DAY 11:

Today's topics
Generics
Overview of Collection Framework
Enter List.

1. Generics
What is it ? : Paramterized types
Applicable for : classes , interfaces, enums , methods , constrs.

Why ?
To Add type safety at the compile time
1. Avoids explicit type casting
2. Type mismatch errors are caught at the compile time

2. Collection Framework Overview

Enter : List implementation classes : ArrayList

Test basic API using Integer List n then apply it to vehicle showroom scenario.

Objective

1. Create empty AL to store vehicles in a showroom

2. Accept vehicle details from user n store it in AL(along with validation rules)
No dups please !

2.5 After testing vehicle registration in showroom , add sample data to store 5 vehicles in the showroom.

3. Display all vehicle details
Using for-each & also using Iterator(later)

4. Fetch Vehicle details from AL
i/p : PK (chasis no)
o/p : in case of success : display vehicle details o.w throw custom exception

5. Update Vehicle price
i/p : chasis no & new price
o/p : in case of success : update vehicle details o.w throw custom exception

6. Delete vehicle details
i/p : chasis no

7. Apply discount to all vehicles manufactured before a specific date n category
i/p : date n category n discount

8. Display vehicle chasis no n price , of all the vehicles dispatched to a specific city
i/p : city

100. Exit

Revise
2 types of nested classes
1. non static nested class
(inner class)

```
class Outer
{
  private int i;
  //static n non static members :
//can outer class directly access even private memebrs of the inner cls ?
  void show()
  {
    //how to access j ?


  }
  //static method : in outer clas
  static void print()
  {
    //how to access j ?
  }
  class Inner {
 private int j;
    //can contain only non static members(=non static methods, non static fields,static final
fields)
    // can't contain : static fields, static init blocks, static methods
    //can inner class methods DIRECTLY(w/o instantiation) access outer's even private methods
:
    //WHY : every inner class has  implicitly got the ref : Outer.this
  }
```

```
}
```
Can you create directly inner cls instance ?
YOu will have to create Outer cls instance n within that can create inner cl instance.
```
class Tester
{
  psvm(...)
  {
    Outer o=new Outer(...);
    Outer.Inner i=o.new Inner(....);
   //invoke methods of inner  cls : i.methodName(...)
    //OR
    Outer.Innner in2=new Outer(.....).new Inner(....);
    //invoke methods of inner  cls : in2.methodName(...)
  }
}
```
---

sample data
12342 red 45678 2-6-2021 petrol
12343 white 45890 2-6-2021 petrol
12340 black 55678 2-6-2021 petrol
12347 red 43679 2-6-2021 petrol

Generic syntax ---
Available from Java SE 5 onwards.
Represents Parameterized Types.
Can Create Generic classes, interfaces, methods and constructors.
In Pre-generics world , similar achieved via Object class reference.

Syntax -- similar to c++ templates (angle brackets)
eg : ArrayList<Emp> , HashMap<Integer,Account> .....
1. Syntax is different than C++ --for nested collections only.
2. NO code bloat issues unlike c++;

Advantages
Adds Type Safety to the code @ compile time

Meaning :
1. Can add type safe code where type-mismatch errors(i.e ClassCastExceptions) are detected at compile time.
2. No need of explicit type casting, as all casts are automatic and implicit.

A generic class means that the class declaration includes a type parameter.
eg --- class MyGeneric<T>
{
 private T ref;
}

class MyGeneric<T,U> {...}
T ,U ---type --- ref type
eg : ArrayList<Emp>

Understand why generics with example.
eg : Create a Holder class , that can hold ANY data type (primitive/ref type)

List<E> features
1. List represents ordered collection --- order is significant(It remembers the order of insertion)
2. Allows null references
3. Allows duplicates
4. Supports index based operation

java.util.ArrayList<E> -- E -- type of ref.
1. ArrayList<E> -- constructor
API
ArrayList() -- default constructor. -- creates EMPTY array list object , with init capacity=10,size=0;
eg ---ArrayList<Integer> l1=new ArrayList<>();

1.5 1. ArrayList<E> -- constructor
API
public ArrayList(int capacity) -- -- creates EMPTY array list object , with init capacity=capacity,size=0;
eg ---ArrayList<Integer> l1=new ArrayList<>(100);
l1.add(1);.....l1.add(100);
l1.add(101);//capa=150 --as per JVM spec.

2. add methods
boolean add(E e) --- append
void add(int index,E e) --- insert

void addAll(Collection<E> e) -- bulk append operation
eg : l1 --- AL<Emp>
l1.addAll(.....);
AL,LL,Vector --- legal
HS,TS,LHS --legal
HM,LHM,TM --illegal --javac error

2.5 Retrieve elem from list
E get(int index)
index ranges from ---0 ---(size-1)
java.lang.IndexOutOfBoundsException

3. display list contents using --- toString

4. Attaching Iterator
Collection<E> interface method -- implemented by ArrayList
Iterator<E> iterator()
---places iterator BEFORE 1st element ref.
Iterator<E> i/f methods
boolean hasNext() -- rets true if there exists next element, false otherwise.

E next() --- returns the element next to iterator position

void remove() -- removes last returned element from iterator.

Limitation --- type forward only & can start from 1st elem only.

Regarding exceptions with Iterator/List
1. java.util.NoSuchElementException -- thrown whenever trying to access the elem beyond the size of list via Iterator/ListIterator
2. java.lang.IllegalStateException --- thrown whenever trying to remove elem before calling next().
3. java.util.ConcurrentModificationException-- thrown typically --- when trying to use same iterator/list iterator --after structrually modifying list(eg add/remove methods of list)
Above describes fail-fast behaviour of the Iterator/ListIterator

Exception while accessing element by index.

4. java.lang.IndexOutOfBoundsException -- thrown typically -- while trying to access elem beyond size(0---size-1) --via get

6. Attaching for-each = attaching implicit iterator.

Attaching ListIterator ---scrollable iterator or to beign iteration from a specific element -- List ONLY or list specific iterator.
ListIterator<E> listItearator() --places LI before 1st element
ListIterator<E> listItearator(int index) --places LI before specified index.

4. search for a particular element in list
boolean contains(Object o)

5. searching for 1st occurrence
use -- indexOf
int indexOf(Object o)
rets index of 1st occurrence of specified elem. Rets -1 if elem not found.
searching for last occurrence
use -- lastIndexOf
int lastIndexOf(Object o)
rets index of last occurrence of specified elem. Rets -1 if elem not found.

5.5
E set(int index,E e)
Replaces old elem at spepcified index by new elem.
Returns old elem

6. remove methods
E remove(int index) ---removes elem at specified index & returns removed elem.

boolean remove(Object o) --- removes element specified by argument , rets true -- if elem is removed or false if elem cant be removed.

Objectives in Integer list
0. Create ArrayList of integers & populate it.
1. check if element exists in the list.
2. disp index of 1st occurance of the elem
3. double values in the list --if elem val > 20
4. remove elem at the specified index
5. remove by elem. -- rets true /false.

NOTE :
For searching or removing based upon primary key , in List Implementation classes --- All search/remove methods (contains,indexOf,lastIndexOf,remove(Object o)) -- based upon equals method(of type of List eg --Account/Customer/Emp....)
For correct working
1. Identify prim key & create overloaded constr using PK.
eg : public Emp(int id) { this.id=id;}

2. Using PK , override equals for content equality.

Usage eg : ArrayList<Emp> emps=new AL<>();
emps.add(e1);//id=10
emps.add(e2);//id=20
emps.add(e3);//id=30

int index=emps.indexOf(20);//int ---> Integer --> Object (Integer)
Integer i=new Integer(20); // javac
//internally invokes equals : whose equals --Object | Integer | Emp | NOA
invokes equlas on Integer class
i.equals(e1) ---since it's incomptabile types --rets false
i.equals(e2) ---since it's incomptabile types --rets false
i.equals(e3) ---since it's incomptabile types --rets false
Thus : indexOf rets -1
sop(index);// -1

Solution :
Emp e=new Emp(20);
int index=emps.indexOf(e);
//internally invokes equals : whose equals --Object | Integer | Emp | NOA
invokes equals on Emp class
e.equals(e1) ---it's comptabile types BUT ids are different --rets false
e.equals(e2) --- it's comptabile types --ids are SAME --rets true

Thus : indexOf rets 1
sop(index);// 1

Objective --- Create simple List(ArrayList) of Account & test complete API
1.1
Create Empty Arraylist of Accounts
1.2 Accept a/c info from user till user types "stop" & populate AL.

1.2.1 -- Display AL content using for-each
1.3 Accept account id & display summary or error mesg
1.4 Accept src id , dest id & funds transfer.
1.5 Accept acct id & remove a/c --
1.6 Apply interest on all saving a/cs
1.7 Sort accounts as per asc a/c ids.
1.8 Sort accounts as per desc a/c ids.
1.9 Sort a/cs as per creation date -- w/o touching UDT
2.0 Sort a/cs as per bal

Sorting --- For sorting elements as per Natural(implicit i.e criteria defined within UDT class definition) ordering or Custom(explicit i.e criteria defined outside UDT , in a separate class or anonymus iner class)

Steps for Natural ordering
Natural Ordering is specified in generic i/f
java.lang.Comparable<T>
T -- UDT , class type of the object to be compared.
eg -- Emp,Account , Customer
I/f method
int compareTo(T o)
Steps
1. UDT must implement Comparable<T>
eg : public class Account implements Comparable<Account>
2. Must override method
public int compareTo(T o)
{
use sorting criteria to ret
< 0 if this < o,
=0 if this = o
> 0 if this > o

}
3.Use java.util.Collections class API
Method
public static void sort(List<T> l1)
l1 -- List of type T.

sort method internally invokes compareTo method(prog supplied) of UDT & using advanced sorting algorithm , sort the list elems.

Limitation of natural Ordering
Can supply only 1 criteria at given time & that too is embedded within UDT class definition
Instead keep sorting criteria external --using Custom ordering

Typically use -- Natural ordering in consistence with equals method.

Alternative is Custom Ordering(external ordering)
I/f used is --- java.util.Comparator<T>
T -- type of object to be compared.

Steps
1. Create a separate class (eg. AccountBalComparator) which implements Comparator<T>
eg
public class AccountBalComparator implements Comparator<Account>

2.Implement(override) i/f method -- to supply comparison criteria.

int compare(T o1,T o2)
Must return

< 0 if o1<o2
=0 if o1=o2
> 0 if o1 > o2
3. Invoke Collections class method for actual sorting.
public static void sort(List<T> l1,Comparator<T> c)
parameters
l1 --- List to be sorted(since List is i/f --- any of its implementation class inst. can be passed)
c - instance of the class which has implemented compare method.(or implemented Comparator)

Internally sort method invokes compare method from the supplied Comparator class instance.

More on generic syntax
Constructor of ArrayList(Collection<? extends E> c)
? -- wild card in generic syntax (denotes any unknown type)
--Added for supporting inheritance in generics.'

extends -- keyword in generics, to specify upper bound
? extends E -- E or sub type

Complete meaning --- Can create new populated ArrayList of type E , from ANY Collection(ArrayList,LinkedList,Vector,HashSet,LinkedhashSet,TreeSet) of type E or its sub type.

```
ArrayList<Emp> l1=new ArrayList<>();
l1.add(new Emp(1,"aa",1000);
l1.add(new Emp(2,"ab",2000);
ArrayList<Emp> l2=new ArrayList<>(l1);
sop(l2.size());
```

```
 HashSet<Emp> hs=new HashSet<>();
hs.add(new Emp(1,"aa",1000);
hs.add(new Emp(2,"ab",2000);
l2=new ArrayList<>(hs);
```

```
Vector<Mgr> v1=new Vector<>();
v1.add(new Mgr(....));
v1.add(new Mgr(....));
ArrayList<Emp> l2=new ArrayList<Mgr>(v1);
AL<Mgr> mgrs=new AL<>(hs);
```

Map API

HashMap<K,V> --

1. un-sorted(not sorted as per Natural ordering or custom ordering based criteria) & un-ordered(doesn't remember order of insertion) map implementation class.
2. No duplicate keys.
3. Guarantees constant time performance --- via 2 attributes --initial cpacity & load factor.
4. Allows null key reference(once).
5. Inherently thrd unsafe.

HashMap constrs

1. HashMap<K,V>() --- creates empty map , init capa = 16 & load factor .75
2. HashMap<K,V>(int capa) --- creates empty map , init capa specified & load factor .75
3.HashMap<K,V>(int capa,float loadFactor) --- creates empty map , init capa & load factor specified

4. HashMap constrcutor for creating populated map

HashMap(Map <? extends K,? extends V> m)

? -- wild card in generics, represents unknown type

extends -- represents upper bound

? extends K --- K or its sub type

? extends V -- V or its sub type.

Complete meaning -- Creates populated HM<K,V> from ANY map(ie. any Map imple class) of type K or its sub type & V or its sub type.

eg : Suppose Emp <---- Mgr

```
HM<Integer,Emp> hm=new HM<>();
hm.put(1,e1);
hm.put(2,m1);
HM<Integer,Emp> hm2=new HM<>(hm);
sop(hm2);
LHM<Integer,Emp> lhm=new LHM<>(hm);//legal
HM<Integer,Mgr> hm3=new HM<Integer,Emp>(hm);//javac error

TM<Integer,Mgr> hm4=new TM<>();
hm4.put.....
HM<Integer,Emp> hm5=new HM<>(hm4);
```

HM(Map<? extends K,? extends V>map)

put,get,size,isEmpty,containsKey,containValue,remove

Objective : Create AccountMap
Identify key & value type
create empty unsorted map(HashMap<K,V>) & populate the same

Disp all entries of HM ---can use only toString
1.get acct summary --- i/p --id o/p --- err / dtls
2.Withdraw --- specify Account id & Amt ---- o/p : update acct dtls if acct exists o.w err msg or exc
3.funds transfer ---
i/p sid,dest id, amt

4.remove --- account
i/p id
5.Apply interest on on saving type of a/cs.
or
display all accts created after date.

Attach for-each to map & observe.
Sort the map as per : asc order of accts Ids.
Sort the map as per : desc order of accts Ids
Sort the accts as per : balance

If map sorting involves key based sorting criteria --- can be sorted by converting into TreeMap
Constructors of TreeMap
1. TreeMap() -- Creates empty map , based upon natural ordering of keys
2. TreeMap(Map<? extends K,? extends V> map)
Creates populated map , based upon natural ordering of keys
3. TreeMap(Comparator<? super K> c)
Regarding generic syntax & its usage in TreeMap constructor.
<? super K>
? --- wild card --- any unknown type
super --- gives lower bound
K --- key type
? super K --- Any type which is either K or its super type.

TreeMap(Comparator<? super K> c) --- creates new empty TreeMap, which will sort its element as per custom ordering(i.e will invoke compare(...) of Key type)

<? extends K>
? -- any type or wild card
extends -- specifies upper bound
K -- key type
? extends K --- Any type as Key type or its sub type.

same meaning for <? extends V>
TreeMap(Map<? extends K,? extends V> m)

disp acct ids of all accounts ---impossible directly....(will be done by Collection view of map @ the end)
Apply interest to all saving type a/cs

difficult directly ---so get a collection view of the map & sort the same.

Limitations on Maps
1. Maps can be sorted as per key's criteria alone.
2. can't attach iterators/for-each(till JDK 1.7)/for
3 Maps can be searched as per key's criteria alone.

To fix --- get a collection view of a map (i.e convert map to collection)

API of Map i/f

1. To get set of keys asso. with a Map
Set<K> keySet();
2. To get collection of values from a map
Collection<V> values();
3. To get set of Entries(key & val pair) ---
entrySet
Set<Map.Entry> entrySet()
Methods of Map.Entry
K getKey()
V getValue()

7. conversion from collection to array
Object[] toArray() -- non generic version --- rets array of objects

T[] toArray(T[] type)
T = type of collection .
Rets array of actual type.

8. sorting lists --- Natural ordering creiteria
Using java.util.Collections --- collection utility class.
static void sort(List<E> l1) ---sorts specified list as per natural sorting criteria.

DAY 12:

Today's topics
1. Complete pending work
2. Iterator , ListIterator , Exceptions associated.
3. Sorting in Java
4. ArrayList Vs LinkedList

Revise
Collection Framework Overview
What is it ? : Readymade implementation of dynamic data structures(List,Set,Map) + DSA(data structure algorithms)

Why ? : Supplies readymade implementations of dynamic data structures. Supports insertion /deletion , searching /sorting , CRUD operations

Inheritance Hierarchy
(refer to diagram)

Will you be able to create a populated AL(ArrayList<? extends E> Collection c) from
1. AL , LL , Vector
2. HS, LHS,TS
3. HM , LHM ,TM
Ans : 1, 2

Which of the following will cause a structural modification of the list?(modifying size)
1. add /addAll
2. remove / removeAll
3. retainAll
4. set
Ans : 1,2,3

eg : l1 : [1,2,3,4,5]
l2 : [3,4]
l1.retainAll(l2) => l1 : [3,4]

What will be o/p ?

```
list : [10,20,30,40,50,60]
for (int i = 0; i < list.size(); i++) {
 list.remove(i);
}
System.out.println("list after remove "+list);//[20,40,60]
```

List Features & implementation classes

1. Complete pending objectives

1.0 Purchase a Vehicle

1.1 Apply discount to all vehicles manufactured before a specific date n category
i/p : date n category n discount

1.2 Display vehicle chasis no n price , of all the vehicles dispatched to a specific city
i/p : city

2. Create a simple tester to understand :
Iterator , ListIterator , Exceptions associated.

3. Sorting related objectives

3.1 Sort vehicles as per chasis no (asc order)

3.2 Sort vehicle as per price (desc order)

3.3 Sort vehicles as per date n price

Pending Topics

Anonymous inner class

5. ArrayList Vs LinkedList

---

java.util.LinkedList<E>

Doubly-linked list implementation of the List and Deque interfaces.
It is an ordered collection and supports duplicate elements.
It stores elements in Insertion order.
It supports adding null elements.
It supports index based operations.
Typical use case -- stack or queue.
It does not implement RandomAccess interface.(ArrayList class does!)
So it represents sequential access list.

When we try to access an element from a LinkedList, searching that element starts from the beginning or end of the LinkedList based on whichever is closer to the specified index.(eg : list.get(i))

Structure of LinkedList --refer to diag.

It supports all of List API methods , as seen already in ArrayList.

Java LinkedList methods , inherited from Deque

The following methods are specific to LinkedList class which are inherited from Deque interface:

void addFirst(E e): Inserts the specified element at the beginning of this list.
void addLast(E e): Inserts the specified element at the end of this list.
E getFirst(): Retrieves, but does not remove, the first element of this list. This method differs from peekFirst only in that it throws an exception if this list is empty.
E getLast(): Retrieves, but does not remove, the last element of this list. This method differs from peekLast only in that it throws an exception if this list is empty.
E removeFirst(): Removes and returns the first element from this list.
E removeLast(): Removes and returns the last element from this list.
boolean offerFirst(E e): Inserts the specified element at the front of this list.
boolean offerLast(E e): Inserts the specified element at the end of this list.
E pollFirst(): Retrieves and removes the first element of this list, or returns null if this list is empty.
E pollLast(): Retrieves and removes the last element of this list, or returns null if this list is empty.
E peekFirst(): Retrieves, but does not remove, the first element of this list, or returns null if this list is empty.
E peekLast(): Retrieves, but does not remove, the last element of this list, or returns null if this list is empty.

Java LinkedList Usecases

Best Usecase scenario:-

When our frequently used operation is adding or removing elements in the middle of the List, LinkedList is the best class to use.

Why? Because we don’t need to do more shifts to add or remove elements at the middle of the list.

Worst Usecase scenario:-

When our frequently used operation is retrieving elements from list, then LinkedList is the worst choice.

Why? Because LinkedList supports only sequential access, does NOT support random access.

NOTE:-
LinkedList implements List, Deque, But it does NOT implement RandomAccess interface.

How to use LinkedList as a queue or stack ?
When a deque is used as a queue its FIFO (First-In-First-Out)Elements are added at the end of the deque and removed from the beginning.

Which method will you use ?????????

When a deque is used as stack its LIFO (Last-In-First-Out) (Preferred to the legacy Stack class.) When a deque is used as a stack, elements are pushed and popped from the beginning of the deque.
Which method will you use ?????????

Difference between ArrayList and LinkedList in Java

ArrayList and LinkedList both implements List interface and their methods and results are almost identical. However there are few differences between them which make one better over another depending on the requirement.

ArrayList Vs LinkedList

1) Search: ArrayList search operation is pretty fast compared to the LinkedList search operation. get(int index) in ArrayList gives the performance of O(1) while LinkedList performance is O(n).

Reason: ArrayList maintains index based system for its elements as it uses array data structure implicitly which makes it faster for searching an element in the list. On the other side LinkedList implements doubly linked list which requires the traversal through all the elements for searching an element.

2) Deletion: LinkedList remove operation gives O(1) performance while ArrayList gives variable performance: O(n) in worst case (while removing first element) and O(1) in best case (While removing last element).

Conclusion: LinkedList element deletion is faster compared to ArrayList.

Reason: LinkedList's each element maintains two pointers (addresses) which points to the both neighbor elements in the list. Hence removal only requires change in the pointer location in the two neighbor nodes (elements) of the node which is going to be removed. While In ArrayList all the elements need to be shifted to fill out the space created by removed element.

3) Inserts Performance: LinkedList add method gives O(1) performance while ArrayList gives O(n) in worst case. Reason is same as explained for remove.

4) Memory Overhead: ArrayList maintains indexes and element data while LinkedList maintains element data and two pointers for neighbor nodes hence the memory consumption is high in LinkedList comparatively.

There are few similarities between these classes which are as follows:

Both ArrayList and LinkedList are implementation of List interface.
They both maintain the elements insertion order which means while displaying ArrayList and LinkedList elements the result set would be having the same order in which the elements got inserted into the List.
Both these classes are non-synchronized and can be made synchronized explicitly by using Collections.synchronizedList method.
The iterator and listIterator returned by these classes are fail-fast (if list is structurally modified at any time after the iterator is created, in any way except through the iterator's own remove or add methods, the iterator will throw a ConcurrentModificationException).

When to use LinkedList and when to use ArrayList?

1) As explained above the insert and remove operations give good performance (O(1)) in LinkedList compared to ArrayList(O(n)). Hence if there is a requirement of frequent addition and deletion in application then LinkedList is a best choice.

2) Search (get(index) method) operations are fast in Arraylist (O(1)) but not in LinkedList (O(n)) so If there are less add and remove operations and more search operations requirement, ArrayList would be your best bet.

	ArrayList	LinkedList
Structure	ArrayList is an index based data structure where each element is associated with an index.	Elements in the LinkedList are called as nodes, where each node consists of three things – Reference to previous element, Actual value of the element and Reference to next element.
Insertion And Removal	Insertions and Removals in the middle of the ArrayList are very slow. Because after each insertion and removal, elements need to be shifted.	Insertions and Removals from any position in the LinkedList are faster than the ArrayList. Because there is no need to shift the elements after every insertion and removal. Only references of previous and next elements are to be changed.
Retrieval (Searching or getting an element)	Insertion and removal operations in ArrayList are of order O(n). Retrieval of elements in the ArrayList is faster than the LinkedList . Because all elements in ArrayList are index based.	Insertion and removal in LinkedList are of order O(1). Retrieval of elements in LinkedList is very slow compared to ArrayList. Because to retrieve an element, you have to traverse from beginning or end (Whichever is closer to that element) to reach that element.
Random Access	Retrieval operation in ArrayList is of order of O(1). ArrayList is of type Random Access. i.e elements can be accessed randomly.	Retrieval operation in LinkedList is of order of O(n). LinkedList is not of type Random Access. i.e elements can not be accessed randomly. you have to traverse from beginning or end to reach a particular element.
Usage	ArrayList can not be used as a Stack or Queue.	LinkedList, once defined, can be used as ArrayList, Stack, Queue, Singly Linked List and Doubly Linked List.
Memory Occupation	ArrayList requires less memory compared to LinkedList. Because ArrayList holds only actual data and it's index.	LinkedList requires more memory compared to ArrayList. Because, each node in LinkedList holds data and reference to next and previous elements.
When To Use	If your application does more retrieval than the insertions and deletions, then use ArravList.	If your application does more insertions and deletions than the retrieval, then use LinkedList.

DAY 13 :

Today's Topics
ArrayList Vs LinkedList
Wild Card in generics & bounds
Set interface & implementation classes

Revision
Sorting in Java

Which are 2 ways of sorting in Java ?
Natural Ordering : Sorting criteria kept within generic type of the List/Set

Custom ordering : Sorting criteria kept in separate class / anonymous inner class /lamda expression

Sorting elements in the List
Steps for 1. Sort customer details , as per email (asc) : Use Natural Ordering
1. public class Customer implements Comparable<Customer> {...}

2. imple method in the Customer class
```
@Override
public int compareTo(Customer c2)
{
//  return this.email.compareTo(c2.email);//asc order
return c2.email.compareTo(this.email);//desc order
//c1 : xyz@gmail.com c2 : abc@gmail.com : -1
}
```

3. java.util.Collections : helper class containing static utility methods
public static void sort(List<T> list)
eg : Collections.sort(customers);
Which sorting algo : Tim's sorting algo (iterative merge sort)
What will it call internally ?
eg : [c1,c2,c3,c4.....c100]
c1.compareTo(c2) < 0 or 0 : not swapped
> 0 : swap

2. Sort customer details , as per Dob (desc) n Plan (customer type) : custom ordering , using anonymous inner class.

When u need multiple sorting criteria : that too w/o touching core class (UDT :eg : Customer)
keep sorting criteria : external
Custom ordering
Steps
1,2 Either create a separate class or ano inner class that imple : Comparator<T>
```
public class CustomerDoBPlanComparator imple. Comparator<Customer>
{
  @Override
  public int compare(Customer c1,Customer c2)
```

```
  {
 //dob desc
   int retVal=c2.getDob().compareTo(c1.getDob());
   if(retVal ==0) //=> Dob is same
    return c1.getPlan().compareTo(c2.getPlan());
   return retVal;
  }
}
```

3. Collections.sort(customers);//N.O
Collections.sort(customers,null);//N.O
Collections.sort(customers,new CustomerDoBPlanComparator());//C.O
JVM (sorting ago) : invokes CustomerDoBPlanComparator's compare method

Sort customer details , as per Dob (desc) n Plan (customer type) : custom ordering , using anonymous inner class.
TestCustomer.java

```
Collections.sort(customers,new Comparator<Customer>() //instance of the imple class
{ //ano inner cls begin
@Override
  public int compare(Customer c1,Customer c2)
  {
 //dob desc
   int retVal=c2.getDob().compareTo(c1.getDob());
   if(retVal ==0) //=> Dob is same
    return c1.getPlan().compareTo(c2.getPlan());
   return retVal;
  }
}//ano inner cls end
);
```

.class file : TestCustomers$1

4. LinkedList
Refer to : Collection Framework Overview diagram.
diagrams : Types of linked lists , node , insert n delete operations
Refer to readme
Solve ready code samples.
Compare ArrayList Vs LinkedList

5. More in generics
Refer to : generics sequence

6. Java 8 New Features

Java 8 Date/Time related APIs
Java 8 New Features
java.time : new package is introduced
LocalDate : Date (immutable)(yr-mon-day) : inherently thrd safe.
API
1. public static LocalDate now()
Obtains the current date from the system clock in the default time-zone.
eg :
LocalTime : Time (immutable)(hr-min-sec) : inherently thrd safe.
LocalDateTime : Date n Time : inherently thrd safe.
eg : ???
2. public static LocalDate of(int year,int month,int dayOfMonth)
Obtains an instance of LocalDate from a year, month and day.
eg : ?????
3. public static LocalDate parse(CharSequence text)
Obtains an instance of LocalDate from a text string such as 2007-12-03.
eg : LocalDate dt=parse(sc.next());//0 based dates
4. Methods :
isBefore,isAfter,isEqual
5. Can you change default DateTime format ? : YES

6. Enter Sets
HashSet , LinkedHashSet , TreeSet

7. Hashing Algorithm

5. Enter Sets

Example of strings having same hashcode
Aa BB
Ba CB
Ca DB
Da EB
BBBB AaBB AaAa BBAa
FB Ea

---

1. Generic Method
What is it ? A Method which has it's own type parameter.

Can it exist in a non generic class? : Yes

Generic method example : Arrays.toList
java.util.Arrays class
public static <T> List<T> asList(T... a)
Generic method from a non generic class(Arrays)
i/p : T... a : This method can accept : no args or T[] or T t1,T t2....
o/p : FIXED size list
How will you use it for getting a fixed size List<Integer> ?
---

2. More about generic syntax : ? , extends ,super
Given :
Emp <---- Mgr <----SalesMgr
Emp <---- Worker <---- TempWorker

What will happen ?
Emp e =new Mgr();
e.computeSalary();
e=new TempWorker();
e.computeSalary();

ArrayList<Emp> l1=new ArrayList<>();//what's the inferred type of the RHS ? :
ArrayList<Mgr> l2=new ArrayList<>();//what's the inferred type of the RHS ? :
l1=l2;
Object o=new Mgr();
ArrayList<Object> l3=new ArrayList<>();//what's the inferred type of the RHS ? :
l3=l2;
l3=l1;

What does it imply ?
Does inheritance in generics work in the same manner as learnt earlier ?

Wild card in generics comes to the rescue!
Enter "?"
What is it ?
It's a wild card in generic syntax. Can be replaced by ANY type.
Represents any unknown type.

Now what will happen ?

ArrayList<?> l1=new ArrayList<>();//what's the implicit type of the RHS ? :
ArrayList<Mgr> l2=new ArrayList<>();
l1=l2;

? : is referred to as un bounded wild card.

Hands on
1. Shuffle all the elements of the list randomly.
Collections class Method
public static void shuffle(List<?> list)

2. Collections class
reverse method

3. Write a static method in GenericUtils class : to print elements of ANY List(AL/LL/Vector) of ANY type

1. Add a method printElements to print the elements of ANY List of ANY type.
1.1 Try it with T : parameter Type (lab work!)
1.2 Try it with ?
Any difference ?
Which seems easier to use ?
Test it with tester : AL<Integer> , LinkedList<String>

2. Add "computeSalary" method in Emp based hierarchy
Modify GenericUtils class .
Add static method which can accept ANY List(AL/LL/Vector) of any type of emps(Emp /Mgr / SalesMgr / Worker/ TempWorker)
& compute salary.
Test it from Tester class .
2.1 Try it with T : parameter Type
2.2 Try it with ?
Any difference ?
Which seems easier to use ?

