

Práctica 4: Simulador de tráfico (Parte I)

Puri Arenas Sánchez
Facultad de Informática (UCM)

Clase Main

Realiza el parseo de los argumentos e inicia la aplicación con entrada/salida estándar.

```
public static void main(String[] args) throws IOException {  
    // example command lines:  
    // -i resources/examples/events/basic/ex1.ini  
    // -i resources/examples/events/advanced/ex1.ini  
    // --help  
    Main.ParseaArgumentos(args);  
    Main.iniciaModoEstandar();  
}
```

3

Parte I: Clases

- **Main:** Inicia la solución y realiza el parseo de los argumentos.
- **Controlador:** Carga los eventos de entrada de un fichero y arranca el simulador.
- **SimuladorTrafico:** Lleva a cabo la simulación durante el número de pasos especificados.
- **Evento:**
 - EventoNuevaCarretera:** Para añadir una carretera a la simulación.
 - EventoNuevoVehiculo:** Para añadir un vehículo a la simulación.
 - EventoNuevoCruce:** Para añadir un cruce a la simulación.
- **ObjetoSimulacion:**
 - Vehiculo:** Representa un vehículo.
 - Carretera:** Representa una carretera.
 - Cruce:** Representa un cruce.
- **Mapa de Carreteras:** Almacena el estado de la simulación. Contiene la lista de carreteras, de cruces y de vehículos.

2

Clase Main

Realiza el parseo de los argumentos e inicia la aplicación con entrada/salida estándar.

```
public static void main(String[] args) throws IOException {  
    // example command lines:  
    // -i resources/examples/events/basic/ex1.ini  
    // -i resources/examples/events/advanced/ex1.ini  
    // --help  
    Main.ParseaArgumentos(args);  
    Main.iniciaModoEstandar();  
}  
  
private static void iniciaModoEstandar() throws IOException {  
    InputStream is = new FileInputStream(new File(Main.ficheroEntrada));  
    OutputStream os = Main.ficheroSalida == null ? System.out :  
        new FileOutputStream(new File(Main.ficheroSalida));  
    SimuladorTrafico sim = new SimuladorTrafico();  
    Controlador ctrl = new Controlador(sim, Main.limiteTiempo, is, os);  
    ctrl.ejecuta();  
    is.close();  
    System.out.println("Done!");  
    os.close();  
}
```

4

Clase Controlador

Carga los eventos del fichero e entrada, e invoca al método ejecuta del simulador.

```
public class Controlador {  
  
    private SimuladorTrafico simulador;  
    private OutputStream ficheroSalida;  
    private InputStream ficheroEntrada;  
    private int pasosSimulacion;  
  
    ...  
  
    public void ejecuta() {  
        this.cargaEventos(this.ficheroEntrada);  
        this.simulador.ejecuta(pasosSimulacion, this.ficheroSalida);  
    }  
    private void cargaEventos(InputStream inStream) {...}  
}
```

5

Clase Controlador: Cargar Eventos

```
private void cargaEventos(InputStream inStream) {  
    Ini ini;  
    try {  
        // lee el fichero y carga su atributo iniSections  
        ini = new Ini(inStream);  
    }  
    catch (IOException e) {  
        throw new ErrorDeSimulacion("Error en la lectura de eventos: " + e);  
    }  
    // recorremos todas los elementos de iniSections para generar el evento  
    // correspondiente  
    for (IniSection sec : ini.getSections()) {  
        // parseamos la sección para ver a que evento corresponde  
        Evento e = ParserEventos.parseaEvento(sec);  
        if (e != null) this.simulador.insertaEvento(e);  
        else  
            throw new ErrorDeSimulacion("Evento desconocido: " + sec.getTag());  
    }  
}
```

7

Representación de los Eventos

■ Para leer los eventos del fichero se utiliza la clase Ini.

■ Esta clase contiene el atributo `List<IniSection> iniSections`, para almacenar cada una de las entradas del fichero. Su método `List<IniSection> getSections()`, devuelve dicho atributo.

■ La clase `IniSection` tiene como atributos más destacados:

* `String tag`: Identificador de la sección ("new_junction", "new_road", "new_vehicle", "make_vehicle_faulty").

* `Map<String,String>` que contiene cada entrada de la sección. Por ejemplo ("time","0"), ("max_speed","10"), ("id","j3"), etc.

* El método `String getTag()`, devuelve el valor del campo tag.

* El método `String getValue(String key)`, devuelve el valor asociado a la clave key. Por ejemplo `getValue("time")` devolvería "0", `getValue("max_speed")` devolvería "10", etc.

* El método `void setValue(String id, valor)`, coloca en el campo id el valor value.

6

Clase ParserEventos

```
public class ParserEventos {  
  
    private static ConstructorEventos[] eventos = {  
        new ConstructorEventoNuevoCruce(),  
        new ConstructorEventoNuevaCarretera(),  
        new ConstructorEventoNuevoVehiculo(),  
        new ConstructorEventoAveriaCoche()  
    };  
  
    // bucle de prueba y error  
    public static Evento parseaEvento(IniSection sec) {  
        int i = 0;  
        boolean seguir = true;  
        Evento e = null;  
        while (i < ParserEventos.eventos.length && seguir) {  
            // ConstructorEventos contiene el método parse(sec)  
            e = ParserEventos.eventos[i].parser(sec);  
            if (e!=null) seguir = false;  
            else i++;  
        }  
        return e;  
    }  
}
```

8

Clase ConstructorEventos

- De esta clase heredan ConstructorEventoAveriaCoche, ConstructorEventoNuevaCarretera, ConstructorEventoNuevoCruce, ConstructorEventoNuevoVehiculo

```
public abstract class ConstructorEventos {  
    // cada clase dará los valores correspondientes a estos atributos  
    // en la constructora  
    protected String etiqueta; // etiqueta de la entrada ("new_road", etc..)  
    protected String[] claves; // campos de la entrada ("time", "vehicles", etc.)  
    ...  
    ConstructorEventos() {  
        this.etiqueta = null;  
        this.claves = null;  
    }  
    ...  
    public abstract Evento parser(IniSection section);  
}
```

