Programación concurrente y distribuida

Boletín de practicas

Convocatoria de Julio 2013/2014

Documentación de la practica

Folea Ilie Cristian
Grupo 3
2º Grado Ingeniería Informática

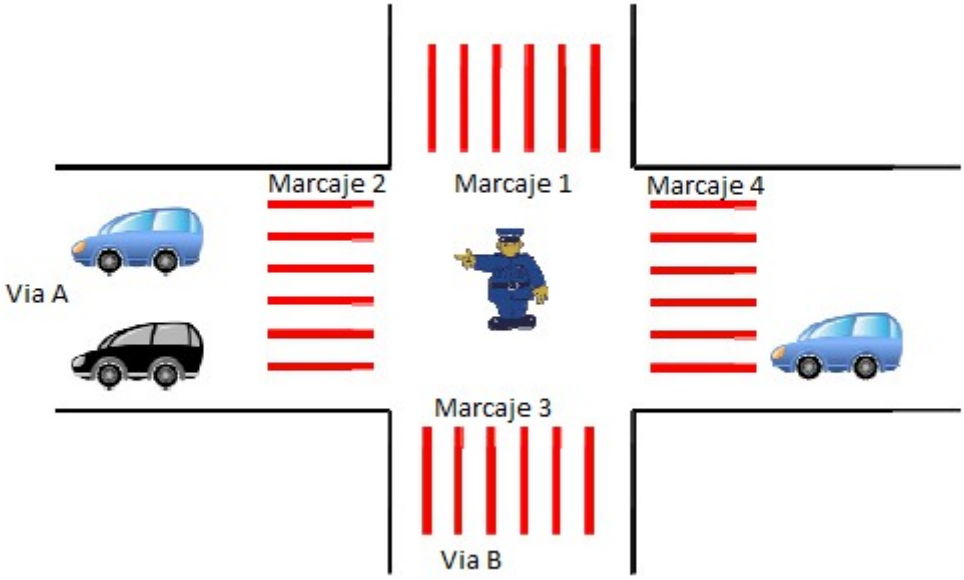
Profesor:
Juan Antonio Sánchez Laguna

Índice

1. Introducción

La practica propuesta por los profesores de la asignatura consiste en la simulación de un cruce con dos vías y 4 pasos de peatones. Tanto el paso de peatones como el de los coches por dicho cruce sera controlado por un guarida. Se considera una vía A y una vía B. Los peatones de los marcajes 1 y 3 pueden cruzar la via solo cuando los coches pasan por la vía A y los peatones que cruzan por los marcajes 2 y 3 pueden cruzar la vía solo cuando los coches cruzan por la vía B.

El problema propuesto consiste en sincronizar el paso de peatones y los coches, para que los peatones no crucen la vía activa mientras hay peatones cruzando y que el cambio de vía no se realice mientras hay peatones o coches cruzando. Para evitar que al llegar muchos coches por una vía los coches que esperan por la otra vía no consigan pasar nunca, se considera que cuando llega un coche por la vía inactiva solo se permiten pasar a 3 coches por la vía activa hasta que se produzca el cambio de vía.



2.Diseño de la solución

Para el diseño de la solución se han considerado once peatones que se reparten de la siguiente manera en los marcajes: tres peatones cruzan el marcaje uno, dos peatones cruzan el marcaje dos, cuatro peatones cruzan el marcaje tres y dos peatones cruzan el marcaje cuatro. Los peatones indicados cruzan el marcaje, se esperan un tiempo determinados y vuelven a cruzar el marcaje. También se han considerado diez coches, cinco para cada vía.

Dicho lo anterior y teniendo cuenta que cada coche y peatón es representado por un hilo podemos observar que tenemos veintiuno hilos de dos tipos distintos. Un tipo de hilo es para representar los peatones y otro tipo de hilo para representar los coches. A los veintiuno hilo se le suma el hilo que representa el guardia y en total tenemos veintidós hilo.

