Informe Trabajo Final: Programación Concurrente

# Sistema de manufacturación robotizado

Profesores: Dr. Orlando Micolini, Ing. Luis Ventre

Institución: F.C.E.F.Y.N – U.N.C

Alumnos: Nicolás Passaglia, Agustín Colazo

Fecha: 12/08/2017

# Índice

1. Objetivos
2. Introducción
3. Desarrollo
   1. Red de Petri
      1. Red Inicial
      2. Red destrabada
      3. Tablas de eventos y estados
   2. Programa en Java
      1. Requerimientos
      2. Interfaz
      3. Gestor de Monitor
      4. Tiempos
      5. Políticas
4. Problemas presentados
5. Testing
   1. Unit Tests y tests a invariantes
   2. System test con red alternativa
6. Conclusión

# Objetivos

El objetivo del trabajo fue la resolución de un problema de concurrencia a través de la implementación de una red de Petri con monitor en el lenguaje de programación Java. Dicho problema correspondía a una fábrica que contaba con varias líneas de producción de 3 diferentes productos y había que lograr la ejecución concurrente de las máquinas y robots que conformaban la fábrica.

# Introducción

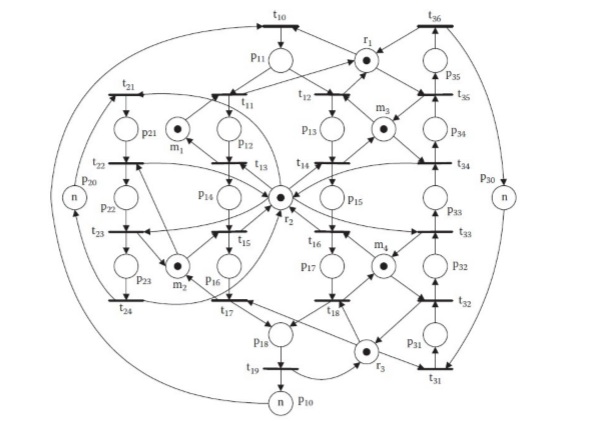
En el trabajo que se desarrollara a continuación vemos los diferentes pasos a travesados para conseguir implementar una red de Petri en Java. La idea de dicha implementación era lograr un programa general que pueda ser capaz de correr cualquier red.

Se comienza el problema con la red de la fábrica, pero con esta inicialmente trabada, es decir con la existencia de un deadlock.

# Desarrollo

## Red de Petri

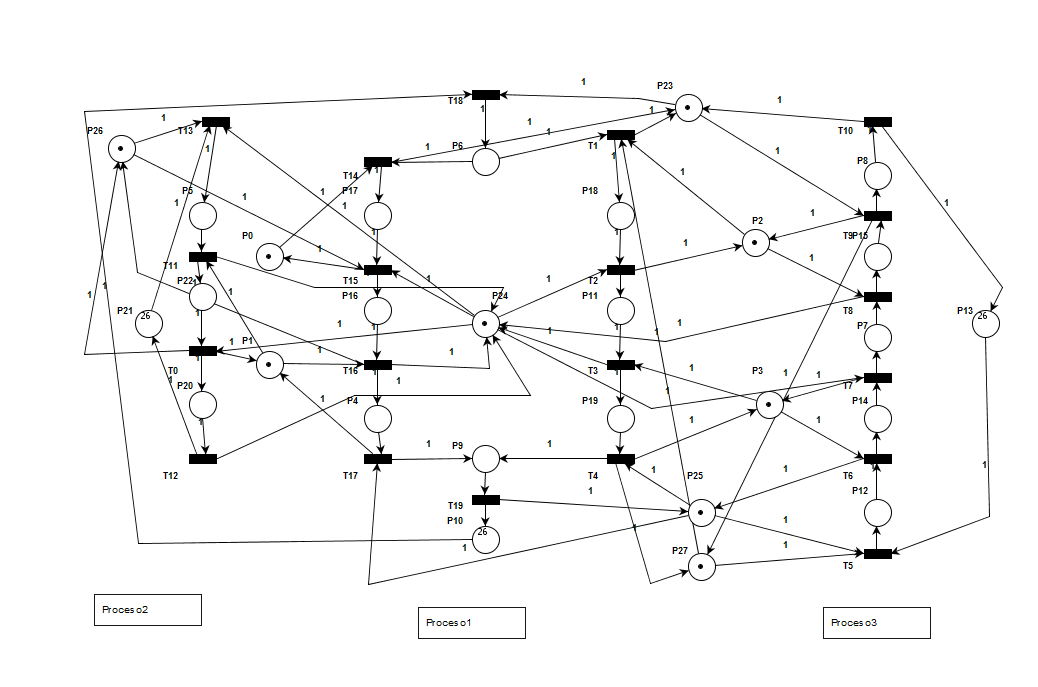
Se muestra a continuación la red inicial la cual contaba con un deadlock.



