**Java**:

IT : non IT:

Computer: system: binary: 0,1::::::::0000001111110000000: Hello All: 00000011111

COBOL, PASCAL, FORTON, FOXPRO::::

Categories: Special purpose & General:

Low Level programming : basically in 0 ‘s and 1’s: : -----

Middle level /Assembly: ------- 00001111: English: Store, Register…load…

**High level language**: : these languages: code is exactly to English:

PASCAL:

C Programming: Structural languages: /Function-Data Languages:

Data growing: main focus functions: data:

**Function** were active holder: **data** passive: add(10,20);

**void add(int a, int b){**

**Int c=a+b;**

**Printf(c);**

**}**

**1990: OOPL: C++, Java:**

**Object:**

Function: data:

Data:

How we can make our Data more secure:

Technology: object: object is a basic building block of ur system:

You have to construct an application for Library management system: function:

Real world object: librarian: books, students, professor: distributors:

Convert all these real world objects in to the soft objects;

class Librarian{

id, name, salary, age…..

Procuring book, issuingbooks, return(), calculatingfine()

}

Librarian: id, name, salary, age….

Procuring book, issuingbooks, return(), calculatingfine()

Librarian :

Mechanical: Programming:

Low: 0000111: computer system: No translator:

Middle : LOAD, STORE, : **Assembler**: machine level language: CS:

High: :::: English Like: Compiler and Interpreters: : machine language: compiler or it can have interpreter: either or:

Any programing:

class Employee{

}

Execute: compiler / Interpreters : compile: I ll convert/ translate that code from source code to Machine code:

{

State1; machine code

State2; machine code

Stat10;

}

At a time to machine code: if there is no error: mistake: highlight all the mistakes: 3 displayed:

Translators : Indian: Hindi, Marathi: English ::::: English : translator: Japanese: Japan:

Features of oo: function data languages:

1. **Inheritance**: reusability of the code: Process in which a class/object can acquiring the properties of another class:

**Inheriting** : hair color, skin color, face str, behavior

**Shape**: area: perimeter:

Rectangle: Circle.. square

1. Polymorphism: : area(){ area=pi\*r\*r; }// circle
   * 1. area({area=side\*side; }// squre:
     2. + ::: 10 + 10=20: Addition of 2 nos
     3. +: Simpli + Learn: Concatenation: SimpliLearn
2. Data Abstraction: ATM: With, ChangePIN, MiniStatements… logs , maintain ATM: ADMIN:

Order meal: 15 - 20: meal: enjoy: taste : abstraction: getMoney(){ }

Data hiding:

1. Encapsulation: wrapping up of the data: capsules: ingr: insulation:

Translator: machine:

What is java?: OOPL: pure OOPL: not 100% OOPL: **Primitive DT**: Do not have direct Multiple inheritance:

High-level oopl, **robust**: OOPL: chance of error is very less: Abc.java: compile: javac Abc.java: : Abc.class

Both : Compiler and interpreter: byte code: interpret that code : output:

History of Java:1991: ;;;; 1995 James Gostling known as a father of java: Green team: Sun microsystem: green team: green talk: OAK : JAVA:

Features of Java:

1. Simple:
2. OO: pure
3. Portable: compile: JVM: .class:
4. Secured:
5. Robust:
6. Architectural Neutral: soft/hw:
7. Interpreted:
8. Platform independent: hw sw
9. Multi-threading: T1:
10. Dynamic:
11. Distributed: client- server:

**JVM architecture: queries: JVM:**

**JVM: java virtual machine: heart and brain of java: portable: WORA: write once and run anywhere:**

**Virtual machine :**

**P3… p4::::: windows98…. xp: execute on linux: 128 mb 16 gb: physical: 2 dual OS:**

**Windows: Virtaul Box: Linux: illusion linux: windows: VM: .exe…. .txt. doc::: linux:**

**7-8 years: desktop: android newly : games.. application APK: windows xp: : bluestack : illusion of Android OS; on our computers: whats app… games:**