El funcionamiento del hilo que representa al **peatón** es el siguiente:
El peatón imprime por pantalla que quiere cruzar la calle indicando su identificador y el marcaje por el que quiere pasar. La impresión de pantalla se gestiona mediante un monitor para que solo un

peatón, coche, o guardia puedan imprimir por pantalla al mismo tiempo. En el caso de que ya haya alguien imprimiendo, el peatón pasa a la cola de espera de impresión y se bloquea. Después de que haber imprimido por pantalla su intención de cruzar la calle, el peatón pide acceso para cruzar la calle. La petición de acceso se hace utilizando paso de mensajes sincrono con canales. Al utilizar este mecanismo tanto la operación de envió como la de recepción son bloqueantes. Una vez que haya enviado por el canal su intención de cruzar la calle se queda esperando a que el proceso controlador (Guardia), en el caso de que la vía activa permite que dicho peatón cruce la calle, le permita cruzar la calle. Una vez que se le haya permitido cruzar la calle, imprime por pantalla que ya este cruzando la calle y después se le hace un sleep con el tiempo que tarda en cruzar la calle. Una vez que ha cruzado la calle, el peatón indica al guardia, que ya ha cruzado la calle y que hay un peatón menos cruzando. Para que el peatón indique al guardia que ya ha cruzado, se utiliza un semáforo, ya que el guardia tiene una variable global que guarda el numero de peatones que están cruzando, y en el caso de que no se accedería a dicha variable en exclusión mutua el guardia podría tener un numero erróneo de los peatones que están cruzando y afectaría al cambio de vía.

El funcionamiento del hilo que representa a un **coche** es el siguiente:
El coche comprueba cual es la vía por la que debe pasar. Una vez que sabe la vía que va a utilizar para pasar, imprime por pantalla su intención, utilizando el mismo mecanismo que en el caso del peatón, y pide acceso al guardia mediante el mecanismo de invocación remota. En dicho mecanismo tanto el emisor como el receptor quedan bloqueados hasta que se produce el encuentro pero, a diferencia del mecanismo de paso de mensajes utilizando canales, el emisor debe conocer la identidad del receptor. Una vez que se le acepta la petición del cruce de la vía imprime por pantalla que ya va a cruzar la vía y también se lo indica al guardia para que sea consciente de ello y sepa el numero de coche que están cruzando. Al entrar en la vía se hace un sleep del tiempo que el coche tarda en cruzar dicha vía. Después de que haya cruzado la vía, imprime por pantalla que ya ha cruzado la vía y le indica al guardia mediante el mismo mecanismo que ya ha cruzado la vía para que este sea consciente y actualice el numero de coches que están cruzando la vía.

El funcionamiento del hilo que representa a un **guardia** es el siguiente:
El guardia tiene varias funciones. Hace de controlador para los mecanismos de paso de mensajes. Al ejecutar el hilo, el guardia selecciona una vía aleatoria entre A y B y espera que reciba peticiones de los demás hilos. En el caso de que la vía activa sea la A el guardia recibe solo por los canales correspondientes a los peatones que pasan por los marcajes uno y tres, caso contrario recibe por los canales correspondientes a los peatones que pasan por los marcajes dos y cuatro. En el caso de que le llega un petición por un hilo del coche la analiza. En el caso de que le llega una petición de un coche por una vía y la via activa es la contraria a la que quiere pasar el coche, el guardia impide a que entren nuevos peatones a cruzar y también indica que hay coches esperando para que no se les permita a mas de tres coches, desde ese momento, a cruzar la vía. Esto últimos e hace para que permita una cierta igualdad entre los coches que pasan por una vía y los coches que pasar por la via contraria, porque si no, si por una vía pasan muchos coches, los de la vía contraria podrían estar esperando un periodo largo de tiempo o puede ser que nunca se les permita pasar. En el caso de que la vía de la que se hace la petición es la vía activa, se le indica de que es posible que pase. También se analizan las indicaciones de que ya se ha empezado a cruzar la vía y de que se ha dejado de cruzar la vía, para que se actualicen las variable de coches cruzando. El guardia también controla el numero de peatones que están cruzando o que han dejado de cruzar, para el caso de que los peatones entren a cruzar, el guardia actualiza la variable de los peatones cruzando. Para el caso de que los peatones dejen de cruzar, el guardia tiene un semáforo mediante el cual garantiza que cada peatón entra en exclusión mutua a actualizar la variable de peatones cruzando. También tiene un monitor

del que hacen uso los tres tipos de hilos para imprimir por pantalla.

3.Recursos no compartibles

- **Pantalla:** La impresión de pantalla se debe hacer en exclusión mutua para evitar que varios procesos escriban al mismo tiempo y los datos mostrados no tengan coherencia.
- **Peatones cruzando:** La actualización del numero de peatones cruzando se debe hacer en exclusión mutua ya que no podemos permitir que varios peatones entren simultáneamente a actualizar dicha variable debido a que al actualizar, los peatones pueden coger la variable al mismo tiempo con un determinado valor y a la hora de actualizarlo en vez de decrementarla dos veces, la debe decrementar solo una ya que los dos han actualizado el mismo numero y no el numero de peatones cruzando que el primero ha actualizado.

