TEHNOLOGII JAVA
PENTRU DEZVOLTAREA APLICAȚIILOR
LUCRARE DE LABORATOR 18

Fire de execuție în Java. Aplicatii

I. SCOPUL LUCRĂRII

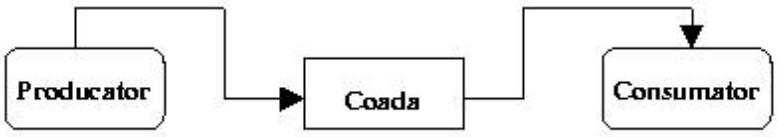
Lucrarea de față are rolul de a prezenta şi familiariza studentul cu modul de construire şi utilizare a firelor de execuție în Java si de a rezolva diverse probleme folosind firele de executie.

La sfârşitul acestei lucrări, studentul va avea posibilitatea să scrie programe Java în care să utilizeze noțiunile învățate.

II. NOȚIUNI TEORETICE

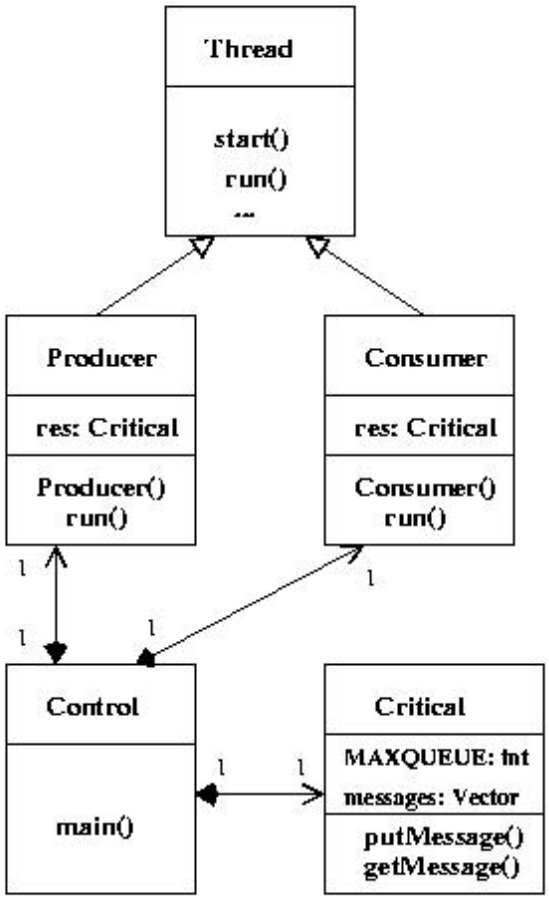
1. Problema producatorului si a consumatorului

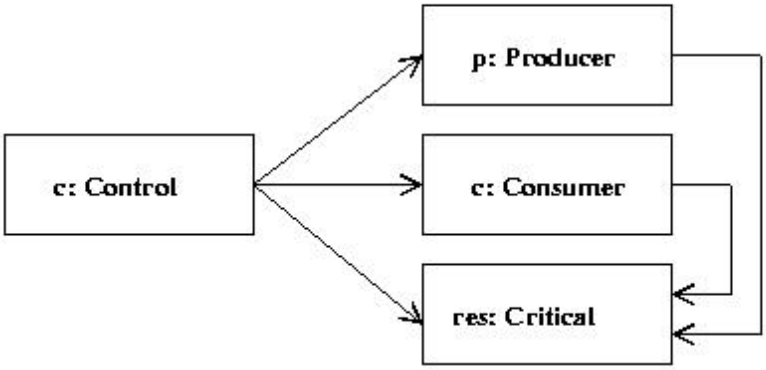
Specificatia problemei:



Producatorul produce mesaje care contin data si ora emiterii mesajului. Produsele sunt stocate intr-o coada. Consumatorul "consuma" mesajele din coada afisandu-le la terminal. Coada reprezinta o resursa critica, la care accesul trebuie sincronizat. Producatorul va bloca accesul la coada in momentul depunerii "produsului" in coada, si va elibera accesul dupa ce depunerea s-a terminat. Daca coada se umple, producatorul intra in asteptare. Daca coada se goleste, atunci consumatorul va intra in asteptare.

