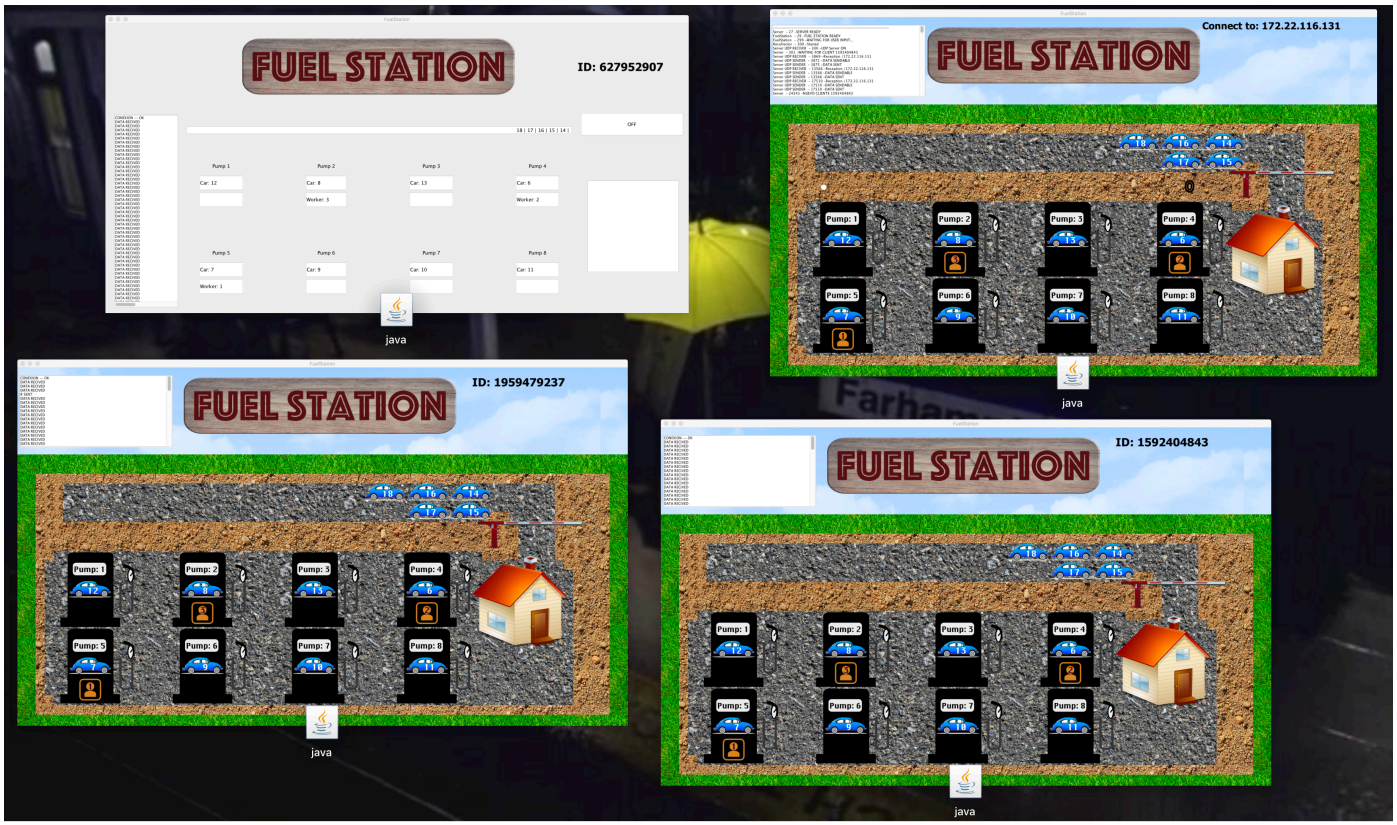
PECL3

SIMULACIÓN DEL FUNCIONAMIENTO DE UNA GASOLINERA

Juan Casado Ballesteros 0908762A
GII Programación Avanzada
Laboratorio 08:00 a 12:00

- Antes de ejecutar leer el archivo README.txt
- Los diagramas UML se adjuntan para poder ser visualizados a un tamaño adecuado



Varios clientes dos con interface gráfica y uno con interface textual conectados a un mismo servidor (superior derecha) mientras que ejecutan el programa (estado pausa).

Índice

Descripción de las clases

Clases de concurrencia

Car

Esta clase que hereda de Thread se corresponde con los coches de la gasolinera, está diseñada para ser ejecutada dentro de un pool de hilos en la clase FuelStation.

Su comportamiento es el siguiente:

- Toma un surtidor de la cola de surtidores libres ordenado por el número de surtidor de menor a mayor.
- Si no hubiera un surtidor espera en orden de llegada de los coches a que uno sea liberado.
- Entra en el surtidor y lo coloca en la cola de surtidores con coche.
- Espera a que llegue un trabajador y le sirva.
- Sale de la gasolinera.

Todas las fases de su comportamiento quedarán notificadas en un archivo de log propio y en el log común de toda las clases. Utiliza un semáforo fair para controlar que los coches tomen los surtidores de forma ordenada. A bajo nivel su comportamiento es más sencillo, toma objetos Pump de una cola en la que se ordenan por su número y los deja en otra en la que se ordenan por orden de llegada y espera a ser notificado.

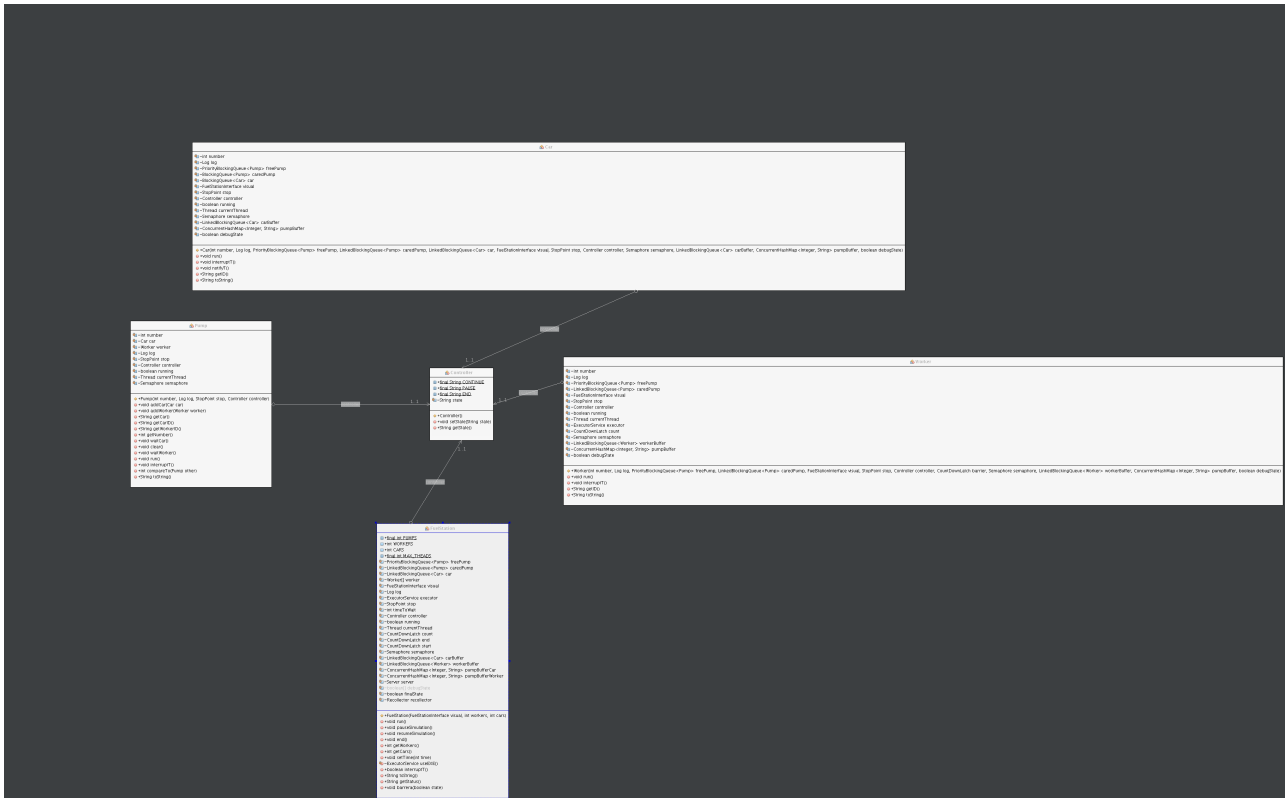
Ya que se va a ejecutar dentro de un pool de hilos tiene dos métodos para poderlo seguir interrumpiendo y notificando a pesar de no tener la referencia directa de su objeto Runnable de forma externa.

Controller

Esta clase se utiliza para descodificar lo que las interrupciones implican y también para indicar el estado de la ejecución de la gasolinera, tiene tres estados: PAUSA, FINALIZADO, CONTINUE.

Cuando una clase es interrumpida mira el estado de esta variable para saber por qué se ha producido la interrupción, también es comprobada en algunos puntos críticos de programa.

Vemos como Car, Worker, Pump y FuelStation que son los cuatro hilos principales que animan la gasolinera toan un objeto Controller compartido por todos ellos.



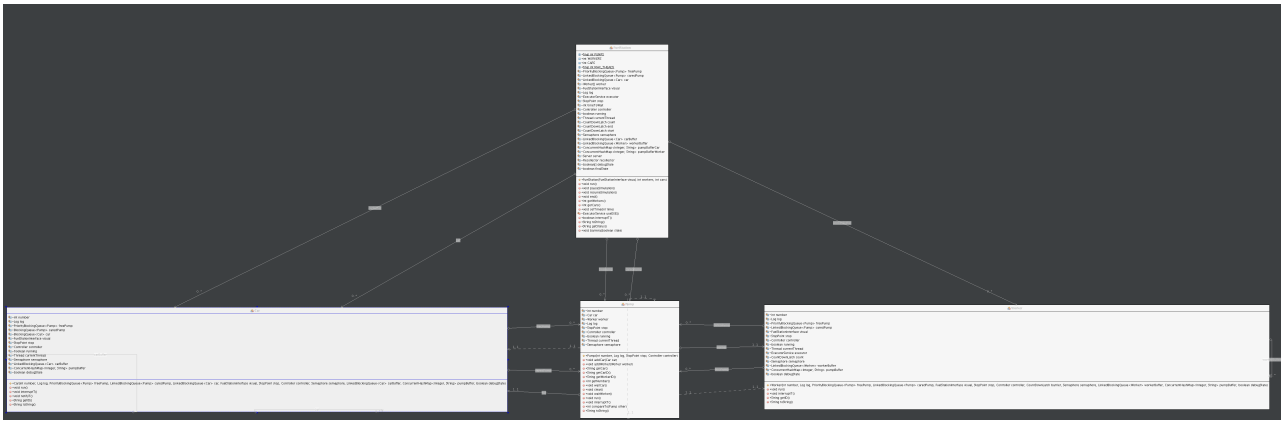
UML: CONTROLLER.png

FuelStation

Esta es la clase principal del programa, su función es iniciar todas las demás clases que participan de la simulación de la gasolinera. Para hacerlo tiene un hilo que inicia el servidor, los trabajadores y los surtidores. Posteriormente mediante un pool de hilos introduce los coches cada cierto tiempo, es ajustable mediante métodos.
Una vez hecho todo esto espera el fin de las clases iniciadas y el desalojo de los recursos que ocupen.
La simulación puede ser finalizada, parada o resumida en todo momento y de forma segura.

El funcionamiento de la gasolinera se basa principalmente en dos colas, una que ordena los surtidores por su número y de la que los coches extraen los surtidores para colocarlos en otra cola donde se ordenan por llegada, de esa cola será de la que los trabajadores extraigan los surtidores en los que hay un coche, entonces el surtidor hace esperar a ambos un tiempo y finalmente el trabajador deja el surtidor libre en la cola inicial.
Los coches serán tareas de un pool de hilos y los trabajadores hilos en si mismos.

Esta clase puede ejecutarse en dos modos, uno activo en el que los cambios producidos en el estado de la gasolinera son puestos al momento sobre la interface gráfica. Al ver que esto producía un cuello de botella fue creado el modo pasivo de ejecución. El modo pasivo establece un productor consumidor entra la interface gráfica y las clases de negocio a partir de los buffers de esto, de este modo la interface se actualiza de forma pasiva agilizando la ejecución.
En términos de rendimiento esto no era algo necesario pero fue realizado para preservar la concurrencia y el liveness.



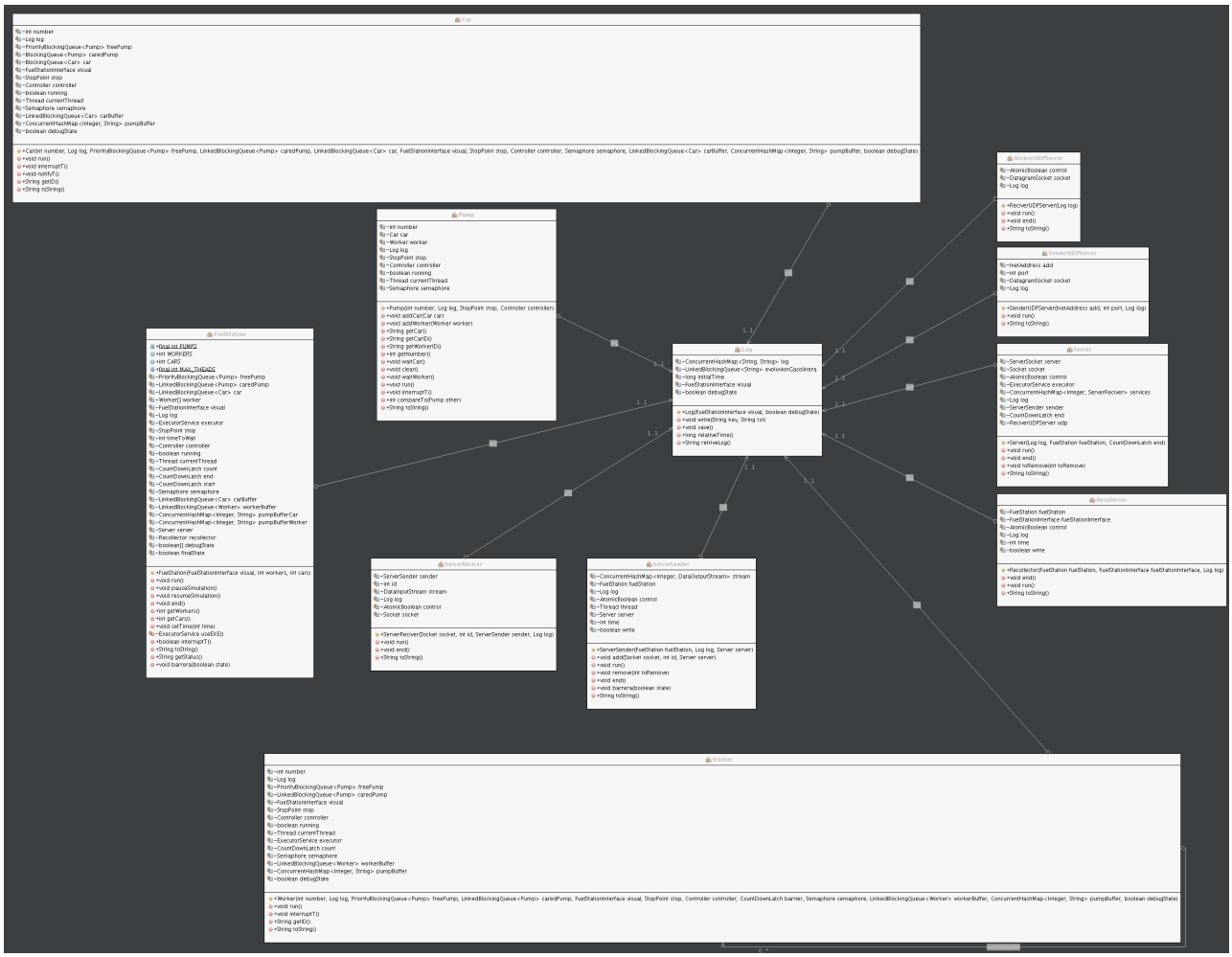
UML: FUEL_STATION.png

Observando el diagrama UML vemos como una de las principales motivaciones del diseño ha sido que Coches y Trabajadores no dependieran los unos de los otros y solo se relaciones mediante los surtidores, podemos ver también como los surtidores existen mediante las dos colas citadas.

Log

Esta clase es la encargada de crear, guardar y gestionar los metadatos producidos por la aplicación.
Su funcionamiento interno consta de un ConcurrentHashMap sobre el que cada clase al identificarse puede escribir sus datos, los cuales luego se guardarán cuando la simulación finalice. Adicionalmente esos datos son guardados también en una LinkedBlockingQueue, se utilizan estos dos buffers para permitir el guardado concurrente por varias clases a la vez.
El guardado se hace una sola vez al finalizar la simulación para mejorar la eficiencia.

Podemos ver que clases utilizan el log para guardar sus metadatos. Estos principalmente consistirán de los eventos relevantes que se han producido en dicha clase a lo largo de su vida para luego poder trazar su ejecución.

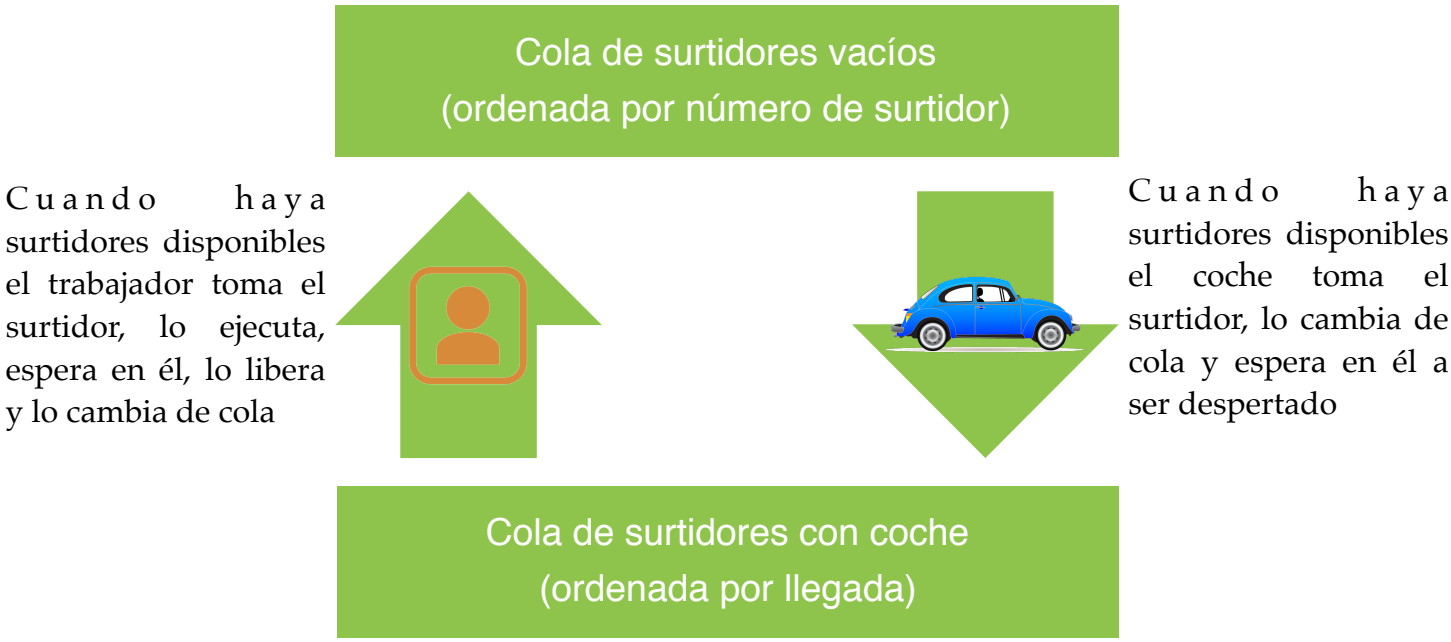


UML: LOG.png

Pump

Los surtidores son la clase que relacionan entre si a los trabajadores con los coches, en el diseño de la gasolinera se pretende evitar a toda costa que esa relación se realice de forma directa. Por comodidad y debido a como ha sido realizado el diseño al surtidor se le ha dotado de un papel activo, es decir, es el que sincroniza el repostaje.
Tiene métodos para interrumpir la espera que realiza como hilo ya que está pensado para ser ejecutado dentro de un pool de hilos que contiene el trabajador. También tiene métodos para que los coches y trabajadores puedan entrar en él, para vaciarse y para hacerlos esperar.
La ejecución de su hilo es sencilla y únicamente realiza un sleep tras el cual se despertará al hilo y al coche que en él estén esperando, posteriormente se vaciará.

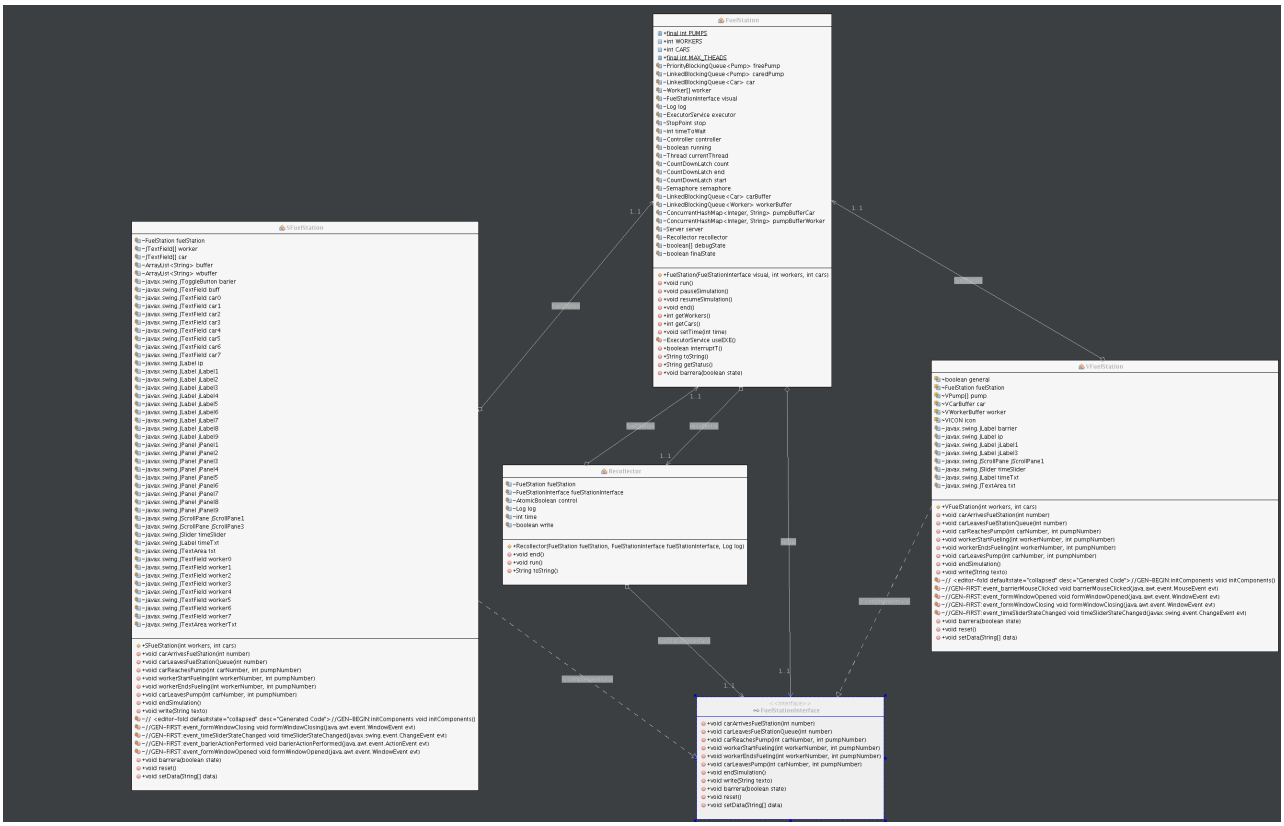
El papel principal del surtidor no obstante no es su hilo si no el cambio de colas que trabajadores y coches realizan sobre él. Los surtidores comienzan en una cola de surtidores vacíos que se ordena por el número de surtidor (los surtidores implementan Comparable), de esta cola los coches los toman y los introducen en otra donde se ordenan por llegada. De esa ultima cola los trabajadores los tomarán y ejecutarán su run para hacerse esperar a si mismos, cuando termine su ejecución trabajador y coche despertarán y el trabajador volverá a dejar el surtidor en la cola de surtidores vacíos.



Recollector

Esta clase es la encargada de realizar la actualización en modo pasivo de la interface gráfica. Consta de un hilo que realiza un productor consumidor entre la gasolinera y la y esta utilizando las colas de recogida de datos como buffers de modo que la gasolinera no tenga que actualizarla de forma activa.
Gracias a esta clase aumenta en gran medida el rendimiento de la gasolinera ya que tampoco es necesario una actualización en tiempo real, no obstante se puede configurar para que lo sea si esta se ejecuta en modo activo.

