**Ref : https://anshuchoudhury.wordpress.com/**

**Hello world :**

**public** **class** HelloWorld {

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

System.***out***.println("Hello World");

}

}

**System** is the class in lang package

**Out** I the static reference in the System class

**Println** is the method

-------------------------------------------------------------------------

**Object creation :**

**1 ) Using new :**

Helloworld helloWorld = **new** HelloWorld();

**2 ) Using Class.forName():**

HelloWorld object = (HelloWorld)**Class.forName**(“com.gbn.object.HelloWorld”).newInstace();

**3) Using Clone :**

**Note : class should implement Clonable interface**

HelloWorld **object1**= new HelloWorld();

HellWorld object2 = (HelloWorld) **object1.clone();**

**Cross Question : How to stop cloning of the object ?**

**Ans : override**

**4) Using DeSerialization:**

**class** ReadyToSerialization **implements** Serializable {

**public** **void** display() {

System.***out***.println("Hello some one deserialized me!");

}

}

**public** **class** UsingDeSerialization {

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

UsingDeSerialization obj = **new** UsingDeSerialization();

ReadyToSerialization readyToSerialization = **new** ReadyToSerialization();

obj.serialize(readyToSerialization);

ReadyToSerialization getObject = obj.deSerialize();

getObject.display();

}

**public** **void** serialize(ReadyToSerialization obj) {

**try** {

ReadyToSerialization si = **new** ReadyToSerialization();

FileOutputStream fos = **new** FileOutputStream("ReadyToSerialization.ser");

ObjectOutputStream oos = **new** ObjectOutputStream(fos);

oos.writeObject(si);

oos.close();

fos.close();

} **catch** (Exception e) {

e.printStackTrace();

}

}

**public** ReadyToSerialization deSerialize() {

ReadyToSerialization si = **null**;

**try** {

FileInputStream fis = **new** FileInputStream("ReadyToSerialization.ser");

ObjectInputStream ois = **new** ObjectInputStream(fis);

si = (ReadyToSerialization) ois.readObject();

} **catch** (Exception e) {

e.printStackTrace();

}

**return** si;

}

}

**5)Using factory method :**

**public** **class** UsingFactory {

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

FactoryExample obj = FactoryExample.*getFactoryExample*();

obj.display();

}

}

**class** FactoryExample {

**public** **static** FactoryExample getFactoryExample() {

**return** **new** FactoryExample();

}

**public** **void** display() {

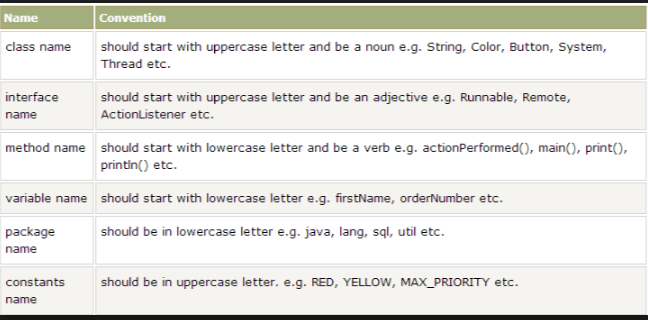
System.***out***.println("object created using factory method !");

}

}

------------------------------------------------------

Naming conventions :



--------------------------------------------------------------------

**Constructors :**

Default Constructor :

**public** **class** DefaultConstructor {

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

Test test = **new** Test();

test.display();

}

}

**class** Test {

/\* Test(){ // By default JVM provides

} \*/

**public** **void** display() {

System.***out***.println("display() called !");

}

}

**Parameterized constructor:**

**public** **class** ParamConstructor {

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

**new** ConstructorEx();

**new** ConstructorEx(1);

}

}

**class** ConstructorEx {

ConstructorEx() {

}

ConstructorEx(**int** a) {

}

ConstructorEx(String s) {

}

ConstructorEx(**double** d) {

}

ConstructorEx(**int** a, String s) {

}

ConstructorEx(Object o) {

}

}

**Constructor flow :**

**public** **class** ConstructorFlow {

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

**new** Four();

}

}

**class** One {

**public** One()

{

**super**();

System.***out***.println("Called One default constructor !");