3. Write a method to Find max number from List of any numbers (integer / float / double ...) n return it to the caller.
(Lab exercise)

4. Write a method to Find max element from numbers / strings/ dates
Lab Exercise

5. For more details : refer to Collections.copy
For T , ? , extends , super
(Lab exercise)

eg of bounded type in Collection Framework
ArrayList<E> : generic class
public ArrayList(Collection<? extends E> c)
E : Emp
? extends Emp
Meaning : This ctor will create a non-empty AL of Emp type , from AL/LL/Vector/HS/LHS/TS (any Collection imple class) , from any type : Emp or it's sub type
Q : Will you be able populate AL<Mgr> from
1. AL<Emp>
2. HS<Worker>
3. LL<SalesMgr>
4. Vector<Fruit>
5. TS<Object>
6. LHS<HRMgr>
Ans :

Generic syntax :
? : wild card in generic syntax (it can be replaced by ANY type) : un bounded wild card
extends : Represents upper bound
super : Represents lower bound
? extends E : ANY type E or its sub type
? super E : E or its super type
eg : ? extends Emp => Emp or it's sub type(Mgr,Worker.....)
? super Mgr => Mgr or it's super type (Emp , Object)

What will happen ? (javac error or no error?)
1. Vector<Mgr> mgrs=new Vector<>();
mgs.add(m1)....m10
ArrayList<Emp> emps=new AL<>(mgrs);

2. HashSet<HRMgr> hrMgrs=new HS<>();

hrMgrs.add(hrm1)....hrm10
ArrayList<Emp> emps=new AL<>(hrMgrs);
3.
ArrayList<Mgr> mgrs=new AL<>();
mgrs.add(m1).....m10
LinkedList<HRMgr> hrMgrs=new LinkedList<>(mgrs);

Ans this !
ArrayList<Mgr> mgrs=new AL<>();
mgrs.add(m1).....m10
ArrayList<? extends Emp> emp=new AL<>(mgrs);
ArrayList<? extends Object> objs=mgrs;

3.Collections class(Non generic class) : Can contain a generic Method
eg : super keyword in generics : lower bound
Method of Collections class
public static <T> void sort(List<T> list,Comparator<? super T> c)
generic method :
where does type declaration fit ? : It's placed between method modifiers n ret type
1st arg : List<T> list : You can pass List of ANY type (eg : AL/LL/Vector : ??
HS/LHS/TS : ??
HM/LHM/TM : ??) :
Can you pass List of the following types :
T --Customer , Student(extends Person) , BankAccount,Flight, Movie,Person , DacStudent (extends Student)
if above classes have not imple Comparable or Comparator :

2nd arg : Comparator<? super T> c
Instance of the class which imple Comparator
eg : Collections.sort(studentList,comp);//comp : imple Comparator<Student>
Collections.sort(studentList,comp);//comp : imple Comaprator<Fruit>
Collections.sort(studentList,comp);//comp : imple Comaprator<Person>
Collections.sort(studentList,comp);//comp : imple Comaprator<Object>
Collections.sort(studentList,comp);//comp : imple Comaprator<eDacStudent>

More Details

? : wild card (represents ANY unknown type) => un bounded wildcard

extends => upper bound (type of the upper bound super class / interface)

If your collection(list/set/map) is acting as a producer (of data) i.e while using retrieve operation, use upper bound
eg : ? extends Number => Number or it's subtype (Byte,Short.....Double)
get : type of the data that you get Number or it's subtype, BUT you can't add Integer , if it's collection of Double
Can be accessed using Number type of the ref.
(Producer extends)

super : lower bound
? super T => T or it's super type
Use it whenever your collection is acting as a consumer (data sink) : i.e wheneve you want to add data , to a collection.
(Consumer super)

1. Solve generic questions

2. Solve (Lab work)
Collections class API
public static <T> void copy(List<? super T> dest, List<? extends T> src)

If you consider Emp<---Mgr <---- SalesMgr

Which of the following is valid ?
1. You should be able to copy SalesMgr type of refs into Emp type of the list
2. You should be able to copy SalesMgr type of refs into Mgr type of the list
3. You should be able to copy Emp type of refs into Mgr type of the list
4. You should be able to copy Mgr type of refs into SalesMgr type of the list

It means
If src type of the list for copying is SalesMgr (i.e you have a list of SalesMgrs)
dest type of the list for copy should be ????

If src type of the list for copying is Mgr
dest type of the list for copy should be ????

If src type of the list for copying is Emp
dest type of the list for copy should be ????

If the dest type of the list Mgr
src type of the list should be ???

What is the difference between a wildcard bound and a type parameter bound?

A wildcard can have only one bound, while a type parameter can have several bounds.

A wildcard can have a lower or an upper bound, while there is no such thing as a lower bound for a type parameter.

Wildcard bounds and type parameter bounds are often confused, because they are both called bounds and have in part similar syntax.

Example (of type parameter bound and wildcard bound):

```
class Box< T extends Appendable & Flushable > {
  private T theObject;
  public Box(T arg)              { theObject = arg; }
  public Box(Box< ? extends T > box) { theObject = box.theObject; }
  ...
}
```

List<E> features
1. List represents ordered collection --- order is significant(It remembers the order of insertion)
2. Allows null references
3. Allows duplicates
4. Supports index based operation

java.util.ArrayList<E> -- E -- type of ref.
1. ArrayList<E> -- constructor
API
ArrayList() -- default constructor. -- creates EMPTY array list object , with init capacity=10,size=0;
eg ---ArrayList<Integer> l1=new ArrayList<>();

1.5 1. ArrayList<E> -- constructor
API
public ArrayList(int capacity) -- -- creates EMPTY array list object , with init capacity=capacity,size=0;
eg ---ArrayList<Integer> l1=new ArrayList<>(100);
l1.add(1);.....l1.add(100);

l1.add(101);//capa=150 --as per JVM spec.

2. add methods
boolean add(E e) --- append
void add(int index,E e) --- insert
void addAll(Collection<E> e) -- bulk append operation
eg : l1 --- AL<Emp>
l1.addAll(.....);
AL,LL,Vector --- legal
HS,TS,LHS --legal
HM,LHM,TM --illegal --javac error

2.5 Retrieve elem from list
E get(int index)
index ranges from ---0 ---(size-1)
java.lang.IndexOutOfBoundsException

3. display list contents using --- toString

4. Attaching Iterator
Collection<E> interface method -- implemented by ArrayList
Iterator<E> iterator()
---places iterator BEFORE 1st element ref.
Iterator<E> i/f methods
boolean hasNext() -- rets true if there exists next element, false otherwise.

E next() --- returns the element next to iterator position

void remove() -- removes last returned element from iterator.

Limitation --- type forward only & can start from 1st elem only.

Regarding exceptions with Iterator/List
1. java.util.NoSuchElementException -- thrown whenever trying to access the elem beyond the size of list via Iterator/ListIterator
2. java.lang.IllegalStateException --- thrown whenever trying to remove elem before calling next().
3. java.util.ConcurrentModificationException-- thrown typically --- when trying to use same iterator/list iterator --after structrually modifying list(eg add/remove methods of list)
Above describes fail-fast behaviour of the Iterator/ListIterator

Exception while accessing element by index.

4. java.lang.IndexOutOfBoundsException -- thrown typically -- while trying to access elem beyond size(0---size-1) --via get

6. Attaching for-each = attaching implicit iterator.

Attaching ListIterator ---scrollable iterator or to beign iteration from a specific element -- List ONLY or list specific iterator.
ListIterator<E> listItearator() --places LI before 1st element
ListIterator<E> listItearator(int index) --places LI before specified index.

4. search for a particular element in list
boolean contains(Object o)

5. searching for 1st occurrence
use -- indexOf
int indexOf(Object o)
rets index of 1st occurrence of specified elem. Rets -1 if elem not found.
searching for last occurrence
use -- lastIndexOf
int lastIndexOf(Object o)
rets index of last occurrence of specified elem. Rets -1 if elem not found.

5.5
E set(int index,E e)
Replaces old elem at spepcified index by new elem.
Returns old elem

6. remove methods
E remove(int index) ---removes elem at specified index & returns removed elem.

boolean remove(Object o) --- removes element specified by argument , rets true -- if elem is removed or false if elem cant be removed.

Objectives in Integer list
0. Create ArrayList of integers & populate it.
1. check if element exists in the list.

2. disp index of 1st occurance of the elem
3. double values in the list --if elem val > 20
4. remove elem at the specified index
5. remove by elem. -- rets true /false.

NOTE :
For searching or removing based upon primary key , in List Implementation classes --- All search methods (contains,indexOf,lastIndexOf,remove(Object o)) -- based upon equals method(of type of List eg --Account/Customer/Emp....)
For correct working
1. Identify prim key & create overloaded constr using PK.
2. Using PK , override equals for content equality.

Objective --- Create simple List(ArrayList) of Account & test complete API
1.1
Create Empty Arraylist of Accounts
1.2 Accept a/c info from user till user types "stop" & populate AL.
1.2.1 -- Display AL content using for-each
1.3 Accept account id & display summary or error mesg
1.4 Accept src id , dest id & funds transfer.
1.5 Accept acct id & remove a/c --
1.6 Apply interest on all saving a/cs
1.7 Sort accounts as per asc a/c ids.
1.8 Sort accounts as per desc a/c ids.
1.9 Sort a/cs as per creation date -- w/o touching UDT
2.0 Sort a/cs as per bal

Sorting --- For sorting elements as per Natural(implicit i.e criteria defined within UDT class definition) ordering or Custom(explicit i.e criteria defined outside UDT , in a separate class or anonymus iner class)

Steps for Natural ordering
Natural Ordering is specified in generic i/f
java.lang.Comparable<T>
T -- UDT , class type of the object to be compared.
eg -- Emp,Account , Customer
I/f method
int compareTo(T o)
Steps
1. UDT must implement Comparable<T>

eg : public class Account implements Comparable<Account>
2. Must override method
public int compareTo(T o)
{
 use sorting criteria to ret
< 0 if this < o,
=0 if this = o
> 0 if this > o

}
3.Use java.util.Collections class API
Method
public static void sort(List<T> l1)
l1 -- List of type T.

sort method internally invokes compareTo method(prog supplied) of UDT & using advanced sorting algorithm , sort the list elems.

Limitation of natural Ordering
Can supply only 1 criteria at given time & that too is embedded within UDT class definition
Instead keep sorting criteria external --using Custom ordering

Typically use -- Natural ordering in consistence with equals method.

Alternative is Custom Ordering(external ordering)
I/f used is --- java.util.Comparator<T>
T -- type of object to be compared.

Steps
1. Create a separate class (eg. AccountBalComparator) which implements Comparator<T>
eg
public class AccountBalComparator implements Comparator<Account>{...}

2.Implement(override) i/f method -- to supply comparison criteria.

int compare(T o1,T o2)
Must return

< 0 if o1<o2
=0 if o1=o2

> 0 if o1 > o2
3. Invoke Collections class method for actual sorting.
public static void sort(List<T> l1,Comparator<T> c)
parameters
l1 --- List to be sorted(since List is i/f --- any of its implementation class inst. can be passed)
c - instance of the class which has implemented compare method.(or implemented Comparator)

Internally sort method invokes compare method from the supplied Comparator class instance.

More on generic syntax
Constructor of ArrayList(Collection<? extends E> c)
? -- wild card in generic syntax (denotes any unknown type)
--Added for supporting inheritance in generics.'

extends -- keyword in generics, to specify upper bound
? extends E -- E or sub type
Complete meaning --- Can create new populated ArrayList of type E , from ANY Collection(ArrayList,LinkedList,Vector,HashSet,LinkedhashSet,TreeSet) of type E or its sub type.

```
ArrayList<Emp> l1=new ArrayList<>();
l1.add(new Emp(1,"aa",1000);
l1.add(new Emp(2,"ab",2000);
ArrayList<Emp> l2=new ArrayList<>(l1);
sop(l2.size());
----------
 HashSet<Emp> hs=new HashSet<>();
hs.add(new Emp(1,"aa",1000);
hs.add(new Emp(2,"ab",2000);
l2=new ArrayList<>(hs);
----
Vector<Mgr> v1=new Vector<>();
v1.add(new Mgr(....));
v1.add(new Mgr(....));
ArrayList<Emp> l2=new ArrayList<Mgr>(v1);
AL<Mgr> mgrs=new AL<>(hs);
```

Map API
HashMap<K,V> --
1. un-sorted(not sorted as per Natural ordering or custom ordering based criteria) & un-ordered(doesn't remember order of insertion) map implementation class.
2. No duplicate keys.
3. Guarantees constant time performance --- via 2 attributes --initial cpacity & load factor.

4. Allows null key reference(once).
5. Inherently thrd unsafe.

HashMap constrs
1. HashMap<K,V>() --- creates empty map , init capa = 16 & load factor .75
2. HashMap<K,V>(int capa) --- creates empty map , init capa specified & load factor .75
3.HashMap<K,V>(int capa,float loadFactor) --- creates empty map , init capa & load factor specified

4. HashMap constrcutor for creating populated map
HashMap(Map <? extends K,? extends V> m)
? -- wild card in generics, represents unknown type
extends -- represents upper bound
? extends K --- K or its sub type
? extends V -- V or its sub type.
Complete meaning -- Creates populated HM<K,V> from ANY map(ie. any Map imple class) of type K or its sub type & V or its sub type.
eg : Suppose Emp <---- Mgr

```
HM<Integer,Emp> hm=new HM<>();
hm.put(1,e1);
hm.put(2,m1);
HM<Integer,Emp> hm2=new HM<>(hm);
sop(hm2);
LHM<Integer,Emp> lhm=new LHM<>(hm);//legal
HM<Integer,Mgr> hm3=new HM<Integer,Emp>(hm);//javac error

TM<Integer,Mgr> hm4=new TM<>();
hm4.put.....
HM<Integer,Emp> hm5=new HM<>(hm4);
```

HM(Map<? extends K,? extends V>map)

put,get,size,isEmpty,containsKey,containValue,remove

Objective : Create AccountMap
Identify key & value type
create empty unsorted map(HashMap<K,V>) & populate the same

Disp all entries of HM ---can use only toString
1.get acct summary --- i/p --id o/p --- err / dtls
2.Withdraw --- specify Account id & Amt ---- o/p : update acct dtls if acct exists o.w err msg or exc
3.funds transfer ---
i/p sid,dest id, amt

4.remove --- account
i/p id
5.Apply interest on on saving type of a/cs.
or
display all accts created after date.

Attach for-each to map & observe.
Sort the map as per : asc order of accts Ids.
Sort the map as per : desc order of accts Ids
Sort the accts as per : balance

If map sorting involves key based sorting criteria --- can be sorted by converting into TreeMap
Constructors of TreeMap
1. TreeMap() -- Creates empty map , based upon natural ordering of keys
2. TreeMap(Map<? extends K,? extends V> map)
Creates populated map , based upon natural ordering of keys
3. TreeMap(Comparator<? super K> c)
Regarding generic syntax & its usage in TreeMap constructor.
<? super K>
? --- wild card --- any unknown type
super --- gives lower bound
K --- key type
? super K --- Any type which is either K or its super type.

TreeMap(Comparator<? super K> c) --- creates new empty TreeMap, which will sort its element as per custom ordering(i.e will invoke compare(...) of Key type)

<? extends K>
? -- any type or wild card
extends -- specifies upper bound
K -- key type
? extends K --- Any type as Key type or its sub type.

same meaning for <? extends V>
TreeMap(Map<? extends K,? extends V> m)

disp acct ids of all accounts ---impossible directly....(will be done by Collection view of map @ the end)
Apply interest to all saving type a/cs

difficult directly ---so get a collection view of the map & sort the same.

Limitations on Maps
1. Maps can be sorted as per key's criteria alone.
2. can't attach iterators/for-each(till JDK 1.7)/for
3 Maps can be searched as per key's criteria alone.

To fix --- get a collection view of a map (i.e convert map to collection)

API of Map i/f

1. To get set of keys asso. with a Map
Set<K> keySet();
2. To get collection of values from a map
Collection<V> values();
3. To get set of Entries(key & val pair) ---
entrySet
Set<Map.Entry> entrySet()
Methods of Map.Entry
K getKey()
V getValue()

7. conversion from collection to array
Object[] toArray() -- non generic version --- rets array of objects

T[] toArray(T[] type)
T = type of collection .
Rets array of actual type.

8. sorting lists --- Natural ordering creiteria
Using java.util.Collections --- collection utility class.
static void sort(List<E> l1) ---sorts specified list as per natural sorting criteria.

DAY 14 :

Today's Topics

Hashing algorithm
LinkeHashSet & TreeSet
Java 8 Date/Time Handling API
Map i/f & implementation classes

1.Revise Sets overview
(refer to diagram : "Regarding Sets.png")

2. HashSet & hashing algorithm

Any problems observed in TestStringSet ?
Any problems observed in EmpSet ?

Solution

What is hashing ?
Hashing means using some function or algorithm to map object data to some representative integer value.

Objective : Emp : PK : composite PK
id (int) n deptId (string)
Emps are same iff : id n deptId is same

3. LinkedHashSet & TreeSet

4. Java 8 Date/Time related APIs
Java 8 New Features
java.time : new package is introduced
LocalDate : Date (immutable)(yr-mon-day) : inherently thrd safe.
API
1. public static LocalDate now()
Obtains the current date from the system clock in the default time-zone.
eg :

LocalTime : Time (immutable)(hr-min-sec) : inherently thrd safe.
LocalDateTime : Date n Time : inherently thrd safe.
eg : sop("curnt date "+now());
2. public static LocalDate of(int year,int month,int dayOfMonth)
Obtains an instance of LocalDate from a year, month and day.
eg : ?????
3. public static LocalDate parse(CharSequence text)
Obtains an instance of LocalDate from a text string such as 2007-12-03.
eg : LocalDate dt=LocalDate.parse(sc.next());//0 based dates
4. Methods :
isBefore,isAfter,isEqual
5. Can you change default Date format ? : YES

6. Computing Age

5. Enter Maps
Maps Overview
Refer to Map API

Objective :

5.1Store a/c details in a suitable map to ensure const time performance for :
put/get/remove...
What will the type of the map ? : HashMap<K,V>
K : acct no (int) ---> Integer
V : Account details (BankAccount)
eg : HashMap<Integer,BankAccount>

5.2 Create a populated map with sample data.

5.3 Create A/C

5.4 Display details of all accts

5.6 Get A/C summary

5.7 Funds Transfer

5.8 Close A/C

Example of strings having same hashcode
Aa BB

Ba CB
Ca DB
Da EB
BBBB AaBB AaAa BBAa
FB Ea

MAP API :
Map Overview (refer to the diagram "regarding Maps")

Map Implementation class
1. HashMap<K,V>
1.1 Constructors
1. HashMap()
2. HashMap(int initCapa)
3. HashMap(int initCapa,float loadFactor)
4. public HashMap(Map<? extends K,? extends V> m)
Meaning : Creates populated HashMap of type K,V from any Map (AL/LL/Vector : javac err
HS/LHS/TS : javac err
HM/LHM/TM : no error) having generic type of K or its sub type & V or its sub type

Steps n API

0. Create new empy map to store account details
HashMap<Integer,BankAccount> hm=new HM<>();//size=0,capa=16,L.F=0.75

0.5 Create new account
Map i/f API
1. public V put(K key,V value)
Meaning : It will insert the new entry into map.If key already exists : it will replace old value by new value.
Rets : null in case of new entry or old value ref. in case of existing entry.
eg : sop(map.put(k1,v1));//null
sop(map.put(k2,v2));//null
sop(map.put(k3,v3));//null
sop(map.put(k1,v4));//v1
//which entries : { k1:v4,k2:v2 ,k3:v3}

2. public V putIfAbsent(K key,V value)
eg : sop(map.putIfAbsent(k1,v1));//null
sop(map.putIfAbsent(k2,v2));//null
sop(map.putIfAbsent(k3,v3));//null
sop(map.putIfAbsent(k1,v4));//v1

//which values(entries) : k1:v1 k2:v2 k3:v3

3. public void putAll(Map<? extends K,? extends V> m)
eg : map1.putAll(map2);
Meaning : It will copy all entries from map2 ----> map1
(put : replace)

4. public V get(Object key)
Rets value type of ref if key is found else rets null.
eg : map.get(k2) : v2
map.get(k10) : null

5. boolean containsKey(Object key)
Returns true if this map contains a mapping for the specified key , otherwise false;
eg : map.containsKey(k1) ---true

6. boolean containsValue(Object value)
Returns true if this map maps one or more keys to the specified value.
eg : map : k1:v1 k2:v2 k3:v3
map.containsValue(v3) ---- true

containsKey : O(1)
containsValue : O(n)

7. public V remove(Object K)
Tries to remove the entry(=mapping=key n value pair) if key is found --rets existing value ref.
Rets null if key is not found.;
eg : map : k1:v1 k2:v2 k3:v3
sop(map.remove(k2));//v2
sop(map);//k1:v1 k3:v3
sop(map.remove(k20));//null
sop(map);//k1:v1 k3:v3

How to overcome limitations of Map (can't iterate over map , can't search/sort/remove by any value based criteria)

Solution : Convert the map into its Collection view

1. How to extract key type refs from a Map ?
public Set<K> keySet()
eg : HM<Integer,BankAccount> hm=new HM<>();
add some a/cs

Set<Integer> keys =hm.keySet();//O(n)

2. How to get value type of references from a Map ?
public Collection<V> values();
eg : HM<Integer,BankAccount> hm=new HM<>();
added some a/cs ....
Collection<BankAccount> accts =hm.values();//O(n)

3. How to get key-value pair(entry) of references from a Map ?
Map : i/f
Nested i/f : Map.Entry<K,V> : Entry in a Map
public Set<Map.Entry<K,V>> entrySet();

4. Method of Map.Entry i/f
public K getKey()
public V getValue();

HASHMAP README

How HashMap internally works in Java

Hash Map is one of the most used collection. It doesn't extend from Collection i/f.
BUT collection view of a map can be obtained using keySet,values or entrySet()

Internal Implementation

HashMap works on the principal of hashing.

What is hashing ?
Hashing means using some function or algorithm to map object data to some representative integer value.

Map.Entry interface ---static nested interface of Map i/f

This interface represents a map entry (key-value pair).

HashMap in Java stores both key and value object ref , in bucket, as an object of Entry class which implements this nested interface Map.Entry.

hashCode() -HashMap provides put(key, value) for storing and get(key) method for retrieving Values from HashMap.

When put() method is used to store (Key, Value) pair, HashMap implementation calls hashcode on Key object to calculate a hash that is used to find a bucket where Entry object will be stored.

When get() method is used to retrieve value, again key object is used to calculate a hash which is used then to find a bucket where that particular key is stored.

equals() - equals() method is used to compare objects for equality. In case of HashMap key object is used for comparison, also using equals() method Map knows how to handle hashing collision (hashing collision means more than one key having the same hash value, thus assigned to the same bucket. In that case objects are stored in a linked list (growable --singly linked)

Bucket term used here is actually an index of array, that array is called table in HashMap implementation. Thus table[0] is referred as bucket0, table[1] as bucket1 and so on.

HashMap uses equals() method to see if the key is equal to any of the already inserted keys (Recall that there may be more than one entry in the same bucket). Note that, with in a bucket key-value pair entries (Entry objects) are stored in a linked-list . In case hash is same, but equals() returns false (which essentially means more than one key having the same hash or hash collision) Entry objects are stored, with in the same bucket, in a linked-list.

In short , there are three scenarios in case of put() -

Using hashCode() method, hash value will be calculated. Using that hash it will be ascertained, in which bucket particular entry will be stored.
equals() method is used to find if such a key already exists in that bucket, if no then a new node is created with the map entry and stored within the same bucket. A linked-list is used to store those nodes.
If equals() method returns true, which means that the key already exists in the bucket. In that case, the new value will overwrite the old value for the matched key.

How get() methods works internally

As we already know how Entry objects are stored in a bucket and what happens in the case of Hash Collision it is easy to understand what happens when key object is passed in the get method of the HashMap to retrieve a value.

Using the key again hash value will be calculated to determine the bucket where that Entry object is stored, in case there are more than one Entry object with in the same bucket stored

as a linked-list equals() method will be used to find out the correct key. As soon as the matching key is found get() method will return the value object stored in the Entry object.

In case of null Key

Since HashMap also allows null, though there can only be one null key in HashMap. While storing the Entry object HashMap implementation checks if the key is null, in case key is null, it always map to bucket 0 as hash is not calculated for null keys.

HashMap changes in Java 8

Though HashMap implementation provides constant time performance O(1) for get() and put() method but that is in the ideal case when the Hash function distributes the objects evenly among the buckets.

But the performance may worsen in the case hashCode() used is not proper and there are lots of hash collisions. In case of hash collision entry objects are stored as a node in a linked-list and equals() method is used to compare keys. That comparison to find the correct key with in a linked-list is a linear operation so in a worst case scenario the complexity becomes O(n).

To address this issue in Java 8 hash elements use balanced trees instead of linked lists after a certain threshold is reached. Which means HashMap starts with storing Entry objects in linked list but after the number of items in a bucket becomes larger than a certain threshold, the bucket will change from using a linked list to a balanced tree, this will improve the worst case performance from O(n) to O(log n).

More details
Hash collision degrades the performance of HashMap significantly. With this new approach, existing applications can expect performance improvements in case they are using HashMaps having large number of elements by simply upgrading to Java 8.

Hash collisions have negative impact on the lookup time of HashMap. When multiple keys end up in the same bucket, then values along with their keys are placed in a linked list. In case of retrieval, linked list has to be traversed to get the entry. In worst case scenario, when all keys are mapped to the same bucket, the lookup time of HashMap increases from O(1) to O(n).

Java 8 has come with the following improvements/changes of HashMap objects in case of high collisions.

The alternative String hash function added in Java 7 has been removed.
Buckets containing a large number of colliding keys will store their entries in a balanced tree instead of a linked list after certain threshold is reached.

Above changes ensure performance of O(log(n)) in worst case scenarios (hash function is not distributing keys properly) and O(1) with proper hashCode().

How linked list is replaced with binary tree?

In Java 8, HashMap replaces linked list with a binary tree when the number of elements in a bucket reaches certain threshold. While converting the list to binary tree, hashcode is used as a branching variable. If there are two different hashcodes in the same bucket, one is considered bigger and goes to the right of the tree and other one to the left. But when both the hashcodes are equal, HashMap assumes that the keys are comparable, and compares the key to determine the direction so that some order can be maintained. It is a good practice to make the keys of HashMap comparable.

This JDK 8 change applies only to HashMap, LinkedHashMap and ConcurrentHashMap.

README HASHSET

Regarding Hashing based Data structures....(eg : HashSet,HashTable,HashMap)

Steps for Creating HashSet

1. Type class in HashSet must override : hashCode & equals method both in consistent manner.

Object class API
public int hashCode() --- rets int : which represents internal addr where obj is sitting on the heap(typically -- specific to JVM internals)

public boolean equals(Object ref) -- Object class rets true : iff 2 refs are referring to the same copy.

2. Rule to observe while overriding these methods
If 2 refs are equal via equals method then their hashCode values must be same.
eg : If ref1.equals(ref2) ---> true then ref1.hashCode() = ref2.hashCode()
Converse may not be mandatory.(i.e if ref1.equals(ref2) = false then its not mandatory that ref1.hashCode() != ref2.hashCode() : but recommended for better working of hashing based D.S)

String class , Wrapper classes , Date related classes have already folowed this contract.

Questions :
1. How does hashing based data structure ensure constant time performance?

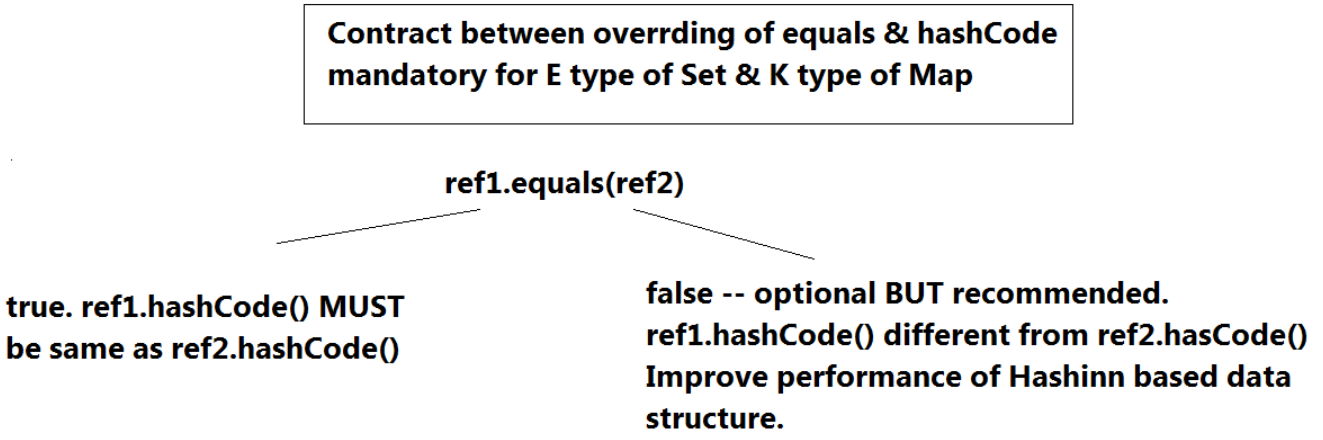
If no of elements(size) > capacity * load factor --- re-hashing takes place ---
New data structure is created --(hashtable) -- with approx double the original capacity --- HS takes all earlier entries from orig set & places them in newly created D.S -- using hashCode & equals. -- ensures lesser hash collisions.

2. Why there is a guarantee that a duplicate ref can't exist in yet another bucket ?

Answer is thanks to the contract between overriding of hashCode & equals methods

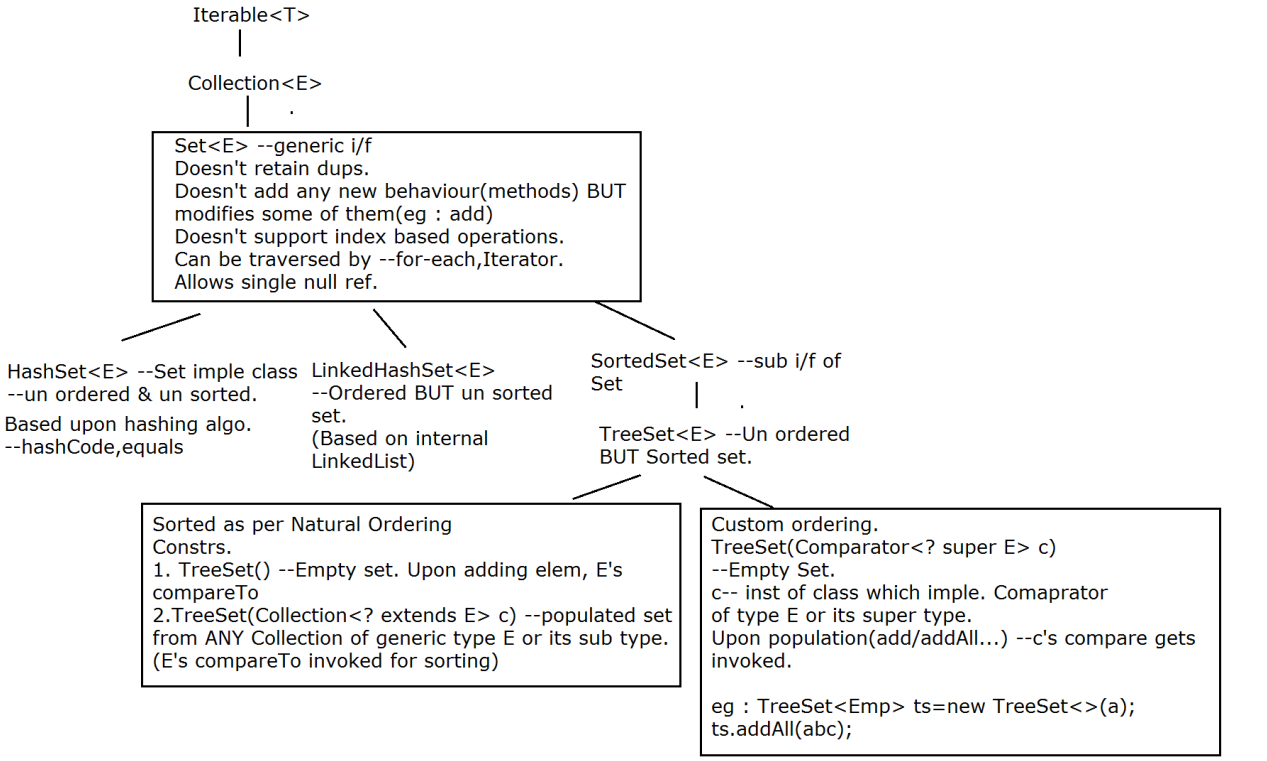
If two elements are the same (via equals() returns true when you compare them), their hashCode() method must return the same number. If element type violate this, then elems that are equal might be stored in different buckets, and the hashset would not be able to find elements (because it's going to look in the same bucket).

