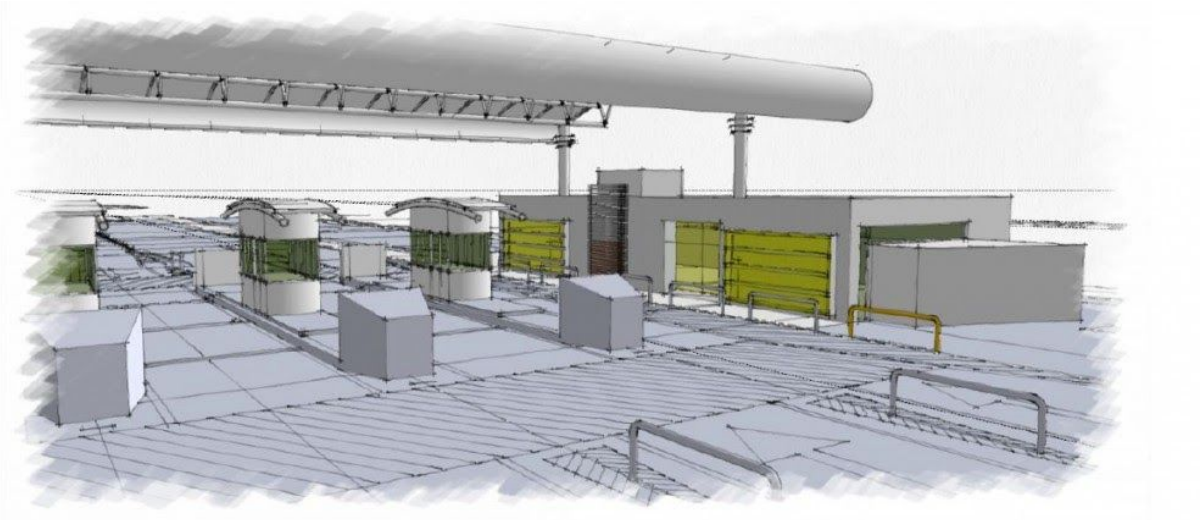
Práctica 7. **La Autopista**

1 OBJETIVO

Esta práctica pretende simular todo el ciclo de construcción de software:

- ❏ Requisitos. A partir de una descripción de los problemas del cliente, se refinan los requisitos que debe suplir el producto software.
- ❏ Análisis y Diseño. Se ofrecerá al alumno un análisis de la solución, así como parte de los “planos”.
- ❏ Construcción. El alumno deberá codificar los “planos”.
- ❏ Pruebas. El alumno deberá proponer un plan de pruebas para verificar y validar el software entregado.

2 ENUNCIADO



AUDENASA, Autopistas de Navarra Sociedad Anónima, es la concesionaria de las autopistas de la Comunidad Foral. Es la responsable de diseñar, construir, operar y mantener las autopistas navarras.

La normativa española fija en cinco el número de cabinas en cada sentido de un peaje troncal, distribuidas de la siguiente forma:

- Dos cabinas de tipo A. Requieren el pago con tarjeta de crédito. El tiempo de cobro se distribuye uniformemente entre 15 y 30 segundos.
- Dos cabinas de tipo B. Permiten el pago en metálico y con tarjeta de crédito. El tiempo de cobro se distribuye uniformemente entre 15 y 45 segundos.
- Una cabina de tipo C. Requiere el pago en metálico. El tiempo de cobro se distribuye uniformemente entre 30 y 60 segundos.

Al diseñar los peajes troncales AUDENASA debe indicar en el Estudio de Viabilidad algunas variables exigidas para suplir las necesidades de diferentes partícipes y actores:

- Las asociaciones de conductores desean reducir el tiempo de espera (incluido el tiempo de cobro), en el peaje para llegar a su destino lo antes posible.
- La concesionaria de los peajes quiere maximizar el número de coches atendidos en cada cabina para aumentar sus beneficios por cabina.
- Las directrices europeas obligan a estimar el número de coches máximo en cola y el tiempo de espera medio por cada cabina para acceder a los fondos de financiación.

