override this method. The compiler checks and gives an error if you try to override the method.

When we want to restrict overriding, and then make a method as a final.

**Example**

**public** **class** A {

**public** **void** fun1() {

// .......

}

**public** **final** **void** fun2() {

// .......

}

}

**class** B **extends** A {

**public** **void** fun1() {

// .......

}

**public** **void** fun2() {

// it gives an error because we cannot override final method

}

}

**Example of final keyword at method level**

**class** Employee {

**final** **void** disp() {

System.***out***.println("Hello Good Morning");

}

}

**class** Developer **extends** Employee {

**void** disp() {

System.***out***.println("How are you ?");

}

}

**class** FinalDemo {

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

Developer obj = **new** Developer();

obj.disp();

}

}

**Output**

It gives an error

**Final at class level**

It makes a class final, meaning that the class cannot be inheriting by other classes. When we want to restrict inheritance then make class as a final.

**Example**

**public** **final** **class** A {

}

**public** **class** B **extends** A {

// it gives an error, because we cannot inherit final class

}

**Example of final keyword at class level**

**Example**

**final** **class** Employee {

**int** salary = 10000;

}

**class** Developer **extends** Employee {

**void** show() {

System.***out***.println("Hello Good Morning");

}

}

**class** FinalDemo {

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

Developer obj = **new** Developer();

Developer obj = **new** Developer();

obj.show();

}

}

**Output**

It gives an error

***Static keyword in java***

The **static keyword** is used in java mainly for memory management. It is used with variables, methods, blocks and nested class. It is a keyword that are used for share the same variable or method of a given class. This is used for a constant variable or a method that is the same for every instance of a class. The main method of a class is generally labeled static.

No object needs to be created to use static variable or call static methods, just put the class name before the static variable or method to use them. Static method cannot call non-static method.

**In java language static keyword can be used for following**

* variable (also known as class variable)
* method (also known as class method)
* block
* nested class

**Static variable**

If any variable we declared as static is known as static variable.

* Static variable is used for fulfill the common requirement. For Example company name of employees, college name of students etc. Name of the college is common for all students.
* The static variable allocate memory only once in class area at the time of class loading.

**Advantage of static variable**

Using static variable we make our program memory efficient (i.e. it saves memory).

**When and why we use static variable**

Suppose we want to store record of all employee of any company, in this case employee id is unique for every employee but company name is common for all. When we create a static variable as a company name then only once memory is allocated otherwise it allocates a memory space each time for every employee.

**Syntax for declare static variable:**

public static variableName;

**Syntax for declare static method:**

public static void methodName()

{

}

Syntax for access static methods and static variable

**Syntax**

className.variableName=10;

className.methodName();

**Example**

**public** **static** **final** **double** PI=3.1415;

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

{

}

**Difference between static and final keyword**

**static** keyword always fixed the memory that means that will be located only once in the program where as **final** keyword always fixed the value that means it makes variable values constant.

**Note:** As for as real time statement there concern every final variable should be declared the static but there is no compulsion that every static variable declared as final.

**Example of static variable.**

In the below example College\_Name is always same, and it is declared as static.

**Example**

**class** Student {

**int** roll\_no;

String name;

**static** String *College\_Name* = "ITM";

}

**class** StaticDemo {

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

Student s1 = **new** Student();

s1.roll\_no = 100;

s1.name = "abcd";

System.***out***.println(s1.roll\_no);

System.***out***.println(s1.name);

System.***out***.println(Student.*College\_Name*);

Student s2 = **new** Student();

s2.roll\_no = 200;

s2.name = "zyx";

System.***out***.println(s2.roll\_no);

System.***out***.println(s2.name);

System.***out***.println(Student.*College\_Name*);

}

}

Output:

100

abcd

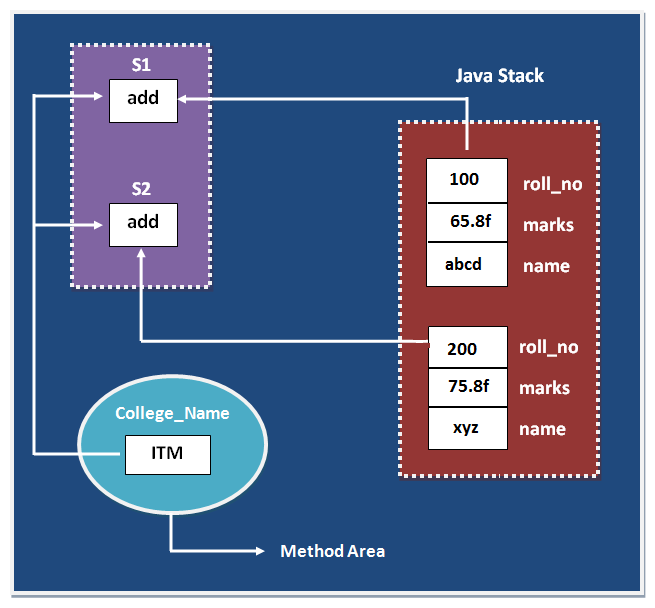
ITM

200

zyx

ITM

In the above example College\_Name variable is commonly sharable by both S1 and S2 objects.



In the above image static data variable are store in method area and non static variable is store in java stack. Read more about this in JVM Architecture chapter

**Why main method is static?**

Because object is not required to call static method if main() is non-static method, then jvm create object first then call main() method due to that face the problem of extra memory allocation.

***This keyword in java***

**this** is a reference variable that refers to the current object. It is a keyword in java language represents current class object

**Usage of this keyword**

* It can be used to refer current class instance variable.
* this() can be used to invoke current class constructor.
* It can be used to invoke current class method (implicitly)
* It can be passed as an argument in the method call.
* It can be passed as argument in the constructor call.
* It can also be used to return the current class instance.

**Why use this keyword in java?**

The main purpose of **using this keyword** is to differentiate the formal parameter and data members of class, whenever the formal parameter and data members of the class are similar then jvm get ambiguity (no clarity between formal parameter and member of the class)

To differentiate between formal parameter and data member of the class, the data member of the class must be preceded by "this".

**"this"** keyword can be use in two ways.

* this. (this dot)
* this() (this off)

**this . (this dot)**

Which can be used to differentiate variable of class and formal parameters of method or constructor?

**"this"** keyword are used for two purpose, they are

* It always points to current class object.
* Whenever the formal parameter and data member of the class are similar and JVM gets an ambiguity (no clarity between formal parameter and data members of the class).

To differentiate between formal parameter and data member of the class, the data members of the class must be preceded by **"this"**.

**Syntax**

this.data member of current class.

**Note:** If any variable is preceded by **"this"** JVM treated that variable as class variable.

**Example without using this keyword**

**class** Employee {

**int** id;

String name;

Employee(**int** id, String name) {

id = id;

name = name;

}

**void** show() {

System.***out***.println(id + " " + name);

}

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

Employee e1 = **new** Employee(111, "Harry");

Employee e2 = **new** Employee(112, "Jacy");

e1.show();

e2.show();

}

}

**Output**

Output:

0 null

0 null

In the above example, parameter (formal arguments) and instance variables are same that is why we are using **"this"** keyword to distinguish between local variable and instance variable.

**Example of this keyword in java**

**class** Employee {

**int** id;

String name;

Employee(**int** id, String name) {

**this**.id = id;

**this**.name = name;

}

**void** show() {

System.***out***.println(id + " " + name);

}

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

Employee e1 = **new** Employee(111, "Harry");

Employee e2 = **new** Employee(112, "Jacy");

e1.show();

e2.show();

}

}

**Output**

111 Harry

112 Jacy

**Note 1:** The scope of **"this"** keyword is within the class.

**Note 2:** The main purpose of using **"this"** keyword in real life application is to differentiate variable of class or formal parameters of methods or constructor (it is highly recommended to use the same variable name either in a class or method and constructor while working with similar objects).

