# Laboratory 7

# Java SE runtime applications - Testing and implementing Petri nets and Time Petri nets, using synchronization mechanisms from the java.util.concurrent package

## 1. Laboratory objectives

- developing the following skills:
  - understanding of a discrete system described by Petri nets and / or time Petri nets;
  - the ability to design a system described by Petri nets and / or time Petri nets;
  - the ability to implement a system (*Java SE* application) described by Petri and / or time Petri nets;
  - use of synchronization mechanisms in the *java.util.concurrent* package.

## 2. Applications

Problem statement:

It requires the design and implementation in *Java SE* of the following systems (applications) described through Petri timed networks (Figures 7.1, 7.2, 7.3 and 7.4).

For the design phase the system will be described using the following types of diagrams:

- state machines diagram;

- class diagram;

- sequence diagram.

*Conventions:*

Timings on places correspond to activities that will be implemented using code sequences in the following form:

```
int k = ...;//nr. aleator în intervalul specificat
for (int i = 0; i < k * 100000; i++) {
        i++;
        i--;
}
```

Timed transitions are pure delays (which do not load the processor) and will be implemented by using the *Thread.sleep*(*x*) method.

*Testing applications:*

The applications will be implemented so that the synchronization between the threads of execution can be tested during run time. Tip: Use additional *sleep*() calls and display text messages in the console.

**2.1 Application 1:**

The Petri network of Figure 7.1 is given. The synchronization elements represented by the P9 and P10 places will be implemented using the *ReentrantLock* and *Semaphore* classes (provide two programs).

The synchronization represented by the T8 transition will be implemented using the *CyclicBarrier* class.

Is there a risk that the network will enter the interlock? If so, modify the net and deployment so that there is no interlock.

Test the use of the CountDownLatch class instead of the CyclicBarrier class. What can be observed?

**2.2 Application 2:**

The Petri net of Figure 7.2 is given. The synchronization elements represented by the P9 and P10 places will be implemented using the *ReentrantLock* and *Semaphore* classes (provide two programs).

The synchronization is represented by the T8 transition will be implemented using the *CyclicBarrier* class.

Test the use of the *CountDownLatch* class instead of the *CyclicBarrier* class. What can be observed?

**2.3 Application 3:**

The Petri net of Figure 7.3 is given. The *wait*()/*notify*() methods will be used for T6-P6-T7 and T6-P10-T12 synchronizations.

The final synchronization from T11 will be implemented using the *CountDownLatch* class.

Initially, set the values of the variables: x = y = 5. Then test the functionality of the application.

Change the values of the variables: x = y = 10. What can be observed?

It is required to modify the implementation (synchronization T6-P6-T7 and T6-P10-T12) so that the application works correctly regardless of the values of the variables. What synchronization might be used in this case?

**2.4 Application 4:**

The Petri net of Figure 7.4 is given. P8 place contains two tokens in the initial marking. What does this mean in terms of implementation?

Choose the appropriate mechanism for implementing the synchronization represented by the P8 place.

**2.5 Application 5:**

The state machine of Figure 7.5 is given. Draw the Petri-Net graph for it.