Esta red se disparó un gran número de veces a través del programa PIPE, mostrando siempre que estaba trabada.

La solución se realizó agregando dos semáforos para controlar el recurso de ciertos robots y maquinas.

A continuación la red destrabada.



Donde P26 y P27 son los respectivos semáforos utilizados para destrabar la red.

Una vez destrabada la red se hace un siguiente análisis de la RdP, que es la realización de la tabla de eventos y estados. Donde la tabla de eventos relaciona un evento con una transición.

|  |  |
| --- | --- |
| Evento | Transición |
| Extracción de insumo para Producto B de I2 con R2 | T3 |
| Devolución R2 y comienzo de producción con M2 | T2 |
| Depósito de Producto B con R2 | T20 |
| Extracción de insumo para Producto C de I3 con R1 | T8 |
| Comienzo de producción de A con M1 | T4 |
| Comienzo de producción de A con M3 | T10 |
| Producción de A con M2 luego de M1 | T6 |
| Producción de A con M4 luego de M3 | T12 |
| Depósito de Producto A con R3 | T7, T13 |
| Extracción de insumo para Producto C con R3 | T14 |
| Comienzo de producción de C con M4 | T15 |
| Producción de C con M3 luego de M4 | T17 |
| Depósito de Producto C con R1 | T19 |

Y la tabla de estados, relaciona un estado del sistema a un conjunto de transiciones

TABLA DE EXCEL MAL->CORREGIR

## Programa en Java

### Requerimientos

Funcionales:

1. El programa debe ser capaz de modelar y seguir la lógica de la red de Petri a la perfección, es decir ejecutar solo transiciones sensibilizadas, calcular nuevo marcado y nuevo vector de sensibilizadas.
2. El programa debería ser lo suficientemente general para poder correr cualquier red.
3. El programa debe anunciar luego de una cierta cantidad de disparos los valores de los t-invariantes que en el caso de la fábrica nos dirá cuantas piezas se produjo de cada una.
4. El programa tiene que modelar a la perfección el tiempo en una red de Petri.

No Funcionales:

1. La ejecución continua del programa no debería consumir mucha memoria RAM
2. El gestor del monitor no debe tomarse más de 100ms en decidir.

### Interfaz de Usuario

Una vez comenzada la ejecución, el programa le da a elegir al usuario que red desea ejecutar. Para lograr esto dentro de la carpeta del proyecto se crearon subcarpetas individuales para cada red, donde se encuentra archivos que modelan dicha red. Primero un archivo html generado con el PIPE que contiene el marcado y la incidencia, luego archivos .txt como hilos.txt que contiene que hilo ejecuta cada transición, tiempo.txt que muestra los alfa y beta de cada transición (w en beta significa infinito), y por ultimo invariantes.txt que indica los t y p invariantes.

### Gestor de Monitor

La clase Gestor De Monitor es la encargada de tomar la decisión sobre la transición que se ejecuta, es de común creencia que esta implementación causaría una ejecución secuencial del programa y no concurrente, pero esto no es así ya que el Gestor solo decide y luego la ejecución de la tarea se da “fuera” del monitor, generando así una correcta ejecución concurrente.

Este gestor está compuesto por una cola fifo de entrada, un mutex que funciona como lock permitiendo que los hilos entren de a uno al monitor y además contiene campos de Colas y Políticas que ayudan a dicho Gestor a decidir en caso de las políticas y para mandar a los hilos a hacer cola en el caso de las Colas. Además de esto cuenta con la Red de Petri, ya que tiene que saber si dicha transición que quiere ejecutar el hilo esta sensibilizada.

