



Tecnología de la Programación

Implementación de la Práctica 5

(Basado en la práctica de Samir Genaim)

Ana M. González de Miguel (ISIA, UCM)

Índice

1. Introducción
2. Diseño de Alto Nivel de la Arquitectura
3. Modificación del Simulador de Tráfico con MVC
4. Construcción de la GUI con Swing
5. La Clase Main
6. Editor de Eventos
7. Cola de Eventos
8. Zona de Informes
9. Tablas con Objetos del Simulador
10. Mapa de Carreteras

Índice

- 11. Barra de Estado (Opcional)
- 12. Interactuando con el Simulador
- 13. Barra de Menús
- 14. Barra de Herramientas
- 15. Plantillas para Eventos (Opcional)
- 16. Material de la Práctica

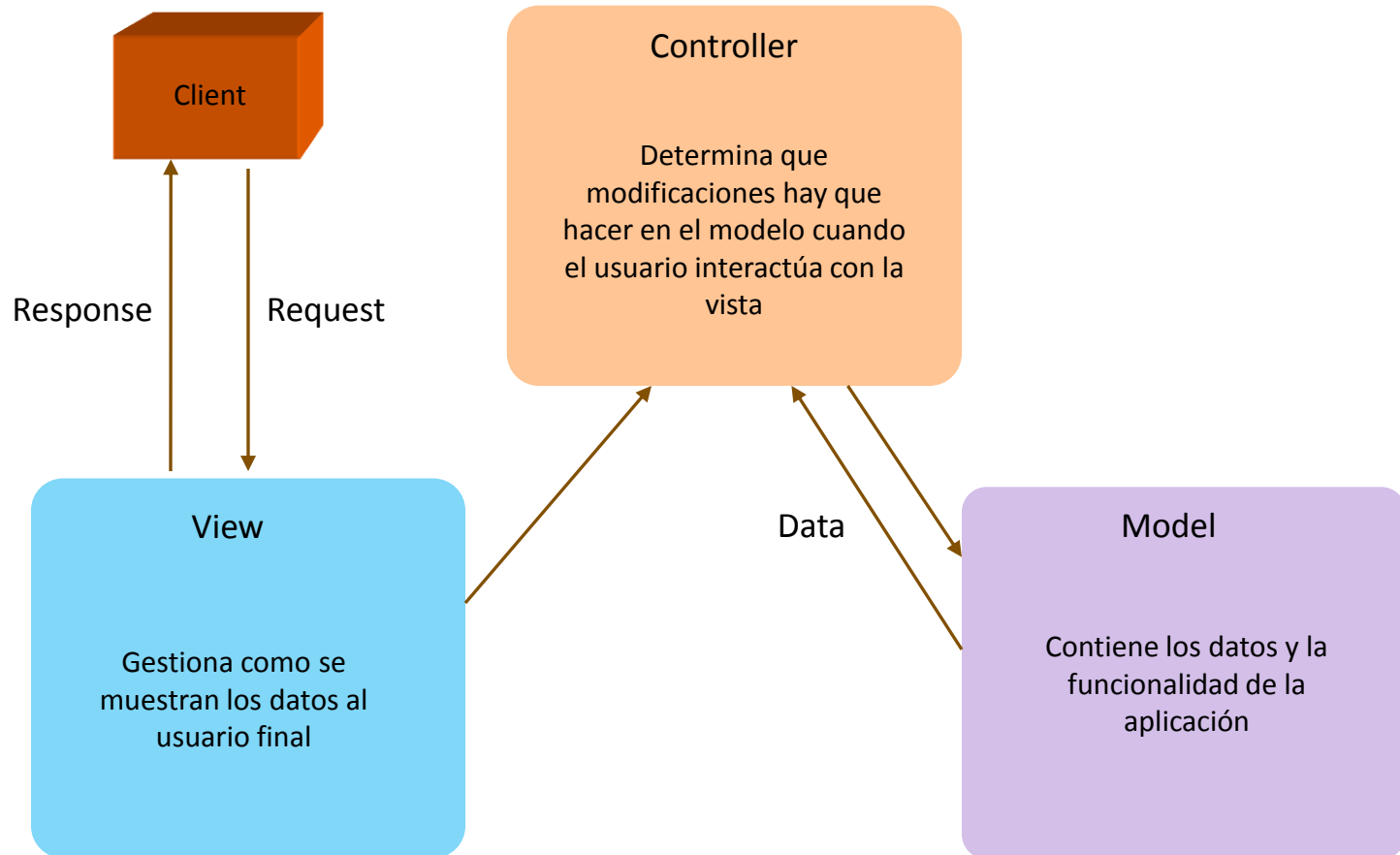
1. Introducción

- ✓ En esta práctica se construye una GUI para el simulador de tráfico utilizando arquitectura MVC.
- ✓ Nuevos objetivos: patrón MVC y GUI con Swing.
- ✓ La práctica se divide en dos partes:
 - En la primera parte se modifica el simulador de tráfico para utilizar el patrón MVC con observadores.
 - En la segunda parte se construye una GUI que permite al usuario interactuar con el simulador.
- ✓ Algunas secciones son opcionales. Estas se indican con la palabra “opcional”.