AUDENASA desea dotarse de un simulador que permita predecir,en función del tráfico, el valor de las siguientes variables:

- El tiempo medio de espera por coche
- El número medio de coches atendidos en cada cabina
- El número total de coches servidos en el peaje

- El número de coches máximo en cola
- El tiempo de espera media por cada cabina

El tráfico de coches simulará la llegada al peaje según una distribución exponencial, con un tiempo medio entre llegadas de *Tc* segundos, siendo este un valor a fijar en la simulación con objeto de poder representar distintas situaciones de tráfico.

Cuando un coche llega al peaje se incorpora de forma aleatoria a una de las tres colas con menos coches, con una probabilidad de 0.6 (la más corta), 0.3 (la segunda más corta) y 0.1 (la tercera más corta).

AUDENASA desea simular el funcionamiento del peaje durante tres horas. Los resultados esperados de la simulación deben ser:

- Para un tráfico muy ligero (1 coche cada aproximadamente 100 segundos)
    - No se formarán colas (el tamaño máximo de la cola será de un coche).
    - El tiempo de espera en el peaje de cada coche, será aproximado al tiempo medio de cobro de la cabina.
    - Pocos coches llegarán a la cuarta cabina y probablemente ninguno pase por la quinta cabina.
- Para un tráfico intenso (1 coche cada 10 segundos):
    - Las posibles colas que se formen serán pequeñas.
    - El tiempo de espera en el peaje será algo mayor que en caso de tráfico muy ligero debido a la aparición ocasional de colas que provocarán el incremento del tiempo medio de espera.
- Para un tráfico muy intenso (1 coche cada segundo). El peaje colapsará:
    - Se formarán grandes colas en todas las cabinas.
    - El tiempo de espera en cada cabina será enorme.

En todos los casos es de esperar que las cabinas de tipo A sirvan más coches que las de tipo B. Y las cabinas de tipo B servirán más coches que la de tipo C.

3 REQUERIMIENTOS

- **REQ01**. Como asociación de conductores quiero que el tiempo medio de espera en el peaje se indique en los estudios de viabilidad para analizar si resulta aceptable.
- **REQ02**. Como concesionaria del peaje quiero que el número medio de coches atendidos en cada cabina y el número total de coches servidos en el peaje se indique en los estudios de viabilidad para ayudarme a estimar el retorno de inversión (ROI) que obtendré tras construir las cabinas.
- **REQ03**. Como Unión Europea quiero que el número de coches máximo en cola así como el tiempo de espera media por cada cabina consten en el estudio de viabilidad para ayudarme a evaluar si cumplen las directivas de tráfico europeas y así dar el visto bueno a la financiación.

4 ANÁLISIS Y DISEÑO

4.1 Análisis

Se entregará a AUDENASA una aplicación de consola. La aplicación de consola solicitará por pantalla el tiempo medio entre llegadas de coches en segundos. La aplicación mostrará el tiempo medio de espera por coche, el número medio de coches atendidos en cada cabina, el número total esperado de coches servidos en el peaje y el número de coches máximo en cola por cada cabina.

4.2 Diseño

Para facilitarle un poco la tarea, se han desarrollado algunos documentos de diseño, quizá no del todo correctos. En concreto, un diseñador inexperto a creado los siguientes diagramas UML para explicar el programa:

Diagrama de secuencia **Simular**:

