Examen concurrencia Nov 2011

Programación concurrente y Distribuída

Curso 2011-12

Miguel Telleria, Laura Barros, J.M. Drake

`telleriam AT unican.es`

Computadores y Tiempo Real

http://www.ctr.unican.es

Contenido

- Código base
- Examen del miércoles
  - Solución con 4 estados
  - Solución con 3 estados
- Examen del jueves
  - Solución sin basarse en estados
  - Solución basada en estados

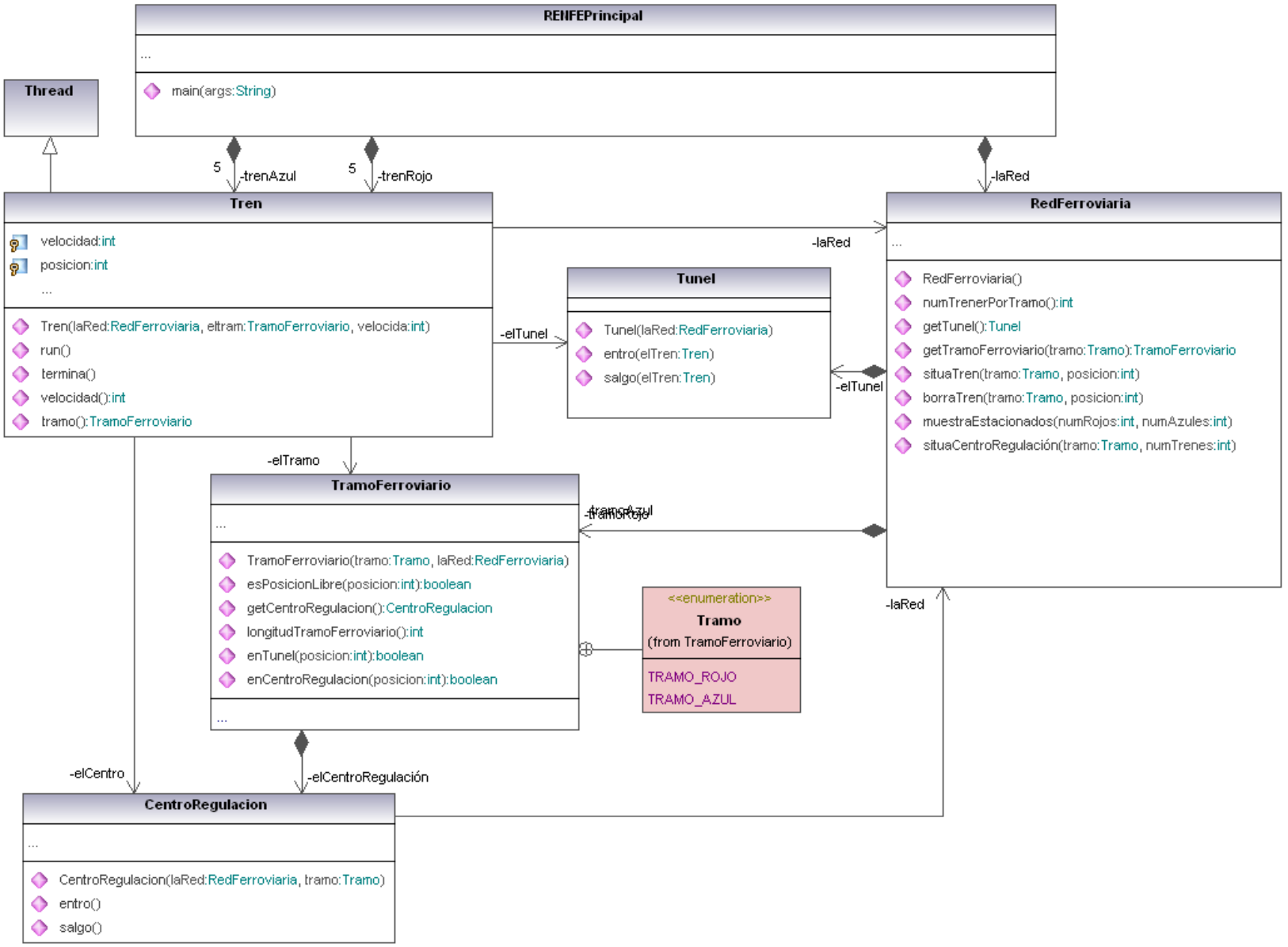
Código base

Los códigos del miércoles y jueves fueron distintos

- Pero la interfaz es la misma

- Se esperaba un diseño desde cero.
  - El código base sirve como interfaz a respetar desde el tren.
  - También sirve como prueba de que la funcionalidad clásica es implementable.