9

Clase ConstructorEventoNuevoCruce

```
public class ConstructorEventoNuevoCruce extends ConstructorEventos {  
    public ConstructorEventoNuevoCruce() {  
        this.etiqueta = "new_junction";  
        this.claves = new String[] { "time", "id" };  
        this.valoresPorDefecto = new String[] { "", "", };  
    }  
    @Override  
    public Evento parser(IniSection section) {  
        if (!section.getTag().equals(this.etiqueta) ||  
            section.getValue("type") != null) return null;  
        else  
            return new EventoNuevoCruce(  
                // extrae el valor del campo "time" en la sección  
                // 0 es el valor por defecto en caso de no especificar el tiempo  
                ConstructorEventos.parseaIntNoNegativo(section, "time", 0),  
                // extrae el valor del campo "id" de la sección  
                ConstructorEventos.identificadorValido(section, "id"));  
    }  
    @Override  
    public String toString() { return "New Junction"; }  
}
```

```
[new_junction]  
time = 0  
id = j4
```

10

ConstructorEventos: Parser

```
protected static String identificadorValido(IniSection seccion, String clave) {  
    String s = seccion.getValue(clave);  
    if (!esIdentificadorValido(s))  
        throw new IllegalArgumentException("El valor " + s + " para " + clave +  
                                           " no es un ID valido");  
    else return s;  
}  
  
// identificadores válidos  
// sólo pueden contener letras, números y subrayados  
  
private static boolean esIdentificadorValido(String id) {  
    return id != null && id.matches("[a-z0-9_]+");  
}
```

11

ConstructorEventos: Parser

```
protected static int parseaInt(IniSection seccion, String clave) {  
    String v = seccion.getValue(clave);  
    if (v == null)  
        throw new IllegalArgumentException("Valor inexistente para la clave: " +  
                                           clave);  
    else return Integer.parseInt(seccion.getValue(clave));  
}  
  
protected static int parseaInt(IniSection seccion,  
    String clave,  
    int valorPorDefecto) {  
    String v = seccion.getValue(clave);  
    return (v != null) ? Integer.parseInt(seccion.getValue(clave)) :  
        valorPorDefecto;  
}  
  
protected static int parseaIntNoNegativo(IniSection seccion,  
    String clave,  
    int valorPorDefecto) {  
    int i = ConstructorEventos.parseaInt(seccion, clave, valorPorDefecto);  
    if (i < 0)  
        throw new IllegalArgumentException("El valor " + i + " para " + clave +  
                                           " no es un ID valido");  
    else return i;  
}
```

12

ConstructorEventos: Parser

```
protected static int parseaInt(IniSection seccion, String clave) {
    String v = seccion.getValue(clave);
    if (v == null)
        throw new IllegalArgumentException("Valor inexistente para la clave: " +
                                           clave);
    else return Integer.parseInt(seccion.getValue(clave));
}

protected static int parseaInt(IniSection seccion,
                               String clave,
                               int valorPorDefecto) {
    String v = seccion.getValue(clave);
    return (v != null) ? Integer.parseInt(seccion.getValue(clave)) :
        valorPorDefecto;
}

protected static int parseaIntNoNegativo(IniSection seccion,
                                          String clave,
                                          int valorPorDefecto) {
    int i = ConstructorEventos.parseaInt(seccion, clave, valorPorDefecto);
    if (i < 0)
        throw new IllegalArgumentException("El valor " + i + " para " + clave +
                                           " no es un ID valido");
    else return i;
}
```

El resto de constructores de eventos se programan de forma similar

13

SimuladorTrafico

```
private MapaCarreteras mapa;
private List<Evento> eventos;
private int contadorTiempo;
```

```
public SimuladorTrafico() {
    this.mapa = new MapaCarreteras();
    this.contadorTiempo = 0;
    Comparator<Evento> cmp = new Comparator<Evento>() {...};
    this.eventos = new SortedArrayList<>(); // estructura ordenada por "tiempo"
}
```

15

SimuladorTrafico

- Es el encargado de realizar la simulación durante un número determinado de pasos, y en cada paso ir mostrando el estado de la simulación, bien en un fichero de texto, o bien en la salida estándar.
- Para realizar la simulación ejecuta en cada paso su lista de eventos (**List<Evento> eventos**).
- Para contabilizar los pasos de la simulación utiliza un contador de tiempo (**int contadorTiempo**).
- Necesita almacenar el estado de la simulación en alguna estructura. Para ello utilizamos un **mapa de carreteras**, que guarda el estado de cada uno de los objetos de la simulación.
- Su constructora por defecto, inicializa por defecto sus atributos. Para el atributo **eventos** necesitas utilizar una estructura ordenada, ya que los eventos van ordenados por su tiempo de ejecución.

