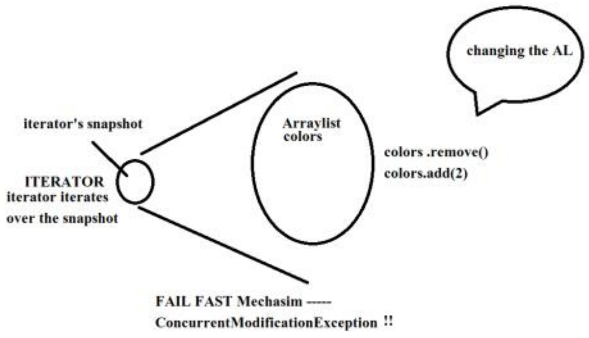
ConcurrentModificationException -----------

Concurrent = at a time
IF a collection is getting modified at the same time when the iterator is iterating over it .



```
public static void main(String[] args) {
        ArrayList<Integer> al = new ArrayList<Integer>();
        al.add(12);
        al.add(122);
        al.add(112);
        al.add(121);
        al.add(212);

        Iterator<Integer> iter = al.iterator();
        while(iter.hasNext())
        {
                System.out.println(iter.next());

                iter.remove(); //this is managed by iterator , so allowed


                //al.remove((Object)12); //DONT dare to change the list , snapshot is wrong then
                //al.add(422);
                //al.add(1,444);
                //al.set(0, 11);   //SIZE of the al is not changing
        }
}

}
```

___________________________________________________________________________

Scopes = Access Specifiers in Java

4

| | | | |
|---|---|---|---|
| 1 | Private = private access specifier is given | 1. Properties<br>2. Methods<br>3. Constructors<br>4. Inner class<br>5. Outer class CANNOT be private | The element can be accessed ONLY within the class where it is declared |
| 2 | Default ---**no access specifier** is given<br><br>**Package scope** is same as default scope | 1. Properties<br>2. Methods<br>3. Constructors<br>4. Inner class<br>5. Outer class | The element can be accessed<br>a. Within the same class<br>b. Within all classes that are in same package |

| | | | |
|---|---|---|---|
| 3 | Protected --- protected access specifier is given | 1.properties<br>2.methods<br>3.Constructor | The element can be accessed<br>a. Within the same class<br>b. Within all classes that are in same package<br>c. Within the subclasses ( in any package ) |
| 4 | Public ----public access specified is given | 1.properties<br>2.methods<br>3.Constructors<br>4.inner classes<br>5.Outer classes | The element can be accessed<br>a. Within the same class<br>b. Within all classes that are in same package<br>c. Within the subclasses ( in any package )<br>d. Within any class |

Public file has file name and class name SAME !!!

Code is divided into 4 main features
Each feature can have one public class
And in the same file that feature related other HELPER class can be present

From main we can access only the PUBLIC FEATURES !!!1 and the features may use the HELPER classes internally , helper classes non public !!

So generally 1 file represents one feature/module/functionality of the Entire!!!!

---

Two ways to create thread
1. extends Thread
2. Implements Runnable

Thread API ----
1. Start
2. Run
3. *currentThread*
4. *Sleep*
5. setName
6. getName
7. Join

Join = BLOCKING CALL --------- it will block a **thread A** that is calling the join method till the threadB on which the join is called does not terminate
threadA is blocked till threadB is not completed

8. setDaemon --- service thread !!--- **GC** is a service thread
   service thread keeps on running till non service threads are running!!!
   When all non service threads TERMINATE the service threads **automatically terminate**
9. setPriority ( Thread.MAX_PRIORITY )
10. getPriority

---

THREAD SAFETY ----
Hashap HashTable
ArrayList Vector
StringBuffer StringBuilder

---

Data Sharing between threads !!!
While the data is shared there MAY be a problem === RACE CONDITION !!
Solution to race condition = MUTUAL EXCLUSION -------
The critical sections should not run at a time !!!
When one critical section is running , the other critical section must wait !!!

CRITICAL SECTION ---------------- code to access the SHARED DATA !!!

JAVA uses a concept of MONITORS in multi threading
a. MONITORS provide mutual exclusion based on LOCKs ( this is same as semaphores )
b. Monitors help inter thread communication using **wait and notify , notifyAll()**

---

CRITICAL SECTION is defined using a keyword **synchronized**

a. Synchronized non static methods ==== lock is "this"
b. Synchronized static methods === lock is "class Class object"
c. Synchronized blocks === lock is the object passed to it

---

1. Shared Data ------------- between threads -----------Account class
2. Two Threads DepositThread, WithdrawThread !!!
3. Main thread !!!

```
public synchronized void deposit(int amount)
        {
                this.balance = this.balance +amount;
        }
```

The lock is "this" object !!!! Both critical sections should have same "this" then only the mutual exclusion will happen !!!!

OVERDOING Synchronized ------disadavantage --------Multi threading effect is compromised !!!! It works like single thread application

---

Nested synchronized blocks --- deadlocks due to synchronized ====extra reading!!!!

---

Producer Consumer -------------

Producer continuously produces item
    Adds it to the **buffer ( bounded buffer === array )**
Consumer continuously consumes item by reading it from the buffer !!!

If array is full ???? Producer must wait
If array is empty ??? Consumer must wait

If consumer consumes from a full array } Consumer NOTIFIES Producer
If Producer adds at least one element to empty array } Producer NOTIFIES CONSUMER

---

HW ---- Implement Producer Consumer ----

```
Class Buffer ---shared data
        Int [] numbers = int[10];
        addToArray
        removeFromArray

Class ProducerThread
        Run
          while(true)
                { generate a number    java.util. Random()   )
                        AddToArray , notifyAll
                If array full wait


        Class ConsumerThread
                Run
                        While(true)
                                {
                                        Read from array  ---- notifyAll
                                        If array empty  wait


Class User
        Main
          create one Buffer pass to Producer and Consumer and start both thread
```

---

HW --- try daemon code , priority code, join code done in class
HW --- try the Account deposit withdraw code done in class!!!

---

99.9 percent !!!!

Practice assignments !!!