4. Condiciones de sincronización

La **primera** condición de sincronización que encontramos es entre los peatones y el guardia. Los peatones deben pedir permiso al guardia para que puedan cruzar el marcaje ya que puede ser que en ese momento la vía activa no es la correspondiente y pasen coches por donde los peatones quieren cruzar. El guardia analiza la petición de los peatones para ver por donde quieren cruzar y si pueden cruzar por ese marcaje en dicho momento o no. En el caso de que sea posible cruzar el marcaje el guardia se lo permite. Esta sincronización se realiza utilizando el mecanismo de paso de mensajes síncrono usando canales.

La **segunda** condición de sincronización que encontramos es entre los coches y el guardia. Los coches no deben poder pasar si hay coches pasando por la otra vía. Cuando un coche quiere pasar por una vía el guardia analiza su petición y en el caso de que la vía activa sea la vía por la que el coche ha pedido cruzar se lo permite, siempre y cuando no se ha llegado al limite de los coches que pueden cruzar por esta vía. Como se ha indicado anteriormente existe un limite de coches que pueden cruzar por una vía. En la solución que se propone, dicho limite es de tres coches y solo se activa el limite cuando llega un coche por la vía contraria a la activa y este esté esperando, entonces el guardia solo deja pasar tres coches mas de los que están pasando, una vez que han pasado esos tres coches, o si la vía se ha liberado antes, el guardia procede a hacer un cambio de vía para que los coches de la otra vía también puedan cruzar. Si no hay ningún coche esperando a cruzar por la otra vía la vía activa sigue siendo la misma hasta que llegue un coche por la vía contraria. Esta sincronización se realiza utilizando el mecanismo de sincronización por invocación remota.

La **tercera** condición de sincronización que encontramos es entre los peatones y el guardia para que el guardia sepa cuando los peatones ya han cruzado. Esta sincronización se resuelve mediante semáforos y es necesaria para que los peatones le comuniquen al guardia cuando ya han cruzado el marcaje ya que el guardia contiene una variable global que representa el numero de peatones cruzando y si no se acede en exclusión mutua se pueden producir errores a la hora de actualizar dicha variable.

La **cuarta** condición de sincronización que encontramos es entre peatones, coches y guardia a la hora de imprimir por pantalla. Esta sincronización se resuelve mediante el uso de un monitor y es

necesario para que los procesos impriman por pantalla uno por uno y no todos al mismo tiempo.

5. Secciones criticas

Como secciones criticas nos encontramos las siguientes:

1. La primera sección critica nos la encontramos en el monitor a la hora de gestionar el acceso a la pantalla. La pantalla es un recurso no compratible y por tanto se debe gestionar el acceso a la misma. La gestión de este recurso se hace mediante un monitor. Todos los procesos que quieren utilizar la pantalla deben hacer uso del monitor, en el caso de que el monitor este ocupado pasa a la cola de este.

2. La segunda sección critica nos la encontramos a la hora de gestionar el numero de peatones que están cruzando. Esta variable es una variable no compartible ya que cada peaton debe decrementarla a la hora de pasar el cruce y en el caso de que dos peatones pasen el cruce al mismo tiempo se puede dar el caso de que la variable se decremente mal. Para la gestión de dicha variable se utiliza un semáforo binario que permite decrementar el numero o no.

6.Pseudocodigo

Monitor Pantalla

var

escribiendoPantalla : boolean;

escribirPantalla : Condition

export entrar_imprimirPantalla, salir_imprimirPantalla

procedure entrar_imprimirPantalla

begin

if(!escribiendoPantalla)

then

delay(escribirPantalla)

end;

escribiendoPantalla = true;

end;

procedure salir_imprimirPantalla

begin

escribiendoPantalla = false;

resume(escribirPantalla);