14

SimuladorTrafico

```
private MapaCarreteras mapa;
private List<Evento> eventos;
private int contadorTiempo;
```

```
public SimuladorTrafico() {
    this.mapa = new MapaCarreteras();
    this.contadorTiempo = 0;
    Comparator<Evento> cmp = new Comparator<Evento>() {...};
    this.eventos = new SortedArrayList<>(); // estructura ordenada por "tiempo"
}

public void ejecuta(int pasosSimulacion, OutputStream ficheroSalida) {
    int limiteTiempo = this.contadorTiempo + pasosSimulacion - 1;
    while (this.contadorTiempo <= limiteTiempo) {
        // ejecutar todos los eventos correspondientes a "this.contadorTiempo"
        // actualizar "mapa"
        // escribir el informe en "ficheroSalida", controlando que no sea null.
    }
}
```

16

SimuladorTrafico

```
private MapaCarreteras mapa;  
private List<Evento> eventos;  
private int contadorTiempo;
```

```
public SimuladorTrafico() {  
    this.mapa = new MapaCarreteras();  
    this.contadorTiempo = 0;  
    Comparator<Evento> cmp = new Comparator<Evento>() {...};  
    this.eventos = new SortedArrayList<>(); // estructura ordenada por "tiempo"  
}  
  
public void ejecuta(int pasosSimulacion, OutputStream ficheroSalida) {  
    int limiteTiempo = this.contadorTiempo + pasosSimulacion - 1;  
    while (this.contadorTiempo <= limiteTiempo) {  
        // ejecutar todos los eventos correspondientes a "this.contadorTiempo"  
        // actualizar "mapa"  
        // escribir el informe en "ficheroSalida", controlando que no sea null.  
    }  
  
    public void insertaEvento(Evento e) {  
        // inserta un evento en "eventos", controlando que el tiempo de  
        // ejecución del evento sea menor que "contadorTiempo"  
    }  
}
```

17

Mapa de Carreteras

- Almacena los objetos de la simulación.
- Por tanto necesita atributos para almacenar las carreteras, los vehículos y los cruces. No se requiere orden en estas estructuras.
- Además para avanzar la simulación, hay que realizar búsquedas de carreteras, vehículos y cruces, y por lo tanto conviene tener estructuras auxiliares para agilizar dichas búsquedas.

```
public class MapaCarreteras {  
  
    private List<Carretera> carreteras;  
    private List<Cruce> cruces;  
    private List<Vehiculo> vehiculos;  
  
    // estructuras para agilizar la búsqueda (id,valor)  
    private Map<String, Carretera> mapaDeCarreteras;  
    private Map<String, Cruce> mapaDeCruces;  
    private Map<String, Vehiculo> mapaDeVehiculos;  
    ...  
}
```

19

SortedArrayList

```
public class SortedArrayList<E> extends ArrayList<E> {  
  
    private Comparator<E> cmp;  
  
    public SortedArrayList(Comparator<E> cmp) {...}  
  
    @Override  
    public boolean add(E e) {  
        // programar la inserción ordenada  
    }  
  
    @Override  
    public boolean addAll(Collection<? extends E> c) {  
        // programar inserción ordenada (invocando a add)  
    }  
  
    // sobrescribir los métodos que realizan operaciones de  
    // inserción basados en un índice para que lancen excepción.  
    // Ten en cuenta que esta operación rompería la ordenación.  
    // estos métodos son add(int index, E element),  
    // addAll(int index, Collection<? Extends E>) y E set(int index, E element).  
}
```

18

Mapa de Carreteras

```
public MapaCarreteras() {  
    // inicializa los atributos a sus constructoras por defecto.  
    // Para carreteras, cruces y vehículos puede usarse ArrayList.  
    // Para los mapas puede usarse HashMap  
}  
  
public void addCruce(String idCruce, Cruce cruce) {  
    // comprueba que "idCruce" no existe en el mapa.  
    // Si no existe, lo añade a "cruces" y a "mapaDeCruces".  
    // Si existe lanza una excepción.  
}
```

"Solo se ejecuta una vez por Cruce. Cuando se procesa su evento".

20

Mapa de Carreteras

```
public void addVehiculo(String idVehiculo,Vehiculo vehiculo) {  
    // comprueba que "idVehiculo" no existe en el mapa.  
    // Si no existe, lo añade a "vehiculos" y a "mapaDeVehiculos",  
    // y posteriormente solicita al vehiculo que se mueva a la siguiente  
    // carretera de su itinerario (moverASiguienteCarretera).  
    // Si existe lanza una excepción.  
}
```

"Solo se ejecuta una vez por vehículo. Cuando se procesa el evento".

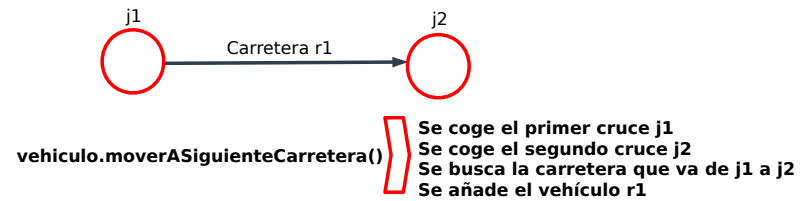
21

Mapa de Carreteras

```
public void addVehiculo(String idVehiculo,Vehiculo vehiculo) {  
    // comprueba que "idVehiculo" no existe en el mapa.  
    // Si no existe, lo añade a "vehiculos" y a "mapaDeVehiculos",  
    // y posteriormente solicita al vehiculo que se mueva a la siguiente  
    // carretera de su itinerario (moverASiguienteCarretera).  
    // Si existe lanza una excepción.  
}
```

"Solo se ejecuta una vez por vehículo. Cuando se procesa el evento".