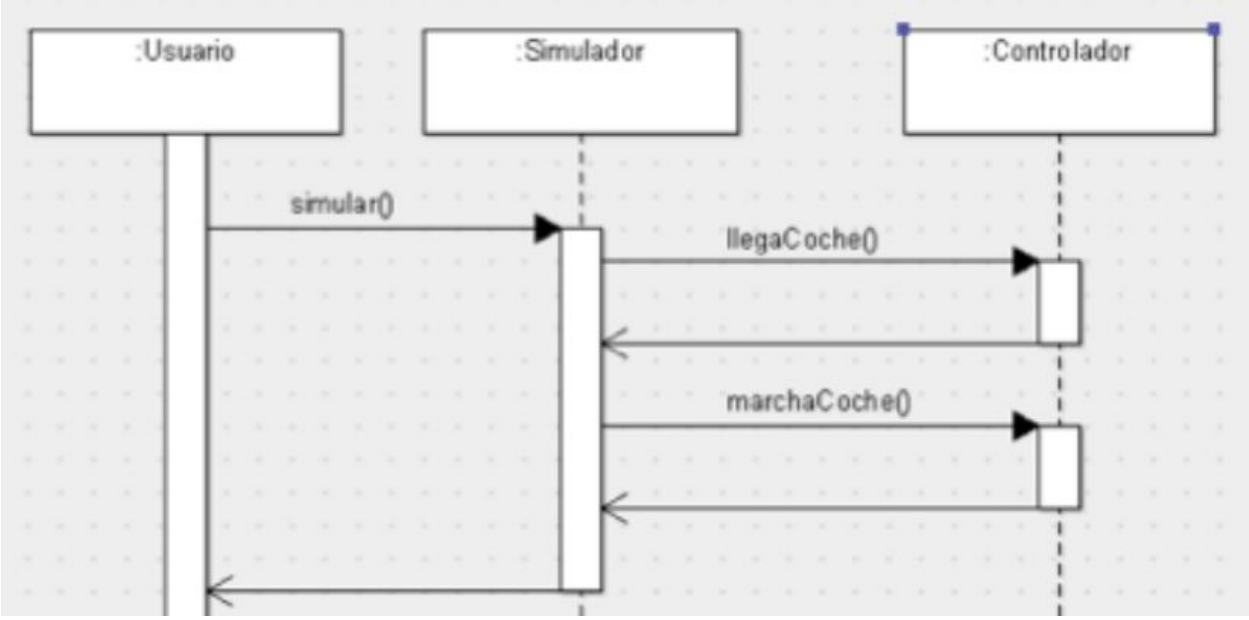


Diagrama de secuencia **Llega Coche**:

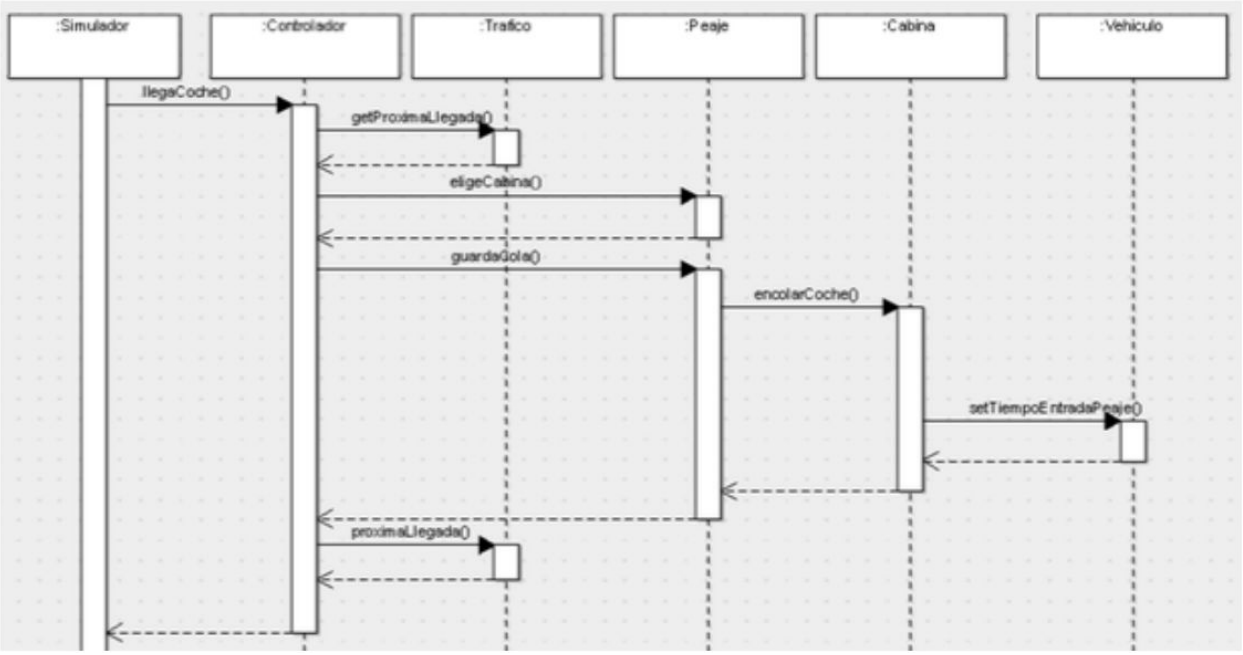


Diagrama de secuencia **Mostrar Resultados**

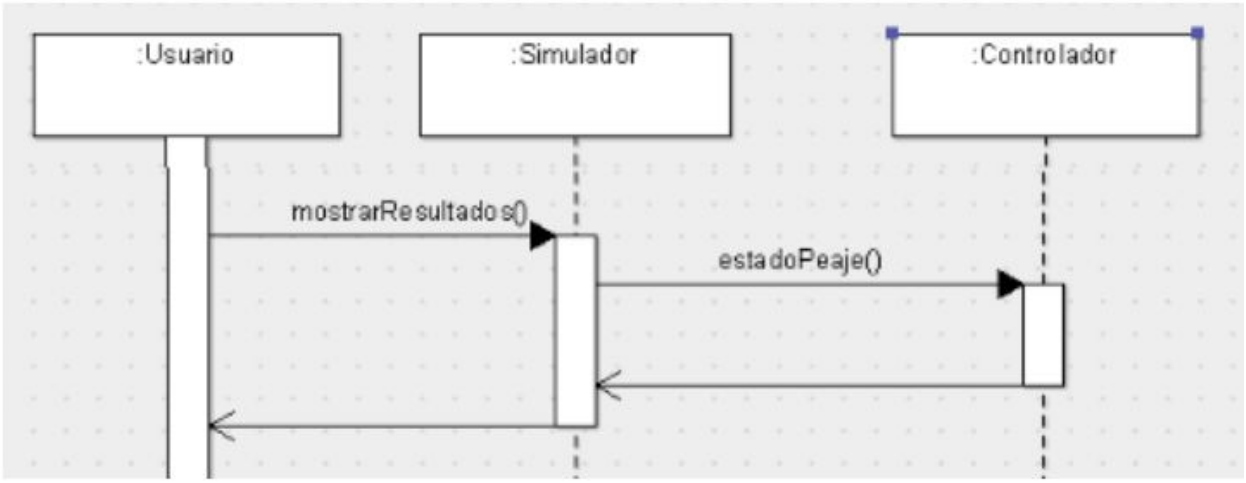
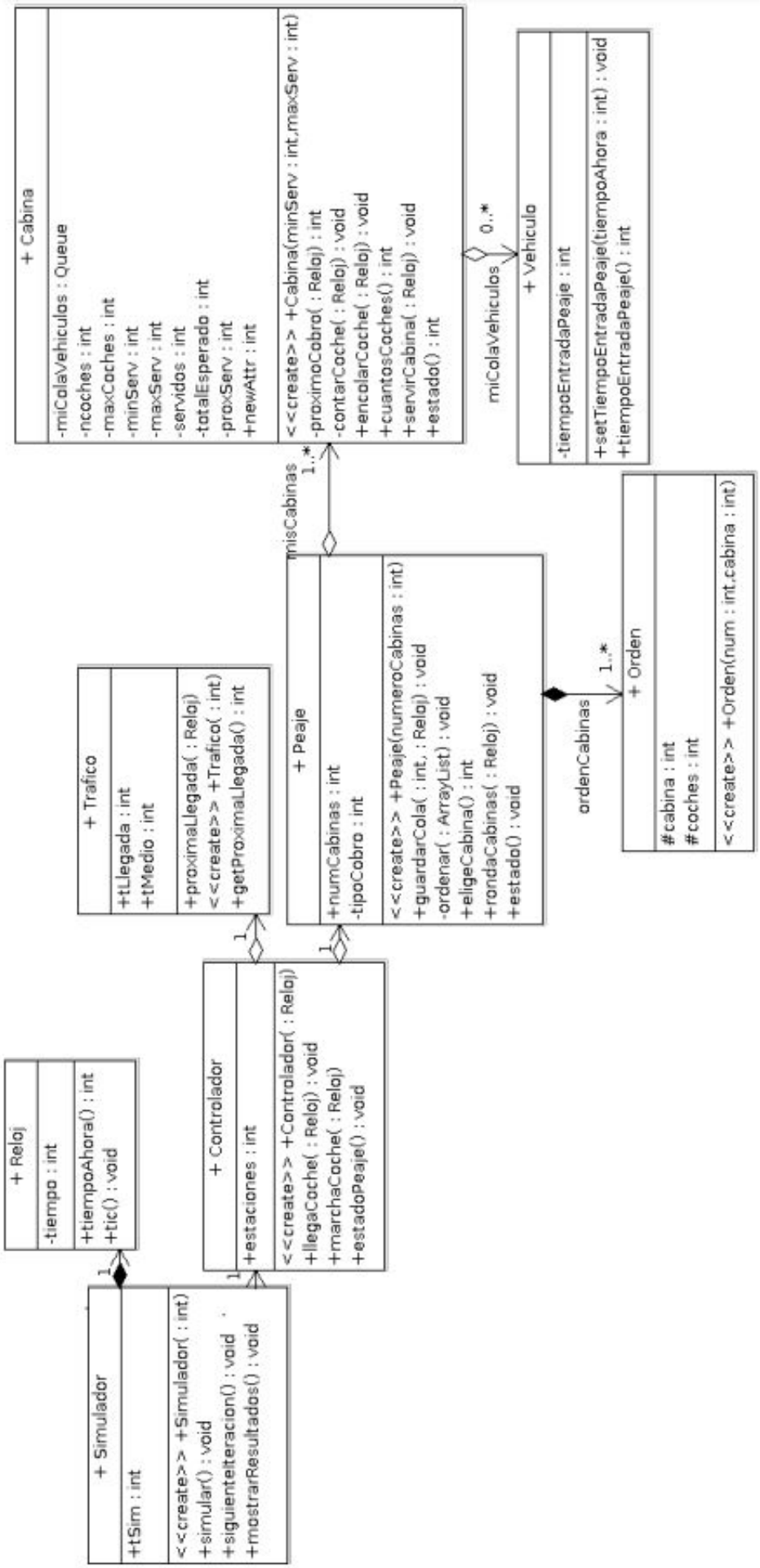


Diagrama de clases de la aplicación.

En base a los diagramas anteriores se ha construido un diagrama de clases que vamos a utilizar como guía para construir la aplicación.


