**Concurrent execution**

* When more than one thread need access to one resource at a time that is called concurrent execution, which may creates a conflict.

**Example**

class Account

{

float balance = 20000;

void PassChq(String ThreadName, float amount)

{

if( amount < balance)

{

System.out.println("Processing " + ThreadName);

try

{

Thread.sleep(2000);

}

catch(Exception e)

{

System.out.println("Error : " + e);

}

balance = balance - amount;

System.out.println(ThreadName + " is passed");

}

else

{

System.out.println(ThreadName + " is not passed");

}

}

}

class Process extends Thread

{

String ThreadName;

float amount;

Account obj;

Process(Account obj, String ThreadName, float amount)

{

this.obj = obj;

this.ThreadName = ThreadName;

this.amount = amount;

}

public void run()

{

obj.PassChq(ThreadName, amount);

}

}

class Bank

{

public static void main(String ar[])

{

Account obj = new Account();

Process chq1 = new Process(obj, "cheque1", 15000);

Process chq2 = new Process(obj, "cheque2", 10000);

chq1.start();

chq2.start();

}

}

**Note**

* So the threads need some way to ensure that the resource will be used by only one thread at a time.
* The process by which this is achieved is called synchronization.

**Synchronization**

* ***Synchronized*** keyword is used to implement synchronization.
* If a method or block declared as synchronized then at a time only one thread is allowed to execute that method or block on the given object.
* Synchronized is the modifier applicable only for methods and blocks and we can't apply for classes and variables.

**Example**

class Account

{

float balance = 20000;

synchronized void PassChq(String ThreadName, float amount)

{

if( amount < balance)

{

System.out.println("Processing " + ThreadName);

try

{

Thread.sleep(2000);

}

catch(Exception e)

{

System.out.println("Error : " + e);

}

balance = balance - amount;

System.out.println(ThreadName + " is passed");

}

else

{

System.out.println(ThreadName + " is not passed");

}

}

}

class Process extends Thread

{

String ThreadName;

float amount;

Account obj;

Process(Account obj, String ThreadName, float amount)

{

this.obj = obj;

this.ThreadName = ThreadName;

this.amount = amount;

}

public void run()

{

obj.PassChq(ThreadName, amount);

}

}

class Bank

{

public static void main(String ar[])

{

Account obj = new Account();

Process chq1 = new Process(obj, "cheque1", 15000);

Process chq2 = new Process(obj, "cheque2", 10000);

chq1.start();

chq2.start();

}

}

**Inter-Thread communication**

* Following methods are used to implement IPC
* wait() :
  + Tell the calling thread to give up the lock and go to sleep until some other thread enters the same lock and calls notify().
* notify()
  + Wakes up a thread that called wait().
* notifyAll()
  + Wakes up all the threads that called wait() and one of the threads will be granted access.

**Example**

class Customer

{

int amount=10000;

synchronized void withdraw(int amount)

{

System.out.println("going to withdraw...");

if(this.amount<amount)

{

System.out.println("Less balance; waiting for deposit...");

try

{

wait();

}

catch(Exception e)

{

}

}

this.amount = this.amount - amount;

System.out.println("withdraw completed...");

}

synchronized void deposit(int amount)

{

System.out.println("going to deposit...");

this.amount+=amount;

System.out.println("deposit completed... ");

notify();

}

}

class MyThread1 extends Thread

{

Customer c;

MyThread1(Customer c)

{

this.c = c;

}

public void run()

{

c.withdraw(15000);

}

}

class MyThread2 extends Thread

{

Customer c;

MyThread2(Customer c)

{

this.c = c;

}

public void run()

{

c.deposit(10000);

}

}

class Test

{

public static void main(String args[])

{

Customer c=new Customer();

MyThread1 t1 = new MyThread1(c);

MyThread2 t2 = new MyThread2(c);

t1.start();

t2.start();

}

}