

ELO329

TAREA 1: SEMÁFOROS COMO OBJETOS DE SOFTWARE

*Cristóbal Garrido Valenzuela
Matías Hernández Maturana
Ulises Leiva Bahamondes*

Índice

1. Explicacion de las clases implementadas en la Etapa 4	2
2. Problemas enfrentados	3
3. Diagrama de clases	4
4. Comandos Git	5

1. Explicacion de las clases implementadas en la Etapa 4

Entre los principales módulos que componen la etapa cuatro esta la utilización de las clases de las etapas anteriores, que se utilizaran para poder interactuar y relacionar sus métodos con los de otras clases. Además se incluyo la clase Imprimir, la cual funciona de modo paralelo con un timer e imprime el estado de los semaforos cada un segundo.

La clase TestStage4 de esta etapa crea dos instancias de la clase SemaforoP, una instancia de la clase SemaforoDeGiro, tres de la clase Semaforo3, tres de la clase DetectorDeRequerimiento, una de la clase Imprimir, una del Controlador y una del SimuladorEntradas, donde el SimuladorEntradas recibe como atributos las instancias del DetectorDeRequerimiento, el Controlador recibe de atributos las instancias del DetectorDeRequerimiento y las instancias de todos los tipos de semáforos anteriores para poder manejarlos y la clase Imprimir recibe instancias de todos los tipos de semaforos. Luego, se inicia en paralelo la lectura del archivo y la impresion de los estados, para despues llamar al método manageTraffic() en la cual se queda ejecutando hasta que se cierre el programa desde la clase SimuladorEntradas.

En manage traffic() el programa se queda a la espera de las señales de la clase SimuladorEntradas, la cual esta leyendo un archivo de texto con las ordenes que serán enviadas al controlador y que ejecutara los respectivos cambios de luces. Mientras manageTraffic() no reciba una orden de la clase SimuladorEntradas , esta se encuentran encendiendo y cambiando las luces de las 3 instancias de la clase semáforo3 simulando la intersección de un cruce entre 2 calles.

Cuando la clase SimuladorEntradas detecte una solicitud peatonal o del sensor inductivo, se activara la clase DetectorDeRequerimiento y se accionara el método isOn() respectivo a esa instancia, por lo que el controlador en el managetraffic() apagara los semáforos que correspondan para así encender los semáforos peatonal o de giro que se estén solicitando. Finalmente, se utiliza la clase creada Imprimir la cual lee el estado de los semáforos cada un segundo e imprime en la consola (de acuerdo al formato descrito en la descripción general) el estado de todos los semáforos.

Una vez acabados las líneas de código del archivo ingresado, la clase SimuladorEntradas cierra el programa.

2. Problemas enfrentados

Uno de los problemas que se presentaron en el desarrollo de la tarea fue con respecto a la clase controlador, ya que esta debía ser implementada del modo correcto para que se obtuviera la salida requerida, además de tener que ejecutar e imprimir el estado de los semáforos cada un segundo. Finalmente, se hicieron varias pruebas y modificaciones pequeñas en el código para que este resultara dando la salida requerida y se decidió utilizar el comando... para correr el código e imprimir cada un segundo.

También ocurrieron problemas para poder leer un archivo de texto externo, el cual se probó varias veces en las que no se lograba leer el archivo por la incorrecta implementación del código. Para solucionar esto se corrigieron varios errores en el código, errores de mala utilización de los códigos, de modo que los métodos que se necesitaban fueran implementados del modo correcto, además de añadir ciertos import al código, para poder utilizar las funciones que se querían utilizar sin ningún problema.

Además, ocurrieron problemas para poder implementar ciertas clases que utilizaban como atributos la instancia de otras clases, y que este atributo era compartido por dos clases, de modo que si la clase SimuladorEntradas modificaba algún método de la clase DetectorDeRequerimiento, (la cual era instanciada como atributo), la clase Controlador también viera modificada el valor dentro del método de la clase DetectorDeRequerimiento de su propia instancia. Esto se logró solucionar definiendo variables de instancias que tomaran el valor de estos atributos para así poder modificar los metodos de la clase DetectorDeRequerimiento dentro de SimuladorEntradas y que la clase Controlador pudiera leer esta modificación.