+ Cabina
-miColaVehiculos : Queue
-ncoches : int
-maxCoches : int
-minServ : int
-maxServ : int
-servidos : int
-totalEsperado : int
-proxServ : int
+newAttr : int
<<create>> +Cabina(minServ : int,maxServ : int)
-proximoCobro(: Reloj) : int
-contarCoche(: Reloj) : void
+encolarCoche(: Reloj) : void
+cuantosCoches() : int
+servirCabina(: Reloj) : void
+estado() : int
miColaVehiculos
0..*
+ Vehiculo
-tiempoEntradaPeaje : int
+setTiempoEntradaPeaje(tiempoAhora : int) : void
+tiempoEntradaPeaje() : int
misCabinas
1..*
+ Peaje
+numCabinas : int
-tipoCobro : int
<<create>> +Peaje(numeroCabinas : int)
+guardarCola(: int, : Reloj) : void
-ordenar(: ArrayList) : void
+eligeCabina() : int
+rondaCabinas(: Reloj) : void
+estado() : void
ordenCabinas
1..*
+ Orden
#cabina : int
#coches : int
<<create>> +Orden(num : int,cabina : int)
+ Trafico
+tLlegada : int
+tMedio : int
+proximaLlegada(: Reloj)
<<create>> +Trafico(: int)
+getProximaLlegada() : int
1
1
+ Controlador
+estaciones : int
<<create>> +Controlador(: Reloj)
+llegaCoche(: Reloj) : void
+marchaCoche(: Reloj)
+estadoPeaje() : void
1
1
+ Reloj
-tiempo : int
+tiempoAhora() : int
+tic() : void
1
+ Simulador
+tSim : int
<<create>> +Simulador(: int)
+simular() : void
+siguienteIteracion() : void
+mostrarResultados() : void

5 DESARROLLO

Obviamente, para programar el diseño que le he proporcionado basta con que cree que las clases que aparecen en el diagrama de clases. En lo que sigue le voy a ofrecer algunas explicaciones adicionales sobre cada clase que pueden ser necesarias para poner todo esto en marcha. También le explicaré cómo hay que programar algunos aspectos de los que aparecen en los diagramas

La clase Simulador

Según el diagrama de clases esta clase contiene un atributo (el tiempo de simulación) entero. Pero el diagrama también refleja que desde ella parte una relación de composición hacia la clase reloj además de un uso hacia la clase controlador (yo creo que más bien debería ser composición, pero bueno…). Esto quiere decir que realmente la clase dispone de tres atributos. Uno de tipo del `Reloj` otro de tipo `Controlador` y el entero comentado.

El constructor de esta clase recibe un valor entero con el que inicializa el tiempo de simulación. Además de esto, instancia los objetos de las clases `Reloj` y `Controlador` que necesita. Además del constructor tiene tres métodos:

1. El público `simular()` que consiste en un bucle en el que mientras el tiempo del reloj sea menor que el de simulación se aumenta un segundo el tiempo del Reloj y se invoca el método `siguienteIteracion()`
2. El método privado `siguienteIteracion()` que se limita a invocar los métodos del controlador que indica el diagrama de secuencia, pasándole en ambos el objeto Reloj
3. El método público `mostrarResultados()` que se encarga de sacar los resultados de la simulación por pantalla

Esta es la clase ejecutora (*con main()*), pero es conveniente que el main no haga nada más que instanciar un Simulador e invocar sus métodos.

La clase Reloj

La clase Reloj contiene una única propiedad privada (`tiempo`). El constructor debe poner el `tiempo` a cero. Sus otros dos miembros son los métodos públicos `tiempoAhora()` que devuelve el valor del tiempo y `tic()` que aumenta en un segundo el tiempo.

La clase Controlador

Esta clase contiene dos relaciones de agregación a otras clases, que nuevamente se traducen en atributos. Además de estas propiedades públicas miPeaje y miTrafico, existe otro atributo final que es el número de cabinas de las que consta el peaje.

El constructor de la clase recibe un parámetro de tipo Reloj. Su código comienza instanciando un Peaje (pasándole el número de cabinas que debe tener). Tras ello se instancia un Scanner conectado con System.in y solicita al usuario el tiempo medio de llegada de vehículos. El tiempo que indica el usuario se utiliza como parámetro a continuación para crear un Tráfico. Tras instanciar el `miTrafico` se le consulta el tiempo de la próxima llegada (pasándole el reloj).

El resto de métodos de esta clase aparecen reflejados en los diagramas. Sólo falta por indicar que los métodos adicionales de la clase usan un parámetro de tipo Reloj y que la secuencia de eventos que según el diagrama se invocan en el método `llegaCoche()` sólo se ejecutan si la próxima llegada de la que informa el tráfico coincide con el tiempo que indica el reloj que recibe el método.

