6/-12/22

Interrupting a Thread.

depends on situation

public void interrupt()

→ if thread is in sleeping state (or) in waiting state we can interrupt a thread.

eg#1

```
class mythread extends Thread {
    @override
    public void run(){
        try {
            for (int i=1; i<=10; i++) {
                System.out.println("I am Lazy thread");
                Thread.sleep(2000);
            }
        }
        catch (InterruptException e){
            System.out.println("I got interrupted");
        }
    }
}

public class Test3 {
    public static void main(String ... args) thrw
interruptedException {
        MyThread t=new MyThread();
        t.start();
        t.interrupt(); //line-n1
    System.out.println("End of main...");
}}
```

Scenario : if a comment line-ni

2 Thread

a. main thread

End of main...

b. Child thread

I am Lazy thread

...

...

if t.interrupt() then

Scenario :

2 thread.

a. main thread

main thread

b. Child thread

I am Lazy thread

~~I got~~ Interupted

Case 2

```
class mythread extends Thread {
    @override
    public void run(){
        for (int i=1; i<=10000; i++){
            System.out.println("I am Lazy thread : "+i);
        }
    System.out.println ("I am entering into sleeping state");
    try
    {
        Thread.sleep (2000);
    }
    catch (InterruptedException ie){
        ie.printstackTrace();
    }
}
}
public class TestApp {
    public static void main(String [] args) throws IE {
```

```
mythread t = new mythread();
t.start();
t.interrupt(); // line-n-1
System.out.println("main thread");
}
}
```

line-n1 is commented then no problem.
line-n1 is not commented, then interrupt() will wait till then the thread enters into waiting state / sleeping state.

Note:- if thread is interrupting another thread, but target thread is not in waiting state / sleeping state then there would be no exception.
interrupt() call be waiting till the target thread enters into waiting ~~once the target thread~~ state so this call won't be wasted.
once the target thread enters into waiting state/sleeping state then interrupt() will interrupt and it raises the exception
interrupt() call will be wasted only if the thread does not enters into waiting state / sleeping state.

yeild() join() Sleep()

i) purpose
1) yeild()
To pause current executing thread for giving the chance of remaining waiting threads of same priority
2) join()
if a thread wants to wait until completing some other thread then we should go for join()
3) Sleep() if a thread don't want to perform any operation for a particular amount of time then we should go for Sleep() method.

2) Is it static ?
yeild () yes
join() no
sleep () yes

4) Is it overloaded ?
yeild() no
join() yes
sleep() yes

3) Is it final ?
yeild() no
join() yes
sleep () no

5) Is it throws IE ?
yeild() no
join() yes
sleep () yes

6) Is it native method ?
yeild () yes
join () no
sleep ()
sleep (long ms) --> native
sleep (long ms, int ns) --> non- native.

<u>Note</u>:- using Lambda Expression

```
Runnable r = () -> {}
        for (int i=1; i<=5; i++)
        {
            System.out.println ("Child thread");
        }
    };
Thread t = new Thread (r);
t.start();
```

<u>using</u> <u>anonmyous</u> <u>inner</u> <u>class</u>

```
new Thread (new Runnable(){
        @override
            public void run(){
                for (int i=1; i<=5; i++) {
                    S.O.P("Child Thread")
                }
            }
}).start();
```

Synchronization

① Synchronized is a keyword applicable only for methods and blocks

② if we declare a method/block as synchronized then at a time only one thread can execute that method/block on that object.

③. The main advantage of synchronized keyword is we can resolve data inconsistency problems.

④ But the main disadvantage of synchronized keyword is it increases waiting time of the thread and effects performance of the system.

⑤ Hence if there is no specific requirement then never recommended to use synchronized keyword.

⑥ Internally synchronization concept is implemented by using lock concept.

```
Class X {
    Synchronized void m1() { }
    Synchronized void m2() { }
    void m3() { }
}
```

Key points

① if t_1 thread invokes m1() then on the object X lock will applied.

② if t_2 thread invokes m2() then m2() can't be called because lock of X object is with m1.

③ if t_3 thread invokes m3() then execution will happen becaz m3() is non synchronized.

lock concept is applied at the object level not at

the method level.

(7) Every object in java has a unique lock. whenever we are using Synchronized keyword then only lock Concept will come into the picture.

(8) if a thread wants to execute any Synchronized method on the given object 1st it has to get the lock of that object.

once a thread got the lock of that object then it's allow to execute any synchronized method on the object. if the synchronized method execution completes. then automatically Thread releases lock.

(9) while a thread executing any synchronized method. the remaining Threads are not allowed execute any Synchronized method on that object Simultaneously. But remaining threads are allowed to execute any non-Synchronized method simultaneously. [lock concept is implemented based on the object but not based on method].

Note:- Every object will have 2 area (Synchronized area and non-Synchronized area]

Synchronized Area => write the code only to perform update, insert, delete

Non-Synchronied Area => write the code only to perform select Operation

```
Class ReservationApp {
    CheckAvailability () {
    } //perform read operation
```

```
Synchronized bookTicket(){
    //perform update operation
}
}.
```

eg #1

```
Class Display {
    public void wish (String name){
        for (int i=1; i<=10; i++)
        {
        System.out.print("Good morning:");
        try{ Thread.sleep(2000);
        }
        Catch (Interrupted Exception e){
        }
        System.out.println(name);
    }
}
}
Class mythread extends Thread {
    Display d;
    String name;
    mythread (Display d, String name){
        this.d=d
        this.name=name
    }
    @override
    public void run(){
        d.wish(name);
    }
}.
```

~~state of~~

```
public class Test3 {
    public static void main (String... args) {
        Display d = new Display();
        mythread t1 = new Mythread (d, "dhoni");
        mythread t2 = new mythread (d, "yuvi");
        t1.start();
        t2.start();
    }
}
```

<u>output</u>:- As noticed below the output is irregular becoz at a time on a resource called wish()

2. threads are acting simulataneously

3. Threads -
   a. main Thread
   b. Child Thread -1
   c. child Thread -2

Good morning : Good morning : ....
....
....

<u>eg #2</u>

```
class Display {
    public Synchronized void wish (String name) {
        for (int i=1; i<=10; i++)
        {
            System.out.println ("Good morning:");
            try {
                Thread.sleep (2000);
            }
            catch (InterruptedException e) {
            }
            System.out.println (name); }}}
```

```
Class myThread extends Thread {
    Display d;
    String name;
    myThread (Display d, String name){
        this.d=d;
        this.name = name;
    }
    @override
    public void run() {
        d.wish (name);
    }
}
public class Test3 {
    public static void main (String ... args) throws IE {
        Display d = new Display();
        mythread t1 = new MyThread (d, "dhoni");
        mythread t2 = new mythread (d, "yuvi");
        t1.start();
        t2.start();
    }
}
```

output:
3 threads
a. main Thread
b. child thread-1
Good morning: dhoni
Good morning: dhoni
...

c. child Thread-2
Good morning: yuvi
Good morning: yuvi
...

Note:- As noticed above there are 2 threads which are trying to operate on single object called "Disply" we need Synchronization to resolve the problem of "Data inconsistency".

Casestudy::

```
Display d1 = new Display();
Display d2 = new Display();
myThread t1 = new myThread (d1, "yuraj");
myThread t2 = new myThread (d2, "dhoni");
    t1.start();
    t2.start();
```

In the above case we get irregular output, because two different object and since the method is synchorized lock is applied w.r.t object and both the threads will start simulataneously on different java objects due to which the output is "irregular".

Conclusion:-

if multiple threads are operating on multiple objects then there is no impact of synchronization.

if multiple threads are operating on same java objects then synchorized concept is required (applicable).

Class level lock

1. Every class in java has a unique level lock.

(2) if a thread wants to execute static synchronized method then the thread acquires "class level lock".

(3) while a thread executing any static synchronized method the remaining threads are not allow to execute any static synchronized method of that class simultaneously ~~normal static methods, and normal instance methods~~

(4) But remaining threads are allowed to execute normal synchronized methods normal static methods, and normal instance methods simultaneously

(5) Class level lock and object lock both are different and there is no relationship b/w there two.

eg:: class X {

static synchronized m1() { } // class level lock

static synchronized m2() { }

static m3() { } // no lock required

synchronized m4() { } // object level lock

m5() { } // no lock required.

}.

$t_1 \Rightarrow$ m1() → Class level lock applied and chance is given

$t_2 \rightarrow$ m2() → enter into waiting state

$t_3 \rightarrow$ m3() → gets chance for execution without any lock

$t_4 \rightarrow$ m4() → object level lock applied and chance is given

$t_5 \rightarrow$ m5() → gets a chance for execution without any lock

eg#1

```
Class Display {
    public Synchronized void displayNumbers(){
        for (int i=1; i<=10; i++)
        {
          System.out.print(i);
        try{
            Thread.sleep(2000);
        }
        Catch (InterruptException e){
        }
    }
    public Synchronized void displaycharacters(){
        for (int i=65; i<=75; i++)
        {
          System.out.println((char)i);
        try{
            Thread.sleep(2000);
        }
        Catch (Interrupt Exception e){
        }
    }
}
class mythread1 Extends Thread {
    Display d;
    mythread1 (Display d){
        this.d=d
    }
    @Override
    public void run(){
```

```
            d.displayNumbers();
        }
    }
class myThread2 extends Thread {
    Display d;
    myThreads (Display d) {
        this.d=d;
    }
    @Override
    public void run() {
        d.displayCharacters();
    }
}
public class Test3 {
    public static void main (String...args) {
        Display d1 = new Display();
        myThread1 t1 = new myThread1(d1);
        myThread2 t2 = new myThread2(d1);
        t1.start();
        t2.start();
    }
}
```

output:

<u>3 threads</u>

a. main Thread
b. userdefined thread (displayCharacters())
c. user defined Thread
displayNumbers()

Synchronized block

```
Synchronized void m1() {
    ....
    ....
}
```

if few lines of code is required to get synchronized then it is not recommoned to make method only as synchonied.
if we do this then for threads performance will be low, to resolve this problem we use "Synchronized block", due to Synchronized block performance will be improved..

Case Study

if a thread got a block of current object, then it is allowed to execute that block

a. Synchronized (this){
    ....
    ....
}

To get a lock of particular object :: B

b. Synchronized (B) {
    ....
    ....
}

if a thread got a lock of particular object B, then it is allowed to execute that block.

c. To get class level lock we have to declare Synchronized block as follows.