Vehiculo (idVehiculo) con itinerario [j1,j2,j3]



22

Mapa de Carreteras

```
public void addCarretera(String idCarretera,  
    Cruce origen,  
    Carretera carretera,  
    Cruce destino) {  
    // comprueba que "idCarretera" no existe en el mapa.  
    // Si no existe, lo añade a "carreteras" y a "mapaDeCarreteras",  
    // y posteriormente actualiza los cruces origen y destino como sigue:  
    // - Añade al cruce origen la carretera, como "carretera saliente"  
    // - Añade al cruce destino la carretera, como "carretera entrante"  
    // Si existe lanza una excepción.  
}
```

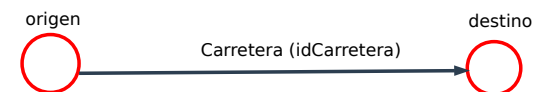
23

Mapa de Carreteras

```
public void addCarretera(String idCarretera,  
    Cruce origen,  
    Carretera carretera,  
    Cruce destino) {  
    // comprueba que "idCarretera" no existe en el mapa.  
    // Si no existe, lo añade a "carreteras" y a "mapaDeCarreteras",  
    // y posteriormente actualiza los cruces origen y destino como sigue:  
    // - Añade al cruce origen la carretera, como "carretera saliente"  
    // - Añade al cruce destino la carretera, como "carretera entrante"  
    // Si existe lanza una excepción.  
}
```

Carreteras salientes: [...idCarretera]

Carreteras entrantes: [...idCarretera]



24

Mapa de Carreteras

```
public String generateReport(int time) {
    String report = "";
    // genera informe para cruces
    // genera informe para carreteras
    // genera informe para vehiculos
}

public void actualizar() {
    // llama al método avanza de cada cruce
    // llama al método avanza de cada carretera
}

public Cruce getCruce(String id) {
    // devuelve el cruce con ese "id" utilizando el mapaDeCruces.
    // sino existe el cruce lanza excepción.
}

public Vehiculo getVehiculo(String id) {
    // devuelve el vehiculo con ese "id" utilizando el mapaDeVehiculos.
    // sino existe el vehiculo lanza excepción.
}

public Carretera getCarretera(String id) {
    // devuelve la carretera con ese "id" utilizando el mapaDeCarreteras.
    // sino existe la carretera lanza excepción.
}
```

25

Eventos

```
public abstract class Evento {

    protected Integer tiempo;

    public Evento(int tiempo) { ... }

    public int getTiempo() { ... }

    ...

    // cada clase que hereda implementa su método ejecuta, que creará
    // el correspondiente objeto de la simulación.

    public abstract void ejecuta(MapaCarreteras mapa);
}
```

27

Eventos

- Todos los eventos se ejecutan en un “tiempo” concreto.
- Tenemos eventos de cuatro tipos:

EventoNuevaCarretera: Crea una nueva carretera

EventoNuevoCruce: Crea un nuevo cruce

EventoNuevoVehiculo: Crea un nuevo vehículo.

EventoAveriaCoche: Avería ciertos coches.

- Todas estas clases heredan de la clase abstracta **Evento**.

26

EventoNuevoVehiculo

```
public class EventoNuevoVehiculo extends Evento {

    protected String id;
    protected Integer velocidadMaxima;
    protected String[] itinerario;

    public EventoNuevoVehiculo(int tiempo, String id, int velocidadMaxima,
                                String[] itinerario) {

        ...
    }

    @Override
    public void ejecuta(MapaCarreteras mapa) {
        List<Cruce> iti = ParserCarreteras.parseaListaCruces(this.itinerario, mapa);
        // si iti es null o tiene menos de dos cruces lanzar excepción
        // en otro caso crear el vehiculo y añadirlo al mapa.
    }

    @Override
    public String toString() {...}
}
```

```
[new_vehicle]
time = 0
id = v2
itinerary = j4,j2,j5
max_speed = 20
```

28

EventoNuevoVehiculo

```
public class EventoNuevoVehiculo extends Evento {  
  
    protected String id;  
    protected Integer velocidadMaxima;  
    protected String[] itinerario;  
  
    public EventoNuevoVehiculo(int tiempo, String id, int velocidadMaxima,  
                                String[] itinerario) {  
        ...  
    }  
  
    @Override  
    public void ejecuta(MapaCarreteras mapa) {  
        List<Cruce> iti = ParserCarreteras.parseaListaCruces(this.itinerario, mapa);  
        // si iti es null o tiene menos de dos cruces lanzar excepción  
        // en otro caso crear el vehículo y añadirlo al mapa.  
    }  
    @Override  
    public String toString() {...}  
}
```

this.itinerario = ["j1", "j2", "j3"] entonces **parseaListaCruces** devuelve la lista de cruces cuyos identificadores son los de "j1", "j2" y "j3". Para ello utiliza el mapa de carreteras.

29

EventoNuevoCruce

```
public class EventoNuevoCruce extends Evento {  
  
    protected String id;  
  
    public EventoNuevoCruce(int time, String id) {...}  
  
    @Override  
    public void ejecuta(MapaCarreteras mapa) {  
        // crea el cruce y se lo añade al mapa  
    }  
  
    @Override  
    public String toString() {...}  
}
```

```
[new_junction]  
time = 0  
id = j1
```

30

EventoNuevaCarretera

```
public class EventoNuevaCarretera extends Evento {  
  
    protected String id;  
    protected Integer velocidadMaxima;  
    protected Integer longitud;  
    protected String cruceOrigenId;  
    protected String cruceDestinoId;  
  
    public EventoNuevaCarretera(int tiempo, String id, String origen,  
                                String destino, int velocidadMaxima, int longitud) {...}  
  
    @Override  
    public void ejecuta(MapaCarreteras mapa) {  
        // obten cruce origen y cruce destino utilizando el mapa  
        // crea la carretera  
        // añade al mapa la carretera  
    }  
  
    @Override  
    public String toString() {...}  
}
```

```
[new_road]  
time = 0  
id = r4  
src = j2  
dest = j5  
max_speed = 20  
length = 100
```

31

Objetos de la Simulación

- Tenemos vehículos, carreteras y cruces.
- Todos ellos contienen un identificador común y métodos comunes para "generarInforme" y "avanza".
- Por lo tanto tendremos una clase abstracta **ObjetoSimulacion** de la que heredan Vehiculo, Carretera y Cruce.

```
public abstract class ObjetoSimulacion {  
  
    protected String id;  
  
    public ObjetoSimulacion(String id) {...}  
    public String getId() {...}  
  
    @Override  
    public String toString() {...}  
  
    public String generaInforme(int tiempo) {...}  
    public abstract void avanza();  
    ...  
}
```