Diagrama de secuencia del Monitor

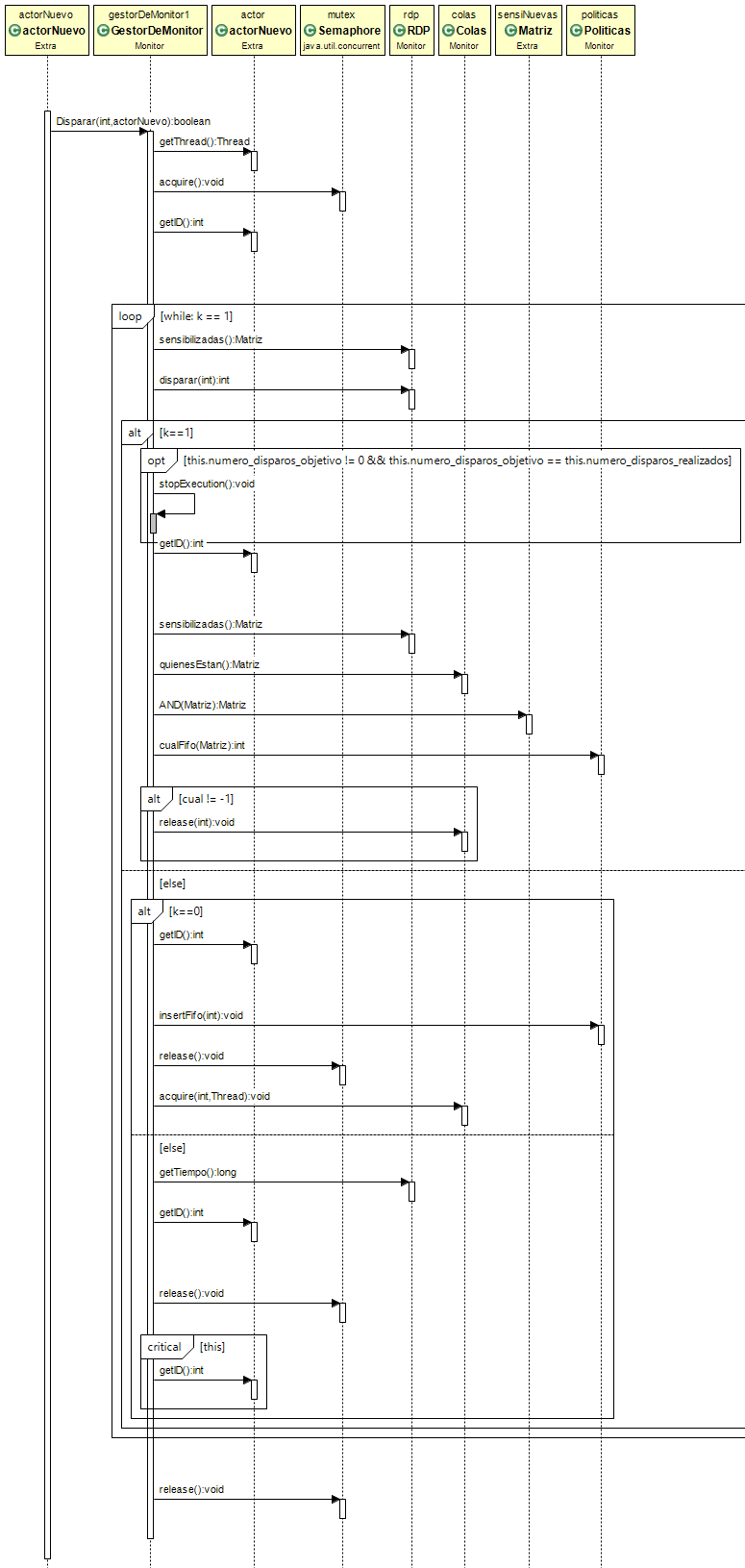


Diagrama realizado con plugin de Eclipse.

### Tiempo

El tiempo en las redes de Petri, ayuda a modelar al sistema a uno de tiempo real, que sea similar a la realidad. Ya que le establece una cierta ventana de tiempo a una transición para ejecutarse una vez sensibilizada. Es decir cuando se sensibiliza una transición no puede ejecutarse cuando quiere sino en esta ventana donde α<T<β, esto simula la realidad ya que en una fábrica si una brazo robótico que se encarga de pintar una pieza recibe la señal de que la pieza esta por pasar por él, no puede hacerlo 1 minuto después o un minuto antes ya que la pieza pasa por el en un cierto determinado tiempo.

En nuestro programa la clase tiempo se encarga de decirle al gestor si la transición que quiere ser ejecutada se encuentra en su ventana, en el caso que si se ejecuta, si esta antes lo duerme hasta que esté en la ventana y si se pasó lo manda a la cola. Además de esto el tiempo modela si alguien ya está durmiendo para ejecutar una transición y llega un hilo que quiere ejecutar la misma, despierta al que estaba esperando.

Para la fábrica trabajada se establecieron los siguientes tiempos:

Para la pieza A:

• M1 = 30; M2 = 5

• M3 = 24; M4 = 10

Para la pieza B: • M2 = 15

Para la pieza C: • M4 = 21; M3 = 18

El correcto funcionamiento de la red de Petri con tiempo ser verifico en el programa TINA.