2. Diseño de Alto Nivel de la Arquitectura

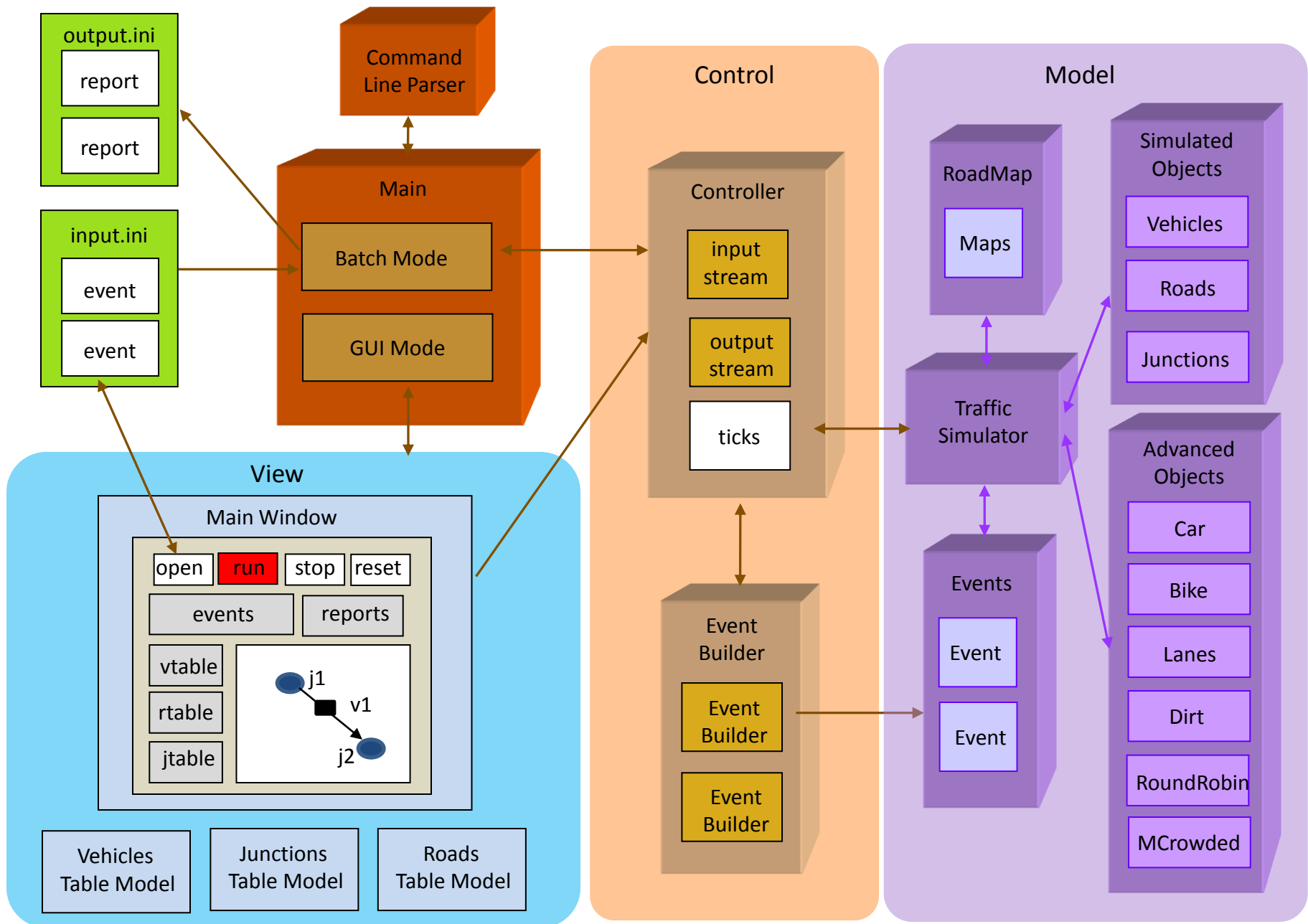
- ✓ MVC es un estilo de arquitectura software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes: *Model*, *View* y *Controller*, respectivamente.
 - *Model* es independiente de la representación visual de los datos y sólo es visible por *Controller*.
 - *View* muestra visualmente la información y usa *Controller*.
 - *Controller* recibe entradas que son traducidas a peticiones a *View* y *Model*.
- ✓ MVC es una arquitectura típica en el diseño de GUI interactivos.