3. Diagrama de clases

A partir de lo desarrollado en el etapa 4, el diagrama de clases de esta etapa es:

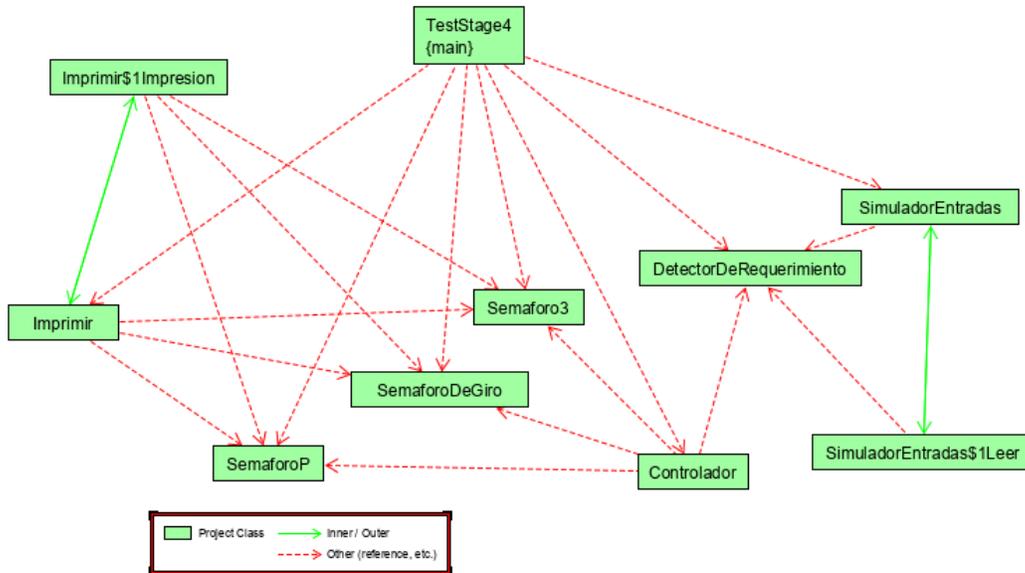


Figura 1: Diagrama de clases de la etapa 4.

En el diagrama se pueden ver todas las referencias y entradas que tienen las clases dentro de la etapa 4.

4. Comandos Git

Para finalizar esta documentación, mostraremos la respuesta de algunos comandos en GIT, con el cual veremos la lista de commits realizados, los cambios realizados en el repositorio, las líneas de código añadidas y/o cambiadas en algún archivo antes de hacer el commit, y las líneas de código añadidas en el ultimo commit. Los resultados obtenidos son los siguientes:

```

matias.hernandez@Aragorn:~/dpoo/TareaN1$ git fetch
From git.elo.utfsm.cl:matias.hernandez/Garrido.Cristobal-Hernandez.Matias-Leiva.Ulises-TareaN1
 c242dfa..7c1d51c master -> origin/master
matias.hernandez@Aragorn:~/dpoo/TareaN1$ git log origin/master
commit 7c1d51ccfc0f051e2d7b879c6756ab2c3d6a0dea
Author: Cristobal Garrido <cristobal.garridov@Aragorn.elo.utfsm.cl>
Date: Mon Apr 15 19:53:16 2019 -0300

    TestStage4 Controlador terminado y nueva clase para Imprimir

commit def09472db88d44484b56ca713909f63420f549f
Author: Cristobal Garrido <cristobal.garridov@Aragorn.elo.utfsm.cl>
Date: Sun Apr 14 13:11:43 2019 -0300

    TestStage3 Corrección de Semaforo Giro

commit 748dc78f00ccfc205c1120935ee1b9da70754df2
Author: ULISES EUGENIO LEIVA BAHAMONDES <ulises.leiva@sansano.usm.cl>
Date: Wed Apr 10 22:11:45 2019 -0300

    TestStage4 Implementacion de SimuladorEntradas (para tres entradas), Semaforo P, DetectorDeRequerimiento y avance de main

commit c242dfa549a3fbf3ce32b80944da78328cc50c2b5
Author: MATÍAS SALVADOR HERNÁNDEZ MATORANA <matias.hernandez@sansano.usm.cl>
Date: Wed Apr 10 22:02:23 2019 -0300

    TestStage3 completados metodos y main

commit 7ed2c665750448e79577241fdda048688cdd49a2
Author: ULISES EUGENIO LEIVA BAHAMONDES <ulises.leiva@sansano.usm.cl>
Date: Wed Apr 10 20:49:12 2019 -0300

    TestStage2 modificacion manageTraffic (implementacion de sleeps en delays)

commit 0528cbacbe8c0acefe1ebddd4df79b7922950de
Author: Cristobal Garrido <cristobal.garridov@Aragorn.elo.utfsm.cl>