**VM:**

**JVM: .class: converting in to machine code:**

**C, C++ : compiled: transalated in to platform specific machine code: compiled languges:**

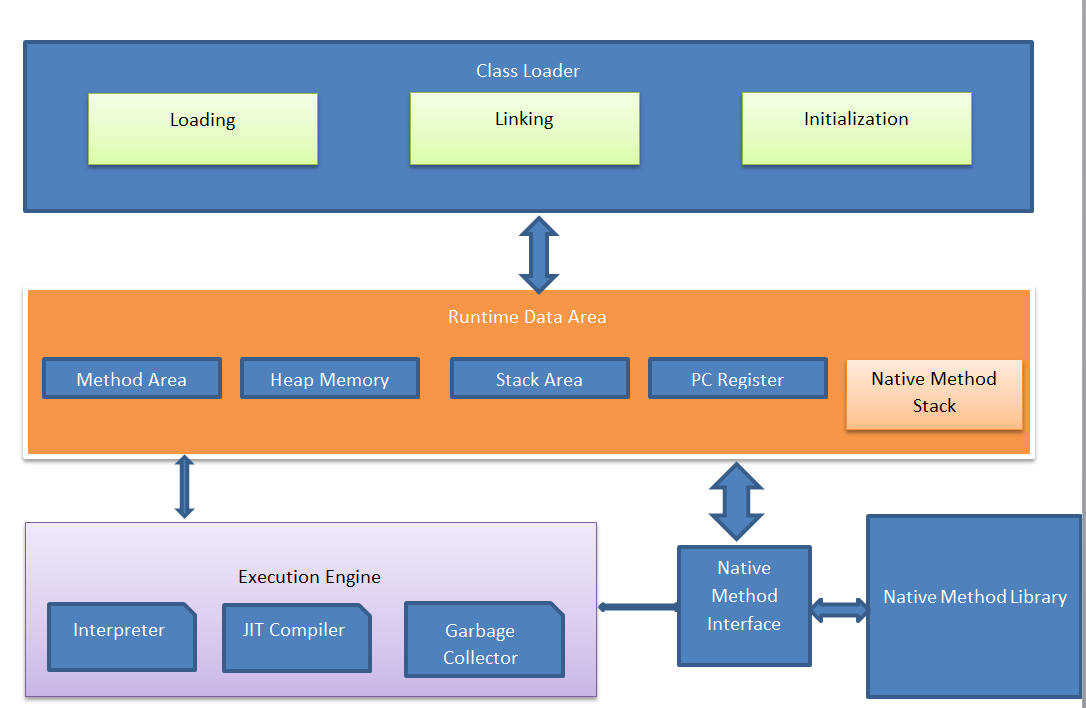
**Javascript, python…. Interpreter:**

**Java: compiler and interpreted**

**Abc.java javac : compiled;;;; class file/byte code: .class**

**JVM: interpreter the respective code: underlying platform: byte code in to the machine code:**

**JVM architecture:**

****

**Class Loader:**

1. **Load class:** 
   1. **Loading :**
      1. **Bootstrap class loader: folder: /packages in java: java.lang, java.net, java.io…**
      2. **Extension class loader: extension for standard java libraries:**
      3. **Application class loader:**
   2. **Linking:**
      1. **Verification: .class : VerifyException may occur: 19: compiled: 11:**
      2. **Preparation: JVM allocates memory for static fields of class and interfaces: and initial values are set by JVM:**
      3. **Resolution : runtime constant pool**

**Employee{**

**Int a, int b, int c;**

**Void getData(){**

**}**

**}**

**Employee e1=new Employee();**

**Employee e2=new Employee();**

**e1.getData()**

* 1. **Initialization : constructors: : in the constructor or static block:**

**Runtime Data Area:**

1. **Method Area: all the class level data: runtime constant pool, fields, methods area, code for method , constructor…. OutofMemoryException**
2. **Heap Memory: all the objects and their corresponding instance variables:**
3. **Stack area: Thread: OS: t1: t2: t3:……. R1: R2: R3: : local varibles, method calls, and partially calculated results** 
   1. **Local variable**
   2. **Operand stack**
   3. **Frame data:**
4. **PC: Program counter register: to hold the**

**Address: of instruction:**

**T1: {**

**1;//addresss must be stored some where**

**2;**

**3**

**4**

**}**

**Native method stack: Native methods: which are not written in Java: C, C++:**

**Execution Engine:**

**Print some document : libraries: :**

Installation: JDK, Eclipse:

Hello world program without using any IDE:

Programming using IDE:

Variables in Java : int a=10

Int b=20;

Whose value can be changed: there values can be changed:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 |
| a | b | C | D | E | F | G | h |

Types of variables:

1 ) Local variables: a variables which is declared inside any method or block is known as local variable: there scope is limited to their block: they are not accessible outside the block

**2) Instance variables: a variable which is accessible using instance/object of the class of the class:**

**3) static variables: We need to have a single copy of the variables:**

**Class is a collection of similar kinds of objects:**

**Students: pankaj, Swapnil…… sid, name, age, opted for some course: object or instance:**

**Student swapnil;**

**Student pankaj;**

FirstSL f1=new FirstSL();

**Global variable: these variables are not declared inside any method nor inside any block**

**They are declared in a class:**

Data types in Java:

1. Primitive Data types:which are used to hold simple values:
2. boolean true **or** false : 1 bit
3. char ‘c’ 2 bytes
4. byte 0 1 byte
5. short 1 2 bytes
6. int 12009 4 bytes
7. float 10.20f 4byte
8. long 1980382038 8 bytes
9. double 10.220 8 byte
10. Non primitive Data types: Array, class, interface….

Operators in Java:

1. Unary operator: operaotrs which works on a single operand: a++
2. Binary operator: 2 operands: :::: a + b, a - b
3. Ternary operator: more than 2 operands: a>b? a is bigger : b is bigger
4. Arithmetic operators: +, -, \*, /,%
5. Logical operators: a > b, a<b , a>=b, a<=b, a!=b Relational operators
6. Assignment operators: a=10, a=a+10; a+=10; a-=10; a=a-10; a\*=10; a=a\*10….a%=10, a/=10;
7. Bit wise operators: bit wise and &, or: |

a>b

syso(“a is bigger”);

syso(“b is bigger”);

Control statements

If-else:

If(condition1)

{

}

else if(condition2){

}

else if(condition3){

}

Else if(condition n){}

Switch case:

Switch(expression ) {boolean expression: evaluated : generate some value: 1

Case value1:

statement1;

statements2;

break;

case value2:

statement 1:

statement2:

break;

case3:

.

.

.

.

.

.

Default :

}

<< >>: bit wise operator

Binary bit:

10 decimal :

Binary are always having their base 2

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| 0 |  |  |  |  | 0 | 0 | 0 |

For: “Hello All”; multiple:

for(initialization; condition/termination condition;inc/decr ){

syso(“Hello All”);

}

Syso()

While:

While(condition){

//code;

//code;

++,--

}

Do while:

do{

state1;

state2;

..

}

while(condition);

String: String : Sequence of character: more than : String: class

Data entry : max data : String

Int n=str1.CompareTo(str2)//

If(n==0){

Syso(“pw is correct”);

}

Else if(n>0){

Syso(“the first String is larger than second”)

}

Else

{

Syso(“Second string is larger”)

}

Concat()

equals()//true

==;

Split(): str1.split(‘’); Hello: H E L L O

length(): 5

replace()

substring():

Name: nilesh

Last name: magar

Username: nilesraghahv

Pw:

Based on declaration : memory:

2 :

1. String str=”Hello All”;//String literal

String constant pool

1. String str=new String(“Hello All”);// new //instantiating string

Heap memory:

PAN card:

10lakh:

City: Pune:

City: pune: poone

String in Java in Immutable: Its contents cannot be changed : once it is declared :

String str=new String(“SimplyLearn”);

Str=str.concat(“Lets Learn”);

StringBuffer: mutable string: String:

Thread safe: at a time StringBuffer: only one thread: only one user can access this String:

Stringbuffer class is bit slow:

1. StringBuffer sb=new StringBuffer();// initial capacity of SB is 16
2. StringBuffer sb1=new StringBuffer(20);
3. StringBuffer sb2=new StringBuffer(“Hello All”);

Public synchronized Stringbuffer append(String s):

Public synchronized StringBuffer replace(int startindex,int ending index, String str)

StringBuilder: multiple thrads can access this StringBuilder:

1. StringBuilder sb=new StringBuilder ();// initial capacity of SB is 16
2. StringBuilder sb1=new StringBuilder (20);
3. StringBuilder sb2=new StringBuilder (“Hello All”);

Object and classes in Java:

Which have some common Properties:

Student: Akash, ROhit, Rimi, Rincy:

Class Librarian{

Int lid;

String Lname;

String desi;

Fields ;

Mehods;

Issue(){}

calculateFine(){}

procurebooks(){}

constructor:

}

Constructor:

Constructor is special kind of method in java: It has special purpose: to initialize instance variables:

Rule:

1. Name must be same as that of your class name
2. Need not to have return type: // instance/object
3. Abstract , static, final, synchronized:

Types of Constructors In java:

1. No argument constructor:
2. Parameterized constructor:

Constructor Overloading: It is decided at the compile time that which constructor it has to call that is why it is also known a compile time polymorphism:

Taking more than one form:

Overloading is not recommended in java with changing the data type:

Constructor overloading happens in the same class:

1. By changing number parameters
2. Changing the data types of the parameters:

Method: :

Function: Method:

2 types of methds:

1. Inbuilt methods: /functions:
2. User defined Method:

If we are performing some activity; process again and again :

We put such a code inside a method

Method: self contain block of the code: : inside which we are having some code:

Upon calling that function /method:

Body:

Signature:

public int add(int a,int b){

Jdjljlkdsjflsfdsf;

;fjdsjflskdjflsdkfj;

}

Calling : how to write a method

How t write method:

1. Signature:

public void display(){

c=a+b;

syso(c);

}

Access specifier/modfiers::

Class ABC {

Public void diplay(){

}

}

Class test{

Psvm(){

ABC a=new ABC();

a.display();//error

}

}

1. By changing number parameters
2. Changing the data types of the parameters:

Compile time polymorphism:

static:

this:to refer the variables, mehods…. We use this keyword:

1. Instance variable
2. Current class method implicitly :
3. If want to invoke/call current class contractor:
4. Pass an argument to the method:
5. Used to pass as an argument in the constructor:
6. Use this keyword in return statement of method:

inheritance:

Mechanism with the help of which object of one class can acquire the properties of another class:

Class Shape{

Int a;

Public void Area(){}

}

Class Squre{

Public void Area(){}

}

Class which provides features to be inherited such class is known as Super class, parent, base class

Classes which are acquiring the features of their parent are known as child class, sub class, derived

Shape

Rectangle

Square

Circle

Is a kind of relation:

Reusability of the code:

extends, implements(interfaces)

classes: Shape and circle

class circle extends shape{

}

Class Square implements Shape{ Shape is an Interface

}

1. Single inheritance : class circle extends shape{}
2. Multiple inheritance: ~~class circle extends shape, Dimension:~~ to avoid the ambiguity
3. Multilevel inheritance: class B extends A{} class C extends B

Vehicle

Bike

Honda

1. Hierarchical inheritance: class A extends B{} class C Extends B

Child acquired the properties of the parent class: All:

Child: all liabilities of their parent: land flat bungalow: must take care of your parents:

Overridding:

If your subclass is having a method with the same name as that of your parent class :

Method name, parameter, inheritance :

We achieve the run time polymorphism:

Bank class: Parent: findrateofInterest()

BankOfIndia: child: findrateofInterest()

SBI: findrateofInterest()

HDFC: findrateofInterest()

**Super**: is used in child class constructor: to call the parent class constructor :

Reference variable: but parent class object :

Where we can use the super keyword:

1. Immediate parent class instance variable:
2. Immediate parent class method
3. Immediate parent class Constructor: super();

Abstraction in Java: important functionality:

Abstract class: abstract as well as non-abstract method: 0 % to 100%

Interface: 100%

Abstract class:

abstract class Bike{

abstract void showBike();

public void getdata(){

stat1;

state2;

}

}

Class Bajaj extends Bike{

}

Class Emp{}

Class test{

Psvm(){

Emp e1=new Emp();

Bike b=new Bike();// abstract class: its child can be istansited :

}

}

Abstract class must be inherited:

Abstract class must have a child class:

Class which is inheriting the abstract class is normally known as Concrete class

Abstract class can be root class

Interface: 100% : blueprint of class: constant: abstract

**Since Java 8 : default and static**;

**Since java 9**: Private method: restricted from child classes to be accessed

Interface I1{

}

Either a class or another interface can inherit it:

class xyz implements I1{

}

Interface I2 extends I1{

}

Mutiple inheritance in java:

Class A extends I1,I2{}

Packages: windows: folder: files: class, interface:

Packages:

1. In built : import : io, util, sql, awt….
2. User defined packages: discussing without using Eclipse:

Acess Mdofier: public, private, protected, **default**(not keyword)

Public Class XYZ{

Public Int a;

Void add(){

}}

Int a;//package private//default

* **private**: only inside the same class
* **Default**: we do not use default keyword: it can be accessible only inside the same package:
* **protected: accessible outside the package but through the subclasses**
* public : inside class, inside package, outside;;;;

1. Write a java program with abstract Bank class which has abstract methods deposit and withdraw. When customer try to withdraw some amount if available balance is less than withdraw amount it should give error message. Make use of proper access modifiers.

Array in Java:

Wrapper classes:

Command line argument:

Scanner class:

Exception Handling:

Introduce: Threading in java

Collection framework:

Array in Java: :

Int a=100;

a=200;

a=300;   
list: 10,20,30,40,50….int: “Akash”,”nitin”,”Vishwa”,”Aprijta”,10.20f,10.30f,20,30f…. String, float, double

Array: Array is collection of similar kind of elements: occupying contiguous memory:

10,20,30,40,50

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Randomly:

A[4]: 50

A[9]: 100

A[10]: ArrayIndexOutOfBoundException

1. Single Dimensional Array
2. Multidimensional array: 2 d:

**Single dimensional**: 10,20,30,40….. row:

Syntax: **Declaration**:

Datatype[] a; //int[] a;

Datatype []a; //int []a;

Datatype a[]; //int a[];

Nameofthearray=new int[size]

int[] a;

a=new int[5];//size of the array: add more elements in the array: array’s cannot grow in java:

a[0]=100;

a[1]=200;

a[2]=300;

a[3]=400;

a[4]=500;

a[5]=600;// ArrayIndexOutOfBoundException

Multi-dimensional array: 4 rows and 3 columns:

10 20 30

40 50 60

70 80 90

|  |  |  |
| --- | --- | --- |
| 10: 00 | 20: 01 | 30: 02 |
| 40: 10 | 50: 1: 1 | 60: 1 2 |
| 70: 20 | 80: 21 | 90: 22 |

Row: 3

Columns: 3

3X3 Matrix: Array: a[rows][column]

Syntax: Multi dimensional array:Declaration of Multi dimensional array

Datatype[][] a; //int[][] a;

Datatype [][]a; //int [][]a;

Datatype a[][]; //int a[][];

Int a[][]=new int [3][3];

a[0][0]=10

a[0][1]=20

a[0][2]=30

a[1][0]=40

a[1][1]=50

a[1][2]=60…

a[2][0]=70

a[2][1]=80

a[2][2]=90

int a[][]={{ 10,20,30},{40,50,60 },{70,80,90 }};

Jagged array:

10 20 20 30 15

40 50 60

70 80 90 100

**Wrapper classes**: Java is not 100% OOPL:

For every Primitive DT: Wrapper classes:

Int a=10;// single value:

OOPL:

Primitive values:

Objects: Integer: String. Float, Double:

Integer a=10; lot of methods:

Int a=10; // inconvenient to use that value:

Integer i=a;

I=10;

Auto boxing and unboxing: 5.0: J2SE: 5.0

int a=10;

mobile: wrapping: gift wrapped: boxing: unbox;

1. boolean Boolean
2. char Character
3. byte Byte
4. short Short
5. int Integer
6. float Float
7. long Long
8. double Double
9. Command Line argument: java abc 10 20 30 40

At the beginning or while running the program we suppose to specify the values:

Please enter your age:

Pls enter your name:

Scanner class: : which a wide ragne of methods with which we can accept data from the user and perform many operation:

Import java.util.Scanner;

Scanner sc=new Scanner(System.in);

hasNext(): true when: some value:

next(): string from the user

nextInt()

nextFloat()

nextXXX()

nextLine(): String

sc.close();