Arquitectura MVC

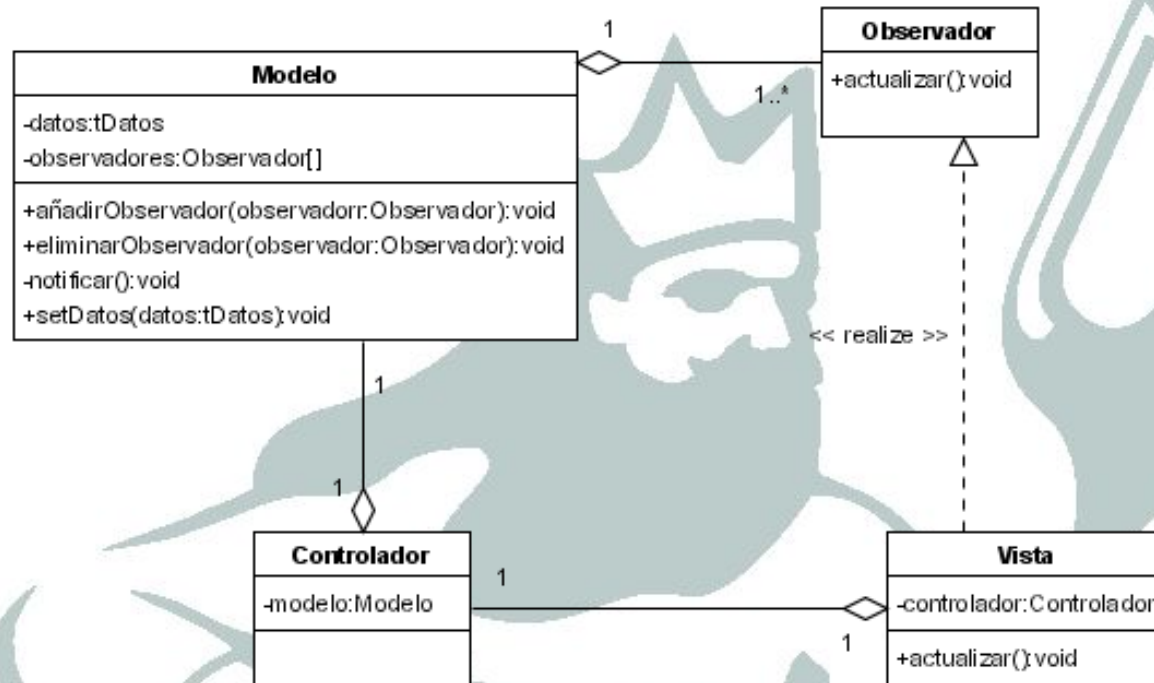


- ✓ En esta práctica utilizamos el patrón MVC con los componentes *Model* y *Control* de la práctica anterior e implementamos una interfaz gráfica (*View* o GUI) que permite al usuario interactuar con el simulador.
- ✓ Esta interfaz incluye un **menú** de opciones y una **barra de herramientas** (botones) necesarios para lanzar las operaciones del simulador. Además, incluye **7 zonas** para visualizar distintos datos del simulador: editor de eventos, cola de eventos, zona de informes, tabla de vehículos, tabla de carreteras, tabla de cruces y mapa de carreteras.
- ✓ En la práctica se incluye un nuevo método en *Main* (*GUIMode*) para ejecutar la interfaz gráfica con el thread de Swing. El método *BatchMode* de la práctica anterior se mantiene por lo que también es posible ejecutarlo.

Diseño de Alto Nivel de la Arquitectura



- ✓ Con esta arquitectura, la GUI no puede saber cuando hay cambios en el modelo del simulador. Por ejemplo, la tabla de los vehículos no se puede actualizar cada vez que avanzan en el mapa de carreteras. Para conseguir esto, es necesario que el patrón MVC incorpore la interfaz *Observer*.
- ✓ Según esta actualización de MVC, el modelo debe contener métodos *addObserver* para registrar a los observadores (la vista) y métodos *notify* para notificar a dichos observadores de posibles modificaciones llamando a sus métodos *update* de la interfaz *Observer*.
- ✓ Así, cada vez que cambia el modelo se notifica a los observadores (la vista) y éstos se actualizan.



3. Modificación del Simulador de Tráfico con MVC

- ✓ El simulador de tráfico debe ser modificado para utilizar el patrón de diseño MVC con observadores. Así los observadores (vista) podran registrarse en el modelo para ser notificados cuando haya cambios en el simulador (tambien llamados *eventos*, aunque no tengan nada que ver con los de la práctica anterior).
- ✓ El simulador debe poder notificar los siguientes eventos:
 - Registered. Cuando un observador se registre recibirá esta notificación.
 - Reset. Notificado cuando el simulador se reinicie.
 - New_event. Notificado cuando se añadan eventos al simulador.
 - Advanced. Notificado cuando se avance el tiempo de simulación.
 - Error. Cuando se produzca un error durante la simulación (con descripción del error).

- ✓ Añade una interfaz interna *TrafficSimulatorObserver* en la clase *TrafficSimulator* con los siguientes métodos *update* que permitirán a la vista actualizar sus datos:

```
public void registered(int time, RoadMap map, List<Event> events);  
public void reset(int time, RoadMap map, List<Event> events);  
public void eventAdded(int time, RoadMap map, List<Event> events);  
public void advanced(int time, RoadMap map, List<Event> events);  
public void simulatorError(int time, RoadMap map, List<Event> events,  
    SimulatorError e);
```

- ✓ Añade también el siguiente atributo en *TrafficSimulator*

```
private List<TrafficSimulatorObserver> observers;
```

- ✓ Implementa los métodos *notify* en *TrafficSimulator*.

```
private void notifyRegistered(TrafficSimulatorObserver o)  
private void notifyReset()  
private void notifyEventAdded()  
private void notifyAdvanced()  
private void notifyError(SimulatorError e)
```

- ✓ Implementa métodos para registrar y eliminar observadores en *TrafficSimulator*

```
public void addObserver(TrafficSimulatorObserver o)
public void removeObserver(TrafficSimulatorObserver o)
```

- ✓ Modifica el código de *TrafficSimulator* llamando a los métodos *notify* donde sea necesario.
- ✓ Finalmente, recuerda que al desarrollar la vista (observador) debes implementar los métodos *update* de la interfaz *TrafficSimulatorObserver*. Estos métodos son llamados dentro de los métodos *notify* del modelo (simulador).
- ✓ Todos los eventos notificados por el modelo deben incluir el estado completo de la simulación en el momento de lanzarse y ser notificados a todos los observadores registrados a excepción de *registered*. Este último evento se notifica cuando se ha registrado un observador.

4. Construcción de la GUI con Swing

- ✓ En esta segunda parte se construye la GUI para el simulador con Swing. Las clases de esta interfaz se incluyen en:

```
package es.ucm.fdi.view
```

- ✓ Esta interfaz contiene los siguientes componentes:
 - Barra de menú.
 - Barra de herramientas.
 - Editor de eventos.
 - Cola de eventos.
 - Zona de informes.
 - Tablas de vehiculos, carreteras y cruces.
 - Mapa de carreteras (grafo).
 - Barra de estado.

