# Java made easy

Java developed by a team led by James Gosling at sun microsystem, in beginning it is called as oak and later in a coffee shop he finds indonesian coffee taste well so called Java

## **JDK Java development kit:** Compiler+ libraries

## **JRE Java runtime environment**: Interpreter +JVM

## **JVM Java virtual machine:** abstract machine which execute the byte code ie.java code

## **Memories:**

1.Class(Method) Area

2.Heap: It is the runtime data area in which objects are allocated.

3.Stack

4. Program Counter Register

5. Native Method Stack

## **My First program:**

### How to excute:

|  |  |
| --- | --- |
| javac filename.java  java filename | javac –cp <class path> -d <other class> filename.java  java –cp<classpath> packagename.filename |

import java.Io.\*;//this package used while code in notepad

class Hello{

public static void main(String[]args){

System.out.println(“hii rajesh kanna”); } }

-static have only one copy

-Strings arg[] array is used to pass it to command line

Commanline Arguments: javac filename

### **Input and output:**

System.out.println(); //print only one line and move to next line

System.out.print(); //print line and remain in same line we have to give \n

System.out.printf(“%d”,$i); //print string stream

import java.util.Scanner ;

Scanner input = new Scanner(Sytem.in);

a=input.nextInt();

### **Identifiers:** names that given to class, method, variable

### **Seperators:** {} () [] , ’: ; ::

### **Escape sequences:** \n \t \’ \” \\ \b \f

### **Comments:**1.Singleline: // 2.Multiline: /\*…. \*/ 3.Documentation: /\*\*……\*/

### **Naming conventions**

|  |  |
| --- | --- |
| Name | Convention |
| class name  interface name  method name  variable name  package name  constants | should start with uppercamel letter and be a noun e.g. String, Color, etc  should start with uppercamel letter and be adjective e.g. Runnable, Remote  should start with lowercamel letter and be a verb e.g. actionPerformed(),  should start with lowercase letter e.g.name,age.  should be in lowercase letter e.g. java, lang, sql, util etc.  should be in uppercase letter. e.g. RED, YELLOW, MAX\_PRIORITY etc. |

### **Packages:**

Collection of similar classes, interfaces and sub packages. 1.built-in packages 2.userdefined

import pkg1 [.pkg2].(classname | \*); //import java.io.\*;

how to compile: javac -d directory javafilename.java

how to run: java myppack.javafilename

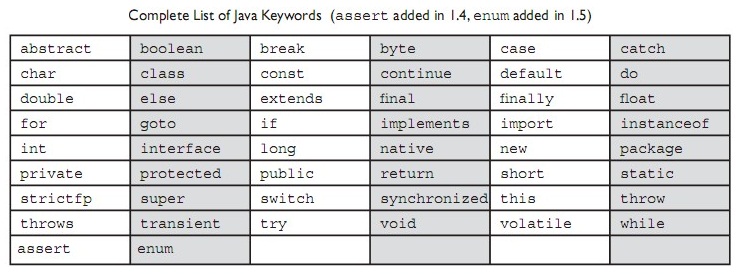
how to decfine a package in that program: package packagename;

how to import package: import packagename.\*;

how to import using fully qulified name without import keyword: package classname;

# **Initial topics:**

## **Keywords:** All predefined meaning words. 50 keywords in Java.(just remember) in addition to there are true, false , null are the keywords so totally 53 sometimes.



## **Constants:** non changeble value by user, in Java it declare by word final, static

## **Variables:** the value that can change, basic unit of storage

### 1.Local vaiable: declare inside a method

### 2.Instance variable: declared inside class but outside the method,

### 3.Static variable: declared as static, not be local. have only one copy

## **Datatype:**

Predefined memory location , storage and variable declaration**,**These are called wrapper classes

|  |  |  |
| --- | --- | --- |
| byte | -128 to 127 | 1 byte |
| short | -32,768 ot 32,768 | 2 byte |
| int | -2,14,74,83,648 to 2,14,74,83,648 | 4 byte |
| long | -9,22,33,72,03,68,54,775 to 9,22,33,72,03,68,54,775 | 8 byte |
| float | 1.4e-045 to 3.4e+038 | 4 byte |
| double | 4.9e-324 to 1.8e+308 | 8 byte |
| char | A-Z a-z 0-9 0-127ASCII | 2 byte |
| boolean | 0 1 | depends on jvm |

### **Typecasting:** way to convert a variable from one data type to another data type

int a = (int) 3.14; //3

Byte->short->int->long->float->double

### **Array:** group of similar datatype variables that are referred to by a common name

int rajesh[]; rajesh=new int[3]; (or) int rajesh[]=new int[3];

Using for loop, arraylenght is get by, rajesh.length

### **String handling:** predefined class file, sequence of char or array of char

|  |  |
| --- | --- |
| str1.charAt(index) | String s3=”rajesh”; sop(s3.charAt(3)); //gives ‘e’ |
| str1.compareTo(str2) | //0 if(s1=s3) //1 if(s1>s3) //-1 if(s3 < s1 ) |
| str1.concat(“string…..”) | String s="Sachin"; s=s.concat(" Tendulkar"); |
| str1.contains(“str2”) | return true or false, if str2 is present in str1 |
| str1.endsWith(“str2”) | return true or false if str1 lastword is str2 |
| str1.equals(str2) | based on content return true or false |
| str1.equalsIgnoreCase(str2) |  |
| String.format(“float %f”,32.433) | it will print float 32.433 |
| str1==str2 | if reference is same return true or false |
| str2=str1.intern() | copy reference of str1 to str2 |
| str1.isEmpty() | if empty returns true |
| Str1=String.join(“hi”,”hello”) | hi-hello join it with delimiter |
| str1.lastIndexOf(‘char’) | return index of char that present last time in the str1 |
| str1.length() | return string length |
| str1.replace(“word ”, “word ”) |  |
| str1.replaceAll(“word”,”word”) |  |
| str1.spilt(“//s”) | spilt the string using whitespace |
| str1.startsWith(‘a’) | return true if str1 start with a |
| str1.substring(2,4) | returns words between index 2 - 3 |
| str1.toCharArray(); | convert string to chararray |
| str1.toLowerCase() |  |
| str1.toUpperCase() |  |
| str1.trim() | omit unnessesary white spaces |
| str1.String.valueOf(variablename) | gives the given value for variable |

toString() If you just print s1, it will print reference of string, so what u need to print , put it inside toString() method, then this method will execute for s1 printing

