# Leksioni 7
# Threads, Synchronization

## 7.1 Cfarë janë Threads në Java?

Një thread është një:

1. Lehtësi për të lejuar aktivitete të shumta brenda një procesi të vetëm
2. Proces i lehtë
3. Një thread (fije) është një seri deklaratash të ekzekutuara
4. Çdo thread ka programin e vet, stokun dhe ndryshoret lokale
5. Një thread është një sekuencë e ndërfutur e thirrjeve të metodës
6. Ndan memorjen, skedarët dhe gjendjen për çdo proces

## 7.2 Pse është nevoja e një threads apo pse ne përdorim Threads?

1. Për të kryer përpunim asinkron ose begraundin
2. Rrit përgjegjësinë e aplikacioneve GUI
3. Përfitoni nga sistemet multiprocesoriale
4. Thjeshtoni logjikën e programit kur ka shumë njësi të pavarura

## 7.3 Çfarë ndodh kur thirret një thread?

Kur thirret një thread, do të ketë dy rrugë të ekzekutimit. Një rrugë do të ekzekutojë fillin dhe rruga tjetër do të ndjekë deklaratën pas thirrjes së thread. Do të ketë një grumbull të veçantë dhe hapësirë kujtese për secilin thread.

Faktori i rriskut
Kërkohet një bashkërendim i duhur midis threads që hyjnë në variablat e zakonshëm [përdorimi i sinkronizuar dhe i paqëndrueshëm] për shikimin e qëndrueshmërisë së të dhënave. Përdorimi i tepërt i threads në java mund të jetë i rrezikshëm për performancën e programit dhe qëndrueshmërinë e tij.

## 7.4 Threads në Java

Çdo programues në java krijon të paktën një thread [main () thread]. Threads shtesë krijohen përmes konstruktorit të Thread-it ose përmes klasave të menjëhershme që zgjasin klasën e Thread-it.

### *7.4.1 Krijimi i threads në Java*

Zbatimi i thread në java mund të arrihet në dy mënyra:

a) Zgjatja e klasës java.lang.Thread
b) Zbatimi i ndërfaqes java.lang.Runnable

Shënim: Thread dhe Runnable janë në dispozicion në paketën java.lang. *

a) Duke zgjatur klasën e fijeve

Klasa duhet të zgjasë klasën Java Thread.
Klasa duhet të anashkalojë metodën run ().
Funksionaliteti që pritet nga Thread që do të ekzekutohet shkruhet në metodën run ().
*void start ():* Krijon një thread të re dhe e bën atë të funksionueshme.
*void run ():* Thread i ri fillon jetën e tij brenda kësaj metode.

Shembull:

```
public class MyThread extends Thread {
   public void run(){
    System.out.println("thread is running...");
  }
   public static void main(String[] args) {
     MyThread obj = new MyThread();
     obj.start();
}
```

b) Duke zbatuar ndërfaqen Runnable

Klasa duhet të implementojë ndërfaqen Runnable
Klasa duhet të implementojë metodën *run ()* në ndërfaqen Runnable
Funksionaliteti që pritet nga Thread që do të ekzekutohet vendoset në metodën *run ()*

Shembull:

```
public class MyThread implements Runnable {
   public void run(){
     System.out.println("thread is running..");
   }
   public static void main(String[] args) {
     Thread t = new Thread(new MyThread());
     t.start();
}
```