32

Objetos de la Simulación

```
public String generaInforme(int tiempo) {
    IniSeccion is = new IniSeccion(this.getNombreSeccion());
    is.setValue("id", this.id);
    is.setValue("time", tiempo);
    this.completaDetallesSeccion(is);
    return is.toString();
}

// Los métodos getNombreSeccion y completaDetallesSeccion
// tendrán que implementarlos cada subclase de ObjetoSimulacion
```

33

Vehículos

```
public int getLocalizacion() {...}

public int getTiempoDeInfraccion() {...}

public void setVelocidadActual(int velocidad) {
    // Si "velocidad" es negativa, entonces la "velocidadActual" es 0.
    // Si "velocidad" excede a "velocidadMaxima", entonces la
    // "velocidadActual" es "velocidadMaxima"
    // En otro caso, "velocidadActual" es "velocidad"
}
```

35

Vehículos

```
public class Vehiculo extends ObjetoSimulacion {

    protected Carretera carretera; // carretera en la que está el vehículo
    protected int velocidadMaxima; // velocidad máxima
    protected int velocidadActual; // velocidad actual
    protected int kilometraje; // distancia recorrida
    protected int localizacion; // localización en la carretera
    protected int tiempoAveria; // tiempo que estará averiado
    protected List<Cruce> itinerario; // itinerario a recorrer (mínimo 2)
    ...

    public Vehiculo(String id, int velocidadMaxima, List<CruceGenerico?> iti) {
        super(id);

        // comprobar que la velocidadMaxima es mayor o igual que 0, y
        // que el itinerario tiene al menos dos cruces.
        // En caso de no cumplirse lo anterior, lanzar una excepción.

        // inicializar los atributos teniendo en cuenta los parámetros.
        // al crear un vehículo su "carretera" será inicialmente "null".
    }
}
```

34

Vehículos

```
public int getLocalizacion() {...}

public int getTiempoDeInfraccion() {...}

public void setVelocidadActual(int velocidad) {
    // Si "velocidad" es negativa, entonces la "velocidadActual" es 0.
    // Si "velocidad" excede a "velocidadMaxima", entonces la
    // "velocidadActual" es "velocidadMaxima"
    // En otro caso, "velocidadActual" es "velocidad"
}

public void setTiempoAveria(Integer duracionAveria) {
    // Comprobar que "carretera" no es null.
    // Se fija el tiempo de avería de acuerdo con el enunciado.
    // Si el tiempo de avería es finalmente positivo, entonces
    // la "velocidadActual" se pone a 0
}
```

36

Vehículos

```
public int getLocalizacion() {...}

public int getTiempoDeInfraccion() {...}

public void setVelocidadActual(int velocidad) {
    // Si "velocidad" es negativa, entonces la "velocidadActual" es 0.
    // Si "velocidad" excede a "velocidadMaxima", entonces la
    // "velocidadActual" es "velocidadMaxima"
    // En otro caso, "velocidadActual" es "velocidad"
}

public void setTiempoAveria(Integer duracionAveria) {
    // Comprobar que "carretera" no es null.
    // Se fija el tiempo de avería de acuerdo con el enunciado.
    // Si el tiempo de avería es finalmente positivo, entonces
    // la "velocidadActual" se pone a 0
}

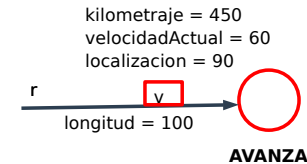
@Override
protected void completaDetallesSeccion(IniSection is) {
    ...
    is.setValue("location", this.haLlegado ? "arrived" :
        this.carretera + ":" + this.getLocalizacion());
}
```

[vehicle_report]
id = v1
time = 1
speed = 20
kilometraje = 20
faulty = 0
location = r1:20

37

Vehículos

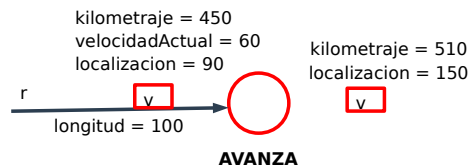
```
@Override
public void avanza() {
    // si el coche está averiado, decrementar tiempoAveria
    // si el coche está esperando en un cruce, no se hace nada.
    // en otro caso:
    1. Actualizar su "localizacion"
    2. Actualizar su "kilometraje"
    3. Si el coche ha llegado a un cruce (localizacion >= carretera.getLength())
        3.1. Poner la localización igual a la longitud de la carretera.
        3.2. Corregir el kilometraje.
        3.3. Indicar a la carretera que el vehículo entra al cruce.
        3.4. Marcar que éste vehículo está en un cruce (this.estEnCruce = true)
}
```



38

Vehículos

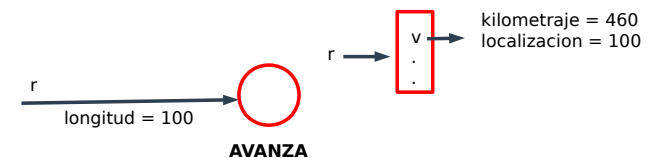
```
@Override
public void avanza() {
    // si el coche está averiado, decrementar tiempoAveria
    // si el coche está esperando en un cruce, no se hace nada.
    // en otro caso:
    1. Actualizar su "localizacion"
    2. Actualizar su "kilometraje"
    3. Si el coche ha llegado a un cruce (localizacion >= carretera.getLength())
        3.1. Poner la localización igual a la longitud de la carretera.
        3.2. Corregir el kilometraje.
        3.3. Indicar a la carretera que el vehículo entra al cruce.
        3.4. Marcar que éste vehículo está en un cruce (this.estEnCruce = true)
}
```



