Multithreading

Multitasking
- at a time typing in word and hearing song like that

Multithreading
- each task can split called thread
- In word typing, at the same time it check for spelling mistake.
  - sub process
- ie) The process is breakdown it to subprocess called threads

- nowadays there are 8 core processor. But the main take one thread as to increase the use of 8 cores we go for multithreading. ie) create multiple threads and run at the same time with different cores processor.
- Ticket booking
- Eg: whenever use the apps like amazon, it send request and wait for response at the time use can also use other application like. So request is send only by subthread.
- e.g: Servelt can get request and give response to user so at the time several user access same servelt it creat multithreads.
- e.g: Gaming at a time many are playing so thread is there.

Thread:
→ It is smallest programed instruction managed by scheduler.

Types

Demon thread
1) It do not require user interaction

e.g:
- Garbage collects in Java
  - heap memory managment

User thread
1) Demon thread allow to create new user thread runs without user interaction

Asyncronized request

→ A and B thread work independently

e.g:
↳ AJAX
interactive design (name can be changed) like that

Syncronize

→ works on dependent and wait of return

Note:
→ All process share same process resources.
→ It manage by schedullar.

Creating thread

∟ Two ways

- implements runnable interface (1)
- extends thread class (2)

```
2) class name extends Thread   // because we not to extend another class here so multiple inheritance is not support so we move to another method
{
    public void run()   // overwrite the run() method
    {

    }
    public static void main(String a[])
    {
        name obj = new name();   ; (or) Thread t1 = new name();
        obj.start();
    }
}
```

whenever create object thread is created automatic calls

It automatically calls obj.run()

Thread — it take here only run()

name() — because in this we have the definition

best

name t1 = new name();

t1 can access other method also.

Note

⇒ Why directly if we run it means thread is not created

```
class Hi
{
    Public  void run()
    {
        for(i=0; i<5; i++)
        {
          System.out.println("Hi");
          try{Thread.sleep(500);} catch (Exception e) {}
        }
    }
class Hello
    {
     Pulic void run()
     {
       for(i=0;i<5;i++)
        {
         System.out.println("Hello");
         try{. . . }
        }
     }
Public class ThreadDemo
    {
     Main()
     {
     Hi obj1 = new Hi();
     Hello obj = new Hello();
       obj1.start();  ⟶ Thread.sleep(100);
       obj.start();
     }
          ↘
           for( )
            {
            printf("num");
```

because it is checked exception

Note

here whoever thread will get a time it give first as priority but here all are same so it execute randomly

with some constrain

because we used random numbers instead of leaving scheduler choose random

o/p

num
hi
hello
.
.
.

2) method

```
class Task Implements Runnable
{
public void run()
 {
 }
}
Class Runtask
 {
 Public static void main(String a[])
 {
  Task t = new Task();
   (or)
  Runnable t = new Task();
  Thread T1 = new Thread(t);
  Thread T2 = new Thread(t);
    T1.start();
    T2.start();
 }
}
```