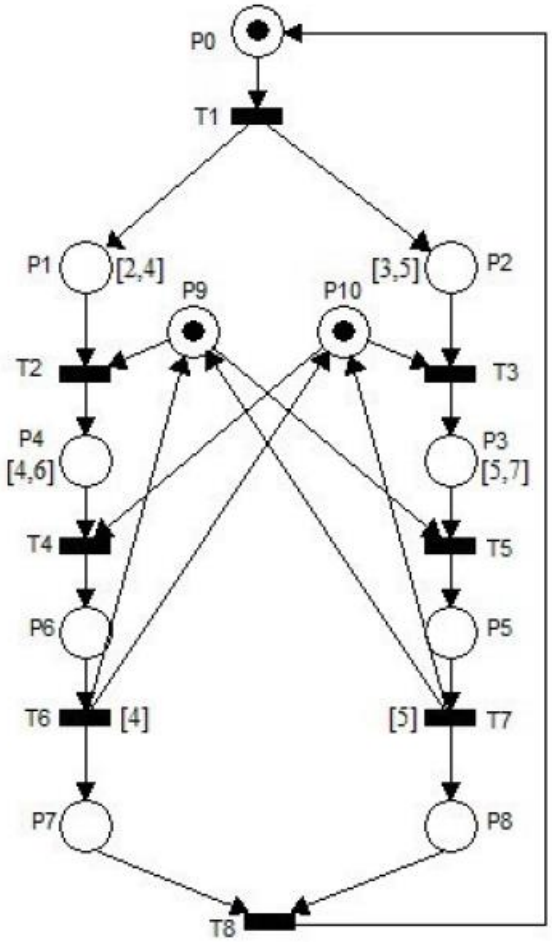


Figure 7.1 Application 1

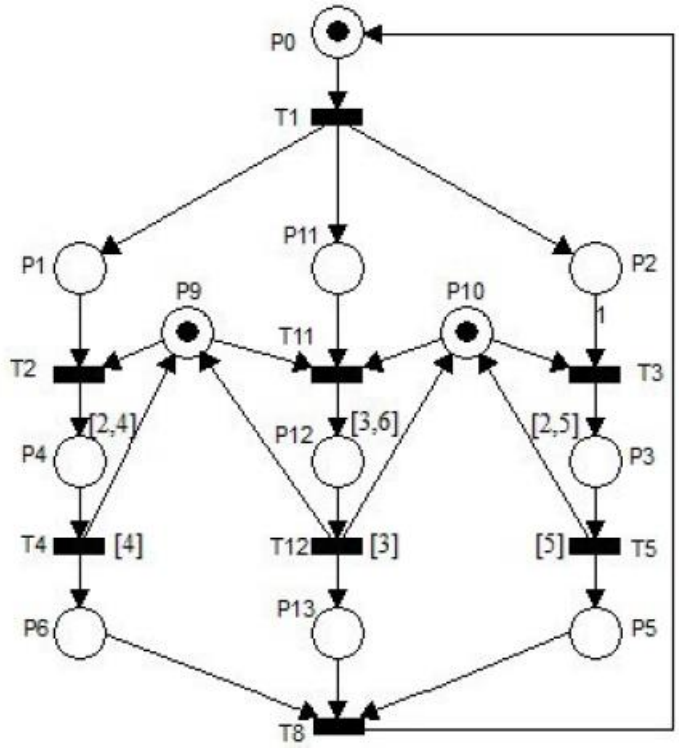


Figure 7.2 Application 2

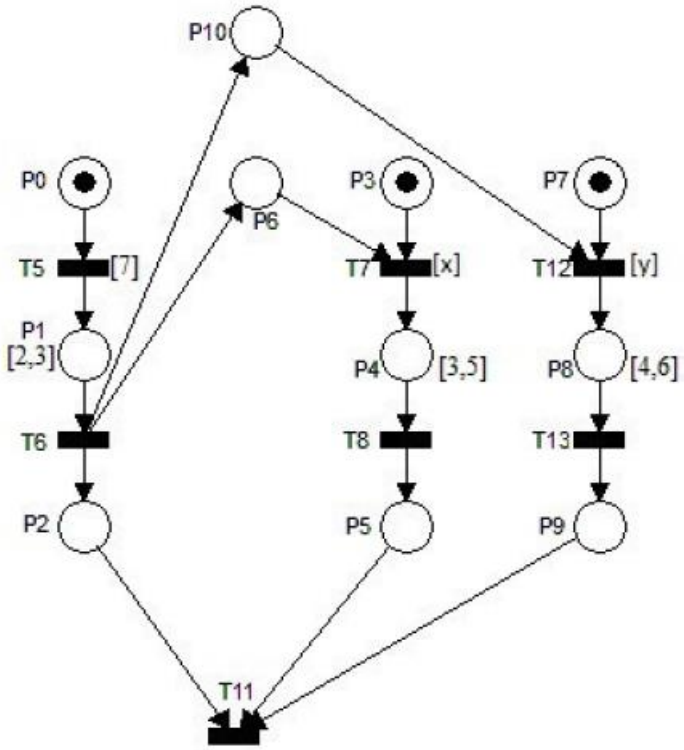