end;

```
process salir_pasar_cruce
initial(mutex, 1);
begin
        wait(mutex);
end
        // Seccion critica, peatonesCruzando--;
begin
        signal(mutex);
end;
process Peaton (var id: Integer; via : boolean)
while(true){
        send(canal[id], via);
        sleep(200 milisegundos);
        salir_pasar_cruce();
}

process Coches (var id: Integer; via: String);
while(true){
        if(via == A){
                guardia.peticionEntradaA
                guardia.entrarA;
                sleep(100 milisegundos);
                guardia.salirA;
        } else if(via == B){
                guardia.peticionEntradaB
                guardia.entrarB;
                sleep(100 milisegundos);
                guardia.salirB;
        }
}


process Guardia()
int C = 3;
String viaActiva = A;
```

```
int cochesRestantes = 0;
boolean pasarPeatones = true;
int cochesCruzando = 0;
boolean cochesEsperando = false;
int peatonesPasando = 0;
while (true){
        select{
                when viaActiva == A && pasarPeatones =>
                        recive(canal[0], msj);
                        peatonesPasando++;
                or when viaActiva == A && pasarPeatones  =>
                        recive(canal[1], msj);
                        peatonesPasando++;
                or when viaActiva == A && pasarPeaton =>
                        recive(canal[2], msj);
                        peatonesPasando++;
                or when viaActiva == B && pasarPeaton =>
                        recive(canal[3], msj);
                        peatonesPasando++;
                or when viaActiva == B && pasarPeaton =>
                        recive(canal[4], msj);
                        peatonesPasando++;
                or when viaActiva == A && pasarPeaton =>
                        recive(canal[5], msj);
                        peatonesPasando++;
                or when viaActiva == A && pasarPeaton =>
                        recive(canal[6], msj);
                        peatonesPasando++;
                or when viaActiva == A && pasarPeaton =>
                        recive(canal[7], msj);
                        peatonesPasando++;
                or when viaActiva == A && pasarPeaton =>
                        recive(canal[8], msj);
                        peatonesPasando++;
                or when viaActiva == B && pasarPeaton =>
```

```
                recive(canal[9], msj);
                peatonesPasando++;
        or when viaActiva ==  B && pasarPeaton =>
                recive(canal[10], msj);
                peatonesPasando++;
        or
                peticionEntradaA.accept(){
                        cochesEsperando = true;
                        pasarPeatones = false;
                }
        or
                peticionEntradaB.accept(){
                        cochesEsperando = true;
                        pasarPeatones = false;
                }
        or when viaActiva == A && cochesCruzando < C =>
                entrarA.accept(){
                        if(cochesEsperando){
                                cochesRestantes++;
                        }
                        cochesCruzando++;
                }
        or when viaActiva == A && cochesCruzando < C =>
                entrarB.accept(){
                        if(cochesEsperando){
                                cochesRestantes++;
                        }
                        cochesCruzando++;
                }
        or
                salirA.accept(){
                        cochesCruzando--;
                        if(cochesEsperando && cochesCruzando == 0 &&
peatonesCruzando==0){
                        cambiarVia();
```

```
                }
                }
        or
                salirB.accept(){
                        cochesCruzando--;
                        if(cochesEsperando && cochesCruzando == 0 &&
    peatonesCruzando==0){
                        cambiarVia();
                }
                }
    }
}
```

