Práctica 4: Simulador de tráfico: nuevas carreteras, cruces y vehículos

Puri Arenas Sánchez
Facultad de Informática (UCM)

Parte II: Clases

- Aparecen nuevos tipos de **carreteras**, **vehículos** y **cruces**:
- Además de las carreteras de la Parte 1, tendremos:

  1. Autopista: Puede tener varios carriles. Su *velocidad base* y *factor de reducción* dependen del número de carriles.

  2. Camino: Su *velocidad base* es la misma que la velocidad máxima de la carretera. Su *factor de reducción* es 1 más el número de coches averiados.

- Además de los cruces de la Parte 1, tendremos dos tipos más. Se diferencian unos de otros en su forma de elegir el semáforo a cambiar. Los nuevos cruces, a diferencia de los anteriores, permanecen en verde durante un intervalo de tiempo. Consumido el intervalo, se vuelve a fijar dependiendo del use del cruce, y posteriormente:

  1. Circulares: Se pone en verde el semáforo de la siguiente carretera entrante (una vez consumido un intervalo de tiempo).

  2. Congestionados: Se pone en verde la siguiente carretera entrante que tiene más vehículos esperando (una vez consumido un intervalo de tiempo)..

- Vehiculos: Se diferencian en su tiempo de avería. Además de los normales tenemos:

  1. Coches: Se averían de forma aleatoria, con una cierta probabilidad.

  2. Bicicletas: Se averían con menos frecuencia.

Comentarios Generales

- El **Controlador** es igual que en la Parte I.
- El **SimuladorTrafico** es igual que en la Parte I.
- **ParserEventos** es igual, ampliándola con los nuevos tipos de eventos que se admiten.

  ConstructorEventoNuevaAutopista
  ConstructorEventoNuevoCamino
  ConstructorEventoNuevoCoche
  ConstructorEventoNuevaBicicleta
  ConstructorEventoNuevoCruceCircular
  ConstructorEventoNuevoCruceCongestionado

- **ConstructorEventos** es igual que la Parte I. Las clases que heredan deben implementarse de forma similar a las existentes para que creen objetos de los nuevos tipos que se admiten.

Mapa de Carreteras

- Ahora tenemos jerarquías de **Carreteras**, **Cruces** y **Vehículos**. El **MapaCarreteras** tiene que poder representar a todos ellos. Si estamos utilizando genéricos para implementar la jerarquía de **Cruces**, entonces:

```
public class MapaCarreteras {

  private List<Carretera> carreteras;
  private List<CruceGenerico<?>> cruces;
  private List<Vehiculo> vehiculos;

  // estructuras para agilizar la búsqueda (id,valor)
  private Map<String, Carretera> mapaDeCarreteras;
  private Map<String, CruceGenerico<?>> mapaDeCruces;
  private Map<String, Vehiculo> mapaDeVehiculos;
  ...
}
```

Los métodos son iguales cambiando Cruce por CruceGenerico<?>

Eventos

- Además de los eventos de la Parte I, tenemos más tipos de eventos:

  EventoNuevaAutopista

  EventoNuevoCamino

  EventoNuevoCruceCircular

  EventoNuevoCruceCongestionado

  EventoNuevoCoche.

  EventoNuevaBicicleta.

- Todas estas clases heredan de la clase abstracta **Evento,** que es igual que en la Parte I.

EventoNuevoCruce

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

```java
public class EventoNuevoCruce extends Evento {

 protected String id;

 public EventoNuevoCruce(int time, String id) {
  super(time);
  this.id = id;
 }

 @Override
 public void ejecuta(MapaCarreteras mapa) {
   mapa.addCruce(this.id, this.creaCruce()); // lo sobreescriben las
 }                                           // clases que heredan

 protected CruceGenerico<?> creaCruce() {
   return new Cruce(this.id);
 }

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

EventoNuevoCruceCircular

```
public class EventoNuevoCruceCircular extends EventoNuevoCruce {

  protected Integer maxValorIntervalo;
  protected Integer minValorIntervalo;

  public EventoNuevoCruceCircular(int time, String id,
                                  int minValorIntervalo, int maxValorIntervalo) {
   super(time, id);
   this.maxValorIntervalo = maxValorIntervalo;
   this.minValorIntervalo = minValorIntervalo;
  }

  @Override
  protected CruceGenerico<?> creaCruce() {
    return new CruceCircular(this.id, this.minValorIntervalo,
                                      this.maxValorIntervalo);
  }

  ...
}
```

```
[new_junction]
time = 0
id = j3
type = rr
```

EventoNuevoCruceCongestionado

```
public class EventoNuevoCruceCongestionado extends EventoNuevoCruce {

 ...