If two elements are different(i.e equals method rets false) , then it doesn't matter if their hash codes are the same or not. They will be stored in the same bucket if their hash codes are the same, and in this case, the hashset will use equals() to tell them apart.



Contract for overriding equals & hashCode methods--for correct working of HashSet/HashMap/Hashtable (any hashing based data structure)
Equal objects MUST produce SAME hash code.

HOW ?
1. Identify PK(object's identity) data members & override equals.
2. Use the same data members in overriding hashCode



Map<K,V>
K --Key Type , V -- Value Type
Map consists of Key & value pairs(refs)=entry(Map.Entry<K,V>)=mapping
Has distinct keys , can contain duplicate values.
Typically in HashMap<K,V> -- key's hashCode , governs where entry has to be inserted in the map.
One key ---> Value

HashMap<K,V>
--un sorted & un ordered Map.
Based upon hashing algo.
Ensures const time perf(O(1))
for --
put,putIfAbsent,get,remove,cotainsKey,size

thread un safe (un synchronized)

LinkedHashMap<K,V>
--un sorted BUT ordered (remembers order of insertion) Map.

SortedMap<K,V> --sub i/f

TreeMap<K,V> -- sorted BUT un ordered Map

1. TreeMap<K,V>() --empty map.
Upon population(put/putAll...) --Natural ordering --Key's compareTo invoked.

2. TreeMap<K,V>(Map<? extends K,? extends V) m) --- populated map of type <K,V> from ANY Map(eg :HM,LHM,Ht,TM..) having Key type as K or its sub-type & value type as V or its sub type.

3. TreeMap<K,V>(Comparator<? super K> c) --empty map . Upon population , invokes c's compare.

Limitations
1. Since Map deosn't extend Collection ---can't attach iterator & traverse the map directly.
2. Searching n sorting n removing as per Key based criteria -- directly supported.
For searching/sorting/removing as per value based criteria --involves converting map ---> collection(collection view of a map)

put(k,v)

New Entry
--inserts the entry
& rets null

Existing entry(same key already exists)
--replaces old value by new value & rets old value.

```
map.put(k1,v1);
map.put(k2,v2);
map.put(k1,v3);
```

```
Set<K> keys=map.keySet();
Collection<V> vals=map.values();
Set<Map.Entry<K,V>>
entries=map.entrySet();
```

putIfAbsent(K,V)

New entry --- inserts the entry & rets null

Existing entry(same key alrdy exists) --- doesn't replace old entry by new.
returns value ref.

HashMap Internals
HashMap<I,A> hm=new HM<>();

def init capa=16 & Load Factor =.75

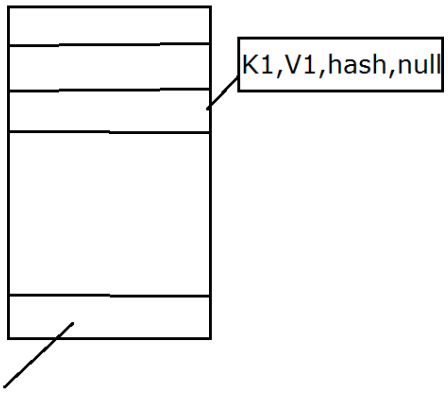
hm.put(k1,v1);

hm.put(k2,v2);

hm.put(k5,v5);

hm.get(k5)

JVM invokes
k1.hashCode() -- (some internal hash computation) % capacity ---Bucket ID
JVM Checks For Bucket



Empty (null)
Inserts the entry LinkedList --node

put() Rets null

non-empty (non null) hash collision
JVM invokes key's equals(eg : k5.equals (k1))

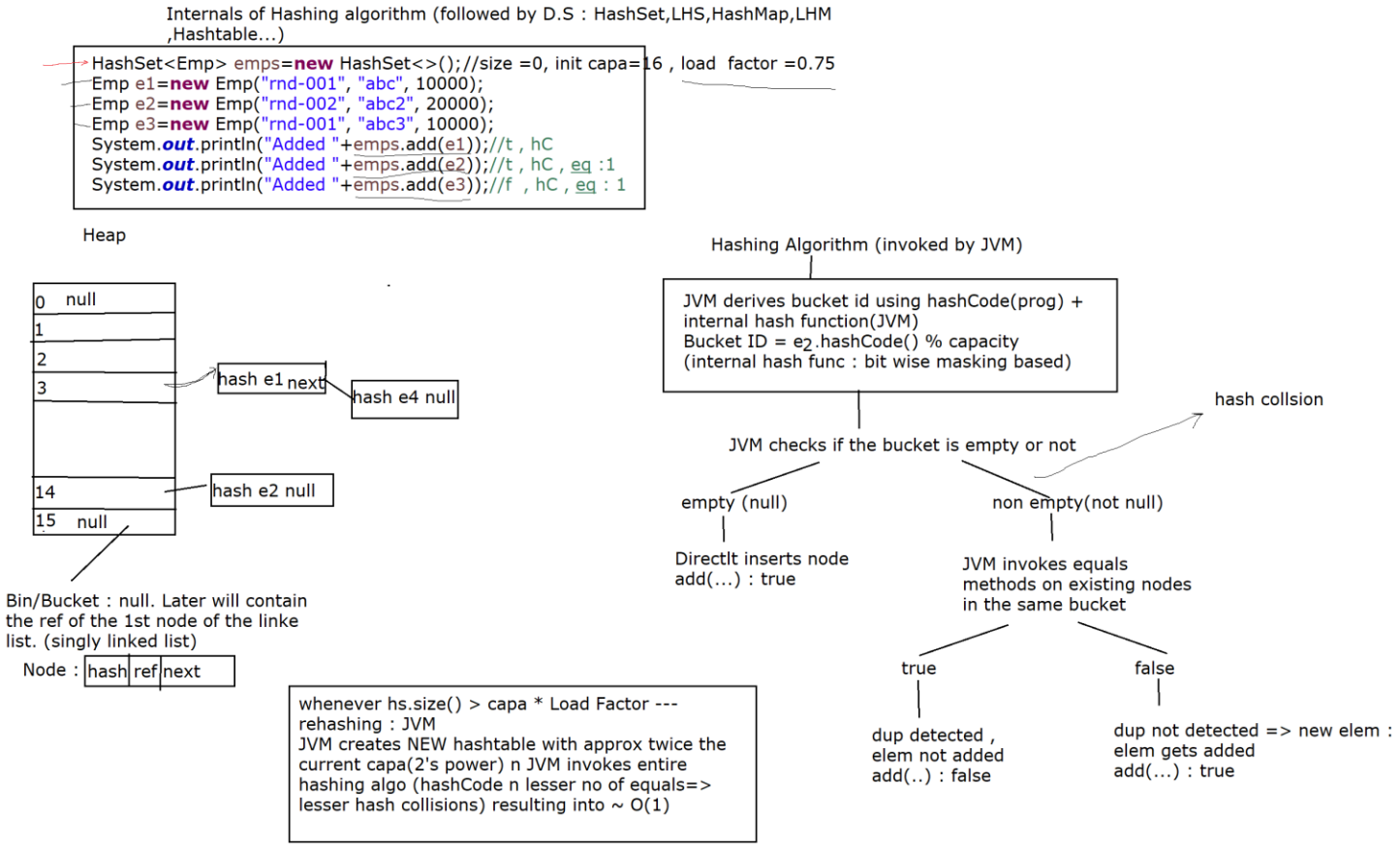
true --- same Key alrdy exists in the same bucket.
Replaces old value by new val & rets old val ref.
Rets non - null
eg : v1

false -- new entry gets inserted.
k1,v1,hash,next---k5,v5,hash,null

Rets --NULL

Bucket -- null
Node -- Map.Entry <K,V> + Next Ptr + hash
LinkedList<Node>

In case of increasing hash collisions --- HM can't ensure const time performance.
If total no of entries(map.size()) > capa * L.F --- Re -hashing takes place. JVM create new HM -- typically twice no of buckets --all entries copied into new HM .



DAY 15 :

Any Specific questions ?
Q & A session

Today's topics
HashMap Implementation
Java 8 New Features
Lambda Expressions
Java 8 Streams

Revise

Why Hashing ?

Hashing is a technique to make things more efficient by effectively narrowing down the search

What is hashing?

Hashing means using some function or algorithm to map object data to some representative integer value.

This hash code (or simply hash) can then be used as a way to narrow down our search when looking for the item in the hash-based data structure (eg : HashSet,LinkedHashSet,HashMap,LinkedHashMap,Hashtable...)

If you want to add elements(refereneces) in hashing based data structure, which contract to be followed between hashCode n equals?

Mandatory : Equal objects MUST produce SAME hash code

Optional BUT Recommended : Un equal objects SHOULD produce (for the improvemnt of hashtable) distinct hashcodes.

HOW ?
1. Identify PK n override equals
2. Use same fields (PK) in generating hashcode
Hint : Use prime nos in generation of hash codes
 eg : String , Wrapper class(Integer,Double....) , Date , LocalDate....
: Have already followed contract(both of the branches)

eg : HashMap<Integer,BankAccount> accts=new HM<>();//size=0,capa=16, L.F =.75
sop(hm.put(101,new BankAccount(101,.....));//new entry : inserted Node : hash | k1 | v1 | next=null
rets null
sop(hm.put(101,new BankAccount(101,.....));//existing entry : replaces old value by new value , rets old val ref.

sop(hm.putIfAbsent(101,new BankAccount(101,.....));//existing entry : DOES NOT replace old value by new value , rets old val ref.

Objective : Banking application using HashMap

BankAccount : acctNo , type(enum) , balance , creationDate , customerName
ctor, toString
B.L
withdraw,deposit, transferFunds...

1.1Store a/c details in a suitable map to ensure const time performance for : put/get/remove...
What will the type of the map ? : HashMap<K,V>
K : acct no (int) ---> Integer

V : Account details (BankAccount)
eg : HashMap<Integer,BankAccount>

1.2 Create a populated map with sample data.

1.3 Create A/C

1.4 Display details of all accts

1.5 Get A/C summary
i/p acct number => searching by key based criteria : ready made method : get

1.6 Funds Transfer

1.7 Close A/C

1.8 Display all account details of specific account type
i/p : a/c type
search criteria : acType : value based criteria ---convert it into Collection view ---> iterate n filter

1.9 Search A/cs by balance > specific balance , display their details.

1.10 Remove all loan accounts.
remove --value based criteria --convert into Collection view : values --
Collection<BankAccount>

Refer to "java8_new_features.txt"

Refer to "regarding lambdas"

Objectives

1. Create your own functional interface n use it in lambda expression
eg : Perform ANY arithmetic operation on 2 double values & return the result
eg --add/multiply/subtract/divide....

1.1 Create i/f : SAM => Functional i/f
1.2 Create an i/f : ComputationUtils : to add a static method
to return result of ANY operation performed on 2 double operands
static double computeResult(double a,double b,Operation op)
{

```
 return op.performAnyOperation(a,b);
}
```

1.3 Create a Tester n use it.

Explore Existing higher order functions

2. Collection : forEach

3. Collection : removeIf

4. Map : forEach

5. Sorting : custom ordering

---

1. How to create stream of ints from array?
java.util.Arrays class method
public static IntStream stream(int[] array)
Returns a sequential IntStream with the specified array as its source.

2. How to create stream(sequential) from Collection ?
Collection i/f method
public Stream<E> stream()

3.How to create stream(parallel) from Collection ?
Collection i/f method
public Stream<E> parallelStream()

IntStream Methods

1. public void forEach(IntConsumer action)
Performs an action for each element of this stream.

Solve :
1.Create int[] ---> IntStream & display its contents.
2. Create AL<Integer> , populate it.
Convert it to seq stream & display elems
2.5 Convert it to parallel stream & display elems
3. Create stream of ints between 1-100 & display all even elements.
(Hint : IntStream methods --range,filter,forEach)

4. Create IntStream from a fixed size list (un sorted) of integers , sort n display it.

4. Display all product names of a particular category , exceeding specific price.
I/P category name & price.
(stream,filter,forEach)
eg : productList : List<Product>

5. Display sum of all even nos between 1-100 .

6. Display average of all even nos between 1-100 .

7.Print avg of odd numbers between 10 & 50 ((if strm is empty print -100 or else print avg)
(filter,average,optional)

8. Display sum of product prices of a specific category.
I/P category name
o/p sum.

8.5 Accept 10 numbers from user & add them to IntStream
Print if all the numbers are > 20.

8.1 Create Student class -- rollNo(string),name,dob(LocalDate),subject(enum),gpa(double)
Create Subject enum --JAVA,DBT,ANGULAR,REACT,SE
Add constr & to string & getters.
8.2 Create CollectionUtils ---to ret populated FIXED size list of students (4 records)

8.3 Write a tester to print avg gpa of students opted for subject JAVA
(Hint : filter,map,reduce)
8.4 Print name of specified subject topper
(filter reduce -- max)
8.5 Print names of failures for the specified subject chosen from user.
(gpa < 5 : failed case)

8.7 How many distinctions for a specific subject
(gpa > 7.5 : distinction)
filter,count

Exam objectives
Important Facts :
1. Streams are implicitely closed , after terminal operation (i.e they can't be re used after terminal operation)
Otherwise throws --IllegalStateException (reason :stream has already been operated upon or closed)
Where as , collections are re-usable.

2. Streams follow , vertical execution order.

3. Streams support lazy evaluation (meaning none of the intermediate operations are performed , until its closed by terminal operation)

Method References in Java 8

Method reference is a shorthand notation of a lambda expression to call a method.
Can all lambda expressions be concised into method reference ? NO

eg :
If your lambda expression is like this:

s -> System.out.println(s)
then you can replace it with a method reference like this:
(since we are directly calling an existing method in a lambda expression , we can refer to the method itself)

System.out::println
The :: operator is used in method reference to separate the class or object from the method name

Four types of method references
1. Method reference to an instance method of an object – object::instanceMethod
2. Method reference to a static method of a class – Class::staticMethod
3. Method reference to an instance method of an arbitrary object of a particular type – Class::instanceMethod

4. Method reference to a constructor – Class::new

1. Method reference to an instance method of an object

```
@FunctionalInterface
interface MyInterface{
   void display();
}
public class Example {
   public void myMethod(){
         System.out.println("Instance Method");
   }
   public static void main(String[] args) {
         Example obj = new Example();
         // Method reference using the object of the class
         MyInterface ref = obj::myMethod;
         // Calling the method of functional interface
         ref.display();
   }
}
```

2. Method reference to a static method of a class

```
import java.util.function.BiFunction;
class Multiplication{
  public static int multiply(int a, int b){
         return a*b;
  }
}
public class Example {
  public static void main(String[] args) {
         BiFunction<Integer, Integer, Integer> product = Multiplication::multiply;
         int pr = product.apply(11, 5);
         System.out.println("Product of given number is: "+pr);
  }
}
```

3. Method reference to an instance method of an arbitrary object of a particular type

```
import java.util.Arrays;
public class Example {

  public static void main(String[] args) {
         String[] stringArray = { "aa", "bb", "cc", "dd", "ee"};
         /* Method reference to an instance method of an arbitrary
```

```
         * object of a particular type
         */
//Arrays.sort(stringArray, (s1,s2)->s1.compareTo(s2));
        Arrays.sort(stringArray, String::compareTo);
        Arrays.stream(stringArray).forEach(System.out::println);
  }
}
```

4. Method reference to a constructor

```
@FunctionalInterface
interface MyInterface{
   Hello display(String say);
}
class Hello{
   public Hello(String say){
      System.out.print(say);
   }
}
public class Example {
   public static void main(String[] args) {
         //Method reference to a constructor
      MyInterface ref = Hello::new;
      ref.display("Hello World!");
   }
}
```

4.1

```
@FunctionalInterface
interface MyFunctionalInterface {

   Student getStudent();
}
class Student {

   private String name;

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }
```

```
}
```
Following example uses constructor reference.

```
public class ConstructorReferenceDemo {

   public static void main(String[] args) {

      MyFunctionalInterface ref = Student::new;

      Supplier<Student> s1 = Student::new;// Supplier Example
      Supplier<Student> s2 = () -> new Student();// equals to above line

      System.out.println(ref.getStudent());//Student class toString() call
      System.out.println(s1.get());//Student class toString() call
   }
}
```

Lambda expressions

It's derived from lambda caculus.
It was a big change in calculus world , which gave tremedous ease in maths

Now the same concept is being used in programming languages.
1st language to use lambda
LISP
c , c++ , c# , scala , javascript , python

Finally in java also(Java SE 8 onwards)

Background

Java is an object-oriented language. With the exception of primitive data types, everything in Java is an object. Even an array is an Object. Every class creates instances that are objects. There is no way of defining just a function / method which stays in Java all by itself. There is no way of passing a method as argument or returning a method body for that instance.
i.e passing the behaviour was not possible till java 8.

It was slightly possible using anonymous inner classes --but that still required us to write a class !

What is lambda expression ?

Concise anonymous function which can be passed around
It has
1. list of params
2. body
3. return type.(optional)

Lambda expressions in Java is usually written using syntax (argument) -> (body). For example:

(type1 arg1, type2 arg2...) -> { body }

Following are some examples of Lambda expressions.

1.(int a, int b) -> { return a + b; }
OR can be reduced to
(int a, int b) -> a + b
OR further can be reduced to
(a,b) -> a+b

2. () -> System.out.println("Hello World")

3. s -> System.out.println(s)

4. () -> 42

5. () -> 3.1415

Above is just a syntax of lambda . But how to use them ?
Answer is ---You can use lambda expressions as targets of functional i/f reference.

Why lambdas --
Easy way of passing a behaviour.
Till Java SE 7 , there was no way of passing a method as argument or returning a method body for that instance.
To enable this style of functional programming , lambdas are introduced.
How to pass a behaviour ?

What is a functional programming paradigm ?
A language where below features are supported.

Functions are treated as a first class citizens.
Meaning : You can
1.1 define anonymous functions
1.2 assign a function to a variable (function literal)
1.3 pass function as a parameter
1.4 return function as a return value

Why FP ?

1. To write more readable , maintainable , clean & concise code.
2. To use APIs easily n effectively.
3. To enable parallel processing

OOP uses imperative style of programming (where you will have to specify what's to be done & how --both) .
FP uses declarative style of programming (where you will just have to specify what's to be done

eg : Objective
Find out the average salary of emp from the specified dept.

How will you do it in imperative manner?

```
eg : List<Emp> l1=new AL<>();
l1.add(..);.....
String dept=sc.next();
double total=0;
int num=0;
for(Emp e : l1)
  if(e.getDept().equals(dept)) {
   total += e.getSal();
   num++;
  }
sop(total/num);
```

OR

Vs
How to do it in declarative style ?

eg : List<Emp> l1=new AL<>();
l1.add(..);.....

l1.stream().filter(e->e.getDept().equals(dept)).mapToDouble(Emp::getSal).average().getAsDouble()

Objective --
1. Perform ANY operation on 2 double values & return the result
eg --add/multiply/subtract/divide....

2. Convert from ANY src type to ANY dest type
eg : String ---> length
String ----> upper case string
celcius ---> fahrenheit (f=c*1.8+32)
Student ---> GPA
number ---> square root

How do we declare just behaviour in java ?

Refer to Summable interface , implementation & its conversion to lambda.

Using interfaces

1. Create generic interface Converter<F,T> to specify single abstract method --convert , from F ---> T

2. Create a Tester class (with main method)
Add a static method to test the converter.

I/P -- 1. conversion source type(From)
2. conversion behaviour

O/P -- conversion result.(To)

```
public static <F,T> testConverter(F from, Converter<F,T> c)
{
  return c.convert(from);
}
```

3. main(..) invokes this static method for testing Converter.
But what will be 2nd argument ?

Applying lambdas to refer to func i/f.
Objectives
1. Collections.sort
2. List -- forEach , removeIf

Main Differences between Lambda Expression and Anonymous class

1. One key difference between using Anonymous class and Lambda expression is the use of "this" keyword.

For anonymous class ‘this’ keyword resolves to anonymous class, whereas for lambda expression ‘this’ keyword resolves to enclosing class where lambda is written.

2. Another difference between lambda expression and anonymous class is in the way these two are compiled.
Java compiler compiles lambda expressions and convert them into private method of the class.

What is a Stream?

A sequence of elements from a source that supports data processing operations.

• Sequence of elements - Like a collection, a stream provides an interface to a sequenced set of values of a specific type.

• Source - Streams refer to collections, arrays, or I/O resources.

• Data processing operations - Supports common operations from functional programming languages.
• e.g. filter, map, reduce, find, match, sort etc

They have nothing to do with java.io -- InputStream or outputstream

The Streams also support Pipelining and Internal Iterations. The Java 8 Streams are designed in such a way that most of its stream operations returns Streams only. This help us creating chain of various stream operations. This is called as pipelining. The pipelined operations looks similar to a sql query.(or Hibernate Query API)

Concurrency is IMPORTANT. But it comes with a learning curve.

So , Java 8 goes one more step ahead and has developed a Streams API which allows us to use multi cores easily.

Parallel processing = divide a larger task into smaller sub tasks (forking), then processing the sub tasks in parallel and then combining the results together to get the final output (joining).

Java 8 Streams API provides a similar mechanism to work with Java Collections.

The Java 8 Streams concept is based on converting Collections to a Stream (or arrays to a stream) , processing the elements in parallel and then gathering the resulting elements into a Collection.

Most stream operations accept some kind of lambda expression parameter, a functional interface specifying the exact behavior of the operation. Most of those operations must be both non-interfering and stateless. What does that mean?

A function is non-interfering when it does not modify the underlying data source of the stream, e.g.

```
List<String> myList =Arrays.asList("a1", "a2", "b1", "c2", "c1");
myList.stream().filter(s -> s.startsWith("c")).map(String::toUpperCase) .sorted()
  .forEach(System.out::println);
```

In the above example no lambda expression does modify myList by adding or removing elements from the collection.

A function is stateless when the execution of the operation is deterministic,
e.g. in the above example no lambda expression depends on any mutable variables or states from the outer scope which might change during execution.

API
The starting point is java.util.stream.Stream i/f
Different ways of creating streams
1. Can be created of any type of Collection (Collection, List, Set):

java.util.Collection<E> API

1.1 default Stream<E> stream()
1.2 public default Stream<E> parallelStream()

NOTE that Java 8 streams can't be reused, will raise IllegalStateException

2. Stream of Array
How to create stream from an array?

Arrays class API
public static <T> Stream<T> stream(T[] array)
Returns a sequential Stream with the specified array as its source.

3. Can be attached to Map ,via entrySet method.
Refer to CreateStreams.java

4. To create streams out of three primitive types: int, long and double.
As Stream<T> is a generic interface , can't support primitives.
So IntStream, LongStream, DoubleStream are added.
API of java.util.stream.IntStream
4.1 static IntStream of(int... values)
Returns a sequential ordered stream whose elements are the specified values.

4.2 static IntStream range(int startInclusive,int endExclusive)
Returns a sequential ordered IntStream from startInclusive (inclusive) to endExclusive (exclusive) by an incremental step of 1.

4.3 static IntStream rangeClosed(int startInclusive,int endInclusive)
Returns a sequential ordered IntStream from startInclusive (inclusive) to endInclusive (inclusive) by an incremental step of 1.

5. To perform a sequence of operations over the elements of the data source and aggregate their results, three parts are needed – the source, intermediate operation(s) and a terminal operation.

6.java.util.stream.Stream<T> i/f API

6.1 Stream<T> skip(long n)
Returns a stream consisting of the remaining elements of this stream after discarding the first n elements of the stream(stateful intermediate operation)

6.2 map
<R> Stream<R> map(Function<? super T,? extends R> mapper)
Returns a stream consisting of the results of applying the given function to the elements of this stream(intermediate stateless operation)

mapToInt
IntStream mapToInt(ToIntFunction<? super T> mapper)
Returns an IntStream consisting of the results of applying the given function to the elements of this stream.

6.3 filter

Stream<T> filter(Predicate<? super T> predicate)
Returns a stream consisting of the elements of this stream that match the given predicate.(intermediate stateless operation)

ref : StreamAPI1.java

---

7. Confirm lazyness of streams.
Intermediate operations are lazy. This means that they will be invoked only if it is necessary for the terminal operation execution.
ref : LazyStreams.java

8. Reduce operation
Readymade methods of IntStream
count(), max(), min(), sum(),average()

9. Customized reduce operation

ref : ReduceStream.java

10 collect

Reduction of a stream can also be executed by another terminal operation – the collect() method.
eg : StreamCollect.java
Good examples in java.util.stream.Collectors -api docs.

Details ---

1. Streams are functional programming design pattern for processing sequence of elements sequentially or in parallel.(a.k.a Monad in functional programming)

2. Stream represents a sequence of elements and supports different kind of operations to perform computations upon those elements

3. Stream operations are either intermediate or terminal. Intermediate operations return a stream so we can chain multiple intermediate operations .
Terminal operations are either void or return a non-stream result.

4. They can't be reused.

5. Collections vs Streams:
Collections are in-memory data structures which hold elements within it. Each element in the collection is computed before it actually becomes a part of that collection. On the other hand Streams are fixed data structures which computes the elements on-demand basis.

The Java 8 Streams -- lazily constructed Collections, where the values are computed when user demands for it.
Actual Collections behave absolutely opposite to it and they are set of eagerly computed values (no matter if the user demands for a particular value or not).

Regarding functional programming

What is functional programming ?

Functional programming is the way of writing s/w applications that uses only pure fuctions & immutable values.

Main concepts of FP are
1. Pure functions & side effects
2. Referential transperency
3. First class functions & higher order funcs.
4. Anonymous functions
5. Immutability
6. Recursion & tail recursion
7. Statements
8. Strict & Lazy evaluations
9. Pattern Matching
10 Closures

Why Functional Programming paradigm
Elegance and simplicity
Easier decomposition of problems
Code more closely tied to the problem domain

Through these , one can achieve :

Straightforward unit testing
Easier debugging
Simple concurrency

---

1. Addition of "default" keyword to add default method implementation , in interfaces.

Java 8 enables us to add non-abstract method implementations to interfaces by utilizing the default keyword. This feature is also known as Extension Methods.

Why default keyword ?
1. To maintain backward compatibility with earlier Java SE versions
2. To avoid implementing new functionality in all implementation classes.
eg : Java added in Iterable<T> interface
default void forEach(Consumer<? super T> action) -- as a default method implementation

eg :

```
interface Formula {
   double calculate(double a);//public n abstract
         //public
   default double sqrt(double a,double b) {
      return Math.sqrt(a+b);
   }
}
```

In case of ambiguity or to refer to def imple. from i/f -- use
InterfaceName.super.methodName(...) syntax

2 Can add static methods in java interfaces --- It's a better alternative to writing static library methods in helper class(eg --Arrays or Collections)

Such static methods can't be overridden in implementation class.
BUT can be re-declared.

They have to be invoked using interface name , even in implementation or non implementation classes.(otherwise compiler error)

3. Functional interfaces ---An interface which has exactly single abstract method(SAM) is called functional interface.
eg Runnable,Comparable,Comparator,Iterable,Consumer,Predicate...
New annotation introduced -- @FunctionalInterface
(since Java SE 8)
Functional i/f references can be substituted by lambda expressions, method references, or constructor references.

Solve -- Is following valid functional interface ?

public interface A { double calc(int a,int b);} : YES (SAM)
public interface B extends A {} : //YES : inherting SAM : calc
public interface C extends A { void show();} : NO , 2 abstract methods :calc n show
public interface D {} --NO (marker)
public interface E extends A {default void show(){}
static void test() {...}
} --YES : SAM : calc

4. Lambda Expressions

Objective -- Perform add/subtract/multiply/divide ... operations on 2 double i/ps
& return result.

How ?

refer to readme of lambda expressions.

Functional programming In java

Functional Programming (FP) is one of the type of programming pattern that helps the process of building application by using of higher order functions, avoiding shared state, mutable data

Functional programming vs OOP

Declarative vs Imperative :

Functional programming is a declarative pattern, meaning that the program logic is expressed without explicitly describing the flow control.Imperative programs spend lines of code describing the specific steps used to achieve the desired results — the flow control

Declarative programs remove the flow control process, and instead spend lines of code describing the data flow.

In Functional programming

Functions are treated as a first class citizens.
Meaning : You can
1.1 define anonymous functions

1.2 assign a function to a variable (function literal)
1.3 pass function as a parameter
1.4 return function as a return value

Why FP ?

1. To write more readable , maintainable , clean & concise code.
2. To use APIs easily n effectively.
3. To enable parallel processing

OOP uses imperative style of programming (where you will have to specify what's to be done & how --both) .
FP uses declarative style of programming (where you will just have to specify what's to be done

2. Functional interfaces
An interface which has exactly single abstract method(SAM) is called functional interface.

eg Runnable,Comparable,Comparator,Iterable,Consumer,Predicate,Supplier,Function...
Java SE 8 has introduced a new package for functional i/f
java.util.function

New annotation introduced -- @FunctionalInterface
(since Java SE 8)

Functional i/f references can be substituted by lambda expressions, method references, or constructor references.

Solve -- Is following valid functional interface ?
public interface A { double calc(int a,int b);} :
public interface B extends A {} :
public interface C extends A { void show();} :
public interface D {} -- Marker / empty / tag i/f :
public interface E extends A {default void show(){}} --

13. Addition of "default" keyword to add default method implementation , in interfaces.

Java 8 enables us to add non-abstract method implementations to interfaces by using the default keyword. This feature is also known as Extension Methods.

Why default keyword ?
1. To maintain backward compatibility with earlier Java SE versions

2. To avoid implementing new functionality in all implementation classes.

eg : Java added in Iterable<T> interface
default void forEach(Consumer<? super T> action) -- as a default method implementation

eg :

```
interface Formula {
   double calculate(double a);//javac adds implicit keywords : public n abstract
         //javac adds implicit keyword public
   default double sqrt(double a,double b) {
      return Math.sqrt(a+b);
   }
}
```

Q : If you write an implementation class MyFormula

```
public class MyFormula imple Formula
{
 .....
}
```

Which methods have to be implemented to avoid javac err?
1. calculate
2. sqrt
3. both
4. neither

Q : Can MyFormula class override the def. of sqrt ?