### 7.5 Zgjat klasën e threads vs Zbaton ndërfaqen e ekzekutueshme?

Zgjatja e klasës Thread do ta bëjë klasën tuaj të paaftë të zgjasë klasa të tjera, për shkak të veçorisë së trashëgimisë së vetme në JAVA. Sidoqoftë, kjo do t'ju japë një strukturë më të thjeshtë të kodit. Nëse zbatoni Runnable, ju mund të fitoni një dizajn dhe qëndrueshmëri më të mirë të orientuar drejt objektit dhe gjithashtu të shmangni problemet e vetme të trashëgimisë. Nëse thjesht doni të arrini funksionalitetin themelor të një fijeje, thjesht mund të implementoni ndërfaqen Runnable dhe të mbivendosni metodën run (). Por nëse doni të bëni diçka serioze me objektin e threads pasi ajo ka metoda të tjera si suspend (), resume (), ..etj të cilat nuk janë të disponueshme në ndërfaqen e ekzekutueshme, atëherë mund të preferoni të zgjeroni klasën e Thread. Cikli jetësor i threads në java, mbaron për shkak të arsyeve të mëposhtme: 1. Fije mbaron kur metoda run () përfundon ekzekutimin e saj. 2. Kur thread afishon një përjashtim ose gabim që nuk po kapet në program. programi në java përfundon ose mbaron. Një tjetër thread është dhe metoda stop (). Sinkronizimi i Threads në shumë raste ndodh atëhere kur, threads që funksionojnë njëkohësisht ndajnë të dhëna dhe dy thread përpiqen të bëjnë veprime në të njëjtat variabla në të njëjtën kohë. Kjo shpesh rezulton në të dhëna të korruptuara pasi dy thread përpiqen të veprojnë në të njëjtat të dhëna. Një zgjidhje e thjeshtë është të sigurojmë një lloj bllokimi primitiv. Vetëm një fije mund të marrë një bllokim të veçantë në çdo kohë të veçantë. Kjo mund të arrihet duke përdorur një fjalë kyçe "të sinkronizuar". Duke përdorur sinkronizimin vetëm një fije mund të ketë qasje në metodë në të njëjtën kohë dhe një thirrje e dytë do të bllokohet derisa thirrja e parë të kthehet ose të presë të thirret brenda metodës së sinkronizuar.

### 7.5.1 Bllokim

Kurdoherë që ka procese të shumta që pretendojnë për qasje ekskluzive në procese të shumta, ekziston mundësia e bllokimit. Një grup procesesh ose fijesh (threads) thuhet se janë të bllokuara kur secili pret një veprim që vetëm një nga të tjerët mund të kryejë.
Në mënyrë që të shmanget bllokimi, duhet të sigurohet që kur fitoni shumë procese (veprime), gjithmonë të fitoni proceset në të njëjtin rend në të gjitha fijet.

## 7.6 Udhëzime për sinkronizimin

1. Mbani blloqet të shkurtra. Blloqet e sinkronizuara duhet të jenë të shkurtra - sa më të shkurtër të jetë e mundur ndërkohë që kryeni një veprom me të dhënat përkatëse.
2. Mos bllokoni. Mos thirrni kurrë një metodë që mund të bllokojë, të tilla si InputStream.read (), brenda një blloku ose metode të sinkronizuar.
3. Mos thërrisni metoda në objekte të tjera ndërsa kryeni një veprim. Kjo mund të tingëllojë ekstreme, por eliminon burimin më të zakonshëm të bllokimit.

## 7.7 Sinkronizimi në Java

Programet me shumë threads shpesh mund të vijnë në një situatë ku shumë threads përpiqen të kenë të njëjtat burime dhe më në fund të prodhojnë rezultate të gabuara dhe të paparashikuara.
Pra, duhet të sigurohet me anë të disa metodave të sinkronizimit që vetëm një fije mund të ketë qasje në burim në një moment të caktuar kohe.
Java ofron një mënyrë për krijimin e threads dhe sinkronizimin e detyrës së tyre duke përdorur blloqe të sinkronizuara. Blloqet e sinkronizuara në Java shënohen me fjalën kyçe të sinkronizuar. Një bllok i sinkronizuar në Java është i sinkronizuar në ndonjë objekt. Të gjithë blloqet e sinkronizuara në të njëjtin objekt mund të kenë vetëm një thread ekzekutuese brenda tyre njëkohësisht. Të gjitha threads e tjera që përpiqen të hyjnë në bllokun e sinkronizuar bllokohen derisa thread brenda bllokut të sinkronizuar të dalë nga blloku.

Më poshtë është forma e përgjithshme e një blloku të sinkronizuar:

```
// Only one thread can execute at a time.
// sync_object is a reference to an object
// whose lock associates with the monitor.
// The code is said to be synchronized on
// the monitor object
synchronized(sync_object)
{
   // Access shared variables and other
   // shared resources
}
```

Ky sinkronizim zbatohet në Java me një koncept të quajtur monitorë. Vetëm një fije mund të posedojë një monitor në një kohë të caktuar. Kur një fije fiton një veprim, thuhet se ka hyrë në monitor. Të gjitha fijet e tjera që përpiqen të hyjnë në monitorin e bllokuar të cilat do të pezullohen derisa fija e parë të dalë nga monitori.