Descrierea claselor:





Sursa Java:

```
import java.util.*;
class Critical{
   static final int MAXQUEUE = 5;
   private Vector messages = new Vector();
   public synchronized void putMessage() throws InterruptedException{
       while( messages.size() == MAXQUEUE )
          wait();
       String message = new String( new java.util.Date().toString() );
       System.out.println("Producator: "+message);
       messages.addElement( message );
       notify();
   }
   public synchronized String getMessage() throws InterruptedException{
       notify();
       while( messages.size() == 0 )
          wait();
       String message = (String) messages.firstElement();
       messages.removeElement( message );
       return message;
   }
}
class Producer extends Thread{
   private Critical res;
   public Producer( String name, Critical res )
   {
       super(name);
       this.res = res;
   }
   public void run()
   {
       try{
          while( true ) {
              res.putMessage();
              sleep( 1000 );
          }
       }
       catch( InterruptedException e ){}
   }
}
class Consumer extends Thread{
   private Critical res;
   public Consumer( String name, Critical res )
   {
       super(name);
       this.res = res;
   }
```

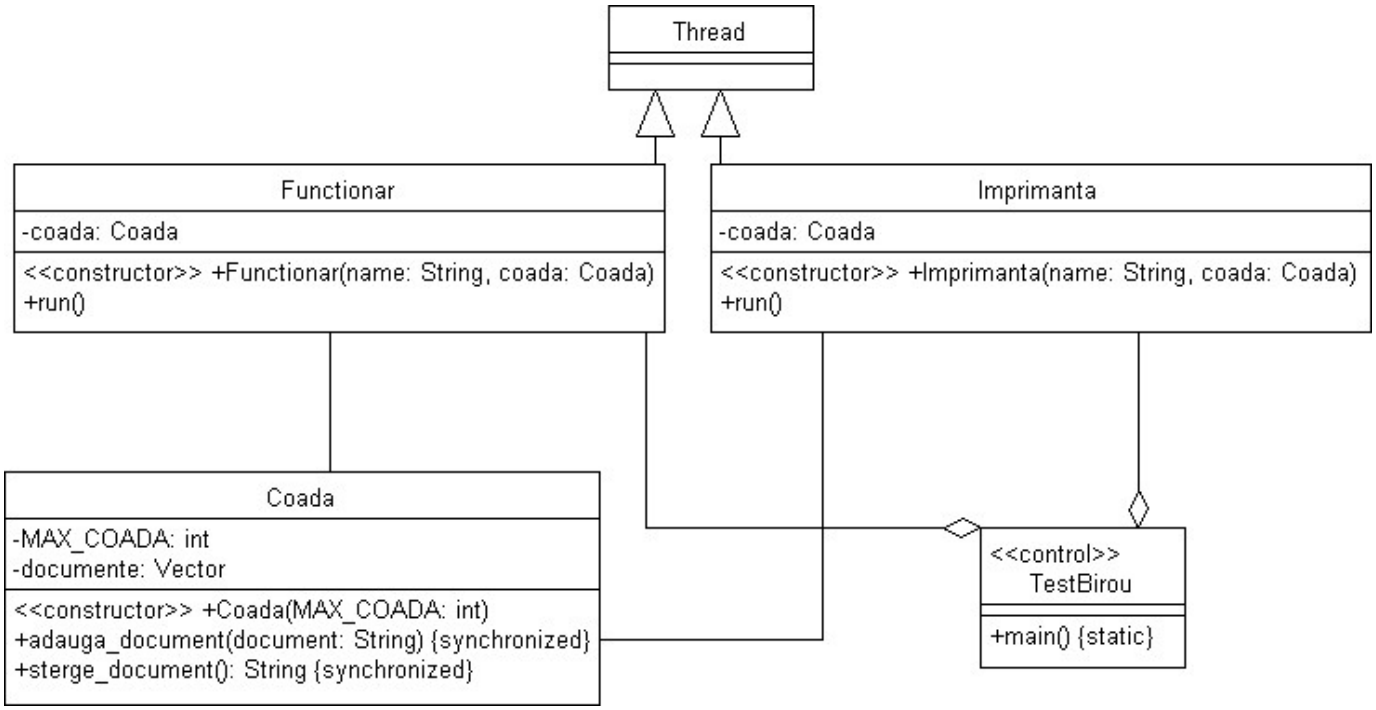
```
   public void run()
   {
      try{
         while( true ) {
            String message = res.getMessage();
            System.out.println("Consumator: "+message);
            sleep(2000);
         }
      }
      catch( InterruptedException e ) {}
   }
}
public class Control6{
   public static void main( String args[])
   {
      Critical res = new Critical();
      Producer p = new Producer( "Fir producator",res);
      Consumer c = new Consumer( "Fir consumator",res);
      p.start();
      c.start();
   }
}
```

2.