6.Codigo

Guardia

```
public class Guardia extends RemoteServer implements Runnable{
	private Via viaActiva; // Para indicar la via activa
	private Selector s = null; // Para el select
	private final static int C = 3; // numero maximo de coches que pueden
pasar una vez que ha llegado un coche por la via contraria.
	private int cochesRestantes; // numero de coches que han cruzado la via
una vez que han llegado coches por la via que no esta activa.
	private boolean cochesEsperando; // boolean para indicar que hay coches
del otro carril esperando un cambio de via
	private boolean pasarPeatones; // boolean para permitir o no el paso de
los peatones.
	private static int peatonesCruzando; // controlar el nº de peatones
cruzando
	private Channel peatones[]; // canales de los peatones
	private int cochesCruzando; // numero de coches que estan cruzando
actualmente.
	private EntryPoint peticionEntrarA, peticionEntrarB, entrarA, entrarB,
salirA, salirB;
	public static Semaphore mutex = new Semaphore(1); // Semaforo para
gestionar el cruce de los marcajes
	public static ReentrantLock monitor = new ReentrantLock(); //Monitor para
gestionar la impresión en pantalla
	public static Condition escribirPantalla = monitor.newCondition();
//Condicion del monitor
	public static boolean escribiendoPantalla; //indica si se esta escribiendo
o no en la pantalla

	/**
	 * Constructor de la clase guardia.
	 * @param peatones Los canales por los que se va a recibir la petición de
los peatones.
	 */
	public Guardia(Channel peatones[]){
		viaActiva = Via.viaAleatoria();
```

```
		cochesEsperando = false;
		pasarPeatones = true;
		peatonesCruzando = 0;
		escribiendoPantalla = false;

		peticionEntrarA = new EntryPoint("peticionEntrarA");
		peticionEntrarB = new EntryPoint("peticionEntrarB");
		entrarA = new EntryPoint("entrarA");
		entrarB =  new EntryPoint("entrarB");
		salirA = new EntryPoint("salirA");
		salirB = new EntryPoint("salirB");

		registerEntryPoint(peticionEntrarA);
		registerEntryPoint(peticionEntrarB);
		registerEntryPoint(entrarA);
		registerEntryPoint(entrarB);
		registerEntryPoint(salirA);
		registerEntryPoint(salirB);
		this.peatones = peatones;

		try{
			imprimirViaActiva();
		}catch (InterruptedException e) {
		}

	}

	/**
	 * Metodo que, utilizando la exclusion mutua mediante monitores, imprime
en panntalla el cambio de via.
	 * @throws InterruptedException En el caso de que no se puede utilizar el
monitor devuelve la excepción.
	 */
	private void imprimirCambioVia() throws InterruptedException{
		Guardia.monitor.lock();
		try{
			if(Guardia.escribiendoPantalla){
				Guardia.escribirPantalla.await();
			}
			Guardia.escribiendoPantalla = true;
			System.out.println("El numero de coches cruzando al cambiar la
via es " + cochesCruzando);
			System.out.println("Cambio de vía, la via activa es " +
viaActiva);
			Guardia.escribiendoPantalla = false;
			Guardia.escribirPantalla.signal();
			}finally{
				Guardia.monitor.unlock();
			}
	}

	/**
	 * Metodo que, utilizando exclusion mutua mediante monitores, imprime en
pantalla la via activa.
	 * @throws InterruptedException En el caso de que no se puede utilizar el
monitor devuelve la excepción.
	 */
	private void imprimirViaActiva() throws InterruptedException{
		Guardia.monitor.lock();
		try{
			if(Guardia.escribiendoPantalla){
```

```
                        Guardia.escribirPantalla.await();
                  }
                  Guardia.escribiendoPantalla = true;
                  System.out.println("La via activa es " + viaActiva);
                  Guardia.escribiendoPantalla = false;
                  Guardia.escribirPantalla.signal();
                  }finally{
                        Guardia.monitor.unlock();
                  }
      }

      /**
       * Metodo que, utilizando exlcusion mutua mediante monitores, imprime en
pantalla el numero de peatones que
       * hay cruzando en los marcajes que no estan en la via activa.
       * @throws InterruptedException En el caso de que no se puede utilizar el
monitor devuelve la excepción.
       */
      private static void imprimirPeatonesCruzando() throws
InterruptedException{
            Guardia.monitor.lock();
            try{
                  if(Guardia.escribiendoPantalla){
                        Guardia.escribirPantalla.await();
                  }
                  Guardia.escribiendoPantalla = true;
                  System.out.println("El numero de peatones que estan cruzando
alguno de los dos marcajes es " + peatonesCruzando);
                  Guardia.escribiendoPantalla = false;
                  Guardia.escribirPantalla.signal();
                  }finally{
                        Guardia.monitor.unlock();
                  }
      }

      /**
       * Metodo que se utiliza para cambiar la via.
       */
      private void cambioVia(){
            if(viaActiva == Via.A) {
                  viaActiva = Via.B;
                  cochesEsperando = false;
                  cochesRestantes = 0;
                  pasarPeatones = true;
            }
            else {
                  viaActiva = Via.A;
                  cochesEsperando = false;
                  cochesRestantes = 0;
                  pasarPeatones = true;
            }
            try{
                  imprimirCambioVia();
            } catch (InterruptedException e){
            }
      }

      /**
       * Metodo que se utiliza para decrementar el numero de peatones que estan
cruzando. Es un metodo estatico par que lo puedan utilizar
       * clases ajenas sin tener ninguna instancia de esta clase.
```

```
	 */
	public static void decrementarPeatonesCruzando(){
		peatonesCruzando--;
		try{
			imprimirPeatonesCruzando();
		} catch (InterruptedException e){
		}
	}

	@Override
	public void run() {
		try{
			Thread.sleep(500);
		} catch (InterruptedException e) {
			e.printStackTrace();
		}
		// Creamos el selector y añadimos las posibles selecciones.
		s = new Selector();
		for(int i = 0; i < peatones.length; i++){
			s.addSelectable(peatones[i], false);
		}
		s.addSelectable(peticionEntrarA, false);
		s.addSelectable(peticionEntrarB, false);
		s.addSelectable(entrarA, false);
		s.addSelectable(entrarB, false);
		s.addSelectable(salirA, false);
		s.addSelectable(salirB, false);

		while(true){
		// Añadimos las condiciones de seleccion
			for(int i = 0; i < peatones.length; i++){
				if(i < 3){
					peatones[i].setGuardValue(viaActiva == Via.A &&
pasarPeatones);
				}
				else if(i > 2 && i < 5){
					peatones[i].setGuardValue(viaActiva == Via.B &&
pasarPeatones);
				}
				else if(i > 4 && i < 9){
					peatones[i].setGuardValue(viaActiva == Via.A &&
pasarPeatones);
				}
				else if(i > 8){
					peatones[i].setGuardValue(viaActiva == Via.B &&
pasarPeatones);
				}
			}
			entrarA.setGuardValue(viaActiva == Via.A && cochesRestantes <
C);
			entrarB.setGuardValue(viaActiva == Via.B && cochesRestantes <
C);

			int select = s.selectOrBlock();

			if(select < 12){
				peatones[select-1].receive();
				peatonesCruzando++;
			}

			else{
```

```
                        switch(s.selectOrBlock()){
                              case 12: // peticion entrar A
                                    peticionEntrarA.accept();
                                    if(viaActiva != Via.A){
                                          cochesEsperando = true;
                                          pasarPeatones = false; // una vez que
hay coches esperando por la otra vía no se deja pasar mas peatones
                                    }
                                    peticionEntrarA.replay(null);
                                    break;
                              case 13: // petición entrar B
                                    peticionEntrarB.accept();
                                    if(viaActiva != Via.B){
                                          cochesEsperando = true;
                                          pasarPeatones = false;
                                    }
                                    peticionEntrarB.replay(null);
                                    break;
                              case 14: // entrar A
                                    entrarA.accept();
                                    if(cochesEsperando){
                                          cochesRestantes++;
                                    }
                                    cochesCruzando++;
                                    entrarA.replay(null);
                                    break;
                              case 15: // entrar B
                                    entrarB.accept();
                                    if(cochesEsperando){
                                          cochesRestantes++;
                                    }
                                    cochesCruzando++;
                                    entrarB.replay(null);
                                    break;
                              case 16: // salir A
                                    salirA.accept();
                                    cochesCruzando--;
                                    if(cochesEsperando && cochesCruzando == 0 &&
peatonesCruzando == 0){
                                          cambioVia();
                                    }
                                    salirA.replay(null);
                                    break;
                              case 17: // salir B
                                    salirB.accept();
                                    cochesCruzando--;
                                    if(cochesEsperando && cochesCruzando == 0 &&
peatonesCruzando == 0){
                                          cambioVia();
                                    }
                                    salirB.replay(null);
                                    break;
                        }
                  }
            }
      }
}
```