}

**public** One(**int** a)

{

**this**();

System.***out***.println("Called One parameterized constructor !");

}

}

**class** Two **extends** One {

**public** Two()

{

**super**(1);

System.***out***.println("Called Two default constructor !");

}

**public** Two(**int** a)

{

**this**();

System.***out***.println("Called Two parameterized constructor !");

}

}

**class** Three **extends** Two {

**public** Three()

{

**super**(1);

System.***out***.println("Called Three default constructor !");

}

**public** Three(**int** a)

{

**this**();

System.***out***.println("Called Three parameterized constructor !");

}

}

**class** Four **extends** Three {

**public** Four()

{

**this**(1);

System.***out***.println("Called Four default constructor !");

}

**public** Four(**int** a)

{

**super**(1);

System.***out***.println("Called Four parameterized constructor !");

}

}

----------------------------------------------------------------

**Keywords :**

**Static :**

In Java, a static member is a member of a class that isn’t associated with an instance of a class. Instead, the member belongs to the class itself. As a result, you can access the static member without first creating a class instance.

The static can be:

1. variable (also known as class variable)
2. method (also known as class method)
3. block
4. nested class

**static on variable level:**

in the below example changing the value effecting all the objects

**public** **class** StaticOnVariable {

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

Test t1 = **new** Test();

Test t2 = **new** Test();

Test t3 = **new** Test();

t1.display();

t2.display();

t3.display();

t1.*a* = 20;

//Test.a = 20;

t1.display();

t2.display();

t3.display();

t2.*a* = 30;

//Test.a = 30;

t1.display();

t2.display();

t3.display();

}

}

**class** Test {

**static** **int** *a* = 10;

**public** **void** display() {

System.***out***.println("Value of a : " + *a*);

}

}

**2 )Static on Method level :**

**Without instance creation we can call the static method**

**Becaue they are belongs to class level**

**public** **class** StaticOnMethod {

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

TestStaticOnMethod.*display*();

}

}

**class** TestStaticOnMethod

{

**public** **static** **void** display()

{

System.***out***.println("you called me without creating object !");

}

}

3 )Static on Inner class :

Its not possible to make outer class as static

But we can create static inner class

**public** **class** StaticOnInnerClass {

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

StaticOnInnerClass.TestInnerClass.*display*();

}

**static** **class** TestInnerClass

{

**static** **void** display()

{

System.***out***.println("Static inner class");

}

}

}

Out put :

Static inner class

Counter Example :

**public** **class** CounterEx {

**static** **int** *counter* = 0;

**public** **static** **void** hitCounterEx() {

*counter* += 1;

}

**public** **static** **void** numberOfCounts()

{

System.***out***.println("Number of Hits : "+ *counter*);

}

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

CounterEx.*hitCounterEx*();

CounterEx.*hitCounterEx*();

CounterEx.*hitCounterEx*();

CounterEx.*hitCounterEx*();

CounterEx.*numberOfCounts*();

}

}

Out put : Number of Hits : 4

**Static Block vs instance block :**

Static block executes at the time of class loading

Instance block executes when object creates

**public** **class** StaticBlockVsInitBlock {

**static**{

System.***out***.println("static block");

}

{

System.***out***.println("initialization block");

}

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

**new** StaticBlockVsInitBlock();

}

}

Out put :

static block

initialization block

---------------------------------------------------------------

**Final keyword :**

The final keyword in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

1. variable
2. method
3. class
4. **final on variable level :**

If you make any variable as final, you cannot change the value of final variable(It will be constant).