 @Override
 protected CruceGenerico<?> creaCruce() {
   return new CruceCongestionado(this.id);
 }

 ...

}
```

```
[new_junction]
time = 0
id = j3
type = mc
```

EventoNuevaCarretera

```
[new_road]
time = 0
id = r1
src = j1
dest = j2
max_speed = 40
length = 100
```

```
public class EventoNuevaCarretera extends Evento {

  ...

  @Override
  public void ejecuta(MapaCarreteras mapa) {
   CruceGenerico<?> cruceOrigen = mapa.getCruce(this.cruceOrigenId);
   CruceGenerico<?> cruceDestino = mapa.getCruce(this.cruceDestinoId);
   Carretera c = creaCarretera(cruceOrigen,cruceDestino); // lo sobreescriben
   mapa.addCarretera(this.id,cruceOrigen,c,cruceDestino);
  }

  protected Carretera creaCarretera(CruceGenerico<?> cruceOrigen,
                                    CruceGenerico<?> cruceDestino) {
      return new Carretera(this.id, this.longitud, this.velocidadMaxima,
                           cruceOrigen, cruceDestino);
  }
...
}
```

Tengo que revisar el tema de genericos, cambiar en la practica4

EventoNuevaAutopista

```
[new_road]
time = 0
id = r1
src = j1
dest = j3
max_speed = 20
length = 100
lanes = 2
type = lanes
```

```
public class EventoNuevaAutopista extends EventoNuevaCarretera {

  protected Integer numCarriles;

  public EventoNuevaAutopista(...){ ... }

  @Override
  protected Carretera creaCarretera(CruceGenerico<?> cruceOrigen,
                                    CruceGenerico<?> cruceDestino) {
    return new Autopista(...);
  }
 ...
}
```

EventoNuevoCamino

```
[new_road]
time = 0
id = r1
src = j1
dest = j3
max_speed = 20
length = 100
type = dirt
```

```
public class EventoNuevoCamino extends EventoNuevaCarretera {

 ...

@Override
protected Carretera creaCarretera(CruceGenerico<?> cruceOrigen,
                                  CruceGenerico<?> cruceDestino) {
 return new Camino(this.id, this.longitud, this.velocidadMaxima,
                   cruceOrigen, cruceDestino);
}

 ...

}
```

Objetos de la Simulación: Vehiculos

- La clase **ObjetoSimulación** es igual.
- La clase **Vehiculo** es igual.
- La clase **Coche** que extiende a **Vehiculo**, y tiene las siguientes peculiaridades en su método **avanza**:

1. Si el coche está averiado, entonces actualizamos el kilómetro donde se ha producido dicha averia `this.kmUltimaAveria = this.kilometraje`

2. Si no está averiado, ha recorrido un número determinado de kilómetros (**resistenciaKm**) y en ese instante la probabilidad de avería calculada es menor que su **probabilidadDeAveria**, el coche pone su tiempoAveria a

```
this.tiempoAveria = this.numAleatorio.nextInt(this.duracionMaximaAveria) + 1;
```

3. Se llama al método **avanza** de la super clase (**Vehiculo**).

Clase Coche

```
protected int kmUltimaAveria;
protected int resistenciaKm;
protected int duracionMaximaAveria;
protected double probabilidadDeAveria;
protected Random numAleatorio;
```

```
[new_vehicle]
time = <NONEG-INTEGER>
id = <VEHICLE-ID>
itinerary = <JUNC-ID>,<JUNC-ID>(,<JUNC-ID>)*
max_speed = <POSITIVE-INTEGER>
type = car
resistance = <POSITIVE-INTEGER>              // resistenciaKm
fault_probability = <NONEG-DOUBLE>           // probabilidadDeAveria
max_fault_duration = <POSITIVE-INTEGER>      // duracionMaximaAveria
seed = <POSITIVE-LONG>                       // numAleatorio
```

Clase Coche

```
public class Coche extends Vehiculo {

  public Coche(String id, int velocidadMaxima, int resistencia,
               double probabilidad, long semilla, int duracionMaximaInfraccion,
               List<CruceGenerico<?>> itinerario) {
  ...
  this.numAleatorio = new Random(semilla);
  }

  @Override
  public void avanza() {
   // - Si el coche está averiado poner “kmUltimaAveria” a “kilometraje”.
   // - Sino el coche no está averiado y ha recorrido “resistenciakm”, y además
   //   “numAleatorio”.nextDouble() < “probabilidadDeAveria”, entonces
   //   incrementar “tiempoAveria” con “numAleatorio.nextInt(“duracionMaximaAveria”)+1
   // - Llamar a super.avanza();
 }

 @Override
 protected void completaDetallesSeccion(IniSection is) {...}
}
```

Clase Bicicleta

Avanzan como los vehículos normales. Cuando un evento **make_vehicle_faulty** hace referencia una bicicleta, su tiempo de avería sólo se modificará (**setTiempoAveria(...)**) si la bicicleta está circulando a una velocidad mayor o igual que la mitad de su **velocidad máxima**.