use in when two or more require same object

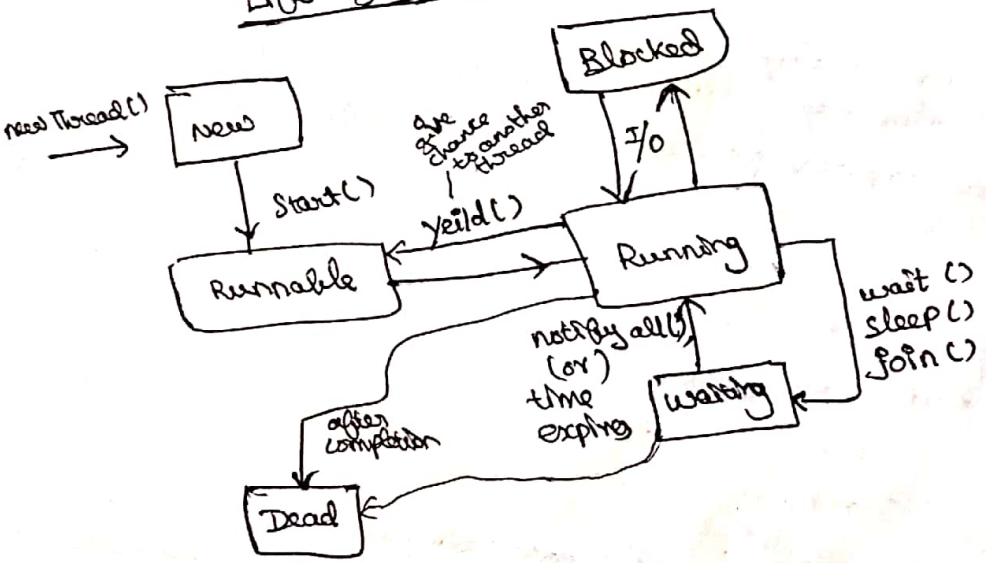
also we can write two different objects

So here we also create two thread for the same class

Note
Runnable interface has no → start method by default

So we pass the object as the parameter to Thread class

Life Cycle of Thread



→ Thread m = Thread.currentThread();
m.setName("main thread");
Thread T1 = new Thread()
T1.setName("T1 child");

→ Thread.sleep(100);

→ currentThread() , getName()
(it return the current running thread object)
Print(currentThread().getName());
(get the name)

→ obj.getState(); // It gives the current state i.e) life cycle

→ join();
• we can divide and do operation and can merge.
• They run parallely.
• here main thread wait. i.e) until child threads completed parent thread has to wait

{
Thread t1 = new Thread();
t1.start();
t1.join(); // until here main has wait after complet the t1 "Hi" is execute
Printf("Hi");

i) Note
Interrupted Ex

ii) Note
Thread default have name like
Thread-1
Thread-2

iii) during the time of I/O thread in block state

setPriority()

It is used to give the priority from 1 to 10 (1st, 10th)

```
t1. setPriority(10);
t1.start();
```

Collection Framework

All collections are in java.util.*;