Podemos observar como se relacionan las clases gráficas con la gasolinera siendo esta el único nexo entre la interface y las clases de negocio por medio de Recollector que toma sus datos y los muestra. Vemos también como múltiples interfaces gráficas podrán ser utilizadas por el programa siempre y cuando implementen de FuelStationInterface



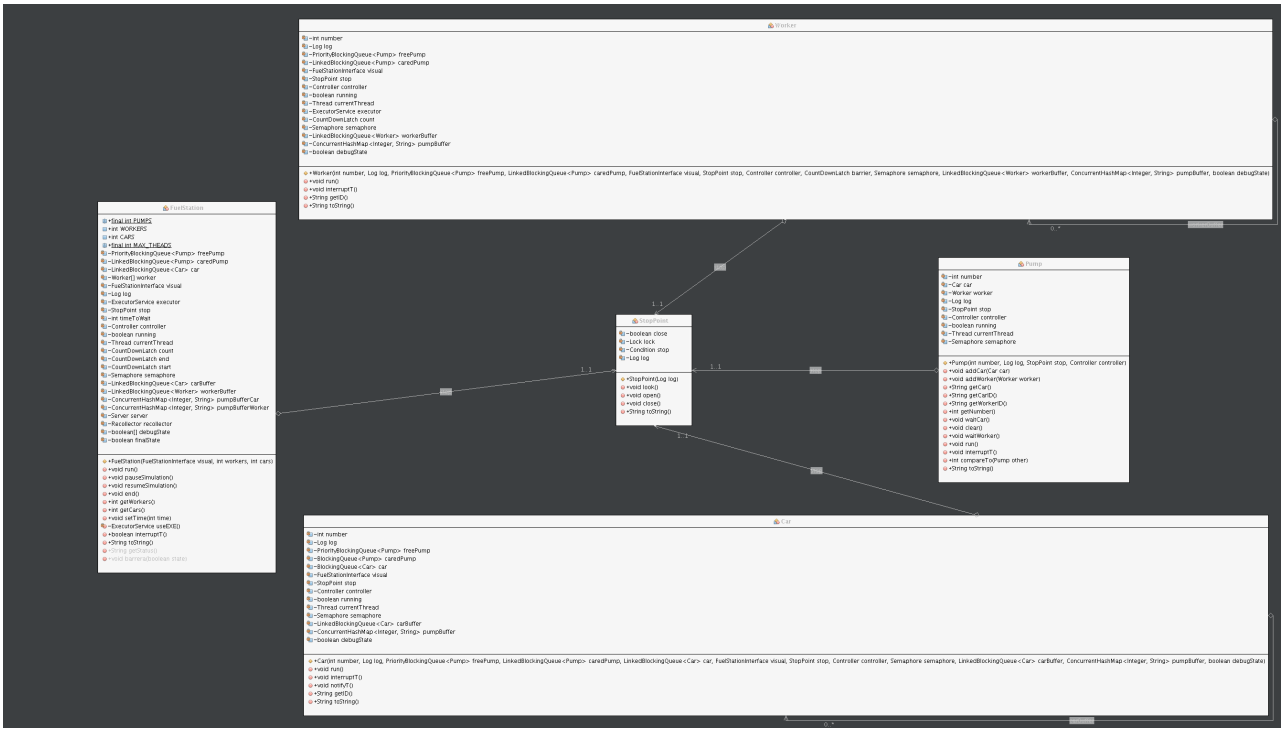
UML: RECOLLECTOR.png

StopPoint

Esta clase es utilizada para que los hilos que la observen se queden dormidos siempre y cuando esa acción esté activada. De no estarlo la observarán y ninguna acción adicional será realizada.
Con respecto a la PECL1 y PECL2, en las que también se utiliza esta misma clase, se ha realizado una modificación en la forma de utilizarla. Para evitar un polling constante sobre ella por parte de las demás clases que la compartan. Ahora esta clase solo será mirada cuando alguna interrupción sea realizada en alguno de los hilos en los que sea necesario. Esto es así pues por probabilidad es en las pausas en los puntos en los que es más fácil encontrar un hilo y si aprovechamos que estas son interrumpirles podremos asegurarnos de que solo se realizará el polling sobre esta clase cuando sea necesario y no en ningún otro momento.

La reanudación de la ejecución también ha sido modificada parlamentar el realismo. Ahora tras pausar la gasolinera esta no reanudará después del sleep que los hilos estuvieran realizando sin más, si no que volverá a ejecutar de nuevo el sleep que hubiera sido interrumpido .

Vemos las clases que comparte una misma instancia de StopPoint



UML: STOP_POINT.png

Worker

Esta clase contiene un hilo que modela el comportamiento de trabajador de la siguiente forma:

- Toma un surtidor de la cola de surtidores con coche ordenada por el orden de entrada a la cola.
- Si no hubiera un surtidor espera en orden a que uno sea liberado.
- Entra en el surtidor y lo ejecuta desde el pool de hilos que tiene el trabajador.
- Cuando es despertado introduce el surtidor que había tomado en la cola de surtidores vacíos.
- Vuelve a realizar todos los pasos hasta que no haya más coches que servir.

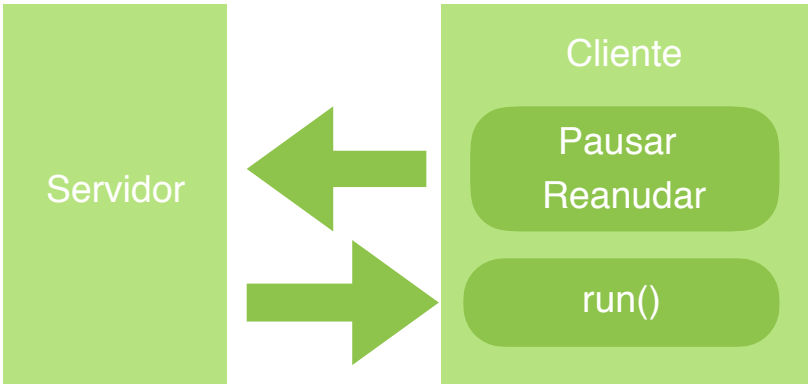
Adicionalmente tiene atributos que comparte con las otras clases que participan de la simulación como las colas de surtidores, semáforos para sincronizarse con los coches, un log para poder trazar su ejecución, clases adicionales para poder pausar la simulación y buffers para realizar la actualización pasiva de la interface gráfica.

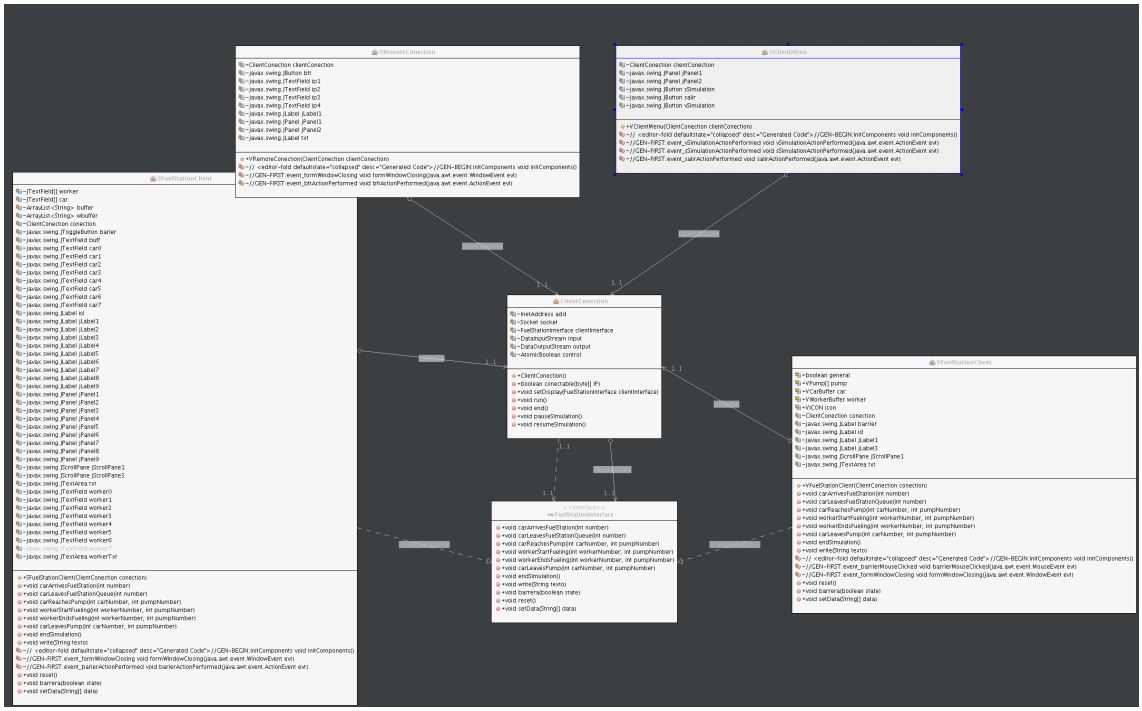
Clases de comunicación

ClientConection

Esta clase es la que los clientes utilizarán para crear conexiones TCP implementadas sobre sockets para comunicarse con el servidor.
Contiene un hilo cuya función es escuchar al servidor hasta que de este reciba una “X” lo cual significa que la comunicación debe terminar. Una vez recibidos los datos del Servidor los mostraría por pantalla.
Tiene un método que pasada una IP dice si esa dirección es alcanzable, de no serlo no se procedería al establecimiento de la conexión y de serlo lanzará un menú para que el usuario elija el tipos de visualización de datos que desea.
Tiene otro método que finaliza el cliente, tanto su hilo como los canales y sockets que esté ocupando, si de detectan errores en la comunicación también finalizará.
Adicionalmente los clientes no solo recibirán datos si no que también podrán enviarlos. Concretamente los clientes podrán notificar al servidor para que este pause o reanude la comunicación.

Vemos como la client conecto toma información del exterior para realizar la conexión y luego actúa como si fuera la gasolinera en si misma comunicándose con las interfaces gráficas adoptando su mismo lugar el la arquitectura.





UML: CLIENT_CONECTION.png

ReciverUDPClient

Esta clase pretende recibir un paquete UDP del Servidor para obtener la dirección IP a la que conectarse mediante TCP con la ClientConnection.
Utiliza un datagram socket para escuchar con un timeout, si hubiera algún error se daría por imposible la conexión y se intentaría realizar de forma manual, es decir el usuario tendría que introducirla manualmente.
En concreto esta clase solo realiza la parte de recepción del paquete, actuando en conjunto con otras termina realizando toda la funcionalidad explicada, es lanzada por SenderUDPClient en su pool de hilos.

ReciverUDPServer

Es un servidor UDP que se integra, es lanzado y controlado, en la clase Server. Escucha peticiones UDP de forma continua en un hilo, si recibe alguna lanzará otro hilo mediante una pool de hilo único que intentará comunicarse con la dirección de la que vino el paquete.

SenderUDPClient

Intenta hasta tres veces conectarse con el Servidor UDP mediante paquetes de broadcast, para ello envía dichos paquetes lanzando en un pool de hilo único receptores que escuchen los envíos del Servidor. Si estos no se producen se dará por inviable la obtención de la IP del Servidor por este método y se procederá a la introducción de la IP de forma manual.

SenderUDPServer

Es lanzado en un pool por ReciverUDPServer, su función es lanzar un paquete UDP hacia el Cliente que intentó conectarse con el Servidor, de este modo dicho Cliente podrá obtener la IP del Servidor.

En cierto modo estas clase buscan hacer un DNS “mal”, ya que el servidor no es público ni aparece en el DNS se sustituye por el de broadcast que por lo general no funcionará en una red como la de la UAH, no obstante de forma local en un mismo equipo o en una red que si retransmita los broadcasts si funcionaría. Debido a todas estas limitaciones por lo que en caso de no poderse realizar la conexión automática se permite la manual.
En la interface del Servidor se dirá la IP a la que conectarnos y en la del Cliente aparecerá dónde introducirla.

Server

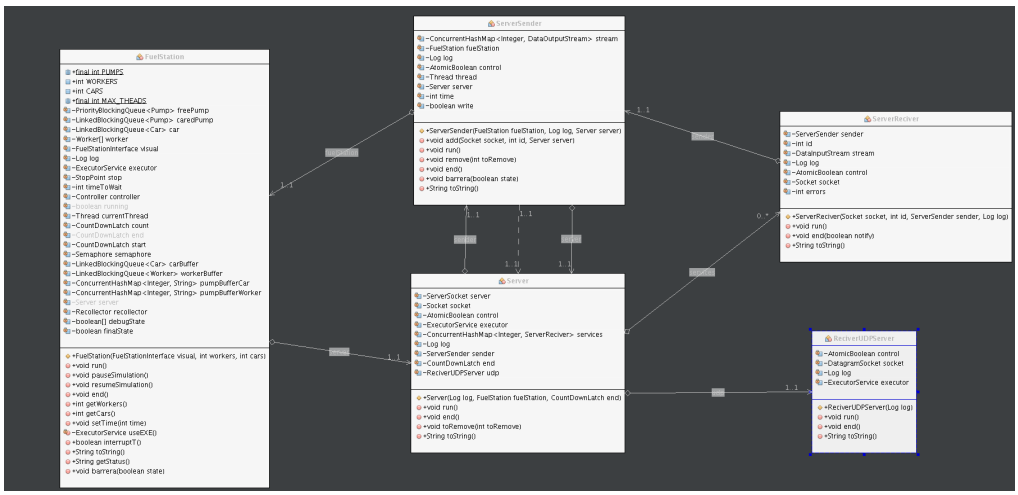
Esta clase tiene dos funcionalidades, por un lado controla el servidor UDP, su inicio y fin controlados y por otro admite conexiones TCP.
En un hilo y sobre un puerto escucha la llegada de las peticiones de conexión, cuando lo hacen lanza sobre un pool de hilos un receptor TCP el cual también almacenará en una lista para llevar un control sobre ellos, al emisor TCP lo añadirá a un ServerSender que controla la lista de equipos a los que hay que dar servicio.
A cada socket TCP abet le asigna una identidad única utilizada para reconocer qué equipo es del que recibimos o al que enviamos.

ServerReciver

A partir de cada petición se crea un socket TCP, esta clase es la encargada de abrir un canal de recepción sobre dicho socket y escucharlo en un hilo. Esta escucha se utiliza para parar o reanudar la simulación así como para saber cuando un cliente se desconecta de forma adecuada. Ya que cada socket está identificado de forma única podremos saber quien nos envió los datos.
También tiene un método para cerrar el canal de forma segura así com el socket y un modo de detección de fallos que eliminaría al Cliente por completo del servidor, si se producen tres fallos de recepción seguidos lo elimina.

ServerSender

Esta clase consiste en un servicio de envío al cual se pueden subscribir sockets TCP a los que se quiera enviar información de la gasolinera. Con un ConcurrentHasMap que contiene los canales de emisión identificados cae forma única cada segundo recogemos los datos procesados de los buffers de la gasolinera y se los enviamos a todos los clientes.



UML: SERVER.png

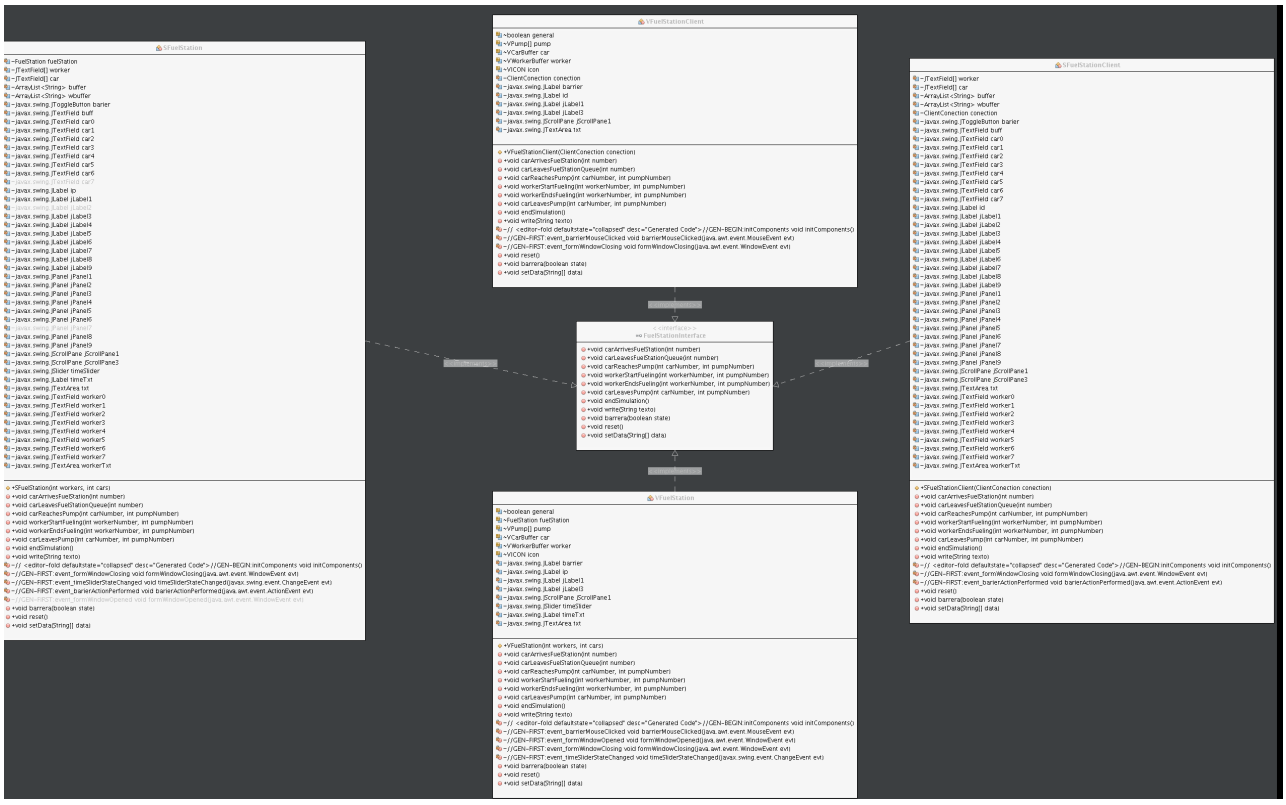
Clases Gráficas

FuelStationInterface

Esta interface es implementada por las cuatro clases gráficas que muestran los datos de la gasolinera, permitiendo que cualquier clase que quiera se pueda comunicar con cualquiera de ellas de forma idéntica instanciando la interface y no la clase.
La interface admite ambos modos de ejecución de la gasolinera, modo activo y pasivo. En el modo activo se dice directamente qué acciones debe la gasolinera realizar, dónde dibujar cada trabajador y cada empleado. Por el contrario para el modo pasivo se recurre a simplificar todos estos datos en un Array de String codificado de la siguiente forma:

- Se dice el número de coches en el buffer
- Se escribe el ID de cada coche del buffer
- Se dice el número de trabajadores descansando
- Se escriben los ID de los trabajadores que descansan.
- Por cada surtidor, siempre ocho se envía el ID del coche y del trabajador que estén en él o “N” si de alguno no hay, esto se envía de forma ordenada de modo que no haya que identificar a cada surtidor por su número si no que se haga por la posición de los datos.
- Número de identidad del Cliente en el servidor.

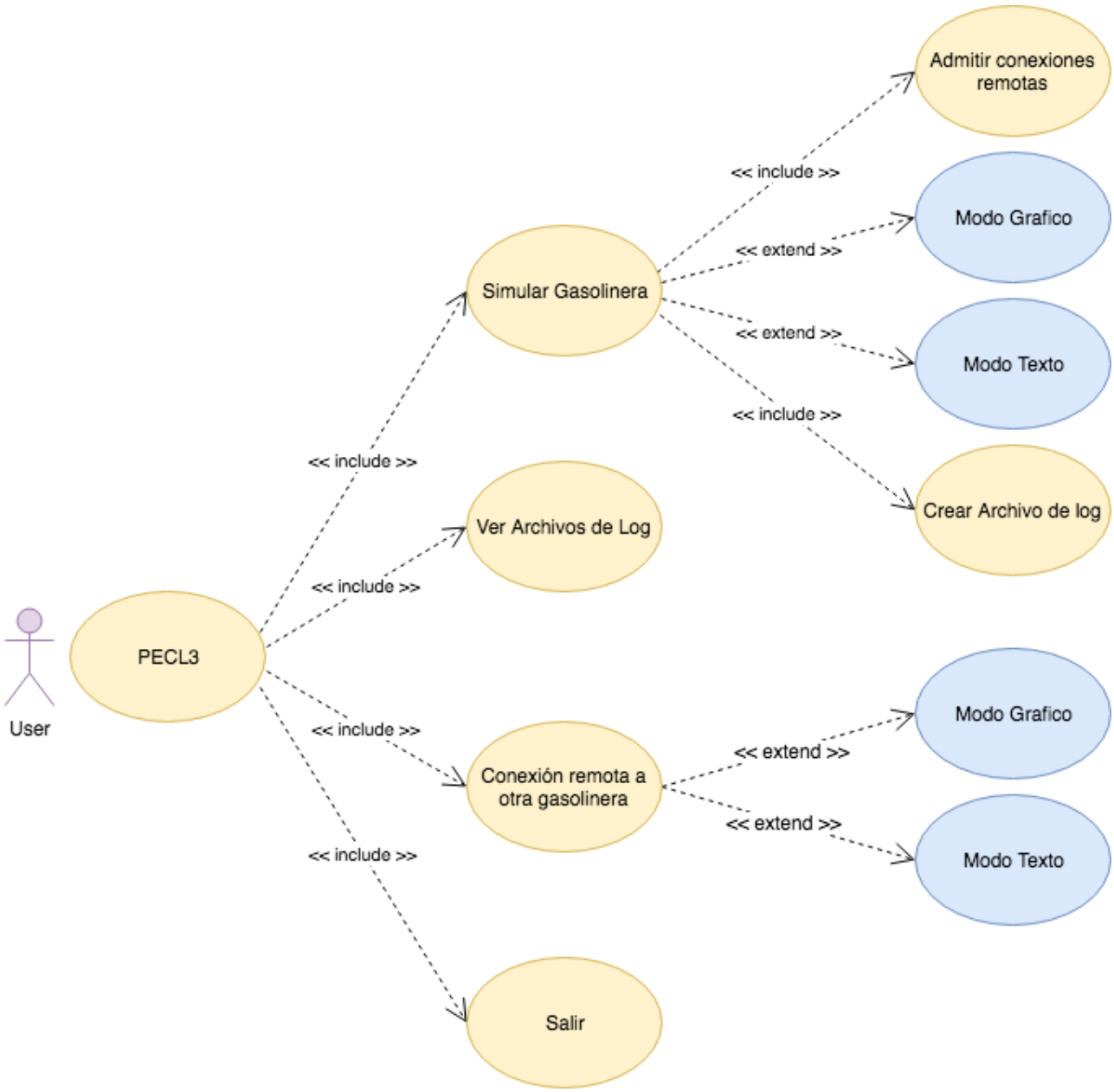
Hemos dicho que hay cuatro clases que implementan la interface, dos son textuales una para el cliente y otra para el servidor y dos son gráficas.



UML: IMPLEMENT.png

Acciones de la interface

Cuando entramos al programa tendremos la opción de entrar en una simulación de la gasolinera en el modo gráfico o textual, conectarnos a un servidor ya abierto, ver el Log de la simulación anterior, elegir el número de trabajadores y coches para nuestra siguiente simulación o salir.



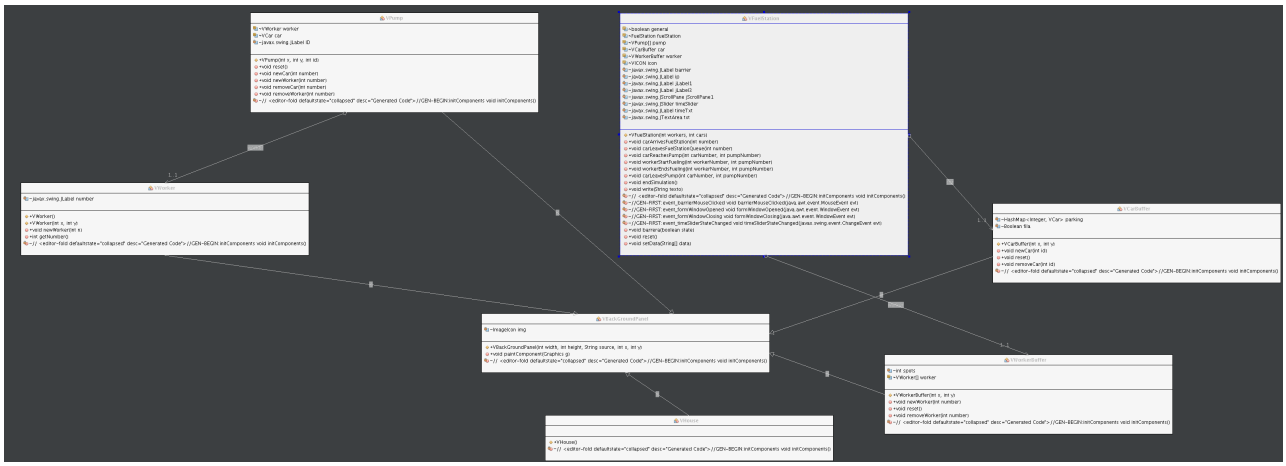
UML: CASOS_DE_USO.png

Si entramos en el modo Online primero intentaremos conectarnos de forma automática al Servidor por medio de UDP en broadcast para obtener su IP, de no ser posible pasaremos a un menú en el que podremos introducir de forma sencilla la IP del Servidor, dicha IP la podremos visualizar en su interface gráfica.

Desde el modo de visualización del log tendremos una interface que nos permitirá navegar por los archivos de metadatos que el programa crea y desde los cuales se puede trazar su ejecución se creará uno por cada clase que quiera volcar datos y otro común a todas. En esta clase se utilizan como Apollo dos clases llamadas Sorter y Sortable que ordenan los archivos por su nombre para que a la hora de buscarlos se más sencillo.

Para pasar de unos menú a otros cuando ese paso pueda tardar algo más de lo debido porque haya que lanzar múltiples hilos y espera a que finalicen nos ayudamos de la clase VLoad que se ayuda del hilo LoadController para de forma concurrente a la carga de datos mostrar una barra de carga.

Para crear la interface gráfica extiendo de JPanel para crea un nievo tipo de panel en el que poner fotos de fondo, de este heredarán el resto de clases visuales.