### Políticas

Las políticas es una clase de gran utilidad para el Gestor y para el programa en si ya que son las que le dicen al monitor que transición es la que se va a ejecutar a continuación, en nuestro caso se implementó una política FIFO.

## Diagrama de clases del programa.

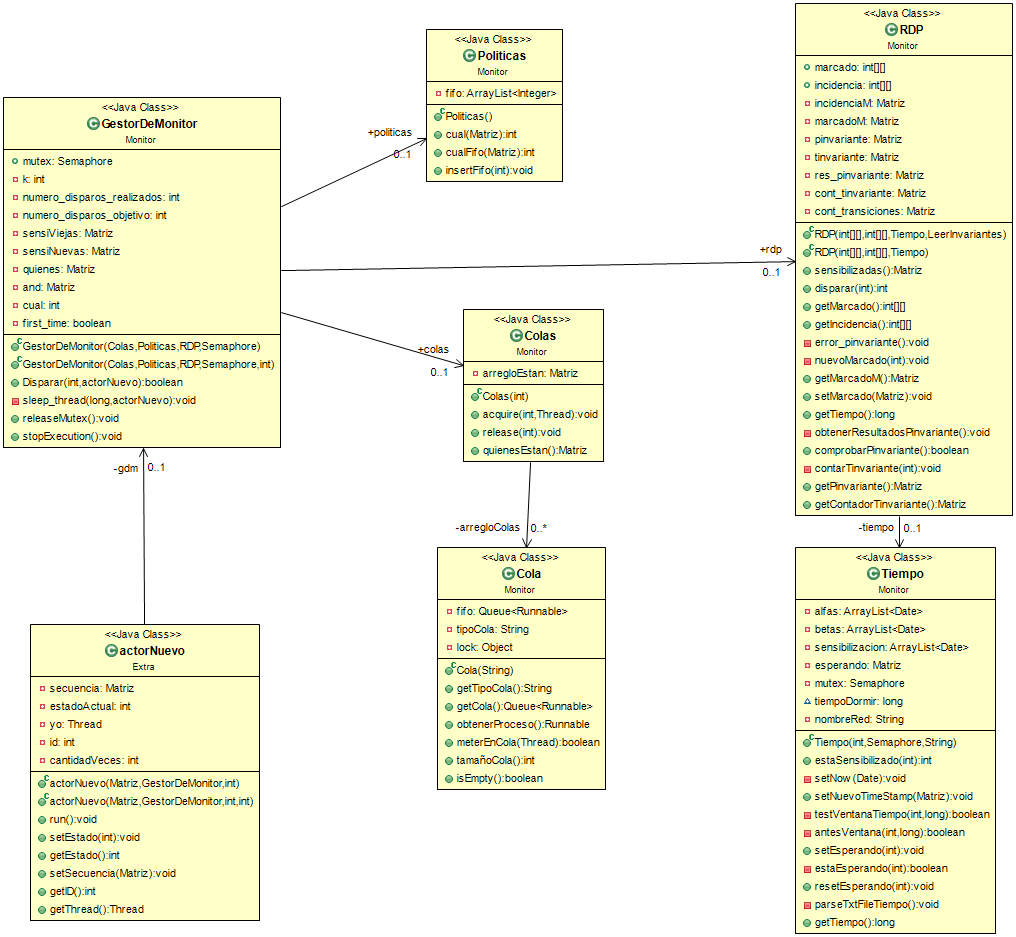


Diagrama generado con plugin de eclipse.

# Problemas presentados

# Testing

# Unit Tests

## Unit tests de clase Matriz.

Clase: UnitTestMatriz

SumaMatrices

|  |  |
| --- | --- |
| **Descripción del Test** | Chequea que el método suma de matrices funcione correctamente |
| **Ejecución** | * Se crean 3 matrices de 1x1. * A con valor 1, B con valor 2 y C con valor 3 * Se hace un assertArrayEquals con C y la suma de A y B |
| **Resultado esperado** | 3 (Pass) |
| **Pass/Fail** | **PASS** |

RestaMatrices

|  |  |
| --- | --- |
| **Descripción del Test** | Chequea que el método resta de matrices funcione correctamente |
| **Ejecución** | * Se crean 3 matrices de 1x1. * A con valor 5, B con valor 2 y C con valor 3 * Se hace un assertArrayEquals con C y la resta de A y B |
| **Resultado esperado** | 3 (Pass) |
| **Pass/Fail** | **PASS** |