1 Display all elements of ArrayList
forEach
2. Create AL of integers
remove all odd numbers.

3. Create AL of emps
Remove underperforming employees (performance index < 7)
Dsiplay the list

4. Enter Java 8 Streams
1.Create int[] ---> IntStream & display its contents.
2. Create AL<Integer> , populate it.
Convert it to seq stream & display elems
Convert it to parallel stream & display elems

3. Create stream of ints between 1-100 & display all even elements.
(Hint : IntStream methods --range,filter,forEach)

4. Display all emp names from a particular dept , joined after specific date
(stream,filter,forEach)

5. Display sum of all even nos between 1-100 .
(stream , filter ,sum)
6. Display sum of salaries of all emps from a specific dept

7. Create a supplier of random numbers
eg :

```
Supplier<Double> randomValue = () -> Math.random();

// Print the random value using get()
System.out.println(randomValue.get());
```

8. Create a supplier of LocalDate & Time

eg :

```
Supplier<LocalDateTime> s = () -> LocalDateTime.now();
LocalDateTime time = s.get();

System.out.println(time);

Supplier<String> s1 = () -> dtf.format(LocalDateTime.now());
String time2 = s1.get();

System.out.println(time2);
```

- **Stream Source**
  - Streams can be created from Collections, Lists, Sets, ints, longs, doubles, arrays, lines of a file
- Stream operations are either intermediate or terminal.
  - **Intermediate operations** such as filter, map or sort return a stream so we can chain multiple intermediate operations.
  - **Terminal operations** such as forEach, collect or reduce are either void or return a non-stream result.

Collections Vs Java 8 Streams

1. Collections(Set/List/Map) actually stores the elements (i.e references)	1. Doesn't store all the elements (represents only sequence of operations to be performed on elements
2. Support add/remove (API supports)	2. Stream API doesn't support add/remove operations.
3. Eagerly evaluated	3. Lazily evaluated.
4. Supports external iteration (using for-each/iterators)	4. Supports internal iteration
5. Don't support easy way for concurrent handling	5. Easy way for parallel processing
6. Can be iterated over multiple times.	6. Can't be iterated over multiple times ,can be processed only once.
7. Don't support aggregate functionality (eg : match,filter count,limit,reduce,flatMap ..)	7. Support functional features like match,filter....

DAY 16 :

Today's topics

Intro to Method reference
Java 8 Functional Streams
Enter I/O programming

Revise
1. Can you add default methods in i/f ? YES
Which keyword is added implicitly by javac ? : public

eg of default keyword
Iterable i/f
for internal iteration : forEach

2. What will happen ?
eg : interface A
```
{
 default void show()
 {
   sop("in A's show");
 }
}
 interface B
{
 default void show()
 {
   sop("in B's show");
 }
}
class C implements A,B {
  //javac forces imple class to override dup def method
@Override
public  void show()
 {
   sop("in imple cls' s show");
   //A.super.show();
 }
}
```

3. Can you add static methods in i/f ?
Which keyword is added implicitly by javac ? : public

3. Since Java 8 can you pass behavior directly to a method ? : YES
Can you return it from the method ? : YES
What is a higher order function/method ? Method/Func where u can pass or return the behavior .

Is the following functional i/f : YES
eg : interface A
```
{
  default void show()
 {
   sop("in A's show");
 }
   static isEven testMe(int num)
   {
     return num % 2 ==0;
   }
   double calc(double a,double b);
}
```
eg : class Test
```
{
   void testMe(int c,int d,A ref)
   {....}
}
```

```
Class Driver
{
    main() {
      Test t1=new Test();
      t1.testMe(a,b,(a,b) -> a/b);
    }

}
```

Objectives

1. Explore Existing higher order functions(methods)
Have you solved day15 's lambda expression based assignment ?
Let's revise / solve it now !

1.5 Function literal

2. Method references
refer to "regarding method reference.txt"

3. Using above API(higher order funcs) , can you chain multiple operations like sorting & displaying the sorted data ? NO
OR
Filter the elements n display filtered list.
Objective : Accept product category from user
remove all products from specified category n display remaining list

3. Enter Java 8 Streams
3.1 Refer to <streams> : diagrams
collections vs streams n streams basics
3.2 Refer to "streams_sequence"

Java 8 Functional streams

Streams are wrappers around a data source(eg : array , collection , file) ,
They allow us to operate with that data source and makes bulk processing convenient and fast.

A stream does not store data meaning is not a data structure. It also never modifies the underlying data source.

API – java.util.stream – supports functional-style operations on streams of elements, such as filter -map-reduce transformations on collections.

Typically Stream will have
source
intermediate operations(0 or multiple)
terminal opeartion (single)

Details

1. How to create stream of ints from array?
java.util.Arrays class method
public static IntStream stream(int[] array)
Returns a sequential IntStream with the specified array as its source.

2. How to create stream(sequential) from Collection ?
Collection i/f method
public Stream<E> stream()

3.How to create stream(parallel) from Collection ?
Collection i/f method
public Stream<E> parallelStream()

IntStream Methods

1. public void forEach(IntConsumer action)
Performs an action for each element of this stream.

Solve :
1.Create int[] ---> IntStream & display its contents.
2. Create AL<Integer> , populate it.
Convert it to seq stream & display elems
2.5 Convert it to parallel stream & display elems
3. Create stream of ints between 1-100 & display all even elements.
(Hint : IntStream methods --range,filter,forEach)
3.5. Create Stream<Integer> from a fixed size list (un sorted) of integers , sort n display it.

4. Display all product names of a particular category , exceeding specific price.
I/P category name & price.
(stream,filters,forEach)
eg : productList : List<Product>

4.5 Prompt user for category n discount.
Apply that discount on all products of specified category n print it.

5. Display sum of all even nos between 1-100 .

6. Display average of all odd nos between 1-100 .

7.Print avg of odd numbers between 10 & 50 ((if strm is empty print -100 or else print avg)
(filter,average,optional)

8. Display sum of product prices of a specific category.
I/P category name
o/p sum.

9. Display average of product prices of a specific category.
I/P category name
o/p sum.

10. Sort products of a specific category as per price n display their names.
i/p category name

Exam objectives
Important Facts of Java 8 Streams:
1. Streams are implicitely closed , after terminal operation (i.e they can't be re used after terminal operation)
Otherwise throws --IllegalStateException (reason :stream has already been operated upon or closed)
Where as , collections are re-usable.

2. Streams follow , vertical execution order.

3. Streams support lazy evaluation (meaning none of the intermediate operations are performed , until its closed by terminal operation)

I/O handling

Desc of FileInputStream --- java.io.FileInputStream
bin i/p stream connected to file device(bin/char) -- to read data.

Desc of FileOutputStream --- java.io.FileOutputStream
bin o/p stream connected to file device(bin/char) -- to write data.

Desc of FileReader--- java.io.FileReader
char i/p stream connected to file device(char) -- to read data.

Desc of FileWriter--- java.io.FileWriter
char o/p stream connected to file device(char) -- to write data.

Objective --- Read data from text file in buffered manner.

1. java.io.FileReader(String fileName) throws FileNotFoundException
--- Stream class to represent unbuffered char data reading from a text file.
Has methods -- to read data using char/char[]
eg -- public int read() throws IOException

public int read(char[] data) throws IOException
Usage eg-- char[] data=new char[100];
int no= fin.read(data);

public int read(char[] data,int offset,int noOfChars) throws IOException
Usage eg-- char[] data=new char[100];

int no= fin.read(data,10,15);
eg -- 12 chars available
no=12;data[10]----data[21]

1.5 FileReader(File f) throws FileNotFoundException
java.io.File -- class represents path to file or a folder.

2. Improved version -- Buffered data read .
For char oriented streams--- java.io.BufferedReader(Reader r)

API of BR ---
String readLine() --- reads data from a buffer in line by line manner-- & rets null at end of Stream condition.

Objective -- Replace JDK 1.6 try-catch-finally BY JDK 1.7 try-with-resources syntax.
Meaning --- From Java SE 7 onwards --- Introduced java.lang.AutoCloseable -- i/f
It represents --- resources that must be closed -- when no longer required.
i/f method
public void close() throws Exception-- closing resources.
java.io --- classes -- have implemented this i/f -- to auto close resource when no longer required.
syntax of try-with-resources
try (//open one or multiple AutoCloseable resources)
{ .....
} catch(Exception e)
{
}

confirm device independence of Java I/O --- replace File device by Console
i.e --- Read data from console i/p --- in buffered manner till 'stop' & echo back it on the console.

required stream classes --- BR(ISR(System.in))
Alternative is --- use Scanner class.

Adv. of Scanner over above chain ----- contains ready-made parsing methods(eg --- nextInt,nextDouble.....)
But Scanner is not Buffered Stream
Can combine both approaches.(new Scanner(br.readLine())

Objective --- Combine scanner & buffered reader api --- to avail buffering + parsing api. --- BufferedReader provides buffering BUT no simple parsing API. -- supplies br.readLine only

Scanner -- Can be attached to file directly
Constr -- Scanner(File f)
BUT no buffering .

How to use both?
Create BR br=new BR(new FR(...));
while ((s=br.readLine())!=null)
 {
   //scanner can be attached to string ---Scanner(String s)
Scanner sc=new Scanner(s);
// parse data using Scanner API --next,nextInt,nextBoolean
}

Overloaded constructor of FileReader(File f)

java.io.File ---- class represents path to file / folder
Regarding java.io.File -----
Does not follow stream class hierarchy, extends Object directly.
File class --- represents abstract path which can refer to file or folder.
Usage --- 1. To access/check file/folder attributes(exists,file or folder,read/w/exec permisssions,path,parent folder,create new empty file,create tmp files & delete them auto upon termination,mkdir,mkdirs,rename,move,size,last modified ,if folder---list entries from folder,filter entries)
Constructor ---
File (String path) ---
eg --- File f1=new File("abc.dat");
if (f1.exists() && f1.isFile() && f1.canRead())
...attach FileInputStream or FileReader

File (String path) ---
File class API --- boolean exists(),boolean isFile() , boolean canRead()

Objective --- Text File copy operation --- in buffered manner.

For writing data to text file using Buffered streams

java.io.PrintWriter --- char oriented buffered o/p stream --- which can wrap any device.(Binary o/p stream or Char o/p stream)
Constructors---
PrintWriter(Writer w) --- no auto flushing,no conversion, only buffering
PrintWriter(Writer w, boolean flushOnNewLine)--- automatically flush buffer contents on to the writer stream --upon new line
PrintWriter(OutputStream w) --- can wrap binary o/p stream -- buffering +conversion(char-->binary),no auto-flush option
PrintWriter(OutputStream w , boolean flushOnNewLine) ---
API Methods----print/println/printf same as in PrintStream class(same type as System.out)

Stream class which represents --- Char o/p stream connected to Text file. --- java.io.FileWriter
Constructor
FileWriter(String fileName) throws IOException -- new file will be created & data will be written in char format.
FileWriter(String fileName,boolean append) --- if append is true , data will be appended to existing text file.

Collection & I/O
Objective ---
Items Inventory
Item -- code(String-Prim key),desc,category,quantity,price,shipmentDate
constr,toString.

Create suitable collection of Items(HashMap) --- sort map as per desc item code ,& store sorted item dtls in 1 text file .
NOTE : individual item rec MUST be written on separate line.

Sort items as shipment Date & store sorted dtls in another file . Before exiting ensure closing of data strms .
(buffered manner)

Objective -- Restore collection of items created in above requirement ---in form of HashMap .
-- buffering is optional.

Objective --- using Binary file streams.
Classes --- FileInputStream -- unbuffered bin i/p stream connected to bin file device.
FileOutputStream --unbuffered bin o/p stream connected to bin file device.

But these classes --- dont provide buffering & have only read() write() methods in units of bytes/byte[]

API of InputStream class
1. int read() throws IOException
Will try to read 1 byte from the stream.
Data un-available method blocks.
Returns byte--->int to caller.
eg -- int data=System.in.read();
2. int read(byte[] bytes) throws IOException
Will try to read data from underlying stream.
Data un-available -- method blocks.
Rets actual no of bytes read.

eg :
byte[] bytes=new byte[100];
int no=System.in.read(bytes);
no data --method blocks.
10 bytes available -no =10;bytes[0]-----bytes[9]
110 bytes available -- no=100;bytes[0]....bytes[99]

3. int read(byte[] bytes,int offset,int maxNoOfBytes) throws IOException
Will try to read data from underlying stream.
Data un-available -- method blocks.
Rets actual no of bytes read.
eg :
byte[] bytes=new byte[100];
int no=System.in.read(bytes,10,15);
no data --BLOCKS
5 bytes available --no=5;bytes[10].....bytes[14]
110 bytes available -- no=15;bytes[10]..bytes[24]

4. int available() throws IOException
Returns no of available bytes in the stream
no data ---DOESN't block -- rets 0.

Important API of OutputStream
1. public void write(int byte) throws IOException

2. public void write(byte[] bytes) throws IOException
3. public void write(byte[] bytes,int offset,int maxNo) throws IOException
bytes[offset].....bytes[offset+maxNo-1] -- written out to stream
4. void flush() throws IOException
5. void close() throws IOException

Using BIS(BufferedInputStream) -- enables buffering BUT doesn't provide any advanced API(ie. read(), read(byte[]), read(byte[] b,int off,int len) . Same is true with BOS.(BufferedOutputStream)

Objective ---
Create Customer/Account based collection. Sort if reqd.
Store Sorted collection to bin file in buffered manner --
& re-store the same.

Use advanced streams in such cases ---
Mixed Data streams
java.io.DataOutputStream ---implements DataOutput i/f
(converter stream) prim types / string ---> binary
Constructor -- DataOutputStream (OutputStream out)
API ---
public void writeInt(int i) throws IOExc
public void writeChar(char i) throws IOExc
public void writeFloat,writeDouble.....
For Strings
public void writeUTF(String s) throws IOExc ---uses Modified UTF 8 convention
or
public void writeChars(String s) throws IOExc --- uses UTF16 convention

eg : Items Inventory
Item -- code(String-Prim key),desc,category,quantity,price,shipmentDate
constr,toString.

Objective ---
Customer data is already stored in bin file.
Read customer data from Bin file --- in buffered manner & upload the same in HM .display customer details.

Stream class --- java.io.DataInputStream -- implements DataInput
Conversion stream(converts from bin ---> prim type or String)

Constructor
DataInputStream(InputStream in)
API Methods
public int readInt() throws IOException
public double readDouble() throws IOException
public char readChar() throws IOException
public String readUTF() throws IOException(must be used with writeUTF)
public String readChars() throws IOException(must be used with writeChars)

Most Advanced streams ---
Binary streams which can read/write data from/to binary stream in units of Object/Collection of Object refs (i.e Data Transfer Unit = Object/Collection of Objects)

Stream Class for writing Objects to bin. stream
java.io.ObjectOutputStream implements DataOutput,ObjectOutput

Description --- ObjectOutputStream class performs serialization.
serialization= extracting state of object & converting it in binary form.

state of object = non-static & non-transient data members
Constructor
ObjectOutputStream(OutputStream out)
out--- dest Binary o/p stream --- where serialized data stream has to be sent.

API methods ---
public void writeInt(int i) throws IOExc
public void writeChar(char i) throws IOExc
public void writeFloat,writeDouble.....
For Strings
public void writeUTF(String s) throws IOExc ---uses Modified UTF 8 convention

public void writeObject(Object o) throws IOException,NotSerializableException

De-serialization---- conversion or re-construction of Java objs from bin stream.
java.io.ObjectInputStream --- performs de-serialization.--- implements DataInput,ObjectInput
Constructor --- ObjectInputStream(InputStream in)

API methods ---
readInt,readShort,readUTF,readChars..... +
public Object readObject() throws IOException

Objective --- attach OIS to the bin file using FIS & display customer data.

Objective :
Confirming concepts of serialization & de-serialization
Emp -- int id, String name,double salary,Address adr;
Address -- String state,city,street.

Objective -- Understanding Set & its implementation classes
HashSet -- based upon hashing algorithm

More involved scenario.
(store customer info & Items to be purchased)
Data members - int no,Customer info, AL<Item>, Date creationDate

---

Need -- In the absence of Object streams, if u want to persist(save in permanent manner) state of objects or application data in binary manner --- prog has to convert all data to binary & then only it can be written to streams.

Object streams supply ready made functionality for the same.

Persistence ---saving the state of the object on the binary stream.

Serialization/De-serialization

Ability to write or read a Java object to/from a binary stream

Supported since JDK 1.1

Saving an object to persistent storage(current example -- bin file later can be replaced by DB or sockets) is called persistence

Java provides a java.io.Serializable interface for checking serializability of java classes.(object)

Meaning --- At the time of serialization(writeObject) or de-serialization(readObject) --- JVM checks if the concerned object is Serializable(i.e has it implemented Serializable i/f) --if yes then only proceeds , otherwise throws Exception ---java.io.NotSerializableException

Serializable i/f has no methods and is a marker(tag) interface. Its role is to provide a run time marker for serialization.

Details
What actually gets serialized?
When an object is serialized, mainly state of the object(=non-static & non-transient data members) are preserved.

If a data member is an object(ref) , data members of the object are also serialized if that object's class is serializable

eg : If Item class HAS - A reference of ShippingAddress

The tree of object's data, including these sub-objects constitutes an object graph

eg : HM<String,Product> hm .....
out.writeObject(hm);
HM -- String --Product (id,name,price,qty,category +shippingDetails)

If a serializable object contains reference to non-serializable element, the entire serialization fails

If the object graph contains a non-serializable object reference, the object can still be serialized if the non-serializable reference is marked "transient"

Details --- transient is a keyword in java.
Can be applied to data member.(primitive or ref)
transient implies ---skip from serialization.(meant for JVM)
Usage -- To persist --partial state of the serializable object
---

If super-class is serializable, then sub-class is automatically serializable.

If super-class is NOT serializable --- sub-class developer has to explicitely write the state of super-class.

What happens during deserialization?

When an object is deserialized, the JVM tries to bring the object back to life by making a new object on the heap that has the same state the serialized object had at the time it was serialized. 1. (Class.forName("com.app.core.Account")--class loading purpose,
2. Class.newInstance(),
3. setting state of the object from bin stream)
The static/transient variables, which come back have either null (for object references) or as default
primitive values.
Constructor of serializable class does not get called during de-serialization.

What are pre-requisites for de-serialization?
.class file for Class Obj to be de-serialized + Bin data stream containing state.

What is serialversion UID?
Each time an object is serialized, the object (including every object in its graph) is 'stamped' with a version ID number for the object's class. The ID is called the serialVersionUlD, and it's computed based on information about the class structure. As an object is being deserialized, if the class has changed since the object was serialized, the class could have a different serialVersionUID, and deserialization will fail.(java.lang.InvalidClassException). But you can control this - by adding your own UID.

Serialization format overview
1. Magic no.
2. Serialization format version no.
3. Class desc -- class name,serial version uid,desc of data members(class signature)
4. State of the object.(non static & non transient data members)

Limitations
1. Java technology only
2. Difficult to maintain in case of changing class format
3. May lead to security leaks.

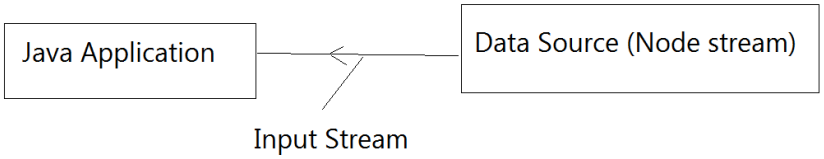
Types of Streams in java.io

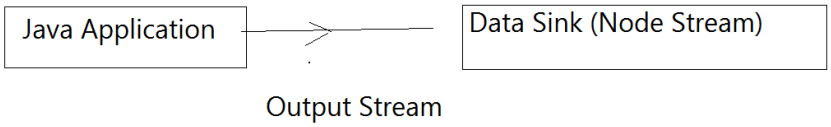
Node Streams (Device Handling streams)
eg : FileInputStream, PipedInputStream, Socket Streams

Buffering Streams -- used typically in larger data volumes . Can be added for reading as well as writing . JA , instead of reading data directly from a node stream, data can be read into a buffer & then into JA.
Similarly , JA can dump data into a buffer , when buffer becomes full, it can flush out the contents to device (node stream)

eg : BufferedInputStream,BufferedOutputStream,PrintWriter.

Conversion/Filtering streams
Instead of reading data in chunks of bytes or chars --formatting strms can convert the data into primitive types or java objects.
eg : DataInputStream DataOutputStream, ObjectOutputStream,ObjectInputStream





Scanner vs BufferedReader
Common Points -- Both represent buffered character input streams.

Scanner	BufferedReader -- a subclass of Writer
1. Smaller inherent buffer (~ 1K)	1. Larger inherent buffer (~ 8k)
2. Inherently thread un safe	2. Inherently thread safe
3. Does have parsing support (eg : nextInt,nextFloat....)	3. No parsing API

Scanner can be combined with BR , to achieve both of the advantages.
eg : Scanner sc=new Scanner(br.readLine()); & then parse....

java.io Overview (java.io)

Any run time err inread/write --java.io.IOException

java.io API --blocked API. Any read/write operation causes the blocking of invoker thread.

till the operation is complete.

Binary data streams

For read/write operation ,
data xfer unit is byte/byte[]

Char data streams

D.T.U --- char / char[]

InputStream OutputStream Reader Writer

abstract super classes

Bin i/p stream Bin o/p stream char i/p strm char o/p strm

FileInputStream FileOutputStream FileReader FileWriter

Node streams for handling files -- un buffered.

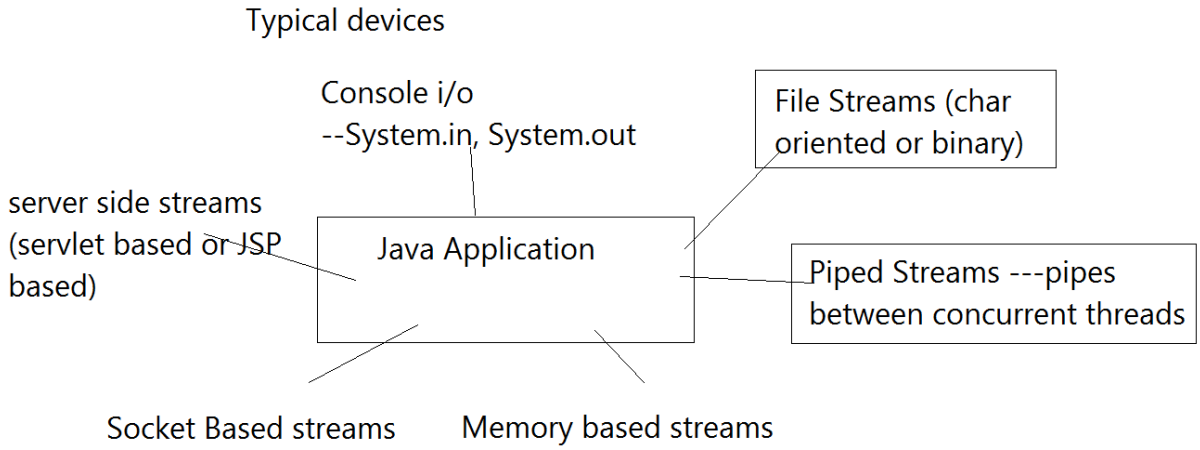
PipedInputStream PipedOutputStream PipedReader PipedWriter

Node strms for handling data pipes between thrds unbuffered

BufferedInputStream BufferedOutputStream BufferedReader BufferedWriter

PrintWriter

ObjectInputStream ObjectOutputStream



BufferedWriter vs PrintWriter

common --char o/p strms , buffered strms(have buffering capability.

BufferedWriter(Writer w, int size)
can specify size of buffer.

API(methods) --limited
write methods only

can only wrap char o/p strms (doesn't support the conversion char --bin)

does't support auto flushing

PrintWriter(Write w) -- can't specify size of buffer.(def size is supplied)

API -- write ... + print,println,printf

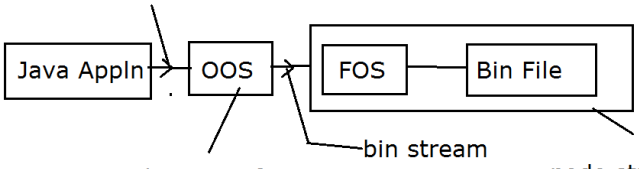
can wrap either char or bin strms
PrintWriter(Write w)
PrintWriter(OutputStream out)

Supports auto flushing(flush on new line)

PrintWriter(Writer w,boolean autoFlush)
PrintWriter(OutputStream out,boolean autoFlush)

---

prim types/strings/object/coll of refs

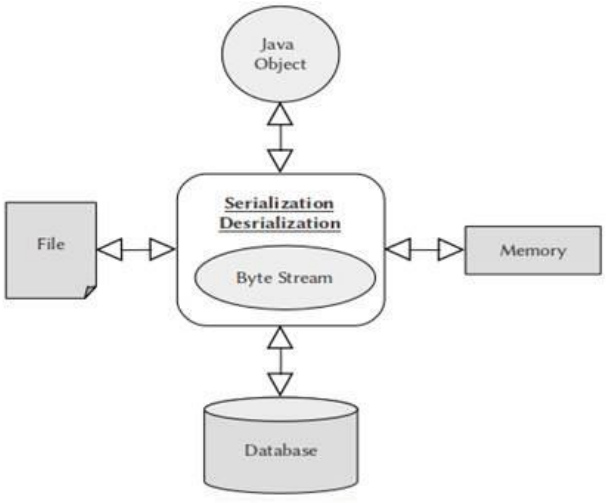


serializer--performs serialization (conversion of state(non static n non transient d.m) --> bin +add meta data info.

node stream -- device handling stream class.

```
OOS out=new OOS(new FOS(fileName));
out.writeObject(hm);
```

```
OIS in=new OIS(new FIS(fileName));
HM<I,S> hm=(HM)in.readObject();
```



What is the need of ObjectInputStream & ObjectOutputStream ?

To achieve Persistence.

Persistence=Saving the state of the java object in permanent manner.

In the absence of Object streams, if you want to persist(save in permanent manner) state of objects or application data in binary manner --- programmer has to convert all data to binary & then only it can be written to streams.

Object streams supply ready made functionality for the same.

Stream Class for writing Objects to bin. stream
java.io.ObjectOutputStream implements DataOutput,ObjectOutput

Description --- ObjectOutputStream class performs serialization.
serialization= extracting state of object & converting it in binary form.
(Details --Serialization literally refers to arranging something in a sequence. It is a process in Java where the state of an object is transformed into a stream of bits. The transformation maintains a sequence in accordance to the metadata supplied)

state of object = non-static & non-transient data members
Constructor
ObjectOutputStream(OutputStream out)
out--- dest Binary o/p stream --- where serialized data stream has to be sent.
API methods ---
public void writeInt(int i) throws IOExc
public void writeChar(char i) throws IOExc
public void writeFloat,writeDouble.....
For Strings
public void writeUTF(String s) throws IOExc ---uses Modified UTF 8 convention
+
public void writeObject(Object o) throws IOException,NotSerializableException

De-serialization---- conversion or re-construction of Java objs from bin stream.
java.io.ObjectInputStream --- performs de-serialization.--- implements DataInput,ObjectInput
Constructor --- ObjectInputStream(InputStream in)
API methods ---
readInt,readShort,readUTF,readChars..... +
public Object readObject() throws IOException,ClassNotFoundException,InvalidClassException

Serialization/De-serialization

Ability to write or read a Java object to/from a binary stream

Supported since JDK 1.1

Saving an object to persistent storage(current example -- bin file later can be replaced by DB or sockets) is called persistence

Java provides a java.io.Serializable interface for checking serializability of java classes.(object)

Meaning --- At the time of serialization(writeObject) or de-serialization(readObject) --- JVM checks if the concerned object is Serializable(i.e has it implemented Serializable i/f) --if yes then only proceeds , otherwise throws Exception ---java.io.NotSerializableException

Serializable i/f has no methods / data members and is a marker(tag) interface. Its role is to provide a run time marker for serialization.

Details
What actually gets serialized?
When an object is serialized, mainly state of the object(=non-static & non-transient data members) are preserved.

If a data member is an object(ref) , data members of the object are also serialized if that objects class is serializable

eg : If Product class HAS - A reference of ShippingAddress

The tree of objects data, including these sub-objects constitutes an object graph

eg : HM<String,Product> hm .....
out.writeObject(hm);
HM -- String --Product (id,name,price,qty,category +shippingDetails)

If a serializable object contains reference to non-serializable element, the entire serialization fails

If the object graph contains a non-serializable object reference, the object can still be serialized if the non-serializable reference is marked transient

Details --- transient is a keyword in java.
Can be applied to data member.(primitive as well as ref types)
transient implies ---skip from serialization.(meant for JVM)
During de-serialization ---transient(or even static) members are initialized to def values.
Usage -- To persist --partial state of the serializable object
---

If super-class is serializable, then sub-class is automatically serializable.

If super-class is NOT serializable --- super class must have a default constructor (otherwise InvalidClassException is thrown by JVM during de serilaization)
sub-class developer has to explicitely write the state of super-class.

What happens during deserialization?(in.readObject())

When an object is deserialized, the JVM tries to bring the object back to life by making a new object on the heap that has the same state the serialized object had at the time it was serialized.

1. (Class.forName("com.app.core.Account")--class loading purpose,
1.5 Matches incoming Serial version UID with the computed one
If matches --continues to steps 2.
Otherwise --- InvalidClassException is thrown.
2. If JVM comes across any non serializable super class , having no def constr --- InvalidClassExc is thrown.
Otherwise continues
Class.newInstance() or similar reflection API -- EMPTY/BLANK object is created on heap.
3. setting state of the object from bin stream)
The static/transient variables, which come back have either null (for object references) or as default primitive values.

4. Constructor of serializable class does not get called during de-serialization.
why ?

Think -- what is the need of constructor?
Constructor initializes the object variables with either default values or values which is assigned inside constructor. BUT we want to initialize the state of the object from binary stream.

What are pre-requisites for de-serialization?
Byte codes (.class file) for entire object graph to be de-serialized + Bin data stream containing state.

Details

Java Deserializing process says, "For serializable objects, the no-arg constructor for the first non-serializable supertype is run."

It means during deserialization process, JVM checks the inheritance class hierarchy of instance in process.

It checks, if the Class instance in process has implemented Serializable interface, If yes, then JVM will check Parent class(If any) of the instance to see if Parent has implemented Serializable interface, If yes, then JVM continues to check all its Parents upwards till it encounters a class which doesn't implement Serializable interface. If all inheritance hierarchy of instance has implemented Serializable interface as one shown above then JVM will end up at default extended Object class which doesn't implemented Serializable interface. So it will invoke a default constructor of Object class.

If in between searching the super classes, any class is found non-serializable then its default constructor will be used . If any super class of instance to be de-serialized is non-serializable and also does not have a default constructor then the java.io.InvalidClassException is thrown by JVM.