UML: PANEL.png

Anexo de código

Clases de Concurrencia

Car

```
package pecl3.src;

import java.util.NoSuchElementException;
import java.util.concurrent.BlockingQueue;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.LinkedBlockingQueue;
import java.util.concurrent.PriorityBlockingQueue;
import java.util.concurrent.Semaphore;
import pecl3.screens.FuelStationInterface;

/**
 *
 * @author mr.blissfulgrin
 */
public class Car extends Thread
{
    private final int number;
    private final Log log;
    private final PriorityBlockingQueue <Pump> freePump;    //PRIORITY LIST
    private final BlockingQueue <Pump> caredPump;          //QUEUE
    private final BlockingQueue <Car> car;                 //QUEUE
    private final FuelStationInterface visual;
    private final StopPoint stop;
    private final Controller controller;
    private boolean running;
    private Thread currentThread;
    private final Semaphore semaphore;
    private final LinkedBlockingQueue <Car> carBuffer;
    private final ConcurrentHashMap <Integer,String> pumpBuffer;
    private final boolean debugState;

        public Car (int number, Log log, PriorityBlockingQueue <Pump> freePump,
LinkedBlockingQueue <Pump> caredPump,
                LinkedBlockingQueue <Car> car, FuelStationInterface visual, StopPoint stop,
Controller controller,
```

```
                                        Semaphore semaphore,LinkedBlockingQueue <Car>
carBuffer,ConcurrentHashMap <Integer,String> pumpBuffer,boolean debugState)
   {
      this.number = number;
      this.log = log;
      this.freePump = freePump;
      this.caredPump = caredPump;
      this.car = car;
      this.visual = visual;
      this.stop = stop;
      this.controller = controller;
      this.running = false;
      this.semaphore = semaphore;
      this.carBuffer = carBuffer;
      this.pumpBuffer = pumpBuffer;
      this.debugState = debugState;
   }

    /**
     * Funcionamiento de un coche:
     *  · Toma un surtidor de la cola de surtidores libres ordenado por el número de surtidor
de menor a mayor
      *  · Si no hubiera un surtidor espera en orden de llegada de los coches a que uno sea
liberado
     *  · Entra en el surtidor y lo coloca en la cola de surtidores con coche
     *  · Espera a que llegue un trabajador y le sirva
     *  · Sale de la gasolinera
     */
   @Override
   public void run()
   {
      currentThread = Thread.currentThread();
      this.running = true;
      boolean repeatSleep;
      boolean retakePump;
      log.write(this.toString(), "Ha llegado a la gasolinera");
      if (debugState)
         visual.carArrivesFuelStation(number);

      do
      {
         retakePump = false;
         //Whait for a free Pump
         do
```

```
        {
            retakePump = false;
            try
            {
                log.write(this.toString(), "Entra al parking");
                if (semaphore.availablePermits() == 0)
                {
                    log.write(this.toString(), "Los surtidores están llenos");
                }
                if (!carBuffer.contains(this))
                    carBuffer.add(this);
                semaphore.acquire();
                carBuffer.remove(this);
                log.write(this.toString(), "Hay un surtidor libre");

            }
            catch (InterruptedException ex)
            {
                    log.write(this.toString(), "INTERRUPTION WHILE WAITING FOR FREE
PUM");
                switch (controller.getState())
                {
                    case Controller.CONTINUE:
                        log.write(this.toString(), "MISS MATCH");
                        break;
                    case Controller.PAUSE:
                        log.write(this.toString(), "CAR PAUSED");
                        stop.look();
                        retakePump = true;
                        log.write(this.toString(), "CAR RESUMED");
                        break;
                    case Controller.END:
                        log.write(this.toString(), "CAR FORCED END");
                        break;
                    default:
                        log.write(this.toString(), "CONTROLLER BADLY USED");
                        break;
                }
            }
        }
                                    while ((freePump.isEmpty() || retakePump) && !
controller.getState().equals(Controller.END));
```

```
if (!controller.getState().equals(Controller.END))
{

   try
   {
      //Adquire a Pump
      Pump pump = freePump.remove();
      pumpBuffer.put(pump.getNumber(), this.getID());
      log.write(this.toString(), "Surtidor libre adquirido");
      if (debugState)
      {
         visual.carLeavesFuelStationQueue(number);
         visual.carReachesPump(number, pump.getNumber());
      }

      pump.addCar(this);
      caredPump.add(pump);
      synchronized (caredPump)
      {
         caredPump.notify();
      }

      log.write(this.toString(), "Espera en el " + pump.toString());

      //Wait for a Worker
      do
      {
         repeatSleep = false;
         try
         {
            pump.waitCar();
            if (debugState)
               visual.carLeavesPump(number, pump.getNumber());
            pumpBuffer.remove(pump.getNumber());
            log.write(this.toString(), "Ha terminado de repostar");
         }
         catch (InterruptedException i)
         {
                  log.write(this.toString(), "INTERRUPTION WHILE WAITING FOR A
WORKER");
            switch (controller.getState())
            {
               case Controller.CONTINUE:
```

```
                        log.write(this.toString(), "MISS MATCH");
                        break;
                    case Controller.PAUSE:
                        log.write(this.toString(), "CAR PAUSED");
                        repeatSleep = true;
                        stop.look();
                        log.write(this.toString(), "CAR RESUMED");
                        break;
                    case Controller.END:
                        log.write(this.toString(), "CAR FORCED END");
                        break;
                    default:
                        log.write(this.toString(), "CONTROLLER BADLY USED");
                        break;
                }
              }
            }
             while (repeatSleep);

        }
        catch (NoSuchElementException n)
        {
            log.write(this.toString(), "FREE_PUMP WAS EMPTY");
            retakePump = !controller.getState().equals(Controller.END);
        }
      }
   }
   while(retakePump);
   //The Car lefts the FuelStation
   stop.look();
   car.remove(this);
   log.write(this.toString(), "Abandona la gasolinera");
   this.running = false;
}

/**
 * Permite interrumpir al Coche cuando el pool de hilos lo esté ejecutando
 */
public synchronized void interruptT ()
{
   if (running)
   {
      currentThread.interrupt();
   }
```

```
    }

    /**
     * Permite realizar un notify sobre el cCoche cuando esté siendo ejecutado por el pool
     */
    public void notifyT ()
    {
        log.write(this.toString(), "NOTIFY DONE");
        synchronized (currentThread)
        {
            if (running)
            {
                currentThread.notify();
            }
        }
    }

    /**
     * Retrieves the Car ID (used for connection)
     * @return String ID
     */
    public String getID ()
    {
        return String.valueOf(number);
    }
    /**
     * Retrieves the textual representation of the car (used for graphics)
     * @return String CAR ID
     */
    @Override
    public String toString()
    {
        return "CAR " + number;
    }
}
```

Controller

```
package pecl3.src;

/**
 *
 * @author mr.blissfulgrin
 */
public class Controller
{
    //VALORES RECONOCIDOS POR EL CONTROLADOR
    public static final String CONTINUE = "CONTINUE";
    public static final String PAUSE = "PAUSE";
    public static final String END = "END";

    private String state;

    public Controller ()
    {
        state = CONTINUE;
    }

    /**
     * Aplica un estado al controlador
     * @param state SOLO SON VALIDOS LOS ESTADOS PROPIOS
     */
    public synchronized void setState (String state)
    {
        switch (state)
        {
            case CONTINUE:
                this.state = CONTINUE;
            break;
            case PAUSE:
                this.state = PAUSE;
            break;
            case END:
                this.state = END;
            break;
        }
    }

    /**
```

```
     * Retorna el estado del Controlador
     * @return state
     */
    public synchronized String getState ()
    {
        return state;
    }
}
```

FuelStation

```
package pecl3.src;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.CountDownLatch;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.LinkedBlockingQueue;
import java.util.concurrent.PriorityBlockingQueue;
import java.util.concurrent.RejectedExecutionException;
import java.util.concurrent.Semaphore;
import java.util.concurrent.TimeUnit;
import pecl3.src.conectivity.Server;
import pecl3.screens.FuelStationInterface;

/**
 *
 * @author mr.blissfulgrin
 */
public class FuelStation extends Thread
{
    public static final int PUMPS = 8;
    public final int WORKERS;
    public final int CARS;
    public static final int MAX_THEADS = 28;
    private final PriorityBlockingQueue <Pump> freePump;  //PRIORITY LIST
    private final LinkedBlockingQueue <Pump> caredPump;   //QUEUE
    private final LinkedBlockingQueue <Car> car;          //QUEUE
    private final Worker[] worker;                        //QUEUE
    private final FuelStationInterface visual;
    private final Log log;
    private final ExecutorService executor;
    private final StopPoint stop;
    private int timeToWait;
    private Controller controller;
    private boolean running;
    private Thread currentThread;
    private final CountDownLatch count;
    private final CountDownLatch end;
    private final CountDownLatch start;
```

```
    private final Semaphore semaphore;
    private final LinkedBlockingQueue <Car> carBuffer;
    private final LinkedBlockingQueue <Worker> workerBuffer;
    private final ConcurrentHashMap <Integer,String> pumpBufferCar;
    private final ConcurrentHashMap <Integer,String> pumpBufferWorker;
    private final Server server;
    private final Recollector recollector;

    /*
    * Estos atributos indican el modo de transferencia de los datos de las clases de trabajo a la interface gráfica:
     * Si alguno de debugger está a true la clase correspondiente mostrará sus datos de forma activa
    * Si todos están en false el hilo Recollector actualizará los gráficos
    */
    private final boolean [] debugState;
    private final boolean finalState;

    /**
     * Crea una simulación de una gasolinera
      * Crea los trabajadores y coches necesarios así como los surtidores y el servidor para permitir conexiones remotas
     * @param visual a JFrame implementing FuelStationInterface
     * @param workers Número de trabajadores
     * @param cars  Número de coches
     */
    public FuelStation (FuelStationInterface visual, int workers, int cars)
    {
        debugState = new boolean []
        {
            false,  //Cars        0
            false,  //Workers     1
            false,  //Log         2
        };
        finalState = !(debugState [0] || debugState [1] || debugState [2]);

        this.visual = visual;
        this.log = new Log(visual,debugState[2]);
        this.recollector = new Recollector(this,visual,log);
        this.stop = new StopPoint(log);
        this.executor = Executors.newFixedThreadPool(MAX_THEADS);
        this.WORKERS = workers;
        this.CARS = cars;
        this.controller = new Controller();
```

```
controller.setState(Controller.PAUSE);
this.running = false;
this.count = new CountDownLatch(workers);
this.end = new CountDownLatch(2);
this.start = new CountDownLatch(1);
this.semaphore = new Semaphore (8,true);
timeToWait = 5500;

this.carBuffer = new LinkedBlockingQueue <>();
this.workerBuffer = new LinkedBlockingQueue <>();
this.pumpBufferCar = new ConcurrentHashMap <>();
this.pumpBufferWorker = new ConcurrentHashMap <>();
//Create the Pumps
freePump = new PriorityBlockingQueue<>();
caredPump = new LinkedBlockingQueue<>();
for (int i = 0; i < PUMPS; i++)
{
   try
   {
      freePump.offer(new Pump(i+1,log, stop, controller));
   }
   catch(ClassCastException c)
   {
      log.write(this.toString(), "PUMP NOT SORTABLE");
   }
}

//Create Queue of incoming Cars
car = new LinkedBlockingQueue<>();
for (int i = 0; i < CARS; i++)
{
                Car c = new Car (i+1,log, freePump, caredPump,car, visual, stop,
controller,semaphore,carBuffer,pumpBufferCar,debugState[0]);
   car.add(c);
}

//Create a Queue of Workers
worker = new Worker[WORKERS];
for (int i = 0; i < WORKERS; i++)
{
             worker[i] = new Worker(i+1,log, freePump, caredPump, visual, stop,
controller,count,semaphore,workerBuffer,pumpBufferWorker,debugState[1]);
}
```

```
    //Create Server
    server = new Server(log,this,end);
    log.write(this.toString(), "FUEL STATION READY");
}

/**
 * Este hilo inicia el servidor, los trabajadores, y el pool de hilos de coches.
 * Posteriormente espera a que terminen y lo cierra todo se forma correcta.
 */
@Override
public void run ()
{
    //Inicia el servidor
    server.start();
    currentThread = Thread.currentThread();
    this.running = true;
    if (finalState)
        recollector.start();

    log.write(this.toString(), "WAITING FOR USER INPUT...");
    try
    {
        start.await();
    }
    catch (InterruptedException e)
    {
        log.write(this.toString(), "INITIAL WAIT INTERRUPTED");
    }

    log.write(this.toString(), "FUEL STATION SIMULATION STARTS");

    if (controller.getState().equals(Controller.CONTINUE))
    {
        //Start Workers
        for (Worker w : worker)
        {
            w.start();
        }

        //Start Cars
        boolean repeatSleep;
        int elapsedTime;
        for (Car c : car)
        {
```

```
do
{
   repeatSleep = false;
   elapsedTime = (int)(Math.random()*timeToWait+500);
   log.write(this.toString(), c.toString()+" llegará en "+elapsedTime+" ms");
   try
   {
      //Wait for arrival
      Thread.sleep(elapsedTime);
      try
      {
         if (!controller.getState().equals(Controller.END))
            this.useEXE().execute(c);
      }
      catch (RejectedExecutionException e)
      {
         log.write(this.toString(), "CAR REJECTED "+c.toString());
      }
   }
   catch(InterruptedException ex)
   {
      log.write(this.toString(),"CAR CREATION SLEEP INTERRUPTED");
      switch (controller.getState())
      {
         case Controller.CONTINUE:
            log.write(this.toString(), "MISS MATCH");
            repeatSleep = true;
            break;
         case Controller.PAUSE:
            log.write(this.toString(), "CAR PRODUCTION PAUSED");
            stop.look();
            repeatSleep = true;
            log.write(this.toString(), "CAR PRODUCTION RESUMED");
            break;
         case Controller.END:
            log.write(this.toString(), "FORCE TERMINATION");
            this.car.clear();
            break;
         default:
            log.write(this.toString(), "CONTROLLER BADLY USED");
            break;
      }
   }
}
```

```
            while (repeatSleep);
        }

        log.write(this.toString(), "ALL CARS CREATED");

        //Wait for Cars
        this.useEXE().shutdown();

        do
        {
            repeatSleep = false;
            try
            {
                this.useEXE().awaitTermination(20, TimeUnit.MINUTES);
            }
            catch (InterruptedException ex)
            {
                 log.write(this.toString(), "INTERRUPTION WHILE WAITING FOR CARS TO
END");
                if (controller.getState().equals(Controller.CONTINUE))
                {
                    log.write(this.toString(), "MISS MATCH");
                }
                else
                {
                    log.write(this.toString(), "NOT POSSIBLE");
                }
                repeatSleep = true;
            }
        }
        while (repeatSleep);

        log.write(this.toString(), "ALL CARS ENDED, WORKERS GET INTERRUPT");
        controller.setState(Controller.END);
        //Wait for Workers
        for(Worker w : worker)
        {
            w.interruptT();
        }

        do
        {
            repeatSleep = false;
            try
```

```
            {
                count.await();
            }
            catch (InterruptedException ex)
            {
                log.write(this.toString(), "INTERRUPTION WHILE WAITING FOR WORKERS
TO END");
                if (controller.getState().equals(Controller.CONTINUE))
                {
                    log.write(this.toString(), "MISS MATCH");
                }
                else
                {
                    log.write(this.toString(), "NOT POSSIBLE");
                }
                repeatSleep = true;

            }
        }
        while(repeatSleep);
    }
    //END SERVER
    server.end();
    log.write(this.toString(), "FUEL STATION SIMULATION ENDED CORRECTLY");
    visual.endSimulation();
    end.countDown();
    this.running = false;
}

/**
 * Pausa la simulación
 */
public synchronized void pauseSimulation()
{
    log.write(this.toString(), "SIMULATION PAUSED");

    controller.setState(Controller.PAUSE);
    stop.close();

    this.interruptT();
    for (Worker w : worker)
    {
        w.interruptT();
    }
```

```
    stop.close();
    if (debugState[0])
        visual.barrera(false);
}

/**
 * Reanuda la simulación
 */
public synchronized void resumeSimulation()
{
    controller.setState(Controller.CONTINUE);
    if (start.getCount()==1)
    {
        start.countDown();
    }
    else
    {
        log.write(this.toString(), "SIMULATION RESUMED");
        stop.open();
    }
    if (debugState[0])
        visual.barrera(true);
}

/**
 * Termina la simulación
 */
public void end()
{
    if (this.running)
    {
        log.write(this.toString(), "BRUTE FORCE CLOSING");
        controller.setState(Controller.END);
        stop.open();
        this.interruptT();
        executor.shutdownNow();
        if (finalState)
        {
            recollector.end();
        }
        try
        {
            end.await();
```

```
        }
        catch (InterruptedException ex)
        {
            log.write(this.toString(), "BAD SYNCRONIZATION END");
        }
    }
    log.save();
}

/**
 * Da el número de trabajadores
 * @return int WORKERS
 */
public int getWorkers()
{
    return WORKERS;
}
/**
 * Da el número de coches
 * @return int CARS
 */
public int getCars()
{
    return CARS;
}

/**
 * Permite modificar el tiempo de espera entre los coches para entrar en la gasolinera.
 * No modifica el tiempo de forma absoluta, siempre se esperará de forma aleatoria
 * sobre el valor pasado con un mínimo de 500ms añadidos sobre el aleatorio
 * @param time
 */
public synchronized void setTime (int time)
{
    if (time > 0)
        this.timeToWait = time;
}

/**
 * Retorna el executor
 * @return
 */
private synchronized ExecutorService useEXE ()
{
```

```
    return this.executor;
}

/**
 * Permite interrumpir el hilo
 * @return Boolean Indica si la interrupción se ha realizado
 */
public synchronized boolean interruptT()
{
    if (running)
    {
        currentThread.interrupt();
    }
    return running;
}

/**
 * Se utiliza para el log
 * @return FuelStation
 */
@Override
public String toString()
{
    return "FuelStation";
}

/**
 * Se resume la posición de todos los coches en la gasolinera en una cadena de datos
 * Procesa los buffers de estado de la gasolinera para obtener esta información
 * Es utilizada para mostrar los datos si se utiliza el modo se ejecución pasivo
 * @return Cadena de caracteres que describe el estado de la gasolinera
 */
public String getStatus()
{
    String data = "";
    ArrayList <String> bufferC = new ArrayList <>();
    HashMap <Integer,String> pumpsC = new HashMap <>();
    ArrayList <String> bufferW = new ArrayList <>();
    HashMap <Integer,String> pumpsW = new HashMap <>();

    carBuffer.forEach((c) ->
    {
        bufferC.add(c.getID());
    });
```

```
workerBuffer.forEach((w) ->
{
   bufferW.add(w.getID());
});
pumpBufferCar.entrySet().forEach((p) ->
{
   pumpsC.put(p.getKey(), p.getValue());
});
pumpBufferWorker.entrySet().forEach((p) ->
{
   pumpsW.put(p.getKey(), p.getValue());
});

pumpsC.values().forEach((p) ->
{
   bufferC.remove(p);
});
pumpsW.values().forEach((p) ->
{
   bufferW.remove(p);
});

data += (!controller.getState().equals(Controller.PAUSE))? "1 ":"0 ";
data += bufferC.size() + " ";
data = bufferC.stream().map((c) -> c + " ").reduce(data, String::concat);
data += bufferW.size() + " ";
data = bufferW.stream().map((w) -> w + " ").reduce(data, String::concat);

for (int x = 1; x <= PUMPS; x++)
{
   if (pumpsC.containsKey(x))
   {
      data += pumpsC.get(x) + " ";
   }
   else
   {
      data += "N ";
   }
   if (pumpsW.containsKey(x))
   {
      data += pumpsW.get(x) + " ";
   }
   else
   {
```

```
                data += "N ";
            }
        }

        return data;
    }

    /**
     * Permite configurar el estado parado o contiar a partir de un boolean
     * @param state true=resume, false=pause
     */
    public void barrera (boolean state)
    {
        if (state)
        {
            this.resumeSimulation();
        }
        else
        {
            this.pauseSimulation();
        }
    }
}
```

Log

```
package pecl3.src;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Date;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.LinkedBlockingQueue;
import pecl3.screens.FuelStationInterface;

/**
 *
 * @author mr.blissfulgrin
 */
public class Log
{
   private final ConcurrentHashMap <String,String> log;
   private final LinkedBlockingQueue <String> evoluvionGasolinera;
   private final long initialTime;
   private final FuelStationInterface visual;
   private final boolean debugState;

    /**
     * Crea un log para cada clase que lo necesita y uno general de lo que escribieron todas
     *
     * @param visual Donde aplicar el texto
     * @param debugState Indica si modo pasivo
     */
   public Log (FuelStationInterface visual, boolean debugState)
   {
      this.evoluvionGasolinera = new LinkedBlockingQueue <>();
      this.log = new ConcurrentHashMap<>();
      this.initialTime = new Date().getTime();
      evoluvionGasolinera.offer("------------------------------------------------------------\n");
      this.visual = visual;
      this.debugState = debugState;
   }

    /**
```

```
 * Crea un log para cada clase que lo necesita y uno general de lo que escribieron todas
 * @param key Clase que escribe
 * @param txt Datos a escribir
 */
public void write(String key, String txt)
{
   if(!log.containsKey(key))
   {
      log.put(key, "------------------------------------------------------------\n");
   }
   String str = " - "+relativeTime()+" -" + txt+"\n";
   evoluvionGasolinera.offer(key + " " + str);
   log.put(key,  log.get(key) + str);
   if (debugState)
      visual.write(retriveLog());
   //System.out.println(key + " - " + str);
}

/**
 * Borra el registro de datos anterior y guarda el registro de la nueva simulación
 */
public void save ()
{
   try
   {
      //Elimina el registro anterior
      String listado[] = new File("./src/txt").list();

      for (String archivo : listado)
      {
         if(archivo.charAt(0)!='.')
         {
            new File("./src/txt/"+archivo).delete();
         }
      }

      //Guarda el registro nuevo
      for (String key : log.keySet())
      {
         log.put(key, log.get(key) + "------------------------------------------------------------\n");
         try(FileWriter stream = new FileWriter("./src/txt/"+key+".txt"))
         {
            try(BufferedWriter buffer = new BufferedWriter(stream))
            {
```

```
                try (PrintWriter writer = new PrintWriter(buffer))
                {
                    writer.print(log.get(key));
                    writer.close();
                }
                buffer.close();
            }
            stream.close();
        }
    }
    evoluvionGasolinera.offer("-----------------------------------------------------------\n");
    try(FileWriter stream = new FileWriter("./src/txt/evoluvionGasolinera.txt"))
        {
            try(BufferedWriter buffer = new BufferedWriter(stream))
            {
                try (PrintWriter writer = new PrintWriter(buffer))
                {
                    writer.print(retriveLog());
                    writer.close();
                }
                buffer.close();
            }
            stream.close();
        }
   }
   catch (IOException e)
   {
      System.out.println("ERROR DE GUARDADO" + e.toString());
   }
}

/**
 * Tiempo transcurrido desde que se inició el log
 * @return tiempo en ms
 */
public long relativeTime()
{
   return new Date().getTime() - initialTime;
}

/**
 * Retorna el log general del conjunto de lo que todas las clases escribieron
 * @return String log
 */
```

```
    public String retriveLog ()
    {
        String str = "";
        str = evoluvionGasolinera.stream().map((s) -> s).reduce(str, String::concat);
        return str;
    }
}
```

Pump

```
package pecl3.src;

import java.util.concurrent.Semaphore;


/**
 *
 * @author mr.blissfulgrin
 */
public class Pump extends Thread implements Comparable<Pump>
{
   private final int number;
   private Car car;
   private Worker worker;
   private final Log log;
   private final StopPoint stop;
   private final Controller controller;
   private boolean running;
   private Thread currentThread;
   private final Semaphore semaphore;

   public Pump (int number, Log log, StopPoint stop, Controller controller)
   {
      this.number = number;
      this.log = log;
      this.stop = stop;
      this.controller = controller;
      this.running = false;
      this.semaphore =new Semaphore (0,true);
   }

   /**
    * Un coche enetra al Surtidor
    * @param car
    */
   public synchronized void addCar (Car car)
   {
      this.car = car;
   }

   /**
```

```
 * Un trabajador entra al surtidor
 * @param worker
 */
public synchronized void addWorker(Worker worker)
{
   this.worker = worker;
}

/**
 * Obtener el coche que está en el surtidor (para la interface gráfiuca)
 * @return
 */
public synchronized String getCar()
{
   if (car != null)
      return car.toString();
   else
      return "null";
}

/**
 * Obtener el coche que está en el surtidor (para la comunicación)
 * @return
 */
public synchronized String getCarID()
{
   if (car != null)
      return car.getID();
   else
      return "N";
}

/**
 * Obtener el trabajador que está en el surtidor (para la comunicación)
 * @return
 */
public synchronized String getWorkerID()
{
   if (worker != null)
      return worker.getID();
   else
      return "N";
}
```

```
    /**
     * Obtener el número de surtidor (para la comunicación)
     * @return
     */
    public synchronized int getNumber ()
    {
        return this.number;
    }

    /**
     * Permite al los coches esperar en el surtidor
     * @throws InterruptedException Admite interrupción de la espera con control esxterno
     */
    public void waitCar () throws InterruptedException
    {
        semaphore.acquire();
    }

    /**
     * Vacia el surtidor
     */
    public synchronized void clear ()
    {
        worker = null;
        car = null;
    }

    /**
     * Permite al los trabajadores esperar en el surtidor
     * @throws InterruptedException Admite interrupción de la espera con control esxterno
     */
    public void waitWorker () throws InterruptedException
    {
        semaphore.acquire();
    }

    /**
     * La vida de este hilo consiste exclusivamente de un sleep y notificaqciones hacia el log
     * Se implementa esto así para evitar la relación directa entre el coche y el trabajador en
todo momento
     */
    @Override
    public void run ()
    {
```

```
        currentThread = Thread.currentThread();
        running = true;
        int time;
        boolean repeatSleep;
        do
        {
           repeatSleep = false;
           try
           {
              time = (int) (Math.random()*4000+4000);
              log.write(this.toString(), car.toString() + " Será servido por " + worker.toString() +
" en " + time + "ms");

              Thread.sleep(time);

               log.write(this.toString(), car.toString() + " Fue servido por " + worker.toString() +
" en " + time + "ms");
              clear();
           }
           catch(InterruptedException e)
           {
              log.write(this.toString(), "SLEEP PUMP INTERRUPTED");
              switch (controller.getState())
              {
                 case Controller.CONTINUE:
                    log.write(this.toString(), "MISS MATCH");
                    repeatSleep = true;
                    break;
                 case Controller.PAUSE:
                    log.write(this.toString(), "PUMP PAUSED");
                    car.interruptT();
                    stop.look();
                    repeatSleep = true;
                    log.write(this.toString(), "PUMP RESUMED");
                    break;
                 case Controller.END:
                    log.write(this.toString(), "PUMP FORCED TERMINATION");
                    car.interruptT();
                    break;
                 default:
                    log.write(this.toString(), "CONTROLLER BADLY USED");
                    break;
              }
           }
```

```
        }
        while (repeatSleep);
        semaphore.release(2);
    }

    /**
     * Permite interrumpir los surtidores pues serán ejecutados dentro de un pool de hilos
     */
    public synchronized void interruptT()
    {
        if (running)
        {
            currentThread.interrupt();
        }
    }

    /**
     * Ordena los surtidores por su número
     * @param other
     * @return
     */
    @Override
    public int compareTo(Pump other)
    {
        if (getNumber() < other.getNumber())
            return -1;
        else if (getNumber() == other.getNumber())
            return 0;
        else
            return 1;
    }

    @Override
    public String toString()
    {
        return "Pump "+ number;
    }
}
```