```

Figura 2: Resultado obtenido del comando git log.

En la imagen se aprecia los últimos commits que se han realizado.

```

matias.hernandez@Aragorn:~/dpoo/TareaN1$ git diff
diff --git a/TestStage4/TestStage4.java b/TestStage4/TestStage4.java
index c5535be..d8c4c76 100644
--- a/TestStage4/TestStage4.java
+++ b/TestStage4/TestStage4.java
@@ -708,7 +708,7 @@ class SimuladorEntradas
     private File temporal;
     private Scanner archivo;
 }
-
+// La siguiente clase fue creada para imprimir de forma paralela los estados de los semaforos
class Imprimir
{
    public Imprimir(SemaforoP placeres, SemaforoP matta, SemaforoDeGiro giro, Semaforo3 Placeres Valp, Semaforo3 Placeres Vina, Sema

```

Figura 3: Resultado obtenido del comando git diff.

Se muestran las líneas añadidas en el código antes de que se haga el commit.

```

matias.hernandez@Aragorn:~/dpoo/TareaN1$ git show
commit 7c1d51ccfc0f051e2d7b879c6756ab2c3d6a0dea
Author: Cristobal Garrido <crisobal.garridov@Aragorn.elo.utfsm.cl>
Date:   Mon Apr 15 19:53:16 2019 -0300

    TestStage4 Controlador terminado y nueva clase para Imprimir

diff --git a/TestStage1/Makefile b/TestStage1/Makefile
new file mode 100644
index 0000000..007cccc
--- /dev/null
+++ b/TestStage1/Makefile
@@ -0,0 +1,5 @@
+all:
+    javac TestStage1.java
+
+clean:
+    rm -f *.class
\ No newline at end of file
diff --git a/TestStage1/TestStage1.java b/TestStage1/TestStage1.java
index 7ea0eac..bd31b96 100644
--- a/TestStage1/TestStage1.java
+++ b/TestStage1/TestStage1.java
@@ -10,13 +10,13 @@ public class TestStage1
     while(i < iteracion)
     {
         semaforo_1.turnGreenLightOn();
-        System.out.println(tiempo + " " + semaforo_1);
+        System.out.println(tiempo + " " + semaforo_1);
         tiempo += semaforo_1.getGreenTime();
         semaforo_1.turnYellowLightOn();
-        System.out.println(tiempo + " " + semaforo_1);
+        System.out.println(tiempo + " " + semaforo_1);
         tiempo += semaforo_1.getYellowLightTime();
         semaforo_1.turnRedLightOn();
-        System.out.println(tiempo + " " + semaforo_1);
+        System.out.println(tiempo + " " + semaforo_1);
     }

```

Figura 4: Resultado obtenido del comando git show.

Este comando muestra todos los cambios realizados en el ultimo commit, es por esa razón que solo enseñamos las primeras líneas de respuesta de este comando, debido a la gran cantidad de cambios que enseña.

```

matias.hernandez@Aragorn:~/dpoo/TareaN1$ git status
# On branch master
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       readme.txt
nothing added to commit but untracked files present (use "git add" to track)
matias.hernandez@Aragorn:~/dpoo/TareaN1$ █

```

Figura 5: Resultado obtenido del comando git status.

Se muestra el estado del repositorio y los cambios que se han hecho y/o los archivos añadidos antes del commit.