39

Vehículos

```
@Override
public void avanza() {
    // si el coche está averiado, decrementar tiempoAveria
    // si el coche está esperando en un cruce, no se hace nada.
    // en otro caso:
    1. Actualizar su "localizacion"
    2. Actualizar su "kilometraje"
    3. Si el coche ha llegado a un cruce (localizacion >= carretera.getLength())
        3.1. Poner la localización igual a la longitud de la carretera.
        3.2. Corregir el kilometraje.
        3.3. Indicar a la carretera que el vehículo entra al cruce.
        3.4. Marcar que éste vehículo está en un cruce (this.estEnCruce = true)
}
```



40

Vehículos

```
public void moverASiguienteCarretera() {
    // Si la carretera no es null, sacar el vehículo de la carretera.
    // Si hemos llegado al último cruce del itinerario, entonces:
    1. Se marca que el vehículo ha llegado (this.haLlegado = true).
    2. Se pone su carretera a null.
    3. Se pone su "velocidadActual" y "localizacion" a 0.
    // y se marca que el vehículo está en un cruce (this.estaEnCruce = true).
    // En otro caso:
    1. Se calcula la siguiente carretera a la que tiene que ir.
    2. Si dicha carretera no existe, se lanza excepción.
    3. En otro caso, se introduce el vehículo en la carretera.
    4. Se inicializa su localización.
}

@Override
protected String getNombreSeccion() {
    return "vehicle_report";
}
```

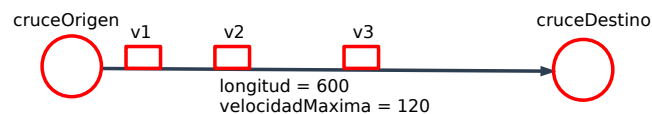
41

Carreteras

```
public class Carretera extends ObjetoSimulacion {

    protected int longitud;           // longitud de la carretera
    protected int velocidadMaxima;    // velocidad máxima
    protected Cruce cruceOrigen;      // cruce del que parte la carretera
    protected Cruce cruceDestino;     // cruce al que llega la carretera
    protected List<Vehiculo> vehiculos; // lista ordenada de vehículos en la
                                     // carretera (ordenada por localización)
    protected Comparator<Vehiculo> comparadorVehiculo; // orden entre vehículos

    public Carretera(String id, int length, int maxSpeed, Cruce src, Cruce dest) {
        // se inicializan los atributos de acuerdo con los parámetros.
        // se fija el orden entre los vehículos: (inicia comparadorVehiculo)
        - la localización 0 es la menor
    }
}
```



Vehiculos[0] = v3, vehiculos[1] = v2, vehiculos[2] = v1
(se ordena la lista vehículos de mayor a menor de acuerdo con la localización)

43

Carreteras

```
public class Carretera extends ObjetoSimulacion {

    protected int longitud;           // longitud de la carretera
    protected int velocidadMaxima;    // velocidad máxima
    protected Cruce cruceOrigen;      // cruce del que parte la carretera
    protected Cruce cruceDestino;     // cruce al que llega la carretera
    protected List<Vehiculo> vehiculos; // lista ordenada de vehículos en la
                                     // carretera (ordenada por localización)
    protected Comparator<Vehiculo> comparadorVehiculo; // orden entre vehículos

    public Carretera(String id, int length, int maxSpeed, Cruce src, Cruce dest) {
        // se inicializan los atributos de acuerdo con los parámetros.
        // se fija el orden entre los vehículos: (inicia comparadorVehiculo)
        - la localización 0 es la menor
    }
}
```

42

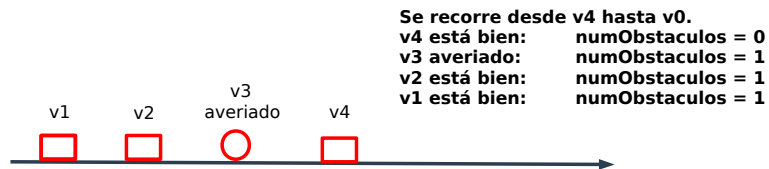
Carreteras

```
@Override
public void avanza() {
    // calcular velocidad base de la carretera
    // inicializar obstáculos a 0
    // Para cada vehículo de la lista "vehiculos":
    1. Si el vehículo está averiado se incrementa el número de obstaculos.
    2. Se fija la velocidad actual del vehículo
    3. Se pide al vehículo que avance.
    // ordenar la lista de vehículos
}
```

44

Carreteras

```
@Override
public void avanza() {
    // calcular velocidad base de la carretera
    // inicializar obstáculos a 0
    // Para cada vehículo de la lista "vehiculos":
    1. Si el vehículo está averiado se incrementa el número de obstáculos.
    2. Se fija la velocidad actual del vehículo
    3. Se pide al vehículo que avance.
    // ordenar la lista de vehículos
}
```



45

Carreteras

```
public void entraVehiculo(Vehiculo vehiculo) {
    // Si el vehículo no existe en la carretera, se añade a la lista de vehículos y
    // se ordena la lista.
    // Si existe no se hace nada.
}

[road_report]
id = r1
time = 1
vehicles = (v1,10),(v2,10)

public void saleVehiculo(Vehiculo vehiculo) {
    // elimina el vehículo de la lista de vehículos
}

public void entraVehiculoAlCruce(Vehiculo v) {
    // añade el vehículo al "cruceDestino" de la carretera"
}

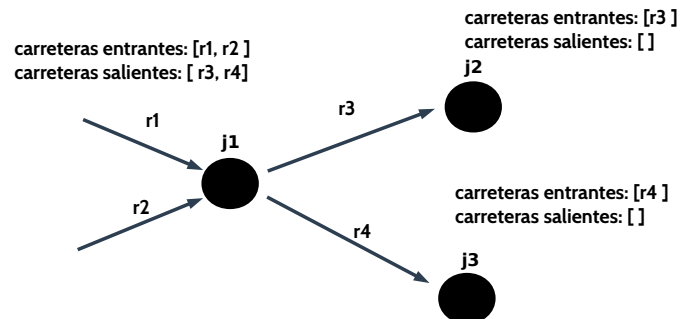
protected int calculaVelocidadBase() {...}
protected int calculaFactorReduccion(int obstaculos) {...}

@Override
protected String getNombreSeccion() {...}

@Override
protected void completaDetallesSeccion(IniSection is) {
    // crea "vehicles = (v1,10),(v2,10) "
}
```