Coche

```
public class Coche implements Runnable {
	private final Guardia guardia;
	private final Via via;
	private static int numeroCoches = 0;
	private int id;

	/**
	 * Constructor de la clase coche.
	 * @param guardia Se le indica el guardia(controlador).
	 * @param via Se le indica la via por la que va a pasar.
	 */
	public Coche(Guardia guardia, Via via){
		this.guardia = guardia;
		this.via = via;
		numeroCoches++;
		id = numeroCoches;
	}

	/**
	 * Metodo que se utiliza para imprimir por pantalla la petición para el
pase de un coche por una via. Esta impresión se hace en exlcusión
	 * mutua utilizando monitores.
	 * @throws InterruptedException En el caso de que no se puede utilizar el
monitor devuelve la excepción.
	 */
	private void imprimirPeticionEntrada() throws InterruptedException {
		Guardia.monitor.lock();
		try{
			if(Guardia.escribiendoPantalla){
				Guardia.escribirPantalla.await();
			}
				Guardia.escribiendoPantalla = true;
				System.out.println("El coche numero " + id + " esta
pidiendo acceso a la vía " + via);
				Guardia.escribiendoPantalla = false;
				Guardia.escribirPantalla.signal();
			}finally{
				Guardia.monitor.unlock();
			}
	}

	/**
	 * Este metodo se utiliza para imprimir por la pantalla que el coche ha
recibido acceso para cruzar la via y empieza a cruzarla.
	 * Debe imprimirlo en exclusión mutua y lo hace utilizando monitores.
	 * @throws InterruptedException En el caso de que no se puede utilizar el
monitor devuelve la excepción.
```

```
	 */
	private void imprimirAccesoVia() throws InterruptedException {
		Guardia.monitor.lock();
		try{
			if(Guardia.escribiendoPantalla){
				Guardia.escribirPantalla.await();
			}
				Guardia.escribiendoPantalla = true;
				System.out.println("El coche numero " + id + " recibe
permiso y accede a la vía " + via);
				Guardia.escribiendoPantalla = false;
				Guardia.escribirPantalla.signal();
			}finally{
				Guardia.monitor.unlock();
			}
	}

	/**
	 * Este metodo se utiliza para imprimir por pantalla que el coche ha
cruzado la via. Debe imprimirlo en exclusión mutua
	 * y lo hace utilizando monitores.
	 * @throws InterruptedException En el caso de que no se puede utilizar el
monitor devuelve la excepción.
	 */
	private void imprimirSalidaVia() throws InterruptedException{
		Guardia.monitor.lock();
		try{
			if(Guardia.escribiendoPantalla){
				Guardia.escribirPantalla.await();
			}
				Guardia.escribiendoPantalla = true;
				System.out.println("El coche numero " + id + " ha
cruzado la vía " + via);
				Guardia.escribiendoPantalla = false;
				Guardia.escribirPantalla.signal();
			}finally{
				Guardia.monitor.unlock();
			}
	}

	@Override
	public void run(){
		while(true){
			/*Solicita el acceso a la via, esta llamada se hace para que
el controlador pueda tener conocimiento
			 * de que hay un coche que quiere entrar en la vía y asi
controlar el paso de los coches por la otra vía
			 * para que en un determinado momento los coches de esta vía
reciban permiso para pasar
			 */
			try{
				if(via == Via.A){
					guardia.call("peticionEntrarA", null);
					try{
						imprimirPeticionEntrada();
					} catch (InterruptedException e) {
					}
				}
				else if(via == Via.B){
					guardia.call("peticionEntrarB", null);
					try{
```

```
                                imprimirPeticionEntrada();
                            } catch (InterruptedException e) {
                            }
                        }
                    } catch (RemoteServerException e) {
                        e.printStackTrace();
                    }

                    // Una vez que la via activa sea la de este coche, el coche
debe poder tener acceso a cruzar.
                    try{
                        if(via == Via.A){
                            guardia.call("entrarA", null);
                            try{
                                imprimirAccesoVia();
                            } catch (InterruptedException e){
                            }
                        }
                        else if(via == Via.B){
                            guardia.call("entrarB", null);
                            try{
                                imprimirAccesoVia();
                            } catch (InterruptedException e){
                            }
                        }
                    } catch (RemoteServerException e) {
                        e.printStackTrace();
                    }

                    // Tiempo de cruce de la vía

                    try{
                        Thread.sleep(1000);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }

                    // Indicar que ha salido de la vía;
                    try{
                        if(via == Via.A){
                            try{
                                imprimirSalidaVia();
                            } catch (InterruptedException e){
                            }
                            guardia.call("salirA", null);

                        }
                        else if(via == Via.B){
                            try{
                                imprimirSalidaVia();
                            } catch (InterruptedException e){
                            }
                            guardia.call("salirB", null);

                        }
                    } catch (RemoteServerException e) {
                        e.printStackTrace();
                    }
                }
            }
        }
```