**Difference between this and super keyword**

**Super keyword** is always pointing to base class (scope outside the class) features and **"this"** keyword is always pointing to current class (scope is within the class) features.

**Example when no need of this keyword**

**class** Employee {

**int** id;

String name;

Employee(**int** i, String n) {

id = i;

name = n;

}

**void** show() {

System.***out***.println(id + " " + name);

}

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

Employee e1 = **new** Employee(111, "Harry");

Employee e2 = **new** Employee(112, "Jacy");

e1.show();

e2.show();

}

}

**Output**

111 Harry

112 Jacy

In the above example, no need of use this keyword because parameter (formal arguments) and instance variables are different. This keyword is only use when parameter (formal arguments) and instance variables are same.

**this ()**

which can be used to call one constructor within the another constructor without creation of objects multiple time for the same class.

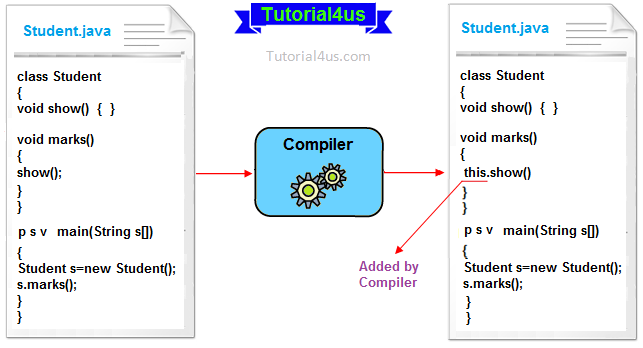
**Syntax**

this(); // call no parametrized or default constructor

this(value1,value2,.....) //call parametrize constructor

**this keyword used to invoke current class method (implicitly)**

By using this keyword you can invoke the method of the current class. If you do not use the keyword, compiler automatically adds this keyword at time of invoking of the method.



**Example of this keyword**

**class** Student {

**void** show() {

System.***out***.println("You got A+");

}

**void** marks() {

**this**.show();//no need to use this here because compiler does it.

}

**void** display() {

show(); // compiler act marks() as this.marks()

}

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

Student s = **new** Student();

s.display();

}

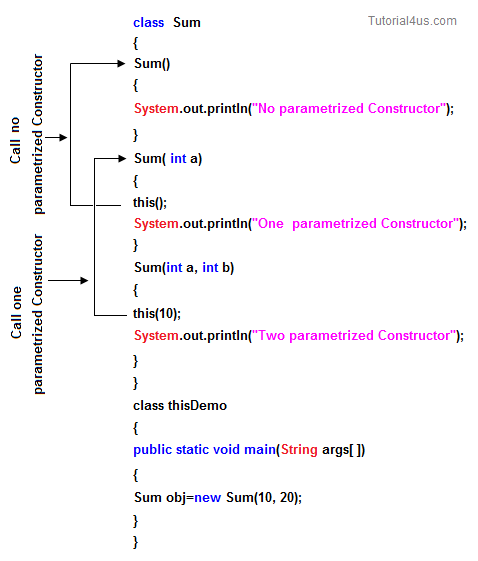
}

**output**

You got A+

**Rules to use this()**

**this()** always should be the first statement of the constructor. One constructor can call only other single constructor at a time by using **this()**.



***Super keyword in java***

**Super** keyword in java is a reference variable that is used to refer parent class object. **Super** is an implicit keyword creates by JVM and supply each and every java program for performing important role in three places.

* At variable level
* At method level
* At constructor level

**Need of super keyword:**

Whenever the derived class is inherits the base class features, there is a possibility that base class features are similar to derived class features and JVM gets an ambiguity. In order to differentiate between base class features and derived class features must be preceded by super keyword.

**Syntax**

super.baseclass features.

**Super at variable level:**

Whenever the derived class inherits base class data members there is a possibility that base class data member are similar to derived class data member and JVM gets an ambiguity.

In order to differentiate between the data member of base class and derived class, in the context of derived class the base class data members must be preceded by super keyword.