**String Tokenizer**: allows you to break a string into tokens.

StringTokenizer st = new StringTokenizer("my name is khan"," ");

while (st.hasMoreTokens()) {

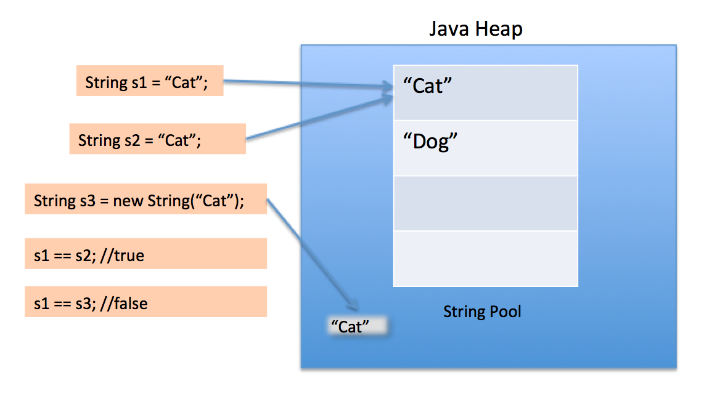
System.out.println(st.nextToken());

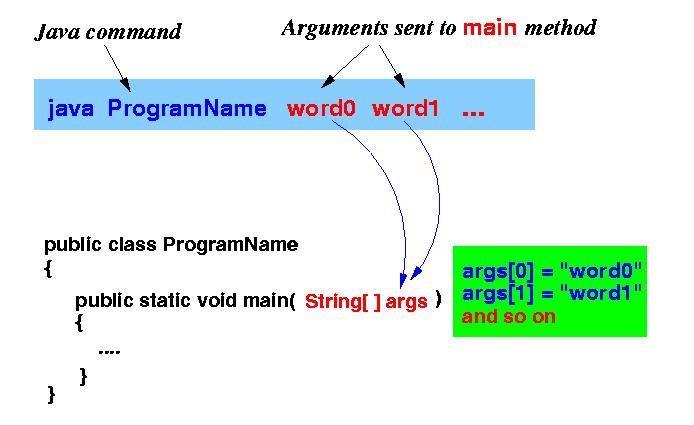
}

**\***u should not use string in loop, each time it create new object in new memory

\*String s1=”raj”; String s2=”raj” both indicates same location no new instance

**\***Command line arguments

****

****

|  |  |
| --- | --- |
| StringBuffer class: | StringBuilder class: |
| class is used to created mutable (modifiable) string. sysnchronized(thread safe).less efficient | class is used to create mutable (modifiable) string. non-synchronized.(not thread safe) more efficient |
| StringBuffer sb=newStringBuffer(“hii”);  sb.append(“hello”)  sb.insert(2,“hello”)  sb.replace(1,3,”java”)  sb.delete(1,3)  sb.reverse() | StringBuilder sb=newStringBuilder(“hii”);  sb.append(“hello”)  sb.insert(2,“hello”)  sb.replace(1,3,”java”)  sb.delete(1,3)  sb.reverse() |

**BufferedReader:**

InputStreamReader isr=new InputStreamReader(System.in);

BufferedReader br=new BufferedReader(isr);

str=br.readLine(); SOP(“hello”+str);

## **Operator:** a symbol tells the compiler to perform operations

1.Arithmetic Operator **+ - \* / %**

2.Relational Operator <, <=, >, >=, ==, !=

3.Logical Operator &&(AND), ||(OR), !(NOT)

4.Assignment Operator a - = 4

5.Inc and Dec Operator a++ a-- ++a --a

6.Conditional Operator exp1? exp2: exp3

7.Bitwise Operator &(AND) !(OR) ^(exclusive OR) ~(1s comp)<<(left shift)

>>(right shift) >>>(unsigned left shift with zero fill)

8.Special Operator

a)member selection operator (.)

b)instanceof (used for downcasting objects)

public static void main(String args[]){

Simple1 s=new Simple1();

System.out.println(s instanceof Simple1);//true

## **Decisions making:**

* 1. **if-** if(condition){exp1;}
  2. **ifelse-**if(condition){exp1;}else{exp2;}
  3. **nested if-** if(condition1){if(condition2){exp1}else{exp2}}else{exp3}}
  4. **tenary operator-**expersion1 ? expersion 1: expersion 2;

## **Looping:**

* 1. **while** while(condition){body;}
  2. **do while** do{body}while(condition)
  3. **for** for(intialise counter;test condition;inc or dec counter){body;}

for(; i<=5; i++); for( ; ; ) {// ...} // infinite loop

for(a=0,b=1;a<3;a++,b--){body} //u can use multiple conditions

* 1. **nesting of for** for(conditions){for(conditions){body}body;}
  2. **switch statement** switch(varible){case constant: operation; break;…..default: operation;break;}

**\***u can combine two cases so make use of it

* 1. **break statement** break; //get out of loop
  2. **continue statement** continue; //back to continue next loop
  3. **continue label** continue loop\_label; loop\_label;{exp}
  4. **goto statement** goto label\_name; label\_name:{exp}
  5. **break goto** break label\_name; label\_name:{exp}
  6. **For each loop(**enhanced for statement**)**

int nums[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 }; for(int x: nums) {SOP(x)}

# **Object Oriented Programming:**

## **Class**: a template or blueprint for object

### 1.member inner class- A.B obj2=obj1.new B(); //A$B.class file will save

### 2.static inner class- A.B obj2 = new A.B();

### 3.Anonymous inner class- A obj=new A(){..code..};

### 4.Lamda expression: A obj=()->{..code…}; obj.show(); //interface A{void show();}

//use only for **functional interface**

## **Objec**t: Instance of class, create memory. three chracteristics: 1.state, 2.behaviour, 3.identity

Class\_name obj= new Class\_name(); //spilt as Class\_name obj; obj= new Class\_name():

LHS = creating references in heap memory

RHS = creating instance as stack memory

### **Anonymous object:** without creating object

//new classname().functioname();

new classname().variablename=value;

if u gonna use object only one times, use it.