Traffic Simulator

File Simulator Reports

Steps: 1 Time: 1

Events: ex1.ini

time = 2
vehicles = v1
duration = 3

[new_vehicle]
time = 4
id = v5
itinerary = j1,j2,j3
max_speed = 20

[make_vehicle_faulty]
time = 10
vehicles = v1,v5
duration = 3

Events Editor

Events Queue

| # | Time | Type |
|---|------|------------------------|
| 0 | 2 | Break Vehicles [v1] |
| 1 | 4 | New Vehicle v5 |
| 2 | 10 | Break Vehicles [v1,v5] |

Events Queue

Reports

id = v2
time = 1
type = bike
speed = 11
kilometrage = 11
faulty = 0
location = (r3,11)

[vehicle_report]
id = v3
time = 1
type = car
speed = 11
kilometrage = 11
faulty = 0
location = (r3,11)

Reports Area

Vehicles

| ID | Road | Location | Speed | Km | Faulty Units | Itinerary |
|----|------|----------|-------|----|--------------|--------------|
| v1 | r1 | 20 | 20 | 20 | 0 | [j1, j2, j3] |
| v2 | r3 | 11 | 11 | 11 | 0 | [j4, j2, j5] |
| v3 | r3 | 11 | 11 | 11 | 0 | [j4, j2, j5] |

Vehicles Table

Roads

| ID | Source | Target | Length | Max Speed | Vehicles |
|----|--------|--------|--------|-----------|----------|
| r1 | j1 | j2 | 100 | 20 | [v1] |
| r2 | j2 | j3 | 100 | 20 | [] |
| r3 | j4 | j2 | 100 | 20 | [v2, v3] |
| r4 | j2 | j5 | 100 | 20 | [] |

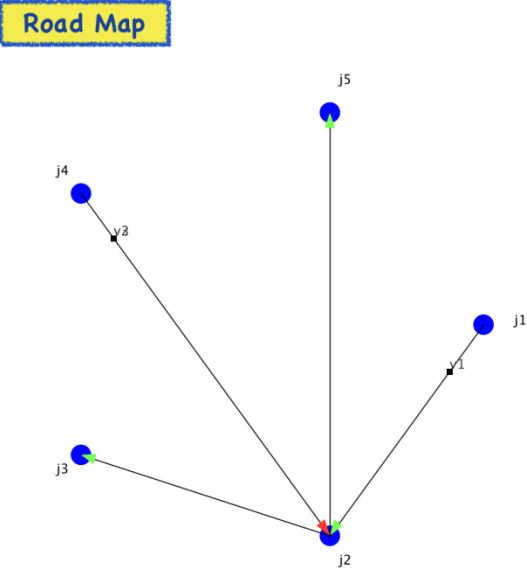
Roads Table

Junctions

| ID | Green | Red |
|----|--------------------|----------------|
| j1 | [] | [] |
| j2 | [(r1,green,[],)] | [(r3,red,[],)] |
| j3 | [(r2,green:1,[],)] | [] |
| j4 | [] | [] |
| j5 | [(r4,green:3,[],)] | [] |

Junctions Table

Road Map



Events have been loaded to the simulator!

- ✓ Implementa una clase *MainWindow* capaz de contener todos estos componentes

```
public class MainWindow extends JFrame implements  
TrafficSimulatorObserver
```

- ✓ Incluye al menos los siguientes atributos en la clase:

```
private Controller ctrl; // la vista usa el controlador  
private RoadMap map; // para los métodos update de Observer  
private int time; // para los métodos update de Observer  
private List<Event> events; // para los métodos update de Observer  
private OutputStream reportsOutputStream;  
  
private JPanel mainPanel;  
private JPanel contentPanel_1; // tantos como areas en la ventana  
private JMenu fileMenu;  
private JMenu simulatorMenu;  
private JMenu reportsMenu;  
private JToolBar toolBar;  
private JFileChooser fc;  
private File currentFile;
```



```
private JButton loadButton;  
private JButton saveButton;  
private JButton clearEventsButton;  
private JButton checkInEventsButton;  
private JButton runButton;  
private JButton stopButton;  
private JButton resetButton;  
private JSpinner stepsSpinner;  
private JTextField timeViewer;  
private JButton genReportsButton;  
private JButton clearReportsButton;  
private JButton saveReportsButton;  
private JButton quitButton;  
private JTextArea eventsEditor; // editor de eventos  
private JTable eventsTable; // cola de eventos  
private JTextArea reportsArea; // zona de informes  
private JTable vehiclesTable; // tabla de vehiculos  
private JTable roadsTable; // tabla de carreteras  
private JTable junctionsTable; // tabla de cruces  
private ReportDialog reportDialog; // opcional
```

- ✓ La constructora de esta clase debe contener al menos las siguientes operaciones:

```
public MainWindow(TrafficSimulator model, String inFileName,
Controller ctrl) {
    super("Traffic Simulator");
    this.ctrl = ctrl;
    currentFile = inFileName != null ? new File(inFileName) : null;
    reportsOutputStream = new JTextAreaOutputStream(reportsArea,null);
    ctrl.setOutputStream(reportsOutputStream); // ver sección 8
    initGUI();
    model.addObserver(this);
}
```

- ✓ El método *initGUI* debe dividir la ventana principal en paneles apropiados, añadir todos los componentes característicos de la interfaz y configurar propiedades generales de la ventana principal. A continuación se muestran parte de las operaciones de este método:

```
private void initGUI() {  
    mainPanel = new JPanel(new BorderLayout());  
    this.setContentPane(mainPanel);  
    contentPanel_1 = new JPanel();  
    contentPanel_1.setLayout(new BoxLayout(contentPanel_1,  
BoxLayout.Y_AXIS));  
    // Divide la ventana en mas paneles y configura layout  
    fc = new JFileChooser();  
    addMenuBar(); // barra de menus  
    addToolBar(); // barra de herramientas  
    addEventsEditor(); // editor de eventos  
    addEventsView(); // cola de eventos  
    addReportsArea(); // zona de informes  
    addVehiclesTable(); // tabla de vehiculos  
    addRoadsTable(); // tabla de carreteras  
    addJunctionsTable(); // tabla de cruces  
    addMap(); // mapa de carreteras  
    addStatusBar(); // barra de estado  
    // Añade configuraciones de la ventana principal  
}
```

5. La Clase Main

- ✓ Deberás modificar la clase principal de la cuarta práctica para que soporte tanto el uso de la interfaz gráfica (esta práctica) como el modo de texto sin interacción ya implementado.

```
private static void startBatchMode() throws ...  
private static void startGUIMode() throws ...
```