Specificatie problema:

Intr-un birou sunt 8 functionari care din cand in cand tiparesc la imprimanta documente, nu toti eleboreaza documentele in acelasi ritm. Fiindca au o singura imprimanta in birou, poate tipari doar o singura persoana la un moment dat. Sa se simuleze functionarea biroului.

Descrierea claselor



Sursa Java:

Coada.java

```
package Birou;
import java.util.*;
public class Coada{
   private Vector documente= new Vector();
   private int MAX_COADA;

   public Coada( int MAX_COADA ){
      this.MAX_COADA=MAX_COADA;
   }
   public synchronized void adauga_document( String document )
     throws InterruptedException{
      while( documente.size() == MAX_COADA )
         wait();
      documente.addElement( document );
      notifyAll();
   }
   public synchronized String sterge_document()
     throws InterruptedException{
      notifyAll();
      while( documente.size() == 0 )
         wait();
      String document = ( String )documente.elementAt(0);
      documente.removeElementAt( 0 );
```

```
        return document ;
    }
}
```

Functionar.java

```
package Birou;
public class Functionar extends Thread{
    private Coada coada;
    private int nr_document;
    public Functionar ( String name, Coada coada ){
        setName( name );
        this.coada = coada;
        nr_document = 0;
    }
    public void run(){
     try{
        while( true ){
         nr_document++;
         coada.adauga_document( getName()+"__"+Integer.toString(nr_document));
         sleep( (int) (Math.random() * 2000) );
        }
     }
     catch( InterruptedException e ){
     }
    }
}
```

Imprimanta.java

```
package Birou;
public class Imprimanta extends Thread{
    private Coada coada;
    public Imprimanta ( String name, Coada coada ){
        setName( name );
        this.coada = coada;
    }
    public void run(){
     try{
        while( true ){
         System.out.println( "TIPARIRE: "+coada.sterge_document());
         sleep( (int) (Math.random() * 500) );
        }
     }
     catch( InterruptedException e ){
     }
    }
```

```
}
```