**Object cloning:**

1.Shallow copy A obj=new A(); A obj1=obj2; //two reference for one object

2.Deep copy A obj1=obj; obj1.i=obj2.i; obj1.j=obj2.j; //have to copy values seperatel

3.Cloning A.obj1=obj.clone(); //combination of shallow and deep copy

**Garbage Collection:**

automatically recycle of dynamic memory allocation, if no references to an object exist, that object is assumed to be no longer needed

**Finalize method:** It is method used to perform clean up processing just before object is garbage collected i.e destructor of java

gc(); method invoke garbage collector to start cleanup process

protected void finalize( )

{

// finalization code here

}

## **Methods:**has lines of instructions

### **Variable length argument:**

public static void main(String[] args) { System.out.println(function(1,2,3,4,5,3,7,5,3,3,3,3,2,2,3,2)); }

public static int function(int...numbers){ int t=0; for(int x:numbers) t+=x; return t/numbers.length; }}

### **Returning of object:**

class Test { int a;

Test(int i) {a = i;}

Test incrByTen() { Test tempobj = new Test(a+10); return tempobj; }}

class RetOb {

public static void main(String args[]) {

Test ob1 = new Test(2);

Test ob2;

ob2 = ob1.incrByTen();

System.out.println("ob1.a: " + ob1.a+"ob2.a: " + ob2.a);

ob2 = ob2.incrByTen();

System.out.println("ob2.a after second increase: "+ ob2.a);}}

The output: ob1.a: 2 ob2.a: 12 ob2.a after second increase: 22

**Pass by object:**

We always passing a object(pass by reference) in java, becz java has no pointers. And also it need constructor to perform it.

|  |  |
| --- | --- |
| class raji{  int a;  raji(int i){  a=i;}  void disp(raji obj2){  a+=1;}}  public class apple {  public static void main(String[] args) {  raji obj=new raji(5);  System.out.println(obj.a);  obj.disp(obj); System.out.println(obj.a); }} | // Objects may be passed to methods.  class Test {  int a, b;  Test(int i, int j) {  a = i; b = j; }  boolean equalTo(Test o) {  if(o.a == a && o.b == b) return true;  else return false; }}  class PassOb {  public static void main(String args[]) {  Test ob1 = new Test(100, 22);  Test ob2 = new Test(100, 22);  Test ob3 = new Test(-1, -1);  System.out.println("ob1 == ob2: " + ob1.equalTo(ob2));  System.out.println("ob1 == ob3: " + ob1.equalTo(ob3)); }} |

## **Constructor**:

used to intinalize the object, has no return type, have the class name.

It returns value at a instance, it is not inherited, we cant make constructor final

1.Default construtor

2.Parametric constructor

a.with parameter

b.withour parameter

3.copy constructor

Student6 s1=new Student6(111,”karan”);

Student6 s2 = new Student6(s1);

s1.display();

s2.display();

## **Encapsulation:**

Public- Any class any package

private - Specified Class

protected –subsiding class(inheritance)

Default- Specific package

Note: we cannot use private, protected in class, but we can use in innerclass, if u not mention anymodifier it means default modifier(default is different from public, private)

## **Inheritances:** one object or class acquires the properties of another object or class. ie.reusability keyword-extends

if I call a constructor of chileclass, first parent class constructors will execute than only baseclass constrcutor

if I call a method of chileclass, base class override parentclass so first childclass method will execute

**Super:**

1.Whenever subclass need to access the members of parentclass

//super.variablename, super.methodname(); construtor(){super();}

2.Also used as this.operator

class B extends A {

int i; // this i hides the i in A

super.i = a; // i in A

i = b; // i in B }

## **Polymorphism:** ability to define a method(constructor) in many forms

Method overloading

Method overriding

## **Absrtaction:**

### 1. Abstracts class:

abstract class have both abstract and non abstract fields, but abstract method should only be calling not declaring,it is done by overriding. partial abstraction. preffered public

abstract class Shape{

abstract void draw(); }

class Rectangle extends Shape{

void draw(){System.out.println("drawing rectangle");} }

class Circle extends Shape{

void draw(){System.out.println("drawing circle");} }

class TestAbstraction1{

public static void main(String args[]){

Shape s=new Circle(); Shape r=new Rectangle();

s.draw(); r.draw(); }}

### 2. Interfaces:

When you implement an interface method, preferred public

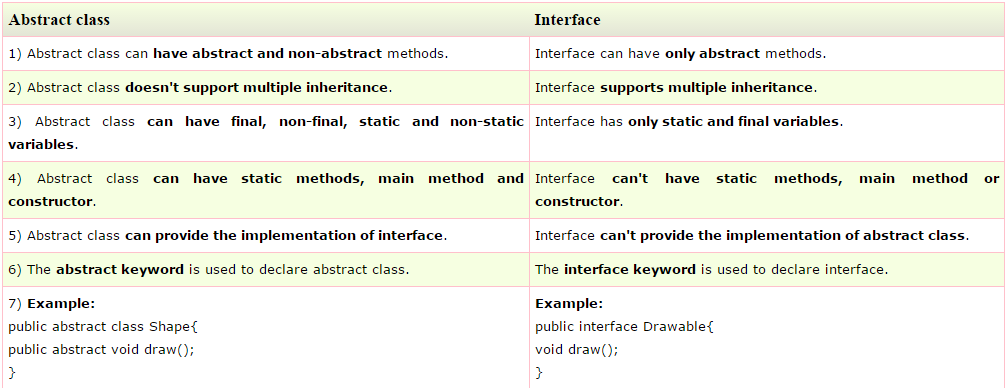
Interface can extends to another interface but class cant extend interface

keywords-interface, implement (c++ we use scope resolution operator to avoid diamond error, but here we use inrefaces instead of multiple inheritance)

3 types: 1.Marker Interface-without any method

2.Single abstract method- having only one abstract method

3.Normal interface



**Static:** used for memory management, have only one copy, not need object to instialize

Note: we cant use static in non static member and also we cant use this and super keyword

**1.static variable** **<**className**>.<**variableName**>;**

**2.static method <**className**>.<**functionName**>**()**;**

**3.static import** import static java.lang.System.out; //hence out is static object, no need classname

**4.static block** static{} //this blocks will execute before main

**5.static class** should be nested class

Finalize:call before unused object is recycled

**Final:** it can assigned only once, make a variable as constant.

Final variable cannot be changed(i.e constant in c++)

Final method cannot be overriden

Final class cannot be extends

**This operator:**When u have variable names same as in method and as in class, using this.variable; u cant point out which variable is to use.(solves confusion between local and global variable)

**Garbage collection:**

garbage means unreferenced objects. process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects.

c language we use free()

c++language we use delete()

it is caused by three ways:

1.while nullfying the object rajesh obj1=new rajesh(); obj1=null;

2.while assigning reference to other obj1=obj2

3.while using anonymous object

public class TestGarbage1{

public void finalize(){System.out.println("object is garbage collected");}

public static void main(String args[]){

TestGarbage1 s1=new TestGarbage1();

TestGarbage1 s2=new TestGarbage1();

s1=null;

s2=null;

System.gc(); } }

output: object is garbage collected

object is garbage collected

# **Other topics:**

## **Exception Handling:** Unexpected event that terminate program

1.Checked

2.Unchecked- compiler cannot found

→we can’t handle exception

Throws:method signature, if anyone called it may throw exception

void funcName ()throws exception{}

try{

…...code………//all try statement exception will goes to catch

throw new exception\_name(“ …this will print along with exception..”);

}

catch(exception e){ ………. }

catch(exception \_name e){…… } //this is multiple catch

catch(arithmeticException|Exception e) {……..} //this is multicatch

finally{……….}// finally will occur for sure even though exception handled or not.