- ✓ Para ello, utiliza un argumento adicional *-m* que pueda tomar valores *batch* o *gui* y que lance la aplicación en el modo correspondiente. Si este argumento no se especifica deberá tomar por defecto el valor *batch*.
- ✓ El resultado de ejecutar el programa con argumento *-h* (que muestra la ayuda) deberá ser similar al siguiente.

```
> java Main -h
usage: Main [-h] [-i <arg>] [-m <arg>] [-o <arg>] [-t <arg>]
-h,--help Print this message
-i,--input <arg> Events input file
-m,--mode <arg> 'batch' for batch mode and 'gui' for GUI mode
(default value is 'batch')
-o,--output <arg> Output file, where reports are written.
-t,--time <arg> Time units to execute the simulator's main loop
(default value is 10).
```

- ✓ En modo no-interactivo (*batch*), todo el comportamiento debe ser idéntico al de la práctica anterior. En el modo gráfico (*gui*), deberás tener en cuenta lo siguiente:
- *-i*: si se proporciona un fichero de eventos iniciales usando esta opción, debe cargarse inmediatamente en la zona de eventos nada más mostrarse la interfaz.
 - *-o* y *-t*: ambas deben ser ignoradas, aunque podrás, a tu elección, usar el valor de *-t* como valor inicial para el número de ciclos de simulación.

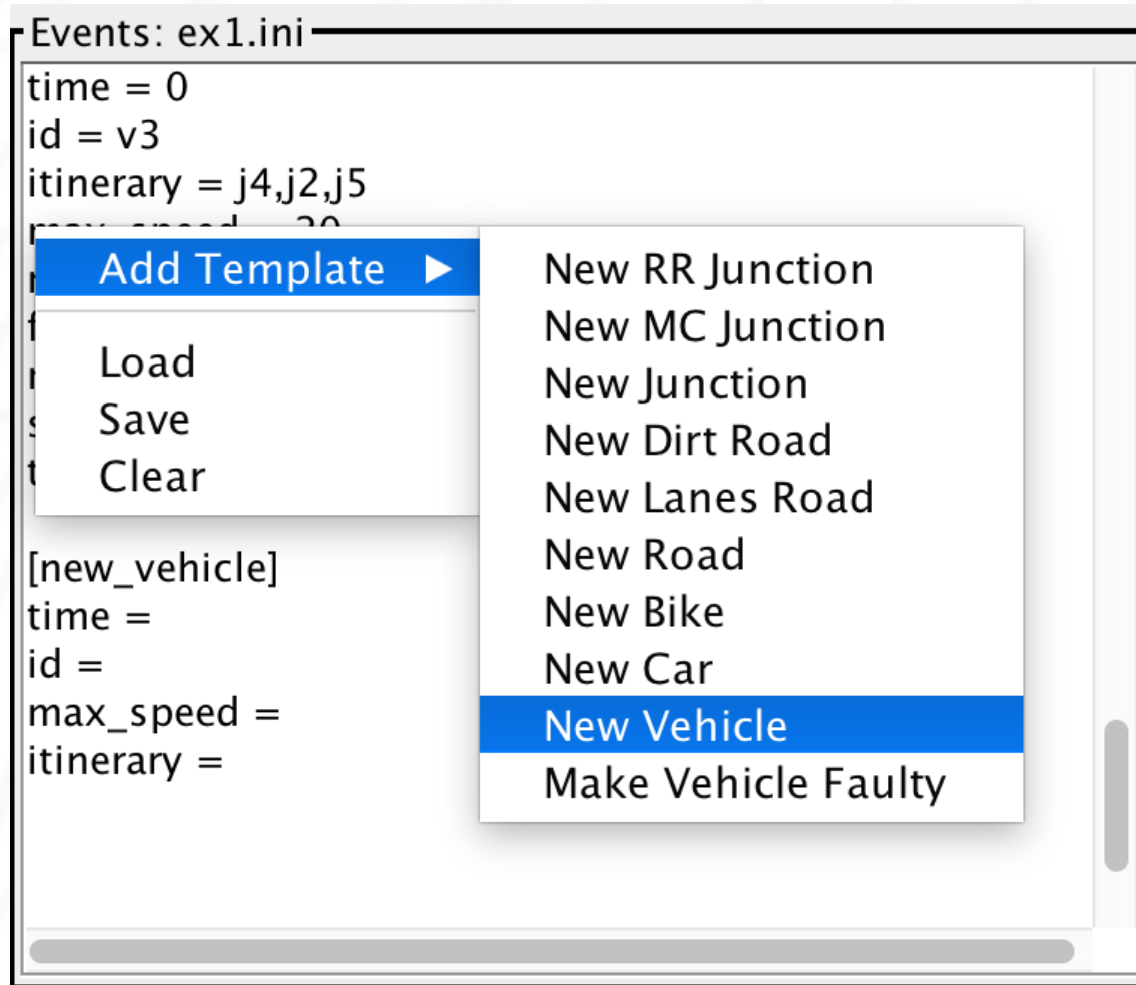
- ✓ El método *startGUIMode* debe crear objetos de *TrafficSimulator*, *OutputStream* y *Controller* de manera similar a *startBatchMode*. Para la ejecución de la interfaz con el thread de Swing utiliza la siguiente invocación dentro del método *startGUIMode*:

```
SwingUtilities.invokeLater(new Runnable() {  
    public void run() {  
        new MainWindow(sim, inFile, ctrl);  
    }  
});
```

- ✓ Prueba tu interfaz con esta clase *Main* cada vez que añadas un nuevo componente según se especifica en las siguientes transparencias de la presentación.