Recollector

```
package pecl3.src;

import java.util.concurrent.atomic.AtomicBoolean;
import pecl3.screens.FuelStationInterface;

/**
 *
 * @author mr.blissfulgrin
 */
public class Recollector extends Thread
{
   private final FuelStation fuelStation;
   private final FuelStationInterface fuelStationInterface;
   private final AtomicBoolean control;
   private final Log log;
   private final int time;
   private final boolean write;

      public Recollector (FuelStation fuelStation, FuelStationInterface fuelStationInterface,
Log log)
   {
      this.fuelStation = fuelStation;
      this.fuelStationInterface = fuelStationInterface;
      this.control = new AtomicBoolean(true);
      this.log = log;
      this.time = 500;
      this.write = false;
   }
   /**
    * Finaliza la ejecución del hilo
    */
   public void end ()
   {
      control.set(false);
      this.interrupt();
   }

   /**
    * Este hilo implementa el modo pasivo de refresco de la interface gráfica
    * Realiza un productor consumidor entre la gasolinera y la interface gráfica
    * utilizando las colas de recogida de datos como buffers de modo que la
```

```
     * gasolinera no tenga que actualizarla de forma activa
     */
    @Override
    public void run ()
    {
        log.write(this.toString(), "Started ");
        while (control.get())
        {
            if (write)
                log.write(this.toString(), "Screen updated ");
            fuelStationInterface.write(log.retriveLog());
            fuelStationInterface.reset();
            fuelStationInterface.setData(fuelStation.getStatus().split(" "));
            try
            {
                Thread.sleep (time);
            }
            catch (InterruptedException e)
            {
                log.write(this.toString(), "Interrupted ");
            }
        }
        log.write(this.toString(), "Ended ");
    }

    @Override
    public String toString ()
    {
        return "Recollector";
    }
}
```

StopPoint

```
package pecl3.src;

import java.util.concurrent.locks.Condition;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

/**
 *
 * @author mr.blissfulgrin
 */
public class StopPoint
{
   private boolean close;
   private final Lock lock;
   private final Condition stop;
   private final Log log;

   public StopPoint (Log log)
   {
      this.close = false;
      this.log = log;
      this.lock = new ReentrantLock();
      this.stop = lock.newCondition();
   }

   /**
    * If this object was closed before when we use this method well be kept waiting until the open method is called.
    * If it was open we do nothing and continue the normal execution path
    */
   public void look()
   {
      try
      {
         lock.lock();
         while(close)
         {
            try
            {
               stop.await();
            }
```

```
            catch(InterruptedException i)
            {
               log.write(this.toString(),"INTERRUPTED");
            }
         }
      }
      finally
      {
         lock.unlock();
      }
   }

   /**
    * This makes the thread available to continue the execution whenever the look method
is called
    */
   public void open()
   {
      try
      {
         lock.lock();
         close=false;
         stop.signalAll();
      }
      finally
      {
         lock.unlock();
      }
   }

   /**
    * This makes the thread stop until we reopen whenever the look method is called
    */
   public void close()
   {
      try
      {
         lock.lock();
         close=true;
         stop.signalAll();
      }
      finally
      {
         lock.unlock();
```

```
        }
    }

    @Override
    public String toString()
    {
        return "StopPoint";
    }
}
```

Worker

```
package pecl3.src;

import java.util.NoSuchElementException;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.CountDownLatch;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.LinkedBlockingQueue;
import java.util.concurrent.PriorityBlockingQueue;
import java.util.concurrent.RejectedExecutionException;
import java.util.concurrent.Semaphore;
import java.util.concurrent.TimeUnit;
import pecl3.screens.FuelStationInterface;

/**
 *
 * @author mr.blissfulgrin
 */
public class Worker extends Thread
{
   private final int number;
   private final Log log;
   private final PriorityBlockingQueue <Pump> freePump;  //PRIORITY LIST
   private final LinkedBlockingQueue <Pump> caredPump;   //QUEUE
   private final FuelStationInterface visual;
   private final StopPoint stop;
   private final Controller controller;
   private boolean running;
   private Thread currentThread;
   private final ExecutorService executor;
   private final CountDownLatch count;
   private final Semaphore semaphore;
   private final LinkedBlockingQueue <Worker> workerBuffer;
   private final ConcurrentHashMap <Integer,String> pumpBuffer;
   private final boolean debugState;

      public Worker(int number, Log log, PriorityBlockingQueue <Pump> freePump,
LinkedBlockingQueue <Pump> caredPump,
                                    FuelStationInterface visual, StopPoint stop, Controller
controller,CountDownLatch barrier,
```

```
                    Semaphore semaphore,LinkedBlockingQueue <Worker> workerBuffer,
ConcurrentHashMap <Integer,String> pumpBuffer, boolean debugState)
  {
    this.number = number;
    this.log = log;
    this.freePump = freePump;
    this.caredPump = caredPump;
    this.visual = visual;
    this.stop = stop;
    this.controller = controller;
    this.running = false;
    this.executor = Executors.newSingleThreadExecutor();
    this.count = barrier;
    this.semaphore =semaphore;
    this.workerBuffer = workerBuffer;
    this.pumpBuffer = pumpBuffer;
    this.debugState = debugState;
  }

  /**
   * La ejecución de este hilo sigue las siguentes fases:
     *  · Toma un surtidor de la cola de surtidores con coche ordenada por el orden de
entrada a la cola
   *  · Si no hubiera un surtidor espera en orden a que uno sea liberado
   *  · Entra en el surtidor y lo ejecuta desde el pool de hilos que tiene el trabajador
      *  · Cuando es despertado introduce el surtidor que había tomado en la cola de
surtidores vacíos
   *  · Vuelve a realizar todos los pasos hasta que no haya más coches que servir
   */
  @Override
  public void run()
  {
    currentThread = Thread.currentThread();
    running = true;
    Pump pump;
    boolean repeatSleep;
    boolean retakePump;

    while (!controller.getState().equals(Controller.END))
    {
      do
      {
        retakePump = false;
        //Whait for a caredPump
```

```
            while (caredPump.isEmpty() && !controller.getState().equals(Controller.END))
            {
               synchronized (caredPump)
               {
                  try
                  {
                     log.write(this.toString(), "Esperando por un coche");
                     if (!workerBuffer.contains(this))
                        workerBuffer.add(this);
                     caredPump.wait();
                     workerBuffer.remove(this);
                     log.write(this.toString(), "Hay coches para servir");
                  }
                  catch (InterruptedException ex)
                  {
                       log.write(this.toString(), "INTERRUPTION WHILE WAITING FOR FREE
PUM");
                     switch (controller.getState())
                     {
                        case Controller.CONTINUE:
                           log.write(this.toString(), "MISS MATCH");
                           break;
                        case Controller.PAUSE:
                           log.write(this.toString(), "WORKER PAUSED");
                           stop.look();
                           break;
                        case Controller.END:
                           log.write(this.toString(), "WORKER FORCED TERMINATION");
                           freePump.clear();
                           caredPump.clear();
                           executor.shutdownNow();
                           break;
                        default:
                           log.write(this.toString(), "CONTROLLER BADLY USED");
                           break;
                     }
                  }
               }
            }

            try
            {
               //Adquire a Pump
               pump = caredPump.remove();
```

```
            pumpBuffer.put(pump.getNumber(), this.getID());
            pump.addWorker(this);

            if (debugState)
               visual.workerStartFueling(number, pump.getNumber());
                log.write(this.toString(), "Comienza a servir en "+ pump.toString() + " a " +
pump.getCar());

            try
            {
               executor.execute(pump);
            }
            catch (RejectedExecutionException e)
            {
               log.write(this.toString(), "REJECTED "+pump.toString());
            }
            //Wait for the Pump to refuel

            do
            {
               repeatSleep = false;
               try
               {
                  pump.waitWorker();
                  pumpBuffer.remove(pump.getNumber());
                  log.write(this.toString(), "Ha terminado de servir");
                  if (debugState)
                     visual.workerEndsFueling(number, pump.getNumber());
                  //Free Pump
                  try
                  {
                     pump = new Pump(pump.getNumber(),log,stop,controller);
                     freePump.offer(pump);
                     synchronized (semaphore)
                     {
                        semaphore.release();
                     }
                  }
                  catch(ClassCastException c)
                  {
                     log.write(this.toString(), "PUMP NOT SORTABLE");
                  }
               }
               catch (InterruptedException ex)
```

```
                {
                    log.write(this.toString(), "WORKER INTERRUPTED");
                    switch (controller.getState())
                    {
                        case Controller.CONTINUE:
                            log.write(this.toString(), "MISS MATCH");
                            repeatSleep = true;
                            break;
                        case Controller.PAUSE:
                            log.write(this.toString(), "WORKER PAUSED");
                            pump.interruptT();
                            stop.look();
                            repeatSleep = true;
                            log.write(this.toString(), "WORKER RESUMED");
                            break;
                        case Controller.END:
                            log.write(this.toString(), "WORKER FORCED TERMINATION");
                            executor.shutdownNow();
                            break;
                        default:
                            log.write(this.toString(), "CONTROLLER BADLY USED");
                            break;
                    }
                }
            }
            while(repeatSleep);

        }
                        catch (NoSuchElementException | RejectedExecutionException |
NullPointerException e)
        {
            log.write(this.toString(), "CARED_PUMP WAS EMPTY");
            retakePump = !controller.getState().equals(Controller.END);
        }
    }
    while (retakePump);
}
try
{
    executor.awaitTermination(20, TimeUnit.MINUTES);
}
catch (InterruptedException ex)
{
```

```
            log.write(this.toString(), "INTERRUPTED WHILE WAITING FOR EXECUTOR TO
END");
        }
        log.write(this.toString(), "Ha terminado");

        count.countDown();
        running = false;
    }

    /**
     * Permite interrumpir al hilo
     * Este hilo podría ejecutarse en un pool y seguir siendo interrumpido
     */
    public synchronized void interruptT()
    {
        if (running)
        {
            currentThread.interrupt();
        }
    }

    /**
     * Da el ID del trabajador, es utilizado por la comunicación y la actualización pasiva
     * @return ID
     */
    public String getID ()
    {
        return String.valueOf(number);
    }
    /**
     * Da el ID del trabajador, es utilizado por la interface gráfica
     * @return WORKER ID
     */
    @Override
    public String toString()
    {
        return "Worker " + number;
    }
}
```

Clases de comunicación

ClientConection

```
package pecl3.src.conectivity;

import java.io.IOException;
import java.net.InetAddress;
import java.net.Socket;
import pecl3.screens.VClientMenu;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.util.concurrent.atomic.AtomicBoolean;
import pecl3.screens.FuelStationInterface;

/**
 *
 * @author mr.blissfulgrin
 */
public class ClientConection extends Thread
{
    private InetAddress add;
    private Socket socket;
    private FuelStationInterface clientInterface;
    private DataInputStream input;
    private DataOutputStream output;
    private final AtomicBoolean control;
    private int errors;

    public ClientConection ()
    {
        this.errors = 0;
        this.control = new AtomicBoolean (true);
    }

    /**
     * Nos indica si la dirección IP proporcionada es accesible desde nuestro equipo
     * @param IP Dirección a probar
     * @return Es o no accesible
     */
    public boolean conectable (byte [] IP)
    {
        try
```

```
    {
        add = InetAddress.getByAddress(IP);
        if (!add.isReachable(5000))
            return false;
    }
    catch (IOException e)
    {
        return false;
    }
    VClientMenu vClientMenu= new VClientMenu (this);
    vClientMenu.setVisible(true);
    return true;
}

/**
 * Inicia los Streams de comunicación a la vez que nos vincula con una interface gráfica
 * en la que mostrar los datos recibidos
 * @param clientInterface
 */
public void setDisplay (FuelStationInterface clientInterface)
{
    this.clientInterface = clientInterface;
    try
    {
        socket= new Socket(add,4444);
        input = new DataInputStream(socket.getInputStream());
        output = new DataOutputStream(socket.getOutputStream());
        clientInterface.write("CONEXION -- OK");
    }
    catch (IOException e)
    {
        clientInterface.write("ERRORR AL CREAR CONEXION");
    }
    this.start();
}

/**
 * El cliente escucha los datos del servidor y los muestra por pantalla
 * de recibir una "X" finaliza su ejecución
 */
@Override
public void run ()
{
    String [] data;
```

```
    while (control.get())
    {
       if (errors > 2)
          this.end();
       try
       {
          data = input.readUTF().split(" ");
          clientInterface.reset();
          errors = 0;
          if (data[0].equals("X"))
          {
             this.end();
          }
          else
          {
             clientInterface.setData(data);
             clientInterface.write("DATA RECIVED");
          }
       }
       catch (IOException e)
       {
          clientInterface.write("ERRORR READING DATA");
          errors++;
       }
    }
}

/**
 * Finaliza de forma segura la simulación y lo notifica al servidor
 */
public void end ()
{
    control.set(false);
    try
    {
       output.writeUTF("X");
       clientInterface.write("X SENT");
    }
    catch (IOException e)
    {
       clientInterface.write("SENDING DATA X");
    }
    try
    {
```

```
            input.close();
            output.close();
            socket.close();
            clientInterface.endSimulation();
            clientInterface.write("CONECTION CLOSED");
        }
        catch (IOException e)
        {
            clientInterface.write("ERROR CLOSING");
        }
    }

    /**
     * Indica al Servidor que queremos pausar la simulación
     */
    public void pauseSimulation ()
    {
        try
        {
            output.writeUTF("P");
            clientInterface.write("P SENT");
        }
        catch (IOException e)
        {
            clientInterface.write("ERROR SENDING DATA P");
        }
    }

    /**
     * Indica al Servidor que queremos reanudar la simulación
     */
    public void resumeSimulation ()
    {
        try
        {
            output.writeUTF("R");
             clientInterface.write("R SENT");
        }
        catch (IOException e)
        {
             clientInterface.write("ERROR SENDING DATA R");
        }
    }
}
```

ReciverUDPClient

```
package pecl3.src.conectivity;

import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.SocketException;
import java.net.SocketTimeoutException;
import java.net.UnknownHostException;
import pecl3.screens.VLoad;

/**
 *
 * @author mr.blissfulgrin
 */
public class ReciverUDPClient extends Thread
{
   private final DatagramSocket socket;
   private byte[] IP;
   private boolean timeout;

   public ReciverUDPClient (DatagramSocket socket)
   {
      timeout = false;
      this.IP = null;
      this.socket = socket;
      try
      {
         this.socket.setSoTimeout(1500);
      }
      catch (SocketException ex){}
      try
      {
         timeout = this.socket.getSoTimeout() != 0;
      }
      catch (SocketException ex){}
   }

   /**
    * Escuchamos durante un máximo de tiempo a recibir un paquete del Servidor
     * Si obtenemos de este un paquete tendremos su IP, pero de no ser así no podremos saberla
```

```
     * Es muy probable no poder obtener la IP del Servidor ya que existen múltipleas
factores que nos impidan hacerlo
     */
    @Override
    public void run ()
    {
        if (timeout)
        {
            try
            {
                byte[] buf = new byte[1];
                try
                {
                    DatagramPacket paquete = new DatagramPacket(buf, buf.length);

                    socket.receive(paquete);
                    IP = paquete.getAddress().getAddress();
                    socket.close();

                } catch (UnknownHostException | SocketTimeoutException ex)
                {
                    VLoad vLoad = new VLoad();
                    vLoad.go("Host failed, Retrying...");
                }
            }
            catch (IOException e)
            {
                VLoad vLoad = new VLoad();
                vLoad.go("IO failed, Retrying...");
            }
        }
        else
        {
            VLoad vLoad = new VLoad();
            vLoad.go("Timeout failed, Retrying...");
        }
    }

    /**
     * Retorna la IP del cliente si logró obtenerla
     * @return De no obtenerse la IP valdrá null
     */
    public byte[] getIP ()
    {
```

```
        return IP;
    }
}
```

ReciverUDPServer

```
package pecl3.src.conectivity;

import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.UnknownHostException;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.atomic.AtomicBoolean;
import java.util.logging.Level;
import java.util.logging.Logger;
import pecl3.src.Log;

/**
 *
 * @author mr.blissfulgrin
 */
public class ReciverUDPServer extends Thread
{
    private final AtomicBoolean control = new AtomicBoolean(true);
    private DatagramSocket socket;
    private final Log log;
    private final ExecutorService executor;

    public ReciverUDPServer (Log log)
    {
        this.log = log;
        this.executor = Executors.newSingleThreadExecutor();
    }

    /**
     * Escuchamos paquetes UDP y en el caso de recibir uno intentamos conectarnos con el
emisor
     */
    @Override
    public void run ()
    {
        log.write(this.toString(), "UDP Server ON");
        while (control.get())
        {
```

```
        try
        {
            socket = new DatagramSocket(2222);
            byte[] buf = new byte[1];
            try
            {
                DatagramPacket paquete = new DatagramPacket(buf, buf.length);
                do
                {
                    socket.receive(paquete);
                    log.write(this.toString(), "Reception "+paquete.getAddress().toString());
                                        SenderUDPServer send = new SenderUDPServer
(paquete.getAddress(),paquete.getPort(), log);
                    executor.execute(send);
                }
                while (control.get());
            } catch (UnknownHostException ex)
            {
                log.write(this.toString(), "NO HOST");
            }
        }
        catch (IOException e)
        {
            log.write(this.toString(), "NO IO");
        }
    }
  }

  /**
   * Finalizar el Servidor UDP
   */
  public void end ()
  {
    this.interrupt();
    executor.shutdown();
    try
    {
      executor.awaitTermination(5, TimeUnit.SECONDS);
    }
    catch (InterruptedException ex)
    {
      log.write(this.toString(), "EXECUTOR AWAIT INTERRUPTED");
    }
    control.set(false);
```

```
        socket.close();
        log.write(this.toString(), "UDP CLOSED");
    }

    @Override
    public String toString()
    {
        return "Server UDP RECIVER";
    }
}
```

SenderUDPClient

```
package pecl3.src.conectivity;

import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.UnknownHostException;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;
import pecl3.screens.VLoad;

/**
 *
 * @author mr.blissfulgrin
 */
public class SenderUDPClient extends Thread
{
   private boolean control;
   private byte[] serverIP;
   private final ExecutorService executor;

   public SenderUDPClient ()
   {
      serverIP = null;
      control = true;
      this.executor = Executors.newSingleThreadExecutor();
   }

   /**
    * Intenta hasta tres veces conectarse con el Servidor UDP por broadcast
    * En caso de lograrlo de esta clase se podrá obtener la IP del servidor
    */
   @Override
   public void run ()
   {
      InetAddress destino;
      byte[] buf = new byte[] {'N'};
      byte [] IP = new byte [] {(byte)(255),(byte)(255),(byte)(255),(byte)(255)};
      int tries = 0;
```

```
    while (tries < 3 && control)
    {
      DatagramSocket socket;
      ReciverUDPClient reciver;
      try
      {
        socket = new DatagramSocket((int)(Math.random()*3000+2000));
        reciver = new ReciverUDPClient(socket);
        executor.execute(reciver);
        try
        {
          destino = InetAddress.getByAddress(IP);
              DatagramPacket paquete = new DatagramPacket(buf, buf.length, destino,
2222);
          socket.send(paquete);
          try
          {
            executor.awaitTermination(5, TimeUnit.SECONDS);
            serverIP = reciver.getIP();
            if (serverIP != null)
              control = false;
            socket.close();
          }
          catch (InterruptedException ex)
          {
            VLoad vLoad = new VLoad();
            vLoad.go("Server fail, Retrying...");
          }
        }
        catch (UnknownHostException ex)
        {
          VLoad vLoad = new VLoad();
          vLoad.go("Server fail, Retrying...");
        }
      }
      catch (IOException e)
      {
        VLoad vLoad = new VLoad();
        vLoad.go("Server fail, Retrying...");
      }
      tries ++;
    }
  }
```

```
    /**
     * Permite obtener la dirección IP del servidor
     * @return Valdrá null si la red no admite broadcast
     */
    public byte[] getAutoIP()
    {
        return serverIP;
    }
}
```

SenderUDPServer

```
package pecl3.src.conectivity;

import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.SocketException;
import java.net.UnknownHostException;
import pecl3.src.Log;

/**
 *
 * @author mr.blissfulgrin
 */
public class SenderUDPServer extends Thread
{
   private final InetAddress add;
   private final int port;
   private DatagramSocket socket;
   private final Log log;

   public SenderUDPServer (InetAddress add, int port,Log log)
   {
      this.add = add;
      this.port = port;
      int tries = 0;
      this.log = log;
      while (tries < 5 && socket == null)
      {
         try
         {
            this.socket = socket = new DatagramSocket((int)(Math.random()*3000+2000));
         }
         catch (SocketException ex)
         {
            tries++;
         }
      }
   }

   /**
```

```
     * Si el socket UDP fue creado correctamente intenta enviar sobre él un paquete hacia la dirección de quien lo enviara
     */
    @Override
    public void run ()
    {
        if (socket!= null)
        {
            log.write(this.toString(), "DATA SENDABLE");
            byte[] buf = new byte[] {'N'};
            {
                try
                {
                    try
                    {
                        DatagramPacket paquete = new DatagramPacket(buf, buf.length, add, port);
                        socket.send(paquete);
                        socket.close();
                        log.write(this.toString(), "DATA SENT");
                    }
                    catch (UnknownHostException ex)
                    {
                        log.write(this.toString(),"NO HOST");
                    }
                }
                catch (IOException e)
                {
                    log.write(this.toString(),"NO IO");
                }
            }
        }
        else
            log.write(this.toString(), "DATA NOT SENDABLE");
    }


    @Override
    public String toString ()
    {
        return "Server UDP SENDER";
    }
}
```

Server

```
package pecl3.src.conectivity;

import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.CountDownLatch;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.RejectedExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.atomic.AtomicBoolean;
import pecl3.src.FuelStation;
import pecl3.src.Log;

/**
 *
 * @author mr.blissfulgrin
 */
public class Server extends Thread
{
   private ServerSocket server;
   private Socket socket;
   private final AtomicBoolean control;
   private final ExecutorService executor;
   private final ConcurrentHashMap <Integer,ServerReciver> services;
   private final Log log;
   private final ServerSender sender;
   private final CountDownLatch end;
   private final ReciverUDPServer udp;

   public Server (Log log, FuelStation fuelStation, CountDownLatch end)
   {
      control = new AtomicBoolean(true);
      executor = Executors.newFixedThreadPool(11);
      services = new ConcurrentHashMap <> ();
      this.log = log;
      this.end = end;
      try
      {
         this.server = new ServerSocket(4444);
```

```
            log.write(this.toString(), "SERVER READY");
        }
        catch (IOException e)
        {
            log.write(this.toString(), "ERROR AL CREAR SERVIDOR");
        }
        sender = new ServerSender (fuelStation, log,this);
        try
        {
            executor.execute(sender);
        }
        catch (RejectedExecutionException e)
        {
            log.write(this.toString(), "ERROR AL CREAR SENDER");
        }
        udp = new ReciverUDPServer(log);
    }

    /**
     * Lanza un servidor UDP que atenderá en paralelo esas peticiones y se queda escuchando
     * a las peticiones TCP de conexión que le lleguen, cuando lo haga lanzará otros hilos que las controlen.
     */
    @Override
    public void run()
    {
        udp.start();
        if (server!=null)
        {
            int id;
            while (control.get())
            {
                do
                {
                    id = (int)(Math.random()*(Integer.MAX_VALUE-1)+1);
                }
                while (services.contains(id));

                try
                {
                    log.write(this.toString(), "WAITING FOR CLIENT "+id);
                    socket = server.accept();
                    log.write(this.toString(), "NUEVO CLIENTE "+id);
```

```
                ServerReciver r = new ServerReciver (socket,id,sender, log);
                services.put(id,r);
                sender.add(socket, id, this);
                executor.execute(r);
            }
            catch (IOException | RejectedExecutionException e)
            {
                log.write(this.toString(), "NO HUBO CLIENTE EN "+id);
            }
        }
    }
}

/**
 * Finaliza de forma segura al servidor UDP y al TCP
 */
public synchronized void end ()
{
    try
    {
        udp.end();
        log.write(this.toString(), "CLOSING SERVER");
        control.set(false);
        sender.end();
        services.keySet().forEach((reciver)->
        {
            services.get(reciver).end();
        });
        executor.shutdown();
        executor.awaitTermination(5, TimeUnit.SECONDS);
        executor.shutdownNow();
        executor.awaitTermination(5, TimeUnit.SECONDS);
        if (server != null)
            server.close();
    }
    catch (IOException | InterruptedException ex)
    {
        log.write(this.toString(),"ERROR AL TERMINAR ");
    }
    end.countDown();
    log.write(this.toString(), "SERVER CLOSED CORRECTLY");
    try
    {
        udp.join();
```

```
        }
        catch (InterruptedException ex)
        {
            log.write(this.toString(), "UDP JOIN INTERRUPTED");
        }
    }

    /**
     * Elimina una conexión TCP de la lista de servicios
     * @param toRemove
     */
    public synchronized void toRemove (int toRemove)
    {
        services.remove(toRemove);
    }

    @Override
    public String toString()
    {
        return "Server";
    }
}
```

ServerReciver

```
package pecl3.src.conectivity;

import java.io.DataInputStream;
import java.io.IOException;
import java.net.Socket;
import java.util.concurrent.atomic.AtomicBoolean;
import pecl3.src.Log;

/**
 *
 * @author mr.blissfulgrin
 */
public class ServerReciver extends Thread
{
    private final ServerSender sender;
    private final int id;
    private DataInputStream stream;
    private final Log log;
    private final AtomicBoolean control;
    private final Socket socket;
    private int errors;

    public ServerReciver (Socket socket, int id,ServerSender sender, Log log)
    {
        this.sender = sender;
        this.id = id;
        this.log = log;
        this.control = new AtomicBoolean (true);
        this.socket = socket;
        this.errors = 0;
        try
        {
            stream = new DataInputStream(socket.getInputStream());
        }
        catch (IOException e)
        {
            log.write(this.toString(), "Error creating InputStream "+id);
        }
    }

    /**
```

```
     * Escucha a los Clientes y decodifica la información que cada uno recibe
      * ya que cada conexión tiene una identidad podremos saber con qué cliente nos comunicamos
     */
    @Override
    public void run ()
    {
        String data = "";
        while (control.get())
        {
            if (errors > 2)
                this.end(true);
            try
            {
                data = stream.readUTF().trim();
                log.write(this.toString(), data + " RECIVED FROM " + id);
                errors = 0;
            }
            catch (IOException ex)
            {
                log.write(this.toString(), "CONECTION CLOSED");
                errors++;
            }
            switch (data)
            {
                case "X":
                    this.end(true);
                    break;
                case "P":
                    sender.barrera(false);
                    break;
                case "R":
                    sender.barrera(true);
                    break;
                default:
                    break;
            }
        }
    }

    /**
     * Cierra de forma segura al receptor TCP
     * @param notify Indica si avisar al emisor o no del borrado
     */
```

```
    public void end (boolean notify)
    {
        control.set(false);
        try
        {
            stream.close();
            log.write(this.toString(), "RECIVER CLOSED "+ id);
        }
        catch (IOException e)
        {
            log.write(this.toString(), "ERROR CLOSING "+ id);
        }
        if (notify)
        {
            sender.remove(id);
        }
        try
        {
            socket.close();
            log.write(this.toString(), "SOCKET CLOSED CORRECTLY "+id);
        }
        catch (IOException e)
        {
            log.write(this.toString(), "ERROR CLOSING "+ id);
        }
    }

    @Override
    public String toString()
    {
        return "ServerReciver "+id;
    }
}
```