So till now we got the instance located in memory using one of superclasss default constructor. Note that after this no constructor will be called for any class. After executing super class constructor, JVM read the byte stream and use instances meta data to set type information and other meta information of instance.

After the blank instance is created, JVM first set its static fields and then invokes the default readObject() method (if its not overridden, otherwise overridden method will be called) internally which is responsible for setting the values from byte stream to blank instance. After the readObject() method is completed, the deserialization process is done and you are ready to work with new deserialized instance.

What is serialversion UID?

The serialVersionUID is a universal version identifier for a Serializable class. Deserialization uses this number to ensure that a loaded class corresponds exactly to a serialized object. If no match is found, then an InvalidClassException is thrown.

How to generate ?
serialver F.Q class name(for a class that imple. Serializable)
eg : serialver java.util.HashMap

Details
Each time an object is serialized, the object (including every object in its graph) is 'stamped' with a version ID number for the object's class. The ID is called the serialVersionUlD, and it's computed based on information about the class structure(class constructors,implemented interfaces,data members).

As an object is being deserialized, if the class has changed since the object was serialized, the class could have a different serialVersionUID, and deserialization will fail.(java.lang.InvalidClassException).

Since different java compilers or even different versions of java compilers can produce different serial version UID , its always recommended , that a programmer should add UID even in the 1st version of class & modify it whenever the class is modified substantially.

examples of incompatible changes
-- Deleting fields , Moving classes up or down the hierarchy ,changing a non-static field to static or a non-transient field to transient ,changing the declared type of a primitive field ....

examples of compatible changes
adding fields,adding classes,adding Serializable,modifying access specifer of the field....

Serialization format overview
Contents of serialized binary stream
It has all the information about the instance which was serialized by serialization process.
This information includes
class's meta data
type information of instance fields
values of instance fields as well.

This same information is needed when object is re-constructed back to a new object instance.
While deserializing an object, the JVM reads its class metadata from the stream of bytes which specifies whether the class of an object implements either 'Serializable' or 'Externalizable' interface.
Detailed format
1. Magic no.

2. Serialization format version no.
3. Class desc -- class name,serial version uid,desc of data members(
4. State of the object.(non static & non transient data members)

Limitations
1. Java technology only
2. Difficult to maintain in case of changing class format
3. May lead to security leaks.

Important facts of serialization n deserialization
1.
Transient and static fields are ignored in serialization. After de-serialization transient fields and non-final static fields will be inited to default values. Final static fields still have values since they are part of the class data.
2.
ObjectOutputStream.writeObject(obj) and ObjectInputStream.readObject() are used in serialization and de-serialization.
3.
During serialization, you need to handle IOException; during de-serialization, you need to handle IOException and ClassNotFoundException. So the de-serializaed class type must be in the classpath.
4.
Uninitialized non-serializable, non-transient instance fields are tolerated. When adding "private Address adr; no error during serialization.

But , private Address adr = new Address(); will cause exception:
Exception in thread "main" java.io.NotSerializableException: com.app.core.Address

5. Serialization and de-serialization can be used for copying and cloning objects. It is slower than regular clone, but can produce a deep copy very easily.

6. If you need to serialize a Serializable class Employee, but one of its super classes is not Serializable, can Employee class still be serialized and de-serialized?

The answer is yes, provided that the non-serializable super-class has a no-arg constructor, which is invoked at de-serialization to initialize that super-class.

What will be the state of data members?
Sub class (serializable) data members will have the restored state & super class(non serializable) data members will have def inited state

7. You must be careful while modifying a class implementing java.io.Serializable. If class does not contain a serialVersionUID field, its serialVersionUID will be automatically generated by the compiler(using serialver tool). Different compilers, or different versions of the same compiler, will generate potentially different values.

Computation of serialVersionUID is based on not only fields, but also on other aspect of the class like implements clause, constructors, etc. So the best practice is to explicitly declare a serialVersionUID field to maintain backward compatibility. If you need to modify the serializable class substantially and expect it to be incompatible with previous versions, then you need to increment serialVersionUID to avoid mixing different versions.

8. Important differences between Serializable and Externalizable

8.1
If you implement Serializable interface , automatically state of the object gets seriliazed. BUT if u implement Externalizable i/f -- you have to explicitly mention which fields you want to serialize.

8.2
Serializable is marker interface without any methods. Externalizable interface contains two methods: writeExternal() and readExternal().

8.3
Default Serialization process will take place for classes implementing Serializable interface. Programmer defined Serialization process for classes implementing Externalizable interface.

8.4
Serializable i/f uses java reflection to re construct object during de-serialization and does not require no-arg constructor. But Externalizable requires public no-arg constructor.

DAY 17 :

Todays's topics
I/O Programming

Revise I/O
Which are the packages for device handling ? java.io / java.nio
java.io : which type of methods ? read / write

java.io API Blocking or non blocking ? BLOCKING (causes the invoker thread : main , to block until read/write operation is over)

In case or read/write errors : java.io.IOException : checked exc

java.io Consists of : I/O streams
What is a stream : data transfer medium (bytes)
Devices : console(stdin / stdout : System.in : InputStream / System.out : PrintStream)
File : File handling streams
Pipe : pipe handling streams
Socket : socket strms .....

What will you use ?
To read data from a data src ----> Java app : input stream
To write data from a Java app ----> data sink : output stream

3 different categories of streams ---
node streams : device handling streams (meant for abstracting actual device handling from the prog)
platform independent) : un buffered strms
eg : FIS, FOS , FR ,FW , PIS,POS,PR,PW...., SIS

buffering streams : in case large data transfer : independent of the device
eg : BIS,BR,BOS,BW,PW

Filtering / conversion streams :
To convert binary data (bytes) ---> prim types / objects
eg : DIS,DOS,OOS,OIS...

Which are 2 types of streams in java.io ?
1. Byte oriented strms
2. char oriented strms

Which are 4 abstract super classes ? : InputStream , OutputStream , Reader , Writer

File Handling Classes ? FIS , FOS , FR , FW

Objectives :
Read data from text file using buffer , till EOF(end of file) n display it : imperative style

Read data from text file using buffer , till EOF(end of file) n display it : declarative (functional) style

Read data from text file using buffer , till EOF(end of file) , filter lines having line length > 50 , upper case contents n then display it.
Hint : BR ---> lines() --> Stream<String> ---> filter : line.length > 50 ---> map : toUpperCase ---> forEach

Objective : Copy files using buffer.
i/p : src file name
dest file name

Objective : Copy only lines from src file with length > 40 : using buffer.
i/p : src file name
dest file name

Objective : Take populated products(list) from sample data.
Accept product category from user.
Filter out the products from the specified category
Sort them as per their date
Save sorted product details in a text file using buffer.

Design
1. core classes : Product ...
2. custom exception
3. validation rules
4. CollectionUtils
4.1 populate data : all products list
4.2 Add a static method to return filtered n sorted products to the caller.
i/p : productList ,category
o/p : stream OR List(collect)

5. IOUtils
Add a static method to save product details to the text file : buffered manner
i/p : stream OR List
o/p : void

6. Tester
6.1 get populated data from collection utils
apply pipeline of ops (call the method!)

store details

Enter Binary Streams for file handling
Which node streams ? FileInputStream , FileOutputStream
Buffering : BIS , BOS

Need of filtering streams

Objective :Write Java App to accept SINGLE product details from User n store it in a binary file
Tester : UI
BinIOUtils : storeDetails(Product p,String fileName) throws ...
{
 //Java App   ----->BOS --->   FOS(String fileName) --->   BIn File
 try(BOS out=new BOS(new FOS(fileName))
{
   //prog has to convert product dtsl --> bin
  //to avoid this : DOS
}
}

101 bread food 2021-05-19 50

Product [id=101, name=bread, productCatgeory=FOOD, manufactureDate=2021-05-19, price=50.0]

Write Java App to retrieve product details from bin file n display it .

Which are conversion strms ?
DataOutputStream n DataInputStream

Objective : Restore product details from a bin file
Before attaching I/O streams , how to validate if the file exists , regular file , read permission
API of java.io.File class
In case of successful validations --attach i/o strms n read data.

Object streams
serialization n de serialization
Objective

Get populated HashMap of Products. Save these product details into bin file
Restore these details from binary file.

I/O handling

Desc of FileInputStream --- java.io.FileInputStream
bin i/p stream connected to file device(bin/char) -- to read data.

Desc of FileOutputStream --- java.io.FileOutputStream
bin o/p stream connected to file device(bin/char) -- to write data.

Desc of FileReader--- java.io.FileReader
char i/p stream connected to file device(char) -- to read data.

Desc of FileWriter--- java.io.FileWriter
char o/p stream connected to file device(char) -- to write data.

Objective --- Read data from text file in buffered manner.

1. java.io.FileReader(String fileName) throws FileNotFoundException
--- Stream class to represent unbuffered char data reading from a text file.
Has methods -- to read data using char/char[]
eg -- public int read() throws IOException

public int read(char[] data) throws IOException
Usage eg-- char[] data=new char[100];
int no= fin.read(data);

public int read(char[] data,int offset,int noOfChars) throws IOException
Usage eg-- char[] data=new char[100];
int no= fin.read(data,10,15);
eg -- 12 chars available
no=12;data[10]----data[21]

1.5 FileReader(File f) throws FileNotFoundException
java.io.File -- class represents path to file or a folder.

2. Improved version -- Buffered data read .
For char oriented streams--- java.io.BufferedReader(Reader r)

API of BR ---

String readLine() --- reads data from a buffer in line by line manner-- & rets null at end of Stream condition.

Objective -- Replace JDK 1.6 try-catch-finally BY JDK 1.7 try-with-resources syntax.
Meaning --- From Java SE 7 onwards --- Introduced java.lang.AutoCloseable -- i/f
It represents --- resources that must be closed -- when no longer required.
i/f method
public void close() throws Exception-- closing resources.
java.io --- classes -- have implemented this i/f -- to auto close resource when no longer required.
syntax of try-with-resources
try (//open one or multiple AutoCloseable resources)
{ .....
} catch(Exception e)
{
}

Objective ---To confirm device independence of Java I/O --- replace File device by Console
i.e --- Read data from console i/p --- in buffered manner till 'stop' & echo back it on the console.

required stream classes --- BR(ISR(System.in))
Alternative is --- use Scanner class.

Adv. of Scanner over above chain ----- contains ready-made parsing methods(eg --- nextInt,nextDouble.....)
But Scanner is not Buffered Stream
Can combine both approaches.(new Scanner(br.readLine())

Objective --- Combine scanner & buffered reader api --- to avail buffering + parsing api. --- BufferedReader provides buffering BUT no simple parsing API. -- supplies br.readLine only

Scanner -- Can be attached to file directly
Constr -- Scanner(File f)
BUT no buffering .

How to use both?
Create BR br=new BR(new FR(...));
while ((s=br.readLine())!=null)
 {
   //scanner can be attached to string ---Scanner(String s)
Scanner sc=new Scanner(s);
// parse data using Scanner API --next,nextInt,nextBoolean
}

Overloaded constructor of FileReader(File f)

java.io.File ---- class represents path to file / folder
Regarding java.io.File -----
Does not follow stream class hierarchy, extends Object directly.
File class --- represents abstract path which can refer to file or folder.
Usage --- 1. To access/check file/folder attributes(exists,file or folder,read/w/exec permisssions,path,parent folder,create new empty file,create tmp files & delete them auto upon termination,mkdir,mkdirs,rename,move,size,last modified ,if folder---list entries from folder,filter entries)
Constructor ---
File (String path) ---
eg --- File f1=new File("abc.dat");
if (f1.exists() && f1.isFile() && f1.canRead())
...attach FileInputStream or FileReader

File (String path) ---
File class API --- boolean exists(),boolean isFile() , boolean canRead()

Objective --- Text File copy operation --- in buffered manner.
For writing data to text file using Buffered streams

java.io.PrintWriter --- char oriented buffered o/p stream --- which can wrap any device.(Binary o/p stream or Char o/p stream)
Constructors---
PrintWriter(Writer w) --- no auto flushing,no conversion, only buffering
PrintWriter(Writer w, boolean flushOnNewLine)--- automatically flush buffer contents on to the writer stream --upon new line

PrintWriter(OutputStream w) --- can wrap binary o/p stream -- buffering +conversion(char-->binary),no auto-flush option
PrintWriter(OutputStream w , boolean flushOnNewLine) ---
API Methods----print/println/printf same as in PrintStream class(same type as System.out)

Stream class which represents --- Char o/p stream connected to Text file. --- java.io.FileWriter
Constructor
FileWriter(String fileName) throws IOException -- new file will be created & data will be written in char format.
FileWriter(String fileName,boolean append) --- if append is true , data will be appended to existing text file.

Collection & I/O
Objective ---
Items Inventory
Item -- code(String-Prim key),desc,category,quantity,price,shipmentDate
constr,toString.

Create suitable collection of Items(HashMap) --- sort map as per desc item code ,& store sorted item dtls in 1 text file .
NOTE : individual item rec MUST be written on separate line.

Sort items as shipment Date & store sorted dtls in another file . Before exiting ensure closing of data strms .
(buffered manner)

Objective -- Restore collection of items created in above requirement ---in form of HashMap .
-- buffering is optional.

Objective --- using Binary file streams.
Classes --- FileInputStream -- unbuffered bin i/p stream connected to bin file device.
FileOutputStream --unbuffered bin o/p stream connected to bin file device.

But these classes --- dont provide buffering & have only read() write() methods in units of bytes/byte[]

API of InputStream class
1. int read() throws IOException

Will try to read 1 byte from the stream.
Data un-available method blocks.
Returns byte--->int to caller.
eg -- int data=System.in.read();
2. int read(byte[] bytes) throws IOException
Will try to read data from underlying stream.
Data un-available -- method blocks.
Rets actual no of bytes read.

eg :
byte[] bytes=new byte[100];
int no=System.in.read(bytes);
no data --method blocks.
10 bytes available -no =10;bytes[0]-----bytes[9]
110 bytes available -- no=100;bytes[0]....bytes[99]

3. int read(byte[] bytes,int offset,int maxNoOfBytes) throws IOException
Will try to read data from underlying stream.
Data un-available -- method blocks.
Rets actual no of bytes read.
eg :
byte[] bytes=new byte[100];
int no=System.in.read(bytes,10,15);
no data --BLOCKS
5 bytes available --no=5;bytes[10].....bytes[14]
110 bytes available -- no=15;bytes[10]..bytes[24]

4. int available() throws IOException
Returns no of available bytes in the stream
no data ---DOESN't block -- rets 0.

Important API of OutputStream
1. public void write(int byte) throws IOException
2. public void write(byte[] bytes) throws IOException
3. public void write(byte[] bytes,int offset,int maxNo) throws IOException
bytes[offset].....bytes[offset+maxNo-1] -- written out to stream
4. void flush() throws IOException
5. void close() throws IOException

Using BIS(BufferedInputStream) -- enables buffering BUT doesn't provide any advanced API(ie. read(), read(byte[]), read(byte[] b,int off,int len) . Same is true with BOS.(BufferedOutputStream)

Objective ---
Create Customer/Account based collection. Sort if reqd.
Store Sorted collection to bin file in buffered manner --
& re-store the same.

Use advanced streams in such cases ---
Mixed Data streams
java.io.DataOutputStream ---implements DataOutput i/f
(converter stream) prim types / string ---> binary
Constructor -- DataOutputStream (OutputStream out)
API ---
public void writeInt(int i) throws IOExc
public void writeChar(char i) throws IOExc
public void writeFloat,writeDouble.....
For Strings
public void writeUTF(String s) throws IOExc ---uses Modified UTF 8 convention
or
public void writeChars(String s) throws IOExc --- uses UTF16 convention

eg : Items Inventory
Item -- code(String-Prim key),desc,category,quantity,price,shipmentDate
constr,toString.

Objective ---
Customer data is already stored in bin file.
Read customer data from Bin file --- in buffered manner & upload the same in HM .display customer details.

Stream class --- java.io.DataInputStream -- implements DataInput
Conversion stream(converts from bin ---> prim type or String)

Constructor
DataInputStream(InputStream in)

API Methods
public int readInt() throws IOException
public double readDouble() throws IOException
public char readChar() throws IOException
public String readUTF() throws IOException(must be used with writeUTF)
public String readChars() throws IOException(must be used with writeChars)

Most Advanced streams ---
Binary streams which can read/write data from/to binary stream in units of Object/Collection of Object refs (i.e Data Transfer Unit = Object/Collection of Objects)

Stream Class for writing Objects to bin. stream
java.io.ObjectOutputStream implements DataOutput,ObjectOutput

Description --- ObjectOutputStream class performs serialization.
serialization= extracting state of object & converting it in binary form.

state of object = non-static & non-transient data members
Constructor
ObjectOutputStream(OutputStream out)
out--- dest Binary o/p stream --- where serialized data stream has to be sent.

API methods ---
public void writeInt(int i) throws IOExc
public void writeChar(char i) throws IOExc
public void writeFloat,writeDouble.....
For Strings
public void writeUTF(String s) throws IOExc ---uses Modified UTF 8 convention

public void writeObject(Object o) throws IOException,NotSerializableException

De-serialization---- conversion or re-construction of Java objs from bin stream.
java.io.ObjectInputStream --- performs de-serialization.--- implements DataInput,ObjectInput
Constructor --- ObjectInputStream(InputStream in)
API methods ---
readInt,readShort,readUTF,readChars..... +
public Object readObject() throws IOException

Objective --- attach OIS to the bin file using FIS & display customer data.

Objective :
Confirming concepts of serialization & de-serialization
Emp -- int id, String name,double salary,Address adr;
Address -- String state,city,street.

Objective -- Understanding Set & its implementation classes
HashSet -- based upon hashing algorithm

More involved scenario.
(store customer info & Items to be purchased)
Data members - int no,Customer info, AL<Item>, Date creationDate

---

DAY 18 :

Today's Topics
1. Continue with Serialization n de-serialiazation
1.1 Serial version unique id
1.2 transient keyword

2. Method overriding n exception handling
3. Multi threading

Revise
What is Serialization ? : Conversion of state of the object(=non static n non transient data memebrs) ---> binary stream

What is converted into binary stream ? : state of the object + metadata info (Magic no , serial format no,F.Q cls name , class desc...)

Serializer Class : java.io.ObjectOutoutStream
I/fs implemented : DataOutput , ObjectOutput

Constructor :ObjectOutputStream(OutputStream out)
out : dest bin o/p stream

Methods : writeInt,writeBoolean....,writeUTF,+ public void writeObject(Object o)

chain of I/O streams for ser. strm , to be stored in bin file
Java App --->OOS --->BOS --> FOS --->Bin File

chain of I/O streams for ser. strm , to be uploaded from clnt --> server
In clnt side app :
Java App --->OOS --->BOS --> SOS (Socket : getOutputStream()) --->Socket : end point of communication

eg : List<Product> products=new ArrayList<>();
//added 10 products in the list
//Java App --->OOS --->BOS --> FOS --->Bin File
try(OOS out=new OOS(new BOS(new FOS(fileName)))) {
out.writeObject(products);
}

What will happen if Product is NOT Serializable ? : JVM throws NotSerializableException : aborts the ser.

At the time of serialization(@ run time) , JVM will check serializability of the ENTIRE object graph --

How ?JVM checks if the class of that object has imple. Serializable i/f (run time marker)
YES --continues with the conversion
If JVM comes across any object --which has not imple Serializable i/f : aborts ser. --throws java.io.NotSerializableExc.

What is de-serialization ? : re construction of the entire object graph from binary stream(FIS,PIS,SIS...)

What are the pre requisites of de-serialization ?
1. Supply(share) .class files of User defined types(eg : Customer, Order,Custom Exc,Category...) : in the run time class path => behaviour
If not found : java.lang.ClassNotFoundException

2.For JVM to create instance/s in the heap : provide .ser file (=metadata + state)

Which is best way of sharing dependencies ? Using JAR files
HOW ? : cmd line tool : jar / IDE

Class used for de-ser : java.io.ObjectInputStream
Constructor : ObjectInputStream(InputStream in)
I/Fs : DataInput , ObjectInput

Methods : readInt,readFloat.....readUTF + public Object readObject() throws
ClassNotFoundException,
InvalidClassException , IOException

chain of I/O streams
Java App <---- OIS <---- FIS <----BIn File(.ser)

Objectives
1. JAR file creation n adding this dependency in de-serial project
2. transient keyword
Suppose application should not store product's manufacturing date , BUT wants to save all other details . How will you manage it ?
3. What's the observation n conclusion ?
Serial version unique id

Objective :
Create Java app for
1. Restore products(map) from bin file : de -serial
for 1st time : no bin file : rets populated map with sample data, o.w ret restored map from bin file
2. Options
2.1 Update product price
2.2 Display sorted products as per date.
2.3 Remove product
2.4 Exit : store products : serial

4. Method overriding n exception handling

Overriding form of the method(in sub class) can't throw any NEW or BROADER CHECKED exceptions.

Confirm with examples.
eg :
What will happen ?
1. package p1;
class A
{

```
  void show()
  {
    sop("1");
  }
}

class B extends A
{
  @Override
  void show() //throws InterruptedException //javac err : NEW checked exc.
  {
    sop("2");
  }
}

class C extends A
{
  @Override
  void show() throws NullPointerException //no javac err!!!!!!
  {
    sop("3");
  }
}
```

In Tester

```
A ref=new B();
ref.show();
```

2.

```
package p1;
class A
{
  void show() throws IOException
  {
    sop("1");
  }
}

class B extends A
{
  @Override
  void show() //no err
  {
```

```
   sop("2");
  }
}
```

3.
```
package p1;
class A
{
  void show() throws IOException
  {
    sop("1");
  }
}

class B extends A
{
  @Override
  void show() throws FileNotFoundException// (NARROWER) no javac err
  {
    sop("2");
  }
}
```
4.
```
package p1;
class A
{
  void show() throws IOException
  {
    sop("1");
  }
}

class B extends A
{
  @Override
  void show() throws Exception  //javac err : BROADER CHECKED EXC
  {
    sop("2");
  }
}
```
---

5. Enter multi threading

What is multi tasking n why ?
Process vs Threads

Refer : Thread state transition diagram
Refer : Threads API

Objectives
Creates multiple threads n test concurrency
1. extends Thread

2. implements Runnable

3. How to ensure no orphan threads ?
(join)

----------Pending----------
1. sleep n interrupt (with separate class , later use ano inner class to create a thread/later replace it by lambda expression)

2. Consider threads : t1 & t2
t1's run() : t2.join()
t2's run() : Loop (indefinitely running) with intermittent sleep
main waiting for t1 n t2 to finish exec.
What will happen ? :
Fix it!

3. Consider threads : t1 & t2
t1's run() : t2.join()
t2's run() : Blocked on I/O
main waiting for t1 n t2 to finish exec.
What will happen ? :

4. Solve practical requirement based on (Collection + I/O + Threads)

Solve : Create a multi threaded application for saving student details in 2 text files(taken from the Map of students) : (in student_gpa.txt : sorted by gpa & in student_dob.txt : sorted by dob) using 2 different threads , concurrently!
(refer to a diag : "assgn-help")

1. CollectionUtils : sorting methods
2. IOUtils : writeData : text data + buffering
Chain of I/O streams
Java app ------> PW-------->FW ----->Text File(device)

3. Create child thrds (implements)
4. main

Thread related API

Starting point
1. java.lang.Runnable --functional i/f
SAM (single abstract method) -- public void run()
Prog MUST override run() -- to supply thread exec. logic.

2. java.lang.Thread --class -- imple . Runnable
It has imple. run() -- blank manner.

3. Constrs of Thread class in "extends" scenario
3.1 Thread() -- Creates a new un-named thrd.
3.2 Thread(String name) -- Creates a new named thrd.

4. Constrs of Thread class in "implements" scneario
4.1 Thread(Runnable instance) -- Creates a new un-named thrd.
4.2 Thread(Runnable instance,String name) -- Creates a new named thrd.

Methods of Thread class
1. public void start() -- To cause transition from NEW -- RUNNABLE
throws IllegalThreadStateException -- if thrd is alrdy runnable or dead.
2. public static void yield() -- Requests the underlying native scheduler to release CPU & enters rdy pool.
Use case -- co operative multi tasking(to allow lesser prio thrds to access processor)

3. public void setName(String nm)
4. public String getName()
5. Priority scale -- 1---10
Thread class consts --MIN_PRIO=1 , MAX_PRIO=10 , NORM_PRIO =5
public void setPriority(int prio)
6. public static Thread currentThread() -- rets invoker(current) thrd ref.
7. public String toString() -- Overrides Object class method , to ret
Thread name,priority,name of thrd grp.
8.public static void sleep(long ms) throws InterruptedException
Blocks invoker thread till specified msecs.
9. public void join() throws InterruptedException
Blocking method(API)
--Causes the invoker thread to block till specified thread gets over.
eg : t1 & t2
t1's run()
{

.....
t2.join();//who is waiting for whom for which purpose ? : t1 is waiting for t2 : to complete exec
....
}

t2's run()
{
  //some B.L :read data from file
}
join method can be used effectively to avoid orphan threads
main has to wait for child thrds to complete exec
How ?
In main(..)
t1.join();
t2.join();

10 public void join(long ms) throws InterruptedException
eg : In main method
t1.join(10000);//main is waiting for t1 to finish exec: upto max 10 sec
//t1 gets over after 2 secs : main un blocks
//If t1 doesn't get over within 10 secs : main will be blocked for 10 sec n auto un block.

--Causes the invoker thread to block till specified thread gets over OR tmout elapsed
11. public void interrupt() -- interrupts(un blocks) the threads blocked on ---sleep/join/wait

Methods of Object class (Use Case : Inter thread communication)

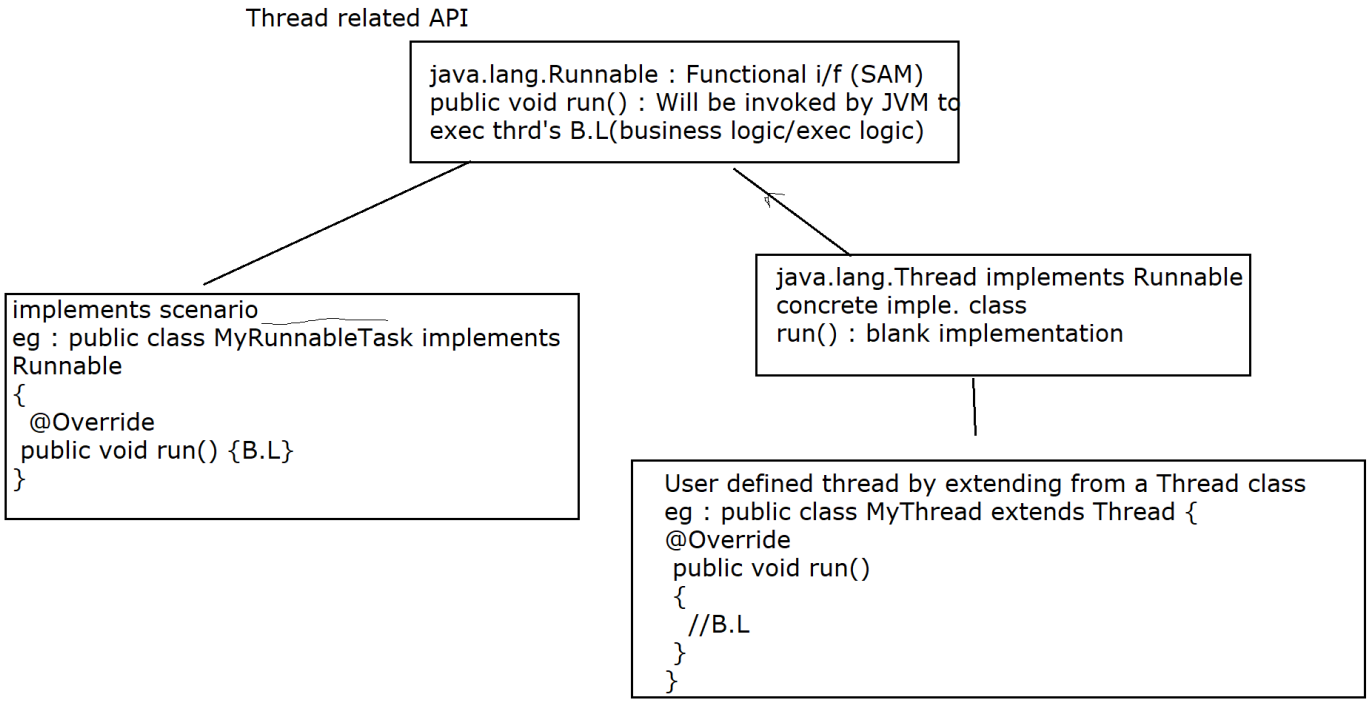
1. public final void wait() throws InterruptedException,IllegalMonitorStateException
Meaning -- Forces the invoker thread to release processor & monitor & wait outside .

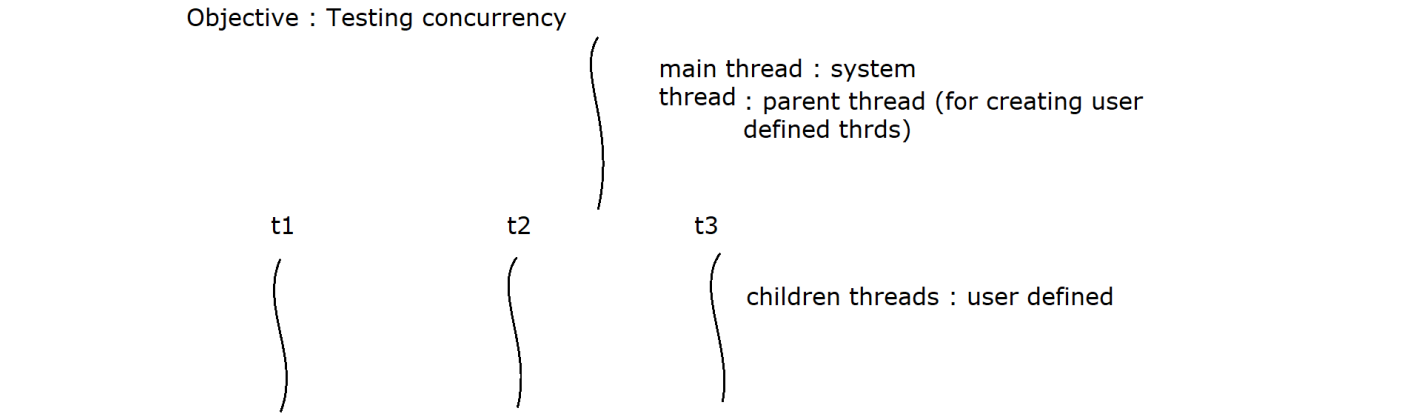
Trigger for InterruptedException
Some other thread sends interrupt signal to the waiting thread.
Trigger for IllegalMonitorStateException
If the invoker thread is not an owner of the monitor
(i.e if its invoking neither a synched method nor a block)

2. public final void notify() throws IllegalMonitorStateException
Meaning -- Un blocks (wakes up) exactly 1 thread , which has invoked wait on the same object's monitor.

May raise IllegalMonitorStateException --if the current thread is not the owner of a lock.