MultAndMatrices

|  |  |
| --- | --- |
| **Descripción del Test** | Chequea que el método de una Multiplicacion miembro a miembro funcione correctamente |
| **Ejecución** | * Se crean 3 matrices de 1x1. * A con valor 10, B con valor 2 y C con valor 20 * Se hace un assertArrayEquals con C y la multAnd de A y B |
| **Resultado esperado** | 20 (Pass) |
| **Pass/Fail** | **PASS** |

Transpose

|  |  |
| --- | --- |
| **Descripción del Test** | Chequea que el método que transpone una matriz funcione |
| **Ejecución** | * Se crean 2 matrices una 1x3 y otra 3x1. * A con 3 valores y B con los valores de A transpuestos * Se hace un assertArrayEquals con B y la transpuesta de A |
| **Resultado esperado** | Transpuesta de A (Pass) |
| **Pass/Fail** | **PASS** |

Clear

|  |  |
| --- | --- |
| **Descripción del Test** | Chequea que el método de limpiar una matriz funcione correctamente |
| **Ejecución** | * Se crean 2 matrices de 3x1. * A con ciertos valores y B matriz con todos 0. * Se hace un assertArrayEquals con B y A luego de un clear. |
| **Resultado esperado** | A con todos elementos 0 |
| **Pass/Fail** | **PASS** |

SetIdentity

|  |  |
| --- | --- |
| **Descripción del Test** | Chequea que el método de crear una matriz identidad funcione |
| **Ejecución** | * Se crean 2 matrices 3x3 * A la matriz A se le aplica el método setIdentity * A la matriz B se la crea como identidad y se las compara. |
| **Resultado esperado** | Matriz A como identidad |
| **Pass/Fail** | **PASS** |

Producto por un escalar

|  |  |
| --- | --- |
| **Descripción del Test** | Chequea que el método de productor por un escalar funcione |
| **Ejecución** | * Se crean 2 matrices de 3x1. * A con valores 5, -2,10 y B con 15,-6,30 * Se hace un assertArrayEquals con B y A productor por 3. |
| **Resultado esperado** | A con valores 15, -6, 30 |
| **Pass/Fail** | **PASS** |

Es Nula

|  |  |
| --- | --- |
| **Descripción del Test** | Chequea que el método es nula funcione correctamente |
| **Ejecución** | * Se crean 1 matriz de 3x1. * Se le hace productor por un escalar con 0 * Se hace un assertTrue con el método es Nula |
| **Resultado esperado** | True |
| **Pass/Fail** | **PASS** |

## Unit Test clase Cola

meterEnCola

|  |  |
| --- | --- |
| **Descripción del Test** | Comprueba que el método meter en cola funcione |
| **Ejecución** | * Se crea un objeto Cola, y dos hilos que van a ser agregados * Se crean dos runnables que llaman al método meterencola * Se crea una Queue local y se la compara |
| **Resultado esperado** | Queue local igual a la campo Queue de la clase Cola. |
| **Pass/Fail** | **PASS** |

testColaDormidos

|  |  |
| --- | --- |
| **Descripción del Test** | Chequea que la cola fifo de hilos dormidos funcione |
| **Ejecución** | * Misma lógica que el test de meterEnCola * La diferencia esta en el Assert * Se compara el primer hilo metido en cola con el hilo obtenido en a través del método obtenerProceso. |
| **Resultado esperado** | Iguales |
| **Pass/Fail** | **PASS** |

testColaDormidosNegativo

|  |  |
| --- | --- |
| **Descripción del Test** | Chequea que la cola fifo de hilos dormidos funcione |
| **Ejecución** | * Igual que el test positivo * La diferencia esta en el Assert * Ahora se compara que el hilo obtenido sea distinto al segundo |
| **Resultado esperado** | Distintos hilo obtenido con el hilo metido en segundo lugar. |
| **Pass/Fail** | **PASS** |

## Unit Tests a leerTxtFile

|  |  |
| --- | --- |
| **Descripción del Test** | Chequea que el método leerTxtFile funcione correctamente |
| **Ejecución** | * Se guarda en una matriz de enteros la lectura de un archivo txt * Se crea una matriz con el resultado esperado * Se hace un assertArrayEquals entre ambas matrices |
| **Resultado esperado** | Iguales |
| **Pass/Fail** | **PASS** |