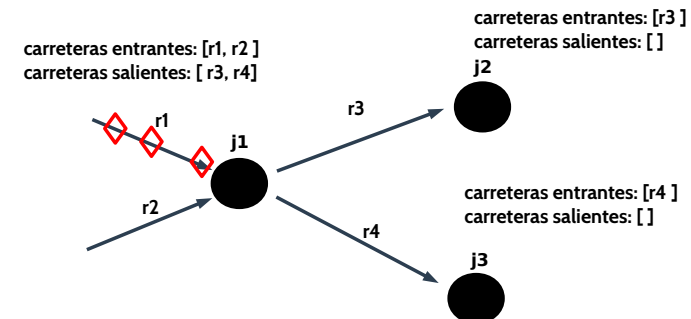
46

Cruce



47

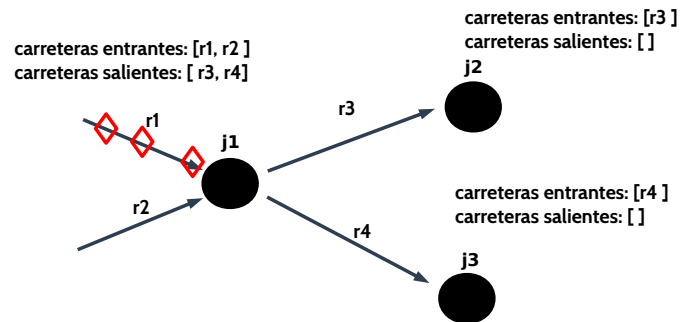
Cruce



Tres vehículos v1, v2 y v3 en r1
v3 ha llegado al cruce j1

48

Cruce

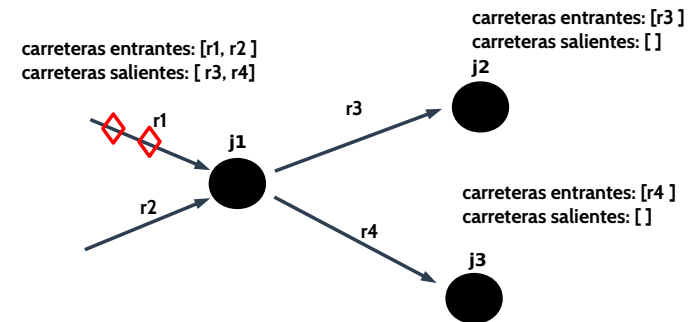


La carretera entrante r1, necesita añadir v3 a su cola de vehículos que están esperando a atravesar el cruce

Tres vehículos v1, v2 y v3 en r1
v3 ha llegado al cruce j1

49

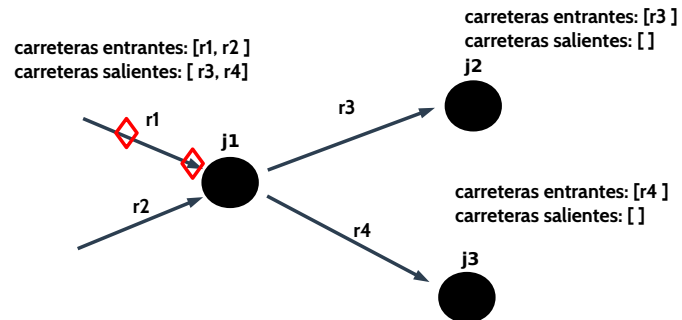
Cruce



Carretera entrante r1 : (v3) // lleva una cola de vehículos esperando a atravesar el cruce

50

Cruce

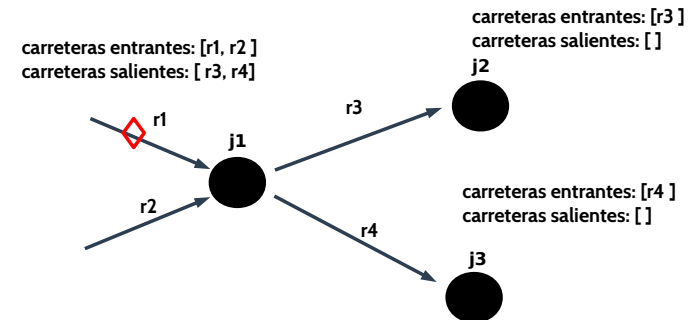


v2 llega al cruce, y pasa a la cola de vehículos de r1

Carretera entrante r1 : (v3) // lleva una cola de vehículos esperando a atravesar el cruce

51

Cruce

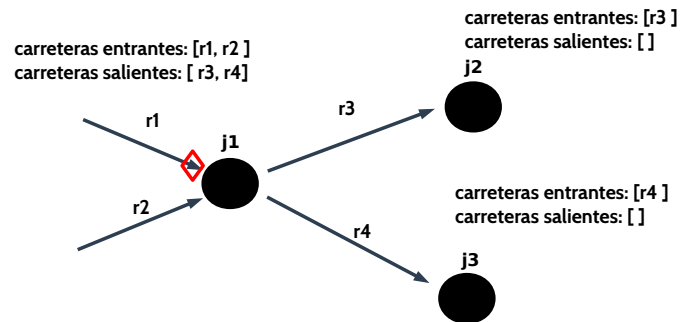


v2 llega al cruce, y pasa a la cola de vehículos de r1

Carretera entrante r1 : (v3,v2) // lleva una cola de vehículos esperando a atravesar el cruce