## **Multi threading:**

Feature that allow concurrent execution of two or more parts of the program

Methods: tObj.start()- start thread by calling run method

run()-entry point for thread

tObj.getName()- Get thread’s name

tObj.getPriority()-Get thread priority

tObj.isAlive()- check if thread is running

tObj.join()-wait for thread to terminate

tObj.wait()- notify and wake up thread

Thread.sleep()-suspend thread for period of time

-we can’t use both thread extend and other class extend,

1.Extends Thread class-each of your thread creates unique object and associate with it.

2.Implements Runnable- it shares the same object to multiple threads.

if you will not be overriding any of Thread’s other methods -use implement Runnable.

Also, by implementing Runnable, your thread class does not need to inherit Thread, making it free to inherit a different class.

|  |  |
| --- | --- |
| Implements Runnable interface | Extending Thread class |
| class Multi3 implements Runnable{  public void run(){  System.out.println("thread is running..."); }    public static void main(String args[]){  Multi3 m1=new Multi3();  Thread t1 =new Thread(m1);  t1.start(); } } | class Multi extends Thread{  public void run(){  System.out.println("thread is running..."); }    public static void main(String args[]){  Multi t1=new Multi();  t1.start(); } } |
| ouput:thread is running | output:thread is running |

**sleep method in java:**

class TestSleepMethod1 extends Thread{

public void run(){

for(int i=1;i<5;i++){

try{Thread.sleep(500);} catch(InterruptedException e){System.out.println(e);}

System.out.println(i); } }

public static void main(String args[]){

TestSleepMethod1 t1=new TestSleepMethod1();

TestSleepMethod1 t2=new TestSleepMethod1();

t1.start();

t2.start(); } }

Output:

1122334455

**Join() method –** make wait and join for specified thread

the below line execution wait for t1 thread completion.

t1.join(); //t1.join(milliseconds\_num); //ipadiyum use panalam, joining times

**getName(),setName(String), getId(), currentThread() method:**

class TestJoinMethod3 extends Thread{

public void run(){ System.out.println("running...");

System.out.println(Thread.currentThread().getName()); } //displaying current thread name

public static void main(String args[]){

TestJoinMethod3 t1=new TestJoinMethod3();

TestJoinMethod3 t2=new TestJoinMethod3();

System.out.println("Name of t1:"+t1.getName());

System.out.println("Name of t2:"+t2.getName()); //displaying name of thread

System.out.println("id of t1:"+t1.getId()); //displaying id of thread

t1.start();

t2.start();

t1.setName("Sonoo Jaiswal"); //newname for thread

System.out.println("After changing name of t1:"+t1.getName()); } }

Output:

Name of t1:Thread-0

Name of t2:Thread-1

id of t1: 8

running... Thread-0

After changling name of t1:Sonoo Jaiswal

running... Thread-1

Thread sheduler: part of jvm and decide which thread should run

Two types: 1. Preemptive scheduling – the two thread priority value different, based on priority schdule takes place

2.Time sliced (round robin scheduling)- priority value is same, so it fix certain time for each thread

**Priority to set:**

T1.setPriority();(Thread.MIN\_PRIORITY) T2.setPriority();(Thread.MAX\_PRIORITY)

T1.setDeamon(true); //deamon is thread, when all user thread is dead, just to help the users

To show:

SOP(thread.currentName.getPriority);

SOP(thread.currentName.isDeamon);

program: class TestMultiPriority1 extends Thread{

public void run(){

System.out.println("running thread name is:"+Thread.currentThread().getName());

System.out.println("running thread priority is:"+Thread.currentThread().getPriority()); }

if(Thread.currentThread().isDaemon()){//checking for daemon thread

System.out.println("daemon thread work"); }

public static void main(String args[]){

TestMultiPriority1 m1=new TestMultiPriority1();

TestMultiPriority1 m2=new TestMultiPriority1();

TestMultiPriority1 m3=new TestMultiPriority1();

m1.setPriority(Thread.MIN\_PRIORITY);

m2.setPriority(Thread.MAX\_PRIORITY);

m1.start();

m2.start();

m3.setDaemon(true); //now m3 is daemon thread

} }

Output:

running thread name is:Thread-0

running thread priority is:10

running thread name is:Thread-1running thread priority is:1

**Thread pool:**

a group of fixed size threads are created. A thread from the thread pool is pulled out and assigned a job by the service provider. After completion of the job, thread is contained in the thread pool again.

Reasons for using:

1.instead of using one thread for one task, it prevent time and memory

2. The threads in the pool take tasks off the queue, perform them, then return to the queue for their next task.

3.avoid running out of resources

package practise;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

class WorkerThread implements Runnable {

private String message;

public WorkerThread(String s){

this.message=s; }

public void run() {

System.out.println(Thread.currentThread().getName()+" (Start) message = "+message);

processmessage(); //call processmessage method that sleeps the thread for 2 seconds

System.out.println(Thread.currentThread().getName()+" (End)");//prints thread name }

private void processmessage() {

try { Thread.sleep(2000); } catch (InterruptedException e) { e.printStackTrace(); } } }

public class sample {

public static void main(String[] args) {

ExecutorService executor = Executors.newFixedThreadPool(5); //creating a pool of 5 threads

for (int i = 0; i < 10; i++) {

Runnable worker = new WorkerThread("" + i);

executor.execute(worker); //calling execute method of ExecutorService }

executor.shutdown();

while (!executor.isTerminated()) { } System.out.println("Finished all threads"); } }

**Shutdown hook:**

The shutdown hook can be used to perform cleanup resource or save the state when JVM shuts down normally or abruptly. Performing clean resource means closing log file, sending some alerts or something else. So if you want to execute some code before JVM shuts down, use shutdown hook.

class MyThread extends Thread{

public void run(){

System.out.println("shut down hook task completed..");

}

}

public class TestShutdown1{

public static void main(String[] args)throws Exception {

Runtime r=Runtime.getRuntime();

r.addShutdownHook(new MyThread());

System.out.println("Now main sleeping... press ctrl+c to exit");

try{Thread.sleep(3000);}catch (Exception e) {}

}

}

Output:

Now main sleeping... press ctrl+c to exit

shut down hook task completed..