Synchronized (Display.class) {
    ....
}

if a thread gets class level block, then it is allowed to execute that block//

7/12/22

Synchronized block.

eg:1

eg#1.

```
class Display {
    public void wish (String name) {
        ;;;;;; // 1-lakh lines of code

    Synchronized (this) {
        for (int i=1; i<=10; i++)
        {
            System.out.println ("Good morning:");
            try {
                Thread.sleep (2000);
            }
            Catch (IE e) { }
            System.out.println (name);
        }
    }
    ;;;; // 1-lakh lines of code
} }
class myThread extends Thread {
    Display d;
    String name;
myThread (Display d, String name) {
        this.d=d
        this.name=name;
    }
    public void run() {
        d.wish (name);
} }
class Test {
    public Static void main (String [] args) {
        Display d=new Display ();
    myThread t1=new myThread (d, "dhoni");
    myThread t2=new MyThread (d, "yuvi");
```

```
t1.start();
t2.start();
}
}
```

IE => InterruptException

eg#2:

```
class Display {
    public void wish (string name){
        .... // 1-lakh of code
        ////
    synchronized (this){
        for (int i=1; i<=10; i++) {
            System.out.print ("Good morning:");
        try{
            Thread.sleep(2000);
        }
        catch (IE e){}
        System.out.println (name);
    }
    .... // 1-lakh lines of code
}
}
class Mythread extends Thread {
    display d;
    String name;
mythread (Display d, String name) {
    this.d=d
    This.name =name
}
public void run() {
    d.wish (name);
}
}
public class Test{
public static void main (String[] args) {
    Display d1=new Display();
    Display d2=new Display();
mythread t1=new mythread (d1, "dhoni");
mythread t2=new mythread (d2, "yuvi");
t1.start()
t2.start();
}
}
```

output :- Irregular output becoz two object and two threads acting on two different objects

eg#3

```
class Display {
    public void wish (string name){
        ;;;;;; //1- lakh lines of code
Synchronized (Display.class){
        for (int i=1; i<=10; i++){
            System.out.println ("Good morning!");
        try {Thread.sleep (2000);
        }
        Catch (InterruptedException e){}
        System.out.println (name);
    }}
    ;;;;;; // 1-lakh lines of code
}}
Class mythread extends Thread {
    Display d;
    String name;
my thread (Display d, string name) {
        this.d=d;
        this.name =name;
    }
    public void run(){
        d.wish (name);
}}
public class Test {
    public static void main (String[] args) {
        Display d1=new Display();
        Display d2= new Display();
    mythread t1=new mythread (d1, "dhoni");
    mythread t2 = new mythread (d2, "yuvi");
    t1. start();
    t2. start();
}
}.
```

Note:- 2 object, 2 thread, but the thread which gets a chance applied class level lock so output is regular.

Note: lock concept applicable only for objects and class types, but not for primitive types. if we try to do it would result in compile time error saying "unexcepted type"

eg: int x = 10;
Synchronized (x) { // CE : unexcepted type found: int required : reference }

(Interthread Communication (remember postbox to example))

Two threads can communicate each other with the help of
a. notify()
b. notifyAll()
c. wait()

notify() ⇒ Thread which is performing updation should call notify(), so the waiting thread will get notification so it will continue with its execution with the updated items.

wait() → Thread which is expecting notification/updation should call wait() Immediately the thread will enter into waiting state.

If a thread wants to call wait(), notify()/notifyAll() then compulsory the thread should be the owner of the object otherwise it would result in "Illegal monitorstate exception".

we say thread to be owner of that object if thread has lock of that object.

It means these methods are part of synchronized block or synchronized method, if we try to use outside.

Synchronized area then it would result in Runtime Exception called "Illegal monitor state exception".

if a thread calls wait() on any object, then first it immediately releases the lock on that object and it enters into waiting state.

if a thread calls notify() on any object, then he may or may not release the lock on that object immediately.

Except wait(), notify(), notifyAll() lock can't be realased by other methods.

<u>Note</u>: yield(), sleep(), join() → can't release the lock.

wait(), notify(), notifyAll() → will release the lock, otherwise interthread communication can't happen.

Once a thread calls wait(), notify(), notifyAll() methods on any object then it releases the lock of that particular object but not all locks it has.

Method prototype of wait(), notify(), notifyAll()

~~methods on any object then it releases the lock of that particular object but~~

1. public final void wait() throws IE
2. public final native void main (long ms) throws IE
3. public final void wait(long ms, int ns) throws IE
4. public final native void notify()
5. public final void notifyAll()

<u>Interview Question</u>

method like wait(), notify(), notifyAll() are present inside object class, y not in Thread class.

Interview Question

~~method like wait(), notify(), notifyAll() are present~~

Thread will call wait(), notify(), notifyAll() on objects like postBox, stack, customer, student, ...

→ obj.wait(), obj.notify(), obj.notifyAll()

these methods should be available for every object in java, if the method has to available for every object in java then those methods should come from "Object" class.

program

```
eg#1 → class ThreadB extends Thread {
        int total =0;

        @override
        public void run(){
            for (int i=0; i<=100; i++){
                total+=i;
            }
        }
public class Test {
        public static void main(String[] args) throws IE {
            ThreadB b = new ThreadB();
            b.start();

            stmt-1
            System.out.println(b.total);
        }
}
```

A - Stmt-1

if i replace with thread.sleep(10000) they thread will enter into waiting statement but within 10 sec if the updation is not ready, then we should not use thread.sleep(10000)

B. <u>Stmt 2</u> → if i replace with b.join(), then main thread will enter into waiting state, then ~~main thread has to wait.~~ ~~no~~ childthread will execute for loop, till then main thread has to wait main thread is waiting for updation result

```
for (int i=0; i<=100; i++){
        total += i;
}
```

//10 lines of code is available main thread has to wait till 10 lines of code (why) main thread should wait for the complete code to finish.

<u>eg#12</u>

```
Class ThreadB extends Thread {
    int total = 0;
@ override
    public void run() {
        Synchronized (this) {
            System.out.println ("child thread started
                calculation");
            for (int i=0; i<=100; i++){
                total += i;
            }
            System.out.println ("child thread trying to
                give notification");
        }
    }
}
public class Test {
    public static void main (String[] args) throwing
        IE {
        ThreadB b = new Thread();
        b.start();
        Thread.sleep(10000); //10 sec
Synchronized (b) {
        System.out.println ("main thread is calling wait on
            B object");
```

```
b.wait();
System.out.println ("main thread got notification");
System.out.println (b.total);
}}
}
```

output:- Child thread started calculation
Child thread trying to give notification
main thread is called wait on B object beaz of
Thread.sleep (10000) main thread will never get notification

eg#3

```
class ThreadB extends Thread {
    int total =0;
  @override
    public void run(){
      Synchroized (this){
    System.out.println ("Child thread started calculation");
   for (int i=0; i<=100; i++) {          //Step-2
      total+=i;
   }
   System.out.println ("Child thread trying to give
                        notification"); // Step-3
   this.notify();
}}
}

public class Test {
    public static void main (String[] args) throws
                                          IE {
    ThreadB b= new ThreadB();
      b.start();
  Synchronized(b) {
      System.out.println ("main thread is calling wait
        on B object "); //Step-1
      b.wait (10000); //10sec
    S.o.pln ("main thread is ~~calling wait~~ on notification"); //
}}}  S.o.pln( b.total);
```

outputs: child thread started calculation.
Child thread trying to give notification.
main thread is calling wait on B object for 10sec
main thread got notification
5050

producer/consumer problem

producer → produce the item and update in the Queue (notify())
consumer → consume the item from the Queue. (wait)

```
class producer extends Thread {
    producer () {
        synchronized(q) {
            produce the item and update it to queue
            q.notify();
    } } }

class consumer extends Thread {
    Consumer () {
        Synchronized (q) {
            if(q is empty) {
                q.wait();
            }
            else {
                Consume the item from the queue
    } } } }
```

Difference b/w notify and notifyAll()

notify() → to give notification only for one waiting thread.

notifyAll() → To give notification for many waiting thread.

⇒ we can use notify() method to give notification for only one thread. If multiple Threads are waiting then only one ~~m~~ Thread. If multiple Threads are waiting then only one thread will get the chance and remaining Threads has to wait for further notification But which thread will be notify (inform) we can't expect exactly it depends on

JVM:

eg:

waiting state

obj1.notify() | obj1.wait(); 60 threads are w

Running

Among 60 threads which thread will get a chance we ~~don't~~ have control over that it is decided by JVM (threadschedular).

⇒ we can use notifyAll() method to give the ~~notication~~ notification. for all waiting Threads of particular object.

All waiting threads will be notified and will be executed one by one, because they required lock

eg:: obj1.notifyAll() | waiting state

obj.wait(); 60 threads are waiting no fee

Obj2.wait();

Running state.

Note:- on which object we are calling wait(), notify() and notifyAll() methods that corresponding object lock we have to get but not other object locks.

eg:: Stack S1 = new Stack();
Stack S2 = new Stack();

```
Synchronized (S1) {
    S2.wait(); //RE : IllegalMonitorStateException
}
Synchronized (S2) {
    S2.wait(); (valid)
}
```

Question based on lock

1. If a thread calls wait() immediately it will enter into waiting state without releasing any lock (false)
2. If a thread calls wait() it releases the lock of that object but many not immediately (false).
3. If a thread calls wait() on any object, it releases all locks acquired by that thread and enters into waiting state (false).
4. If a thread calls wait() on any object, it immdiately realeases the lock of that particular object and entered into waiting state (true).
5. if a thread calls notify() on any object, it immediately releases the lock of that particular object (invalid).
6. if a thread calls notify() on any object, it releases the lock of that object but may not immediately (true).

8/12/22

Dead lock

if 2 threads are waiting for each other forever (without end) such type of situation (infitive waiting) is called dead lock. There no resolution techniques for dead lock but several prevention (avoidance) techniques are possible.

Synchronized keyword is the cause for deadlock hence whenever we are using synchronized keyword we have to take special care.

eg#1

```
class A {
    public void d1 (b b) {
        System.out.println ("thread-1 starts execution of d1()");
    try {
        thread.sleep (5000); //sec
    }
    catch (InterruptException e) {
    }
    System.out.println ("thread-1 trying to call b.last()");
    b.last();
    }
    public void last() {
        System.out.println ("Inside A last() method");
    }
}
class B {
    public void d2(A a) {
        System.out.println ("thread-2 starts execution of d2()");
    try {
        thread.sleep(5000); //sec
    }
    catch (InterruptException e) {
    }
    System.out.println ("thread-2 trying to call A.last()");
    a.last();
    }
    public void last() {
        System.out.println ("Inside B last() method");
    }
}
```

```
public class Test extends Thread {
    A a = new A();
    B b=new B();
  public void m1() {
        this.start();
        a.d1 (b); // executed by main thread.
    }
    public void run(){
        b.d2 (a); // executed by child thread
    }
    public static void main (String[] args){
        Test t= new Test();
        t.m1();
    }
}
```

Since methods are not Synchronied, lock is not required, so no deadlock.

<u>output</u>:- Thread-1 starts execution of d1()

thread-2 starts execution of d2()

thread-1 trying to call B last

Inside B last() method

thread-2 trying to call A last()

Inside A last() method.

<u>eg#3</u> :-

```
class A extends Thread {
    public synchronized void d1(B b){
        System.out.println ("thread-1 starts execution of d1()")
    try{ Thread.sleep(5000); //5sec
    }
    Catch (Interrupt Exception e) {
    }
    System.out.println ("thread-1 trying to call B last()");
        b.last();
}
```

```
public Synchronized void last(){
        System.out.println("Inside A last() method");
    }}
Class B extends Thread {
        public Synchronized void d2 (A a) {
        System.out.println("thread-2 starts execution of d2()");
    try {
        Thread.sleep(5000); //5sec
    }
    Catch (InterruptException e) {
    }
    System.out.println("thread-2 trying to call A last()");
    a.last();
    }
    public Synchronized void last(){
        System.out.println("Inside B last() method");
    }
}
public Class Test extends Thread {
        A a= new A();
        B b= new B();
    public void m1(){
        this.start();
        a.d1(b); //line executed by main thread
    }
    public void run(){
        b.d2(a); //line executed by child thread.
    }
    public static void main (String[] args) {
        Test t=new Test();
        t.m1(); //main thread s executing
    }}
}
```

In the above program, there is a possibility of "dead lock".

Output

Thread-1 starts execution of d1()

Thread-2 starts execution of d2()

thread-1 trying to call B last()
thread-2 trying to call A last()
// here cursor will be waiting

$t_1 \rightarrow$ starts d1(), since d1() is synchronized and a part of 'A' class so t_1 applies lock of (A) and starts the execution, while executing it encounters Thread.sleep(). So T.S gives chance for t_2 thread.
After getting a chance again by TS, it tries to execute b.last but lock of b is with t_2 thread, so t_1 enters into waiting state
$t_2 \rightarrow$ starts $d_2()$, since $d_2()$ is synchronized and a part of 'B' class so t_2 applies lock of (B) and starts the execution, while executing it encounter thread.sleep(), so TS gives chance again for t_1 thread.
After getting a chance again by TS it tries to execute a.last but lock of a is with t_1 thread, so t_2 enters into waiting state.
Since both the threads are in waiting state and it would be waiting for ever, so we say the above pgm would in "Deadlock".