ServerSender

```
package pecl3.src.conectivity;

import java.io.DataOutputStream;
import java.io.IOException;
import java.net.Socket;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.atomic.AtomicBoolean;
import pecl3.src.FuelStation;
import pecl3.src.Log;

/**
 *
 * @author mr.blissfulgrin
 */
public class ServerSender extends Thread
{
   private final ConcurrentHashMap <Integer, DataOutputStream> stream;
   private final FuelStation fuelStation;
   private final Log log;
   private final AtomicBoolean control;
   private Thread thread;
   private final Server server;
   private final int time;
   private final boolean write;

   public ServerSender(FuelStation fuelStation, Log log, Server server)
   {
      stream = new ConcurrentHashMap <>();
      this.fuelStation = fuelStation;
      this.log = log;
      this.control = new AtomicBoolean (true);
      this.server = server;
      this.time = 1000;
      this.write = false;
   }

   /**
    * Añade un nuevo cliente a la lista de aquellos a los que hay que dar sevicio de datos
    * @param socket socket a partir del cual crear el canal de envío
    * @param id identidad única para el socket
    * @param server nos permite retroalimental la comunicación para cerrar las conexiones.
```

```
 */
public void add (Socket socket, int id, Server server)
{
   try
   {
      this.stream.put(id, new DataOutputStream(socket.getOutputStream()));
   }
   catch (IOException e)
   {
      log.write(this.toString(), "Error creating OutputStream "+id);
   }
}

/**
 * Cada segundo toma los datos procesados de los buffers de datos de la gasolinera
 * y los envia a todos los crientes adscritos al servicio
 */
@Override
public void run ()
{
   thread = Thread.currentThread();
   String data;
   while(control.get())
   {
      if (!stream.isEmpty())
      {
         data = fuelStation.getStatus();
         for (int key : stream.keySet())
         {
            try
            {
               stream.get(key).writeUTF(data+key);
               if (write)
                  log.write(this.toString(), "DATA SENT TO "+key);
            }
            catch (IOException e)
            {
               log.write(this.toString(),"ERROR SENDING DATA TO " + key);
            }
         }
         try
         {
            Thread.sleep(time);
         }
```

```
            catch (InterruptedException e)
            {
               log.write(this.toString(), "Sender Interrupted");
            }
         }
      }
      log.write(this.toString(), "SENDER END");
   }

   /**
    * Cierra una conexión única se utiliza cuando un cliete se desconecta
    * @param toRemove
    */
   public void remove (int toRemove)
   {
      try
      {
         if (stream.get(toRemove)!=null)
         {
            stream.get(toRemove).close();
            log.write(this.toString(), "NOTYFIED SENDER CLOSED" + toRemove);
         }
         else
         {
            throw new IOException();
         }
      }
      catch (IOException e)
      {
               log.write(this.toString(),"NOTYFIED REMOVING SENDER STREAM IN " +
toRemove);
      }
      server.toRemove(toRemove);
      stream.remove(toRemove);
   }

   /**
    * Finaliza de forma segura al em isor y borra las referencias del receptor en el servidor
    */
   public void end ()
   {
      control.set(false);
      thread.interrupt();
      stream.keySet().forEach((key) ->
```

```
        {
            try
            {
                stream.get(key).writeUTF("X");
                log.write(this.toString(), "CLIENT CLOSED" + key);
            }
            catch (IOException e)
            {
                log.write(this.toString(),"CLIENT ALREADY CLOSED IN " + key);
            }
            try
            {
                stream.get(key).close();
                log.write(this.toString(), "SENDER CLOSED" + key);
            }
            catch (IOException e)
            {
                log.write(this.toString(),"ERROR CLOSING SENDER STREAM IN " + key);
            }
        });
        log.write(this.toString(), "SENDER CLOSED CORRECTLY");
    }

    /**
     * Indica el estado que uno de los clientes solicita sobre la gasolinera, pausa o resumen
     * @param state true = resume; false = pausa
     */
    public synchronized void barrera(boolean state)
    {
        fuelStation.barrera(state);
    }

    @Override
    public String toString()
    {
        return "ServerSender";
    }
}
```

Clases Gráficas

FuelStationInterface

```
package pecl3.screens;

/**
 *
 * @author mr.blissfulgrin
 */
public interface FuelStationInterface
{
    //Métodos que toda interface de un servidor debe cumplir

    //USADOS EN EL MODO ACTIVO
    /**
     * Muestra un coche en el buffer
     * @param number
     */
    public void carArrivesFuelStation(int number);
    /**
     * Elimina al coche del buffer de espera
     * @param number
     */
    public void carLeavesFuelStationQueue(int number);
    /**
     * Pone al coche en un surtidor
     * @param carNumber
     * @param pumpNumber
     */
    public void carReachesPump (int carNumber, int pumpNumber);
    /**
     * Pone al trabajador en un surtidor
     * @param workerNumber
     * @param pumpNumber
     */
    public void workerStartFueling (int workerNumber, int pumpNumber);
    /**
     * Elimina a un trabajador de un surtidor
     * @param workerNumber
     * @param pumpNumber
     */
    public void workerEndsFueling (int workerNumber, int pumpNumber);
```

```
    /**
     * Elimina un coche de un surtidor
     * @param carNumber
     * @param pumpNumber
     */
    public void carLeavesPump (int carNumber, int pumpNumber);
    /**
     * Muestra gráficamente que la simulación ha finalizado
     */
    public void endSimulation ();

    /**
     * Pone el texto en la pantalla
     * En el Servidor lo pone tal cual y en el cliente lo añade al que había
     * @param texto
     */
    public void write(String texto);

    /**
     * Usado por la comunicación, los clientes dice en qué estado querrían la barrera
     * y por ende si quieren pausar o reanudafr la simulación
     * @param state
     */
    public void barrera(boolean state);

    //USADOS EN EL MODO PASIVO
    /**
     * Borra todos los datos de la interface
     */
    public void reset();
    /**
     * Extyrae los dartos de la interface a partidr de un array que los contiene codificados
     * La sintaxis de codificación es idéntica para todas las interfaces
     * @param data
     */
    public void setData(String [] data);
}
```

LoadController

```
package pecl3.screens;

/**
 *
 * @author mr.blissfulgrin
 */
public class LoadController extends Thread
{

    private final VLoad vLoad;
    private final javax.swing.JProgressBar bar;

    public LoadController (VLoad vLoad, javax.swing.JProgressBar bar)
    {
        this.vLoad = vLoad;
        this.bar = bar;
    }
    @Override
    public void run()
    {
        for(int x = 0; x<100; x++)
        {
            try
            {
                Thread.sleep(5);
            }
            catch (InterruptedException i)
            {
                System.out.println("ERROR LOADING!!! " + i.toString());
            }
            bar.setValue(x);
            bar.setString(x + "%");
            bar.update(bar.getGraphics());
        }
        vLoad.setVisible(false);
        vLoad.dispose();
    }

}
```

SFuelStation

```
package pecl3.screens;

import java.io.IOException;
import java.net.InetAddress;
import java.util.ArrayList;
import javax.swing.JTextField;
import pecl3.src.FuelStation;

/**
 *
 * @author mr.blissfulgrin
 */
public class SFuelStation extends javax.swing.JFrame implements FuelStationInterface
{
    private final FuelStation fuelStation;
    private final JTextField [] worker;
    private final JTextField [] car;
    private final ArrayList <String> buffer;
    private final ArrayList <String> wbuffer;

    public SFuelStation(int workers, int cars)
    {
        initComponents();
        fuelStation = new FuelStation(this, workers, cars);
        barier.setSelected(true);

        try
        {
            ip.setText("Connect to: " + InetAddress.getLocalHost().getHostAddress());
        }
        catch (IOException e){}

        worker = new JTextField[8];
        worker[0] = worker0;
        worker[1] = worker1;
        worker[2] = worker2;
        worker[3] = worker3;
        worker[4] = worker4;
        worker[5] = worker5;
        worker[6] = worker6;
        worker[7] = worker7;
```

```
    wbuffer = new ArrayList <>();
    workerTxt.setText("");
    for (int i = 0; i < fuelStation.WORKERS; i++)
    {
        wbuffer.add("Worker "+(i+1));
    }
    wbuffer.forEach((s) ->
    {
        workerTxt.setText(workerTxt.getText() + s+"\n");
    });

    car = new JTextField [8];
    car [0] = car0;
    car [1] = car1;
    car [2] = car2;
    car [3] = car3;
    car [4] = car4;
    car [5] = car5;
    car [6] = car6;
    car [7] = car7;
    buffer = new ArrayList <>();

    for (JTextField w : worker)
    {
        w.setText("");
    }
    for (JTextField c : car)
    {
        c.setText("");
    }
    buff.setText("");
}


@Override
public synchronized void carArrivesFuelStation(int number)
{
    buffer.add(String.valueOf(number));
    buff.setText("");
    for (int i = buffer.size()-1; i >= 0; i--)
    {
        buff.setText(buff.getText() + buffer.get(i) + " | ");
    }
```

```
}
@Override
public synchronized void carLeavesFuelStationQueue(int number)
{
   buffer.remove(String.valueOf(number));
   buff.setText("");
   for (int i = buffer.size()-1; i >= 0; i--)
   {
      buff.setText(buff.getText() + buffer.get(i)+" | ");
   }
}
@Override
public synchronized void carReachesPump(int carNumber, int pumpNumber)
{
   car[pumpNumber-1].setText("Car "+carNumber);
}
@Override
public synchronized void workerStartFueling(int workerNumber, int pumpNumber)
{
   worker[pumpNumber-1].setText("Worker "+workerNumber);
   wbuffer.remove("Worker "+workerNumber);
   workerTxt.setText("");
   wbuffer.forEach((s) ->
   {
      workerTxt.setText(workerTxt.getText() + s+"\n");
   });
}
@Override
public synchronized void workerEndsFueling(int workerNumber, int pumpNumber)
{
   worker[pumpNumber-1].setText("");
   wbuffer.add("Worker "+workerNumber);
   workerTxt.setText("");
   wbuffer.forEach((s) ->
   {
      workerTxt.setText(workerTxt.getText() + s+"\n");
   });
}
@Override
public synchronized void carLeavesPump(int carNumber, int pumpNumber)
{
   car[pumpNumber-1].setText("");
}
@Override
```

```
public synchronized void endSimulation()
{
   buff.setText("SIMULATION ENDED, SAVING DATA");
}
@Override
public synchronized void write(String texto)
{
   txt.setText(texto);
}

@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">
private void initComponents()
{
   java.awt.GridBagConstraints gridBagConstraints;

   jPanel1 = new javax.swing.JPanel();
   jPanel2 = new javax.swing.JPanel();
   jLabel1 = new javax.swing.JLabel();
   car0 = new javax.swing.JTextField();
   worker0 = new javax.swing.JTextField();
   timeSlider = new javax.swing.JSlider();
   timeTxt = new javax.swing.JLabel();
   jPanel3 = new javax.swing.JPanel();
   jLabel3 = new javax.swing.JLabel();
   car1 = new javax.swing.JTextField();
   worker1 = new javax.swing.JTextField();
   jPanel4 = new javax.swing.JPanel();
   jLabel4 = new javax.swing.JLabel();
   car2 = new javax.swing.JTextField();
   worker2 = new javax.swing.JTextField();
   jPanel5 = new javax.swing.JPanel();
   jLabel5 = new javax.swing.JLabel();
   car3 = new javax.swing.JTextField();
   worker3 = new javax.swing.JTextField();
   jPanel6 = new javax.swing.JPanel();
   jLabel6 = new javax.swing.JLabel();
   car4 = new javax.swing.JTextField();
   worker4 = new javax.swing.JTextField();
   jPanel7 = new javax.swing.JPanel();
   jLabel7 = new javax.swing.JLabel();
   car5 = new javax.swing.JTextField();
   worker5 = new javax.swing.JTextField();
   jPanel8 = new javax.swing.JPanel();
```

```
jLabel8 = new javax.swing.JLabel();
car6 = new javax.swing.JTextField();
worker6 = new javax.swing.JTextField();
jPanel9 = new javax.swing.JPanel();
jLabel9 = new javax.swing.JLabel();
car7 = new javax.swing.JTextField();
worker7 = new javax.swing.JTextField();
buff = new javax.swing.JTextField();
barier = new javax.swing.JToggleButton();
jScrollPane1 = new javax.swing.JScrollPane();
workerTxt = new javax.swing.JTextArea();
jLabel2 = new javax.swing.JLabel();
ip = new javax.swing.JLabel();
jScrollPane3 = new javax.swing.JScrollPane();
txt = new javax.swing.JTextArea();

setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
setTitle("FuelStation");
setResizable(false);
addWindowListener(new java.awt.event.WindowAdapter()
{
   public void windowOpened(java.awt.event.WindowEvent evt)
   {
      formWindowOpened(evt);
   }
   public void windowClosing(java.awt.event.WindowEvent evt)
   {
      formWindowClosing(evt);
   }
});
getContentPane().setLayout(new java.awt.CardLayout());

jPanel1.setLayout(new java.awt.GridBagLayout());

jPanel2.setLayout(new java.awt.GridLayout(0, 1));

jLabel1.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
jLabel1.setText("Pump 1");
jLabel1.setIconTextGap(20);
jPanel2.add(jLabel1);

car0.setEditable(false);
car0.setText("car0");
car0.setMaximumSize(new java.awt.Dimension(41, 24));
```

```
car0.setMinimumSize(new java.awt.Dimension(41, 24));
jPanel2.add(car0);

worker0.setEditable(false);
worker0.setText("worker0");
worker0.setMaximumSize(new java.awt.Dimension(41, 24));
worker0.setMinimumSize(new java.awt.Dimension(41, 24));
worker0.setPreferredSize(new java.awt.Dimension(41, 24));
jPanel2.add(worker0);

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 1;
gridBagConstraints.gridy = 3;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
gridBagConstraints.ipadx = 68;
gridBagConstraints.ipady = 57;
gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
gridBagConstraints.insets = new java.awt.Insets(49, 53, 49, 81);
jPanel1.add(jPanel2, gridBagConstraints);

timeSlider.setMaximum(10000);
timeSlider.setValue(5500);
timeSlider.addChangeListener(new javax.swing.event.ChangeListener()
{
   public void stateChanged(javax.swing.event.ChangeEvent evt)
   {
      timeSliderStateChanged(evt);
   }
});
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 1;
gridBagConstraints.gridy = 2;
gridBagConstraints.gridwidth = 4;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
gridBagConstraints.ipadx = 154;
gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
gridBagConstraints.insets = new java.awt.Insets(10, 43, 10, 43);
jPanel1.add(timeSlider, gridBagConstraints);

timeTxt.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
timeTxt.setText("5500");
timeTxt.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
timeTxt.setIconTextGap(10);
timeTxt.setMaximumSize(new java.awt.Dimension(41, 24));
```

```
timeTxt.setMinimumSize(new java.awt.Dimension(41, 24));
timeTxt.setPreferredSize(new java.awt.Dimension(41, 24));
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 5;
gridBagConstraints.gridy = 2;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
gridBagConstraints.insets = new java.awt.Insets(0, 0, 0, 13);
jPanel1.add(timeTxt, gridBagConstraints);

jPanel3.setLayout(new java.awt.GridLayout(0, 1));

jLabel3.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
jLabel3.setText("Pump 2");
jPanel3.add(jLabel3);

car1.setEditable(false);
car1.setText("car1");
car1.setMaximumSize(new java.awt.Dimension(41, 24));
car1.setMinimumSize(new java.awt.Dimension(41, 24));
jPanel3.add(car1);

worker1.setEditable(false);
worker1.setText("worker1");
worker1.setMaximumSize(new java.awt.Dimension(41, 24));
worker1.setMinimumSize(new java.awt.Dimension(41, 24));
worker1.setPreferredSize(new java.awt.Dimension(41, 24));
jPanel3.add(worker1);

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 2;
gridBagConstraints.gridy = 3;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
gridBagConstraints.ipadx = 68;
gridBagConstraints.ipady = 57;
gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
gridBagConstraints.insets = new java.awt.Insets(49, 81, 49, 81);
jPanel1.add(jPanel3, gridBagConstraints);

jPanel4.setLayout(new java.awt.GridLayout(0, 1));

jLabel4.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
jLabel4.setText("Pump 3");
jPanel4.add(jLabel4);
```

```
car2.setEditable(false);
car2.setText("car2");
car2.setMaximumSize(new java.awt.Dimension(41, 24));
car2.setMinimumSize(new java.awt.Dimension(41, 24));
jPanel4.add(car2);

worker2.setEditable(false);
worker2.setText("worker2");
worker2.setMaximumSize(new java.awt.Dimension(41, 24));
worker2.setMinimumSize(new java.awt.Dimension(41, 24));
worker2.setPreferredSize(new java.awt.Dimension(41, 24));
jPanel4.add(worker2);

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 3;
gridBagConstraints.gridy = 3;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
gridBagConstraints.ipadx = 68;
gridBagConstraints.ipady = 57;
gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
gridBagConstraints.insets = new java.awt.Insets(49, 81, 49, 81);
jPanel1.add(jPanel4, gridBagConstraints);

jPanel5.setLayout(new java.awt.GridLayout(0, 1));

jLabel5.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
jLabel5.setText("Pump 4");
jPanel5.add(jLabel5);

car3.setEditable(false);
car3.setText("car3");
car3.setMaximumSize(new java.awt.Dimension(41, 24));
car3.setMinimumSize(new java.awt.Dimension(41, 24));
car3.setRequestFocusEnabled(false);
jPanel5.add(car3);

worker3.setEditable(false);
worker3.setText("worker3");
worker3.setMaximumSize(new java.awt.Dimension(41, 24));
worker3.setMinimumSize(new java.awt.Dimension(41, 24));
worker3.setPreferredSize(new java.awt.Dimension(41, 24));
jPanel5.add(worker3);
```

```
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 4;
gridBagConstraints.gridy = 3;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
gridBagConstraints.ipadx = 68;
gridBagConstraints.ipady = 57;
gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
gridBagConstraints.insets = new java.awt.Insets(49, 81, 49, 52);
jPanel1.add(jPanel5, gridBagConstraints);

jPanel6.setLayout(new java.awt.GridLayout(0, 1));

jLabel6.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
jLabel6.setText("Pump 5");
jPanel6.add(jLabel6);

car4.setEditable(false);
car4.setText("car4");
car4.setMaximumSize(new java.awt.Dimension(41, 24));
car4.setMinimumSize(new java.awt.Dimension(41, 24));
jPanel6.add(car4);

worker4.setEditable(false);
worker4.setText("worker4");
worker4.setMaximumSize(new java.awt.Dimension(41, 24));
worker4.setMinimumSize(new java.awt.Dimension(41, 24));
worker4.setPreferredSize(new java.awt.Dimension(41, 24));
jPanel6.add(worker4);

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 1;
gridBagConstraints.gridy = 4;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
gridBagConstraints.ipadx = 68;
gridBagConstraints.ipady = 57;
gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
gridBagConstraints.insets = new java.awt.Insets(49, 53, 49, 81);
jPanel1.add(jPanel6, gridBagConstraints);

jPanel7.setLayout(new java.awt.GridLayout(0, 1));

jLabel7.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
jLabel7.setText("Pump 6");
jPanel7.add(jLabel7);
```

```
car5.setEditable(false);
car5.setText("car5");
car5.setMaximumSize(new java.awt.Dimension(41, 24));
car5.setMinimumSize(new java.awt.Dimension(41, 24));
jPanel7.add(car5);

worker5.setEditable(false);
worker5.setText("worker5");
worker5.setMaximumSize(new java.awt.Dimension(41, 24));
worker5.setMinimumSize(new java.awt.Dimension(41, 24));
worker5.setPreferredSize(new java.awt.Dimension(41, 24));
jPanel7.add(worker5);

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 2;
gridBagConstraints.gridy = 4;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
gridBagConstraints.ipadx = 68;
gridBagConstraints.ipady = 57;
gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
gridBagConstraints.insets = new java.awt.Insets(49, 81, 49, 81);
jPanel1.add(jPanel7, gridBagConstraints);

jPanel8.setLayout(new java.awt.GridLayout(0, 1));

jLabel8.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
jLabel8.setText("Pump 7");
jPanel8.add(jLabel8);

car6.setEditable(false);
car6.setText("car6");
car6.setMaximumSize(new java.awt.Dimension(41, 24));
car6.setMinimumSize(new java.awt.Dimension(41, 24));
jPanel8.add(car6);

worker6.setEditable(false);
worker6.setText("worker6");
jPanel8.add(worker6);

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 3;
gridBagConstraints.gridy = 4;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
```

```
gridBagConstraints.ipadx = 68;
gridBagConstraints.ipady = 57;
gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
gridBagConstraints.insets = new java.awt.Insets(49, 81, 49, 81);
jPanel1.add(jPanel8, gridBagConstraints);

jPanel9.setLayout(new java.awt.GridLayout(0, 1));

jLabel9.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
jLabel9.setText("Pump 8");
jPanel9.add(jLabel9);

car7.setEditable(false);
car7.setText("car7");
car7.setMaximumSize(new java.awt.Dimension(41, 24));
car7.setMinimumSize(new java.awt.Dimension(41, 24));
jPanel9.add(car7);

worker7.setEditable(false);
worker7.setText("worker7");
worker7.setMaximumSize(new java.awt.Dimension(41, 24));
worker7.setMinimumSize(new java.awt.Dimension(41, 24));
worker7.setPreferredSize(new java.awt.Dimension(41, 24));
jPanel9.add(worker7);

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 4;
gridBagConstraints.gridy = 4;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
gridBagConstraints.ipadx = 56;
gridBagConstraints.ipady = 45;
gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
gridBagConstraints.insets = new java.awt.Insets(49, 81, 49, 52);
jPanel1.add(jPanel9, gridBagConstraints);

buff.setEditable(false);
buff.setHorizontalAlignment(javax.swing.JTextField.RIGHT);
buff.setText("buff");
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 1;
gridBagConstraints.gridy = 1;
gridBagConstraints.gridwidth = 4;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
gridBagConstraints.insets = new java.awt.Insets(42, 19, 12, 18);
```

```
        jPanel1.add(buff, gridBagConstraints);

        barier.setText("OFF");
        barier.setMaximumSize(new java.awt.Dimension(41, 24));
        barier.setMinimumSize(new java.awt.Dimension(41, 24));
        barier.setPreferredSize(new java.awt.Dimension(41, 24));
        barier.addActionListener(new java.awt.event.ActionListener()
        {
           public void actionPerformed(java.awt.event.ActionEvent evt)
           {
              barierActionPerformed(evt);
           }
        });
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 5;
        gridBagConstraints.gridy = 1;
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.insets = new java.awt.Insets(7, 7, 7, 12);
        jPanel1.add(barier, gridBagConstraints);

        workerTxt.setEditable(false);
        workerTxt.setColumns(20);
        workerTxt.setRows(5);
        jScrollPane1.setViewportView(workerTxt);

        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 5;
        gridBagConstraints.gridy = 3;
        gridBagConstraints.gridheight = 2;
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.weightx = 0.3;
        gridBagConstraints.insets = new java.awt.Insets(109, 26, 109, 26);
        jPanel1.add(jScrollPane1, gridBagConstraints);

        jLabel2.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
                    jLabel2.setIcon(new   javax.swing.ImageIcon(getClass().getResource("/img/
SING.png"))); // NOI18N
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 1;
        gridBagConstraints.gridy = 0;
        gridBagConstraints.gridwidth = 3;
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.weightx = 4.0;
        gridBagConstraints.insets = new java.awt.Insets(41, 46, 41, 46);
```

```
    jPanel1.add(jLabel2, gridBagConstraints);

    ip.setFont(new java.awt.Font("Tahoma", 1, 30)); // NOI18N
    ip.setText("Conect to: 255.255.255.255");
    gridBagConstraints = new java.awt.GridBagConstraints();
    gridBagConstraints.gridx = 4;
    gridBagConstraints.gridy = 0;
    gridBagConstraints.gridwidth = 2;
    jPanel1.add(ip, gridBagConstraints);

    txt.setColumns(20);
    txt.setFont(new java.awt.Font("Lucida Grande", 0, 9)); // NOI18N
    txt.setRows(5);
    jScrollPane3.setViewportView(txt);

    gridBagConstraints = new java.awt.GridBagConstraints();
    gridBagConstraints.gridx = 0;
    gridBagConstraints.gridy = 1;
    gridBagConstraints.gridheight = 4;
    gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
    gridBagConstraints.weightx = 4.2;
    gridBagConstraints.insets = new java.awt.Insets(15, 25, 19, 7);
    jPanel1.add(jScrollPane3, gridBagConstraints);

    getContentPane().add(jPanel1, "card2");

    pack();
    setLocationRelativeTo(null);
}// </editor-fold>

private void formWindowClosing(java.awt.event.WindowEvent evt)
{
    fuelStation.end();
    VInicio i = new VInicio();
    i.setVisible(true);
}

private void timeSliderStateChanged(javax.swing.event.ChangeEvent evt)
{
    timeTxt.setText(String.valueOf(timeSlider.getValue()));
    fuelStation.setTime(timeSlider.getValue());
}

private void barierActionPerformed(java.awt.event.ActionEvent evt)
```

```
{
    if (barier.isSelected())
    {
        barier.setText("OFF");
        fuelStation.pauseSimulation();
    }
    else
    {
        barier.setText("ON");
        fuelStation.resumeSimulation();
    }
}

private void formWindowOpened(java.awt.event.WindowEvent evt)
{
   fuelStation.start();
}
@Override
public synchronized void barrera(boolean state)
{
    if (state)
    {
        barier.setText("ON");
    }
    else
    {
        barier.setText("OFF");
    }
}
// Variables declaration - do not modify
private javax.swing.JToggleButton barier;
private javax.swing.JTextField buff;
private javax.swing.JTextField car0;
private javax.swing.JTextField car1;
private javax.swing.JTextField car2;
private javax.swing.JTextField car3;
private javax.swing.JTextField car4;
private javax.swing.JTextField car5;
private javax.swing.JTextField car6;
private javax.swing.JTextField car7;
private javax.swing.JLabel ip;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel2;
private javax.swing.JLabel jLabel3;
```

```
private javax.swing.JLabel jLabel4;
private javax.swing.JLabel jLabel5;
private javax.swing.JLabel jLabel6;
private javax.swing.JLabel jLabel7;
private javax.swing.JLabel jLabel8;
private javax.swing.JLabel jLabel9;
private javax.swing.JPanel jPanel1;
private javax.swing.JPanel jPanel2;
private javax.swing.JPanel jPanel3;
private javax.swing.JPanel jPanel4;
private javax.swing.JPanel jPanel5;
private javax.swing.JPanel jPanel6;
private javax.swing.JPanel jPanel7;
private javax.swing.JPanel jPanel8;
private javax.swing.JPanel jPanel9;
private javax.swing.JScrollPane jScrollPane1;
private javax.swing.JScrollPane jScrollPane3;
private javax.swing.JSlider timeSlider;
private javax.swing.JLabel timeTxt;
private javax.swing.JTextArea txt;
private javax.swing.JTextField worker0;
private javax.swing.JTextField worker1;
private javax.swing.JTextField worker2;
private javax.swing.JTextField worker3;
private javax.swing.JTextField worker4;
private javax.swing.JTextField worker5;
private javax.swing.JTextField worker6;
private javax.swing.JTextField worker7;
private javax.swing.JTextArea workerTxt;
// End of variables declaration

@Override
public synchronized void reset()
{
   for (int x = 0; x < worker.length; x++)
   {
      worker[x].setText("");
      car[x].setText("");
   }
   buff.setText("");
   workerTxt.setText("");
   buffer.clear();
   wbuffer.clear();
}
```

```
    @Override
    public synchronized void setData(String[] data)
    {
        int index = 1;
        int times1;
        int times2;
        int gap1;
        int gap2;
        barrera(data[0].trim().equals("1"));
        times1 = Integer.parseInt(data[index].trim());
        index++;
        gap1 = index;
        for (; index < times1+gap1; index++)
        {
            carArrivesFuelStation(Integer.parseInt(data[index].trim()));
        }
        times2 = Integer.parseInt(data[index].trim());
        index++;
        gap2 = index;
        for (; index < times2+gap2; index++)
        {
            wbuffer.add("Worker "+Integer.parseInt(data[index].trim()));
            workerTxt.setText("");
        }
        wbuffer.forEach((s) ->
        {
            workerTxt.setText(workerTxt.getText() + s+"\n");
        });
        for (int x = 0; x < worker.length; x++)
        {
            if (!data[index].trim().equals("N"))
            {
                car[x].setText("Car: "+data[index].trim());
            }
            index++;
            if (!data[index].trim().equals("N"))
            {
                worker[x].setText("Worker: "+data[index].trim());
            }
            index++;
        }
    }
}
```

