**Synchronization:**

**It is the process of letting only one thread at one time to execute some set of statements.**

**Two ways-**

1. **Sync blocks**
2. **Sync methods**

**public** **class** Account{

**int** balance;

**public** Account(**int** balance) {

**super**();

**this**.balance = balance;

}

**public** **void** withDraw(**int** amount) {

**if**(amount > balance) {

System.***out***.println("Insufficient balance");

**return**;

}

balance -= amount;

System.***out***.println("Rs. " + amount + " withdrawn successfully. \n Remaning balance "+ balance);

}

**synchronized** **public** **void** deposit(**int** amount) {

**int** temp = balance;

temp = temp + amount;

balance = temp;

System.***out***.println("Deposit is successful");

}

**public** **void** checkBalance() {

System.***out***.println("Current balance : " + balance);

}

}

**public** **class** Hubby **extends** Thread{

Account a;

**public** Hubby(Account a) {

**super**();

**this**.a = a;

}

@Override

**public** **void** run() {

a.deposit(1000);

System.***out***.println(Thread.*currentThread*().getName());

}

}

**public** **class** Wife **extends** Thread{

Account a;

**public** Wife(Account a) {

**super**();

**this**.a = a;

}

@Override

**public** **void** run() {

a.deposit(1000);

System.***out***.println(Thread.*currentThread*().getName());

}

}

**public** **class** AccHolder {

**public** **static** **void** main(String[] args) {

Account syn = **new** Account(10000);

System.***out***.println(Thread.*currentThread*().getName());

Hubby hubby = **new** Hubby(syn);

hubby.start();

Wife wife = **new** Wife(syn);

wife.start();

syn.checkBalance();

}

}

Locks:

Whenever a thread executes a sync method or a sync block, there is a lock put on the resource by the current executing thread.

Object level lock:

* Non static method when declared as sync,during execution by a thread the lock is put on the object.

Class level lock:

* static method when declared as sync,during execution by a thread the lock is put on the class.

Deadlock situation:

It is a situation which appears only in sync methods or blocks.

In this situation a thread(t1) excuting a sync method needs help of another thread(t2).

But, t2 will not be able to help t1 as t1 will put a lock on the resource.

**public** **class** Godown {

**int** products;

**public** Godown(**int** products) {

**super**();

**this**.products = products;

}

**synchronized** **public** **void** store(**int** noOfItems) {

products = products + noOfItems;

System.***out***.println("Items stored");

**this**.notify(); //write this line later

}

**synchronized** **public** **void** purchase(**int** noOfItems) {

**while**(noOfItems > products) {

System.***out***.println("Out of stock..! Please wait.!");

**try** {

**this**.wait(); //write this line later

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

products = products - noOfItems;

System.***out***.println("Purchase complete");

}

}

**public** **class** Customer **extends** Thread{

Godown godown;

**int** items;

**public** Customer(Godown godown, **int** items) {

**super**();

**this**.godown = godown;

**this**.items = items;

}

@Override

**public** **void** run() {

godown.purchase(items);

}

}

**public** **class** Manufacturer **extends** Thread{

Godown godown;

**int** items;

**public** Manufacturer(Godown godown, **int** items) {

**super**();

**this**.godown = godown;

**this**.items = items;

}

@Override

**public** **void** run() {

godown.store(items);

}

}

**public** **class** Test {

**public** **static** **void** main(String[] args) {

Godown godown = **new** Godown(100);

**new** Customer(godown, 90).start();

**new** Customer(godown, 60).start();

**try** {

Thread.*sleep*(3000);

} **catch** (InterruptedException e) {

e.printStackTrace();

}

**new** Manufacturer(godown, 100).start();

}

}

Diff between sleep & wait.

Sleep🡪 thread class

Wait🡪 object class

Sleep🡪 static method

Wait🡪 non-static method

Sleep🡪 delays w/o releasing lock on object

Wait🡪 delays by releasing the lock on object

Sleep🡪 waiting thread comes back for execution automatically.

Wait🡪 we may need to call notify method.

Thread lifeCyle

Instantiation🡪 Start 🡪 run(wait/sleep) 🡪 destroy