Më poshtë është një shembull i multi threading me sinkronizim

```
// A Java program to demonstrate working of
// synchronized.
```

```
import java.io.*;
import java.util.*;

// A Class used to send a message
class Sender
{
    public void send(String msg)
    {
        System.out.println("Sending\t"  + msg );
        try
        {
            Thread.sleep(1000);
        }
        catch (Exception e)
        {
            System.out.println("Thread  interrupted.");
        }
        System.out.println("\n" + msg + "Sent");
    }
}

// Class for send a message using Threads
class ThreadedSend extends Thread
{
    private String msg;
    Sender  sender;

    // Recieves a message object and a string
    // message to be sent
    ThreadedSend(String m,  Sender obj)
    {
        msg = m;
        sender = obj;
    }

    public void run()
    {
        // Only one thread can send a message
        // at a time.
        synchronized(sender)
        {
            // synchronizing the snd object
            sender.send(msg);
        }
    }
}

// Driver class
class SyncDemo
{
    public static void main(String args[])
    {
```

```
        Sender snd = new Sender();
        ThreadedSend S1 =
            new ThreadedSend( " Hi " , snd );
        ThreadedSend S2 =
            new ThreadedSend( " Bye " , snd );

        // Start two threads of ThreadedSend type
        S1.start();
        S2.start();

        // wait for threads to end
        try
        {
            S1.join();
            S2.join();
        }
        catch(Exception e)
        {
            System.out.println("Interrupted");
        }
    }
}
Output:

Sending Hi

 Hi Sent

Sending  Bye

 Bye Sent
```

Dalja është e njëjtë çdo herë që ekzekutojmë programin.

Në shembullin e mësipërm, kemi zgjedhur të sinkronizojmë objektin Sender brenda metodës run () të klasës ThreadedSend. Përndryshe, ne mund të përcaktojmë të gjithë bllokun e dërgimit () si të sinkronizuar dhe do të prodhonte të njëjtin rezultat. Atëherë nuk duhet të sinkronizojmë objektin Message brenda metodës run () në klasën ThreadedSend.

```
// An alternate implementation to demonstrate
// that we can use synchronized with method also.
class Sender
{
    public synchronized void send(String msg)
    {
        System.out.println("Sending\t" + msg );
        try
        {
            Thread.sleep(1000);
        }
        catch (Exception e)
```

```
        {
            System.out.println("Thread interrupted.");
        }
        System.out.println("\n" + msg + "Sent");
    }
}
```

Jo gjithmonë duhet të sinkronizojmë një metodë të tërë. Ndonjëherë është e preferueshme të sinkronizoni vetëm një pjesë të një metode. Blloqet e sinkronizuara Java brenda metodave e bëjnë të mundur këtë.

```
// One more alternate implementation to demonstrate
// that synchronized can be used with only a part of
// method
class Sender
{
    public void send(String msg)
    {
        synchronized(this)
        {
            System.out.println("Sending\t" + msg );
            try
            {
                Thread.sleep(1000);
            }
            catch (Exception e)
            {
                System.out.println("Thread interrupted.");
            }
            System.out.println("\n" + msg + "Sent");
        }
    }
}
```

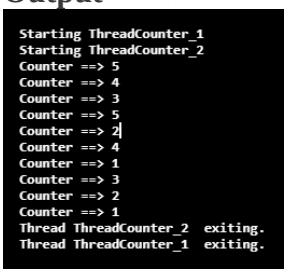
### 7.8 Llojet e Sinkronizimit

Ekzistojnë 2 lloje të sinkronizimit siç shpjegohet më poshtë: # 1) Sinkronizimi i procesit. Sinkronizimi i procesit përfshin procese ose fije të shumëfishta që ekzekutohen njëkohësisht. Në fund të fundit ata arrijnë një gjendje ku këto procese ose fije angazhohen për një sekuencë specifike veprimesh. # 2) Sinkronizimi i Thread. Në Sinkronizimi i Thread, më shumë se një fije po përpiqet të hyjë në një hapësirë të përbashkët. Threads sinkronizohen në një mënyrë të tillë që hapësira e përbashkët të arrihet vetëm nga një thread në të njëjtën kohë. Sinkronizimi i Procesit është jashtë fushës së këtij tutoriali. Prandaj këtu do të diskutojmë vetëm për Sinkronizimin e Thread-it. Në Java, ne mund të përdorim fjalën kyçe të sinkronizuar me: Një bllok kodi Një metodë Llojet e mësipërme janë lloje përjashtuese reciprokisht të sinkronizimit të threads. Përjashtimi i ndërsjellë i mban threads që kanë të dhëna të përbashkëta të mos ndërhyjnë me njëri-tjetrin. Lloji tjetër i sinkronizimit të threads është "Komunikimi InterThread" që bazohet në bashkëpunimin midis tyre. Para se të vazhdojmë me sinkronizimin e blloqeve dhe metodave, le të zbatojmë një program Java për të demonstruar sjelljen e threadsfijeve kur nuk ka sinkronizim.