<u>Note</u>:- Synchronized is the only reason why there is a deadlock, so we should be careful when we use synchronized keyword, if we remove atleast one synchronized word then the program won't enter into "deadlock".

Deadlock vs Starvation

long waiting of a thread, where waiting never ends is termed "deadlock". ~~long~~
long waiting of a thread, where waiting ends at certain point is called "Starvation".

eg: Assume we have 10 threads, where all 10 threads have priority is 10, but one thread is there which has priority 1, now the thread with a priority-1 has to wait for long time but still it gets a

a chance, but it has to wait for long time scenario is called "Starvation".

Note:- low priority thread has to wait untill completing all priority threads but ends at certain point which is nothing but Starvation

Daemon Threads

The thread which is executing in the background is called "Daemon Thread".

eg: Attach listener, Signal Dispatcher, Garbage Collector, ...

remember the example of movie.

1. producer
2. director
3. Music director
4. ...
5. ...
6. ...

main objective of Daemon Thread

the main objective of Daemon Thread, to provide support for non-Daemon threads (main thread).

eg:: if main threads runs with low memory then will call GarbageCollector thread, to destory the useless objects so, that no. of bytes of free memory will be improved with this free memory main thread can continue its execution usually Daemon threads having low priority, but based on our requirement daemon threads can run with high priority also.

JVM ⇒ Creates 2 threads.

a. Daemon Thread (priority = 1, priority = 10)

b. main (priority = 5)

while executing the main code, if there is a shortage of memory then immediately jvm will change the priority of Daemon thread to 10, so Garbage collector activities Daemon thread and its frees the memory after doing it immediately it changes the priority to 1, so main thread it will continue.

How to check whether the thread is Daemon or not?

public boolean isDaemon() ⇒ To check whether the thread is "Daemon".

public void setDaemon (boolean b) throws IllegelThread State exception

b → true, means the thread will become Daemon, before starting the thread we need to make the thread as "Daemon" otherwise it would result in "Illegal Thread state Exception".

What is the deafault nature of the Thread?

Ans:- By deafault the main thread is "Non Daemon". For all remaining thread Daemon nature is inherited from parent to child, that is if the parent thread is "Daemon" then child thread will become "Daemon" and if the parent thread is "Non Daemon" then automatically child thread is also "Non Daemon".

Is it possible to change the Non Daemon nature of main Thread?

Ans:- Not possible to change the Non-Daemon nature of main thread & starting is not in our hands, it will be started by 'jvm'.

eg:: Class MyThread extends Thread { }

```
public class Test { }
    public static void main (String[] args){
        System.out.println (Thread.currentThread().isDaemon());  //false
        Thread.currentThread().setDaemon(true); //RE: IllegalThreadStateException.

        MyThread t=new MyThread();
        System.out.println (t.isDaemon()); //false
        t.setDaemon(true);
        t.start();
        System.out.println (t.isDaemon()); //true
    }
}
```

Note. whenever last Non-Daemon threads terminates, automatically all Daemon Threads will be terminated irrespective of their position.

eg:: makeup man in shooting is a DeamonThread.
hero is main thread.
if hero role is over, then automactically the makeup role is also over automactically.

eg:: class MyThread extends Thread {

```
    public void run(){
        for (int i=1; i<=10; i++) {
            System.out.println("Child thread");
            try {
                Thread.sleep(2000); //2sec
            }
            catch (InterruptedException e){
                System.out.println(e);
            }
        }
    }
}
```

```
public class Test {
    public static void main (String[] args) {
        myThread t=new myThread();
        t.setDaemon(true); //Stmt-1
        t.start();
        System.out.println("end of main thread");
    }
}
```

<u>output</u>:- if we comment stmt-1 then both the threads are Non Deamon threads it would continue with its execution.

end of main thread
Child thread
Child thread
...

<u>output</u>: if we remove comment on stmt-1, then main thread is Non-Daemon thread where as user defined thread is DaemonThread, if the main thread finishes the execution then automatically the DeamonThread also will finish the execution.

---

xxx

→ <u>green Thread</u> - only with JVM (without taking OS suppm

→ <u>Native os model</u> → taking suppst <u>OS</u> and <u>JVM</u> - life lifecycle of thread many jum takin suppst by os unda OS.

→ to start - t.start

→ to stop → t. if I call it will enter into dead state)

→ by using stop method of thread class

→ suspend() and resume() →

12-12-2022 Java 1.2 Hyder Sir

① day

Collections in Java

framework/API /-

→ To store data small data create variable name and we are storing

ex: int age; variable.

string name;

→ if data is large value it is difficult to create so many variables.

→ So we have array (large data ~~in sig~~ in single variable).

* single var name
* large volume of data
* indexed Data Structure.

→ store only similar data / Homogenous data.

→ Can array store mixed data (Hetergenous data) (no) - Answer is no.

→ disadvantage of array

* Size is fixed
* it demands contigous memory locations.
* no inbuilt class (no methods)
* because of this array had limitations

inbuilt (vector) class are legacy class

↳ not maintaining uniformed data structure.

→ Java happends to be open source programming language.

→ Joshva – creating hierarchy of Collection interfaces and class.

D.S

↓

inbuit methods

he gave name as Collections. → Sunmicrosystem - JDK 1.2

they collection in API

→ we have to understand and use it

don't need to code anything just we are using.

→ collection of major 7 classes :-

(class)	(interfaces)	(package)
① ArrayList	List (I)	util
② LinkedList		
③ Array Deque	Queue (I)	~~java.lang.util~~ ;
④ priority Queue		java.util.*;
⑤ Treeset	Set (I)	
⑥ Hash set		
⑦ Linked Hashset		

map → important and legacy classes.

① ArrayList → List (I) Interface

inbuit in Collection API

ArrayList internally it follows dynamic ~~da~~ Array data structure.

→ Create

ArrayList al1 = new ArrayList();

it is class ✓

inbuit class it is

al1.add(10);
al1.add(20);
al1.add(30); output: [10, 20, 30].
s.o.pln(al1);

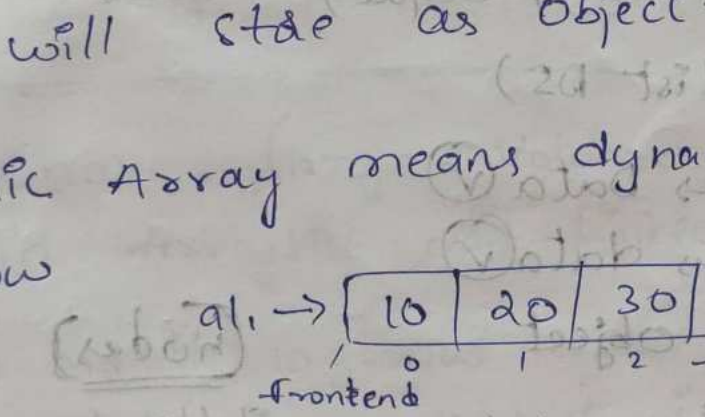
in collection what ever ~~to~~ u add it will store as an "object" - Auto boxing.

→ if data hetergeous it will store as object and in array

→ ArrayList we can store hemogenous also and hetergenous.

→ it will store as object.

→ Dynamic Array means dynamically ~~rear~~ end will grow (here size growing)



→ size will grow based on data
it is grow dynamically not fixed number.

→ Array size fixed it will not grow.
it has guarded with boundaries

→ here utility class not required.

→ Can we copy one collection into another ft new collection (or) existing collection.

we adding and we can able add to frontend middleend and rearend.

→ Size is growing

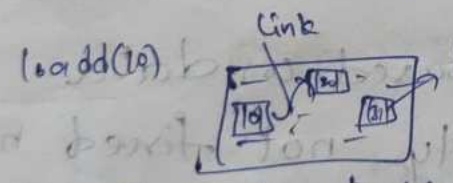
⇒ we are able to add object in ArrayList at any given index.

frontend (✗)
middle (✗) we can add but (✗) not recommend
rear end (✓) — recommended

it is taking time because it is shifting data to next index.

→ <u>LinkedList</u> → (List & deque) - follows doubly LinkedList DS)

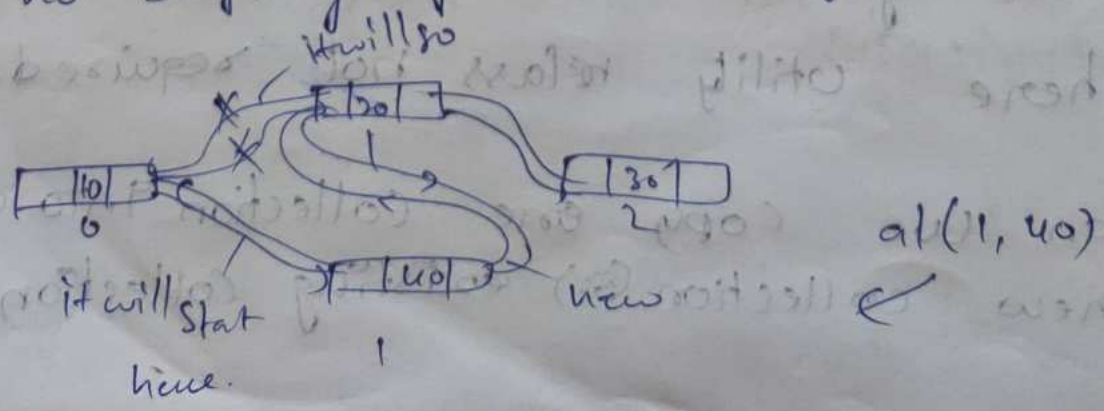
1) Homogenous Data (✓)
2) Heterogenous data (✓)
3) Stored as object [nodes]



Right link to left link (vice versa)

it will link like nodes end to end.

→ here no shifting of index only link will change



Ex:- LinkedList lli = new LinkedList();

lli.add(10);

lli.add("iNeuron");

lli.add(20);

System.out.println(lli); → output : [10, ineuron, 20]

lli.addFirst("Vijay");

S.o.p(lli); → output : [vijay, 10, ineuron, 20]

lli.add(3, "Bengaluru");

lli.addLast("Richmond town");

System.out.println(lli); → [Vijay, 10, inuron, Bengaluru, 20, Richmondtown]

→ we have large volume data we can store in ArrayList and LinkedList.

→ Homo and Heterogenous also

→ every data as Object it will store.

→ index based adding is allowed

→ in array index based adding possible but ⊗ valid

→ if we want to add firstend, lastend at given position LinkedList is recommended

→ linked list is more efficient than ArrayList in data structures.

→ we can make use disprox memory location in linked list

→ After these class is array concept gone? Outdated?

- Answer is "no"

when we have storing large data

Arrays | Collection: (ArrayList, LinkedList)

→ when to use array over ArrayList?

Ans when ever size of the data is known

if you are sure that data is homogeous/similar then you must go with array.

→ array is faster than array list

ArrayList → collection object } internally Autoboxing it takes time compare to array

primitive - object → new instance has created

array → primitive data object also it is fast

→ In the application when you know homogenous and size is fixed.

→ In the application when you don't know size and heterogeonous then go with ArrayList.

→ At given position to performe insertion to use LinkList.

Array List & Linked List

→ duplicates are allowed in both

like l1.add(10); and l1.add(10);

→ when we are using inbuilt method.

① that inbuilt method what type of data it is accepted.