```
this.velocidadActual >= this.velocidadMaxima / 2
```

Objetos de la Simulación: Carreteras

- La clase **Carretera** es igual, cambiando **Cruce** por **CruceGenerico<?>**.
- La clases **Autopista** y **Camino** que extienden a **Carretera**, sólo
- Se diferencian de ésta en la forma de **calcular la velocidad base** y el **factor de reducción**.

Además la clase **Autopista** contiene un atributo **numCarriles** .

```
public class Autopista extends Carretera {

  private int numCarriles;

  @Override
  protected int calculaVelocidadBase() {
     return ...
  }

  @Override
  protected int calculaFactorReduccion(int obstacles) {
    return obstacles < this.numCarriles ? 1 : 2;
  }

  ...
}
```

Objetos de la Simulación: Carreteras

```
public class Camino extends Carretera {

  public Camino(String id, int longitud, int velocidadMaxima,
                CruceGenerico<?> cruceOrigen,
                CruceGenerico<?> cruceDestino) {
  ...
  }

  @Override
  protected int calculaVelocidadBase() { return this.velocidadMaxima; }

  @Override
  protected int calculaFactorReduccion(int obstacles) {
    return obstacles + 1;
  }

  @Override
  protected void completaDetallesSeccion(IniSection is) {...}
}
```

Objetos de la Simulación: Cruces

- Ahora vamos a realizar una jerarquía de **Cruces** utilizando *tipos genéricos*.
- Hay tres tipos de cruces: los de la Parte I, y *dos nuevos tipos de cruces que cambian el semáforo en función de un intervalo de tiempo*.
- Los cruces de la Parte I utilizan una colección de **CarreteraEntrante**. Los nuevos cruces, que manejan intervalos, utilizan una colección de **CarreteraEntranteConIntervalo**, que hereda de la clase anterior, pero contiene nuevos atributos para manejar los intervalos.

```
public class CarreteraEntranteConIntervalo extends CarreteraEntrante {


private int intervaloDeTiempo;      // Tiempo que ha de transcurrir para poner
                                    // el semáforo de la carretera en rojo
private int unidadesDeTiempoUsadas;     // Se incrementa cada vez que
                                        // avanza un vehículo
private boolean usoCompleto;    // Controla que en cada paso con el semáforo
                                // en verde, ha pasado un vehículo
private boolean usadaPorUnVehiculo;     // Controla que al menos ha pasado un
                                        // vehículo mientras el semáforo estaba
                                        // en verde.
...
```

CarreteraEntranteConIntervalo

```
public class CarreteraEntranteConIntervalo extends CarreteraEntrante {
...
  protected CarreteraEntranteConIntervalo(Carretera carretera, int intervalTiempo) {
    super(carretera);
    ...
  }

  @Override
  protected void avanzaPrimerVehiculo() {
    // Incrementa unidadesDeTiempoUsadas
    // Actualiza usoCompleto:
    //   - Si “colaVehiculos” es vacía, entonces “usoCompleto=false”
    //   - En otro caso saca el primer vehículo “v” de la “colaVehiculos”,
    //     y le mueve a la siguiente carretera (“v.moverASiguienteCarretera()”)
    //     Pone “usadaPorUnVehiculo” a true.
  }

  public boolean tiempoConsumido() {
   // comprueba si se ha agotado el intervalo de tiempo.
   // “unidadesDeTiempoUsadas >= “intervaloDeTiempo”
  }
  public boolean usoCompleto() { ...  } // método get
  public boolean usada() {...}   // método get
  ...
}
```

Jerarquía de Cruces

(abstract) CruceGenerico<T extends CarreteraEntrante> extends ObjetoSimulacion

Cruce extends CruceGenerico<CarreteraEntrante>

CruceCircular extends CruceGenerico<CarreteraEntranteConIntervalo>

CruceCongestionado extends CruceGenerico<CarreteraEntranteConIntervalo>

CruceGenerico

- Similar a **Cruce**, pero introduciendo los parámetros de tipo. Ahora pasa a ser abstracta.
- El método **actualizaSemaforos** pasa a ser abstracto.
- El método **addCarreteraEntranteAlCruce(...)** ya no crea objetos de tipo **Cruce**, sino que usa un método abstracto **T creaCarreteraEntrante(Carretera carretera)**.
- El resto de métodos son similares a los de **Cruce**, adaptando la sintáxis a los genéricos. Utiliza **CruceGenerico<?>** para que Java infiera el tipo siempre que sea posible.