*Multi-Threading Without Synchronization*
*The following Java program has multiple threads that are not synchronized.*

```
class PrintCount {
   //method to print the thread counter
    public void printcounter() {
      try {
         for(int i = 5; i > 0; i--) {
            System.out.println("Counter ==> "  + i );
         }
      } catch (Exception e) {
         System.out.println("Thread  interrupted.");
      }
   }
}
//thread class
class ThreadCounter extends Thread {
   private Thread t;
   private String threadName;
   PrintCount  PD;
   //class constructor for initialization
   ThreadCounter( String name,  PrintCount pd) {
      threadName = name;
      PD = pd;
   }
   //run method for thread
   public void run() {
      PD.printcounter();
      System.out.println("Thread " +  threadName + " exiting.");
   }
    //start method for thread
   public void start () {
      System.out.println("Starting " +  threadName );
      if (t == null) {
         t = new Thread (this, threadName);
         t.start ();
      }
   }
}
public class Main {
   public static void main(String args[]) {
      PrintCount PD = new PrintCount();
        //create two instances of thread class
      ThreadCounter T1 = new ThreadCounter( "ThreadCounter_1 ", PD );
      ThreadCounter T2 = new ThreadCounter( "ThreadCounter_2 ", PD );
      //start both the threads
      T1.start();
      T2.start();      // wait for threads to end
         try {
    T1.join();
         T2.join();
      } catch ( Exception e) {
         System.out.println("Interrupted"); }
}
```

**Output**




Nga dalja, mund të shohim që pasi threads nuk janë të sinkronizuara, dalja është jo konsistente. Të dy thread fillojnë dhe pastaj ata shfaqin counter njëra pas tjetrës. Të dyja afishohen në fund.

Nga programi i dhënë, thread i parë duhet të ketë dalë pasi të shfaqen vlerat e numëruesit, dhe pastaj thread i dytë duhet të ketë filluar të shfaqë vlerat e numëruesit.

Tani le të shkojmë për sinkronizim dhe të fillojmë me sinkronizimin e bllokut të kodit.
*Blloku i Kodit të Sinkronizuar*
Një bllok i sinkronizuar përdoret për të sinkronizuar një bllok kodi. Ky bllok zakonisht përbëhet nga disa rreshta. Një bllok i sinkronizuar përdoret kur nuk duam që një metodë e tërë të sinkronizohet. Për shembull, ne kemi një metodë me të themi 75 rreshta të kodit. Nga kjo kërkohet që vetëm 10 rreshta të kodit të ekzekutohen nga një thread në të njëjtën kohë. Në këtë rast, nëse e bëjmë të gjithë metodën të sinkronizuar, atëherë ajo do të jetë një barrë për sistemin. Në situata të tilla, ne shkojmë për blloqe të sinkronizuara. Shtrirja e metodës së sinkronizuar është gjithmonë më e vogël se ajo e një metode të sinkronizuar. Një metodë e sinkronizuar bllokon një objekt të një burimi të përbashkët që do të përdoret nga shumë thread.

Sintaksa e përgjithshme e një blloku të sinkronizuar është siç tregohet më poshtë:

```
i sinkronizuar (objekt_ kyçje) {
        // deklaratat e kodit të sinkronizuar
    }
```

Këtu "bllok_objekti" është një shprehje referuese e objektit mbi të cilën do të veprohet. Pra, sa herë që një thread dëshiron të ketë qasje në deklaratat e sinkronizuara brenda bllokut për ekzekutim, atëherë duhet të fitojë bllokimin në monitorin 'lock_object'.
Siç është diskutuar tashmë, fjala kyçe e sinkronizuar siguron që vetëm një thread mund të fitojë një bllokim në të njëjtën kohë dhe të gjitha fijet e tjera duhet të presin derisa thread që mban bllokimin të përfundojë dhe të lëshojë bllokimin.