## Unit Tests Politicas

testPoliticas

|  |  |
| --- | --- |
| **Descripción del Test** | Chequea que las políticas fifo funcionen correctamente. |
| **Ejecución** | * Creo una matriz que simula la matriz de tr sensibilizadas * A un objeto políticas se le llama el método insertFifo con 3 nros. * Se compara el primero valor ingresado y el valor devuelto por   el método cual de políticas. |
| **Resultado esperado** | Que el método cual devuelva el primer elemento ingresado (fifo) |
| **Pass/Fail** | **PASS** |

testPoliticasNegativo

|  |  |
| --- | --- |
| **Descripción del Test** | Chequea que las políticas fifo funcionen correctamente. |
| **Ejecución** | * Creo una matriz que simula la matriz de tr sensibilizadas * A un objeto políticas se le llama el método insertFifo con 3 nros. * Se compara el segundo valor ingresado y el valor devuelto por   el método cual de políticas. |
| **Resultado esperado** | Que el método cual devuelva un numero distinto al segundo ingresado. |
| **Pass/Fail** | **PASS** |

## Test de P-Invariantes

|  |  |
| --- | --- |
| **Descripción del Test** | Test de los p-invariantes. |
| **Ejecución** | * Se inicializa la red de Petri con su marcado e incidencia * Se le pasa un marcado distinto del inicial. * Se hace assertTrue o assertFalse al método   comprobarPinvariante depediendo si el marcado es uno valido o no |
| **Resultado esperado** | True |
| **Pass/Fail** | **PASS** |

Distintos Marcados pasados:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Marcado no valido -> Devuelve false y el tast pasa correctamente

Marcado 2:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 24 | 0 | 0 | 26 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 26 | 0 | 1 | 0 | 1 | 0 | 1 |

Marcado valido -> Devuelve true y pasa el test correctamente.

Marcado 3:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 26 | 0 | 0 | 26 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 26 | 0 | 1 | 1 | 1 | 1 | 1 |

Marcado No Valido -> Devuelve false y el test pasa correctamente

Marcado 4:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 25 | 0 | 0 | 24 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 25 | 0 | 0 | 0 | 1 | 0 | 0 |

Marcado Valido -> Devuelve true y el test pasa correctamente.

## Unit test T-Invariante

|  |  |
| --- | --- |
| **Descripción del Test** | Test de los t-invariantes. |
| **Ejecución** | * Se inicializa la red de Petri, el gestor de monitor y las colas. * Ahora los actores se construyen limitando la cantidad de veces   que ejecutan su secuencia.   * Se compara un contador de t-invariantes obtentido después   de que los actores terminaran con sus secuencias y el vector esperado  que se relaciona con la cantidad de piezas producidas. |
| **Resultado esperado** | Que ambos vectores sean iguales. |
| **Pass/Fail** | **PASS** |

Hilo 1 ejecuta -> 13 11 0 12 y le pasamos 15 repeticiones

Hilo 2 ejecuta -> 18 y le pasamos 35 repeticiones

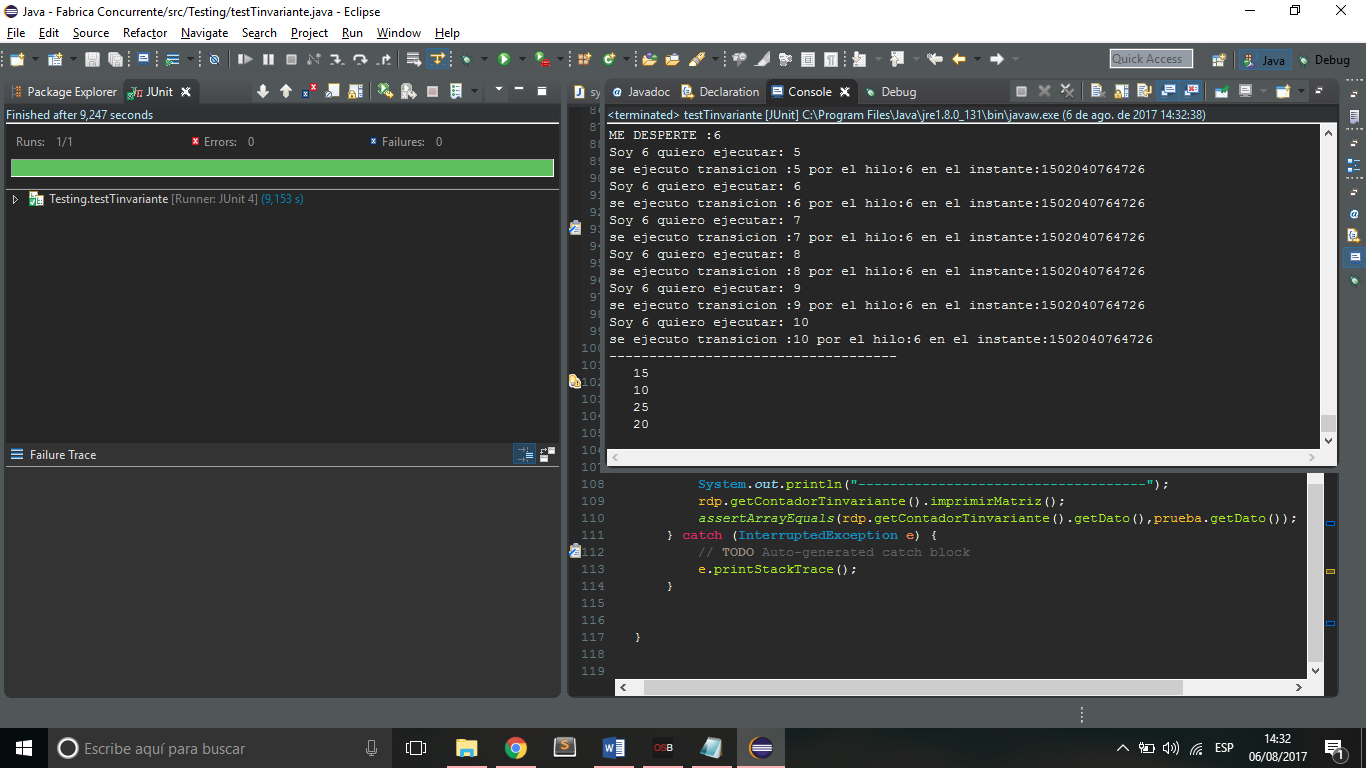
Hilo 3 ejecuta -> 19 y le pasamos 35 repeticiones