TestBirou.java

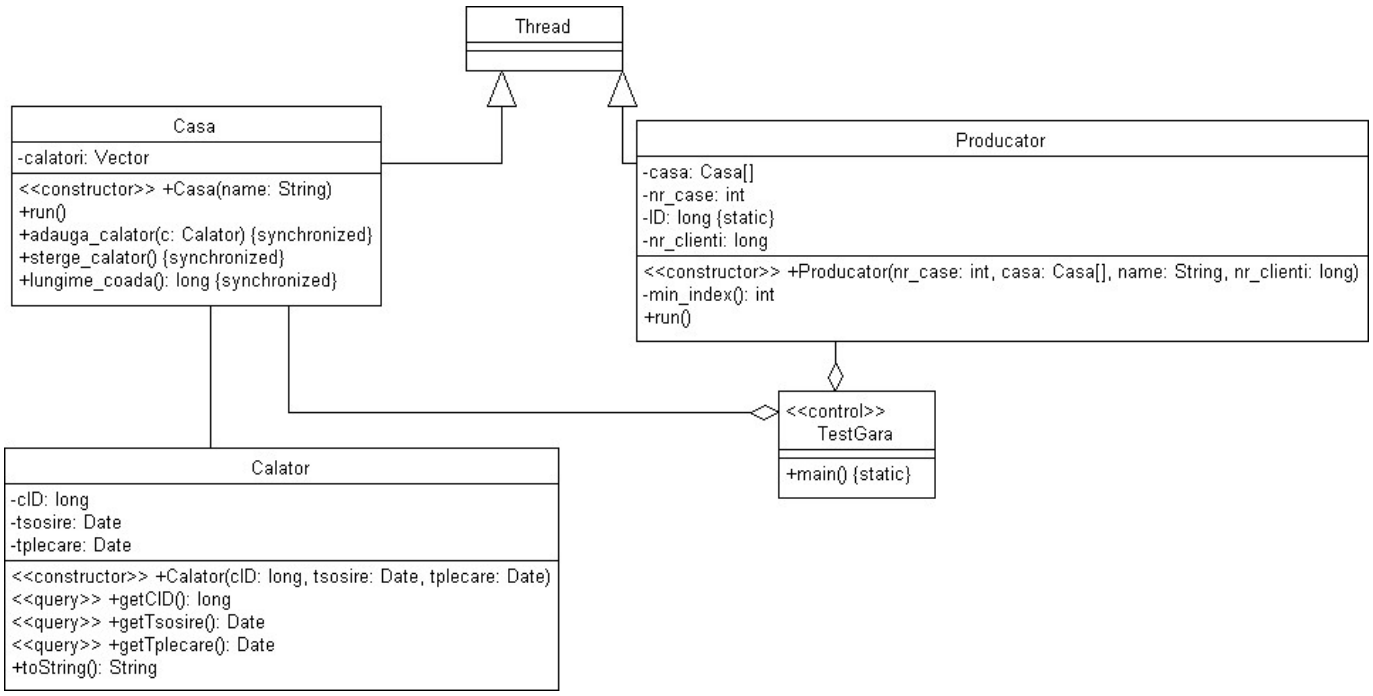
```
import Birou.*;
public class TestBirou{
   public static void main( String args[] ){
       int nr_functionari =4;
       Coada c = new Coada( 10 );
       Functionar f[] = new Functionar[5];
       for( int i=0;i<nr_functionari; i++ ){
           f[ i ] = new Functionar( Integer.toString(i+1), c);
           f[ i ].start();
       }
       Imprimanta i = new Imprimanta( "Imprimanta" , c );
       i.start();
   }
}
```

3.

Specificatie problema:

Intr-o gara sunt 3 case de bilete. Calatorii se aseaza la coada la una dintre cozi (de ex: la care e mai scurta). Dar casieritele nu vand biletele in acelasi ritm. Simulati functionarea caselor de bilete.

Descrierea claselor



Sursa Java:

Calator.java

```
package Gara;
import java.util.*;
public class Calator{
   private long cID;
   private Date tsosire;
   private Date tplecare;
   public Calator( long cID, Date tsosire, Date tplecare ){
       this.cID = cID;
       this.tsosire  = tsosire;
       this.tplecare = tplecare;
   }
   public long getCID(){
       return cID;
   }
   public Date getTsosire(){
       return tsosire;
   }
   public Date getTplecare(){
       return tplecare;
   }
   public String toString(){
       return ( Long.toString(cID)+" "+tsosire.toString()+" "+tplecare.toString());
   }
}
```

Casa.java

```
package Gara;
import java.util.*;
public class Casa extends Thread{
   private Vector calatori=new Vector();
      public Casa( String name ){
      setName( name );
   }
   public void run(){
      try{
       while( true ){
            sterge_calator();
            sleep( (int) (Math.random()*4000) );
       }
      }
      catch( InterruptedException e ){
          System.out.println("Intrerupere");
          System.out.println( e.toString());
       }
```

```
    }
    public synchronized void adauga_calator( Calator c ) throws InterruptedException
    {
        calatori.addElement(c);
        notifyAll();
    }
    public synchronized void sterge_calator() throws InterruptedException
    {
        while( calatori.size() == 0 )
            wait();
        Calator c = ( Calator )calatori.elementAt(0);
        calatori.removeElementAt(0);
        System.out.println(Long.toString( c.getCID())+" a fost deservit de casa "+getName());
        notifyAll();
    }
    public synchronized long lungime_coada() throws InterruptedException{
        notifyAll();
        long size = calatori.size();
        return size;
    }
}
```