*Shënim*: Një "NullPointerException" hidhet nëse objekti i bllokimit të përdorur është Null.
Nëse një fije fle ndërsa mban ende bllokuesin, atëherë bllokimi nuk lirohet. Temat e tjera nuk do të jenë në gjendje të hyjnë në objektin e përbashkët gjatë kësaj kohe gjumi.
Tani do të paraqesim shembullin e mësipërm që tashmë ishte zbatuar me ndryshime të vogla. Në programin e mëparshëm, ne nuk e sinkronizuam kodin. Tani do të përdorim bllokun e sinkronizuar dhe do të krahasojmë prodhimin.

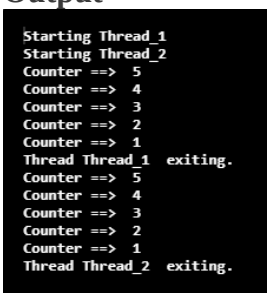
### 7.9 Multi-Threading Me Sinkronizimi

Në programin Java më poshtë, ne përdorim një bllok të sinkronizuar. Në metodën e ekzekutimit, ne sinkronizojmë kodin e linjave që shtypin counter për secilën thread.

```
class PrintCount {
   //print thread counter
   public void printCounter() {
      try {
         for(int i = 5; i > 0; i--) {
            System.out.println("Counter ==>  "  + i );
         }
      } catch (Exception e) {
         System.out.println("Thread  interrupted.");
      }
   }
}
//thread class
class ThreadCounter extends Thread {
   private Thread t;
   private String threadName;
   PrintCount  PD;
   //class constructor for initialization
   ThreadCounter( String name,  PrintCount pd) {
      threadName = name;
      PD = pd;
   }
   //run () method for thread with synchronized block
   public void run() {
      synchronized(PD) {
         PD.printCounter();
      }
      System.out.println("Thread " +  threadName + " exiting.");
   }
    //start () method for thread
   public void start () {
      System.out.println("Starting " +  threadName );
      if (t == null) {
         t = new Thread (this, threadName);
         t.start ();
      }
   }
}
public class Main {
   public static void main(String args[]) {
      PrintCount PD = new PrintCount();
      //create thread instances
      ThreadCounter T1 = new ThreadCounter( "Thread_1 ", PD );
      ThreadCounter T2 = new ThreadCounter( "Thread_2 ", PD );
      //start both the threads
      T1.start();
      T2.start();      // wait for threads to end
      try {
         T1.join();
         T2.join();
      } catch ( Exception e) {
         System.out.println("Interrupted");
```

```
        }
    }
}
```

**Output**




Tani rezultati i këtij programi duke përdorur bllokun e sinkronizuar është mjaft i qëndrueshëm. Siç pritej, të dy threads fillojnë të ekzekutohen. Fija e parë përfundoi duke shfaqur vlerat dhe daljet e numëruesit. Pastaj filli i dytë tregon vlerat e numëruesit dhe del.

***Metoda e Sinkronizuar***

Le të diskutojmë metodën e sinkronizuar në këtë moment. Më parë kemi parë që mund të deklarojmë një bllok të vogël që përbëhet nga më pak linja kodi si një bllok të sinkronizuar. Nëse duam që i gjithë funksioni të sinkronizohet, atëherë mund të deklarojmë një metodë si të sinkronizuar. Kur një metodë bëhet e sinkronizuar, atëherë vetëm një fije do të jetë në gjendje të bëjë një thirrje metode në të njëjtën kohë.

```
Sintaksa e përgjithshme për të shkruar një metodë të sinkronizuar është:
<access_modifier> synchronized method_name (parameters){
        //synchronized code
    }
```