② that inbuit method give me what.

③ when we any inbuilt method it will show what type and return)

-① alu.contains(44); – it will give wheather 44 is ArrayList contains.

methods

→ boolean res = alu.contains(44);

System.out.println(res); //true

④ System.out.println – align we will use debugging purpose.

① * Capacity – How much you have

③ .indexOf(22) – index * Size – How much you utilized

④ .isEmpty(); – boolen checking is it empty

⑤ .ensureCapacity(10); – ~~size~~ print(size.alu)

⑥ .trimToSize(); – removing extra size.

⑦ getClass(); – entire qualify name we will get

⑧ .clear(); it will clear data – size 0

⑨ Conclusion point:

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

generics

only integer will store. not other type of data.

→ ArrayList → large Data ✓
→ Both Hemoand Heter ✓
→ Dynamically DS → ✓
→ Indexed Based accessing ✓
→ Duplicates allowed → ✓
→ efficient insertion at rear end → ✓
→ List (I)

⇒ Using inbuilt methods ⇒ ~~lang~~ can we perform ✓

→ LinkedList → doubly linked List
↳ Nodes & links with previous nodes & next node

→ Can we perform insertion at any given position → ✓

→ one collection another collection ✓

→ store data as object ✓

→ List (I) & Deque (I) ✓ – Duplicate allowed, index based access allowed

→ Extra methods in linked list.

<u>Common method in Linked list</u>

$ll_1 = \{10, 20, 30, 40\}$ (first, last; 0, 1, 2, 3)

① ll1.addFirst();
② getFirst(); → it will get first object
③ getLast(); → it will get last object
④ getIndex(0) → it will get index
⑤ getLastIndex(0) → it will get last index of same data

112. OfferFirst(10); } ① - it will add first 10
112. OfferLast(55); → it will add last 55 - it may reject

addfirst } ② ①and② similar working
addlast
- it should it will get add.

→ peek(); and poll(); - both are fetching data.

peekFirst(); - it will get first collection of data.

pollfirst(); - it will get first collection of data that element will remove from collection
it will take and give you

→ Array Deque - Dynamic only (doubleend Queue)
internally ended Queue

Queue → DS ⇒ FIFO - first in first out

index based accesing is not allowed ⊗

insertion will happen rearend also frontend also.

both end will do insertation

addfirst(), addlast(); ⇒ ✓ add(index, obj);

duplites are allowed ✓

middle we can't add ⊗

→ Deque (Interface).

- collection, maps, generics, enum annotation inner class package.

→ Order of insertion is maintained in above 3 class

13/12/22 (2 day)

priority Queue : → ① It follows min-heap Data Structure

(Queue (I) – it implements

import java.util.*; – All packages.

* index based is not allowed.
* add (object), add (collection). } methods present in
* add (100); , addAll (all). } priority queue.
* output will not get as order of insertion.
* In entire collection which ever highest object that will be avaliable at begining.

```
pq.add(70);
pq.add(60);
pq.add(50);
S.o.p(pq)
```

output:
[50, 70, 60]

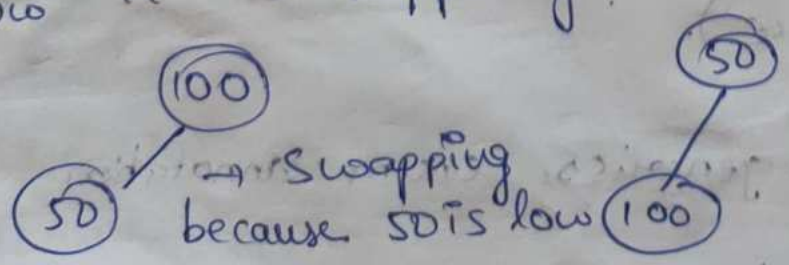
⇒ min-heap

→ 100, 50, 150, 25, 75, 125, 175 input →

→ 25, 50, 125, 100, 75, 150, 175 output

How it is happening?

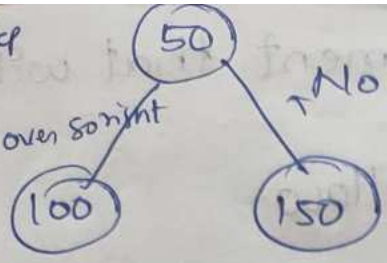
① ⇒



adding of data will be left and right.

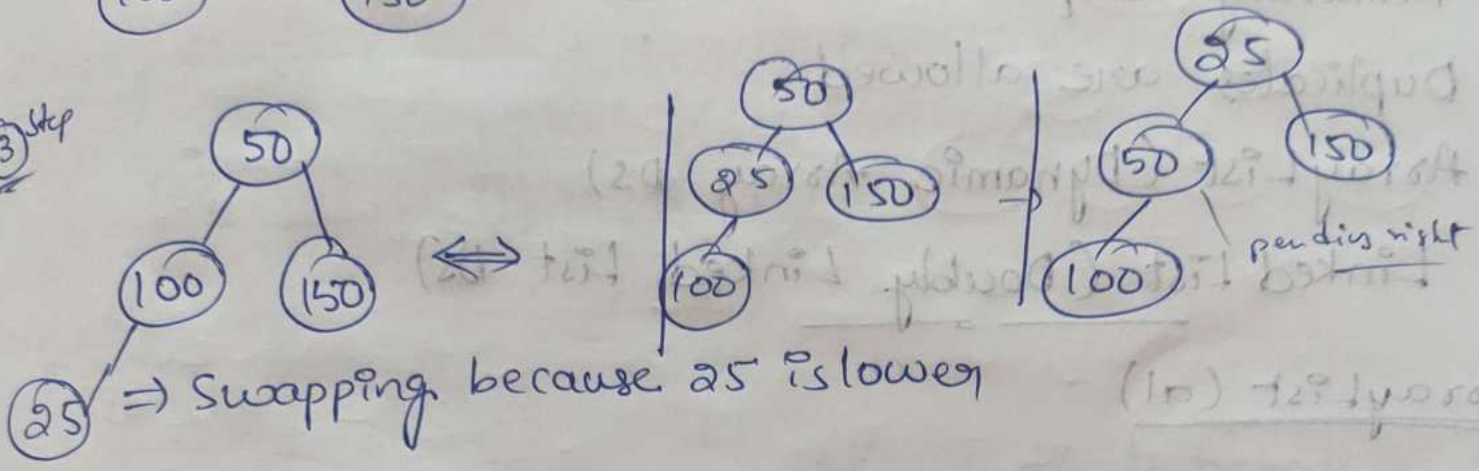
(2) step

left over so right



No swaping because 50 is low than 150.

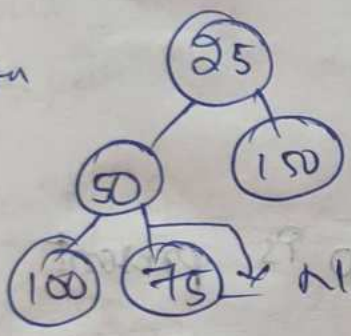
(3) step



pendin right

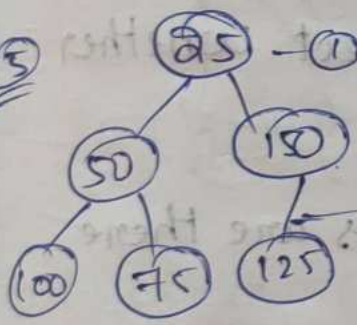
⇒ Swapping because 25 is lower

(4) step



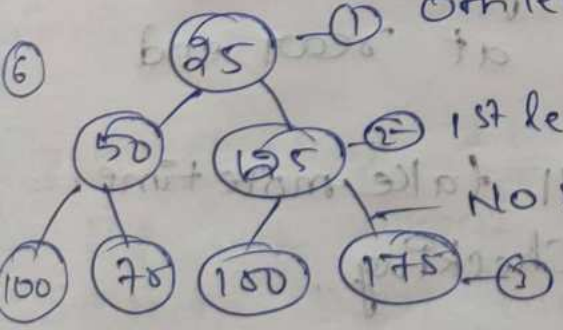
No swapping required because 50 is lower than 75

(5)



swapping because 125 less than 150

(6)



No swapping 175 is greater than 125

Output [0th, 1st, 2nd]

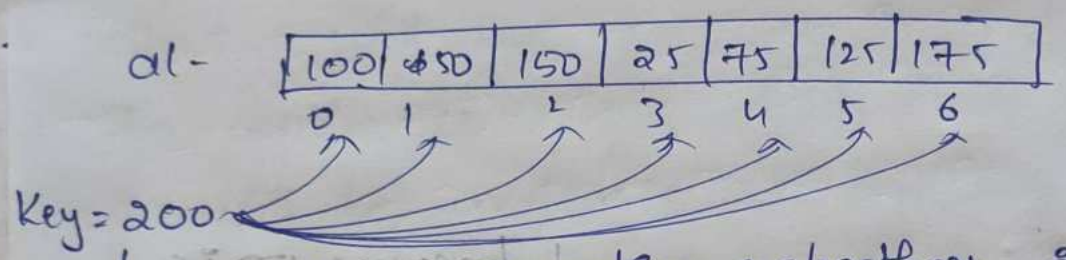
= [25, 50, 125, 100, 75, 150, 175]

* which ever highest priority element that will come at begining.
* Remaining objects will not follow
* Duplicates are allowed.

⇒ ArrayList (Dynamic Array DS)

⇒ Linked List (Doubly Linked List DS)

- ArrayList (al)



(searching operation wheather 200 is present in Arraylist or not

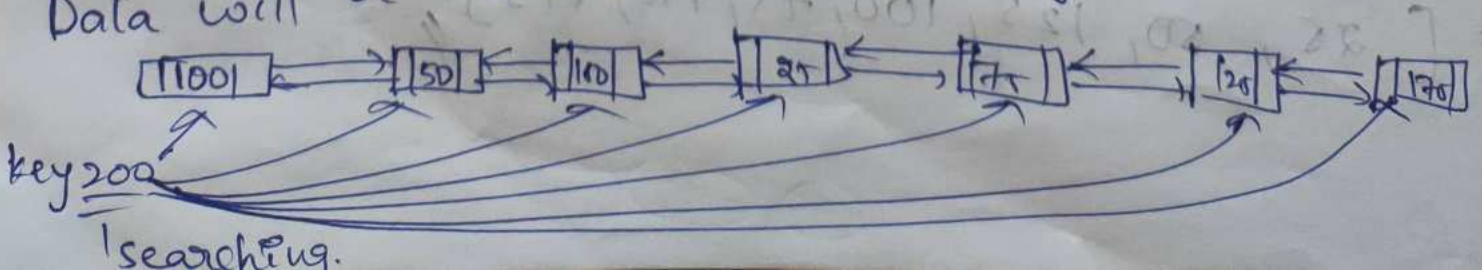
It is going and checking every object whether 200 is present or not.

O(n) → Big of n → how many elements are there that many searching is happening.

* whenever we want insertion at rear end then will go to ArrayList.
* where as searching it will take more time by going every element and checking.

LinkedList (ll) it follows doubly linked list)

Data will be stored in the form of nodes



(searching.

How many elements are there that many comparisons
7 comparisons. O(n) → Big of n.