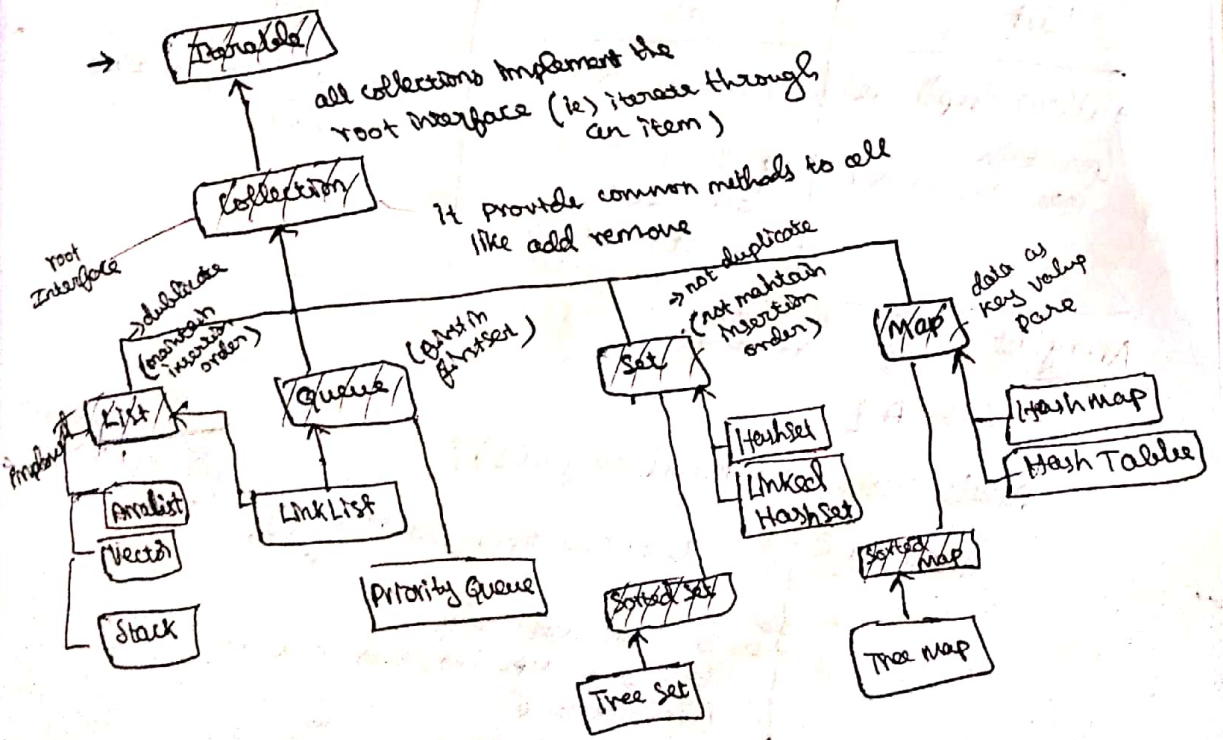
Data structure

→ Specified format of organizing and store data

→ Array, file, tree, list, queue, stack

→ Java provide API or Framework to work with datastructure called Collection Framework

→ operations
1) Insert
2) delete
3) search
4) update
5) sorting



Generics

→ same as collections. but differences is it create object specific data type.

→ In collections the function parameters accept only object type only ie) java.lang.Object

(here there is boxing and unboxing needed leads overhead)

Before 1.5 JDK

→ needs explicit typecasting because object can hold any datatype so in BR we need to explicy convert it using that (int)

e.g) int a = obj.b; // here may also error if it is string

a++;

To avoid this generics come into

→ It allow to create collection of specific data type

Iterate:

→ Traverse through an collection. (for doing operation)

List	Set	Map
→ allow duplicate (maintain insertion order)	→ unique item → order items (asc, dec) ie) not maintain insertion order	→ data in key value pairs.

ArrayList (use this for random access faster reading)

```
Public class A {
    Public static void main(String a[]) {
        ArrayList<int> a = new ArrayList<int>();
    }
}
```

ie) create object (reference) of seperate type

because all collection accept only reference type ie) class not an primitive type

Integer (wrapper class)

Integer

```
a.add(1);  // to add the value in record
a.add(3);
a.add(2);
System.out.println(a);          // op [1,3,2]

boolean b = a.contains(1) // It check the number or item present or not
                          it returns, boolean value
                          (true/false)
print(b); // op true

ArrayList<Integer> B = new ArrayList<Integer>();
    B.add(4);
    B.add(5);
    System.out.println(B); // [4,5]

a.addAll(B); // used to add list of items naturally in last append  // op [1,3,2,4,5]

a.addAll(1,B)  //  op [1,4,5,3,2]
   ↓ position to be inserted

a.remove(1);   //  op [1,5,3,2]
   ↓ position to be removed

here auto matically position can rearrange ie) after that position all reduce by 1

System.out.println(a.get(1));
                      ↓ position
   to get an element at that position

System.out.println(a.indexOf(5));
                      ↓ element to be searched
   return the index value of 5

// will happen when the particular value is not there
```

LinkedList:

→ allow to navigate both the direction
→ not for randomly access value
→ used to access sequentially one after another, one before another

```
LinkedList<Integer> a = new LinkedList<Integer>();
```

additional methods in LinkedList

o/p
[1,2,3,4,5]

	o/p
a.addLast(1);	[1,2,3,4,5,1]
a.addFirst(2);	[2,1,2,3,4,5,1]
a.removeFirst();	[1,2,3,4,5,1]
a.removeLast();	[1,2,3,4,5]

```
a.getFirst();
a.getLast();
```

SetInterface:

here Insertion order not maintained

```
HashSet<Integer> A = new HashSet<Integer>();
```
(or) String (or)

```
A.add(3)
A.add(1);
A.add(2);
A.add(1);   // o/p [3,1,2]
```

// duplication not allowed similar of that method

```
TreeSet<Integer> A = new TreeSet<Integer>();
```