6. Editor de Eventos

- ✓ Este componente permite a los usuarios editar eventos. Contiene un editor de texto, basado por ejemplo en *JTextArea*, y debe suministrar la siguiente funcionalidad:
 - Cargar un fichero de texto en el editor.
 - Guardar su contenido en un fichero de texto.
 - Borrar el contenido dejando un texto vacío.
- ✓ Como parte del enunciado, recibirás un editor de texto Swing que podrás adaptar para su uso en la realización de este editor de eventos.
- ✓ Implementa el método *addEventsEditor()* para incorporar este componente a la interfaz.
- ✓ El editor de eventos incluye un menú desplegable como el que se muestra en la siguiente figura.




```
private void addEventsEditor() {  
    // Crea un JPanel para este componente  
    eventsEditor = new JTextArea(40,30);  
    // Suma eventsEditor al JPanel y añade JScrollPane  
    // Añade JPanel al panel correspondiente de la ventana principal  
    if (currentFile != null) {  
        eventsEditor.setText(readFile(currentFile));  
    }  
    refreshEventsAreaBorder();  
    // Crea el PopupMenu de eventos según la figura  
    // Crea y configura el JMenu de Add Template  
    // Crea y configura el JMenuItem de Load  
    // Crea y configura el JMenuItem de Save  
    // Crea y configura el JMenuItem de Clear  
    // Suma los componentes creados al PopupMenu  
    // Registra MouseListener en eventsEditor  
}
```

7. Cola de Eventos

- ✓ Este componente contendrá una lista de los eventos pendientes en la cola del simulador. Cada elemento representará un evento, mediante su tiempo de ejecución y una descripción textual.
- ✓ Puedes usar un *JTable* o un *JList* para implementar este componente. En esta presentación se utiliza *JTable*.
- ✓ Crea la clase *EventsTableModel* para el modelo de descripción de la tabla de eventos. Mira los ejemplos de *JTable* vistos en clase para usar estos modelos.
- ✓ Implementa el método *addEventsView()* con las operaciones que se muestran en la siguiente transparencia.

```
private void addEventsView() {  
    // Crea un JPanel para este componente  
    // Pon el borde en el JPanel  
    eventsTableModel = new EventsTableModel();  
    eventsTable = new JTable(eventsTableModel);  
    // Suma eventsTable al JPanel y añade JScrollPane  
    // Suma el JPanel al panel correspondiente de la ventana principal  
}
```

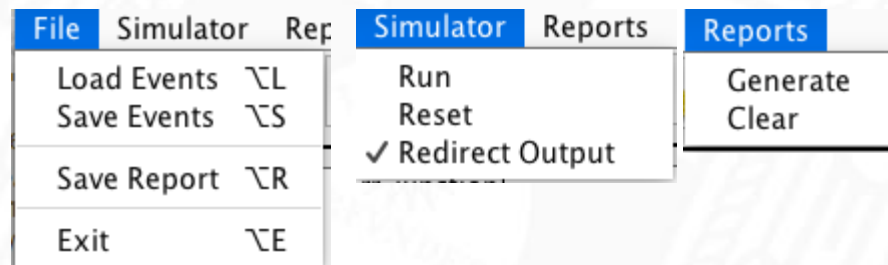
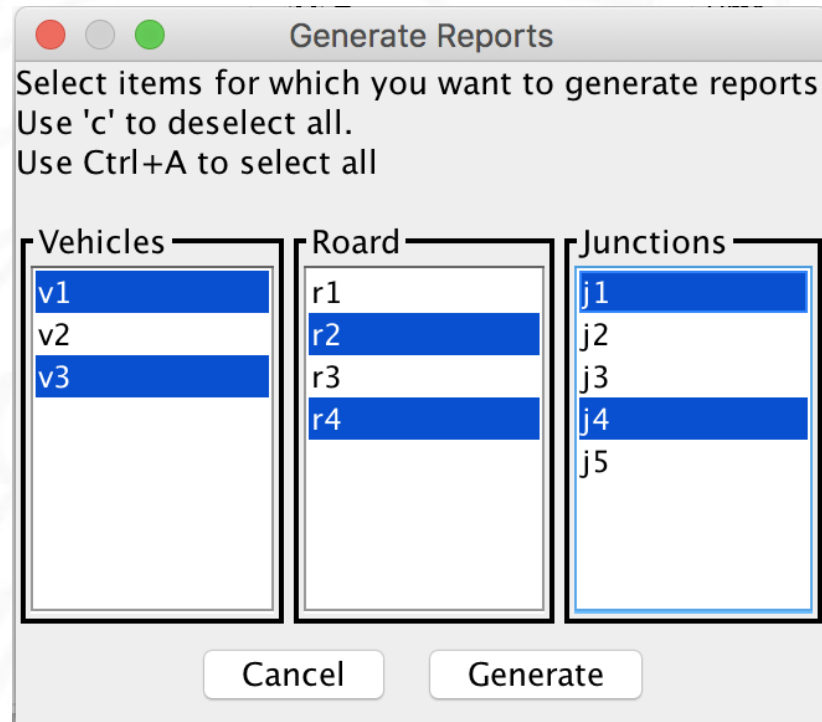
8. Zona de Informes

- ✓ Este componente consta, esencialmente, de un *JTextArea* donde se escriben los informes. Estos informes se pueden generar de las siguientes formas:
 - Todos los informes. La interfaz debe permitir generar todos los informes del estado actual del simulador, que se mostrarán a continuación en esta zona. Por ejemplo, puedes hacer que se muestren todos tras pulsar sobre un botón.
 - Informes concretos (Opcional). La interfaz permitirá seleccionar vehículos, carreteras e intersecciones y generar informes de los así seleccionados. La selección se podrá hacer mediante (1) un diálogo; (2) una selección en las correspondientes tablas, usando *getSelectionModel()* sobre la *JTable* correspondiente; ó (3) cualquier otro método que tenga sentido y funcione. Junto con este enunciado, recibirás un ejemplo de cómo crear un diálogo. Si implementas esta opción, no es necesario implementar la anterior.