No olvide importar java.util.Scanner para implementar esta clase.

La clase Tráfico

Esta es una clase muy sencilla. Tiene dos propiedades: el tiempo medio de llegada y el tiempo de la próxima llegada (privado). El constructor se limita a poner como tiempo medio de llegada el tiempo que recibe a través de su parámetro.

Además del constructor la clase tiene dos métodos : `getProximaLlegada()` que devuelve el valor del parámetro privado y `proximaLlegada(Reloj r)` que calcula, teniendo en cuenta el reloj que recibe como parámetro, el tiempo de la próxima llegada. Para implementar la distribución que indica el enunciado vamos a utilizar la clase `Math` que contiene un conjunto de métodos estáticos para realizar cálculos complejos. En nuestro caso el próximo tiempo de llegada se calcula como:

```
(int) (r.tiempoAhora() + 1 - tmedio*Math.log(1 - Math.random()))
```

La clase Peaje

La clase Peaje es más compleja. Contiene cuatro propiedades. Una es un array constante: final `int [][] tipoCobro = {{15,30}, {15,30}, {15,45}, {30,60},{45,60}}`. Otra es un valor entero que contiene el número de cabinas. Los otros dos son privados `misCabinas` y `ordenCabinas`. Puede apreciar que, según el diagrama vienen de dos relaciones con otras clases. Las relaciones indican que son atributos de multiplicidad diferente a uno. Este tipo de relaciones se implementan en Java haciendo uso de clases de tipo `collection` de JDK. Estas clases implementan varios métodos para manejar las colecciones de forma muy sencilla. En nuestro caso vamos a utilizar la clase `ArrayList` que está en `java.util.ArrayList`.

Un `ArrayList` es un contenedor de tamaño modificable que contiene elementos iguales. Tiene la ventaja de que está programado haciendo uso de `Generics`, por lo que permite que incluyamos el tipo de objeto que queramos. Vamos a aprovechar esa circunstancia para declarar un `ArrayList` de Cabinas y otro de Orden (el primero sería `private ArrayList myCabinas;`).

Le paso el código del constructor para que vea cómo se usa el ArrayList

```
public Peaje(int estaciones)
{
        misCabinas = new ArrayList<Cabina>();
        this.numCabinas = estaciones;
        for (int i = 0; i < numCabinas; i++)
                myCabinas.add(new Cabina(tipoCobro[i][0], tipoCobro[i][1]));
}
```

El método `guardarCola(int nCab, Reloj myReloj)` recibe como parámetro la cabina en la que se guarda el nuevo coche y el reloj. Como indica el diagrama de secuencia invoca un método de la cabina, pero, previamente debe obtener del ArrayList la cabina con la que debe comunicarse. Para ello se usa el método `get()` del ArrayList.

El método `eligeCabina()` comienza instanciando el objeto `ordenCabinas` de tipo ArrayList<Orden>. Posteriormente lo llena de objetos de tipo Orden capturando de cada cabina del ArrayList de cabinas cuantos coches almacena (invocación que falta del diagrama de secuencias) mediante el método `cuantosCoches()`. Posteriormente llama a un método privado que ordena el ArrayList `ordenCabinas` en función de los coches de cada cola y elige una de las tres mejores cabinas mediante el código:

```
final double [] elec = {0.6, 0.9, 1.0};
double x = Math.random();
int n = 0;
while ((elec[n]<x  && (n<2)){
        n++;
}
return ordenCabinas.get(n).cabina;
```

¿lo entiende?

El método `rondaCabinas()` como su nombre indica, recorre el ArrayList de cabinas invocando el método `servCabina()` de cada cabina (pasándole el reloj, obviamente).