// Difference:
It automatically maintain sorting order
Default: Ascending

o/p
[1,2,3]

```
SOP((TreeSet<Integer>) A.headSet(2)); // ie) 1
```

typecast to assign

It gives the view of element

it display the from first element before 2

// here it is costlier because each element can compare with other elements

" A.tailSet(2); it display from that to last
ie) 2,3

Map Interface

- TreeMap <String, Integer> A = new TreeMap<String, Integer>();
  - String ↑ name / key
  - Integer ↑ value

```
A.put("pan", 12);    // to Insert
A.put("Be", 13);
```

O/P:

{Be=13, pan=12}

Note
- here keys are sorted
- here Duplication of values allowed but not keys

Comparable and Comparator (Interface)

Comparable

→ These are used for searching and sorting process, whenever comparision required.

→ CompareTo method provide comparisons → returns (0, -1, 1)
↓ one comparision logic here

Comparator:

→ Also searching and sorting

→ Compare method and CompareTo method
- Compare ↓ one parameter as argument
- CompareTo ↓ Two parameters

(0, -1, 1)

```
import java.util.*;
class name implements Comparable<name>
{
```

↓ here we want to specify type, must be reference type

```
public int compareTo(name obj)
{
  if(this.sal == obj.sal)
     return 0;
  else if(this.sal > obj.sal)
     return 1;
  else
     return -1;
```

→ here by default the Integer class implements Comparable Interface and define compareTo()

Note: it is default method of Collections.sort(obj); Collections.reverse(obj);

sorting of Integer sort

```
main()
{
  List<name> obj = new ArrayList<>();
  Collections.sort(obj); // this use natural ordering
                         // (ie) we not we provide
                         // inside that
                         // compareTo()
```

plus name

Comparator

~~A ClassObj new A()~~

→ So we need not that we need to overwrite we go for this

Collections.sort(obj,) ← object of comperater ← But it is interface

how we create objects of Interface
1) Annomious class
2) we also create the class and use the reference of Interface

```
Comparator<Integer> ob = new Comparator<Integer>()
  {
    public int compare(Integer i, Integer j)
    {
      // logic; return ;
    }
  };
Collections.sort(obj, ob);
```

because we create of Integer Class it be any user defined also

Comparable:

```
Class A implement Comparable
{
  public int compareTo(int a)
    {
    }
  public String toString(){
    return "
  }
```

here not need to implement

collections.sort(obj, ob);

Q) user defined class

```
ArrayList <Stu> L= new ArrayList <Stu> ();
Stu obj1 = new Stu ("hi", 1);
obj2 = new Stu ("  ";
obj3 = new Stu  "  ";
```

L. add (obj1);

Collection Framework — Mainly in util Package

→ It is a framework

→ Hierachy

→ List - In the order we inserted
Allows the modification

→ Dynamic array
can increase the size itself if required

1) ArrayList

- add (E e) — used to store return type is boolean
- add (int index, E element)

API - predefined class

```
import java.util.ArrayList;

main(){
ArrayList al = new ArrayList();   — use this we store different data type

    al.add ("SREC");
    al.add (12);
    al.add (12.45);

for (int i=0; i<al.size(); i++)
  {
System.out.println(al.get(i));
  }
}
```

non-generic
we use advance for loop because of different datatype

Note

size() - gives the size of the ArrayList

get(position) - it gives the

```
main () {
ArrayList<String> al = new ArrayList<String>();
        al.add ("SREC");
        al.add ("SRIT");

for (String s:al)
  {
  System.out.println (s);
  }
```

generic type same datatype, we can able to use advanced for loop

3) Iterator for retrival

```
import java.util.Iterator

Iterator itr = al.iterator();   // it is the object
while(itr.hasNext())            // it check next it having or not
{
  System.out.println(itr.next());   // move to next after printing
}
```

4) ListIterator (for reverse order) or front same

```
ListIterator itr = al.listIterator(al.size())   // give the size
while(ptr.hasPrevious())
  {
    System.out.println(ptr.previous());
  }
```

5) Lambda function

```
al.forEach( i->{
    System.out.println(i);
});
```

6) Collections.sort(al);   // now it is in sorted order