## **Synchronisation:**

to control multiple threads to share resourse. Before that understand locks. Every object has a lock or moniter, a thread need to acquire lock before accessing

Types:

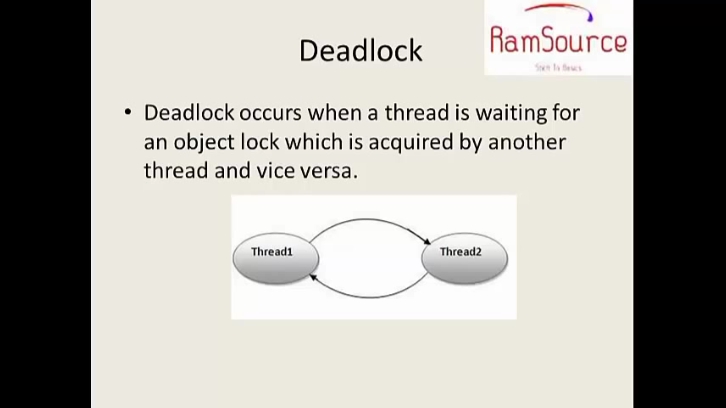
1.Process Synchronization

2.Thread Synchronization-

(i)Mutual Exclusive(not allow interfere)- Synchronized method, block, static synchronization

(ii)Co-operation

|  |  |  |
| --- | --- | --- |
| Synchronized method | Synchroised block | Static synchronization |
| public class synchronization {  public static void main(String[] args) {  Sample s = new Sample();  Mythread1 t1=new Mythread1(s);  Mythread2 t2=new Mythread2(s);  t1.start();  t2.start(); }}  class Mythread1 extends Thread{  Sample s; Mythread1(Samples1){this.s=s1;}  public void run(){ s.compute(5);}}  class Mythread2 extends Thread{  Sample s;  Mythread2(Sample s1){this.s=s1;}  public void run(){s.compute(100);}}  class Sample{  synchronized void compute(int n){  for(int i=0;i<=5;i++){System.out.println(n\*i);}  try{Thread.sleep(500);}catch(Exception e) {System.out.println(e);} } }} | class Sample{  public void compute(int n){  synchronized(this){  for(int i=0;i<=5;i++){  System.out.println(n\*i);  try{Thread.sleep(500);  }catch(Exception e)  {System.out.println(e);} }} | public class synchronization {  public static void main(String[] args) {  Mythread1 t1=new Mythread1();  Mythread2 t2=new Mythread2();  t1.start();  t2.start(); }}  class Mythread1 extends Thread{  public void run(){  sample.compute(5);}}  class Mythread2 extends Thread{  public void run(){  sample.compute(100);}}  class Sample{  synchronized static void compute(int n){  for(int i=0;i<=5;i++){  System.out.println(n\*i);  try{Thread.sleep(500);}catch(Exception e){System. out.println(e);} } }} |

**DeadLock:**occurs two threads waiting for each other

public class deadlock extends Thread {

public static void main(String[] args) {

final String s1="rajesh";

final String s2="mass";

Thread t1= new Thread(){

public void run(){

synchronized(s1){

System.out.println("Thread 1 lock s1");

try{Thread.sleep(1000);}catch(Exception e){System.out.println(e);}

}

synchronized(s2){

System.out.println("Thread 1 lock s2");

try{Thread.sleep(1000);}catch(Exception e){System.out.println(e);}}}};

Thread t2=new Thread(){

public void run(){

synchronized(s2){

System.out.println("Thread 1 lock s2");

try{Thread.sleep(1000);}catch(Exception e){System.out.println(e);}}

synchronized(s1){

System.out.println("Thread 1 lock s1");

try{Thread.sleep(1000);}catch(Exception e){System.out.println(e);}}}};

t1.start();

t2.start(); }}

**Inter thread co operation:**

sync threads communicate with each other by:

1.wait-causes current thread to release lock wait until notify(), notifyall()

2.notify()-wakes up a single thread that waiting for object monter

3.notifyAll()- wakes up all threads that waiting for object monter

class waiting {

public static void main(String[] args) {

final Customer c = new Customer();

Thread t1= new Thread(){

public void run(){

c.withdraw(1000); System.out.println("balance enquiry+c.amt");

}

};

Thread t2= new Thread(){

public void run(){

c.deposit(5000); System.out.println("balance enquiry+c.amt");

}

};

t1.start();

t2.start(); }}

class Customer{

int amt=0;

int f=0;

synchronized int withdraw(int amount){

System.out.println(Thread.currentThread().getName()+"is going to withdraw");

if(f==0){ try {System.out.print("waiting”");

wait();}

catch(InterruptedException e){}}

this.amt=amount;

System.out.println("amount withdrawn"+amount);

return amt;

}

synchronized void deposit(int amount){

System.out.println(Thread.currentThread().getName()+"is going to deposit");

this.amt=amount;

notifyAll();

System.out.println("amount deposit" + amount);

}}

Interrupting thread:

if any thread in sleeping or waiting state, calling interrupt make them breaks out sleep and wait state. if there is not sleep or wait state thread, the program behaves normally

add this inside the program

t1.start();

t1.interrupt();

t2.start();

## **Collections:**

dynamic type of array that can perform searching, sorting, insertion, manipulation, deletion etc.

map list set queue

**ArrayList:**

\*Dynamic array for sorting

\*It extends AbstractList class and implements List interface

\* can contain duplicate

\* maintain insertion order

\* non synchronized so slow

\* lot of shifing needs

\*better for store and accessing data

ArrayList al=new ArrayList();//creating old non-generic arraylist

ArrayList<String> al=new ArrayList<String>();//creating new generic arraylist

import java.util.\*;

class TestCollection1{

public static void main(String args[]){

ArrayList<String> al=new ArrayList<String>();//creating arraylist

al.add("Vijay"); al.add("Ravi"); al.add("Ajay"); al.add("Ravi"); //adding object in arraylist

al.remove(“ravi”); al.remove(2); //remove object in arraylist

ArrayList<String> al2=new ArrayList<String>();

al2.add("Sonoo"); al2.add("Hanumat");

a1.addAll(a2); //join a1 and a2

al.removeAll(a2); //remove similar

al.retainAll(a2); //have only similar

Long variable\_name=System.CurrentTimeMillis(); //print millisecound taken from jan 1 1970 12am

Long start=System.CurrentTimeMillis(); //start time

Long stop=System.CurrentTimeMillis(); // end time

System.out,println(end-start); //print time taken

Iterator itr=al.iterator();//getting Iterator from arraylist to traverse elements

while(itr.hasNext()){

System.out.println(itr.next()); } } }

//for loop

for(int i:al)

SOP(i);

**LinkedList:**

\*doubly linked list to store elements

\*It extends the AbstractList class and implements List and Deque interfaces.

