Busy Spin - What is busy spin? Consumer Producer problem with busy spin and solution to busy spin in java.

What is busy spin?

When one thread loops continuously waiting for another thread to signal.

Performance point of view - Busy spin is

very bad from performance point of view, because one thread keeps on looping continuously (and consumes CPU) waiting for another thread to signal.

Solution to busy spin -

We must use sleep() or wait() and notify() method. Using wait() is better option.

Why using wait() and notify() is much better option to solve busy spin?

Because in case when we use sleep() method, thread will wake up again and again after specified sleep time until boolean variable is true. But, in case of wait() thread will wake up only when when notified by calling notify() or notifyAll(), hence end up consuming CPU in best possible manner

Program - Consumer Producer problem with ***busy spin*** *>*

Note: In below program, Producer will allow consumer to consume only when 10 products have been produced (i.e. when production is over)

Consumer thread continuously execute (**busy spin**) in while loop (till `productionInProcess` is true) and wait for producer thread to get over. Once producer thread has ended it will make boolean variable `productionInProcess` false and **busy spin** will be over.

```
while(this.prod.productionInProcess){
  System.out.println("BUSY SPIN - Consumer waiting for production to get over");
}
```

*But how **performance** is **impacted** in this program?*

Performance is impacted because consumer thread is continuously executing and unnecessarily consuming CPU. It would have been better if consumer would have called wait() and waited until being notified by producer. **So, consumer thread consumes CPU unnecessarily and didn't allow producer to utilize complete CPU better performance.**

```
import java.util.ArrayList;


/* Producer will allow consumer to
 * consume only when 10 products have been produced (i.e. when production is over).
 */
class Producer implements Runnable{

    boolean productionInProcess;
    ArrayList<Integer> list;

    Producer(){
           productionInProcess=true;  //initially Producer will be producing, so
                                      //make this productionInProcess true.
           list=new ArrayList<Integer>();
    }

    @Override
    public void run(){

           for(int i=1;i<=10;i++){ //Producer will produce 10 products
                  list.add(i);
                  System.out.println("Producer is still Producing, Produced : "+i);

                  try{
                        Thread.sleep(500);
                  }catch(InterruptedException e){e.printStackTrace();}

           }
           productionInProcess=false; // Once production is over, make
                                      //this productionInProcess false.
                                      //Production is over, consumer can consume.

    }

}

class Consumer extends Thread{
    Producer prod;

    Consumer(Producer obj){
     prod=obj;
    }

    public void run(){
           /*
              * consumer will continuously loop until productionInProcess is true.
            */
```

```
            while(this.prod.productionInProcess){ //BUSY SPIN
                 System.out.println("BUSY SPIN - Consumer waiting for production to get over.");
              }


            /*productionInProcess is false means production is over,
                consumer will start consuming. */
            System.out.println("Production is over, consumer can consume.");
            int productSize=this.prod.list.size();
            for(int i=0;i<productSize;i++)
                   System.out.println("Consumed : "+ this.prod.list.remove(0) +" ");

    }

}

/** Copyright (c), AnkitMittal JavaMadeSoEasy.com */
public class BusySpin{
    public static void main(String args[]){

            Producer prod=new Producer();
            Consumer cons=new Consumer(prod);

            Thread prodThread=new Thread(prod,"prodThread");
            Thread consThread=new Thread(cons,"consThread");

            prodThread.start();     //start producer thread.
            consThread.start();     //start consumer thread.


    }

}
```

If we execute this program we will note in output that **"BUSY SPIN - Consumer waiting for production to get over."** is printed several times.

Solution *to busy spin using wait() and notify() methods-*

In program consumer thread will start() and wait by calling wait() method till producer is producing. Once production is over, producer thread will call notify() method, which will notify consumer thread and consumer will start consuming.

*But how **performance** is **improved** in this program?*

Performance is improved because consumer thread called wait() method. Once production is over, producer thread will call notify() method, which will notify consumer thread and consumer will start consuming. **So, consumer thread will not consume CPU unnecessarily and allow producer to utilize complete CPU for better performance.**

```
import java.util.ArrayList;
```

```
/* Producer is producing, Producer will allow consumer to
 * consume only when 10 products have been produced (i.e. when production is over).
 */
class Producer implements Runnable{

    ArrayList<Integer> sharedQueue;

    Producer(){
           sharedQueue=new ArrayList<Integer>();
    }

    @Override
    public void run(){

           synchronized (this) {
                  for(int i=1;i<=10;i++){ //Producer will produce 10 products
                        sharedQueue.add(i);
                        System.out.println("Producer is still Producing, Produced : "+i);

                        try{
                               Thread.sleep(1000);
                        }catch(InterruptedException e){e.printStackTrace();}

                  }
                  System.out.println("Production is over, consumer can consume.");
                  this.notify();  //Production is over, notify consumer thread
                                  //so that consumer can consume.
           }
    }

}

class Consumer extends Thread{
    Producer prod;

    Consumer(Producer obj){
     prod=obj;
    }

    public void run(){
           /*
            * consumer will wait till producer is producing.
            */
           synchronized (this.prod) {  //NO BUSY SPIN

                  System.out.println("NO BUSY SPIN, Consumer waiting for production to get over.");
                     try{
                         this.prod.wait();
                        }catch(InterruptedException e){e.printStackTrace();}

           }


           /*production is over, consumer will start consuming.*/
           int productSize=this.prod.sharedQueue.size();
           for(int i=0;i<productSize;i++)
                  System.out.println("Consumed : "+ this.prod.sharedQueue.remove(0) +" ");

    }
```

```
}

/** Copyright (c), AnkitMittal JavaMadeSoEasy.com */
public class ProducerConsumerWithWaitNotify {
    public static void main(String args[]) throws InterruptedException{

            Producer prod=new Producer();
            Consumer cons=new Consumer(prod);

            Thread prodThread=new Thread(prod,"prodThread");
            Thread consThread=new Thread(cons,"consThread");

            consThread.start();  //start consumer thread.
            Thread.sleep(100);  //This minor delay will ensure that consumer
                                  //thread starts before producer thread.
            prodThread.start();     //start producer thread.


    }

}

/*OUTPUT

NO BUSY SPIN, Consumer waiting for production to get over.
Producer is still Producing, Produced : 1
Producer is still Producing, Produced : 2
Producer is still Producing, Produced : 3
Producer is still Producing, Produced : 4
Producer is still Producing, Produced : 5
Producer is still Producing, Produced : 6
Producer is still Producing, Produced : 7
Producer is still Producing, Produced : 8
Producer is still Producing, Produced : 9
Producer is still Producing, Produced : 10
Production is over, consumer can consume.
Consumed : 1
Consumed : 2
Consumed : 3
Consumed : 4
Consumed : 5
Consumed : 6
Consumed : 7
Consumed : 8
Consumed : 9
Consumed : 10

*/
```

If we execute this program we will note in output that **"NO BUSY SPIN - Consumer waiting for production to get over."** is printed only one time.