Ashtu si një bllok i sinkronizuar, në rastin e një metode të sinkronizuar, ne kemi nevojë për një objekt_ celës që do të përdoret nga fijet që hyjnë në metodën e sinkronizuar. Për metodën e sinkronizuar, objekti i kyçjes mund të jetë një nga më poshtë: Nëse metoda e sinkronizuar është statike, atëherë objekti i bllokimit jepet nga objekti '.class'. Për një metodë jo-statike, objekti i bllokimit jepet nga objekti aktual d.m.th. objekti 'ky'. Një tipar i veçantë i fjalës kyçe të sinkronizuar është se ajo është ri-hyrëse. Kjo do të thotë që një metodë e sinkronizuar mund të thërrasë një metodë tjetër të sinkronizuar me të njëjtën bllokim. Pra, një thread që mban bllokimin mund të hyjë në një metodë tjetër të sinkronizuar pa pasur nevojë të fitojë një bllokim tjetër. Metoda e Sinkronizuar demonstrohet duke përdorur shembullin e mëposhtëm.

```
class NumberClass {
    //synchronized method to print squares of numbers
    synchronized void printSquares(int n) throws InterruptedException
    {
        //iterate from 1 to given number and print the squares at each iteration
        for (int i = 1; i <= n; i++)
        {
            System.out.println(Thread.currentThread().getName() + " :: "+  i*i);
            Thread.sleep(500);
        }
    }
}
```

```
public class Main {
    public static void main(String args[])   {
        final NumberClass number = new NumberClass();
       //create thread
        Runnable thread = new Runnable()   {
            public void run()   {
                try {
                    number.printSquares(3);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        };
        //start thread instance
        new Thread(thread, "Thread One").start();
        new Thread(thread, "Thread Two").start();
    }
}
```

**Output**

```
Thread One :: 1

Thread One :: 4

Thread One :: 9

Thread Two :: 1

Thread Two :: 4

Thread Two :: 9
```

Në programin e mësipërm, ne kemi përdorur një metodë të sinkronizuar për të shtypur katrorët e një numri. Kufiri i sipërm i numrit i kalohet metodës si argument. Pastaj duke filluar nga 1, sheshet e secilit numër shtypen derisa të arrihet kufiri i sipërm. Në funksionin kryesor, krijohet instanca e thread. Çdo instancë thread i kalon një numër për të shtypur vlerat.
Siç u përmend më lart, kur një metodë që do të sinkronizohet është statike, atëherë objekti bllokues përfshihet në klasë dhe jo objekti. Kjo do të thotë që ne do të kyçemi në klasë dhe jo në objekt. Ky quhet sinkronizimi statik.

**Shembull**

```
class Table{
  //synchronized static method to print squares of numbers
  synchronized static void printTable(int n){
   for(int i=1;i<=10;i++){
     System.out.print(n*i + " ");
     try{
       Thread.sleep(400);
     }catch(Exception e){}
   }
   System.out.println();
 }
}

//thread class Thread_One
```

```
class Thread_One extends Thread{
    public void run(){
        Table.printTable(2);
    }
}

//thread class Thread_Two
class Thread_Two extends Thread{
    public void run(){
        Table.printTable(5);
    }
}

public class Main{
public static void main(String t[]){
        //create instances of Thread_One and Thread_Two
        Thread_One t1=new Thread_One ();
        Thread_Two t2=new Thread_Two ();
        //start each thread instance
        t1.start();
        t2.start();
    }
}
```

**Output**

```
2 4 6 8 10 12 14 16 18 20

5 10 15 20 25 30 35 40 45 50
```

Në programin e mësipërm, ne shtypim tabelën të shumëzimit të numrave. Çdo numër, tabela e të cilit do të shtypet është një shembull i një klase të klasës së ndryshme. Kështu që ne shtypim tabela të shumëzimit të 2 & 5, kështu që kemi dy klasa 'thread_one dhe thread_two për të shtypur përkatësisht tabelat 2 dhe 5.

Për ta përmbledhur, fjala kyçe e sinkronizuar Java kryen funksionet e mëposhtme:

Fjala kyçe e sinkronizuar në Java garanton qasje reciproke ekskluzive në burimet e përbashkëta duke siguruar një mekanizëm mbylljeje. Kyçja parandalon edhe kushtet e garës. Duke përdorur fjalën kyçe të sinkronizuar, ne parandalojmë gabimet e njëkohshme të programimit në kod. Kur një metodë ose bllok deklarohet i sinkronizuar, atëherë një thread ka nevojë për një celës ekskluziv për të hyrë në metodën ose bllokun e sinkronizuar. Pas kryerjes së veprimeve të nevojshme, thread lëshon bllokimin dhe do të shpëtojë operacionin e shkrimit. Në këtë mënyrë do të eliminojë gabimet e kujtesës në lidhje me mospërputhjen.