Peaton

```
public class Peaton implements Runnable{
	private final CommunicationScheme canal;
	private final int marcaje;
	private static int numeroPeatones;
	private int id;

	/**
	 * Constructor de la clase peaton.
	 * @param canal El canal por el que el peaton pide acceso al marcaje.
	 * @param marcaje Marcaje por el que va a pasar el peaton.
	 */
	public Peaton (CommunicationScheme canal, int marcaje){
		this.canal = canal;
		this.marcaje =  marcaje;
		numeroPeatones++;
		id = numeroPeatones;
	}

	/**
	 * Metodo que se utiliza para que, cuando un peaton cruce una via, indique
al constructor que ya ha cruzado
	 * y decremente el numero de peatones cruzando. Este metodo debe
ejecutarse en exclusión mutua y lo hace utilizando
	 * semaforos.
	 */
	public void decrementar(){
		try{
			Guardia.mutex.acquire();
		} catch (InterruptedException e){}
		Guardia.decrementarPeatonesCruzando();
		Guardia.mutex.release();
	}

	/**
	 * Este metodo imprime por pantalla, en exclusion mutua utilizando
monitores, que que un peaton ha
	 * pedido permiso para cruzar una via.
	 * @throws InterruptedException
	 */
	private void imprimirPeticionPeaton() throws InterruptedException{
		Guardia.monitor.lock();
		try{
			if(Guardia.escribiendoPantalla){
				Guardia.escribirPantalla.await();
			}
				Guardia.escribiendoPantalla = true;
				System.out.println("El peaton " + id + " pide permiso
para cruzar por el marcaje " + marcaje);
				Guardia.escribiendoPantalla = false;
				Guardia.escribirPantalla.signal();
			}finally{
				Guardia.monitor.unlock();
			}
	}

	/**
	 * Este metodo imprime por pantalla que un peaton ha empezado a cruzar la
```

```
via. Lo hace mediante exclusión mutua
	 * utilizando monitores.
	 * @throws InterruptedException En el caso de que no se puede utilizar el
monitor devuelve la excepción.
	 */
	private void imprimirEntradaPeaton() throws InterruptedException{
		Guardia.monitor.lock();
		try{
			if(Guardia.escribiendoPantalla){
				Guardia.escribirPantalla.await();
			}
				Guardia.escribiendoPantalla = true;
				System.out.println("El peaton " + id + " entra a cruzar
el marcaje " + marcaje);
				Guardia.escribiendoPantalla = false;
				Guardia.escribirPantalla.signal();
			}finally{
				Guardia.monitor.unlock();
			}
	}

	/**
	 * Este metodo imprime por pantalla que un peaton ha cruzado la via. Lo
hace mediante exclusión mutua
	 * utilizando monitores.
	 * @throws InterruptedException En el caso de que no se puede utilizar el
monitor devuelve la excepción.
	 */
	private void imprimirSalidaPeaton() throws InterruptedException{
		Guardia.monitor.lock();
		try{
			if(Guardia.escribiendoPantalla){
				Guardia.escribirPantalla.await();
			}
				Guardia.escribiendoPantalla = true;
				System.out.println("El peaton " + id + " ha cruzado el
marcaje " + marcaje);
				Guardia.escribiendoPantalla = false;
				Guardia.escribirPantalla.signal();
			}finally{
				Guardia.monitor.unlock();
			}
	}

	@Override
	public void run() {
		while(true){
			// Pide acceso a la via
			try{
				imprimirPeticionPeaton();
			} catch (InterruptedException e){
			}

			canal.send(marcaje);

			try{
				imprimirEntradaPeaton();
			} catch (InterruptedException e){
			}

			// Peaton cruza la calle
```

```
                        try{
                              Thread.sleep(2000);
                        } catch (InterruptedException e) {
                              e.printStackTrace();
                        }
                        // Sale de la via
                        try{
                              imprimirSalidaPeaton();
                        } catch (InterruptedException e){
                        }
                        decrementar();
                  }
            }
}
```