52

Cruce

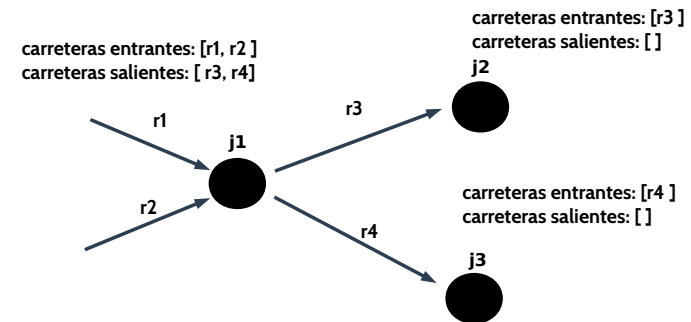


v1 llega al cruce, y pasa a la cola de vehículos de r1

Carretera entrante r1 : (v3,v2) // lleva una cola de vehículos esperando a atravesar el cruce

53

Cruce



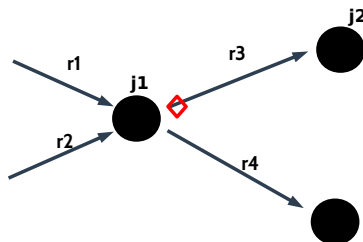
v1 llega al cruce, y pasa a la cola de vehículos de r1

Carretera entrante r1 : (v3,v2,v1) // lleva una cola de vehículos esperando a atravesar el cruce

54

Cruce

- La forma de avanzar del cruce es buscar la carretera entrante con su **semáforo en verde**. Solo pueda haber una carretera entrante con el semáforo verde. Supongamos que es r1
- Entonces coge el primer vehículo de la cola, en este caso v3, lo elimina de la cola, y le pide al vehículo que se mueva a la siguiente carretera.
- v3 tiene un itinerario [j1,j2]. Entonces utiliza j1 para preguntarle qué carretera va de j1 a j2. El cruce contestaría que r3, y v3 se coloraría en r3.



Carretera entrante r1 : (v2,v1)

55

CarreteraEntrante

```
public class CarreteraEntrante {
    protected Carretera carretera;
    protected List<Vehiculo> colaVehiculos;
    protected boolean semaforo; // true=verde, false=rojo

    public CarreteraEntrante(Carretera carretera) {
        // inicia los atributos.
        // el semáforo a rojo
    }

    void ponSemaforo(boolean color) {...}

    public void avanzaPrimerVehiculo() {
        // coge el primer vehiculo de la cola, lo elimina,
        // y le manda que se mueva a su siguiente carretera.
    }

    @Override
    public String toString() {...}
}
```

56

Cruce

```
abstract public class Cruce extends ObjetoSimulacion {
    protected int indiceSemaforoVerde; // lleva el índice de la carretera entrante
                                     // con el semáforo en verde
    protected List<CarreteraEntrante> carreterasEntrantes;

    // para optimizar las búsquedas de las carreterasEntrantes
    // (IdCarretera, CarreteraEntrante)
    protected Map<String,CarreteraEntrante> mapaCarreterasEntrantes;
    protected Map<Cruce, Carretera> CarreterasSalientes;
}
```

57

Cruce

```
public Cruce(String id) {...}

public Carretera carreteraHaciaCruce(Cruce cruce) {
    // devuelve la carretera que llega a ese cruce desde "this"
}

public void addCarreteraEntranteAlCruce(String idCarretera, Carretera carretera) {
    // añade una carretera entrante al "mapaCarreterasEntrantes" y
    // a las "carreterasEntrantes"
}

public void addCarreteraSalienteAlCruce(Cruce destino, Carretera road) {
    // añade una carretera saliente
}

public void entraVehiculoAlCruce(String idCarretera, Vehiculo vehiculo){
    // añade el "vehículo" a la carretera entrante "idCarretera"
}

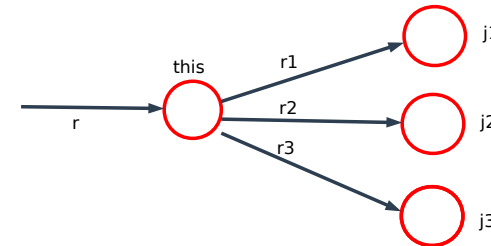
protected void actualizaSemaforos(){
    // pone el semáforo de la carretera actual a "rojo", y busca la siguiente
    // carretera entrante para ponerlo a "verde"
}
```

59

Cruce

```
abstract public class Cruce extends ObjetoSimulacion {
    protected int indiceSemaforoVerde; // lleva el índice de la carretera entrante
                                     // con el semáforo en verde
    protected List<CarreteraEntrante> carreterasEntrantes;

    // para optimizar las búsquedas de las carreterasEntrantes
    // (IdCarretera, CarreteraEntrante)
    protected Map<String,CarreteraEntrante> mapaCarreterasEntrantes;
    protected Map<Cruce, Carretera> CarreterasSalientes;
}
```



Carreteras salientes a
"this" almacena:

(j1,r1)
(j2,r2)
(j3,r3)

Si v tiene trayectoria (this,j2), y llega a "this", entonces para encontrar su siguiente carretera se pregunta this.carreteraHaciaCruce(j2), que devolvería r2. Así "v" se coloca en r2

58

Cruce

```
@Override
public void avanza() {
    // Si "carreterasEntrantes" es vacío, no hace nada.
    // en otro caso "avanzaPrimerVehiculo" de la carretera con el semáforo verde.
    // Posteriormente actualiza los semáforos.
}

@Override
protected String getNombreSeccion() {...}

@Override
protected void completaDetallesSeccion(IniSection is) {
    // genera la sección queues = (r2,green,[]),...

    [junction_report]
    id = j3
    time = 10
    queues = (r2,green,[])
}
```

60