\*maintains insertion order.

\*non synchronized.

\*no shifting needs so fast

\*can be used as list, stack or queue.

\*better for implementing data

|  |  |
| --- | --- |
| **Commonly used methods of List Interface:** | **Commonly used methods of ListIterator Interface:** |
| public void add(int index,Object element);  public boolean addAll(int index,Collection c);  public object get(int Index position); //display particular value  public object set(int index,Object element);  public object remove(int index);  public ListIterator listIterator();  public ListIterator listIterator(int i); | public boolean hasNext();  public Object next();  public boolean hasPrevious();  public Object previous(); |

**HashSet:**

\*uses hashtable to store the elements

\*It extends AbstractSet class and implements Set interface.

\*contains unique elements only.

**LinkedHashSet class**

\*contains unique elements only like HashSet

\*It extends HashSet class and implements Set interface.

\*maintains insertion order.

**TreeSet class**

\*contains unique elements only like HashSet.

\*implements NavigableSet interface that extends the SortedSet interface.

\*maintains ascending order.

**Queue:**  is FIFO manner

**Queue Interface methods :**

public boolean a1.add(object);

public boolean a1.offer(object);

public a1.element();

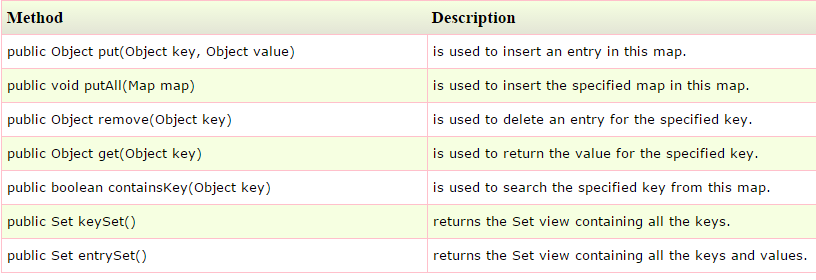
public a1.peek();

public a1.remove();

public a1.poll();

a1.size();

**HashMap:**contains key along with value



import java.util.\*;

class TestCollection13{

public static void main(String args[]){

HashMap<Integer,String> hm=new HashMap<Integer,String>();

hm.put(100,"Amit");

hm.put(101,"Vijay");

hm.put(102,"Rahul");

for(Map.Entry m:hm.entrySet()){

System.out.println(m.getKey()+" "+m.getValue()); } } }

// Set set=hm.entrySet();

Iterator i=set.iterator();

while(i.hasNext()){

Map.Entry mo=(Map.Entry)i.next();

System.out.print(mo.getKey()+":");

System.out.print(mo.getValue()+"\n");}}}}

**LinkedHashMap class**

\*contains values based on the key

\*It implements the Map interface and extends HashMap class.

\*contains only unique elements.

\*have one null key and multiple null values.

\*same as HashMap instead maintains insertion order.

**TreeMap class**

\*contains values based on the key

\* implements the NavigableMap interface and extends AbstractMap class.

\*contains only unique elements.

\*cannot have null key but can have multiple null values.

\*same as HashMap instead maintains ascending order.

**Hashtable class**

\*array of list.Each list is known as a bucket.The position of bucket is identified by calling the hashcode() method

\*contains values based on the key. It implements the Map interface and extends Dictionary class.

\*contains only unique elements.

\*may have not have any null key or value.

\* synchronized.

\*slow when compare with hashmap

Collection.sort(a1); //arranged

**Comparable:**

|  |  |
| --- | --- |
| import java.util.\*;  import java.io.\*;  public class TestSort3{  public static void main(String args[]){  ArrayList<Student> al=new ArrayList<Student>();  al.add(new Student(101,"Vijay",23));  al.add(new Student(106,"Ajay",27));  al.add(new Student(105,"Jai",21));    Collections.sort(al,);  for(Student st:al){  System.out.println(st.rollno+" "+st.name+" "+st.age);  }  }  } | class Student implements Comparable<Student>{  int rollno;  String name;  int age;  Student(int rollno,String name,int age){  this.rollno=rollno;  this.name=name;  this.age=age;  }    public int compareTo(Object obj){  Student st=(Student)obj;  if(age==st.age)  return 0;  else if(age>st.age)  return 1;  else  return -1;  }  } |

**Comparator:**

|  |  |
| --- | --- |
| import java.util.\*; import java.io.\*;  class Simple{ public static void main(String args[]){    ArrayList al=new ArrayList();  al.add(new Student(101,"Vijay",23));  al.add(new Student(106,"Ajay",27));  al.add(new Student(105,"Jai",21));    System.out.println("Sorting by Name...");  Collections.sort(al,new NameComparator());  Iterator itr=al.iterator();  while(itr.hasNext()){  Student st=(Student)itr.next();  System.out.println(st.rollno+" "+st.name+" "+st.age); }    System.out.println("sorting by age...");  Collections.sort(al,new AgeComparator());  Iterator itr2=al.iterator();  while(itr2.hasNext()){  Student st=(Student)itr2.next();  System.out.println(st.rollno+" "+st.name+" "+st.age); } } } | class Student{ int rollno; String name; int age;  Student(int rollno,String name,int age){  this.rollno=rollno; this.name=name; this.age=age; } } |
| import java.util.\*;  class AgeComparator implements Comparator{  public int compare(Object o1,Object o2){  Student s1=(Student)o1;  Student s2=(Student)o2;    if(s1.age==s2.age) return 0;  else if(s1.age>s2.age) return 1;  else return -1; } } |
| import java.util.\*;  class NameComparator implements Comparator{  public int compare(Object o1,Object o2){  Student s1=(Student)o1;  Student s2=(Student)o2;  return s1.name.compareTo(s2.name); } } |

### **Wrapper class** provides mechanism to convert primitive into object. object into primitive.

**ex.**int-Integer. float-Float. double-Double. char-Character. byte-Byte. short-Short. long-Long

### **AutoBoxing:**compiler automatically convert primitive types to object of wrapper classes

### **Unboxing:** compiler converts object of wrapper classes to primitive types

public class Autoboxing {

public static void main(String[] args) {

int intValue=50;

Integer integerObj= intValue;

System.out.println(integerObj);

display(intValue); //boxing or autoboxing

Integer integerObj2=new Integer(100);

int intValue2=integerObj2;

System.out.println(intValue2);

display2(integerObj2); }

public static void display(Integer integerObj){

System.out.println(integerObj); }

public static void display2(int intValue2){

System.out.println(intValue2); }}

**Properties:**

**Vector:**

## **IO stream:**

Stream- sequence of data 1.system.in 2.system.out 3.system.err

Byte- read byte by byte- photo, videos- ASCII 7 bits 2^7=128 1.input 2.OutputStream

Char- read char by char- text files- Unicode -16 bits 2^16=65536 1.Reader 2.writer

Types in IO java:

1.FileInput/ OutputStream- If you have to write primitive values then use FileOutputStream

2.ByteArrayInput/OutputStream- holds a copy of data and forwards it to multiple streams.