Producator.java

```
package Gara;
import java.util.*;
public class Producator extends Thread{

    private Casa casa[];
    private int nr_case;
    static long ID =0;
    private int nr_clienti;//Numar calatori care trebuie deserviti de catre casele de bilete
    public Producator( int nr_case, Casa  casa[], String name, int nr_clienti ){
        setName( name );
        this.nr_case = nr_case;
        this.casa = new Casa[ nr_case ];
        this.nr_clienti = nr_clienti;
        for( int i=0; i<nr_case; i++){
            this.casa[ i ] =casa[ i ] ;
        }
    }
    private int min_index (){
      int index = 0;
      try
      {
        long min  = casa[0].lungime_coada();
        for( int i=1; i<nr_case; i++){
            long lung = casa[ i ].lungime_coada();
```

```
            if ( lung < min ){
                min = lung;
                index = i;
            }
        }
      }
      catch( InterruptedException e ){
          System.out.println( e.toString());
       }
      return index;
    }

    public void run(){
      try
      {
        int i=0;
        while( i<nr_clienti ){
            i++;
            Calator c = new Calator( ++ID, new Date(), new Date() );
            int m = min_index();
            System.out.println("Calator :" +Long.toString( ID )+" adaugat la CASA "+
Integer.toString(m));
            casa[ m ].adauga_calator( c );
            sleep( (int) (Math.random()*1000) );
        }
      }
      catch( InterruptedException e ){
          System.out.println( e.toString());
      }
    }
}
```

TestGara.java

```
import Gara.*;
public class TestGara{
    public static void main( String args[] ){
      int i;
      int nr = 4;
      Casa c[] = new Casa[ nr ];
      for( i=0; i<nr; i++){
          c[ i ] = new Casa("Casa "+Integer.toString( i ));
          c[ i ].start();
      }
      Producator p = new Producator( nr , c, "Producator",30);
          p.start();
    }
}
```

III. MODUL DE LUCRU

1. Se editează codul sursă al programului Java folosind un editor de text disponibil(de ex., se poate utiliza UltraEdit, Notepad).
2. Se salvează fişierul cu extensia **.java**. Fişierul trebuie salvat la următoarea locație: **c:\JBulider7\jdk1.3.1\bin**
3. Compilarea aplicației Java se va face din linia de comandă. Se poate proceda astfel. Se deschide o fereastră MS-Dos: Start ->Run, se tipăreşte **command** în căsuța de text şi se apasă butonul OK. Printr-o schimbare de directoare şi subdirectoare se trece la locația: **c:\JBulider7\jdk1.3.1\bin.** Sau, se lansează WindowsCommander. Se trece la locația **c:\JBulider7\jdk1.3.1\bin.** Se deschide o fereastră MS-Dos: Commander ->Run Dos.
4. Pentru compilare, se tipăreşte la prompter **javac nume_fişier_sursă.java** şi se apasă Enter. De ex., dacă fişierul se numeşte Test.java, se va scrie **javac Test.java**. În cazul în care programul conține erori acestea vor fi semnalate şi afişate în fereastra MS-Dos, după care va apare iar prompter-ul. Dacă programul nu conține erori şi compilarea se face cu succes, atunci va apare numai prompter-ul.
5. Pentru rularea unei aplicații Java stand-alone, se lansează interpretorul Java. Se tipăreşte la prompter următoarea comandă **java nume_clasă_care_conține_main** şi se apasă Enter. De ex., dacă clasa din program care conține metoda *main( )* se numeşte Test, se va scrie **java Test**.
6. Dacă programul Java este un applet, se editează fişierul .**html**. Se salvează în acelaşi director cu fişierul **.class** rezultat în urma compilării cu succes a fişierului sursă java. Apoi pentru rulare se poate utiliza **appletviewer nume.html.**
Alternativ, după compilarea aplicației Java, fişierul **.class** împreună cu fişierul **.html** pot fi mutate în orice alt director (nu trebuie neapărat să fie în **c:\JBulider7\jdk1.3.1\bin**). Se încarcă fişierul **.html** într-un browser Web (ex., Internet Explorer).

IV. TEMĂ

1. Se vor parcurge toate exemplele prezentate în platforma de laborator testându-se practic.
3. Scrieți un program Java care generează două fire de execuție pentru parcurgerea unui String de la cele două capete. Folosiți doi pointeri a căror valoarea se incrementează, respectiv se decrementează într-o funcție din memoria comună (synchronized). În memoria comună se află String-ul. Firele de execuție se întâlnesc la „mijloc”.