Hilo 4 ejecuta -> 1 2 3 4 y le pasamos 25 repeticiones

Hilo 5 ejecuta -> 14 15 y le pasamos 10 repeticiones

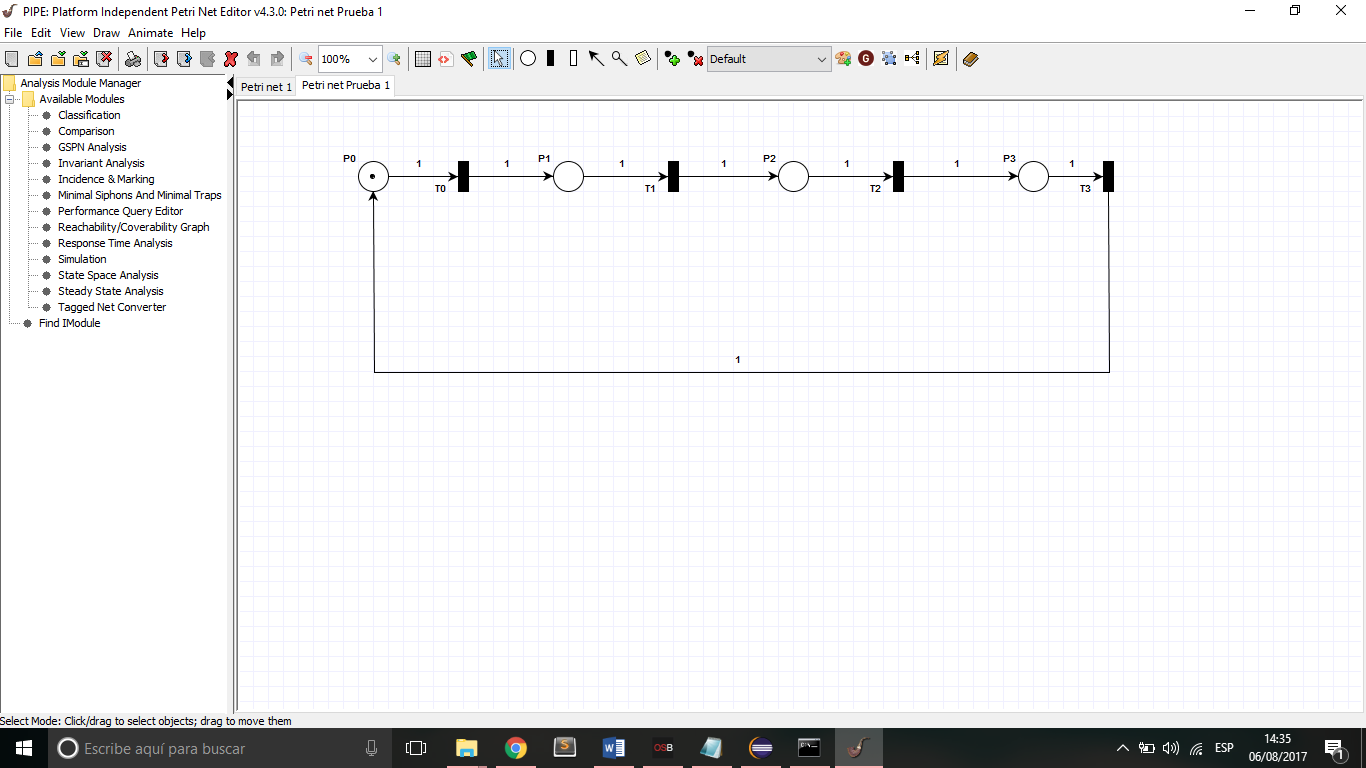
Hilo 6 ejecuta -> 16 17 y le pasamos 10 repeticiones