Programa

```
public class Programa {

      /**
       * @param args
       */
      public static void main(String[] args) {
            Channel[] canalesPeatones = new Channel[11];
            Peaton[] peatones = new Peaton[11];
            Thread[] threadsPeatones = new Thread[11];
            Thread[] threadCoches = new Thread[10];

            //Inicializar los canales
            for(int i = 0; i < canalesPeatones.length; i++){
                  canalesPeatones[i] = new Channel();
            }

            //Consideramos 3 peatones para el marcaje 1, 2 para el marcaje 2, 4
para el marcaje 3
            // y 2 para el marcaje 4 y los inicializamos;
            for(int i = 0; i < peatones.length; i++){
                  if(i < 3){
                        peatones[i] = new Peaton(canalesPeatones[i], 1);
                  }
                  else if(i < 5){
                        peatones[i] = new Peaton(canalesPeatones[i], 2);
                  }
                  else if(i < 9){
                        peatones[i] = new Peaton(canalesPeatones[i], 3);
                  }
                  else{
                        peatones[i] = new Peaton(canalesPeatones[i], 4);
                  }
            }

            Guardia guardia = new Guardia(canalesPeatones);

            //Creación de threads
            for(int i = 0; i < threadsPeatones.length; i++){
                  threadsPeatones[i] = new Thread(peatones[i]);
                  threadsPeatones[i].start();
            }

            Thread threadGuardia =  new Thread(guardia);
```

```
            threadGuardia.start();

            for(int i = 0; i < threadCoches.length; i++){
                  if(i < 5){
                        threadCoches[i] = new Thread(new Coche(guardia, Via.A));
                  }
                  else{
                        threadCoches[i] = new Thread(new Coche(guardia, Via.B));
                  }
                  threadCoches[i].start();
            }


            try{
                  for(int i = 0; i < threadsPeatones.length; i++){
                        threadsPeatones[i].join();
                  }
                  threadGuardia.join();
                  for(int i = 0; i < threadCoches.length; i++){
                        threadCoches[i].join();
                  }
            } catch (InterruptedException e){
                  e.printStackTrace();
            }
      }
}
```