- Generados por el simulador (Opcional). Redirige salida del simulador (es decir, lo que escribe en su *OutputStream*) a la zona de texto. Para ello, crea una clase *JTextAreaOutputStream* con un método *write* que escriba en la *JTextArea* correspondiente. Fíjate en la utilización de esta clase en la constructora de *MainWindow* (sección 4). El usuario también deberá poder deshabilitar esta redirección (según se ve en el menú de la figura siguiente) – esto se puede hacer poniendo el *OutputStream* de salida del simulador a null. Si no vas a implementar esta parte, recomendamos poner la salida del simulador a null desde un principio.
- ✓ La interfaz también debe permitir:
 - Mostrar el contenido actual de los informes a un fichero de texto (Opcional).
 - Borrar el contenido de la zona de informes.

- ✓ Implementa el método *addReportsArea()* para añadir este componente a la interfaz:

```
private void addReportsArea() {  
    // Crea un JPanel para este componente  
    // Pon el borde en el JPanel  
    reportsArea = new JTextArea(40, 30);  
    reportsArea.setEditable(false);  
    // Suma reportsArea al JPanel y añade JScrollPane  
    // Suma el JPanel al panel correspondiente de la ventana principal  
}
```

9. Tablas con Objetos del Simulador

- ✓ La interfaz debe contener una tabla por categoría de objeto simulado, es decir, para vehículos, carreteras y cruces. Estas tablas se deben mostrar usando *JTable* con sus correspondientes modelos (*TableModel*).
- ✓ La información a incluir es similar a la que se muestra en los informes. Tienes libertad para mostrar información adicional de cualquier forma que sea inteligible.
- ✓ Crea clases de modelos similares a *EventsTableModel*..
- ✓ Para mostrar estos componentes en la interfaz implementa los siguientes métodos:

```
private void addVehiclesTable()  
private void addRoadsTable()  
private void addJunctionsTable()
```

- ✓ A continuación se muestra uno de ellos.


```
private void addVehivlesTable() {  
    // Crea un JPanel para este componente  
    // Pon el borde en el JPanel  
    vehiclesTableModel = new VehiclesTableModel();  
    vehiclesTable = new JTable(vehiclesTableModel);  
    // Suma vehiclesTable al JPanel y añade JScrollPane  
    // Suma el JPanel al panel correspondiente de la ventana principal  
}
```

10. Mapa de Carreteras

- ✓ Este es un componente Swing que muestra el mapa de carreteras. Proporcionaremos un componente similar que muestra un grafo (con puntos en las aristas, que podras usar para mostrar coches). Todo lo que necesitas es reemplazar el *Graph* por un *RoadMap*.
- ✓ Implementa el método *addMap()* para incorporar este componente a la interfaz.

11. Barra de Estado (Opcional)

- ✓ La interfaz podrá incluir una barra de estado o equivalente (en el sentido de “zona de información contextual”), donde se mostrarán mensajes informativos mientras se interacciona con la interfaz.
- ✓ En la primera figura de la presentación, esta barra está en la parte mas inferior de la ventana principal.
- ✓ Implementa el método *addStatusBar()* para añadir este componente a la interfaz.

```
private void addStatusBar() {  
    // Crea un JPanel para este componente  
    // Pon el borde en el JPanel  
    statusBarText = new JLabel("Welcome to the simulator!");  
    // Suma statusBarText al JPanel  
    // Suma el JPanel a mainPanel  
}
```

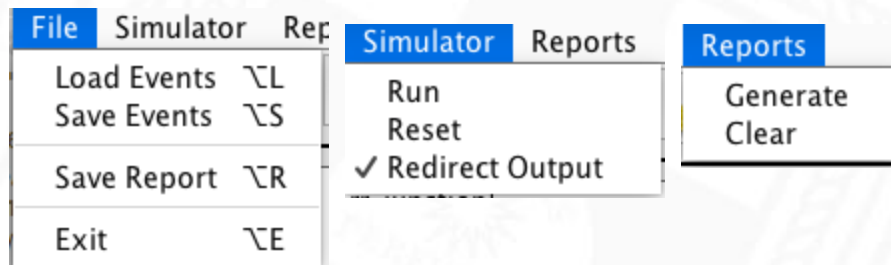
12. Interactuando con el Simulador

- ✓ La interfaz deberá proporcionar las siguientes opciones:
 - Insertar eventos. Añadirá los eventos que contenga el editor de eventos al simulador. Ten en cuenta que puedes crear una *InputStream* para el método *loadEvents* de *Controller* a partir de una cadena *s* mediante *new ByteArrayInputStream(s.getBytes())*.

```
ctrl.loadEvents(new  
    ByteArrayInputStream(eventsEditor.getText().getBytes()));
```
 - Lanzar el simulador durante tantos ciclos de tiempo como haya especificado el usuario. Por ejemplo, puedes usar un *JSpinner* similar al de la primera figura de la presentación.
 - Reiniciar el simulador.

13. Barra de Menus

- ✓ Aunque las barras de menú y herramientas son opcionales, recomendamos encarecidamente usarlas. Si no implementas al menos una de las dos, deberas incluir tu propio panel de control, en el que deberás incluir todas las operaciones/información que proporcionarían estas barras. Notese que tanto la barra de herramientas como la barra de menú permiten hacer esencialmente lo mismo.
- ✓ La barra de menú incluye las siguientes interacciones:



```
public void addMenuBar() {  
    JMenuBar menuBar = new JMenuBar();  
    fileMenu = new JMenu("File");  
    // Añade fileMenu a menuBar  
    // Crea y configura JMenuItem "Load Events"  
    // Crea y configura JMenuItem "Save Events"  
    // Crea y configura JMenuItem "Save Report"  
    // Crea y configura JMenuItem "Exit"  
    // Suma los JMenuItem a fileMenu con Separador  
    simulatorMenu = new JMenu("Simulator");  
    // Añade simulatorMenu a menuBar  
    // Crea y configura JMenuItem "Run"  
    // Crea y configura JMenuItem "Reset"  
    // Crea y configura JMenuItem "Redirect Output"  
    // Suma los JMenuItem a simulatorMenu  
    reportsMenu = new JMenu("Reports");  
    // Añade reportsMenu a menuBar  
    // Crea y configura JMenuItem "Generate"  
    // Crea y configura JMenuItem "Clear"
```



```
// Suma los JMenuItem a reportsMenu  
this.setJMenuBar(menuBar);  
}
```