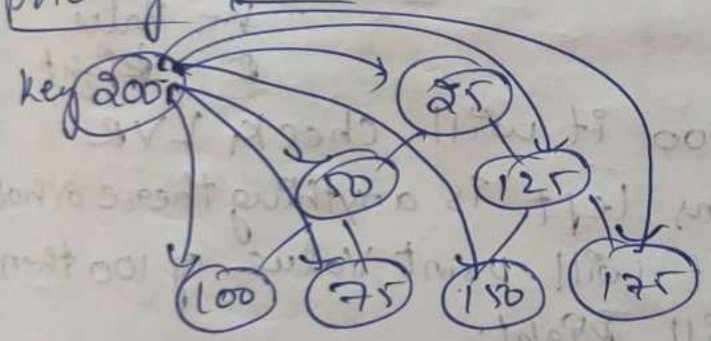
<u>Array Deque</u> (It follows Double ended Queue DS)

0	1	2	3	4	5	6
100	50	150	25	75	125	175

Key = 200
Searching for 200.

How many elements are there that many comparisions. 7 comparisions, O(n) → Big of n

<u>priority Queue</u> :- It follows min-heap



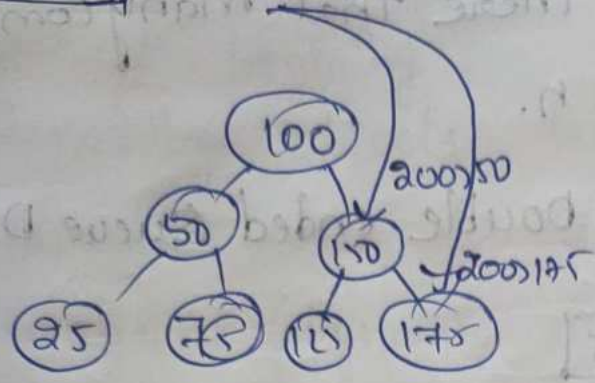
7 comparisions.
Big O of n → O(n)

All above classes are good for insertion whearas coming are taking so much time.

<u>Tree set</u> (TS) It follows Binary Search Tree Ds.
* If we want to get output as ascending sorted order we can go with treeset.

100 50 150 25 75 125 175 → input
25 50 ~~100~~ 75, 100, 125, 150, 175 → output

Key = 200 200 > 100 — Searching



it is balanced Tree

3 comparisions, Searching is fast Compare to other Classes.

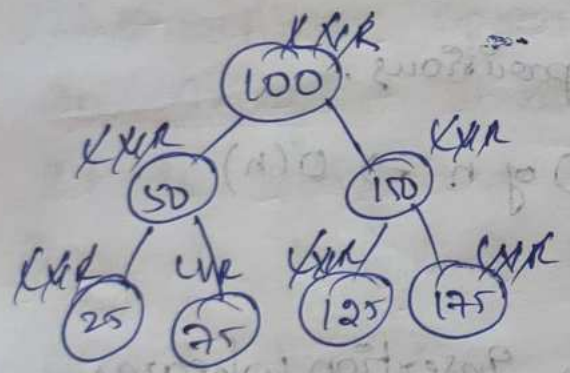
Time Complexity → O (log₂ n)

① If value is lower it will go to left, if it is higher it will go Right.

② adding elements firstly it will check the element is lower or not. than it will go left and right.

③ In case of priority Queue: directly element will go left and Right after swapping will happen based on priority

It follows In order means LVR means L - Left
V - Value
R - Right.

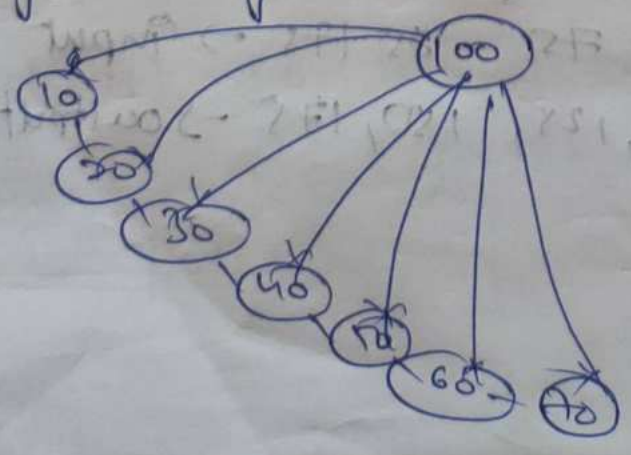


In 100 it will check LVR means left is anything there or not then it will print Value of 100 then it will Right.

25 50 75 100 125 150 175 → we will get output.

input: 10, 20, 30, 40, 50, 60, 70

↳ apply Binary Search Tree



This type of Tree called Skewed Tree

7 objects
7 comparisions
time Complexity O(n).

if we store the data by following Treeset it will follow Binary search Tree.

if we are using Tree set class to store data as sorting order.

If tree follows Balanced binary tree and store output then searching is faster.

If tree follows skewed tree and store output then searching is slower. so we should not go searching here.

inbuit methods :- I/P ⇒ 100, 50, 150, 25, 75, 125, 175.

ts.ceiling(150); //150 →
ts.upper(50); //75 →

Ceiling method if we search for any obj if that object is present that obj will give you same obj

ts.floor(50); //150 →
ts.lower(50); //25 →

Floor method if we search for any obj if that object is present that obj will give you same obj.

ts.ceiling(40); //50 →
ts.higher(40); //50 →

Ceiling method if we search for any obj if that obj is not present then it will give you higher obj

Higher method will give you higher obj

ts.floor(40); //25 →
ts.lower(40); //25 →

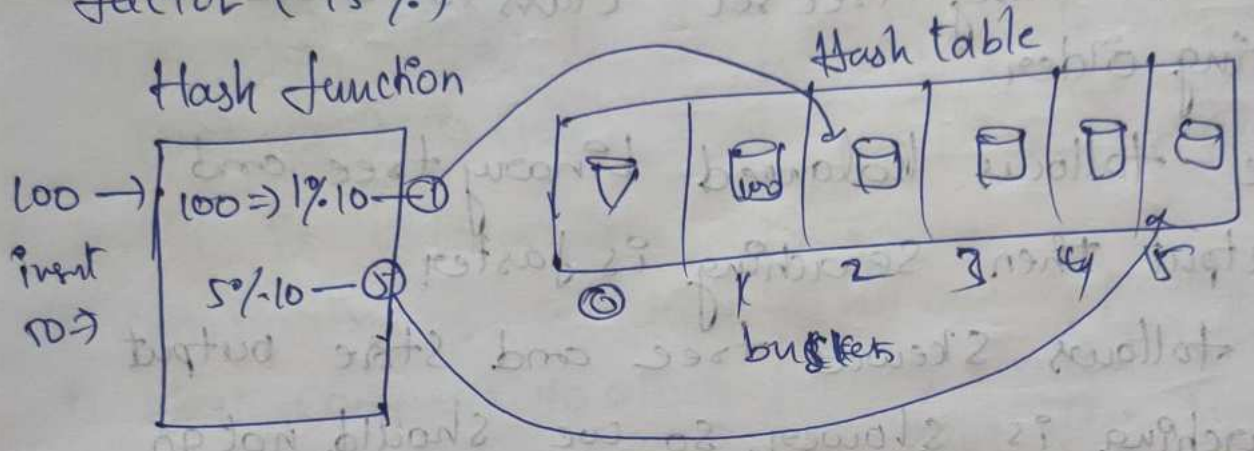
Floor method if we search for any obj if that obj is not present it will give you lower

it will give lower value.

↓

Hash set :- (It follows hashing algorithm)

⇒ Hash function & associated hash table with load factor (75%)



Because of formula which is present in hash function. Hash function and Hash table works that if we insert any value that will go and sit at specific position in Hash table because of formula present in Hash function

75% means In hash table if suppose 100 buckets are there in that 75 buckets are filled then that hash table will become double.

* Searching is very very fast in Hash set
* Index based is not allowed.
* Output will not get as insertion and it will not sort and highest priority not follow

Linked Hash set :- (It follows hashing algorithm

It very fast for searching.

* output will get as insertion
* index based is not allowed.

→ How to access the values present inside collection?

```
ArrayList al = new ArrayList();
    al.add(10); // Autoboxing is happen to store
    al.add(20);   primitive to object.
    al.add(40); // al.add(new int(40));
    al.add(60);
    al.add('c');
    al.add("vijay");
ArrayList al2 = new ArrayList();
    al2.add(10);
    al2.add("vijay");
    al2.add(30);
    al2.add(50);
    S.o.p(al2); [10, vinay, 30, 50] → just to print value.
```

3 ways are there to access: index based is allowed only for List(I) classes)

// loop normal => Here travelling

```
for (int i=0; i<al2.size(); i++) {
    Object o= al2.get(i);
    S.O.p(o);
    (or)
    S.O.p(al2.get(i));
}
```

// for each

```
for (object obj : al2) {
    S.O.p(obj);
}
```

// <u>iterators</u> al2 → | 10 | 20 | 30 | 40 | 50 | 60 |

One method is common for all 7 classes is iterators. (if we want to accesses data for all 7 classes we can go with iterators).

Iterator itr1 = al2.Iterator(); → whenever we call this iterator is active.

if (itr1.hasNext() == true) {

↳ It will ask iterator anywhere is there or not

itr1.next();

↳ It will go to location and give to us

}.

```
while (itr1.hasnext()) {
```
→ How many Values are there that much time body should print.

```
obj o = itr1.next();
```
(or)

```
Integer i = (Integer) itr1.next();
S.o.p (itr1.next());
}
```
→ just for print

It will fetch in reverse direction also. only in ArrayList, Linked list. for only reverse.

```
al2.listIterator(al2.size());

ListIterator litr = al2.listIterator(al2.size());
while (litr.hasprivious()) {
    S.o.p(litr.previous());
}
```

// what if i want to reverse or access data of other class in reverse direction?

```
ArrayDeque ad = new ArrayDeque
    ad.add(10);
    ad.add(20);
    ad.add(30);
    ad.add(40);
```

index based insertion and accessing is not allowed expect ArrayList and LinkedList.

```
Treeset ad = new Treeset();
    ad.add(10);
    ad.add(20);
    ad.add(30);
    ad.add(40);
Iterator it = ad.iterator();
    while (it.hasNext()==true){
    Integer i = (Integer)it.next();
    S.o.p("Array Deque" +i);
  }
linkedlist ll=new linkedlisk().
        ll.addAll(ad);
    S.o.p(ll);
ListIterator l1 = ll.listIterator(ll.size());
    S.o.p("Accessing treeset data");
  while (ll.hasprevious())
  {
  S.o.p(ll.previous()); we will get reverse direction
  }
```

———— o ————

14/12/22

(3) day

Iterator :- through a large volume of data in that data when we are traveling while travelling we need to fetch (get) the data in that case iterator comes into picture

```
Iterator itr = al2.iterator();
```

L, collection name

hasNext(); → Iterator will check whether next value is there or not. next()

al → [10|20|30|40|50]

itr at begining of collection

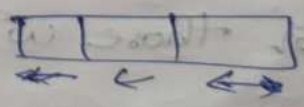
nex(); → iterator will go to next position and fetch (get) the data also

Everything is store as object.

LinkedList, ArrayDeque, TreeSet :- descending order iterator is there.

```
LinkedList ll = new LinkedList();
    ll.add(10);
    ll.add(20);
    ll.add(30);
    ll.add(40);

    ll.descendingIterator();
```



```
Iterator dItr = ll2.descendingIterator();
while (dItr.hasNext()) {
    S.O.P (dItr.next()); (or) Integer i = (Integer) dItr.next();
}
```

Before actual java collections they are lacking uniformly.

Anamerator → before iterator

```
Vector v= new Vector();
    v.add (10);
    v.add (20);
    v.add (30);
    v.add (40);
```

Mentors for this handwritten notes :

Hyder Abbas (linkedin.com/in/hyder-abbas-081820150/)
Nitin M (linkedin.com/in/nitin-m-110169136/)
Navin Reddy (linkedin.com/in/navinreddy20/) (Telusko)
wrttien by pallada vijaykumar
@ineuron.ai

if we want access.

```
v.elements();
```

→ it is working as Iterator

```
enumeration em = v.elements();
```

→ iterator method Same as iterator();

```
while (em.hasMoreElements()) {
    s.o.p (em.nextElement());
}
```

Same like iterator, iterator present in 7 classes. Before collection topic Vector is also there. in vector enumeration is present it is working same as Iterator.

why?