Figure 7.3 Application 3

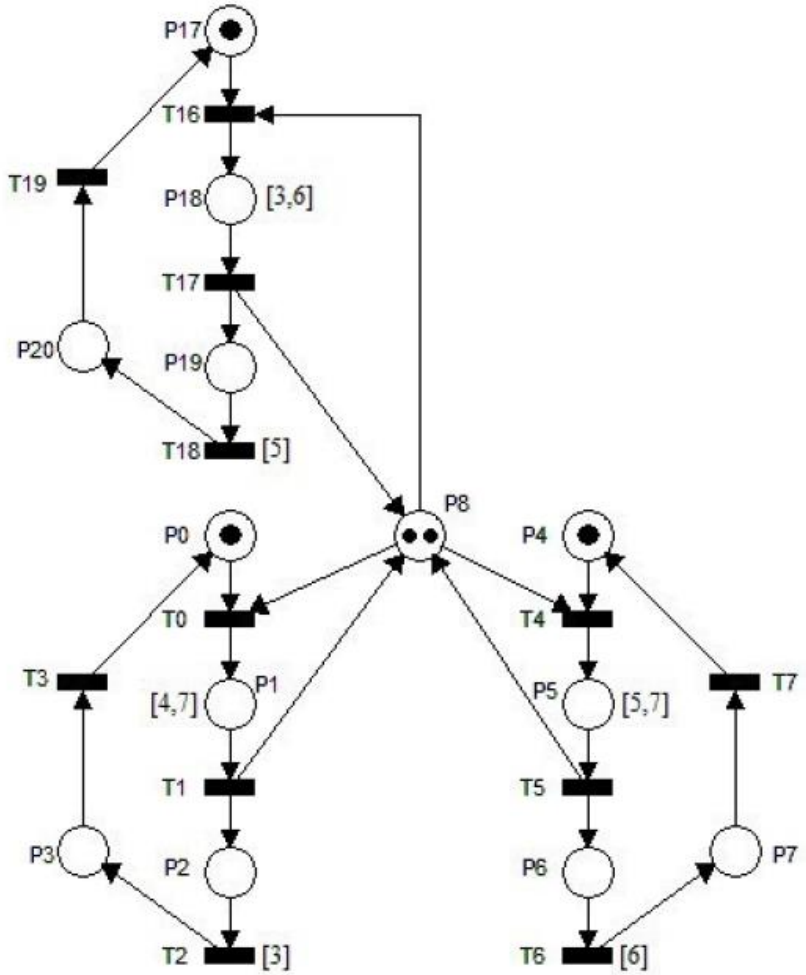


Figure 7.4 Application 4

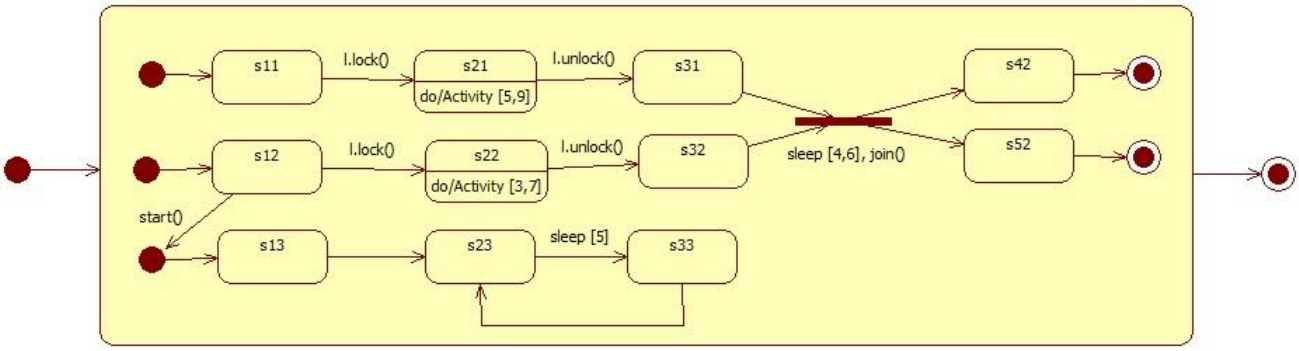


Figure 7.5 Application 5