El método privado de ordenación y el que se emplea para sacar los resultados por pantalla los dejo en sus manos. Tenga en cuenta que para la ordenación debe jugar con el ArrayList<Orden>

El metodo *estado()* saca los resultados por pantalla

La clase Orden

Un Orden almacena simultáneamente el número de una cabina con el número de coches de su cola. Al constructor se le pasan los dos enteros que van a cargar las dos propiedades.

La clase Cabina

La claseCabina es larga pero tiene menos problemas. Tiene un conjunto de propiedades privadas de tipo entero para almacenar el número de coches de la cola, el máximo número de coches que ha tenido la cabina, los tiempos máximo y mínimo de servicio, el total de coches servido, el tiempo total esperado y el tiempo del próximo servicio y una propiedad privada `miColaVehiculos` de tipo `Queue`. `Queue<>` es un interface (utiliza generics) de la Java Collection Framework. Define métodos típicos de las colas. Habrá comprobado que hay dos formas de insertar, examinar y eliminar elementos de la cola. Esto es porque una de las formas de trabajar es mediante el lanzamiento de excepciones cuando se violen los requisitos, pero también es posible devolver un valor especial para indicar tal circunstancia. Puede emplear lo que quiera. El constructor de la clase, recibe dos valores enteros como parámetro (los tiempos los tiempos máximo y mínimo de servicio). Instancia la cola e inicializa todos los valores enteros que se ha comentado. Para crear la cola instancie alguna clase que implemente el interface Queue, por ejemplo LinkedList: `miColaVehiculos = new LinkedList<Vehiculo>();`

La clase implementa diversos métodos adicionales:

- El método privado `proximoCobro(Reloj r)` que calcula el tiempo del próximo cobro mediante la fórmula `(int)(((maxServ-minServ)*Math.random())+ minServ + r.tiempoAhora());` y devuelve el valor obtenido.
- El método privado `contarCoche(Reloj r)` que si no hay coches en la cola programa el próximo servicio `this.proxServ = proximoCobro(r);` contabiliza el nuevo coche y si se superó el máximo modifica dicho valor.
- El método público `encolarCoche(Reloj r)` comienza contando el nuevo coche (invocando al método anteriormente comentado), instancia un nuevo `Vehiculo`

indicando su tiempo de llegada a la cola y mete el nuevo vehículo en la cola de vehículos.

- El método público `servirCabina(Reloj r)`, si `tiempoAhora` coincide con `proxServ`, aumenta el número de vehículos servidos, obtiene el primer vehículo de la cola (y la hace avanzar), actualiza el tiempo total esperado, modifica la información sobre el número de coches de la cola y si dicho número no es cero programa el tiempo del próximo servicio (si el número de coches en cola es cero, entonces el tiempo de próximo servicio será cero).
- Además pueden existir varios métodos para leer los valores de las propiedades privadas

La clase Vehiculo

Para que vea que soy bueno, esta se la doy:

```
class Vehiculo {

    private int tiempoEntradaPeaje;

    public void setTiempoEntradaPeaje(int tiempoAhora) {

         tiempoEntradaPeaje = tiempoAhora;

    }

    public int tiempoEntradaPeaje() {

          return tiempoEntradaPeaje;

    }

}
```

6 PRUEBAS

Redacte un pequeño plan de pruebas y certifique que el simulador entregado lo cumple. Para ello recoga los valores de tres simulaciones cumplimientando esta tabla y valore si son correctos.

Tiempo de llegada		1	10	100
Cabina 1	Máximo de coches en cola			
	Coches Servidos			
	Tiempo de espera medio			
Cabina 2	Máximo de coches en cola			
	Coches Servidos			
	Tiempo de espera medio			
Cabina 3	Máximo de coches en cola			
	Coches Servidos			
	Tiempo de espera medio			
Cabina 4	Máximo de coches en cola			
	Coches Servidos			
	Tiempo de espera medio			
Cabina 5	Máximo de coches en cola			
	Coches Servidos			
	Tiempo de espera medio			
Totales Servidos				