SFuelStationClient

```
package pecl3.screens;

import java.util.ArrayList;
import javax.swing.JTextField;
import pecl3.src.conectivity.ClientConection;

/**
 *
 * @author mr.blissfulgrin
 */
public class SFuelStationClient extends javax.swing.JFrame implements
FuelStationInterface
{
   private final JTextField [] worker;
   private final JTextField [] car;
   private final ArrayList <String> buffer;
   private final ArrayList <String> wbuffer;
   private final ClientConection conection;

   public SFuelStationClient(ClientConection conection)
   {
      initComponents();

      this.conection = conection;
      barier.setSelected(true);

      worker = new JTextField[8];
      worker[0] = worker0;
      worker[1] = worker1;
      worker[2] = worker2;
      worker[3] = worker3;
      worker[4] = worker4;
      worker[5] = worker5;
      worker[6] = worker6;
      worker[7] = worker7;

      wbuffer = new ArrayList <>();
      workerTxt.setText("");
      /*
      for (int i = 0; i < 6; i++)
      {
```

```
        wbuffer.add("Worker "+(i+1));
    }*/
    wbuffer.forEach((s) ->
    {
        workerTxt.setText(workerTxt.getText() + s+"\n");
    });

    car = new JTextField [8];
    car [0] = car0;
    car [1] = car1;
    car [2] = car2;
    car [3] = car3;
    car [4] = car4;
    car [5] = car5;
    car [6] = car6;
    car [7] = car7;
    buffer = new ArrayList <>();

    for (JTextField w : worker)
    {
        w.setText("");
    }
    for (JTextField c : car)
    {
        c.setText("");
    }
    buff.setText("");
    txt.setText("");
}


@Override
public synchronized void carArrivesFuelStation(int number)
{
    buffer.add(String.valueOf(number));
    buff.setText("");
    for (int i = buffer.size()-1; i >= 0; i--)
    {
        buff.setText(buff.getText() + buffer.get(i) + " | ");
    }
}
@Override
public synchronized void carLeavesFuelStationQueue(int number)
{
```

```
    buffer.remove(String.valueOf(number));
    buff.setText("");
    for (int i = buffer.size()-1; i >= 0; i--)
    {
        buff.setText(buff.getText() + buffer.get(i)+" | ");
    }
}
@Override
public synchronized void carReachesPump(int carNumber, int pumpNumber)
{
    car[pumpNumber-1].setText("Car "+carNumber);
}
@Override
public synchronized void workerStartFueling(int workerNumber, int pumpNumber)
{
    worker[pumpNumber-1].setText("Worker "+workerNumber);
    wbuffer.remove("Worker "+workerNumber);
    workerTxt.setText("");
    wbuffer.forEach((s) ->
    {
        workerTxt.setText(workerTxt.getText() + s+"\n");
    });
}
@Override
public synchronized void workerEndsFueling(int workerNumber, int pumpNumber)
{
    worker[pumpNumber-1].setText("");
    wbuffer.add("Worker "+workerNumber);
    workerTxt.setText("");
    wbuffer.forEach((s) ->
    {
        workerTxt.setText(workerTxt.getText() + s+"\n");
    });
}
@Override
public synchronized void carLeavesPump(int carNumber, int pumpNumber)
{
    car[pumpNumber-1].setText("");
}
@Override
public synchronized void endSimulation()
{
    buff.setText("SIMULATION ENDED, CONECTION CLOSED");
}
```

```
@Override
public synchronized void write(String texto)
{
   txt.setText(txt.getText()+texto+"\n");
}

@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">
private void initComponents()
{
   java.awt.GridBagConstraints gridBagConstraints;

   jPanel1 = new javax.swing.JPanel();
   jPanel2 = new javax.swing.JPanel();
   jLabel1 = new javax.swing.JLabel();
   car0 = new javax.swing.JTextField();
   worker0 = new javax.swing.JTextField();
   jPanel3 = new javax.swing.JPanel();
   jLabel3 = new javax.swing.JLabel();
   car1 = new javax.swing.JTextField();
   worker1 = new javax.swing.JTextField();
   jPanel4 = new javax.swing.JPanel();
   jLabel4 = new javax.swing.JLabel();
   car2 = new javax.swing.JTextField();
   worker2 = new javax.swing.JTextField();
   jPanel5 = new javax.swing.JPanel();
   jLabel5 = new javax.swing.JLabel();
   car3 = new javax.swing.JTextField();
   worker3 = new javax.swing.JTextField();
   jPanel6 = new javax.swing.JPanel();
   jLabel6 = new javax.swing.JLabel();
   car4 = new javax.swing.JTextField();
   worker4 = new javax.swing.JTextField();
   jPanel7 = new javax.swing.JPanel();
   jLabel7 = new javax.swing.JLabel();
   car5 = new javax.swing.JTextField();
   worker5 = new javax.swing.JTextField();
   jPanel8 = new javax.swing.JPanel();
   jLabel8 = new javax.swing.JLabel();
   car6 = new javax.swing.JTextField();
   worker6 = new javax.swing.JTextField();
   jPanel9 = new javax.swing.JPanel();
   jLabel9 = new javax.swing.JLabel();
   car7 = new javax.swing.JTextField();
```

```
worker7 = new javax.swing.JTextField();
buff = new javax.swing.JTextField();
barier = new javax.swing.JToggleButton();
jScrollPane1 = new javax.swing.JScrollPane();
workerTxt = new javax.swing.JTextArea();
jLabel2 = new javax.swing.JLabel();
id = new javax.swing.JLabel();
jScrollPane3 = new javax.swing.JScrollPane();
txt = new javax.swing.JTextArea();

setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
setTitle("FuelStation");
setResizable(false);
addWindowListener(new java.awt.event.WindowAdapter()
{
   public void windowClosing(java.awt.event.WindowEvent evt)
   {
      formWindowClosing(evt);
   }
});
getContentPane().setLayout(new java.awt.CardLayout());

jPanel1.setLayout(new java.awt.GridBagLayout());

jPanel2.setLayout(new java.awt.GridLayout(0, 1));

jLabel1.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
jLabel1.setText("Pump 1");
jLabel1.setIconTextGap(20);
jPanel2.add(jLabel1);

car0.setEditable(false);
car0.setText("car0");
car0.setMaximumSize(new java.awt.Dimension(41, 24));
car0.setMinimumSize(new java.awt.Dimension(41, 24));
jPanel2.add(car0);

worker0.setEditable(false);
worker0.setText("worker0");
worker0.setMaximumSize(new java.awt.Dimension(41, 24));
worker0.setMinimumSize(new java.awt.Dimension(41, 24));
worker0.setPreferredSize(new java.awt.Dimension(41, 24));
jPanel2.add(worker0);
```

```
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 1;
gridBagConstraints.gridy = 2;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
gridBagConstraints.ipadx = 68;
gridBagConstraints.ipady = 57;
gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
gridBagConstraints.insets = new java.awt.Insets(49, 50, 49, 81);
jPanel1.add(jPanel2, gridBagConstraints);

jPanel3.setLayout(new java.awt.GridLayout(0, 1));

jLabel3.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
jLabel3.setText("Pump 2");
jPanel3.add(jLabel3);

car1.setEditable(false);
car1.setText("car1");
car1.setMaximumSize(new java.awt.Dimension(41, 24));
car1.setMinimumSize(new java.awt.Dimension(41, 24));
jPanel3.add(car1);

worker1.setEditable(false);
worker1.setText("worker1");
worker1.setMaximumSize(new java.awt.Dimension(41, 24));
worker1.setMinimumSize(new java.awt.Dimension(41, 24));
worker1.setPreferredSize(new java.awt.Dimension(41, 24));
jPanel3.add(worker1);

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 2;
gridBagConstraints.gridy = 2;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
gridBagConstraints.ipadx = 68;
gridBagConstraints.ipady = 57;
gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
gridBagConstraints.insets = new java.awt.Insets(49, 81, 49, 81);
jPanel1.add(jPanel3, gridBagConstraints);

jPanel4.setLayout(new java.awt.GridLayout(0, 1));

jLabel4.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
jLabel4.setText("Pump 3");
jPanel4.add(jLabel4);
```

```
car2.setEditable(false);
car2.setText("car2");
car2.setMaximumSize(new java.awt.Dimension(41, 24));
car2.setMinimumSize(new java.awt.Dimension(41, 24));
jPanel4.add(car2);

worker2.setEditable(false);
worker2.setText("worker2");
worker2.setMaximumSize(new java.awt.Dimension(41, 24));
worker2.setMinimumSize(new java.awt.Dimension(41, 24));
worker2.setPreferredSize(new java.awt.Dimension(41, 24));
jPanel4.add(worker2);

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 3;
gridBagConstraints.gridy = 2;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
gridBagConstraints.ipadx = 68;
gridBagConstraints.ipady = 57;
gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
gridBagConstraints.insets = new java.awt.Insets(49, 81, 49, 81);
jPanel1.add(jPanel4, gridBagConstraints);

jPanel5.setLayout(new java.awt.GridLayout(0, 1));

jLabel5.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
jLabel5.setText("Pump 4");
jPanel5.add(jLabel5);

car3.setEditable(false);
car3.setText("car3");
car3.setMaximumSize(new java.awt.Dimension(41, 24));
car3.setMinimumSize(new java.awt.Dimension(41, 24));
car3.setRequestFocusEnabled(false);
jPanel5.add(car3);

worker3.setEditable(false);
worker3.setText("worker3");
worker3.setMaximumSize(new java.awt.Dimension(41, 24));
worker3.setMinimumSize(new java.awt.Dimension(41, 24));
worker3.setPreferredSize(new java.awt.Dimension(41, 24));
jPanel5.add(worker3);
```

```
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 4;
gridBagConstraints.gridy = 2;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
gridBagConstraints.ipadx = 68;
gridBagConstraints.ipady = 57;
gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
gridBagConstraints.insets = new java.awt.Insets(49, 81, 49, 47);
jPanel1.add(jPanel5, gridBagConstraints);

jPanel6.setLayout(new java.awt.GridLayout(0, 1));

jLabel6.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
jLabel6.setText("Pump 5");
jPanel6.add(jLabel6);

car4.setEditable(false);
car4.setText("car4");
car4.setMaximumSize(new java.awt.Dimension(41, 24));
car4.setMinimumSize(new java.awt.Dimension(41, 24));
jPanel6.add(car4);

worker4.setEditable(false);
worker4.setText("worker4");
worker4.setMaximumSize(new java.awt.Dimension(41, 24));
worker4.setMinimumSize(new java.awt.Dimension(41, 24));
worker4.setPreferredSize(new java.awt.Dimension(41, 24));
jPanel6.add(worker4);

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 1;
gridBagConstraints.gridy = 3;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
gridBagConstraints.ipadx = 68;
gridBagConstraints.ipady = 57;
gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
gridBagConstraints.insets = new java.awt.Insets(49, 50, 49, 81);
jPanel1.add(jPanel6, gridBagConstraints);

jPanel7.setLayout(new java.awt.GridLayout(0, 1));

jLabel7.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
jLabel7.setText("Pump 6");
jPanel7.add(jLabel7);
```

```
car5.setEditable(false);
car5.setText("car5");
car5.setMaximumSize(new java.awt.Dimension(41, 24));
car5.setMinimumSize(new java.awt.Dimension(41, 24));
jPanel7.add(car5);

worker5.setEditable(false);
worker5.setText("worker5");
worker5.setMaximumSize(new java.awt.Dimension(41, 24));
worker5.setMinimumSize(new java.awt.Dimension(41, 24));
worker5.setPreferredSize(new java.awt.Dimension(41, 24));
jPanel7.add(worker5);

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 2;
gridBagConstraints.gridy = 3;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
gridBagConstraints.ipadx = 68;
gridBagConstraints.ipady = 57;
gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
gridBagConstraints.insets = new java.awt.Insets(49, 81, 49, 81);
jPanel1.add(jPanel7, gridBagConstraints);

jPanel8.setLayout(new java.awt.GridLayout(0, 1));

jLabel8.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
jLabel8.setText("Pump 7");
jPanel8.add(jLabel8);

car6.setEditable(false);
car6.setText("car6");
car6.setMaximumSize(new java.awt.Dimension(41, 24));
car6.setMinimumSize(new java.awt.Dimension(41, 24));
jPanel8.add(car6);

worker6.setEditable(false);
worker6.setText("worker6");
jPanel8.add(worker6);

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 3;
gridBagConstraints.gridy = 3;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
```

```
gridBagConstraints.ipadx = 68;
gridBagConstraints.ipady = 57;
gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
gridBagConstraints.insets = new java.awt.Insets(49, 81, 49, 81);
jPanel1.add(jPanel8, gridBagConstraints);

jPanel9.setLayout(new java.awt.GridLayout(0, 1));

jLabel9.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
jLabel9.setText("Pump 8");
jPanel9.add(jLabel9);

car7.setEditable(false);
car7.setText("car7");
car7.setMaximumSize(new java.awt.Dimension(41, 24));
car7.setMinimumSize(new java.awt.Dimension(41, 24));
jPanel9.add(car7);

worker7.setEditable(false);
worker7.setText("worker7");
worker7.setMaximumSize(new java.awt.Dimension(41, 24));
worker7.setMinimumSize(new java.awt.Dimension(41, 24));
worker7.setPreferredSize(new java.awt.Dimension(41, 24));
jPanel9.add(worker7);

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 4;
gridBagConstraints.gridy = 3;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
gridBagConstraints.ipadx = 56;
gridBagConstraints.ipady = 45;
gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
gridBagConstraints.insets = new java.awt.Insets(49, 81, 49, 47);
jPanel1.add(jPanel9, gridBagConstraints);

buff.setEditable(false);
buff.setHorizontalAlignment(javax.swing.JTextField.RIGHT);
buff.setText("buff");
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 1;
gridBagConstraints.gridy = 1;
gridBagConstraints.gridwidth = 4;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
gridBagConstraints.insets = new java.awt.Insets(42, 16, 12, 13);
```

```
        jPanel1.add(buff, gridBagConstraints);

        barier.setText("OFF");
        barier.setMaximumSize(new java.awt.Dimension(41, 24));
        barier.setMinimumSize(new java.awt.Dimension(41, 24));
        barier.setPreferredSize(new java.awt.Dimension(41, 24));
        barier.addActionListener(new java.awt.event.ActionListener()
        {
           public void actionPerformed(java.awt.event.ActionEvent evt)
           {
              barierActionPerformed(evt);
           }
        });
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 5;
        gridBagConstraints.gridy = 1;
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.insets = new java.awt.Insets(7, 7, 7, 12);
        jPanel1.add(barier, gridBagConstraints);

        workerTxt.setEditable(false);
        workerTxt.setColumns(20);
        workerTxt.setRows(5);
        jScrollPane1.setViewportView(workerTxt);

        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 5;
        gridBagConstraints.gridy = 2;
        gridBagConstraints.gridheight = 2;
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.weightx = 0.3;
        gridBagConstraints.insets = new java.awt.Insets(109, 26, 109, 26);
        jPanel1.add(jScrollPane1, gridBagConstraints);

        jLabel2.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
                jLabel2.setIcon(new   javax.swing.ImageIcon(getClass().getResource("/img/
SING.png"))); // NOI18N
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 1;
        gridBagConstraints.gridy = 0;
        gridBagConstraints.gridwidth = 4;
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.weightx = 4.0;
        gridBagConstraints.insets = new java.awt.Insets(41, 46, 41, 46);
```

```
    jPanel1.add(jLabel2, gridBagConstraints);

    id.setFont(new java.awt.Font("Tahoma", 1, 30)); // NOI18N
    id.setText("ID: ");
    gridBagConstraints = new java.awt.GridBagConstraints();
    gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
    gridBagConstraints.weightx = 0.1;
    jPanel1.add(id, gridBagConstraints);

    txt.setColumns(20);
    txt.setFont(new java.awt.Font("Lucida Grande", 0, 9)); // NOI18N
    txt.setRows(5);
    txt.setMinimumSize(new java.awt.Dimension(500, 200));
    txt.setPreferredSize(new java.awt.Dimension(500, 200));
    jScrollPane3.setViewportView(txt);

    gridBagConstraints = new java.awt.GridBagConstraints();
    gridBagConstraints.gridx = 0;
    gridBagConstraints.gridy = 1;
    gridBagConstraints.gridheight = 3;
    gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
    gridBagConstraints.weightx = 4.2;
    gridBagConstraints.insets = new java.awt.Insets(14, 24, 14, 4);
    jPanel1.add(jScrollPane3, gridBagConstraints);

    getContentPane().add(jPanel1, "card2");

    pack();
    setLocationRelativeTo(null);
}// </editor-fold>

private void formWindowClosing(java.awt.event.WindowEvent evt)
{
    conection.end();
    VInicio i = new VInicio();
    i.setVisible(true);
}

private void barierActionPerformed(java.awt.event.ActionEvent evt)
{
    if (barier.isSelected())
    {
        conection.pauseSimulation();
    }
```

```
    else
    {
        conection.resumeSimulation();
    }
}



// Variables declaration - do not modify
private javax.swing.JToggleButton barier;
private javax.swing.JTextField buff;
private javax.swing.JTextField car0;
private javax.swing.JTextField car1;
private javax.swing.JTextField car2;
private javax.swing.JTextField car3;
private javax.swing.JTextField car4;
private javax.swing.JTextField car5;
private javax.swing.JTextField car6;
private javax.swing.JTextField car7;
private javax.swing.JLabel id;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel2;
private javax.swing.JLabel jLabel3;
private javax.swing.JLabel jLabel4;
private javax.swing.JLabel jLabel5;
private javax.swing.JLabel jLabel6;
private javax.swing.JLabel jLabel7;
private javax.swing.JLabel jLabel8;
private javax.swing.JLabel jLabel9;
private javax.swing.JPanel jPanel1;
private javax.swing.JPanel jPanel2;
private javax.swing.JPanel jPanel3;
private javax.swing.JPanel jPanel4;
private javax.swing.JPanel jPanel5;
private javax.swing.JPanel jPanel6;
private javax.swing.JPanel jPanel7;
private javax.swing.JPanel jPanel8;
private javax.swing.JPanel jPanel9;
private javax.swing.JScrollPane jScrollPane1;
private javax.swing.JScrollPane jScrollPane3;
private javax.swing.JTextArea txt;
private javax.swing.JTextField worker0;
private javax.swing.JTextField worker1;
private javax.swing.JTextField worker2;
```

```
private javax.swing.JTextField worker3;
private javax.swing.JTextField worker4;
private javax.swing.JTextField worker5;
private javax.swing.JTextField worker6;
private javax.swing.JTextField worker7;
private javax.swing.JTextArea workerTxt;
// End of variables declaration

@Override
public synchronized void reset()
{
   for (int x = 0; x < worker.length; x++)
   {
      worker[x].setText("");
      car[x].setText("");
   }
   buff.setText("");
   workerTxt.setText("");
   buffer.clear();
   wbuffer.clear();
}

@Override
public synchronized void barrera(boolean state)
{
   if (state)
   {
      barier.setText("ON");
      barier.setSelected(true);
   }
   else
   {
      barier.setText("OFF");
      barier.setSelected(false);
   }
}

@Override
public synchronized void setData(String[] data)
{
   int index = 1;
   int times1;
   int times2;
   int gap1;
```

```
        int gap2;
        barrera(data[0].trim().equals("1"));
        times1 = Integer.parseInt(data[index].trim());
        index++;
        gap1 = index;
        for (; index < times1+gap1; index++)
        {
           carArrivesFuelStation(Integer.parseInt(data[index].trim()));
        }
        times2 = Integer.parseInt(data[index].trim());
        index++;
        gap2 = index;
        for (; index < times2+gap2; index++)
        {
           wbuffer.add("Worker "+Integer.parseInt(data[index].trim()));
           workerTxt.setText("");
        }
        wbuffer.forEach((s) ->
        {
           workerTxt.setText(workerTxt.getText() + s+"\n");
        });
        for (int x = 0; x < worker.length; x++)
        {
           if (!data[index].trim().equals("N"))
           {
              car[x].setText("Car: "+data[index].trim());
           }
           index++;
           if (!data[index].trim().equals("N"))
           {
              worker[x].setText("Worker: "+data[index].trim());
           }
           index++;
        }
        id.setText("ID: "+data[index].trim());
     }
  }
```

Sortable

```
package pecl3.screens;

/**
 *
 * @author mr.blissfulgrin
 */
public class Sortable implements Comparable <Sortable>
{
    private final String str;

    public Sortable (String str)
    {
        this.str = str;
    }

    public String getStr ()
    {
        return str;
    }

    /**
     * Permite la ordenación los archivos por su nombre a la hora de
     * mostrarlos en la interface gráfica
     * @param o
     * @return
     */
    @Override
    public int compareTo(Sortable o)
    {
        return this.getStr().compareTo(o.getStr());
    }
}
```

Sorter

```
package pecl3.screens;

import java.util.Comparator;

/**
 *
 * @author mr.blissfulgrin
 */
public class Sorter implements Comparator <Sortable>
{
    /**
     * Realiza la ordenación entre objetos Sortable que implementan Comprable
     * Permite la ordenación los archivos por su nombre a la hora de
     * mostrarlos en la interface gráfica
     * @param s1
     * @param s2
     * @return
     */
    @Override
    public int compare (Sortable s1, Sortable s2)
    {
        String txt1 = s1.getStr();
        String txt2 = s2.getStr();
        int Ntxt1 = 0;
        int Ntxt2 = 0;
        switch (txt1.charAt(0))
        {
            case 'e':
                Ntxt1 = 0 - txt1.compareTo(txt2);
                break;
            case 'F':
                Ntxt1 = 5 - txt1.compareTo(txt2);
                break;
            case 'W':
                Ntxt1 = 10 - txt1.compareTo(txt2);
                break;
            case 'P':
                Ntxt1 = 20 - txt1.compareTo(txt2);
                break;
            case 'C':
                Ntxt1 = 30 - txt1.compareTo(txt2);
```

```
            break;
        }
        switch (txt2.charAt(0))
        {
           case 'e':
              Ntxt2 = 0 + txt1.compareTo(txt2);
              break;
           case 'F':
              Ntxt2 = 5 + txt1.compareTo(txt2);
              break;
           case 'W':
              Ntxt2 = 10 + txt1.compareTo(txt2);
              break;
           case 'P':
              Ntxt2 = 20 + txt1.compareTo(txt2);
              break;
           case 'C':
              Ntxt2 = 30 + txt1.compareTo(txt2);
              break;
        }


        if(Ntxt1 > Ntxt2)
        {  //Ordenar de menor a mayor (izquierda derecha)
          return 1;
        }
        else if(Ntxt1 < Ntxt2){
          return -1;
        }
        else
          return 0;
    }
}
```

BackGroundPanel

```
package pecl3.screens;

import java.awt.Graphics;
import javax.swing.ImageIcon;

/**
 *
 * @author mr.blissfulgrin
 */
public class VBackGroundPanel extends javax.swing.JPanel
{
    private final ImageIcon img;

    public VBackGroundPanel(int width, int height, String source, int x, int y)
    {
        initComponents();
        this.setSize(width,height);
        this.setLocation(x, y);
        this.img = new ImageIcon("./src/img/"+source+".png");
    }

    @Override
    public void paintComponent (Graphics g)
    {
        g.drawImage(img.getImage(), 0, 0,this.getSize().width,this.getSize().height, null);
        setOpaque(false);
        super.paintComponent(g);
    }

    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents()
    {

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(this);
        this.setLayout(layout);
```

```
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGap(0, 1549, Short.MAX_VALUE)
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGap(0, 950, Short.MAX_VALUE)
        );
    }// </editor-fold>


    // Variables declaration - do not modify
    // End of variables declaration
}
```