Before actual collections upto time Vector in there means upto 1.2 versions Now also Vector is there. if they remove vector then previous projects build by Vector those will gone So thats why still vector is there.

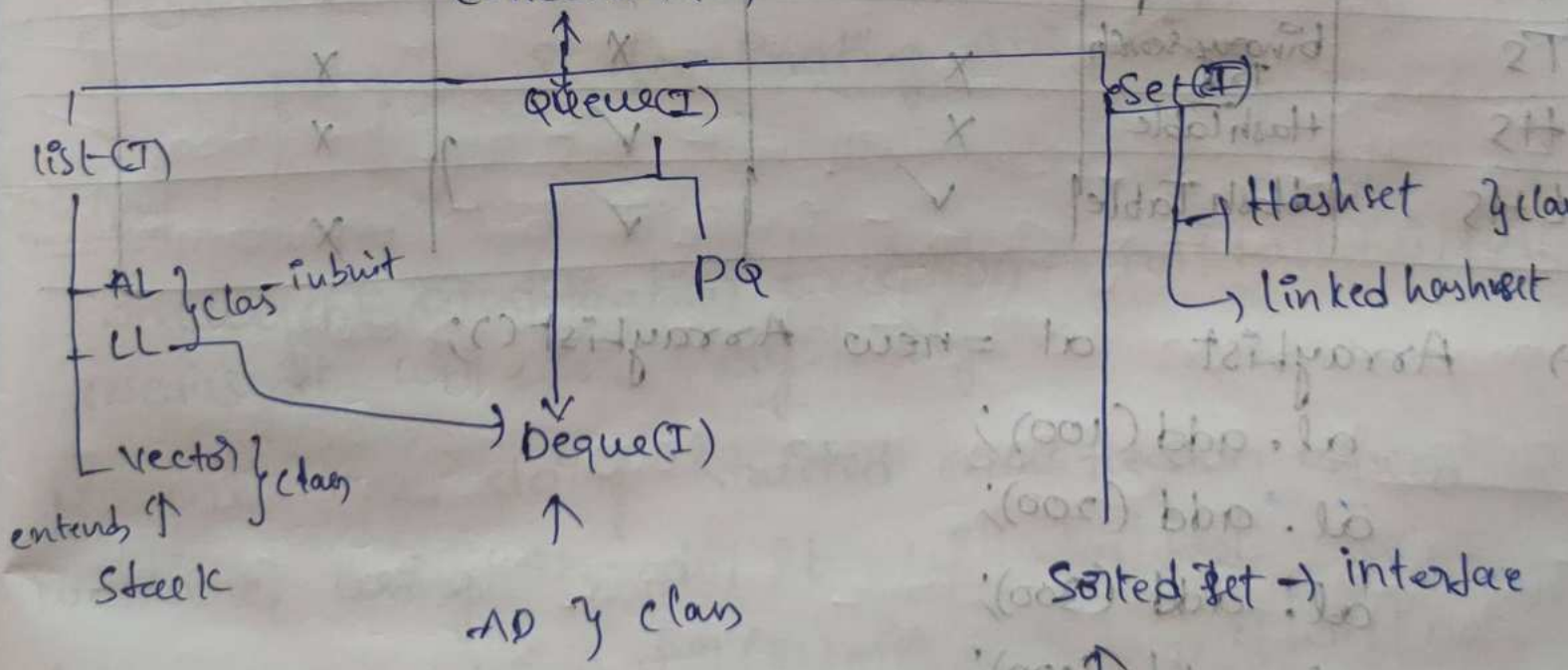
***

classes	index forloop	foreach	Iterator	list iterator	descend iteration	Enumerator
AL	✓	✓	✓	✓	X	X
LL	✓	✓	✓	✓	✓	X
AD	X	✓	✓	X	✓	X
PQ	X	✓	✓	X	X	X
TS	X	✓	✓	X	✓	X
HS	X	✓	✓	X	X	X
LHS	X	✓	✓	X	X	X

***

Hirearcy

(I -> Interface)



List ll = new ArrayList (); loose coupling

Queue q = new ArrayQueue ();

List l1 = new LinkedList ();

Deque D = new LinkedList ();

d.peak();

if we → create Deque with LinkedList we can't access all methods present in LinkedList.

Classes	DS	order of insertion	permit null value	Duplicate.
AL	Dynamic	✓	✓	✓
LL	doubly linked	✓	✓	✓
AD	double ended queue	✓	✗	✓
PQ	min-heap	✗	✗	✓
TS	binary search Tree	✗	✗	✗
HS	HashTable	✗	✓	✗
LHS	Hash Table	✓	✓	✗

→ ArrayList al = new ArrayList();

```
al.add(100);
al.add(200);
al.add(300);
al.add(400);
for (int i=0; i<al.size(); i++){
    S.o.p(al.get(i));
    al.add(10);
```

→ we can add this line. program will not stop.

}.

while accessing collection we attempt to modify collection then it is called structed (or) concurrent modification.

if we use structured modification the output will not stop of execution.

if we use for or for each loop to access collection and then we are using structured modification then output will not stop (or) never fail.

<u>Now by using iterator</u> :- (It is recommended to use iterator for accessing not "for" loop);

```
Iterator i=al.iterator();
while (i.hasNext()){
    s.o.p(i.next());
al.add(12);  — structural modification.
}
```

program is terminating Abruptation.

<u>output</u>

Concurrent modification Exception → fail fast fail fast means it will fail very fastly.

if we are doing structed modifiection then, we are using iterator it will failfast.

if we are using for (or) for each loop it will not stop output (or) it will not fails.

— o —

Fail Safe :- if we are doing Structural modification then concurrent modification Exception should not generate So then we can't use ArrayList (or) normal classes.

We have Some Specialpackages we should use those

```
CopyOnWriteArrayList cal = new CopyOnWriteArrayList();
 cal.add (1000);
 cal.add (500);
 cal.add (2000);
 cal.add (3000);
Iterator ii = cal.iterator();
   while (ii = hasNext()) {
      S.o.p ( ii.next());
      cal.add (10);
   }
```

In this case we will not get any exception (or) not getting Output infinite. but concurrent modification will not allow. Very slowly.

output :- 1000, 500, 2000, 3000

In all above three methods its not possible to modify while access.

In normal for loop (or) for each loop we do Structural modification output leads to infite-

In iterator while using normal ArrayList we do structural ~~while using normal~~ modification output leads to Concurrent modification exception - fail/fast.

In iterator while using CopyOnWriteArrayList we do structural modification output will get safety but it will not empty (or) not leads to exception (or) not leads to infinite output it is called Fail Safe.

```
TreeSet ts = new TreeSet();  (Binary Search Tree)
  ts.add(100);
  ts.add(50);
  ts.add(150);
  ts.add(75);
  ts.add(125);
S.O.P(ts)  50 75 100 125 150  sorted order
ArrayList ts = new ArrayList();
  ts.add(100);
  ts.add(50);      (iwant to sort present
  ts.add(75);       in Arraylist();
  ts.add(150);
Collections.sort(ts);
  S.O.P(ts);
```

Collection → It is interface / framework
Collection → It is Class (individual)

```
ArrayList al2 = new ArrayList();
al2.add(10);
al2.add("Vijay");
al2.add(566);
al2.add("kumar");
Collections.sort(al2);
S.O.p(al2);
```

output
Exception.

- we can store hetergenous data
- Now we are trying to store. we will get exception.
- whenever data is homogenous we can go with collections.sort (a)
- whenever data is hetergenous. we can go with comparator. (or) Comparable interface.

To avoid exception in above progsam we can use generics, means data should same type.

```
ArrayList <String> al = new ArrayList<String>();
```

But we can't add multiple data with different type.

```
ArrayList <Integer> al = new ArrayList <Integer>();
al.add(10);
al.add(20);
```

It accepts only integer we are restricting our arraylist to store only Integer type.

tri.

```
Class Student {
    int age;
    int marks;
```

```
public student (int age, int marks){
    this.age = age;
    this.marks = marks;
}
public int getAge (){
    return age;
}
public int getMarks(){
    return marks;

main() {
    student s1 = new student (12, 75);
    student s2 = new student (13, 76);
    ArrayList al = new ArrayList ();
    al.add (s1);
    al.add (s2);
    collections.sort (al);
    S.O.P (al);
```

sorting can't be done for complex objects.

few more important inbuilt methods of collections class

```
ArrayList al = new ArrayList();
    al.add(10);
    al.add (20);
    al.add (40);
    al.add (50);
int i = collections.binarySearch (al, 60); → it will give index.
```

S.O.p (i)

Collections . shuffle (al); → we can shuffle our Collection

S.O.p (al);

Collections . min (al4);

. max (al4);

. frequency (al4, 40);

(it will check how many times 40 is there in collection.

14/9/22 Day 4

All classes under collection hierarcy /API/ framework has its own specially (features, interna DS) → on obj/elements present using methods we can perform.

Identify card ⇒ passport / Aadhar. Card

name : Vijay } many number of vijay's are there in india
Age : 23

Aadhar number ⇒ aadhar No } 2 types (Data name, age, father nam). these data may be duplicate

↓

it is unique

No have same aadhar no.

Data is present in form of pairs ⇒ Key - Value

key → Associated data/value

↳ By using aadhar number they are able to see our data (name, age, father name)

these type of data will store in map (key-value)

map (key, value) (1.2V)

→ Hash map → widely used → this is class under map interface

→ Hash table

→ properties

Sometimes we go with Treemap (class) under map interface.

if we hear that data is present in form of pairs then we need to store means we should go with map interface.

⇒ if we have simple data

Key: Value

10: Sachin

7: MSD

18: Kohli

key-value we go with map Concept.

each pair of key-value is called entry.

Key should be unique → No duplicates.

Value → need not be unique → Duplicates are allowed Both together call entry.

```
interface Map {
    interface entry {
        ... some methods
    }
}
```

→ Interface inside Interface is allowed.

entry is tightly coupled with map means as long as map is present upto that entry is able to access map.

In map also data is stored as object Hashmap it follows hashing algorithm.

```
HashMap Hp = new HashMap();
    Hp.put (Key: Value);
    Hp.put (10, Sachin);
    S.o.p(Hp); //output null
    Hp.put (7, MSD);
    Hp.put (18, Kohli);
    S.o.p (Hp);
```

<u>output</u>:- we are getting in Key-value

{18 = Kolhi , 7 = MSD , 10 = Sachin}

The output is not getting as order of insertion

```
LinkedHashMap Lhm = new LinkedHashMap();
    Lhm.put (10, Sachin);
    Lhm.put (7, MSD);
```

output :- {10 = Sachin , 7 = MSD}

The output getting as order of insertion

LinkedHashMap is Subclass of Hash Map

Hashmap => jdk 1.2 V

LinkedHashmap => jdk 1.4 V

complex data

```
class Student {
  String name;
  int marks;
  int age;
  String city;
public Student (String name, int marks, int age){
    this. name = name
    this. marks = marks
    this. age = age;
  }
public String getName(){
    return name;
  }
  public int getMarks(){
    return marks;
  }
  public int getAge(){
    return age;
  }
  public String toString(){return name+" "+marks+" "+target+" ";}
}
```

```
main(){
  Student s1=new student("vijay",98,22);
  "     s2= "     "     ("Vannesh",98,20);
  "     s3= "     "     ("Vijay",98,23);

Hashmap  hm = new HashMap();
  hm.put(1,s1);
  hm.put(2,s2);
  hm.put(3,s3);
S.o.p (hm); output : we got hash code like 1=
                                  student@973142
                                  hashcode
int a=10; primitive
S.o.p (a); //10 → String.
  ↳ like method is going to call tostring();
                  → tostring()
⇒ S.o.p (Object ref); hashcode address of Object.
  S.o.p (object ref); Values will get
```

if we are not overriding tostring() we will get hashcode. it is overriden then we will get values.