3. public final void notifyAll() throws IllegalMonitorStateException
Un blocks ALL waiting threads , on the same object's monitor.





Method overriding n exception handling
Overriding form of the method(in sub class) can't add any NEW or BROADER checked excpetions
Confirm with examples.
eg :

```
1.  package p1;
class A
{
  void show()
  {
    sop("1");
  }
}

class B extends A
{
  @Override
  void show() throws InterruptedException //javac error : can't add any NEW checked excpetions
  {
    sop("2");
  }
}
```

In Tester

```
A ref=new B();
ref.show();
```

```
2.  package p1;
class A
{
  void show() throws IOException
  {
    sop("1");
  }
}

class B extends A
{
  @Override
  void show() //no javac error
```

```
 {
   sop("2");
 }
}
```

3.
```
package p1;
class A
{
 void show() throws IOException
 {
   sop("1");
 }
}

class B extends A
{
 @Override
 void show() throws FileNotFoundException// no javac error : FileNotFoundException IS A IOException
 {
   sop("2");
 }
}
```

4.
```
package p1;
class A
{
 void show() throws IOException
 {
   sop("1");
 }
}

class B extends A
{
 @Override
 void show() throws Exception //javac err : Exception is super cls : can't add any BROADER chked excs.
 {
   sop("2");
 }
}
```

What happens when you call start on NEW Thread ?

It internally invokes a native method(not written in java) start0()

Its invocation will --

1. cause a new native thread-of-execution to be created (by native OS)
2. cause the run method to be invoked on that thread.

Thread related API

Starting point
1. java.lang.Runnable --functional i/f
SAM (single abstract method) -- public void run()
Prog MUST override run() -- to supply thread exec. logic.

2. java.lang.Thread --class -- imple . Runnable
It has imple. run() -- blank manner.

3. Constrs of Thread class in "extends" scenario
3.1 Thread() -- Creates a new un-named thrd.
3.2 Thread(String name) -- Creates a new named thrd.

4. Constrs of Thread class in "implements" scneario
4.1 Thread(Runnable instance) -- Creates a new un-named thrd.
4.2 Thread(Runnable instance,String name) -- Creates a new named thrd.

Methods of Thread class
1. public void start() -- To cause transition from NEW --> RUNNABLE
throws IllegalThreadStateException -- if thrd is alrdy runnable or dead.
2. public static void yield() -- Requests the underlying native scheduler to release CPU & enters rdy pool.
Use case -- co operative multi tasking(to allow lesser prio thrds to access processor)

3. public void setName(String nm)
4. public String getName()
5. Priority scale -- 1---10
Thread class consts --MIN_PRIO=1 , MAX_PRIO=10 , NORM_PRIO =5
public void setPriority(int prio)
6. public static Thread currentThread() -- rets invoker(current) thrd ref.
7. public String toString() -- Overrides Object class method , to ret
Thread name,priority,name of thrd grp.

8.public static void sleep(long ms) throws InterruptedException
Blocks invoker thread till specified msecs.
9. public void join() throws InterruptedException
Blocking method(API)
--Causes the invoker thread to block till specified thread gets over.
eg : t1 & t2
t1's run()
{
 .....
 t2.join();//who is waiting for whom for which purpose ? : t1 is waiting for t2 : to complete exec
....
}

t2's run()
{
  //some B.L :read data from file
}
join method can be used effectively to avoid orphan threads
main has to wait for child thrds to complete exec
How ?
In main(..)
t1.join();
t2.join();

10 public void join(long ms) throws InterruptedException
eg : In main method
t1.join(10000);//main is waiting for t1 to finish exec: upto max 10 sec
//t1 gets over after 2 secs : main un blocks
//If t1 doesn't get over within 10 secs : main will be blocked for 10 sec n auto un block.

--Causes the invoker thread to block till specified thread gets over OR tmout elapsed
11. public void interrupt() -- interrupts(un blocks) the threads blocked on ---sleep/join/wait

Methods of Object class (Use Case : Inter thread communication)

1. public final void wait() throws InterruptedException,IllegalMonitorStateException
Meaning -- Forces the invoker thread to release processor & monitor & wait outside .

Trigger for InterruptedException
Some other thread sends interrupt signal to the waiting thread.
Trigger for IllegalMonitorStateException

If the invoker thread is not an owner of the monitor
(i.e if its invoking neither a synched method nor a block)

2. public final void notify() throws IllegalMonitorStateException
Meaning -- Un blocks (wakes up) exactly 1 thread , which has invoked wait on the same object's monitor.

May raise IllegalMonitorStateException --if the current thread is not the owner of a lock.

3. public final void notifyAll() throws IllegalMonitorStateException
Un blocks ALL waiting threads , on the same object's monitor.

Thread state transitions

Thread related API
1. public interface java.lang.Runnable
public void run()
Must be implemented to supply B.L
2. public class java.lang.Thread imple. Runnable
It has imple run() -- blank manner.
Thread class API
Constructors of Thread class to be used in "extends Thread " scenario
1. Thread() -- creates un named thread, in default thread grp.
2. Thread(String name) -- creates a named thread, in default thread grp.
eg : public class MyThread extends Thread
{
public MyThread(String nm)
{
super(nm);//creates a named thrd
....
}
}

Constructors of Thread in implements scenario
1. Thread(Runnable instance) --- Creates un named thread.
instance -- instance of the class which imple Runnable

2. Thread(Runnable instance,String name) --
Creates named thrd using Runnable's imple class instance.
eg : public class MyRunnableTask imple Runnable
{

```
  @Override
  public void run()
  {....}
}
```
In Tester
Thread t1=new Thread(new MyRunnableTask(...),"one");
Thread class Methods
1. public void start() --- Transitions the thread from new ---->Rdy-to-run.
2. public static void yield() -- Requests the scheduler to release processor(so that lower priority threads can access processor). Doesn't release the lock.
3.public static void sleep(long msec) throws InterruptedException
4. public static Thread currentThread() -- rets invoker thread ref.
5. public String toString() --Rets name of thrd,prio , name of thrd grp
6. public String getName() , public void setName(String nm)
7. Priority scale -- 1---10
Thread class constants -- MIN_PRIORITY(1),MAX_PRIORITY(10),NORM_PRIORITY(5)
8. public int getPriority() , public void setPriority(int prio)
9. public boolean isAlive()
10.public Thread.State getState()
Thread.State --static nested enum
constants -- NEW,RUNNABLE,BLOCKED,WAITING,TIMED_WAITING,TERMINATED
11. public void join() throws InterruptedException
Causes the invoker thread to block , till specified thrd becomes dead.
eg : In t1's run() --if u invoke t2.join()
t1 gets blocked till t2 finishes its exec.

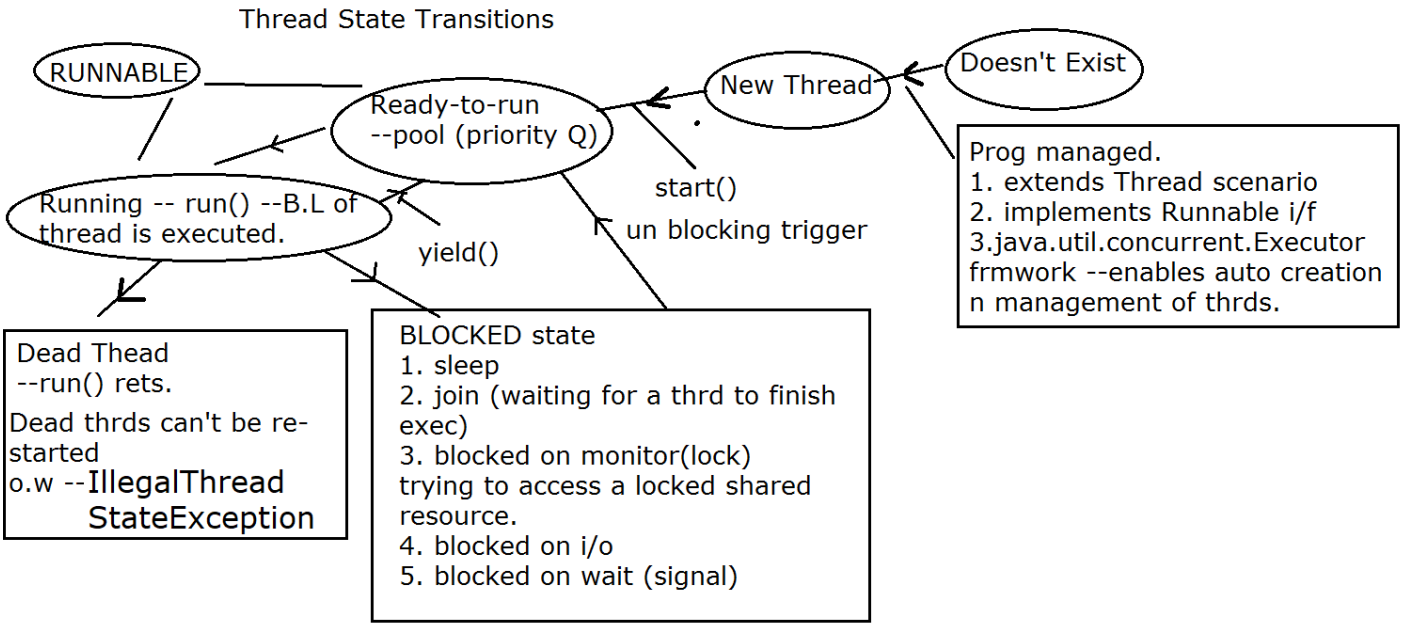
How to ensure "NO Orphans" condition , w/o relying on sleep ?
eg : In main(...)
```
t1.join();
t2.join();
t3.join();
sop("main over");
```

Objective --Test the concurrency
Create a class extending from thrd.
run() -- add a delay loop.
Tester --main(..) --create multiple thrds & check.

Regarding method overriding & exc handling
Overriding form of the method can't throw NEW or BROADER checked exception.

eg :

```
 class A
{
   void show() {...} throws FileNotFoundException
}
 class B extends A
{
   @Override
  public void show() throws Exception
 {...}
}
```



Refer to Thread state transitions figure

A -- transition from rdy to run -----> Running
Triggered by --- in time slice based scheduling --- time slot of earlier thrd over OR in pre-emptive multitasking -- higher prio thrd pre-empts lower prio thrd.
B --- transition from running ---> ready to run
Reverse of earlier transition OR
public static void yield()----

Requests underlying scheduler to swap out current thrd SO THAT some other lower prio thrd MAY get a chance to run. (to avoid thrd starvation -- i.e co-operative multi-tasking.)

C --running state --- Only in this state --- run() method gets executed.
running --->dead --- Triggers -- run() method returns in healthy manner . OR run() aborts due to un-handled , un-checked excs.

D -- blocked ---> rdy to run --- when any of blocking condition is removed --- blocked thrd enters rdy pool & resumes competition among other thrds.

API Involved
1. Thread class constructor to be used in extends Thread scenario
1.1 Thread() --- A new thrd is created BUT with JVM supplied name.
1.2 Thread(String nm) -- creates named thread.

2. Thread class constructor to be used in implments scenario
2.1 Thread (Runnable target/inst) --- Creates a new thrd --- by passing instance of the class which implements Runnable i/f.
Run time significance -- Whenever this thrd gets a chance to run --- underlying task scheduler -- will invoke(via JVM) this class's run() method.
2.2 Thread (Runnable inst,String name)

public class Myclass extends Thread --- start()

Vs

public class Myclass implements Runnable --- This class simply represents a runnable task.
Prog MUST create a thrd class inst BY attaching Runnable task to it.

Thread class API
1.public String getName()
2. public void setName(String nm)
3. public static Thread currentThread() -- rets ref of the invoker thrd.
4. public int getPriority() -- rets current prio.
Prio scale -- 1---10(MIN_PRIORITY,MAX_PRIORITY)
NORM_PRIORITY ---- 5
4.5 public void setPriority(int prio) --- must be invoked before start()

DO NOT rely on priority factor -- since it is ultimately specific to underlying OS prio range.-- may cause loss of portability.
t1 --- max-prio ---run() --- begin up-counter -- obs val after 10 secs

t1 --- min-prio ---run() --- begin up-counter -- obs val after 10 secs
t1 --- max-prio & t2 -- min prio
5. public String toString() --- to ret -- name,prio & thrd grp name
6. public static void sleep(long ms) throws InterruptedException

Objective -1. to test concurrency of thrd : in extends thrd scenario.
Create a Thrd class, add simple loop with dly in run method to display the exec. sequence.
Write main method : which will instantiate thrds(multi-thrded system) & test the concurrent exec. of main thrd along with other thrds.

To ensure that main thrd terminates last : no orphan thrds in the system.

Thread class Method :
1. public void join() throws InterruptedExc.
The invoker thrd gets blocked until the specified thrd joins it(i.e specified thrd becomes dead)
eg : 2 thrds t1 & t2 are running concurrently.
If u invoke : in run() method of t1 :
t2.join() ----> t1 gets blocked until t2 is over.
t3 -- t1.interrupt()
t1 : Blocked on join
Unblcking triggers -- t2 over,getting interrupt signal.

2. public void join(long ms) throws InterruptedExc
waits max for specified tmout .
Unblocking triggers -- t2 over,getting interrupt signal, tmout exceeded.

Objective -- Ensure no orphans , w/o touching child thread class.

Objective
Replace for loop from thrds, by indefinite loop (while true) & still ensure -- no orphans.
Start --- main + 3 child thrds. --- main(parent thrd waits patiently for 5 secs. & then somehow forces termination of child thrds & then terminates last.

API ---
public void interrupt() -- sends interrupt signal to the specified thread. If specified thrd has invoked any method (sleep,join,wait)-- having throws clause of InterruptedExc --- then only thrd gets UNBLOCKED due to InterruptedExc.

NOTE -- Thread which is blocked on I/O -- CANT be un-blocked by interrupt signal.

Threads blocked by invoking any method -- having InterruptedException (sleep,join,wait) can be unblocked by interrupt signal.

2. Objective - to test concurrency of thrds : in implements Runnable scenario.

2.5 implements scenarion

3.

Objective : create 3 thrds with 3 random sleep durations(range is 500ms-5sec)
& start them conc. & ensure that main terminates last.
For random nos--- java.util.Random() , nextInt,nextInt(int upperLim)

Objective : create 3 thrds & start them conc. & ensure that main terminates last.
2nd thrd should accept data from console, dont supply data & observe .
How to unblock a thread , which is blocked on I/O?

Objective : Apply multi threading to Swing application.
Create Swing application -- with start & stop buttons, in south region.
Create JPanel in center region , with some default color.
When start button is clicked, center panel should start changing color(random color) periodically.
When stop button is clicked, stop changing color.

Objective -- To store emp details , dept wise in SAME data file.(text buffered manner--- PrintWriter)

Design :
0. Emp class --- id,deptid,name,sal

1. Write Utils class
d.m --- pw

constr --pw inst ---
void writeData(Emp e) {...}

void cleanUp() {...}

constr ---create PW inst --
PW pw=new PW(new FW("emps.data"),true);
---add writeData(....) : instance method to write Emp dtls (first name, last name ,deptId of the emp)
to the file with small dly in between.(why dly ? --- for simulating practical scenario & also to add randomness to code)
do u need clean up method- -- yes -- close pw.

2. Write Dept Handler runnable task class --- implements
Override run() method which will invoke writeData method till 'exit' condition is encountered.
Add stop/exit method to enable 'exit' flag.
....
{
Emp e;
Utils u;
constr(u,e)
{this.e=e;
//u=new Utils(); ------

public void run()
{
  while(!exit)
    u.writeData(e);
}
3. Write Tester class : accept some emp dtls . Create depthandler task per dept ,attach thrds & start them.
Wait for key stroke ---
System.in.read();
stop all child thrds
ensure no orphans
clean up -- pw

Wait for the key stroke : upon key stroke --- stop all child thrds & then finally terminate main.

Observed o/p ------garbled display or garbled data written in file.

Reason : multiple thrds trying to access the shared resource concurrently.
eg of shared resource : Console or file device,socket,DB table

Solution : Any time when asynch thrds need to access the SHARED resource : LOCK the shared resource --- so that after locking -- only single thread will be able to access the resource concurrently

When is synchronization(=applying thread safety=locking shared resource) required?
In multi-threaded java applns -- iff multiple thrds trying to access SAME copy of the shared resource(eg -- reservation tkt,db table,file or socket or any shared device) & some of the threads are reading n others updating the resource

How to lock the resource?
Using synchronized methods or synchronized blocks.
In either approach : the java code is executed from within the monitor & thus protects the concurrent access.

Note : sleeping thrd sleeps inside the monitor(i.e Thread invoking sleep(...) , DOESn't release the ownership of the monitor)

eg classes :
StringBuilder : thrd-unsafe.--- unsynchronized --- if multiple thrds try to access the same copy of the SB, SB may fail(wrong data)
StringBuffer --- thrd -safe ----synchronized internally--- if multiple thrds try to access the same copy of the SB, only 1 thrd can access the SB at any parti. instance.

which is reco class in single threaded appln? --- StringBuilder
multiple thrds -- having individual copies -- StingBuilder
multiple thrds -- sharing same copy -- StringBuilder --
identify code to be guarded -- sb 's api -- invoke thrd unsafe API -- from inside synched block.

ArrayList(inherently thrd un-safe) Vs Vector(inherently thrd safe)

HashMap(un-safe) Vs Hashtable(thrd safe)

synchronized block syntax --- to apply synchro. externally.
synchronized (Object to be locked--- shared resource)
{
//code to be synchronized --methods of shared res. -- thrd safe manner(from within monitor)

}

1. If any thrd is accessing any synched method of 1 obj, then same thrd or any other thrd CANT concurrently access same method of the same obj.(Tester1.java)

2. If any thrd is accessing any synched method of 1 obj, then same thrd or any other thrd CANT concurrently access same method or any other synchronized method of the same obj.(Tester2.java)

2. If thrds have their own independent copies of resources, synch IS NOT required.(Tester2.java)
3.If u are using any thrd un-safe code(ie. ready code without source) --& want to apply thrd safety externally --- then just wrap the code within synched block to use locking.(Tester3.java)

Objective : Create Producer & Consumer thrds .
Producer produces data samples & consumer reads the same.
For simplicity : let the data be represented by : single Emp record
Producer produces emp rec sequentially & consumer reads the same.

Rules : 1 when producer is producing data , consumer thrd concurrently should not be allowed to read data & vice versa.

Any more rules?????????????
Yes --- correct sequencing is also necessary in such cases.
Rule 2 : Producer must 1st produce data sample ---consumer reads data sample & then producer can produce next data sample. Similarly consumer should not be able to read stale(same) data samples .

ITC --- API level
Object class API
1. public void wait() throws IE ---thrd MUST be owner of the monitor(i.e invoke wait/notify/notifyAll from within synched block or method) --- othewise MAY get IllegalMontitorStateExc

---causes blocking of the thrd outside montitor.

UnBlocking triggers --- interrupt(not reco --- since it may cause death of thrd) , notify/notifyAll --- reco.

2. 1. public void wait(long ms) throws IE
UnBlocking triggers --- interrupt(not reco --- since it may cause death of thrd) , notify/notifyAll --- reco.,tmout exceeded

2.2 public void notify() -- MUST be invoked from within monitor , ow may get IllegalMonitorStateExc

Un-blocks ANY waiting thread , blocked on SAME MONITOR

2.3 public void notifyAll() -- Un-blocks ALL waiting threads , blocked on SAME MONITOR

notify/notifyAll--- DOESN't BLOCK the thread & Doesn't release lock on monitor. --- send wake up signal -- to thrd/s waiting on same monitor.
wait --- Blocks the thread --- Releases lock on the monitor.

volatile --- java keyword, applicable at data member.
typically used in multi-threaded scenario only when multiple thrds are accessing the same data member.
Use --- to specify-- that data var. is being used by multiple thrds concurrently -- so dont apply any optimizations(OR the value of the variable can get modified outside the current thrd) .
With volatile keyword -- its guaranteed to give most recent value.
The volatile modifier tells the JVM that a thread accessing the variable must always get its own private copy of the variable with the main copy in memory

---

Multithreading

Objective
In this session, you will be able to:
• Define a thread

• Create threads in java using two approaches
• Describe the life cycle of a thread
• Synchronize threads

Thread
• A thread is a single sequential flow of control within a program.
• lightweight process
• execution context
Threads in Java
• There are two techniques for creating a thread:
- Subclassing *Thread* and Overriding run
- Implementing the *Runnable* Interface

The *Runnable* Interface
• The *Runnable* interface should be implemented by any class whose instances are intended to be executed by a thread.
• The class must define a method of no arguments called run.
• public void **run()**
- When an object implementing interface *Runnable* is used to create a thread, starting the thread causes the object's run method to be called in that separately executing thread.

The Thread Class
• **Thread()**
- Allocates a new Thread object.
• **Thread(Runnable target)**
- Allocates a new Thread object.
• **Thread(Runnable target,String name)**
- Allocates a new Thread object.
• **Thread(String name)**
- Allocates a new Thread object.

Subclassing Thread
```
class SimpleThread extends Thread {
public SimpleThread(String str) {
super(str);
}
public void run() {
for (int i = 0; i < 10; i++) {
System.out.println(i + " " + getName());
try {
sleep((long)(Math.random() * 1000));
} catch (InterruptedException e) {}
}
System.out.println("DONE! " + getName());
```

```
}
}
```
Implementing the Runnable Interface
```
class SimpleThread implements Runnable {
public void run() {
Thread t = Thread.currentThread();
for (int i = 0; i < 10; i++) {
System.out.println(i + " " + t.getName() );
try {
Thread.sleep((long)(Math.random() * 1000));
} catch (InterruptedException e) {}
}
System.out.println("DONE! " + t.getName());
}
}
```
Implementing the Runnable Interface
```
import java.util.*;
import java.text.DateFormat;
import java.applet.Applet;
public class Clock extends Applet implements Runnable {
private Thread clockThread = null;
public void start() {
if (clockThread == null) {
clockThread = new Thread(this, "Clock");
clockThread.start();
}
}//start
public void run() {
Thread myThread = Thread.currentThread();
while (clockThread == myThread) {
repaint();
try {
Thread.sleep(1000);
} catch (InterruptedException e){}
}//while
}//run
```
Implementing the Runnable Interface
```
public void paint(Graphics g) {
// get the time and convert it to a date
Calendar cal = Calendar.getInstance();
Date date = cal.getTime();
// format it and display it
DateFormat dateFormatter = DateFormat.getTimeInstance();
```

```
g.drawString(dateFormatter.format(date), 50, 50);
}//paint
// overrides Applet's stop method
public void stop() {
clockThread = null;
}//stop
}// Clock
```

The Life Cycle of a Thread

- Creating a Thread
- Starting a Thread
- Making a Thread Not Runnable
- Stopping a Thread

The Life Cycle of a Thread

- Creating a Thread:

```
public void start() {
if (clockThread == null) {
clockThread = new Thread(this, "Clock");
clockThread.start();
}
}
```

- After the statement containing new is executed, clockThread is in the *New Thread* state; no system resources are allocated for it yet.
- When a thread is in this state, we can only call the start method on it.

The Life Cycle of a Thread

- Starting a Thread :

```
public void start() {
if (clockThread == null) {
clockThread = new Thread(this, "Clock");
clockThread.start();}}
```

- The start method creates the system resources necessary to run the thread, schedules the thread to run, and calls the thread's run method.
- After the start method returns, the thread is said to be "running". In reality a thread that has been started is actually in the Runnable state.
- At any given time, a "running" thread actually may be waiting for its turn in the CPU.

The Life Cycle of a Thread

- The run( ) method :

```
public void run() {
Thread myThread = Thread.currentThread();
while (clockThread == myThread) {
repaint();
try {
Thread.sleep(1000);
```

```
} catch (InterruptedException e){}
}}
```

• Clock's run method loops while the condition clockThread = = myThread is true.

• Within the loop, the applet repaints (which further calls paint) itself and then tells the thread to sleep for one second.

The Life Cycle of a Thread

• Making a Thread Not Runnable:

• A thread becomes Not Runnable when one of these events occurs: ▪ Its **sleep( )** method is invoked. ▪ The thread calls the **wait( )** method to wait for a specific condition to be satisifed. ▪ The thread is blocking on I/O.

The Life Cycle of a Thread

• The clockThread in the Clock applet becomes *Not Runnable* when the run method calls *sleep* on the current thread:

```
public void run() {
Thread myThread = Thread.currentThread();
while (clockThread == myThread) {
repaint();
try {
Thread.sleep(1000);
} catch (InterruptedException e){}
}
}
```

The Life Cycle of a Thread

• Stopping a Thread :

• A program doesn't stop a thread, rather, a thread arranges for its own death by having a run method that terminates naturally. For example, the while loop in this run method is a finite loop-- it will iterate 100 times and then exit:

```
public void run() {
int i = 0;
while (i < 100) {
i++;
System.out.println("i = " + i);
} }
```

• A thread with this run method dies naturally when the loop completes and the run method exits.

The Life Cycle of a Thread

• Stopping a Thread : ▪ Stopping the Clock applet thread

```
public void run() {
```

```
Thread myThread = Thread.currentThread();
while (clockThread == myThread) {
repaint();
try {
Thread.sleep(1000);
} catch (InterruptedException e){}
}}
```

• The exit condition for this run method is the exit condition for the while loop because there is no code after the while loop:

```
while (clockThread = = my Thread)
```

The Life Cycle of a Thread

• Stopping a Thread :

– Stopping the Clock applet thread

• When we leave the page, the application in which the applet is running calls the applet's stop method. This method then sets the clockThread to null, thereby telling the main loop in the run method to terminate:

```
public void stop() {// applets' stop method
clockThread = null; }
```

• If we revisit the page, the start method is called again and the clock starts up again with a new thread.

The Life Cycle of a Thread

• The **isAlive()** Method :

– The **isAlive()** method returns true if the thread has been started and not stopped.

– If the **isAlive()** method returns false, you know that the thread either is a New Thread or is dead.

– If the **isAlive()** method returns true, you know that the thread is either Runnable or Not Runnable.

Thread Priority

• The Java runtime supports fixed priority preemptive scheduling.

• When a Java thread is created, it inherits its priority from the thread that created it. Which can be changed later using the *setPriority* method.

• Thread priorities are integers ranging between MIN_PRIORITY and MAX_PRIORITY.

Thread Priority

• The runtime system chooses the runnable thread with the highest priority for execution.

• If two threads of the same priority are waiting for the CPU, the scheduler chooses one of them to run in a round-robin fashion.

• The chosen thread will run until one of the following conditions is true:

- A higher priority thread becomes runnable.
- It yields, or its run method exits.

- On systems that support time-slicing, its time allotment has expired.

Controlling Threads

• It's an art of moving threads from one state to another state.
• It is achieved by triggering state transitions.
• The various pathways out of the running state can be :

- Yielding
- Suspending and then resuming
- Sleeping and then waking up
- Blocking and then continuing
- Waiting and then being notified

Yielding

• A call to yield method causes the currently executing thread to move to the ready state, if the scheduler is willing to run any other thread in place of the yielding thread.
• A thread that has yielded goes into ready state.
• The yield method is a static method of the Thread class.

scheduled yield()
running
ready

Suspending

• It's a mechanism that allows any arbitrary thread to make another thread un-runnable for an infinite period of time.
• The suspended thread becomes runnable when some other thread resumes it
• Deadlock prone, because the control comes from outside the thread.
• The exact effect of suspend and resume is much better implemented using wait and notify.

Sleeping

• A sleeping thread passes time without doing anything and without using the CPU.
• A call to the sleep method requests the currently executing thread to cease it's execution for a specified period of time.

- public static void sleep(long ms)

throws InterruptedException

- public static void sleep(long ms, int ns)

throws InterruptedException

• Both sleep and yield work on currently executing thread.

Blocking

• Methods that perform input or output have to wait for some occurrence in the

outside world before they can proceed, this behavior is known as blocking.

```
try {
Socket s = new Socket("Devil",5555);
InputStream is = s.getInputStream();
int b = is.read();
}
catch(IOException e) {
// Handle the exception}
```

Blocking

- A thread can also become blocked if it fails to acquire the lock for a monitor if it issues a **wait( )** call.
- The **wait( )** method puts an executing thread into the waiting state, and the **notify( )** and **notifyAll( )** put them back to ready state.

Thread Synchronization

```
public class Consumer extends Thread {
private MailBox mailBoxObj;
public Consumer(Mailbox box) {
this.mailBoxObj = box;
}
public void run() {
while(true) {
if(mailBoxObj.request) {
System.out.println(mailBoxObj.msg);
mailBoxObj.request = false;
}
try {
sleep(50);
} catch(InterruptedException e){}
}
}
```

Better Solution

```
class MailBox{
private boolean request;
private String msg;
public synchronized void storeMessage( String s){
request = true;
msg = s;
}
public synchronized String retrieveMessage(){
request = false;
return msg;
}
}
```

The Complete Solution

```
public synchronized String retrieveMessage(){
while(request == false){ // No message to retrieve
try{
wait();
} catch (InterruptedException e) {}
}
request = false;
notify();
return str;
}
```

Summary

In this session, you learnt to:

- Define a thread
- Create threads using two approaches
- Describe the life cycle of a thread
- Synchronize threads

---

extends Vs implements

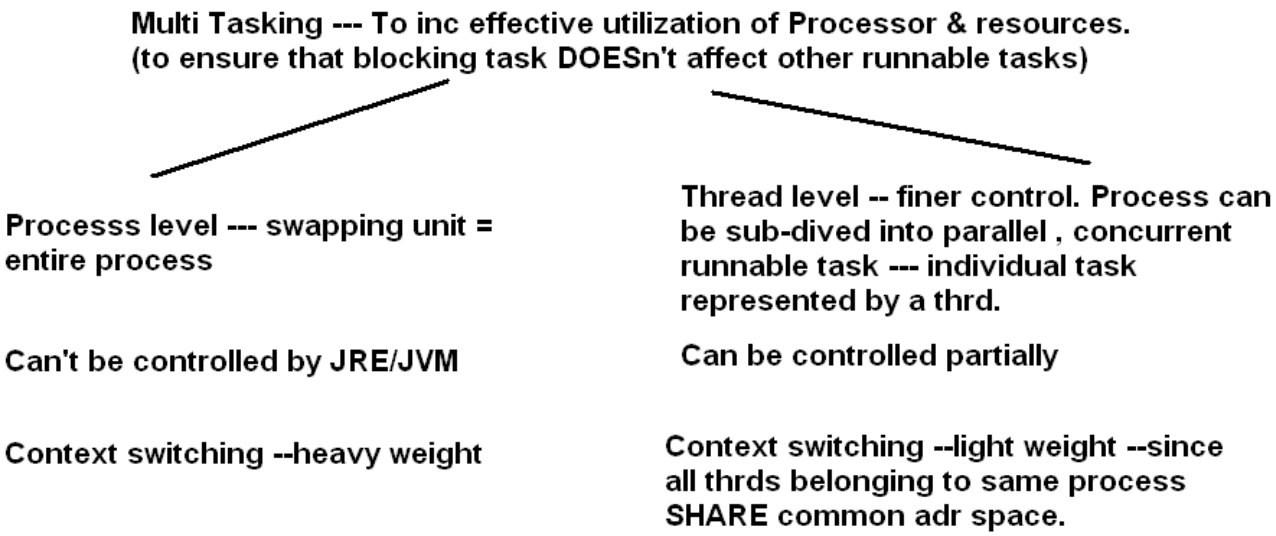
class MyThrd extends Thread
MyThrd --IS A thread. Can use directly Thread's inherited API .
Constr -- Thread(String nm)
start() -- rdy pool.