**Syntax**

super.baseclass datamember name

if we are not writing super keyword before the base class data member name than it will be referred as current class data member name and base class data member are hidden in the context of derived class.

**Program without using super keyword**

**Example**

**class** Employee {

**float** salary = 10000;

}

**class** HR **extends** Employee {

**float** salary = 20000;

**void** display() {

System.***out***.println("Salary: " + salary);// print current class salary

}

}

**class** Supervarible {

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

HR obj = **new** HR();

obj.display();

}

}

**Output**

Salary: 20000.0

In the above program in Employee and HR class salary is common properties of both class the instance of current or derived class is referred by instance by default but here we want to refer base class instance variable that is why we use super keyword to distinguish between parent or base class instance variable and current or derived class instance variable.

**Program using super keyword al variable level**

**Example**

**class** Employee {

**float** salary = 10000;

}

**class** HR **extends** Employee {

**float** salary = 20000;

**void** display() {

System.***out***.println("Salary: " + **super**.salary);// print base class salary

}

}

**class** Supervarible {

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

HR obj = **new** HR();

obj.display();

}

}

**Output**

Salary: 10000.0

**Super at method level**

The **super keyword** can also be used to invoke or call parent class method. It should be use in case of method overriding. In other word **super keyword** use when base class method name and derived class method name have same name.

**Example of super keyword at method level**

**class** Student {

**void** message() {

System.***out***.println("Good Morning Sir");

}

}

**class** Faculty **extends** Student {

**void** message() {

System.***out***.println("Good Morning Students");

}

**void** display() {

message();// will invoke or call current class message() method

**super**.message();// will invoke or call parent class message() method

}

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

Student s = **new** Student();

s.display();

}

}

**Output**

Good Morning Students

Good Morning Sir

In the above example Student and Faculty both classes have message() method if we call message() method from Student class, it will call the message() method of Student class not of Person class because priority of local is high.

In case there is no method in subclass as parent, there is no need to use super. In the example given below message() method is invoked from Student class but Student class does not have message() method, so you can directly call message() method.

**Program where super is not required**

**Example**

**class** Student {

**void** message() {

System.***out***.println("Good Morning Sir");

}

}

**class** Faculty **extends** Student {

**void** display() {

message();// will invoke or call parent class message() method

}

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

Student s = **new** Student();

s.display();

}

}

**Output**

Good Morning Sir

**Super at constructor level**

The super keyword can also be used to invoke or call the parent class constructor. Constructor are calling from bottom to top and executing from top to bottom.

To establish the connection between base class constructor and derived class constructors JVM provides two implicit methods they are:

* Super()
* Super(...)

**Super()**

**Super()** It is used for calling super class default constructor from the context of derived class constructors.

**Super keyword used to call base class constructor**

**Syntax**

**class** Employee {

Employee() {

System.***out***.println("Employee class Constructor");

}

}

**class** HR **extends** Employee {

HR() {

**super**(); // will invoke or call parent class constructor

System.***out***.println("HR class Constructor");

}

}

**class** Supercons {

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

HR obj = **new** HR();

}

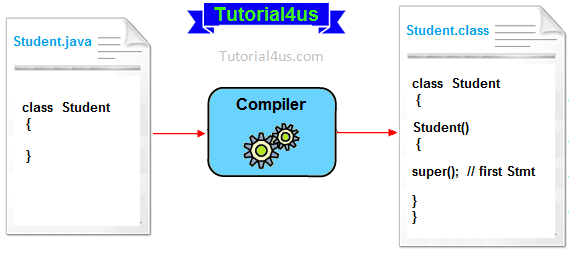
}

**Output**

Employee class Constructor

HR class Constructor

**Note:** super() is added in each class constructor automatically by compiler.



In constructor, default constructor is provided by compiler automatically but it also adds **super()** before the first statement of constructor. If you are creating your own constructor and you do not have either this() or super() as the first statement, compiler will provide super() as the first statement of the constructor.