1. Create bank application: in which you are creating menu driven program:

Case 1: opening new acc; Scanner class:

Case 2: deposit operation:

Case 3: Withdrow popration:

Case 4 : exit:

**Exception Handling: regular: Abnorma condition in the program:**

**Jvm may throw an exception object:**

**Int a[]=new int[5];**

**a[5]=100;// its not logical.. ArrayIndexOutOfBoundExeption**

**int a=10/0; DivideByZeroExceptoin: ArithemeticException**

**try**

**catch**

**ladder of catch block**

**Nesting of try and catch block**

**Finally:**

1. **Our example may or may not have exception**
2. **May have Exception :** 
   1. **Exception is handled**
   2. **Or not handled**
3. **May not have Exception :**

**throw: Explicitly: age<18 is not problem: Custom Exception/User Defined Exception:**

**throw new ExceptionClass(“Message”);**

**Checked Exception: throws:**

**throw new ArithmeticException(“Invalid Age”)l;**

**throws: declare exception :**

**throws keyword is always used only with method signature:**

**public void test() throws FileNotFoundException, SQLException, NullPointerException**

**{**

FileReader file=**new** FileReader("d:\\NPKey.txt");

BufferedReader br=**new** BufferedReader(file);

**}**

**Throws : we normally declare checked exception:**

**Error:**

**Unchecked with throws:**

**Difference between throw and throws keyword:**

**Throw : checked , unchecked and custom : if we wan to throw any exception explicitly we use throw :**

**Throws : checked:**

**User defined Exception /Custome exception:**

**Age is <18:**

**Withdrow bank acc:**

**10000 Rs**

**20000:**

**User inbuld: classes**

**Public class SomeNameExcpetion extends Exception {**

**SomeNameExcpetion(String msg){**

**Super(msg);**

**}}**

**1) Write 3 user defined Exception: if(age<18) : InvalidAgeExpection , if(age>18 and age<55): CorrectVoterException or if (age>55) : DoNot VetException**

**Threading:**

**Process: Program under execution:**

**MS word: Browser:**

**OS on behalf of the user: opening MS word: OS create a process:**

**one program : One process will be created:**

**Single Processing architecture: we can run more than one : separate process: System was Single processor: CPU: At that time CPU: browser: word,… only after that finish its execution: Single processor architecture:**

CPU

P1, p2, p3, p4, …

**Reponses time was very slow, Through put of the system: :**

**Multi processor architure was intro:**

P1, p2, p3, p4, …

CPU1: io: p1

CPU0

GP:

CPU2

CPU3

CPU4

**Multiprocessor: there were:**

1. **Context switch: Process initiated by the OS: Process: need to have its own: Adress space: memory: each process need to have its own resources : file: xyz.txt: printer**

|  |  |  |  |
| --- | --- | --- | --- |
| **P1** | **P2** | **P3** | **P4** |

**Deadlock, starvation :**

**P1 stopping its execution: 4/10 : saved somewhere: start its execution form that point:**

**State store:**

**Restoring the state:**

**Context Switch:**

**P2 Start:**

**Program: p1: : threads: 10 : has some memory: resources:**

**1**

**2**

**3**

**4**

**.**

**.**

**.**

**10**

**10 buildings: 50 : 500:**

**Only one postman: deliver: 9am…. 3pm: 3.30: 7:**

**10 postmans: each postman 1 : building: 9am: 2 pm: 2.30: office work:**

* **public void run():**
* **public void start():**
* **public static void sleep(long miliseconds) throws InterruptedException**
* **public int getPriority():**
* **public int setPriority(int priority):**
* **public String getName():  thread-0, thread-1…** 
  + **public String getName():**
* **public void setName(String name):**
  + **public void setname(Sring name) :**
* **wait()**
* **currentThread()**
* **NORM\_PRIORITY**
* **MAX\_PRIORITY**
* **MIN\_**
* **Join():**

**T1, t2, t3: t1.join()**

**Public final void join() throws IntrrupetedException**

**Thread/Process: thread or process: t1, t2, t3.. t10: Thread priority=10: 7, 4, 3,1 : 6**

* **public final int getPriority(): 5**
* **public final void setPriority(int priority):**
  + **NORM\_PRIORITY: 5**
  + **MAX\_PRIORITY: 10**
  + **MIN\_PRIORITY: 1**