How to get values present inside HashMap.

Now below are for Student Class present in previous page

```
Collection col = hm.values();
  Iterator itr = col.iterator();
```

```
while (itr. hasNext()) {
    S.o.p (itr.next());
}
```

<u>output</u> Values will get

Vinay 98 22

Vannesh 98 20

Vijay 98 23

```
Set s= hm. KeySet();
Setkey. iterator -> Iterator itr = s. iterator();
while (itr. hasNext()) {
    S.o.p (key);
}
```

<u>output</u> key will get

1, 2, 3

```
Set s = hm. entry set ();
Iterator itr = s. Iterator ();
while (itr. has next()) {
    S.o.p (itr. next());
}
```

<u>output</u> :- Values and key we will get.

1. Vinay 98 22
2. Vannesh 9820
3. Vijay 98 23.

```
Collection c = hm.values();
  c.iterator(); -> Iterator itr = c.iterator();

                                  Hashmap hm = new hash Map
                                  hm.put(10, "Vinay");
                                  hm.put(22, "Vijay");
while (itr.hasNext()) {
                        — Stri
  String s = (String) itr.next();
    S.O.P(s);
  S.O.P(itr.next());
}
```

<u>Output</u> :- only we are getting values

Vinay
Vijay

```
Set s = hm.Keyset();
  S.iterator(); -> Iterator itr = S.itoral();
  while (itr.hasNext()) {
  Integer i = (Integer) itr.next();
      S.O.P(i);
  S.O.P (itr.next());
}
```

Output :- we will get only Keys.

10, 12

```
set   s = hm.entry set();
Iterator itr = s.iterator();   Map.Entry e = (entry) itr.next();
  while (itr.hasNext()) {
    S.O.P(itr.next()); S.O.P(e);
}
```

Output :- Both key-value we got
[10 = vinay , 12 = vijay]

Above all three method output for

```
HashMap hm = new HashMap();
hm.put (10, "Sachin");
hm.put (12, "vinay");
hm.put (13, "vijay");
```

HashMap 1.2V	LinkedHashMap 1.4V
DS : Hashtable	DS : Hashtable + doubly LinkedList
order of inseration will not get output same	Same output as order of insertion.
parent of LinkedHashMap	Sub class of Hash Map.

Day 5 15/12/22 passport program

```
class person {
  String name;
  int age;
public Person (Stringname, int age){
    this.name = name;
    this.age = age;
}
```

```
public String getName(){
    return name;
}
public int getAge(){
    return age;
}
public String toString(){
    return name+" "+age+"";
}

main() {
  person p1 =new person("Vinay", 22);
  person p2 =new person("Mannesh", 20);
  HashMap hm = new HashMap();
      hm.put(111, p1);
      hm.put(222, p2);

Scanner scan =new Scanner(System.in);
  S.o.p ("please enter passport No");
    Integer K= scan.nextInt();
    Bodean flag =false;
Set set = hm.entrySet();
Iterator itr = set.iterator();
  while (itr.hasNext()){
  Map.Entry passport=(Entry)itr.next();
Integer key = passport.getKey();
              (Ingeter) Casting.
```

```
    if (k==key){
s.o.p ("Into "+ passport.get Value());
  }
}

if (flag ==false){
   s.o.p ("please passport Not found");
}

class Employee {
  int empid;
  String ~~empid~~ name;
  String ~~ename~~ eadd;
public String toString(){
        return "Hyder";
}
public void finalize() {
So. p(" garbage collecter collected");
  }
main(){
employee e=new Employee();
  HashMap hm= new Hashmap();
  e =null; hm. put (e,"Hyder");

  Systemgc();  s.o.p ("Last line");
```

internally gc internally calling finalize method to clear object where no one is referring.

Output

last line

HashMap is dominating garbage collector because hashMap is using employee reference.

```
import java.util.*;
class Employee {
  String name;
  int age;
public void finalize(){
  S.o.p ("Garbage Collected");
}
}
main()
Employee e=new Employee();
WeakHashMap hm = new WeakHashMap();
  hm.put (e, "Vinay");
    e=null;
  System.gc();
  S.o.p ("last line");
}
```

output :-

Garbage Collected

last line

ge is dominating weak HashMap

Hash Table

```
Hashtable ht = new Hashtable();
    ht.put (1, "vinay");
    ht.put (2, "vijay");
    S.O.P (ht);
```

Difference Hash table & Hash Map

* Hash Map java 1.2
* Hash table java 1.0 → Multithreading is not possible
* Hash table all methods are Synchronized.
* Hash Map methods are non-Synchronized (multithreading is possible
* Hashtable → Thread Safe.
* Hash Map → Thread not Safe.
* Hash table → low performance
* Hash Map → High performance

Hash Map always go.

```
Tree map tm = new TreeMap();
    tm.put (1, "vinay");
    tm.put (2, "vijay");
    S.O.P (tm);
```

Interface (till java 8 from alpha Version)

```
interface Vinay {
    Void teaches ();
    Void Writescode ();
}
```

we can't give implementation for methods which are present inside interface

```
Class Student implements Vinay {
    public void teaches () {
        S.o.p ("Java");
    }
    public void writesCode () {
        S.o.p ("writing Java");
    }
```

from java8 :- Inside interface a method with body is acceptable

```
interface Vinay {
    Void teaches ();
    default
        Void disp () {
        S.o.p ("Hi");
    }
Class Vijay implements Vinay {
    public Void disp () {
        S.o.p ("Hello");
}}
```

we can give implementation for method then we should compulsory use default keyword.

//we can overside the method.

```
interface Vinay {
    Static Void disp() {
        S.O.p ("Hello");
    }
}
```

we can have Static (a) default (n) whose methods have body inside interface.

But if implement interface, if a Method is present inside interface with Static it will not inherited. if we want to print static method. we can create reference of interface. Vinay.disp();
-) we will get output.

<u>Java a features.</u>

```
interface Vijay {
    private Void disp3()
    {
        S.O.p ("private method");
    }
}
```

// private methods are not accessing outside interface a class. But we can use inside interface.

Generics

→ 1. Agenda
2. Type-safety
3. Type-Casting
4. Generic Classes
5. Bounded Types
6. Generic methods and wild card character(?)
7. Communcition with non-generic code
8. Conclusions

Deff: The main objective of generics is to provide Type-safety and to resolve Type-casting problems.

Case 1: Type-Safety

Arrays are always type safe that is we can give the guarantee for the type of elements present inside array.

for example if our programming requirement is to hold String type of objects it is recommend to use String array

In the case of String array we can add only String type of objects by mistake if we compile time error.

eg: String name [ ] = new String [500];

name [0] = "Navin Reddy";

name [1] = "Haider";

name [2] = new Integer (100); // CE:

incompatable types found : java.lang.Integer

required : java.lang.String.

That is, we can always provide guarantee for the type of elements present inside array and hence arrays are safe to use with respect to type that is arrays are type safe.

But collections are not type safe that is we can't provide any guarantee for the type of elements present indside collection.

for example, if our programming requirement is to hold only String type of objects it is never recommended to go for ArrayList

By mistake if we are trying to add any other type we won't get any compile time error but the program may fail at runtime.

eg:- ArrayList al = new ArrayList();

al.add ("Navin Reddy");

al.add ("Haider");

al.add (new Integer(10));

...

String name1 = (String) al.get(0);

String name2 = (String) al.get(1);

String name3 = (String) al.get(2); // Exception in thread "main"

java.lang.ClassCastException:

java.lang.Integer cannot be cast to java.lang.String.

Hence we can't provide guarantee for the type of elements present inside collections are not safe to use with respect to type.

Case 2: Type-casting

In the case of array at the time of retrival it is not required to perform any type casting

eg:-

String name[] = new String[500];

name[0] = "Navin Reddy";

name[1] = "Haider";

...

String data = name[0]; // here typecasting is not required.

But in the case of collection at the time of retrievel compulsory we should perform type casting otherwise we will get compile time error

eg::

```
String name[] = new String[500];
    name[0] = "Navin Reddy";
    name[1] = "Haider";
    ....
    ////
    ....
    ////
String data = name[0]; // here type casting
```

is not required.

But in this case of collection at the time of retrieval compulsory we should perform type casting otherwise we will get compile time error.

eg:

```
ArrayList al = new ArrayList();
    al.add("Navin Reddy");
    al.add("Haider");
String name1 = al.get(0); // CE incompatible types:
found : java.lang.object required. Java.lang.
                                    String.
```

String name = (String) al.get(0); //At the time of retrieval type casting is mandatory.

That is in collections type casting is bigger problem.

To overcome the above problems of collections (type safety, type casting) Sun people introduced generics concept i.e.

1. To provide type safety to the collections.
2. To resolve type casting problems.

To hold only string type of objects we can create a generic version of ArrayList as follows.

```
ArrayList<String> al=new ArrayList<String>();
        al.add("Narin Reddy");
        al.add(10); //CF can't find symbol.
```

Symbol : method add (int) location : class java.uti ArrayList<java.lang.String>

```
al.add(10)
```

for this ArrayList we can add only string type of objects by mistake if we are trying to add any other type we will get compile time error. that is through generics we are getting type safety

At the time of retrieval it is not required to perform any type casting we can assign elements directly to string type variables.

eg: ArrayList <String> al = new ArrayList<String>();
al.add("Navin Reddy");
;;;;;
String name = al.get(0); //type casting is not required as it is an type safe.

That is through generic syntax we can resolve type casting problems

Conclusions :-

1. polymorphism concept is applicable only for the base type but not for parameter type. [usage of parent reference to hold child object is called polymorphism.

eg: ArrayList <String> al = new ArrayList <String>();
List <String> al = new ArrayList <String>();
Collections <String> al = new ArrayList<String>();
Collection <Object> al = new ArrayList<String>(); //CE: incompatiable types

2. Collections concept applicable only for objects, Hence for the parameter type we can use any classes or Interface name but not primitive.

value (type). Otherwise we will get compile time error.

eg: ArrayList<int> al = new ArrayList<int>();

// EE. Unexpected type found : primitive required : pr

19/12/22 monday

Enum : JDK 1.5V introduced.

Constants: Variable → final

↳ Var name ⇒ that name should be in capital

final int data = 10; → we can't change.

DATA = 10; It is recommended to keep in capital which has final variable.

enum → It is group of named constants (or) It is group of predefined constants.

· It refers to enumarted data / our own datatype

⇒ NORTH, SOUTH, EAST, WEST ⇒ Compass

group of similar type of data.

To create enum we use keyword called "enum".

```
enum Result {
    PASS, FAIL, NR; // Static final Result PASS
}
```

(NoResult

(By default

```
enum Gender {
  MALE, FEMALE, OTHERS; // Constanst must be in
}                        capitals.

enum Compass {
  NORTH, SOUTH, EAST, WEST;
}
class Demo {
  public static final int PASS = 35;
//PASS; → It leads to error. inside class we
should declare everything type of data final so on..
}
```

* Outside of class also we can write enum
* Inside of class also we can write enum.

eg:

```
enum Result { //outside class
  PASS, FAIL;
}
class { }
```

eg2

```
class { // Inside class
  enum Result {
    PASS, FAIL;
  }
}
```

In program how many classes → interface / enum are there whenever we compile we will get separate .class file for each class (or) interface (or) enum.