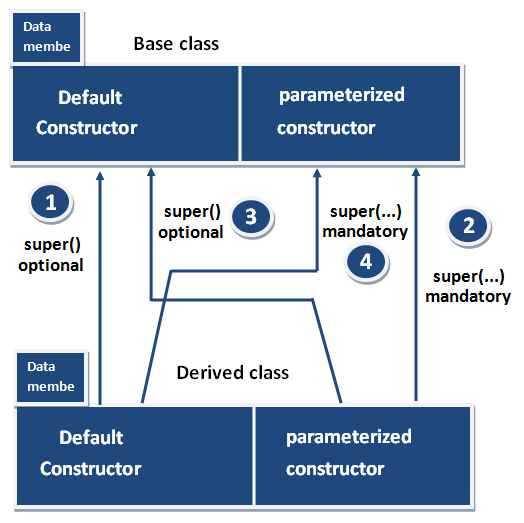
**Super(...)**

**Super(...)** It is used for calling super class parameterize constructor from the context of derived class constructor.

**Important rules**

Whenever we are using either super() or super(...) in the derived class constructors the **super** always must be as a first executable statement in the body of derived class constructor otherwise we get a compile time error.

**The following diagram use possibilities of using super() and super(........)**



**Rule 1 and Rule 3**

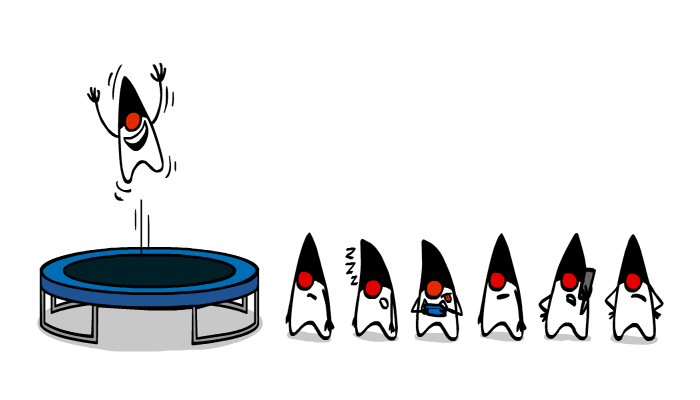
Whenever the derived class constructor want to call default constructor of base class, in the context of derived class constructors we write super(). Which is optional to write because every base class constructor contains single form of default constructor?

**Rule 2 and Rule 4**

Whenever the derived class constructor wants to call parameterized constructor of base class in the context of derived class constructor we must write super (...). Which is mandatory to write because a base class may contain multiple forms of parameterized constructors?

***Synchronized Keyword in Java***

**Synchronized Keyword** is used for when we want to allow only one thread at a time then use Synchronized modifier. If a method or block declared as a Synchronized then at a time only one thread is allowed to operate on the given object.



Synchronized is a Modifier which is applicable for the method or block, we cannot declare class or variable with this modifier.

**Advantage of Synchronized**

The main advantage of Synchronized keyword is we can resolve data inconsistency problem.

**Dis-Advantage of Synchronized**

The main dis-advantage of Synchronized keyword is it increased the waiting time of thread and effect performance of the system, Hence if there is no specific requirement it is never recommended to use synchronized keyword.

**Volatile Keyword in Java**

If the variable keeps on changing such type of variables we have to declare with volatile modifier. Volatile is a modifier applicable only for variables but not for method and class.

If a variable declared as volatile then for every thread a separate local copy will be created. Every intermediate modification performed by that thread will takes place in local copy instead of master copy. Once the value got finalized just before terminating the thread the master copy value will be updated with local stable value.

**Advantage of Volatile**

The main advantage of Volatile -keyword is we can resolve data inconsistency problems.

**Dis-Advantage of Volatile**

The main dis-advantage of Volatile keyword is, creating and maintaining a separate copy for every thread, increases complexity of the programming and effects performance of the system. Hence if there is no specific requirement it is never recommended to use volatile keyword and it is almost outdated keyword.

**Note:** Volatile variable means its value keeps on changing where as final variable means its value never changes. Hence final-Volatile combination is illegal combination for variables.