3.FilterInput/OutputStream

a.DataInput/OutputStream

b.BufferedInput/outputStream- use buffer mechanism(buffer to store data not stream)

c.PushbackInput/OutputStream

4.PipedInput/OutputStream

5.ObjectIniput/OuputStream

6.SequenceInputStream

|  |  |
| --- | --- |
| **FileOutputStream FileInputStream**  If you have to write primitive values then use FileOutputStream | **ByteArrayInputStream OutputStream**  holds a copy of data and forwards it to multiple streams. |
| import java.io.\*;  class Customer{  public static void main(String args[]){ try{  FileOutputStream fot=new FileOutputStream("demo.txt");  int a=10; byte b=(byte)a;  fot.write(b);  FileInputStream fin=new FileInputStream("demo.txt");  int i; while((i=fin.read())!=-1)  System.out.println((int)i);  fot.write(i);  fot.close();fin.close();  } catch(Exception e){  System.out.println(e);}}} | import java.io.\*;  public class bytestr {  public static void main(String[] args) throws IOException {  byte vowel[]={'a','e','i','o','u'};  ByteArrayInputStream bai=new ByteArrayInputStream(vowel);  int temp;  while ((temp=bai.read())!=-1){  char ch=(char)temp;  System.out.println(ch+" ascii value: "+temp); }  System.out.println("\n \n to read again after calling reset()");  bai.reset();  while((temp=bai.read())!=-1){  char ch=(char)temp;  System.out.println(ch+" ascii value: "+temp);}  bai.close();    ByteArrayOutputStream bao= new ByteArrayOutputStream();  bao.write(vowel);  System.o.p("Getting all the bytes from stream "+bao.toString());  bao.close();    ByteArrayOutputStream bao1=new ByteArrayOutputStream();  bao1.write(vowel,0,3);  System.o.p ("Getting all bytes from 4 bytes"+bao1.toString());  bao1.close(); }} |
| Output:  10 //only print below 256 becz it is byte | Output: a ascii value: 97  e ascii value: 101  i ascii value: 105  o ascii value: 111  u ascii value: 117    to read again after calling reset()  a ascii value: 97  e ascii value: 101  i ascii value: 105  o ascii value: 111  u ascii value: 117  Getting all the bytes from stream aeiou  Getting all bytes from 4 bytes aei |

|  |
| --- |
| **Buffered input and output stream-**  use buffer mechanism(internal buffer to store data, instead of write directly into a stream)  more efficincy, fast. |
| import java.io.\*; public class buffer {  public static void main(String[] args) {try{  FileOutputStream fout =new FileOutputStream("file.txt");  BufferedOutputStream bout=new BufferedOutputStream(fout);  String s="Hello rajesh";  byte b[] =s.getBytes();  bout.write(b);  bout.flush();  bout.close();    FileInputStream fin =new FileInputStream("file.txt");  BufferedInputStream bin =new BufferedInputStream(fin);  int i; while((i= bin.read())!=-1)System.out.print((char)i);fin.close();}  catch(Exception e){ System.out.println(e); }}} |

**File handling:**

|  |  |
| --- | --- |
| **Writing in file:** | **Reading file:** |
| import java.io.\*;  class Test{  public static void main(String args[]){  try{  FileOutputstream fout=new FileOutputStream("abc.txt");  String s="Sachin Tendulkar is my favourite player";  byte b[]=s.getBytes();//converting string into byte array  fout.write(b);  fout.close();  System.out.println("success...");  }catch(Exception e){system.out.println(e);} }} | import java.io.\*;  class SimpleRead{  public static void main(String args[]){  try{  FileInputStream fin=new FileInputStream("abc.txt");  int i=0;  while((i=fin.read())!=-1){  System.out.println((char)i);  }  fin.close();  }catch(Exception e){system.out.println(e);} } } |

|  |
| --- |
| **SequenceInputStream**  class is used to read data from multiple files |
| import java.io.\*;  class Simple{  public static void main(String args[])throws Exception{  FileInputStream fin1=new FileInputStream("f1.txt");  FileInputStream fin2=new FileInputStream("f2.txt");    SequenceInputStream sis=new SequenceInputStream(fin1,fin2);  int i;  while((i=sis.read())!=-1){  System.out.println((char)i);  }  sis.close();  fin1.close();  fin2.close(); } } |

|  |  |
| --- | --- |
| **FileWriter class-**  recommended when read write operation done in text files | F**ileReader** |
| import java.io.\*;  class Simple{  public static void main(String args[]){  try{  FileWriter fw=new FileWriter("abc.txt");  fw.write("my name is sachin");  fw.close();  }catch(Exception e){System.out.println(e);}  System.out.println("success"); } } | import java.io.\*;  class Simple{  public static void main(String args[])throws Exception{  FileReader fr=new FileReader("abc.txt");  int i;  while((i=fr.read())!=-1)  System.out.println((char)i);    fr.close(); } } |

|  |
| --- |
| **CharArrayWriter class:**  can be used to write data to multiple files. Its buffer automatically grows when data is written in this stream. Calling the close() method on this object has no effect. |
| import java.io.\*;  class Simple{  public static void main(String args[])throws Exception{  CharArrayWriter out=new CharArrayWriter();  out.write("my name is");  FileWriter f1=new FileWriter("a.txt"); FileWriter f2=new FileWriter("b.txt");  FileWriter f3=new FileWriter("c.txt"); FileWriter f4=new FileWriter("d.txt");  out.writeTo(f1); out.writeTo(f2);  out.writeTo(f3); out.writeTo(f4);  f1.close(); f2.close();  f3.close(); f4.close(); } } |