* Inside enum we can have feilds / Instance Variables.

* Inside enum we can have methods and Constructs

```
enum Result {
    PASS, FAIL, NR; // static final
}
main () {
Result r = Result . PASS;
  S.O.P (r);
```

<u>Output:-</u>

PASS (Itself it is a Variable and Value)

Behind the Scene.

// PASS → public static final Result PASS = new Result();, whenever we create enum internal class is generated.

// FAIL → public static final Result FINAL = new Result();

Inside enum if we write Constructor don't make it as public because it is invalid.

```
enum Result{
PASS, FAIL, NR;
 Result(){
s.o.p("Constructor");
}
}
main(){
Result.FAIL.values();
```

(These are methods which are present in Fail class.)

```
↳ Result res[]
for (Result hader : res){
   s.o.p (hader.ordinal());
```

↳ (It will give indexes.)

```
}
}
```

ex:-

```
enum Course {
   JAVA, JEE;
 int courseId;
 Void setCourseId (Int courseId) {
   this.courseId = courseId;
 }
 int getCourse(){
    return courseId;
 }
 Course(){
  s.o.p ("Constructor"); }
```

Inside enum we have constants, variables, constants, methods.

```
main() {
Course.JAVA.SetCourseId(10);
    int cid = Course.JAVA.getCourseId();
    S.o.p(cid);
}
```

Annotations :-

Comments :- This Comments are only for developer after Compiling Comments will gone

annotations :- This → message will give for developers and Compilers and JVM also.

amotation :- It is a extra Information for develope Compilers and jvm.

Annotation :- Annotation → parent of all Annotation

Annotation → Built in
→ Custome (our own).

```
@functionalInterface
interface Trial{
    int getNum();
}
main(){
  Trail t = ()-> { return 10; };
}
```

```
class Java learning {
public void disp()
{
  s.o.p ("parent");
}
}
class focus extendes Java learning {
  @override (Inbuilt annotation)
public void disp()
  s.o.p ("child");
}
}
```

<u>Annotation</u> → we can use to class, interface, methods, fields (variables), local variables, constructor parameters, enums.

Annotation from jdk 1.5V

Can we create our own annotation

```
@ Target (element Type.TYPE);
@ Rentention (Retention Type.Runtime)
@ Interface cricketplayer {
string country name();
                default "India"
int runs();
        default 20000.
}
@ cricket player (countryname = "India" ; runs = 20000)
}
```

@ tells compiler that it is not interface but it annotation is being created

```
Class Virat Kohli {
    int innings;
    String name;
```

Setters and getters are generated.

```
main(){
    Virat Kohli VK = new Virat Kohli();
        VK.setInnings(300);
        VK.setName("VK");
    S.o.p (VK.getInnings());
    S.o.p (VK.getName());
```

(after 10pm nitin taught)

Generics : (1.5 V)

type parameter

```
ArrayList<T> al = new ArrayList<T>();
```

Base parameter.

Generic classes

Until 1.4V a non-generic version of ArrayList class is declared as follows.

Example:

```
Class ArrayList {
    add(object o);
    object get(int index);
}
```

add() method. can take object as the argument and hence we can add any type of object to the ArrayList. Due to this we are not getting type safety.

The return type of get() method is object hence at the time of retrieval compulsory we should perform type casting.

But in 1.5 v a generic version of ArrayList class is declared as follows.

```
class ArrayList<T>{
    add(T t);
    T get(int index)
}
```

Based on our requirement T will be replaced with our provided type.

for example to hold only string type of objects we can create ArrayList objects as follows.

Example:

```
ArrayList<String> l = new ArrayList<String>();
```

for this requirement compiler considered ArrayList class is

Example:

```
class Arraylist<String>{
    add(String s);
    String get(int index);
}
```

add() method can take only string type as argument hence we can add only string type of objects to the list.

By mistake if we are trying to add any other type we will get compile time error.

eg#1 ArrayList<String> al=new ArrayList<String>();
al.add("Navinreddy");
al.add(10); //CE can't find Symbol
Symbol : method add(int)
location : class java.util.ArrayList<java.lang.String>
al.add(10)

eg2 Arraylist<String> al = new Arraylist<String>();
al.add("Navinreddy");

String name = al.get(0); //type casting is required.

Hence through generics we are getting type safety
At the time of retrival it is not required to perform any type casting we can assign its value directly to string variables.

In Generics we are associating a type a type-parameter to the class, such type of parameterised class are nothing are nothing but Generic classes

Generic class: class with type-parameter.

Based on our requirements we can create our own generic classes also.

Example:
Class Account<T>
{ }

```
Account<Gold> g1 = new Account<Gold>();
Account<Silver> g2 = new Account<Silver>();
```

Example:

Mentors for this handwritten notes :

Hyder Abbas (linkedin.com/in/hyder-abbas-081820150/)
Nitin M (linkedin.com/in/nitin-m-110169136/)
Navin Reddy (linkedin.com/in/navinreddy20/) (Telusko)
wrttien by pallada vijaykumar
@ineuron.ai

```
class Gen<T> {
    T obj;
    Gen (T obj){
        this.obj = obj
    }
    public void show(){
        System.out.println("the Type of object is :"+obj.
                                    getclass().get
                                    Name());
    }
    public T getObject(){
        return obj;
    }
}
class Generics Demo {
    public static void main(String[] args){
        Gen<Integer> g1 = new Gen<Integer>(10);
        g1.show();
        System.out.println(g1.getObject());
    Gen<String> g2 = new Gen<String>("ineuron");
        g2.show();
        System.out.println(g2.getObject());
    Gen<Double> g3 = new Gen<Double>(10.5);
        g3.show();
    }} System.out.println(g3.getObject());
```

Output :

The type of object is: java.lang.Integer 10
The type of object is: java.lang.String ineuron
the type of object is: java.lang.Double 10.5

Note To get the underlying object of any reference type we use a method called ref.getClass().getName()

eg1:

```
interface Calculator{}
class Casio implements Calculator{}
class Quartz implements Calculator{}
class Kadio implements Calculator{}

Calculator c1 = new Casio();
System.out.println(c1.getClass().getName());//Casio
Calculator c2 = new Quartz();
System.out.println(c2.getClass().getName());//Quar
Calculator c3 = new Kadio();
System.out.println(c3.getClass().getName());//Kadio
```

~~Bound~~ Bounded types

we can bound the type parameter for a particular range by using extends keyword such types are called bounded types

Example 1:

```
class Test<T> {}
Test <Integer> t1 = new Test<Integer>();
Test <String> t2 = new Test<String>();
```

Here as the type parameter we can pass any type and there are no restrictions hence it is unbounded type.

Example 2 :

```
class Test<T extends X> {}
```

if X is a class then as the type parameter we can pass either X or its child classes.

if X is an interface then as the type parameter we can pass either X or its implementation classes.

eg#1

```
class Test<T extends Number> {}
class Demo {
    public static void main(String[] args) {
        Test<Integer> t1 = new Test<Integer>();
        Test<String> t2 = new Test<String>(); //CE
    }}
```

eg#2

```
class Test<T extends Runnable> {}
class Demo {
    public static void main(String[] args) {
        Test<Thread> t1 = new Test<Thread>();
        Test<String> t2 = new Test<String>(); //CE
    }}
```

Key points about bounded types

→ we can't define bounded types by using implements and super keyword

→ But implements keyword purpose we can replace with extends keyword.

eg: Class Test <T implements Runnable> { } //invalid

class Test <T super String> { } // invalid.

20/12/22 Tuesday

Keypoints about Bounded Types

⇒ As the type parameter we can use any valid java identifier but it convention to use T always

eg. class Test <T> { }

class Test <INeuron> { }

⇒ we can pass any no of type parameters need not to be one.

eg: Class Hashmap <K, V> { }

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

which of the following are Valid.?

class Test <T extends Number & Runnable > { } //Valid

Number → class

Runnable → Interface

Comparable → interface

class Test <T extends Number & String > { } // invalid.

we can't extends more than one class at a time

class Test <T extends Runnable & Comparable > { } //Valid

Runnable → interface

Comparable → interface

Class Test <T extends Number & String > { } // invalid

we can't extends more than one class at a time

class Test <T extends Runnable & Comparable > { } //Valid

Runnable → interface

Comparable → interface

class Test <T extends Runnable & Number > { } //invalid

Runnable → interface

Number → Class

rule :- first inherit and the implement so invalid.

Generic class

Class: Type parameter

Can we apply Type parameter at Method level?

Ans. Yes, it is possible.

Generic methods and wild-card character (?)

① methodOne (ArrayList <String> al);

This method is applicable for ArrayList of only String type.

```
methodOne (ArrayList <String> al) {
        al.add ("sachin");
        al.add ("navinreddy");
        al.add ("ineuron");
        al.add (new Integer (10)); //invalid
}
```

Within the method we can add only String type of objects and null to the List

② methodOne (new ArrayList <String>());

methodOne (new ArrayList <String>());

methodOne (new ArrayList <Integer>());

methodOne (new ArrayList <Runnable>());

ArrayList <?> l = new ArrayList <Stri

methodOne (ArrayList <?> l);

We can use this method for ArrayList of any type but within the method we can't add anything to the List except null.

Example

```
l.add (null); // (valid.)
l.add ("A") ; //(invalid)
l.add (10); //(invalid)
```

This method is useful whenever we are performing only read operation.

3. methodOne (ArrayList<? extends X> al)

X -> class, we can make a call to method by passing ArrayList of X type or its child type.

X -> interface, we can make a call to method by passing ArrayList of X type or its implementation class.

```
methodOne (ArrayList<? extends X> al) {
    al.add (null);
}
```

Best suited only for read operation.

4. methodOne (ArrayList<? Super X> al)

X - interface, we can make a call to method by passing ArrayList of X type or its super class of implementation class of X.

```
methodOne (ArrayList<? Super X> al){
    al.add (X); al.add (null);
}
```

* which of the following declarations are allowe

(1.) ArrayList<String> l1 = new ArrayList<String>(); // Valid

(2.) ArrayList<?> l2 = new ArrayList<String>(); // Valid

(3) ArrayList<?> l3 = new ArrayList<Integer>(); // Va

(4) ArrayList<? extends number> l4 = new ArrayList<Inte // Valid.

(5) ArrayList<? extends number> l5 = new ArrayList<String // invalid

(6) ArrayList<?> l6 = new ArrayList<? extends Number>(); // Invalid.

(7) ArrayList<?> l7 = new ArrayList<?>(); invalid

Type parameter at method level

```
class Demo<T> {
    public <T> void m1(T t) {
    }
}
```

1 ⇒ Type parameter at class le

1 ⇒ Type parameter defined just before the return type

* which of the following declarations are allow

public <T> void methodOne (T t) { }. // Valid.

public <T extends Number> void methodOne2 (T t){} // Valid

public <T extends Number & Comparable> void methodOne (T, t){} Vali

```
public <T extends Number & Comparable & Runnable> void
        methodOne (T t) { } // Valid.
public <T extends Number & Comparable
                Thread > void methodOne (T t
                { } // Invalid.
public <T extends Runnable & Number > void methodOne
                (T t){ }
                // Invalid.
public <T extends Number & Runnable > void
                methodOne (T t){ }
                // Valid
```

Communication with non generic code

To provide compatibility with old version sunpeople compramized the concept of generics in very few area's the following is one such area.

Example:

```
import java.util.*;
class Test {
    public static void main (String [] args) {
        ArrayList <String> l = new ArrayList <String>();
        l.add ("sachin");
        //l.add (10); // CE : Cannot find symbol.
                        method add(double).
        System.out.println (l); // [sachin, 10, dhoni, true]
    }
```