VCar

```
package pecl3.screens;

/**
 *
 * @author mr.blissfulgrin
 */
public class VCar extends VBackGroundPanel
{
    private static final int W = 115;
    private static final int H = 58;
    private int id;

    public VCar()
    {
        super(115,58,"CAR",22,70);
        initComponents();
        number.setText("");
    }

    public VCar(int x, int y)
    {
        super(115,58,"CAR",x,y);
        initComponents();
        number.setText("");
    }

    public void newCar (int n)
    {
        this.number.setText(String.valueOf(n));
        this.id = n;
    }

    public static int getW()
    {
        return W;
    }
    public static int getH()
    {
        return H;
    }
```

```
public int getNumber()
{
    try
    {
        return Integer.parseInt(number.getText());
    }
    catch(NumberFormatException n)
    {
        return 0;
    }
}

public int getID()
{
    return id;
}

/**
 * This method is called from within the constructor to initialize the form.
 * WARNING: Do NOT modify this code. The content of this method is always
 * regenerated by the Form Editor.
 */
@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">
private void initComponents()
{
    java.awt.GridBagConstraints gridBagConstraints;

    number = new javax.swing.JLabel();

    setLayout(new java.awt.GridBagLayout());

    number.setFont(new java.awt.Font("Silom", 1, 22)); // NOI18N
    number.setForeground(new java.awt.Color(255, 255, 255));
    number.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
    number.setText("NUMBER");
    gridBagConstraints = new java.awt.GridBagConstraints();
    gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
    gridBagConstraints.insets = new java.awt.Insets(9, 16, 0, 0);
    add(number, gridBagConstraints);
}// </editor-fold>


// Variables declaration - do not modify
```

```
    private javax.swing.JLabel number;
    // End of variables declaration
}
```

VCarBuffer

```
package pecl3.screens;

import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;


/**
 *
 * @author mr.blissfulgrin
 */
public class VCarBuffer  extends VBackGroundPanel
{
   private HashMap <Integer,VCar> parking;
   private Boolean fila;

   public VCarBuffer(int x, int y)
   {
      super(1140,110,"VOID",x,y);
      initComponents();
      this.parking = new HashMap <>();
      fila = true;
   }

   public void newCar(int id)
   {
      VCar car = new VCar((this.getWidth())-VCar.getW()*(parking.size()/2+1),(fila)? 0:50);
      car.newCar(id);
      parking.put(id, car);
      this.add(car);
      fila = !fila;
      this.repaint();
   }

   public void reset ()
   {
      parking.clear();
      fila = true;
      this.removeAll();
      this.repaint();
   }
```

```
public void removeCar (int id)
{
   if (parking.containsKey(id))
   {
      VCar car = parking.get(id);
      parking.remove(id);
      car.setVisible(false);
      this.remove(car);

      parking.values().forEach((c) ->
      {
         c.setVisible(false);
         this.remove(c);
         this.repaint();
      });

      ArrayList <Integer> keys = new ArrayList<>();
      parking.keySet().forEach((i) ->
      {
         keys.add(i);
      });
      Collections.sort(keys);
      parking = new HashMap<>();
      fila = true;
      keys.forEach((key) ->
      {
         newCar(key);
      });
   }
}

/**
 * This method is called from within the constructor to initialize the form.
 * WARNING: Do NOT modify this code. The content of this method is always
 * regenerated by the Form Editor.
 */
@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">
private void initComponents()
{

   javax.swing.GroupLayout layout = new javax.swing.GroupLayout(this);
   this.setLayout(layout);
```

```
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGap(0, 1140, Short.MAX_VALUE)
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGap(0, 80, Short.MAX_VALUE)
        );
    }// </editor-fold>


    // Variables declaration - do not modify
    // End of variables declaration
}
```

VClientMenu

```
package pecl3.screens;

import pecl3.src.conectivity.ClientConection;

/**
 *
 * @author mr.blissfulgrin
 */
public class VClientMenu extends javax.swing.JFrame
{
    private final ClientConection clientConection;

    public VClientMenu(ClientConection clientConection)
    {
        initComponents();
        this.clientConection = clientConection;
    }

    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents()
    {
        java.awt.GridBagConstraints gridBagConstraints;

        jPanel1 = new javax.swing.JPanel();
        jPanel2 = new javax.swing.JPanel();
        vSimulation = new javax.swing.JButton();
        sSimulation = new javax.swing.JButton();
        salir = new javax.swing.JButton();


setDefaultCloseOperation(javax.swing.WindowConstants.DO_NOTHING_ON_CLOSE);
        getContentPane().setLayout(new java.awt.CardLayout());

        jPanel1.setLayout(new java.awt.CardLayout());
```

```
jPanel2.setLayout(new java.awt.GridBagLayout());

vSimulation.setBackground(new java.awt.Color(0, 0, 0));
vSimulation.setFont(new java.awt.Font("Phosphate", 1, 48)); // NOI18N
vSimulation.setForeground(new java.awt.Color(0, 153, 204));
vSimulation.setText("VISUAL SIMULATION");
vSimulation.addActionListener(new java.awt.event.ActionListener()
{
   public void actionPerformed(java.awt.event.ActionEvent evt)
   {
      vSimulationActionPerformed(evt);
   }
});
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 0;
gridBagConstraints.ipady = -6;
gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
gridBagConstraints.insets = new java.awt.Insets(3, 9, 3, 9);
jPanel2.add(vSimulation, gridBagConstraints);

sSimulation.setBackground(new java.awt.Color(0, 0, 0));
sSimulation.setFont(new java.awt.Font("Phosphate", 0, 36)); // NOI18N
sSimulation.setForeground(new java.awt.Color(102, 255, 255));
sSimulation.setText("SIMPLE SIMULATION");
sSimulation.addActionListener(new java.awt.event.ActionListener()
{
   public void actionPerformed(java.awt.event.ActionEvent evt)
   {
      sSimulationActionPerformed(evt);
   }
});
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 1;
gridBagConstraints.ipadx = 106;
gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
gridBagConstraints.insets = new java.awt.Insets(44, 6, 0, 6);
jPanel2.add(sSimulation, gridBagConstraints);

salir.setBackground(new java.awt.Color(0, 0, 0));
salir.setFont(new java.awt.Font("Phosphate", 1, 48)); // NOI18N
salir.setForeground(new java.awt.Color(255, 0, 0));
salir.setText("EXIT");
```

```
    salir.addActionListener(new java.awt.event.ActionListener()
    {
        public void actionPerformed(java.awt.event.ActionEvent evt)
        {
            salirActionPerformed(evt);
        }
    });
    gridBagConstraints = new java.awt.GridBagConstraints();
    gridBagConstraints.gridx = 0;
    gridBagConstraints.gridy = 2;
    gridBagConstraints.ipadx = 329;
    gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
    gridBagConstraints.insets = new java.awt.Insets(43, 9, 9, 9);
    jPanel2.add(salir, gridBagConstraints);

    jPanel1.add(jPanel2, "card2");

    getContentPane().add(jPanel1, "card2");

    pack();
    setLocationRelativeTo(null);
}// </editor-fold>

private void vSimulationActionPerformed(java.awt.event.ActionEvent evt)
{
    VLoad l = new VLoad();
    l.go("CONECTING...");
    VFuelStationClient v = new VFuelStationClient(clientConection);
    clientConection.setDisplay(v);
    v.setVisible(true);
    this.setVisible(false);
    this.dispose();
}

private void sSimulationActionPerformed(java.awt.event.ActionEvent evt)
{
    VLoad l = new VLoad();
    l.go("CONECTING...");
    SFuelStationClient s = new SFuelStationClient(clientConection);
    clientConection.setDisplay(s);
    s.setVisible(true);
    this.setVisible(false);
    this.dispose();
}
```

```
    private void salirActionPerformed(java.awt.event.ActionEvent evt)
    {
        VLoad l = new VLoad();
        l.go("DESCONECTING...");
        VInicio vInicio = new VInicio();
        vInicio.setVisible(true);
        this.setVisible(false);
        this.dispose();
    }

    // Variables declaration - do not modify
    private javax.swing.JPanel jPanel1;
    private javax.swing.JPanel jPanel2;
    private javax.swing.JButton sSimulation;
    private javax.swing.JButton salir;
    private javax.swing.JButton vSimulation;
    // End of variables declaration
}
```

VFuelStation

```
package pecl3.screens;

import java.io.IOException;
import java.net.InetAddress;
import javax.swing.ImageIcon;
import pecl3.src.FuelStation;


/**
 *
 * @author mr.blissfulgrin
 */
public class VFuelStation extends javax.swing.JFrame implements FuelStationInterface
{

   boolean general;
   FuelStation fuelStation;
   VPump [] pump;
   VCarBuffer car;
   VWorkerBuffer worker;
   VICON icon;

   public VFuelStation(int workers, int cars)
   {
      initComponents();

      try
      {
         ip.setText("Connect to: " + InetAddress.getLocalHost().getHostAddress());
      }
      catch (IOException e){}

      fuelStation = new FuelStation(this, workers, cars);
      general = false; //Barrier Status
      barrier.setIcon(new ImageIcon("./src/img/BARRERA.png"));

      //Generate the BackGround
                                          VBackGroundPanel backGround = new
VBackGroundPanel(this.getWidth(),this.getHeight(),"GASOLINERA",0,0);

      //Generate The Visual Pumps
```

```
    pump = new VPump [FuelStation.PUMPS];
    int j;
    for (int i = 0; i < pump.length; i++)
    {
       j = (i >= pump.length/2)? 1:0;
       pump[i] = new VPump(115+300*i-1200*j,495+205*j,i+1);
       this.add(pump[i]);
    }

    //Generate the Visual cars
    car = new VCarBuffer(130,305);
    this.add(car);

    //Generate the Visual Workers
    worker = new VWorkerBuffer(1215,500);
    for (int i = 0; i < fuelStation.getWorkers(); i++)
    {
       worker.newWorker(i+1);
    }
    this.add(worker);

    //Generate the end Icon
    icon = new VICON();
    this.add(icon);
    icon.setVisible(false);

    this.add(backGround);
    this.pack();
}

@Override
public synchronized void carArrivesFuelStation(int number)
{
    car.newCar(number);
    this.repaint();
}
@Override
public synchronized void carLeavesFuelStationQueue(int number)
{
    car.removeCar(number);
    this.repaint();
}
@Override
public synchronized void carReachesPump (int carNumber, int pumpNumber)
```

```
{
   pump[pumpNumber-1].newCar(carNumber);
   this.repaint();
}
@Override
public synchronized void workerStartFueling (int workerNumber, int pumpNumber)
{
   worker.removeWorker(workerNumber);
   pump[pumpNumber-1].newWorker(workerNumber);
   this.repaint();
}
@Override
public synchronized void workerEndsFueling (int workerNumber, int pumpNumber)
{
   pump[pumpNumber-1].removeWorker(workerNumber);
   worker.newWorker(workerNumber);
   this.repaint();
}
@Override
public synchronized void carLeavesPump (int carNumber, int pumpNumber)
{
   pump[pumpNumber-1].removeCar(carNumber);
   this.repaint();
}
@Override
public synchronized void endSimulation ()
{
   icon.setVisible(true);
   this.repaint();
}
@Override
public synchronized void write(String texto)
{
   txt.setText(texto);
}


@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">
private void initComponents()
{

   jLabel1 = new javax.swing.JLabel();
   barrier = new javax.swing.JLabel();
```

```
        jLabel3 = new javax.swing.JLabel();
        timeSlider = new javax.swing.JSlider();
        timeTxt = new javax.swing.JLabel();
        jScrollPane1 = new javax.swing.JScrollPane();
        txt = new javax.swing.JTextArea();
        ip = new javax.swing.JLabel();

        setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
        setTitle("FuelStation");
        setResizable(false);
        addWindowListener(new java.awt.event.WindowAdapter()
        {
            public void windowOpened(java.awt.event.WindowEvent evt)
            {
                formWindowOpened(evt);
            }
            public void windowClosing(java.awt.event.WindowEvent evt)
            {
                formWindowClosing(evt);
            }
        });

                    jLabel1.setIcon(new   javax.swing.ImageIcon(getClass().getResource("/img/
SING.png"))); // NOI18N

                     barrier.setIcon(new   javax.swing.ImageIcon(getClass().getResource("/img/
BARRERA.png"))); // NOI18N
        barrier.addMouseListener(new java.awt.event.MouseAdapter()
        {
            public void mouseClicked(java.awt.event.MouseEvent evt)
            {
                barrierMouseClicked(evt);
            }
        });

        jLabel3.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);

        timeSlider.setMaximum(10000);
        timeSlider.setValue(5500);
        timeSlider.addChangeListener(new javax.swing.event.ChangeListener()
        {
            public void stateChanged(javax.swing.event.ChangeEvent evt)
            {
                timeSliderStateChanged(evt);
```

```
        }
    });

    timeTxt.setFont(new java.awt.Font("Phosphate", 1, 48)); // NOI18N
    timeTxt.setForeground(new java.awt.Color(0, 0, 0));
    timeTxt.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
    timeTxt.setText("5500");

    txt.setEditable(false);
    txt.setColumns(20);
    txt.setFont(new java.awt.Font("Lucida Grande", 0, 9)); // NOI18N
    txt.setRows(5);
    jScrollPane1.setViewportView(txt);

    ip.setFont(new java.awt.Font("Tahoma", 1, 26)); // NOI18N
    ip.setHorizontalAlignment(javax.swing.SwingConstants.LEFT);
    ip.setText("Conect to: ");

                                        javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(getContentPane());
    getContentPane().setLayout(layout);
    layout.setHorizontalGroup(
      layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGroup(layout.createSequentialGroup()
        .addContainerGap()
         .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LE
ADING)
          .addGroup(layout.createSequentialGroup()
            .addGap(0, 0, Short.MAX_VALUE)
              .addComponent(timeSlider, javax.swing.GroupLayout.PREFERRED_SIZE,
910, javax.swing.GroupLayout.PREFERRED_SIZE)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATE
D)
            .addComponent(timeTxt, javax.swing.GroupLayout.PREFERRED_SIZE, 152,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addGap(425, 425, 425)
            .addComponent(jLabel3))
                                .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
            .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE,
409, javax.swing.GroupLayout.PREFERRED_SIZE)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATE
D)
            .addComponent(jLabel1)
```

```
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addGroup(layout.createSequentialGroup()
                        .addGap(55, 55, 55)
                        .addComponent(barrier))
                    .addGroup(layout.createSequentialGroup()
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(ip)))
                .addContainerGap(88, Short.MAX_VALUE))))
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING, false)
                    .addGroup(layout.createSequentialGroup()
                        .addGap(20, 20, 20)
                        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                            .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE, 193, javax.swing.GroupLayout.PREFERRED_SIZE)
                            .addComponent(jLabel1, javax.swing.GroupLayout.PREFERRED_SIZE, 165, javax.swing.GroupLayout.PREFERRED_SIZE))
                        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                            .addGroup(layout.createSequentialGroup()
                                .addGap(210, 210, 210)
                                .addComponent(timeTxt))
                            .addGroup(layout.createSequentialGroup()
                                .addGap(228, 228, 228)
                                .addComponent(timeSlider, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))))
                    .addGroup(layout.createSequentialGroup()
                        .addContainerGap()
                        .addComponent(ip)
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
                            .addGroup(layout.createSequentialGroup()
                                .addComponent(jLabel3)
```

```
                    .addGap(427, 427, 427))
                .addComponent(barrier))))
        .addContainerGap(479, Short.MAX_VALUE))
    );

    pack();
    setLocationRelativeTo(null);
}// </editor-fold>

private void barrierMouseClicked(java.awt.event.MouseEvent evt)
{
    general = !general;
    if (general)
    {
        barrier.setIcon(new ImageIcon("./src/img/BARRERA_ALTA.png"));
        fuelStation.resumeSimulation();
    }
    else
    {
        barrier.setIcon(new ImageIcon("./src/img/BARRERA.png"));
        fuelStation.pauseSimulation();
    }
    this.repaint();
}

private void formWindowOpened(java.awt.event.WindowEvent evt)
{
    fuelStation.start();
}

private void formWindowClosing(java.awt.event.WindowEvent evt)
{
    VLoad l = new VLoad();
    l.go("SAVING LOG...");
    fuelStation.end();
    VInicio i = new VInicio ();
    i.setVisible(true);
}

private void timeSliderStateChanged(javax.swing.event.ChangeEvent evt)
{
    fuelStation.setTime(timeSlider.getValue());
    timeTxt.setText(String.valueOf(timeSlider.getValue()));
    this.repaint();
```

```
}

// Variables declaration - do not modify
private javax.swing.JLabel barrier;
private javax.swing.JLabel ip;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel3;
private javax.swing.JScrollPane jScrollPane1;
private javax.swing.JSlider timeSlider;
private javax.swing.JLabel timeTxt;
private javax.swing.JTextArea txt;
// End of variables declaration

@Override
public synchronized void barrera(boolean state)
{
   general = state;
   if (state)
   {
      barrier.setIcon(new ImageIcon("./src/img/BARRERA_ALTA.png"));
   }
   else
   {
      barrier.setIcon(new ImageIcon("./src/img/BARRERA.png"));
   }
   this.repaint();
}

@Override
public synchronized void reset()
{
   car.reset();
   worker.reset();
   for (VPump p : pump)
   {
      p.reset();
   }
   this.repaint();
}

@Override
public synchronized void setData(String[] data)
{
   int index = 1;
```

```
        int times1;
        int times2;
        int gap1;
        int gap2;
        barrera(data[0].trim().equals("1"));
        times1 = Integer.parseInt(data[index].trim());
        index++;
        gap1 = index;
        for (; index < times1+gap1; index++)
        {
           car.newCar(Integer.parseInt(data[index].trim()));
        }
        times2 = Integer.parseInt(data[index].trim());
        index++;
        gap2 = index;
        for (; index < times2+gap2; index++)
        {
           worker.newWorker(Integer.parseInt(data[index].trim()));
        }
        for (VPump p : pump)
        {
           if (!data[index].trim().equals("N"))
           {
              p.newCar(Integer.parseInt(data[index].trim()));
           }
           index++;
           if (!data[index].trim().equals("N"))
           {
              p.newWorker(Integer.parseInt(data[index].trim()));
           }
           index++;
        }

        this.repaint();
     }
}
```

VFuelStationClient

```
package pecl3.screens;

import javax.swing.ImageIcon;
import pecl3.src.FuelStation;
import pecl3.src.conectivity.ClientConection;


/**
 *
 * @author mr.blissfulgrin
 */
public class VFuelStationClient extends javax.swing.JFrame implements
FuelStationInterface
{

   boolean general;
   VPump [] pump;
   VCarBuffer car;
   VWorkerBuffer worker;
   VICON icon;
   private final ClientConection conection;

   public VFuelStationClient(ClientConection conection)
   {
      initComponents();

      this.conection = conection;
      general = false; //Barrier Status
      barrier.setIcon(new ImageIcon("./src/img/BARRERA.png"));

      //Generate the BackGround
                                          VBackGroundPanel backGround = new
VBackGroundPanel(this.getWidth(),this.getHeight(),"GASOLINERA",0,0);

      //Generate The Visual Pumps
      pump = new VPump [FuelStation.PUMPS];
      int j;
      for (int i = 0; i < pump.length; i++)
      {
         j = (i >= pump.length/2)? 1:0;
         pump[i] = new VPump(115+300*i-1200*j,495+205*j,i+1);
```

```
        this.add(pump[i]);
    }

    //Generate the Visual cars
    car = new VCarBuffer(130,305);
    this.add(car);

    //Generate the Visual Workers
    worker = new VWorkerBuffer(1215,500);
    /*
    for (int i = 0; i < 6; i++)
    {
        worker.newWorker(i+1);
    }*/

    this.add(worker);
    //Generate the end Icon
    icon = new VICON();
    this.add(icon);
    icon.setVisible(false);
    txt.setText("");

    this.add(backGround);
    this.pack();
}

@Override
public synchronized void carArrivesFuelStation(int number)
{
    car.newCar(number);
    this.repaint();
}
@Override
public synchronized void carLeavesFuelStationQueue(int number)
{
    car.removeCar(number);
    this.repaint();
}
@Override
public synchronized void carReachesPump (int carNumber, int pumpNumber)
{
    pump[pumpNumber-1].newCar(carNumber);
    this.repaint();
}
```

```
@Override
public synchronized void workerStartFueling (int workerNumber, int pumpNumber)
{
   worker.removeWorker(workerNumber);
   pump[pumpNumber-1].newWorker(workerNumber);
   this.repaint();
}
@Override
public synchronized void workerEndsFueling (int workerNumber, int pumpNumber)
{
   pump[pumpNumber-1].removeWorker(workerNumber);
   worker.newWorker(workerNumber);
   this.repaint();
}
@Override
public synchronized void carLeavesPump (int carNumber, int pumpNumber)
{
   pump[pumpNumber-1].removeCar(carNumber);
   this.repaint();
}
@Override
public synchronized void endSimulation ()
{
   icon.setVisible(true);
   this.repaint();
}
@Override
public synchronized void write(String texto)
{
   txt.setText(txt.getText()+texto+"\n");
}


@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">
private void initComponents()
{

   jLabel1 = new javax.swing.JLabel();
   barrier = new javax.swing.JLabel();
   jLabel3 = new javax.swing.JLabel();
   jScrollPane1 = new javax.swing.JScrollPane();
   txt = new javax.swing.JTextArea();
   id = new javax.swing.JLabel();
```

```
        setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
        setTitle("FuelStation");
        setResizable(false);
        addWindowListener(new java.awt.event.WindowAdapter()
        {
            public void windowClosing(java.awt.event.WindowEvent evt)
            {
                formWindowClosing(evt);
            }
        });

                    jLabel1.setIcon(new   javax.swing.ImageIcon(getClass().getResource("/img/
SING.png"))); // NOI18N

                     barrier.setIcon(new   javax.swing.ImageIcon(getClass().getResource("/img/
BARRERA.png"))); // NOI18N
        barrier.addMouseListener(new java.awt.event.MouseAdapter()
        {
            public void mouseClicked(java.awt.event.MouseEvent evt)
            {
                barrierMouseClicked(evt);
            }
        });

        jLabel3.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);

        txt.setEditable(false);
        txt.setColumns(20);
        txt.setFont(new java.awt.Font("Lucida Grande", 0, 9)); // NOI18N
        txt.setRows(5);
        jScrollPane1.setViewportView(txt);

        id.setFont(new java.awt.Font("Tahoma", 1, 30)); // NOI18N
        id.setText("ID: ");

                                            javax.swing.GroupLayout      layout      =      new
javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
```

```
            .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE, 409,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addGap(30, 30, 30)
            .addComponent(jLabel1)
             .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LE
ADING)
               .addGroup(layout.createSequentialGroup()
                  .addGap(348, 348, 348)
                  .addComponent(jLabel3))
               .addGroup(layout.createSequentialGroup()
                  .addGap(34, 34, 34)
                  .addComponent(barrier))
               .addGroup(layout.createSequentialGroup()
                  .addGap(43, 43, 43)
                  .addComponent(id)))
            .addGap(95, 95, 95))
      );
      layout.setVerticalGroup(
         layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
         .addGroup(layout.createSequentialGroup()
             .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TR
AILING)
               .addGroup(layout.createSequentialGroup()
                  .addGap(20, 20, 20)
                   .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignmen
t.LEADING)
                                                        .addComponent(jScrollPane1,
j a v a x . s w i n g . G r o u p L a y o u t . P R E F E R R E D _ S I Z E ,  1 9 3 ,
javax.swing.GroupLayout.PREFERRED_SIZE)
                      .addComponent(jLabel1, javax.swing.GroupLayout.PREFERRED_SIZE,
165, javax.swing.GroupLayout.PREFERRED_SIZE))
                  .addGap(271, 271, 271))
               .addGroup(layout.createSequentialGroup()
                   .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignmen
t.TRAILING, false)
                     .addGroup(layout.createSequentialGroup()
                        .addGap(57, 57, 57)
                        .addComponent(jLabel3)
                        .addGap(117, 117, 117))
                                 .addGroup(javax.swing.GroupLayout.Alignment.LEADING,
layout.createSequentialGroup()
                        .addGap(20, 20, 20)
                        .addComponent(id)
```

```
                        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.REL
ATED, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)))
                    .addComponent(barrier)))
                .addContainerGap(476, Short.MAX_VALUE))
        );

        pack();
        setLocationRelativeTo(null);
    }// </editor-fold>

    private void barrierMouseClicked(java.awt.event.MouseEvent evt)
    {
        if (general)
        {
            conection.pauseSimulation();
        }
        else
        {
            conection.resumeSimulation();
        }
    }

    private void formWindowClosing(java.awt.event.WindowEvent evt)
    {
        VLoad l = new VLoad();
        l.go("DESCONECTING...");
        conection.end();
        VInicio i = new VInicio ();
        i.setVisible(true);
    }

    // Variables declaration - do not modify
    private javax.swing.JLabel barrier;
    private javax.swing.JLabel id;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JTextArea txt;
    // End of variables declaration

    @Override
    public synchronized void reset()
    {
        car.reset();
```

```
    worker.reset();
    for (VPump p : pump)
    {
        p.reset();
    }
    this.repaint();
}

@Override
public synchronized void barrera(boolean state)
{
    if (state)
    {
        barrier.setIcon(new ImageIcon("./src/img/BARRERA_ALTA.png"));
        general = true;
    }
    else
    {
        barrier.setIcon(new ImageIcon("./src/img/BARRERA.png"));
        general = false;
    }
}

@Override
public synchronized void setData(String[] data)
{
    int index = 1;
    int times1;
    int times2;
    int gap1;
    int gap2;
    barrera(data[0].trim().equals("1"));
    times1 = Integer.parseInt(data[index].trim());
    index++;
    gap1 = index;
    for (; index < times1+gap1; index++)
    {
        car.newCar(Integer.parseInt(data[index].trim()));
    }
    times2 = Integer.parseInt(data[index].trim());
    index++;
    gap2 = index;
    for (; index < times2+gap2; index++)
    {
```

```
            worker.newWorker(Integer.parseInt(data[index].trim()));
        }
        for (VPump p : pump)
        {
            if (!data[index].trim().equals("N"))
            {
                p.newCar(Integer.parseInt(data[index].trim()));
            }
            index++;
            if (!data[index].trim().equals("N"))
            {
                p.newWorker(Integer.parseInt(data[index].trim()));
            }
            index++;
        }
        id.setText("ID: "+data[index].trim());
        this.repaint();
    }
}
```

VHouse

```
package pecl3.screens;

/**
 *
 * @author mr.blissfulgrin
 */
public class VHouse extends VBackGroundPanel
{

    public VHouse()
    {
        super(256,256,"HOUSE",0,0);
        initComponents();
    }

    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents()
    {

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(this);
        this.setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGap(0, 400, Short.MAX_VALUE)
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGap(0, 300, Short.MAX_VALUE)
        );
    }// </editor-fold>


    // Variables declaration - do not modify
    // End of variables declaration
}
```