```
abstract public class CruceGenerico<T extends CarreteraEntrante>
                      extends ObjetoSimulacion {

  protected int indiceSemaforoVerde;
  protected List<T> carreterasEntrantes;
  protected Map<String,T> mapaCarreterasEntrantes;
  protected Map<CruceGenerico<?>, Carretera> carreterasSalientes;
```

CruceGenerico

```
abstract public class CruceGenerico<T extends CarreteraEntrante> extends ObjetoSimulacion {

   public CruceGenerico(String id) {...}

   public Carretera carreteraHaciaCruce(CruceGenerico<?> cruce) {...}

   public void addCarreteraEntranteAlCruce(String idCarretera, Carretera carretera) {
      T ri = creaCarreteraEntrante(carretera);
      ...
   }
   abstract protected T creaCarreteraEntrante(Carretera carretera);

   public void addCarreteraSalienteAlCruce(CruceGenerico<?> destino, Carretera carr) {...}

   public void entraVehiculoAlCruce(String idCarretera, Vehiculo vehiculo){...}

   @Override
   public void avanza()
    ...
    this.actualizaSemaforos();
   }

   abstract protected void actualizaSemaforos();
   ...
}
```

Cruces

```
public class Cruce extends CruceGenerico<CarreteraEntrante> {
   // cruces de la parte I
}

public class CruceCircular extends
CruceGenerico<CarreteraEntranteConIntervalo> {
    protected int minValorIntervalo;
    protected int maxValorIntervalo;

    ...
}

public class CruceCongestionado extends
CruceGenerico<CarreteraEntranteConIntervalo> {
   // no tiene atributos
   ...
}
```

CruceCongestionado

```
public class CruceCongestionado extends CruceGenerico<CarreteraEntranteConIntervalo> {
   // no tiene atributos
   ...
   @Override
   protected void actualizaSemaforos() {
   - Si no hay carretera con semáforo en verde (indiceSemaforoVerde == -1) entonces se
     selecciona la carretera que tenga más vehículos en su cola de vehículos.

   - Si hay carretera entrante "ri" con su semáforo en verde, (indiceSemaforoVerde !=
     -1) entonces:
        1. Si ha consumido su intervalo de tiempo en verde ("ri.tiempoConsumido()"):
           1.1. Se pone el semáforo de "ri" a rojo.
           1.2. Se inicializan los atributos de "ri".
           1.3. Se busca la posición "max" que ocupa la primera carretera entrante
                distinta de "ri" con el mayor número de vehículos en su cola.
           1.4. "indiceSemaforoVerde" se pone a "max".
           1.5. Se pone el semáforo de la carretera entrante en la posición "max" ("rj")
                a verde y se inicializan los atributos de "rj", entre ellos el
                "intervaloTiempo" a Math.max(rj.numVehiculosEnCola()/2,1).
   }
}
```

CruceCircular

```
public class CruceCongestionado extends CruceGenerico<CarreteraEntranteConIntervalo> {
   // no tiene atributos
   ...
   @Override
   protected void actualizaSemaforos() {
   - Si no hay carretera con semáforo en verde (indiceSemaforoVerde == -1) entonces se
     selecciona la primera carretera entrante (la de la posición 0) y se pone su
     semáforo en verde.

   - Si hay carretera entrante "ri" con su semáforo en verde, (indiceSemaforoVerde !=
     -1) entonces:
        1. Si ha consumido su intervalo de tiempo en verde ("ri.tiempoConsumido()"):
           1.1. Se pone el semáforo de "ri" a rojo.
           1.2. Si ha sido usada en todos los pasos ("ri.usoCompleto()"), se fija
                el intervalo de tiempo a ... Sino, si no ha sido usada
                ("!ri.usada()") se fija el intervalo de tiempo a ...
           1.3. Se coge como nueva carretera con semáforo a verde la inmediatamente
                Posterior a "ri".
   }
}
```