Must override
public void run() -- to supply B.L

class MyRunnableTask imple. Runnable
MyRunnableTask -- IS NOT a Thread.
Typically declare --Thread type reference -- private Thread t;
Constr -- Thread(Runnable inst,String nm)
eg : t=new Thread(this,nm);
t.start();
MUST override
public void run() -- B.L

Blocking Triggers Vs Un blocking triggers

1. Blocked on sleep (eg : Thread.sleep(ms))	1.1 sleep over 1.2 interrupt (InterruptedException)
2. Blocked in i/o eg : System.in.read()	2 Data available or write operation complete
3. Blocked on join join() join(long ms)	3.1 specified thread completes exec(dead) 3.2 interrupt 3.3 tmout elapsed.
4. Blocked on monitor	4 Monitor being free(lock released)
5. Blocked on wait Object class API public void wait() throws IE public void wait(long ms) throws IE	5.1 notify 5.2 notifyAll 5.3 interrupt 5.4 tmout elapsed



DAY 19 :

Today's Topics
Revision
Applying threads to a practical scenario
Synchronization
User Threads vs Daemon Threads
Cloning

Revise
Thread state transitions n API
1. Doesn't Exist ---->NEW
Which different techniques ? : extends Thread or imple Runnable i/f

Runnable : i/f : Functional i/f
SAM : public void run()
Any relation between Runnable i/f n Thread class ? IS A

Ctors
extends Thread
Thread(String name)

imple Runnable
Thread(Runnable instance,String name)

2. NEW ---> Ready-To-Run (pool=> sharing of resources & collection of equivalent resources)
Ready-To-Run => thrds in this state : are not blocked on any condition , simply waiting for CPU time
How to trigger this transition ? Method of Thread class : public void start()

3. Ready --> Running : controlled by Native O.S (scheduler)
(execs run() method : B.L / exec logic)

4. Running ---> Ready (context switching)
triggers : time slice over / pre emption / Thread.yield() : un reliable

Ready + Running => RUNNABLE

5. Running ---> dead
run() over

Triggers for java.lang.IllegalThreadStateException : 1.dead : start()
2. Runnable --> start()

6. Running ---> BLOCKED

6.1 sleep : public static void sleep(long msec) throws InterruptedException

eg : t1 : run()
Thread.sleep(1000);
1. After 1 sec : thrd is unblocked ---rdy pool --resume running
OR
2. t1 : run()
Thread.sleep(1000);

t2 : run()
t1.interrupt(); //t2 ---> interrupt ---> t1 , t2 : runnable
t1 : un blocked --- InterruptedException --> rdy pool
t1 : running ---> try block aborted --enters catch block....

6.2 join :
t1 : run :
try {
....
t2.join(1000);//t1 waits for t2 to complete

```
 sop("cntd....");
} catch-all
...
```

Suppose t2 : t2 gets over in 500msec : dead : t1 : unblocked --rdy --running --cntd
OR
t3 : run : t1.interrupt() : 700ms -- : t1 : unblcoked --rdy pool --running --catch-all : InterruptedExc
OR
t2 : runnable
no interrupts from any thrds n t2 does not get over
t1 : after 1sec : t1 : unblocked --rdy --running --cntd

6.3 blocked on i/p
eg : t1 's run : sc.nextInt() OR System.in.read() : causes the invoker thread to block on i/p
t1 : blocked on i/p

t2 : run()
t1.interrupt(); //t2 ---> interrupt ---> t1 , t2 : runnable
t1 : continues to be BLOCKED (i.e interrupt IS NOT an unblocking trigger for the thrds which are blocked on i/p)
unblocking trigger for the thrds which are blocked on i/p : data available

6. When blocking condition is removed
BLOCKED ---> Ready-to-Run(competes with other thrds in ready pool)

What is mandatory in either of scenarios (extends or implements) ?
MUST override/implement run()

What will happen ?
1. extends Thread n don't override run() : no err , do nothing(nop) thrd
2. implements Runnable n don't implement run() : javac err

What will happen ?
Instead of calling start() , run() method is invoked?
no err , FATAL : no multi threading (seq. execution done by main thread)

---

1. Solve practical requirement based on (Collection + I/O + Threads)

Solve : Create a multi threaded application for saving student details in 2 text files(taken from the Map of students) : (in student_gpa.txt : sorted by gpa & in student_dob.txt : sorted by dob) using 2 different threads , concurrently!
(refer to a diag : "collection-io-thrds")

1.1 CollectionUtils : sorting methods

1.2 IOUtils : writeData : text data + buffering
Chain of I/O streams

1.3 Create runnable tasks (implements scenario)
1.4 Tester

Threads :
main : get sample data , scanner , create child thrds n start the same

dob task : constr , state ,run

gpa task : constr , state ,run

2. (refer to a diag : "assgn-help")
1. Consider joint account : BankAccount
inst var : balance
1.1 update balance(double amount)
withdraw amount.... think time(sleep) --- cancel withdraw
what should be the balance after updateBal :
1.2 check balance
return curnt balance

2. Runnable Tasks
UpdaterTask
run() : invoke BankAccount's update balance
jointAccount.updateBalance(...);

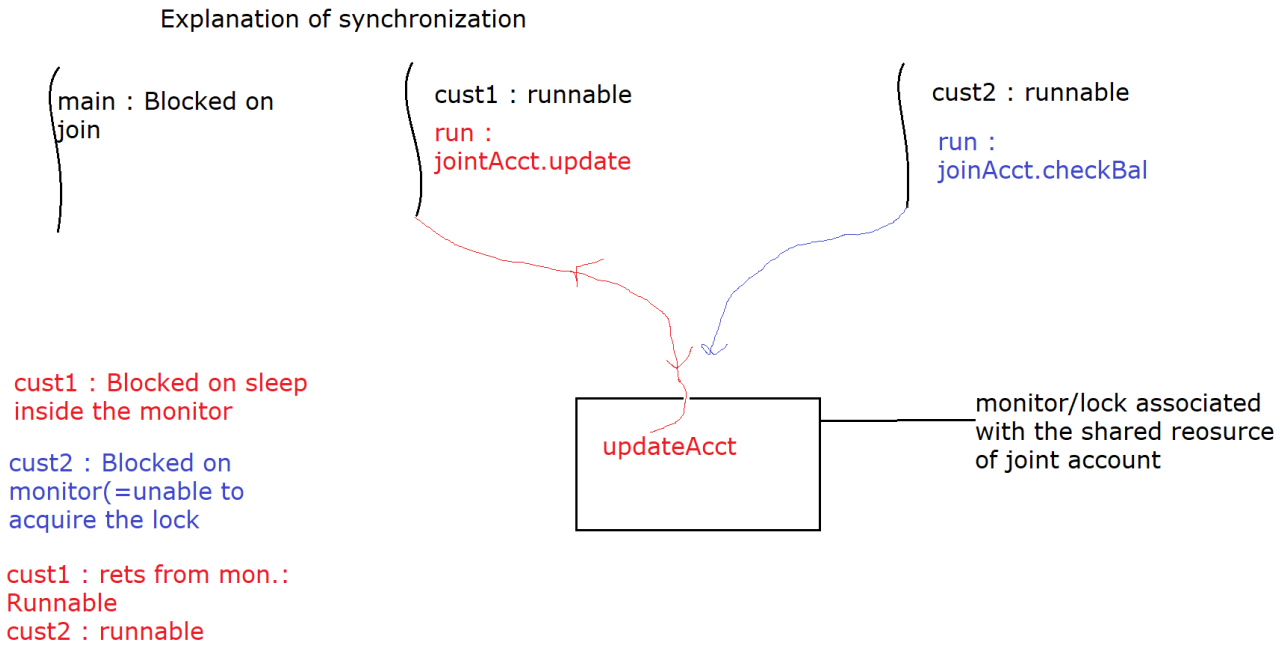
CheckerTask

run() : invoke SAME BankAccount's check balance
jointAccount.checkBalance();

3. main : create single joint acct instance n pass it to these tasks :

Problem : race condition
Solution : synchronization

Synchronized method
Synchronized blocks



A race condition is a special condition that may occur inside a critical section. A critical section is a section of code that is executed by multiple threads and where the sequence of execution for the threads makes a difference in the result of the concurrent execution of the critical section.

When the result of multiple threads executing a critical section may differ depending on the sequence in which the threads execute, the critical section is said to contain a race condition.

Race condition means that the threads are racing through the critical section, and that the result of that race impacts the result of executing the critical section.

Critical Sections
Running more than one thread inside the same application does not by itself cause problems. The problems arise when multiple threads access the same resources. For instance the same memory (variables, arrays, or objects), systems (databases, web services etc.) or files.

In fact, problems only arise if one or more of the threads write to these resources. It is safe to let multiple threads read the same resources, as long as the resources do not change.

Here is a critical section Java code example that may fail if executed by multiple threads simultaneously:

```
public class Counter {

   protected long count = 0;

   public void add(long value){
      this.count = this.count + value;
   }
}
```

Imagine if two threads, A and B, are executing the add method on the same instance of the Counter class. There is no way to know when the operating system(scheduler) switches between the two threads. The code in the add() method is not executed as a single atomic instruction by the Java virtual machine. Rather it is executed as a set of smaller instructions, similar to this:

Read this.count from memory into PC register.
Add value to PC register.
Write register to memory.
Observe what happens with the following mixed execution of threads A and B:

```
    this.count = 0;
```

A: Reads this.count into a register (0)
B: Reads this.count into a register (0)
B: Adds value 2 to register
B: Writes register value (2) back to memory. this.count now equals 2
A: Adds value 3 to register
A: Writes register value (3) back to memory. this.count now equals 3

The two threads wanted to add the values 2 and 3 to the counter. Thus the value should have been 5 after the two threads complete execution. However, since the execution of the two threads is interleaved, the result ends up being different.

In the execution sequence example listed above, both threads read the value 0 from memory. Then they add their individual values, 2 and 3, to the value, and write the result back to memory. Instead of 5, the value left in this.count will be the value written by the last thread to write its value. In the above case it is thread A, but it could as well have been thread B.

Race Conditions in Critical Sections
The code in the add() method in the example earlier contains a critical section. When multiple threads execute this critical section, race conditions occur.

More formally, the situation where two threads compete for the same resource, where the sequence in which the resource is accessed is significant, is called race conditions. A code section that leads to race conditions is called a critical section.

Preventing Race Conditions
To prevent race conditions from occurring you must make sure that the critical section is executed as an atomic instruction. That means that once a single thread is executing it, no other threads can execute it until the first thread has left the critical section.

What happens when you call start on NEW Thread ?

It internally invokes a native method(not written in java) start0()

Its invocation will --

1. cause a new native thread-of-execution to be created (by native OS)
2. cause the run method to be invoked on that thread.

Race condition

The situation where two or more threads compete for the same resource, where the sequence in which the resource is accessed is significant, is called race conditions.

Critical Section
A code section that leads to race conditions is called a critical section.
eg : Joint Bank Account : shared resource
updateBalance n checkBalance

Preventing Race Conditions
To prevent race conditions from occurring you must make sure that the critical section is executed as an atomic instruction. That means that once a single thread is executing it, no other threads can execute it until the first thread has left the critical section.

Synchronization = Applying thread safety or applying locks

When is synchronization required ?
It's required iff multiple threads are sharing SAME common resource(eg : any collection,data file , db table ,socket , reservation...) & one thread is accessing & other one is modifying the resource.

How to apply synchronization in java ?
keyword -- synchronized.
Can appear as method modifier or at block level.

To avoid race condition / guard critical section , you apply synchronization.

Using synchronized keyword , a lock is applied at the object level.(i.e instance of the shared resource
eg : JointAccount)

Important statements

1. lock/monitor can be associated with any java object.
2. When does thrd need to acquire the lock (=enter the monitor)?-- if its invoking either synchronized methods or code from synchronized blocks
3. Can single thrd acquire multiple locks -- YES
4. Blocking trigger
unable to acquire lock(enter monitor) : Blocked on monitor/lock
Un blocked -- lock released / monitor free.(synchronized method rets or synchronized block over)
5. If a thread invokes sleep(or invokes join,yield,notify) or encounters context switching , it holds any locks it has—it doesn't release them.

What's the need of synchronized blocks?

1. Instead of writing long synchronized methods (n thus reducing the performance due to larger extent of the lock) , identify critical section & guard it using synchronized block.

2. While using inherently thread un safe API(StringBuilder, ArrayList,LinkedList,HS,LHS,HM...) in multi thrded environment : you can still apply thread safety : using synchronized blocks.

synchronized block syntax

synchronized(shared resource ref.)
{
Access the methods of shared resource in mutually exclusive manner.
}

1. Only methods (or blocks) can be synchronized, not variables or classes.
2. Each object has just one lock.
3. Not all methods in a class need to be synchronized. A class can have both synchronized and non-synchronized methods.

4. If two threads are about to execute a synchronized method in a class, and both threads are using the same instance of the class to invoke the method,only one thread at a time will be able to execute the method. The other thread will need to wait until the first one finishes its method call. In otherwords, once a thread acquires the lock on an object, no other thread can enter ANY of the synchronized methods in that class (for that object).

5. If a class has both synchronized and non-synchronized methods, multiple threads can still access the class's non-synchronized methods. If you have methods that don't access the data you're trying to protect, then you don't need to synchronize them. Synchronization can cause a hit in some cases (or even deadlock if used incorrectly), so you should be careful not to overuse it.
6. If a thread goes to sleep(or invokes join,yield,notify) or encounters context switching , it holds any locks it has—it doesn't release them.
7. A thread can acquire more than one lock. For example, a thread can enter a synchronized method, thus acquiring a lock, and then immediately invoke a synchronized method on a different object, thus acquiring that lock as well. As the stack unwinds, locks are released again. Also, if a thread acquires a lock and then attempts to call a synchronized method on that same object, no problem. The JVM knows that this thread already has the lock for this object, so the thread is free to call other synchronized methods on the same object, using the lock the thread already has.
eg :

```
class A {
private B b1;
 synchrnized void test()
 {
   ...
   b1.testMe();
 }
}
class B
{

 synchronized void testMe()
 {
  //some B.L
 }
}
```

Similar can be achieved using nested synchronized blocks.

8. You can synchronize a block of code rather than a method.
When to use synched blocks?

Because synchronization does hurt concurrency, you don't want to synchronize any more code than is necessary to protect your data. So if the scope of a method is more than needed, you can reduce the scope of the synchronized part to something less than a full method—to just a block. OR when u are using Thread un-safe(un-sunchronized eg -- StringBuilder or HashMap or HashSet) classes in your appln.

Regarding static & non -static synchronized

1. Threads calling non-static synchronized methods in the same class will only block each other if they're invoked using the same instance. That's because they each lock on "this" instance, and if they're called using two different instances, they get two locks, which do not interfere with each other.
2. Threads calling static synchronized methods in the same class will always block each other—they all lock on the same Class instance.

3. A static synchronized method and a non-static synchronized method will not block each other, ever. The static method locks on a Class instance(java.lang.Class<?>) while the non-static method locks on the "this" instance—these actions do not interfere with each other at all.

DAY 20 :

Continue with Synchronization
ITC
User Threads vs Daemon Threads
Cloning

Revise
What is synchronization ?

What is the need?

When it's required ? :

Immutable objects(eg : String,LocalDate,LocalTime..,wrappers) are inheretly thread safe or thread un safe ?

Which techniques ?

Refer to : "readme synchronization"

Fix the problem.

Understand this.
1. Lock/monitor is assciated with : java object / method / block / class
2. When does thrd need to acquire the lock ?
3. Can single thrd acquire multiple locks ?
4. Can 1 lock be shared across multiple threads ?

5. Blocking trigger : Blocked on a monitor =>
Un blocking trigger : Lock released or monitor free

6. Name the conditions under which lock is not released ? :
7. Can a class contain synchronized as well as non synchronized methods?
8. Can you run thread un safe APIs , in thread safe manner ?
How ?

ITC

Refer to ready code : no_itc
Does Utils class contain Thread safe APIs (read n write methods) :
Is atomicity(mutually exclusive behavior) seen ?
(i.e while producer is producing data , can consumer read it n vice versa :)

Are there any problems observed even after applying synchronization :

What's the cause of it ?

Problem n it's solution

User vs Daemon Threads

Cloning

User Threads Vs Daemon Threads

Created by default (i.e by using any of the Thread class constr)	**Are created using Thread class method public void setDaemon(boolean flag) ---flag =true-- indicates creation of daemon thread. eg -- Thread t1=new Thread(new MyTask(),"abc"); t1.setDaemon(true);t1.start();**
User threads , if running , prevent JVM from termination	**Daemon threads, DON'T prevent JVM from termination (when all user threads have teminated)**
	Usage -- typically as service threads. eg -- Garbage collector

NOTE : Java appln terminates only when all user threads have terminated.

What are design patterns ?

Design patterns are generally sets of standard practices used in the software development industry. They represent the solutions given by the developer community to general problems faced in every-day tasks regarding software development.

Why Java design pattern?

They are reusable in multiple projects.
They provide a solution that helps to define the system architecture.
Capture software engineering experiences.
Provide transparency to the design of an application.
Well-proven since they are built upon the knowledge and experience of expert software developers.

What Is Gang of Four (GOF)?

In 1994, four authors Erich Gamma, Ralph Johnson, Richard Hel, and John Vlissides published a book titled Design Patterns Elements of Reusable Object-Oriented Software. This book introduced the concept of Design Pattern in Software development.
These four authors are known as Gang of Four GOF.

Categories Java Design patterns?

Based on problem they are categorized into

1. Creational patterns

Factory method/Template
Abstract Factory
Builder
Prototype
Singleton

2. Structural patterns

Adapter
Bridge
Filter
Composite
Decorator
Facade
Flyweight
Proxy

3. Behavioral patterns

Interpreter
Template method/ pattern
Chain of responsibility
Command pattern
Iterator pattern
Strategy pattern
Visitor pattern

4.J2EE patterns

MVC Pattern
Data Access Object pattern
Front controller pattern
Intercepting filter pattern
Data Transfer object pattern

Singleton

In object-oriented programming, a singleton class is a class that can have only one object (an instance of the class) at a time, in a particular JVM.
It is a creational design pattern which talks about the creation of an object.

After first time, if we try to instantiate the Singleton class, the new variable also points to the first instance created. So whatever modifications we do to any variable inside the class through any instance, it affects the variable of the single instance created and is visible if we access that variable through any variable of that class type defined.

eg : Device drivers, Cache , DB connection

How To design a singleton class?

Private constructor to restrict instantiation of the class from other classes.
Private static variable of the same class that is the only instance of the class.
Public static method that returns the instance of the class, this is the global access point for outer world to get the instance of the singleton class.

Lazy initialization
Mark constructor as private.
Write a static method that has return type object of this singleton class.

Eager initialization

In eager initialization, the instance of Singleton Class is created at the time of class loading, this is the easiest method to create a singleton class but it has a drawback that instance is created even though client application might not be using it.

Mark constructor as private.
Create static init block to instantiate a singleton

Factory design pattern

It is a creational design pattern which talks about the creation of an object. The factory design pattern says that define an interface (A java interface or an abstract class) and let the subclasses decide which object to instantiate. The factory method in the interface lets a class defer the instantiation to one or more concrete subclasses. It is one of the best ways to create an object where object creation logic is hidden to the client.

Implementation:
1. Define a factory method inside an interface.
2. Let the subclass implements the above factory method and decide which object to create.

In Java constructors are not polymorphic, but by allowing subclass to create an object, we are adding polymorphic behavior to the instantiation. i.e we are trying to achieve Pseudo polymorphism by letting the subclass to decide what to create, and so this Factory method is also called as Virtual constructor.

eg : Shape Scenario

DAY 21 :

Revise
What is synchronization ?
Applying thread safety

What is the need? To avoid race condition among threads

When it's required ? : In a multi threaded system , when multiple threads are sharing the COMMON resource , some threads are updating the state n others are accessing the state o.w might get exceptions(eg : ConcurrentModficiationExc) or wrong results.

Which techniques ? : synchronized methods or blcoks to guard critical section

Immutable objects(eg : String,LocalDate,LocalTime..,wrappers) are inheretly thread safe or thread un safe ?
inheretly thread safe

Refer to : "readme synchronization"

Answer this.
1. Lock/monitor is assciated with : java object / method / block / class : ANY java object
2. When is thrd FORCED to acquire the lock ? : while invoking either synched method or synched block
3. Can single thrd acquire multiple locks ? YES
4. Can 1 lock be shared across multiple threads ? NO

5. Blocking trigger : Blocked on a monitor => Thread is trying to access synched method or a block n lock is already acquired by some other thrd(i.e monitor is occupied)

Un blocking trigger : Lock released i.e monitor free

6. Name the conditions under which lock is not released ? : sleep, yield,join,notify,context switiching(running---> rdy to run)

7. Can a class contain synchronized as well as non synchronized methods? YES

8. Can you write static synched methods ? YES .
eg : You have Emp class's static synched method
Lock will be applied on "Emp.class" ---special immutable obj created by JVM , once per cls loading
(more later in reflection) ---java.lang.Class<Emp>

9. Can you invoke thread un safe APIs(eg : ArrayList's add , HM's put ....) , in thread safe manner ? YESS
How ? Typically using synched block

ITC

Refer to ready code : no_itc
Does Utils class contain Thread safe APIs (read n write methods) : YES
Is atomicity(mutually exclusive behavior) seen ? YES
(i.e while producer is producing data , can consumer read it n vice versa :) NO

Are there any problems observed even after applying synchronization :

What's the cause of it ?

Problem n it's solution

User vs Daemon Threads

Check Mysql DB settings n keep it ready.

Cloning
code sample : Emp e1=new Emp(...);
Emp e2=e1;//copy of refs
AL<Emp> list=new AL<>(Arrays.asList(e1,e2));

Tester : in different pkg than Emp
Emp e3=e1.clone();

Reflection

JVM Architecture

ITC with sleep : deadlock

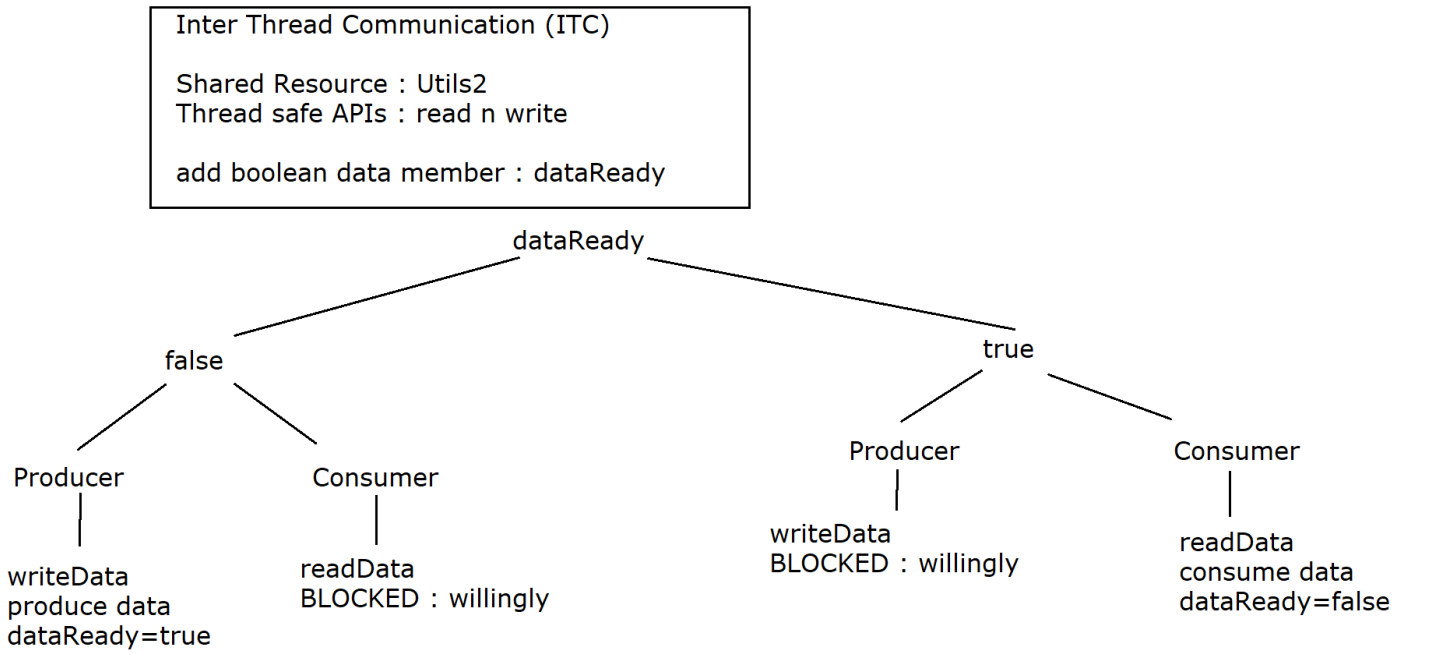
started thrd -- Producer
w entered --- Producer
Write Data Emp [id=1, sal=100.0]
w exited --- Producer
Press enter to exit
started thrd -- Consumer
r entered --- Consumer
Read Data Emp [id=1, sal=100.0]
r exited --- Consumer // dataReady : false
w entered --- Producer
Write Data Emp [id=2, sal=200.0]
w exited --- Producer //dataReady : true
w entered --- Producer //Blocked on sleep : inside the monitor , main: blocked on i/p --join ,
//c : Blocked on monitor : outside
problem : deadlock --- circular dependency

ITC with wait
started thrd -- Producer
w entered --- Producer
Write Data Emp [id=1, sal=100.0]
w exited --- Producer //dataReady=true
w entered --- Producer //p : Blocked on wait : outside
Press enter to exit //main : blocked on i/p ---join
started thrd -- Consumer
r entered --- Consumer
Read Data Emp [id=1, sal=100.0]
r exited --- Consumer // dataReady=false
r entered --- Consumer //c : Blocked on wait : outside

ITC with wait-notify

started thrd -- Producer
w entered --- Producer
Write Data Emp [id=1, sal=100.0]
w exited --- Producer
Press enter to exit
started thrd -- Consumer
r entered --- Consumer
Read Data Emp [id=1, sal=100.0]
r exited --- Consumer
w entered --- Producer
Write Data Emp [id=2, sal=200.0]

w exited --- Producer //dataReady=true
w entered --- Producer //p : Blocked on wait : outside the monitor , main : blocked on i/p , c : runnable
r entered --- Consumer
Read Data Emp [id=2, sal=200.0]
r exited --- Consumer //dataRdy :false , c:runnable , p :runnable , main : blocked on i/p
Write Data Emp [id=3, sal=300.0]
w exited --- Producer
w entered --- Producer
r entered --- Consumer
Read Data Emp [id=3, sal=300.0]
r exited --- Consumer
Write Data Emp [id=4, sal=400.0]
w exited --- Producer
w entered --- Producer
r entered --- Consumer
Read Data Emp [id=4, sal=400.0]
r exited --- Consumer
Write Data Emp [id=5, sal=500.0]
w exited --- Producer
w entered --- Producer
r entered --- Consumer
Read Data Emp [id=5, sal=500.0]
r exited --- Consumer



JVM architecture (refer to a diagram)

There are mainly three sub systems in the JVM
1. ClassLoader
2. Runtime Memory/Data Areas
3. Execution Engine

1. ClassLoader
This component is responsible for loading the class files to the method area (typcially in RAM) since JVM resides on the RAM and it performs : loading, linking, and initialization.

1.1 Loading

This process usually starts with loading the main class (class with the main()method). ClassLoader reads the .class file and then the JVM stores the following information in the method area.
1. The fully qualified name of the loaded class
2. variable information
3. immediate parent information
4. Type : whether it is a class or interface or enum

Note —Only for the first time, JVM creates an object from a class type object(java.lang.Class) for each loaded java class and stores that object in the heap.

The three main ClassLoaders in JVM,
1. Bootstrap ClassLoader — This is the root class loader and it is the superclass of Extension/Platform ClassLoader. This loads the standard java packages which are inside the rt.jar/jrtfs.jar : java.base module (base file and some other core libraries.)
eg : java.util.ArrayList

2. Extension/Platform ClassLoader — This is the subclass of the Bootstrap ClassLoader and a superclass of Applications ClassLoader. This is responsible for loading classes that are present inside the directory (jre/lib/ext). From JDK 9 onwards , it's replace by Platform class loader . Loads additional classes from other modules eg : java.sql

3. Application ClassLoader — This is the subclass of Extension/Platform ClassLoader and this is responsible for loading the class files from the classpath (classpath can be modified by adding the -classpath command-line option)
eg : tester.TestMe

The four main principles in JVM,
1. Visibility Principle — This principle states that the ClassLoader of a child can see the class loaded by Parent, but a ClassLoader of parent can’t find the class loaded by Child.
2. Uniqueness Principle — This principle states that a class loaded by the parent ClassLoader shouldn’t be loaded by the child again. This ensures that there is no class duplicated.

3. Delegation Hierarchy Principle — This rule states that JVM follows a hierarchy of delegation to choose the class loader for each class loading request. Here, starting from the lowest child level, Application ClassLoader delegates the received class loading request to Extension ClassLoader, and then Extension ClassLoader delegates the request to Bootstrap ClassLoader. If the requested class is found in the Bootstrap path, the class is loaded. Otherwise, the request again transfers back to the Extension ClassLoader level to find the class from the Extension path or custom-specified path. If it also fails, the request comes back to Application ClassLoader to find the class from the System classpath and if Application ClassLoader also fails to load the requested class, then we get the run time exception — ClassNotFoundException.
4. No Unloading Principle — This states that a class cannot be unloaded by the Classloader even though it can load a class.

2. Linking
This process is divided into three main parts .
1. Verification
This phase check the correctness of the .class file.
Byte code verifier will check the following:
1.1 whether it is coming from a valid compiler or not (Because anyone can create their own compiler).
1.2 whether the code has a correct structure and format.

if any of these are missing, JVM will throw a runtime exception called “java.lang.VerifyError” Exception. if not, then the preparation process will take place.

2. Preparation
In this phase, variables memory will be allocated for all static data members and assigned with default values based on the data types.
eg : reference — null
int — 0
boolean— false
eg :
static boolean active=true;
So in this phase, it will check the code and the variable status in boolean type so JVM assigns false to that variable.

3. Resolution
This is the process of replacing the symbolic references with direct references and it is done by searching into the method area to locate the referenced entity.

The JVM does not understand the name that we give to create objects(reference variables) . So the JVM will assign memory location for those objects by replacing their symbolic links with direct links.

eg : In TestBox class
Box b1=new Box(1,2,3);
b1.showDims();

3. Initialization
In this phase, the original values will be assigned back to the static variables as mentioned in the code and a static initilizer block will be executed(if any). The execution takes place from top to bottom in a class and from parent to child in the class hierarchy.