If we try to change the value , it throw compile time error

**public** **class** FinalOnVariable {

**final** **double** pi = 3.14;

FinalOnVariable(**double** pi)

{

**this**.pi = pi; // compile time error

}

}

1. **final on method level :**

if method is final, then it is not possible to override it

**public** **class** FinalOnMethod {

**public** **final** **void** display()

{

System.***out***.println("display method");

}

}

**public** **class** TestFinal **extends** FinalOnMethod

{

**public** **final** **void** display() // Compile time error

{

System.***out***.println("display method");

}

}

1. **final on class level :**

if class is final , then it is not possible to extend the class

**public** **final** **class** FinalOnClass {

}

**pulic class** TestFinalEx **extends** FinalOnClass // Compile time error

{

}

--------------------------------------------------------------

**instanceOf :**

The **java instanceof operator** is used to test whether the object is an instance of the specified type

**public** **class** InstanceOf {

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

TestInstanceOf obj = **new** TestInstanceOf();

**if**(obj **instanceof** TestInstanceOf)

{

System.***out***.println("This is a object of :TestInstanceOf");

}

**else**

{

System.***out***.println("This is a object of : "+obj.getClass().getName());

}

}

}

**class** TestInstanceOf

{

}

-----------------------------------------------------

**this keyword :**

**1)this on variable level:**

this refers current object variables

in the below example **this.age** refers class instance variables, where **age** refers local variables.

For better understading remove this in the below example and run.

**public** **class** ThisonVariable {

**int** age;

String name;

**public** ThisonVariable(**int** age, String name)

{

**this**.age = age;

**this**.name = name;

}

**public** **void** display()

{

System.***out***.println("name : "+name+" age :"+age);

}

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

**new** ThisonVariable(28, "ram").display();

}

}

Out put: name : ram age :28

If class **instance variable names** and **local variable names** are difference then no need of this.

Ex:

**int** age;

String name;

**public** ThisonVariable(**int** a, String n)

{

age = a;

name = n;

}

this on constructor level:

used to invoke current class constructor from another current class constructor

**public** **class** ThisonConstructor {

String name;

**public** ThisonConstructor() {

**this**("gbn");

}

**public** ThisonConstructor(String name)

{

**this**.name = name;

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

}

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

**new** ThisonConstructor();

}

}

Output : gbn

this on method level :

**public** **class** ThisOnMethod {

**public** **void** display1()

{

System.***out***.println("super class display");

**this**.display2(); // calling display2 method

}

**public** **void** display2()

{

System.***out***.println("display2 method ");

}

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

**new** ThisOnMethod().display1();

}

}

Out put :

super class display

display2 method

**“this” as method argument :**

We can pass this as a current object reference

**public** **class** ThisAsArgument {

**public** **void** display1() {

display2(**this**);

}

**public** **void** display2(ThisAsArgument arg) {

System.***out***.println("dispaly2() called");

}

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

**new** ThisAsArgument().display1();

}

}

Output : display2() called

**Proving this keyword :**

**public** **class** ProvingThis {

**void** m() {

System.***out***.println(**this**);// prints same reference ID

}

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

ProvingThis obj = **new** ProvingThis();

System.***out***.println(obj);// prints the reference ID

obj.m();

}

}

Output :

com.gbn.thisex.ProvingThis@2a139a55

com.gbn.thisex.ProvingThis@2a139a55

-------------------------------------------------------------

**Super keyword :**

**The super keyword in java is a reference variable that is used to refer immediate parent class object.**

**Whenever you create the instance of subclass, an instance of parent class is created implicitly i.e. referred by super reference variable.**

## Usage of java super Keyword

1. **super is used to refer immediate parent class instance variable.**
2. **super() is used to invoke immediate parent class constructor.**
3. **super is used to invoke immediate parent class method.**

super on variable level :

**class** One

{

**int** age = 28;

}

**class** Two **extends** One

{

String name = "gbn";

**int** age = 20;

**public** **void** display()

{

System.***out***.println("name :"+name+" age: "+age);

System.***out***.println("name :"+name+" age: "+**super**.age); // access super age

}

}

**public** **class** SuperOnVariable {

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

**new** Two().display();

}

}

Output :

name :gbn age: 20

name :gbn age: 28

**super on constructor level:**

**class** One {

**in**t age;

**public** One(int age) {

**t**his.age = age;

}

}

**class** Two **extends** One {

**publi**c Two(**int** age) {

**super**(age);

}

**public** **void** display()

{

System.out.println("age called from super : "+**super**.age);

}

}

**public** **class** SuperOnConstructor {

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

**new** Two(28).display();

}

}

Output : age called from super : 28