**Reading data from keyboard**

|  |  |
| --- | --- |
| Using InputStreamReader class | Using console class |
| import java.io.\*;  class G5{  public static void main(String args[])throws Exception{    InputStreamReader r=new InputStreamReader(System.in);  BufferedReader br=new BufferedReader(r);    String name="";    while(!name.equals("stop")){  System.out.println("Enter data: ");  name=br.readLine();  System.out.println("data is: "+name);  }  br.close(); r.close(); } } | import java.io.\*;  class ReadStringTest{  public static void main(String args[]){  Console c=System.console();  System.out.println("Enter your name: ");  String n=c.readLine();  System.out.println("Enter password: ");  char[] ch=c.readPassword();  System.out.println("Welcome "+n);  System.out.println("Password is: "+pass);  }  } |
| Output:Enter data: Amit  data is: Amit  Enter data: 10  data is: 10  Enter data: stop  data is: stop | Output:  Enter your name: rajesh  welcome rajesh  Enter your password \*\*\*\*  password is raje |

|  |
| --- |
| **PipedInputStream and PipedOutputStream classes**  The PipedInputStream and PipedOutputStream classes can be used to read and write data simultaneously. Both streams are connected with each other using the connect() method of the PipedOutputStream class. |
| import java.io.\*;  class PipedWR{  public static void main(String args[])throws Exception{    final PipedOutputStream pout=new PipedOutputStream();  final PipedInputStream pin=new PipedInputStream();  pout.connect(pin); //connecting the streams  Thread t1=new Thread(){ //creating one thread t1 which writes the data  public void run(){  for(int i=65;i<=90;i++){  try{  pout.write(i);  Thread.sleep(1000);  }catch(Exception e){} } } };    Thread t2=new Thread(){ //creating another thread t2 which reads the data  public void run(){  try{  for(int i=65;i<=90;i++)  System.out.println(pin.read());  }catch(Exception e){} } };  t1.start(); //starting both threads  t2.start(); }} |

**Serializatoin:**

mechanism of writing the state of an object into a byte stream.

1.Object serialization:

Import java.io.\*;

Psvm(){

Save obj=new save();

Obj.i=4;

File f= new File(“demo.txt”);

FileOutputStream fos= new FileOutputStream(f);

ObjectOutputStream oos=new ObejctOutputStream(fos);

Oos.writeUTF(obj);

FileInputStream fis= new FileInputStream(f); //deserialization

ObjectInputStream ois=new ObejctInputStream(fis);

Save obj1=(save) ois.readobject();

System.out.println(obj.i); }

**Accessing private by reflection API:**

|  |  |
| --- | --- |
| Import java.lang.reflect.Method;  Public class ReflectionDemo  {Public stati void main (String[] args) throws Exception  { Class c=Classs.forName(“com.navin.Test”);  Test t= (Test)c.newInstance();  Method m=c.getDeclareMethod(“show”, null);/\*name of method,parameter\*/  m.setAcessible(true);  m.invoke(t.null); }} | Package.com.navin;  Public class Test  {  Private void show()  {  private void show()  { System.out.println(“In show hiii hellow”);  }} |

# Special stuffs:

**1.Random class:**

import java.util.Random;

public class welcme {

public static void main(String[]args){

**Random dice=new Random();**

int a;

a=1+**dice.nextInt(6)**;

System.out.print(a+" ");}}}

**2.Math Class Method:**

int raji=Math.max(2,6); //6

int raji=Math.min(2,6); //2

int raji=Math.sqrt(25);

int raji=Math.pow(3,2);

int b = Math.abs(-20); // absolute=20

double c = Math.ceil(7.342); // 8.0

double f = Math.floor(7.343); // 7.0

equals(1)

3.Enums- members of fixed class //enum animals{raj, cat, dog}

4.How many object used in a program

|  |  |
| --- | --- |
| public static void main(String[] args){  A obj1= new A();  A obj2= new A();  Obj2.counter();}} | class A{ int i;  A(){ System.out.println("hii");  i++; }  void counter() { System.out.println(i); } |

5.static block is a block, the statements inside it will execute instead for public static void main fucntion.//Static{….code …….}

6.static import: import static packagename.classname; u can variable by sample.variableaname; or methods

If u give import java.lang.System.out; u just give out.print(); to print anything no need of SOP

8.Class c= class.forname(“com.rajesh,A”); System.out.println(c.isInterface());//will show true or false if that class is interface

9. Class c= class.forname(“com.rajesh,A”); System.out.println(c.getsuperclass()); //will print the name of superclass

10.array\_name.length -gives the length of array

11.stck.length – gives length of stack

12.Singleton design: 1.static class\_name object\_name=new class\_name();

2.make default constructor private private:class\_name(){}

3.static object\_name method\_name(){return object\_name;}

|  |  |
| --- | --- |
| Class sample{ static sample obj;  private:sample(){}  public:static obj getinstance(){}}  psvp(){  sample obj=sample.getinstance(){}} | Lazy instantiation:  Class sample{ static sample obj=new sample();  private:sample(){}  public: static obj getinstance(){if(obj=null){obj=new sample();}}  psvp(){  sample obj=sample.getinstance();} //if we use thread this method will call two instance |
| //using synchronized in method  Class sample{ static sample obj=new sample();  private:sample(){}  public: static obj synchronised getinstance(){if(obj=null){obj=new sample();}}}  psvp(){  Thread t1=new Thread(new Runnable()){public void run(){sample obj=sample.getinstance();}};  Thread t2=new Thread(new Runnable()){public void run(){sample obj=sample.getinstance();}};  sample obj=sample.getinstance();}  t1.start();  t2.start(): | //using synchronized inside method called double check locking  Class sample{ static sample obj=new sample();  private:sample(){}  public: static obj getinstance(){if(obj==null){synchronized(sample.class)  {if(obj==null){obj=new sample();}} }}}  psvp(){  Thread t1=new Thread(new Runnable()){public void run(){sample obj=sample.getinstance();}}; |
| //work above java5  enum sample{  INSTANCE:  int i;  public void show(){sop(“i”);}}  main(){sample obj=sample.INSTANCE;  obj.i=9;  obj.show();  sample obj=sample.INSTANCE;  obj.i=8;  obj.show():}  output: 9 8 not 9 9 |  |

**13.Assertion:** if condition true it allows normal flow, if condition false, it through exception

syntax: assertion condition;

**14.Deprecated()** function destroy thread without any cleanup. oracle will remove it in next version

15.Transceint- transient keyword is used along with instance variables to prevent it from serialization process

16.Volatile- On the other hand volatile variables are used in Concurrent programming in Java. When we declare a variable volatile, every thread reads its value from main memory and don't used cached value available in every thread stack. volatile variable also prevents compiler from doing reordering which can compromise synchronization. to learn more about volatile variables, read What is volatile variable in Java.

15.Hashcode: int hashcode(float num); //return hashcode for num jdk 8