Important : JVM has a rule saying that the initialization process must be done before a class becomes an active use.

Active use of a class are,
1. using new keyword. (eg: Vehicle car=new Vehicle();).
2. invoking a static method.
3. assigning value to a static field.
4. if a class is an initial class (class with main()method).
5. using a reflection API (Class's newInstance()method).
6. initializing a subclass from the current class.

There are four ways of initializing a class :
using new keyword — this will goes through the initialization process.
using clone(); method — this will get the information from the parent object (source object).
using reflection API (newInstance();) — this will goes through the initialization process.
using IO.ObjectInputStream(); — this will assign initial value from InputStream to all non-transient variable

Runtime Data Area
JVM memory is basically divided into five following parts,

1. Method Area
This is where the class data is stored during the execution of the code and this holds the information of static variables, static methods, static blocks, instance methods, class name, and immediate parent class name(if any). This is a shared resource.

2. Heap Area
This is where the information of all objects(state: non static data members) is stored and it’s a shared resource just like the method area

eg : Book book = new Book(...);
So here, there is an instance of Book is created and it will be loaded into the Heap Area preceded by Book's class information loaded in the method area + creation of Class<Book> instance , in the heap.

Note — there is only one method area and one heap area per JVM.

3. Stack Area
All the local variables, method calls, and partial results of a program (not a native method) are stored in the stack area.

For every thread, a runtime stack will be created. A block of the stack area is known as “Stack Frame” and it holds the local variables of method calls. So whenever the method invocation is completed, the frame will be removed (POP). Since this is a stack, it uses a Last-In-First-Out structure.

4. PC Register (Program Counter Register)
This will hold the thread’s executing information. Each thread has its own PC registers to hold the address of the current executing information and it will be updated with the next execution once the current execution finishes.

5. Native Method Area
This will hold the information about the native methods and these methods are written in a language other than Java, such as C/C++. Just like stack and PC register, a separate native method stack will be created for every new thread.
Take a look at the following diagram,

eg scenario for Thread 1 (T1),

```
M1(){
  M2();
}
----------------
 M2(){
    M3();
 }
```

When the M1 method is called, the first frame will be created in the T1 thread and from there it will go to method M2 at that time the second frame will be created and from there it will go to method M3 as in the above demo code, so a new frame will be created under M2.
Whenever the method exits, the stack frames will be destroyed respectively.

3. Execution Engine

This is where the execution of bytecode (.class) occurs and it executes the bytecode line-by-line. Before running the program, the bytecode should be converted into machine code.

Mainly, Execution Engine has three main components for executing the Java classes,

Components of Execution Engine

3.1 Interpreter

This is responsible for converting bytecode into machine code. This is slow because of the line-by-line execution even though this interprets the bytecode quickly. The main disadvantage of Interpreter is that when the same method is called multiple times, every time a new interpretation is required and this will reduce the performance of the system. So this is the reason where the JIT compiler will run parallel to the Interpreter.

3.2 JIT Compiler (Just In Time Compiler)

This overcomes the disadvantage of the interpreter. The execution engine first uses the interpreter to execute the bytecode line-by-line and it will use the JIT compiler when it finds some repeated code. (Eg: calling the same method multiple times). At that time JIT compiler compiles the entire bytecode into native code (machine code). These native codes will be stored in the cache. So whenever the repeated method is called, this will provide the native code. Since the execution with the native code is quicker than interpreting the instruction, the performance will be improved.

3.3 Garbage Collector

This will check the heap area whether there are any unreferenced objects and it destroys those objects to reclaim the memory. So it makes space for new objects. This runs in the background and it makes the Java memory efficient.

There are two phases involved in this process,
Mark — In this area, Garbage Collector identifies the unsued objects in the heap area.
Sweep — In here, Garbage Collector removes the objects from the Mark.
This process is done by JVM at regular intervals and it can also be triggered by calling System.gc() method.

3.4 Java Native Interface (JNI)

This is used to interact with the Native(non-java) Method libraries (C/C++) required for the execution. This will allows JVM to call those libraries to overcome the performance constraints and memory management in Java.

3.5 Native Method Libraries

These are the libraries that are written in other programming(non-java) languages such as C and C++ which are required by the Execution Engine. This can be accessed through the JNI and these library collections mostly in the form of .dll or .so file extension.

Regarding garbage collection

Garbage Collection is a process to identify and delete the objects from Heap memory which are not in use. GC frees the space after removing unreferenced objects.

Garbage= un -referencable object.

Automatic Gargabe Collection --- to avoid memory. leaks/holes

JVM creates 2 system thrds --- main thrd(to exec main() sequentially) -- foreground thrd
G.C --- daemon thrd ---background thrd --- JVM activates it periodically(only if required) --- GC releases the memory occupied by un-referenced objects allocated on the heap(the objects whose no. of ref=0)

How to request for GC ?
API of System class
public static void gc()
eg : System.gc();//it's simply a REQUEST to JVM , for running GC htread.

Object class API
protected void finalize() throws Throwable
Automatically called by the garbage collector on an object before garbage collection of the object takes place.

Whenever Heap is under pressure (i.e there is no space in heap to create more objects) , JVM activates the GC thread. BUT it refers to Heap only. The classes which are loaded in method area are unloaded : when JVM terminates.

----------1st half over----------

Releasing of non- Java resources.(eg - closing of DB connection, closing file handles,closing socket connections) is NOT done automatically by GC

Triggers for marking the object for GC(candidate for GC)

1. Nullifying all valid refs.
eg : Box b1=new Box(1,2,3);
Box b2=b1;
b1=b2=null;//Box obj is marked for GC
2. re-assigning the reference to another object
eg : Box b1=new Box(10,20,30);
b1=new Box(2,3,4);
3. Object created within a method & its ref NOT returned to the caller.

4. Island of isolation

More Details

Garbage Collection is a process to identify and delete the objects from Heap memory which are not in use. GC frees the space after removing unreferenced objects.

The event in which Garbage Collectors are doing their job is called “Stop the world” event which means all of your application threads are put on hold until the garbage is collected.

How Garbage Collector works

The basic process of Hotspot JVM Garbage collector completes in two phases:

1. Marking
This phase is called marking phase in which GC identifies which objects are in use or which are not. All objects are scanned in the marking phase to make this determination.

2. Deletion
In Deletion phase, the marked object is deleted and the memory is released. Deletion of the unreferenced objects can be done in two ways:

2.1 Normal Deletion: In this phase, all unused objects will be removed and memory allocator has pointers to free space where a new object can be allocated.
OR
2.2 Deletion and Compaction: As you see in normal deletion there are free blocks between referenced objects. To further improve performance, in addition to deleting unreferenced objects, remaining referenced object will be compacted(re aligned).

Why Heap divided into Generations ?
It is a time consuming process to scan all of the objects from a whole heap and further mark and compact them. The list of the object grows gradually which leads to longer garbage collection time as more and more objects are allocated with time.

In General Applications most of the objects are short-lived. Fewer and fewer objects remain allocated over time.

That’s why to enhance the performance of the JVM, Heap is broken up into smaller parts called generations and JVM performs GC in these generations when the memory is about to fill up.

Generational Process of Garbage Collection

1. New objects are allocated in Eden Space of Young Generation. Both Survivor Spaces are empty in starting.
2. A minor garbage collection will trigger once the Eden space fills up.
Referenced objects are moved to the S0 survivor space and Eden Space will be cleared and all unreferenced objects will be deleted.
3. It will happen again to Eden space when next time GC will be triggered. But, in this case, all referenced objects are moved to S1 survivor space. In addition, objects from the last minor GC on the S0 survivor space have their age incremented and get moved to S1. Now both Eden and S0 will be cleared, and this process will repeat every time when GC is triggered. On every GC triggered, survivor spaces will be switched and object's age will be incremented.
4. Once the objects reach a certain age threshold, they are promoted from young generation to old generation. So, this is how objects promotion takes place.
5. The major GC will be triggered once the old generation completely fills up.

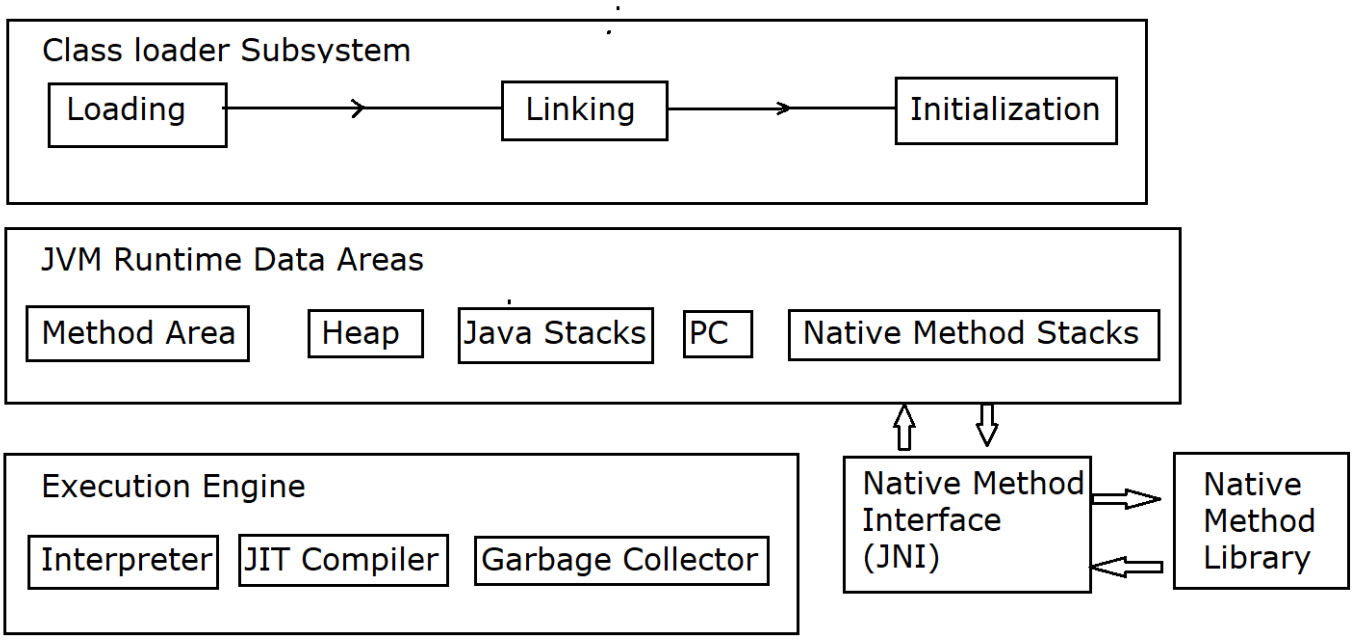
Available Garbage collectors in Hotspot JVM
1. Serial Garbage Collector: Serial GC designed for the single-threaded environments. It uses just a single thread to collect garbage. It is best suited for simple command-line programs. Though it can be used on multiprocessors for applications with small data sets.

2. Parallel Garbage Collector: Unlike Serial GC it uses multiple threads for garbage collection. It is a default collector of JVM and it is also called the Throughput garbage collector.

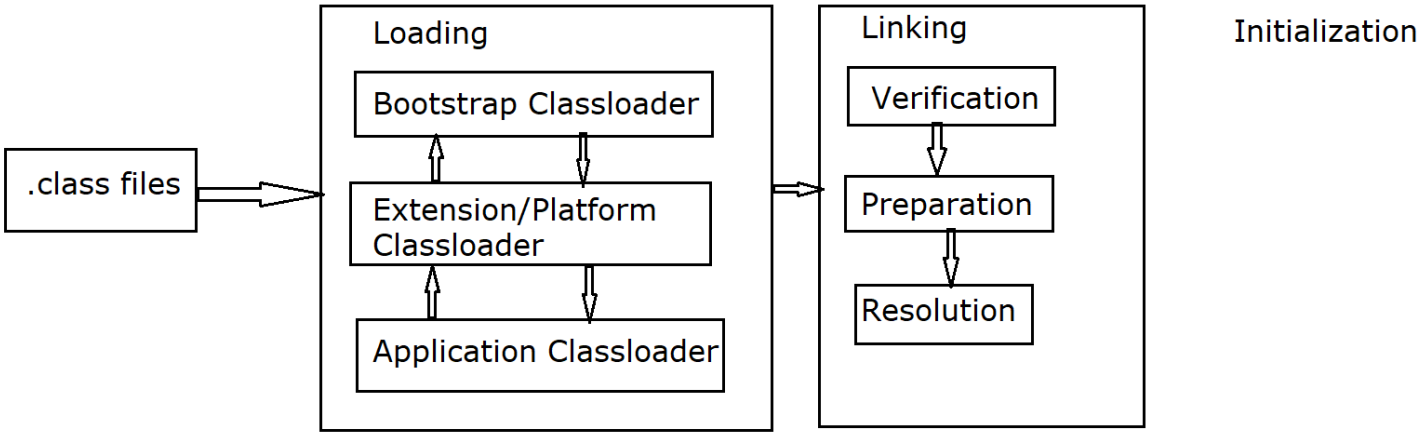
3. CMS(concurrent mark & sweep) Garbage Collector: CMS uses multiple threads at the same time to scan the heap memory and mark in the available for eviction and then sweep the marked instances.

4. G1 Garbage Collector: G1 Garbage collector is also called the Garbage First. It is available since Java 7 and its long-term goal is to replace the CMS collector. The G1 collector is a parallel, concurrent, and incrementally compacting low-pause garbage collector.

JVM Architecture Overview



Class loader subsystem



Introducing NIO

NIO was created to allow Java programmers to implement high-speed I/O without having to write custom native code. NIO moves the most time-consuming I/O activities (namely, filling and draining buffers) back into the operating system, thus allowing for a great increase in speed.

Identifying differences between IO and NIO

1) IO streams versus NIO blocks

The most important distinction between the original I/O library (found in java.io.*) and NIO has to do with how data is packaged and transmitted. As previously mentioned, original I/O deals with data in streams, whereas NIO deals with data in blocks.

A stream-oriented I/O system deals with data one or more bytes at a time. An input stream produces one byte of data, and an output stream consumes one byte of data. It is very easy to create filters for streamed data. It is also relatively simply to chain several filters together so that each one does its part in what amounts to a single, sophisticated processing mechanism. Important thing is that bytes are not cached anywhere. Furthermore, you cannot move forth and back in the data in a stream. If you need to move forth and back in the data read from a stream, you must cache it in a buffer first.

A block-oriented I/O system deals with data in blocks. Each operation produces or consumes a block of data in one step. Processing data by the block can
be much faster than processing it by the (streamed) byte. You can move forth and back in the buffer as you need to. This gives you a bit more flexibility during processing. However, you also need to check if the buffer contains all the data you need in order to fully process it. And, you need to make sure that when reading more data into the buffer, you do not overwrite data in the buffer you have not yet processed. But block-oriented I/O lacks some of the elegance and simplicity of stream-oriented I/O.

Read more: 3 ways to read files using Java NIO

2) Synchronous vs. Asynchronous IO

Java IO's various streams are blocking or synchronous. That means, that when a thread invokes a read() or write(), that thread is blocked until there is some data to read, or the data is fully written. The thread will be in blocked state for this period. This has been cited as a good solid reason for bringing multi-threading in modern languages.

In asynchronous IO, a thread can request that some data be written to a channel, but not wait for it to be fully written. The thread can then go on and do something else in the mean time. Usually these threads spend their idle time on when not blocked in IO calls, is usually performing IO on other channels in the meantime. That is, a single thread can now manage multiple channels of input and output.

Synchronous programs often have to resort to polling, or to the creation of many, many threads, to deal with lots of connections. With asynchronous I/O, you can listen for I/O events on an arbitrary number of channels, without polling and without extra threads.

Java NIO
Important concepts

What's a buffer ?
A Buffer is an object, which holds some data, that is to be written to a channel or that has just been read from a channel

It's main significant difference between the new library and original I/O. In stream-oriented I/O, you wrote data directly to, and read data directly from, Stream objects.

What is a channel?
A Channel is an object from which you can read data and to which you can write data.(similar to I/O streams)

In NIO --- You never write a byte directly to a channel; instead you write to a buffer containing one or more bytes.
Likewise, you don't read a byte directly from a channel; you read from a channel into a buffer, and then get the bytes from the buffer.)

Difference between I/O streams & channels
Kinds of channels
They are bi-directional, streams are uni -directional.
A Channel can be opened for reading, for writing, or for both.

Reading from a file
In I/O we would simply create a FileInputStream and read from that.

In NIO, reading data from a file involves three steps:

1. Get the Channel from FileInputStream or FileChannel's open method
2.Create the Buffer

3.Read data from Channel & write it into the Buffer.
4. Read data from buffer into JA.

Code
```
1.FileInputStream fin = new FileInputStream( "test.txt" );
2. FileChannel fc = fin.getChannel();
3.ByteBuffer buffer = ByteBuffer.allocate( 1024 );
4.fc.read( buffer );
```

Writing to a file

```
1.FileOutputStream fout = new FileOutputStream( "writesomebytes.txt" );
2. FileChannel fc = fout.getChannel();
3. byte[] message="some data".getBytes();
4. ByteBuffer buffer = ByteBuffer.allocate( 1024 );
for (int i=0; i<message.length; ++i) {
buffer.put( message[i] );
}
5. buffer.flip();

6. fc.write( buffer );
```

How to do both (read & write ?)

```
FileInputStream fin = new FileInputStream( infile );
FileOutputStream fout = new FileOutputStream( outfile );

   FileChannel fcin = fin.getChannel();
   FileChannel fcout = fout.getChannel();

   ByteBuffer buffer = ByteBuffer.allocate( 1024 );

   while (true) {
     buffer.clear();

     int r = fcin.read( buffer );

     if (r==-1) {
       break;
     }

     buffer.flip();

     fcout.write( buffer );
   }
  }
```

Unerstanding Buffer state varibales

State variables

Three values can be used to specify the state of a buffer at any given moment in time:

1. position --Current position from where next byte will be read from or written to.

Position can range from : 0 to capacity-1

When you write data into the Buffer, you do so at a certain position. Initially the position is 0. When a byte, long etc. has been written into the Buffer the position is advanced to point to the next cell in the buffer to insert data into.

When you read data from a Buffer you also do so from a given position. When you flip a Buffer from writing mode to reading mode, the position is reset back to 0. As you read data from the Buffer you do so from position, and position is advanced to next position to read.

For data write operation into the buffer
(2 scenarios : 1 Java application is writing the data to a buffer --using put(..) method.

eg : buf.put(bytes)
buf --ByteBuffer
bytes --byte[]

2. Write data from a Channel into a Buffer
From a data sink , data is read into a channel & that u want to write in a buffer.
eg : int bytesRead = fileChannel.read(buf); //read into buffer.
)
2. capacity = total size of the buffer.

3.Limit
In write mode the limit of a Buffer is the limit of how much data you can write into the buffer. In write mode the limit is equal to the capacity of the Buffer.

When flipping the Buffer into read mode, limit means the limit of how much data you can read from the data. So when flipping a Buffer into read mode, limit is set to write position of the write mode. In other words, you can read as many bytes as were written (limit is set to the number of bytes written, which is marked by position).

Buffer's flip() method

The flip() method switches a Buffer from writing mode to reading mode. Calling flip() sets the position back to 0, and sets the limit to where position just was.

In other words, position now marks the reading position, and limit marks how many bytes, chars etc. were written into the buffer - the limit of how many bytes, chars etc. that can be read.

Scenarions for reading Data from a Buffer
There are two ways you can read data from a Buffer.

1. Read data from the buffer into a channel.
//read from buffer into channel.
int bytesWritten = fileChannel.write(buf);

2. Read data from the buffer into java application ,using get() methods.
Here is an example of how you can read data from a buffer into a channel:

Here is an example that reads data from a Buffer using the get() method:

byte aByte = buf.get();

To read file data
1.Create a channel:
The file channel's open() static method is used to create a channel. The method opens a file, returning a FileChannel to access the supplied file.
Path path = Paths.get("readfile.txt");
2.
FileChannel fileChannel = FileChannel.open(path);
3.Create a buffer:
Create a ByteBuffer using the ByteBuffer's allocate() static method. The new buffer's position will be zero, its limit will be its capacity and its elements will be initialized to zero. In this example, the initial capacity is set to 6.

ByteBuffer buffer = ByteBuffer.allocate(6);

4. Read from the channel into the buffer:

FileChannel's read() method reads a sequence of bytes into the given buffer. The method returns the number of bytes read, or -1 if the channel has reached the end-of-stream.

int noOfBytesRead = fileChannel.read(buffer);

position() of channel---
Bytes are read starting at the channel's current file position (initially zero), and then the file position is updated with the number of bytes actually read (in the example, the position will be 6 after initial read). The channel's position() method returns the current position.
The ByteBuffer also has a position() method. Initially this is zero. After the first read, the value is 6. The buffer's flip() method makes a buffer ready for a new sequence of relative get operations: It sets the limit to the current position (in this example, 6) and then sets the position to zero.

buffer.flip();
5. Print the buffer contents:

```
while (buffer.hasRemaining()) {

   System.out.print((char) buffer.get());
```

Example of writing data to a text File using FileChannel

1. String input = "file channel example";
byte [] inputBytes = input.getBytes();

2. Create a buffer:
The ByteBuffer's wrap() static method wraps a byte array into a buffer. The new buffer's capacity and limit will be array.length and its initial position will be zero.

```
ByteBuffer buffer = ByteBuffer.wrap(inputBytes);
```

3. Create the channel:
The FileOutputStream's getChannel() method is used to create a channel. The method returns a file channel that is connected to the underlying file.
FOS fout=new FileOutputStream(fileName);
FileChannel fileChannel = .getChannel();

4. Write the buffer contents into the file channel:

```
int noOfBytesWritten = fileChannel.write(buffer);
```

5. Close resources:
Close the file channel and the file output stream.

```
fileChannel.close();
fout.close();
```
---
java NIO Path/Paths/Files

1. java.nio.file.Path --i/f
It is a part of the Java NIO 2 from Java 7 onwards.

It represents a path in the file system.
A path can point to either a file or a directory.

A path can be absolute or relative. An absolute path contains the full path from the root of the file system down to the file or directory it points to.
A relative path contains the path to the file or directory relative to some other path.

It has nothing to do with the path environment variable.

Similar to the java.io.File class.

How to create a Path Instance
2. To create a Path instance use a static method in the Paths class .(java.nio.file.Paths) : Paths.get().

eg : Path path = Paths.get("c:/tmp/myfile.txt");
OR on linux systems :
Path path = Paths.get("/home/user/myfile.txt");

For relative path
eg : Path projects = Paths.get("d:/data", "projects");

3. The Java NIO Files class -- java.nio.file.Files
Utility class for manipulating files in the file system.

3.1 Files.exists()
The Files.exists() method checks if a given Path exists in the file system.
public static boolean exists(Path path,LinkOption... options)
Returns true if file exists.
eg :
Path path = Paths.get("java/log4j.properties");
boolean exists =Files.exists(path);

3.2 Files.createDirectory()
The Files.createDirectory() method creates a new directory from a Path instance. Files class API
public static Path createDirectory(Path dir,FileAttribute<?>... attrs) throws IOException

eg : Path path = Paths.get("d:/data/my_new_dir");

```
try {
   Path newDir = Files.createDirectory(path);
} catch(FileAlreadyExistsException e){
```

```
   // the directory already exists.
} catch (IOException e) {
   //if the parent directory doesn't exist or no permissions
   e.printStackTrace();
}
```

3.3

Files.copy()
The Files.copy() method copies a file from one path to another.
API
public static Path copy(Path source,Path target,CopyOption... options)
throws IOException

eg :

```
Path sourcePath      = Paths.get(srcFileName);
Path destinationPath = Paths.get(destFileName);

try {
   Files.copy(sourcePath, destinationPath);
} catch(FileAlreadyExistsException e) {
   //destination file already exists
} catch (IOException e) {
   //no permissions / folder doesn't exist
   e.printStackTrace();
}
```

To overwrite Existing Files

```
try {
   Files.copy(sourcePath, destinationPath,
         StandardCopyOption.REPLACE_EXISTING);
} catch(FileAlreadyExistsException e) {
   //destination file already exists
} catch (IOException e) {
   //something else went wrong
   e.printStackTrace();
}
```

3.4
Files.move()

API
public static Path move(Path source,Path target,CopyOption... options)
throws IOException

Move or rename a file to a target file.
Moving a file is the same as renaming it, except moving a file can both move it to a different directory and change its name in the same operation. Similar to File class's renameTo() method.

eg :

```
Path sourcePath      = Paths.get("data/log4j.properties");
Path destinationPath = Paths.get("data/subdir/log4j-new.properties");

try {
   Files.move(sourcePath, destinationPath,
         StandardCopyOption.REPLACE_EXISTING);
} catch (IOException e) {
   //moving file failed.
   e.printStackTrace();
}
```

3.5 To delete a file or directory
API
public static void delete(Path path) throws IOException

The Files.delete() method can delete a file or directory.

```
Path path = Paths.get("my_props/log4j.properties");

try {
   Files.delete(path);
} catch (IOException e) {
   //deleting file failed
   e.printStackTrace();
}
```

Why ?

Java Reflection provides ability to inspect and modify the runtime behavior of application.

What is Java reflection ?

Java API for (java.lang.Class<?> , java.lang.reflect)
1. Inspecting classes,interfaces,fields, methods w/o knowing their names at compile time
2. Instantiating new objects
3. Invoking private methods and get/set field values .
4. Dynamic method invocation

Usage
It's of little use in normal programming but it's the backbone for most of the Java, J2EE frameworks,IDEs,debuggers.

Examples

1. JUnit uses reflection to parse @Test annotation to get the test methods and then invoke it.
2. Spring Framework uses it for dependency injection
3. Web servers use it to forward the request to correct module by parsing their web.xml files and request URI.
4. Eclipse auto completion of method names
5. Struts
6. Hibernate .....

They all use java reflection because all these frameworks have no knowledge and access of user defined classes, interfaces, their methods etc.

Drawbacks

1. Poor Performance Since java reflection resolve the types dynamically, it involves processing like scanning the classpath to find the class to load, causing slow performance.
2.
Security Restrictions Reflection requires runtime permissions that might not be available for system running under security manager. This can cause you application to fail at runtime because of security manager.
3. Security Issues Using reflection we can access part of code that we are not supposed to access, for example we can access private fields of a class and change its value. This can be a serious security threat and cause your application to behave abnormally.
4. High Maintenance Reflection code is hard to understand and debug, also any issues with the code cant be found at compile time because the classes might not be available, making it less flexible and hard to maintain.

Entry Point for all reflection -- java.lang.Class<?> -- represents loaded class information in method area.

Important :

JVM instantiates an immutable instance of java.lang.Class , per every loaded class in method area , which provides methods to examine the runtime properties of the object and create new objects, invoke its method and get/set object fields.

How to get access to loaded class(java.lang.Class) ?
1.Through static variable class
eg : Emp.class ---loaded class info --Class<Emp>
eg : Student.class => Class<Student>
2.Using getClass() method of Object
public final Class<?> getClass()
3.Method of java.lang.Class
public static Class<?> forName(String fullyQualifiedName) throws ClassNotFoundException
code eg : TestReflection.java

For primitive types and arrays, we can use static variable class. Wrapper classes provide another static variable TYPE to get the class.

eg :

```
// Get Class using reflection
Class<?> c = MyClass.class;
c = new MyClass(5).getClass();
try {
	// below method is used most of the times in frameworks like JUnit
	//Spring dependency injection, Tomcat web container
	//Eclipse auto completion of method names, hibernate, Struts2 etc.
	//because MyClass is not available at compile time
	concreteClass = Class.forName("com.app.core.MyClass");
} catch (ClassNotFoundException e) {
	e.printStackTrace();
}
System.out.println(concreteClass.getCanonicalName());
//for primitive types, wrapper classes and arrays
Class<?> booleanClass = boolean.class;
System.out.println(booleanClass.getCanonicalName()); // prints boolean

Class<?> cDouble = Double.TYPE;
System.out.println(cDouble.getCanonicalName()); // prints double

Class<?> cDoubleArray = Class.forName("[D");
System.out.println(cDoubleArray.getCanonicalName()); //prints double[]

Class<?> twoDStringArray = String[][].class;
```

System.out.println(twoDStringArray.getCanonicalName()); // prints java.lang.String[][]

Important Methods of java.lang.Class , for inspection

code eg : Reflection1.java

1. To find fully qualified class name

public String getName()

1.5 To find if its class or interface

public boolean isInterface()

2. To find class modifiers

public int getModifiers();

2.1 java.lang.reflect.Modifier class API

public static boolean isPublic(int m)

public static boolean isAbstract(int m)

puiblic static boolean isFinal(int m)

3. To find super class

public Class<? super T> getSuperclass()

Returns the Class representing the superclass of the entity.

For Object class, an interface, a primitive type, or void, ---rets null.

For an array class --Object class

4. To find interfaces

public Class<?>[] getInterfaces()

Determines the interfaces implemented by the class

5. To get accessible fields

public Field[] getFields() throws SecurityException

Returns an array containing Field objects reflecting all the accessible public fields of the class or interface .

6. To get all fields , including private fields

public Field[] getDeclaredFields() throws SecurityException

Returns an array of Field objects reflecting all the fields(public, protected, default and private fields, but NO inherited fields.)

7. To get all accessible constructors of the class
public Constructor<?>[] getConstructors() throws SecurityException

Returns an array containing Constructor objects reflecting all the public constructors of the class

7.1 To find out constr args
java.lang.reflect.Constructor class API

public Class<?>[] getParameterTypes()

Returns an array of Class objects that represent the formal parameter types, in declaration order.

8. To get all visible methods of the class
public Method[] getMethods() throws SecurityException

Returns an array containing Method objects reflecting all the visible methods(including inherited methods) of the class/interface.

Method class API
8.1 public String getName() --rets method name
8.2 public Class<?> getReturnType() -- rets a class representing Method's ret type.
8.3 public Class<?>[] getParameterTypes()
Returns an array of Class objects that represent the formal parameter types, in declaration order

9. To get all (including private but not inherited) methods of the class.

public Method[] getDeclaredMethods() throws SecurityException
& then use similar API of Method class as mentioned above.

Class inspection part ends here.....

1. How to create an instance of the class using default constr?
Class API
public T newInstance() throws InstantiationException

2. How to create an instance of the class using parameterized constr?
2.1 Get the required constr.
Class API

public Constructor<T> getConstructor(Class<?>... parameterTypes) throws NoSuchMethodException, SecurityException

2.2 Constructor class API
public T newInstance(Object... initargs) throws InstantiationException

3. How to invoke private method of the class dynamically ?
3.1 Get desired method object
Class API
public Method getDeclaredMethod(String name,Class<?>... parameterTypes) throws NoSuchMethodException,SecurityException
3.2 Method class API
public void setAccessible(boolean flag)throws SecurityException

3.3 Invoke the method dynamically
API of Method class
public Object invoke(Object invocationObj,Object... methodArgs)throws IllegalAccessException,IllegalArgumentException, InvocationTargetException
Meaning --
Invokes the underlying method represented by this Method object, on the specified object with the specified parameters.
For static methods --1st arg can be null(its ignored anyway!)