- ✓ Al configurar los *JMenuItem* con *addActionListener* será necesario implementar métodos *actionPerformed()* que llamen a métodos como, por ejemplo, el siguiente:

```
private void clearEventsArea() {  
    eventsEditor.setText("");  
    setStatusBarMsg("Events text-area has been cleared!");  
    currentFile = null;  
    refreshEventsAreaBorder();  
}
```

14. Barra de Herramientas

- ✓ Esta barra incluye los siguientes botones:
 - Cargar un fichero de eventos.
 - Salvar un fichero de eventos.
 - Limpiar la zona de eventos.
 - Insertar eventos en el simulador.
 - Lanzar el simulador.
 - Reiniciar el simulador.
 - Establecer cuantos ciclos se deben simular al lanzarlo.
 - Una etiqueta que indique el ciclo actual de simulación.
 - Generar informes.
 - Limpiar la zona de informes.
 - Limpiar los informes.
 - Salir del simulador.


```
private void addToolBar() {  
    toolBar = new JToolBar();  
    mainPanel.add(toolBar, BorderLayout.PAGE_START);  
    loadButton = new JButton();  
    // Configurar loadButton  
    // Sumar loadButton a toolBar  
    saveButton = new JButton();  
    // Configurar saveButton  
    // Sumar saveButton toolBar  
    clearEventsButton = new JButton();  
    // Configurar clearEventsButton  
    // Sumar clearEventsButton a toolBar  
    checkInEventsButton = new JButton();  
    // Configurar checkInEventsButton  
    // Sumar checkInEventsButton a toolBar  
    runButton = new JButton();  
    // Configurar runButton  
    // Sumar runButton a toolBar
```

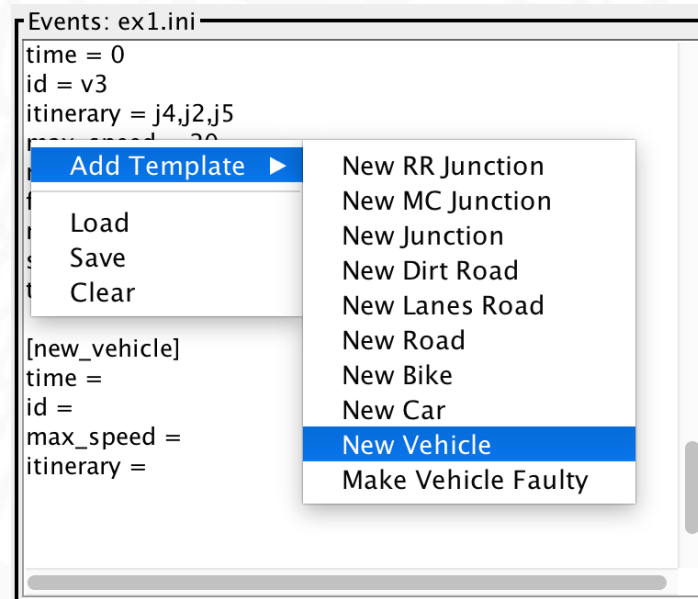
```
stopButton = new JButton();  
// Configurar stopButton  
// Sumar stopButton a toolBar  
resetButton = new JButton();  
// Configurar resetButton  
// Sumar resetButton a toolBar  
toolBar.add(new JLabel(" Steps: "));  
stepsSpinner = new JSpinner(new SpinnerNumberModel(5, 1, 1000,  
1));  
// Configurar stepsSpinner  
// Sumar stepsSpinner a toolBar  
toolBar.add(new JLabel(" Time: "));  
timeViewer = new JTextField("0", 5);  
// Configurar timeViewer  
// Sumar timeViewer a toolBar  
toolBar.addSeparator();  
genReportsButton = new JButton();  
// Configurar genReportsButton  
// Sumar genReportsButton a toolBar
```

```
clearReportsButton = new JButton();  
// Configurar clearReportsButton  
// Sumar clearReportsButton a toolBar  
saveReportsButton = new JButton();  
// Configurar saveReportsButton  
// Sumar saveReportsButton a toolBar  
toolBar.addSeparator();  
quitButton = new JButton();  
// Configurar quitButton  
// Sumar quitButton a toolBar  
}
```

- ✓ Los métodos invocados en los *actionPerformed()* de los botones deben coincidir con aquellos implementados para la barra de menú.

15. Plantillas para Eventos (Opcional)

- ✓ Para facilitar la creación de eventos en el editor, añade un menú contextual (activado vía click derecho en el editor) que permita elegir entre varias plantillas de evento. Si se selecciona una plantilla, deberás insertar el texto correspondiente en el punto de inserción actual del editor.



- ✓ Te proporcionaremos un ejemplo de menú contextual similar al que te pedimos implementar.

16. Material de la Práctica

- ✓ Para la realización de la práctica tenéis el siguiente material:
 - Especificaciones de la Interfaz (Enunciado y Transparencias)
 - Diseño de Alto Nivel de la Arquitectura
 - Código:
 - El editor de texto Swing (extra/texteditor)
 - La ventana de diálogo (extra/dialog)
 - El mapa de carreteras (extra/graphlayout)
 - El menú contextual (extra/popupmenu)
 - Modelo de tabla (ejemplo EjemploJTable4 visto en clase)