VIcon

```
package pecl3.screens;

/**
 *
 * @author mr.blissfulgrin
 */
public class VICON extends VBackGroundPanel
{

    public VICON()
    {
        super(420,313,"ICON",1300,0);
        initComponents();
    }


    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents()
    {

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(this);
        this.setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGap(0, 400, Short.MAX_VALUE)
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGap(0, 300, Short.MAX_VALUE)
        );
    }// </editor-fold>

    // Variables declaration - do not modify
    // End of variables declaration
}
```

VInicio

```
package pecl3.screens;

import pecl3.src.conectivity.ClientConection;
import pecl3.src.conectivity.SenderUDPClient;


/**
 *
 * @author mr.blissfulgrin
 */
public class VInicio extends javax.swing.JFrame
{

    private int carNumber;
    private int workerNumber;

    public VInicio()
    {
        initComponents();
        carNumber = 200;
        workerNumber = 3;
        car.setValue(carNumber);
        carTxt.setText("Cars: "+ carNumber);
        worker.setText("Workers: "+ workerNumber);
    }
     /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents()
    {
        java.awt.GridBagConstraints gridBagConstraints;

        vSimulation = new javax.swing.JButton();
        log = new javax.swing.JButton();
        salir = new javax.swing.JButton();
        sSimulation = new javax.swing.JButton();
        jPanel1 = new javax.swing.JPanel();
```

```
        car = new javax.swing.JSlider();
        jPanel2 = new javax.swing.JPanel();
        lessWorkers = new javax.swing.JButton();
        worker = new javax.swing.JLabel();
        moreWorkers = new javax.swing.JButton();
        carTxt = new javax.swing.JLabel();
        onLine = new javax.swing.JButton();


setDefaultCloseOperation(javax.swing.WindowConstants.DO_NOTHING_ON_CLOSE);
        setTitle("FuelStation");
        setResizable(false);
        getContentPane().setLayout(new java.awt.GridBagLayout());

        vSimulation.setBackground(new java.awt.Color(0, 0, 0));
        vSimulation.setFont(new java.awt.Font("Phosphate", 1, 48)); // NOI18N
        vSimulation.setForeground(new java.awt.Color(0, 153, 204));
        vSimulation.setText("VISUAL SIMULATION");
        vSimulation.addActionListener(new java.awt.event.ActionListener()
        {
            public void actionPerformed(java.awt.event.ActionEvent evt)
            {
                vSimulationActionPerformed(evt);
            }
        });
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 0;
        gridBagConstraints.gridy = 0;
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
        getContentPane().add(vSimulation, gridBagConstraints);

        log.setBackground(new java.awt.Color(0, 0, 0));
        log.setFont(new java.awt.Font("Phosphate", 1, 48)); // NOI18N
        log.setForeground(new java.awt.Color(0, 153, 51));
        log.setText("VIEW LAST LOG");
        log.addActionListener(new java.awt.event.ActionListener()
        {
            public void actionPerformed(java.awt.event.ActionEvent evt)
            {
                logActionPerformed(evt);
            }
        });
        gridBagConstraints = new java.awt.GridBagConstraints();
```

```
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 3;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
getContentPane().add(log, gridBagConstraints);

salir.setBackground(new java.awt.Color(0, 0, 0));
salir.setFont(new java.awt.Font("Phosphate", 1, 48)); // NOI18N
salir.setForeground(new java.awt.Color(255, 0, 0));
salir.setText("EXIT");
salir.addActionListener(new java.awt.event.ActionListener()
{
   public void actionPerformed(java.awt.event.ActionEvent evt)
   {
      salirActionPerformed(evt);
   }
});
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 4;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
getContentPane().add(salir, gridBagConstraints);

sSimulation.setBackground(new java.awt.Color(0, 0, 0));
sSimulation.setFont(new java.awt.Font("Phosphate", 0, 36)); // NOI18N
sSimulation.setForeground(new java.awt.Color(102, 255, 255));
sSimulation.setText("SIMPLE SIMULATION");
sSimulation.addActionListener(new java.awt.event.ActionListener()
{
   public void actionPerformed(java.awt.event.ActionEvent evt)
   {
      sSimulationActionPerformed(evt);
   }
});
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 1;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
getContentPane().add(sSimulation, gridBagConstraints);

jPanel1.setLayout(new java.awt.GridBagLayout());
```

```
car.setMaximum(1000);
car.setMinimum(50);
car.setPaintLabels(true);
car.setValue(200);
car.addChangeListener(new javax.swing.event.ChangeListener()
{
   public void stateChanged(javax.swing.event.ChangeEvent evt)
   {
      carStateChanged(evt);
   }
});
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 1;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
gridBagConstraints.weightx = 5.0;
gridBagConstraints.insets = new java.awt.Insets(0, 8, 0, 8);
jPanel1.add(car, gridBagConstraints);

jPanel2.setLayout(new java.awt.GridBagLayout());

lessWorkers.setText("-");
lessWorkers.addActionListener(new java.awt.event.ActionListener()
{
   public void actionPerformed(java.awt.event.ActionEvent evt)
   {
      lessWorkersActionPerformed(evt);
   }
});
jPanel2.add(lessWorkers, new java.awt.GridBagConstraints());

worker.setText("3");
jPanel2.add(worker, new java.awt.GridBagConstraints());

moreWorkers.setText("+");
moreWorkers.addActionListener(new java.awt.event.ActionListener()
{
   public void actionPerformed(java.awt.event.ActionEvent evt)
   {
      moreWorkersActionPerformed(evt);
   }
});
jPanel2.add(moreWorkers, new java.awt.GridBagConstraints());
```

```
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 1;
gridBagConstraints.gridy = 0;
gridBagConstraints.gridheight = 2;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
gridBagConstraints.weightx = 1.0;
jPanel1.add(jPanel2, gridBagConstraints);

carTxt.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
carTxt.setText("Cars");
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 0;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
jPanel1.add(carTxt, gridBagConstraints);

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 5;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
gridBagConstraints.insets = new java.awt.Insets(0, 0, 8, 0);
getContentPane().add(jPanel1, gridBagConstraints);

onLine.setBackground(new java.awt.Color(0, 0, 0));
onLine.setFont(new java.awt.Font("Phosphate", 0, 48)); // NOI18N
onLine.setForeground(new java.awt.Color(255, 153, 0));
onLine.setText("OnLine");
onLine.setMaximumSize(new java.awt.Dimension(374, 67));
onLine.setMinimumSize(new java.awt.Dimension(374, 67));
onLine.setPreferredSize(new java.awt.Dimension(374, 67));
onLine.addActionListener(new java.awt.event.ActionListener()
{
   public void actionPerformed(java.awt.event.ActionEvent evt)
   {
      onLineActionPerformed(evt);
   }
});
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 2;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
getContentPane().add(onLine, gridBagConstraints);
```

```
    getAccessibleContext().setAccessibleDescription("FuelStation");

    pack();
    setLocationRelativeTo(null);
}// </editor-fold>

private void salirActionPerformed(java.awt.event.ActionEvent evt)
{
    VLoad l = new VLoad();
    l.go("EXITING...");
    this.setVisible(false);
    this.dispose();
    System.exit(0);
}

private void logActionPerformed(java.awt.event.ActionEvent evt)
{
   VLoad l = new VLoad();
    l.go("LOADING...");
   this.setVisible(false);
   this.dispose();
   new VLog().setVisible(true);
}

private void vSimulationActionPerformed(java.awt.event.ActionEvent evt)
{
    VLoad l = new VLoad();
    l.go("LOADING...");
    VFuelStation v = new VFuelStation(workerNumber,carNumber);
    this.setVisible(false);
    this.dispose();
    v.setVisible(true);
}

private void sSimulationActionPerformed(java.awt.event.ActionEvent evt)
{
    VLoad l = new VLoad();
    l.go("LOADING...");
    this.setVisible(false);
    this.dispose();
    SFuelStation s = new SFuelStation(workerNumber,carNumber);
    s.setVisible(true);
}
```

```
    private void lessWorkersActionPerformed(java.awt.event.ActionEvent evt)
    {
        if (workerNumber > 1)
        {
            workerNumber --;
            worker.setText("Workers: " + workerNumber);
        }
    }

    private void moreWorkersActionPerformed(java.awt.event.ActionEvent evt)
    {
        if (workerNumber < 6)
        {
            workerNumber ++;
            worker.setText("Workers: " + workerNumber);
        }
    }

    private void carStateChanged(javax.swing.event.ChangeEvent evt)
    {
        carTxt.setText("Cars: " + car.getValue());
        carNumber = car.getValue();
    }

    private void onLineActionPerformed(java.awt.event.ActionEvent evt)
    {
        VLoad l = new VLoad();
        l.go("AUTO CONECTING...");

        SenderUDPClient sender = new SenderUDPClient();
        sender.start();
        try
        {
            sender.join();
        }
        catch (InterruptedException ex){}

        byte [] IP = sender.getAutoIP();
        ClientConection clientConection = new ClientConection();
        if (!clientConection.conectable(IP))
        {
                                        VRemoteConection vRemoteConection = new
VRemoteConection(clientConection);
            vRemoteConection.setVisible(true);
```

```
        }
        this.dispose();
    }

    // Variables declaration - do not modify
    private javax.swing.JSlider car;
    private javax.swing.JLabel carTxt;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JPanel jPanel2;
    private javax.swing.JButton lessWorkers;
    private javax.swing.JButton log;
    private javax.swing.JButton moreWorkers;
    private javax.swing.JButton onLine;
    private javax.swing.JButton sSimulation;
    private javax.swing.JButton salir;
    private javax.swing.JButton vSimulation;
    private javax.swing.JLabel worker;
    // End of variables declaration
}
```

VLoad

```
package pecl3.screens;

/**
 *
 * @author mr.blissfulgrin
 */
public class VLoad extends javax.swing.JFrame {

    public VLoad()
    {
        initComponents();
    }

    public void go (String txt)
    {
        this.txt.setText(txt);
        this.setVisible(true);
        this.update(this.getGraphics());
        this.setVisible(true);

        new LoadController(this,bar).start();
    }


    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents()
    {

        jPanel1 = new javax.swing.JPanel();
        jPanel2 = new javax.swing.JPanel();
        txt = new javax.swing.JLabel();
        bar = new javax.swing.JProgressBar();


setDefaultCloseOperation(javax.swing.WindowConstants.DO_NOTHING_ON_CLOSE);
        setTitle("FuelStation");

        jPanel1.setLayout(new java.awt.CardLayout(20, 20));

        jPanel2.setLayout(new java.awt.GridLayout(0, 1));
```

```
        txt.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
        txt.setToolTipText("");
        jPanel2.add(txt);
        jPanel2.add(bar);

        jPanel1.add(jPanel2, "card2");

                                        javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED_SIZE, 238,
javax.swing.GroupLayout.PREFERRED_SIZE)
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED_SIZE, 100,
javax.swing.GroupLayout.PREFERRED_SIZE)
        );

        pack();
        setLocationRelativeTo(null);
    }// </editor-fold>

    // Variables declaration - do not modify
    private javax.swing.JProgressBar bar;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JPanel jPanel2;
    private javax.swing.JLabel txt;
    // End of variables declaration
}
```

VLog

```
package pecl3.screens;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;

/**
 *
 * @author mr.blissfulgrin
 */
public class VLog extends javax.swing.JFrame
{

    /**
     * Creates new form Log
     */
    public VLog()
    {
        initComponents();
        getDiskInformation();
    }

    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents()
    {
        java.awt.GridBagConstraints gridBagConstraints;

        jPanel1 = new javax.swing.JPanel();
        jPanel2 = new javax.swing.JPanel();
        jScrollPane1 = new javax.swing.JScrollPane();
        list = new javax.swing.JList<>();
        jScrollPane2 = new javax.swing.JScrollPane();
```

```
txt = new javax.swing.JTextArea();

setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
setTitle("FuelStation");
addWindowListener(new java.awt.event.WindowAdapter()
{
   public void windowClosing(java.awt.event.WindowEvent evt)
   {
      formWindowClosing(evt);
   }
});

jPanel1.setLayout(new java.awt.CardLayout(7, 7));

jPanel2.setLayout(new java.awt.GridBagLayout());

list.setSelectionMode(javax.swing.ListSelectionModel.SINGLE_SELECTION);
list.addListSelectionListener(new javax.swing.event.ListSelectionListener()
{
   public void valueChanged(javax.swing.event.ListSelectionEvent evt)
   {
      listValueChanged(evt);
   }
});
jScrollPane1.setViewportView(list);

gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 0;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
gridBagConstraints.ipadx = 22;
gridBagConstraints.ipady = 119;
gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
gridBagConstraints.weightx = 1.0;
gridBagConstraints.weighty = 1.0;
jPanel2.add(jScrollPane1, gridBagConstraints);

txt.setEditable(false);
txt.setColumns(20);
txt.setRows(5);
txt.setCursor(new java.awt.Cursor(java.awt.Cursor.DEFAULT_CURSOR));
txt.setDragEnabled(false);
txt.setFocusable(false);
txt.setOpaque(false);
```

```
        jScrollPane2.setViewportView(txt);

        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridx = 1;
        gridBagConstraints.gridy = 0;
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.ipadx = 223;
        gridBagConstraints.ipady = 63;
        gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
        gridBagConstraints.weightx = 1.0;
        gridBagConstraints.weighty = 1.0;
        jPanel2.add(jScrollPane2, gridBagConstraints);

        jPanel1.add(jPanel2, "card2");

                                            javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                        .addComponent(jPanel1, javax.swing.GroupLayout.DEFAULT_SIZE, 730,
Short.MAX_VALUE)
                .addContainerGap())
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                        .addComponent(jPanel1, javax.swing.GroupLayout.DEFAULT_SIZE, 373,
Short.MAX_VALUE)
                .addContainerGap())
        );

        pack();
        setLocationRelativeTo(null);
    }// </editor-fold>

    private void formWindowClosing(java.awt.event.WindowEvent evt)
    {
        VInicio i = new VInicio();
        i.setVisible(true);
    }
```

```
public void getDiskInformation ()
{
   String listado[]= new File("./src/txt").list();
   ArrayList <Sortable> listadoOrdenar = new ArrayList <>();

   for (String s : listado)
   {
      listadoOrdenar.add(new Sortable(s));
   }

   Collections.sort(listadoOrdenar, new Sorter());
   int counter = 0;

   for (String listado1 : listado)
   {
      if (listado1.charAt(0) == '.')
      {
         counter ++;
      }
   }

   String listadofinal[]= new String [listado.length-counter];

   int counter2 = 0;
   for (int i = 0; i < listado.length; i++)
   {
      if (listadoOrdenar.get(i).getStr().charAt(0) != '.')
      {
         listadofinal [i - counter2] = listadoOrdenar.get(i).getStr();
      }
      else
         counter2++;
   }

   list.setListData(listadofinal);

   try
   {
      txt.setText("Loading... please wait, it may take some time");
      String buff = "";
      String cadena;
      try(FileReader f = new FileReader("./src/txt/evoluvionGasolinera.txt"))
      {
         try (BufferedReader b = new BufferedReader(f))
```

```
            {
                while((cadena = b.readLine())!=null)
                {
                    buff = (buff+"\n"+cadena);
                }
                b.close();
                txt.setText(buff);
            }
            f.close();
        }
    }
    catch(IOException e)
    {
        System.out.println("IO EXCEPTION IN VLOG" + e.toString());
    }
}
private void listValueChanged(javax.swing.event.ListSelectionEvent evt)
{
    if (!"".equals(list.getSelectedValue()))
    {
        try
        {
            txt.setText("Loading... please wait, it may take some time");
            String buff = "";
            String cadena;
            FileReader f = new FileReader("./src/txt/"+list.getSelectedValue());
            try (BufferedReader b = new BufferedReader(f))
            {
                while((cadena = b.readLine())!=null)
                {
                    buff = (buff+"\n"+cadena);
                }
                b.close();
                txt.setText(buff);
            }
        }
        catch(IOException e)
        {
            System.out.println("IO EXCEPTION IN VLOG" + e.toString());
        }
    }
}

// Variables declaration - do not modify
```

```
    private javax.swing.JPanel jPanel1;
    private javax.swing.JPanel jPanel2;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JScrollPane jScrollPane2;
    private javax.swing.JList<String> list;
    private javax.swing.JTextArea txt;
    // End of variables declaration
}
```

VPump

```
package pecl3.screens;


/**
 *
 * @author mr.blissfulgrin
 */
public class VPump extends VBackGroundPanel
{
   VWorker worker;
   VCar car;

   public VPump(int x, int y, int id)
   {
      super(200,200,"GASOLINE",x,y);
      initComponents();
      this.ID.setText("Pump: "+id);
   }

   public void reset()
   {
      if (car != null)
      {
         car.setVisible(false);
         car.remove(car);
         this.repaint();
      }
      if (worker != null)
      {
         worker.setVisible(false);
         worker.remove(worker);
         this.repaint();
      }
   }

   public void newCar (int number)
   {
      car = new VCar();
      car.newCar(number);
      this.add(car);
      this.repaint();
```

```
}
public void newWorker (int number)
{
   worker = new VWorker();
   worker.newWorker(number);
   this.add(worker);
   this.repaint();
}
public void removeCar(int number)
{
   if(car.getNumber() == number)
   {
      car.setVisible(false);
      car.remove(car);
   }
   this.repaint();
}
public void removeWorker(int number)
{
   if(worker.getNumber() == number)
   {
      worker.setVisible(false);
      worker.remove(worker);
   }
   this.repaint();
}

/**
 * This method is called from within the constructor to initialize the form.
 * WARNING: Do NOT modify this code. The content of this method is always
 * regenerated by the Form Editor.
 */
@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">
private void initComponents()
{

   ID = new javax.swing.JLabel();

   ID.setFont(new java.awt.Font("Silom", 1, 20)); // NOI18N
   ID.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
   ID.setText("Pump: ");

   javax.swing.GroupLayout layout = new javax.swing.GroupLayout(this);
```

```
        this.setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addGap(36, 36, 36)
                .addComponent(ID)
                .addContainerGap(119, Short.MAX_VALUE))
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addGap(33, 33, 33)
                .addComponent(ID)
                .addContainerGap(241, Short.MAX_VALUE))
        );
    }// </editor-fold>


    // Variables declaration - do not modify
    private javax.swing.JLabel ID;
    // End of variables declaration
}
```

VRemoteConection

```
package pecl3.screens;

import pecl3.src.conectivity.ClientConection;

/**
 *
 * @author mr.blissfulgrin
 */
public class VRemoteConection extends javax.swing.JFrame
{

    ClientConection clientConection;

    public VRemoteConection(ClientConection clientConection)
    {
        initComponents();
        this.clientConection = clientConection;
    }

    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents()
    {
        java.awt.GridBagConstraints gridBagConstraints;

        jPanel1 = new javax.swing.JPanel();
        jPanel2 = new javax.swing.JPanel();
        ip1 = new javax.swing.JTextField();
        ip2 = new javax.swing.JTextField();
        ip3 = new javax.swing.JTextField();
        ip4 = new javax.swing.JTextField();
        txt = new javax.swing.JLabel();
        btt = new javax.swing.JButton();
        jLabel1 = new javax.swing.JLabel();

        setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
```

```
addWindowListener(new java.awt.event.WindowAdapter()
{
   public void windowClosing(java.awt.event.WindowEvent evt)
   {
      formWindowClosing(evt);
   }
});
getContentPane().setLayout(new java.awt.CardLayout());

jPanel1.setLayout(new java.awt.CardLayout());

jPanel2.setLayout(new java.awt.GridBagLayout());
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 1;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
gridBagConstraints.ipadx = 70;
gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
gridBagConstraints.insets = new java.awt.Insets(29, 9, 0, 9);
jPanel2.add(ip1, gridBagConstraints);
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 1;
gridBagConstraints.gridy = 1;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
gridBagConstraints.ipadx = 70;
gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
gridBagConstraints.insets = new java.awt.Insets(29, 9, 0, 9);
jPanel2.add(ip2, gridBagConstraints);
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 2;
gridBagConstraints.gridy = 1;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
gridBagConstraints.ipadx = 70;
gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
gridBagConstraints.insets = new java.awt.Insets(29, 9, 0, 9);
jPanel2.add(ip3, gridBagConstraints);
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 3;
gridBagConstraints.gridy = 1;
gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
gridBagConstraints.ipadx = 70;
gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
gridBagConstraints.insets = new java.awt.Insets(29, 9, 0, 9);
jPanel2.add(ip4, gridBagConstraints);
```

```
    txt.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
    gridBagConstraints = new java.awt.GridBagConstraints();
    gridBagConstraints.gridx = 1;
    gridBagConstraints.gridy = 2;
    gridBagConstraints.gridwidth = 2;
    gridBagConstraints.fill = java.awt.GridBagConstraints.HORIZONTAL;
    gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
    gridBagConstraints.insets = new java.awt.Insets(22, 9, 9, 9);
    jPanel2.add(txt, gridBagConstraints);

    btt.setText("CONECTAR");
    btt.addActionListener(new java.awt.event.ActionListener()
    {
       public void actionPerformed(java.awt.event.ActionEvent evt)
       {
          bttActionPerformed(evt);
       }
    });
    gridBagConstraints = new java.awt.GridBagConstraints();
    gridBagConstraints.gridx = 1;
    gridBagConstraints.gridy = 3;
    gridBagConstraints.gridwidth = 2;
    gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
    gridBagConstraints.anchor = java.awt.GridBagConstraints.NORTHWEST;
    gridBagConstraints.insets = new java.awt.Insets(8, 8, 8, 8);
    jPanel2.add(btt, gridBagConstraints);

    jLabel1.setText("Auto Connection failed, try Manualy");
    gridBagConstraints = new java.awt.GridBagConstraints();
    gridBagConstraints.gridx = 1;
    gridBagConstraints.gridy = 0;
    gridBagConstraints.gridwidth = 2;
    gridBagConstraints.insets = new java.awt.Insets(19, 0, 0, 0);
    jPanel2.add(jLabel1, gridBagConstraints);

    jPanel1.add(jPanel2, "card2");

    getContentPane().add(jPanel1, "card2");

    pack();
    setLocationRelativeTo(null);
}// </editor-fold>
```

```
private void formWindowClosing(java.awt.event.WindowEvent evt)
{
   VInicio vinicio = new VInicio ();
   vinicio.setVisible(true);
}

private void bttActionPerformed(java.awt.event.ActionEvent evt)
{
   try
   {
      byte [] IP = new byte [4];
      IP [0] = (byte)Integer.parseInt(ip1.getText());
      IP [1] = (byte)Integer.parseInt(ip2.getText());
      IP [2] = (byte)Integer.parseInt(ip3.getText());
      IP [3] = (byte)Integer.parseInt(ip4.getText());

      VLoad l = new VLoad();
      l.go("CONECTING...");
      txt.setText("CONECTING...");
      if(clientConection.conectable(IP))
      {
         this.dispose();
      }
      else
      {
         txt.setText("SERVER NON REACHABLE");
      }
   }
   catch (NumberFormatException e)
   {
      txt.setText("INVALID IP");
   }
}

// Variables declaration - do not modify
private javax.swing.JButton btt;
private javax.swing.JTextField ip1;
private javax.swing.JTextField ip2;
private javax.swing.JTextField ip3;
private javax.swing.JTextField ip4;
private javax.swing.JLabel jLabel1;
private javax.swing.JPanel jPanel1;
private javax.swing.JPanel jPanel2;
private javax.swing.JLabel txt;
```

```
    // End of variables declaration
}
```

Worker

```
package pecl3.screens;

/**
 *
 * @author mr.blissfulgrin
 */
public class VWorker extends VBackGroundPanel
{

    public VWorker()
    {
        super(60,60,"WORKER",50,133);
        initComponents();
        number.setText("");
    }

    public VWorker(int x, int y)
    {
        super(60,60,"WORKER",x,y);
        initComponents();
        number.setText("");
    }

    public void newWorker (int n)
    {
        this.number.setText(String.valueOf(n));
    }

    public int getNumber()
    {
        try
        {
            return Integer.parseInt(number.getText());
        }
        catch(NumberFormatException n)
        {
            return 0;
        }
    }

    /**
```

```
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents()
    {
        java.awt.GridBagConstraints gridBagConstraints;

        number = new javax.swing.JLabel();

        setLayout(new java.awt.GridBagLayout());

        number.setFont(new java.awt.Font("Silom", 1, 22)); // NOI18N
        number.setForeground(new java.awt.Color(51, 0, 51));
        number.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
        number.setText("NUMBER");
        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints.insets = new java.awt.Insets(0, 1, 14, 0);
        add(number, gridBagConstraints);
    }// </editor-fold>


    // Variables declaration - do not modify
    private javax.swing.JLabel number;
    // End of variables declaration
}
```

VWorkerBuffer

```
package pecl3.screens;

/**
 *
 * @author mr.blissfulgrin
 */
public class VWorkerBuffer extends VBackGroundPanel
{
   private final int spots [][];
   VWorker [] worker;

   public VWorkerBuffer(int x, int y)
   {
      super(300,500,"VOID",x,y);
      initComponents();
      VHouse h = new VHouse ();
      this.add(h);

      worker = new VWorker[8];
      spots = new int [6][2];
      int j;
      for (int i = 0; i < spots.length; i++)
      {
         j = (i < spots.length/2)? 0:1;
         spots[i][0] = 10 + 90*i - 270*j;
         spots[i][1] = 260 + 65 * j;
      }
   }

   public void newWorker (int number)
   {
      if (number-1 > spots.length)
      {
         System.out.println("ERROR!!! TOO MANY WORKERS, MAX: " + spots.length);
      }
      else
      {
         VWorker w = new VWorker(spots[number-1][0],spots[number-1][1]);
         w.newWorker(number);
         worker[number-1] = w;
         this.add(w);
```

```
    }
    this.repaint();
}

public void reset ()
{
    for (int x = 1; x <= worker.length; x++)
    {
        removeWorker(x);
    }
    this.removeAll();
    this.repaint();
    VHouse h = new VHouse ();
    this.add(h);
    this.repaint();
}

public void removeWorker (int number)
{
    if (worker[number-1] != null)
    {
        worker[number-1].setVisible(false);
        this.remove(worker[number-1]);
        this.repaint();
    }
}

/**
 * This method is called from within the constructor to initialize the form.
 * WARNING: Do NOT modify this code. The content of this method is always
 * regenerated by the Form Editor.
 */
@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">
private void initComponents()
{

    javax.swing.GroupLayout layout = new javax.swing.GroupLayout(this);
    this.setLayout(layout);
    layout.setHorizontalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGap(0, 400, Short.MAX_VALUE)
    );
    layout.setVerticalGroup(
```

```
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGap(0, 300, Short.MAX_VALUE)
        );
    }// </editor-fold>


    // Variables declaration - do not modify
    // End of variables declaration
}
```