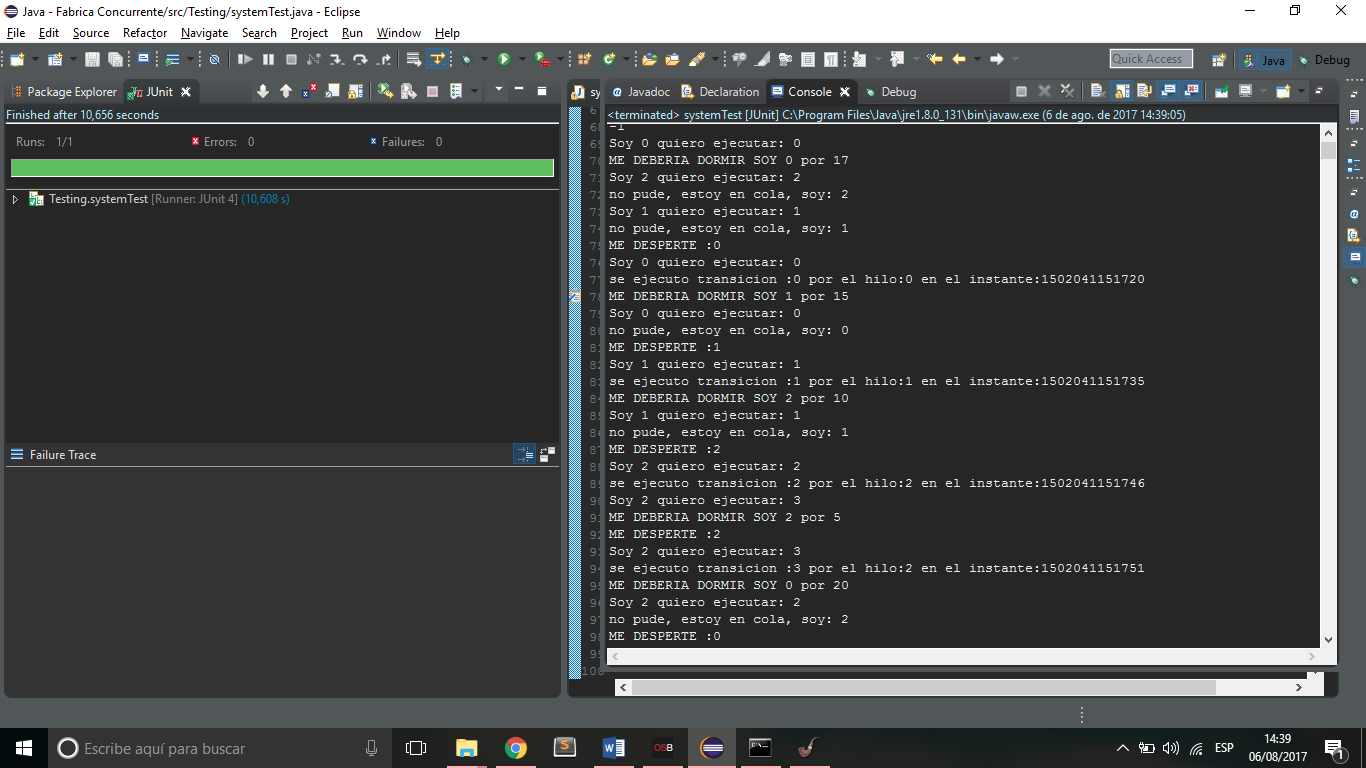
Hilo 7 ejecuta -> 5 6 7 8 9 10 y le pasamos 20 repeticiones

Y deberíamos obtener [15 10 25 20]

# System Test

Se realiza ahora un system test utilizando la siguiente red:

Con 4 hilos, uno para cada transición.



Y se ve claramente una correcta ejecución de la red.