**Synchronzations:**

**T1 t2**

**R(a) R(a)**

**A=a+10=20 a=a+20=30;**

**W(a); w(a):Inconsistency :**

**In a program:**

**{**

**Update operations: //critical section: Interthread comm: a=10+10;**

**}**

**A=10:**

**Types of synchronization:**

1. **Mutual Exclusion: only one thread to be allowed:** 
   1. **Synchronized Method: it is a common resource: t1, t2:**
   2. **Synchronized block**
   3. **Static synchronization**

**T1:**

Obj1

**T2:**

**T3:**

Obj2

**T4:**

1. **Cooperation: (Inter thread communication):**

**T1, t2, t3: semaphore or monitor: : s :0 : p**

**T2.wait() t2.notify(), t2.notifyAll()**

T1

**T2, t3:**

**Go through the Collection framework family:**

**Bank : Withdrow : User defined Exception InsuffcintFundExeption; more than his acc balance;**

**yield():**

1. Collection framework classes:

Int a=10; primitive :

A=20;

A=30;

10,20, 30 , 40….. Array:

Int []a=new int[10];// Array works on primitive DT: 100% OOPL:

Wrapper classes: object types:

Integer

Float

Double…

C: C++:

Linked list: adding node,, removing: DS: frameworks:

Inbuilt classes, interfaces and methods:

Collection Interface:

1. Public boolean add(E e):
2. Public boolean addAll(Collection c) :
3. Public boolean remove(E e)
4. Public boolean removeAll(Collection c)
5. Public boolean retainAll(Collection c)
6. Public int size(): no of elements
7. Public void clear()
8. Public boolean contains(Object o):
9. Public Iterator iterator(): // how to create and use iterator:
10. Public boolean isEmpty():

List : ordered collections: : : indexed based operation: Duplicate elements are allowed: Sugar… sugar

Public Interface List extends Collection {}

List Methods:

1. void add(int index, E e):
2. ArrayList:
   1. Duplicate:
   2. Maintain the insertion order:
   3. Not synchronized: 2 thread are allowed to access it:
   4. ArrayList Allows random access:
   5. Drawback: Manipulation on the ArrayList its bit slow:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |  |
| 10 | 20 | 30 | 40 | 60 | 70 | 80 | 90 | 100 |  |

ArrayList<Integer> a1=new ArrayList<Integer>();

Constructors:

ArrayList()

ArrayList(Collection e)

ArrayList(int capaity) :

Methods:

List and Collection

1. add()
2. addAll()

2) LinkedList: ;Doubly:

1) Duplicate elements

2) maintans the insertion order:

3) Non synchronized

4) Manipulation is faster as compare to ArrayList:

Constructor:

1. LinkedList()
2. LinkedList(Collection c)

Vector: 1.2 Java

1. Vector(): Default size: 10;
2. Vector(int initial capacity);
3. Vector(int initialCapacity, int incr capacity)
4. Vector(Collection C):

4) Stack: Linear: LIFO:

Stackoverflow:

Insert the element:

StackUnderflow:

Stack()

Empty()

Push()

Pop()

Peek(): it is not removing : showing

Search(object o)

Set:

Operations on Set:

1. InterSection: Set1:=[22,**45**,**33**,77]; set2=[**33**,2,83,**45**,3,12,55]=Intersectionset =[33,45]
2. Union: [22,45,33,77,2,83,3,12,55]
3. Difference:

10 HashSet:

1. It contains unique:
2. HashSet contains null values
3. Not synchronized
4. Does not maintain the insertion order:
5. Search : HashSet
6. HashSet has default capacity=16; load factor: 75% 0.75

4 different constuctors:

1. HashSet(): 16, 75%/0.75
2. HashSet(int capacity)
3. HashSet(int capacity, float loadfactor);
4. HashSet(Collection c)

LinkedHashSet:

1. Unique
2. LinkedHashSet contains null values
3. Not synchronized
4. maintain the insertion order:

TreeSet: Does not allow the ducupicate

Non synchor

Asce