Diagrama de clases



API del Tunel que se ha de respetar

- void entro(Tren elTren)
  - La llama el tren **una única vez** en su recorrido esperando ser bloqueado (y aparcado) si no puede entrar.
  - El tren no pide permiso antes, **el permiso y el aparcamiento están incluidos dentro del entro()** del túnel.
  - Pasamos **el objeto tren entero** (y no sólo su color). Para obtener el color y la velocidad existen los métodos públicos de la clase tren:
    - TramoFerroviario.Tramo tramo()
    - int velocidad()
- void salgo(Tren elTren)
  - Lo llama el tren una única vez cuando sale del tunel.
  - También se pasa el objeto tren.

Examen del miércoles

Política del miércoles

- En cuanto un tren rojo esté estacionado, los azules han de dejar de entrar para dar prioridad a los rojos estacionados.
  - En todos los casos se ha de respetar que un sólo color esté en el túnel.
  - Los rojos estacionados han de esperar a que salgan los azules que existen.

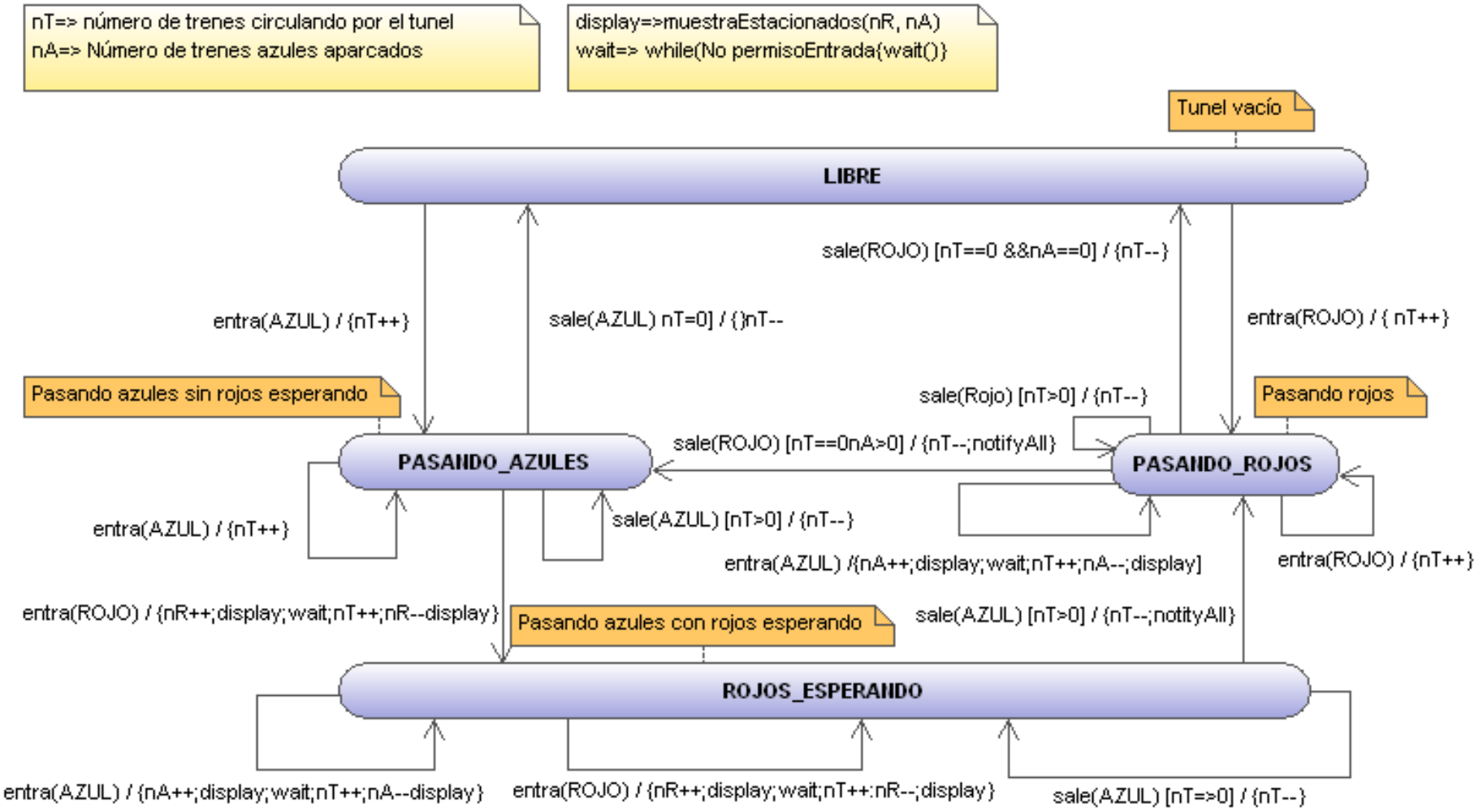
- Se pide:
  - La funcionalidad
  - Diagrama de estados
  - Implementación basada en el diagrama de estados
  - Justificar los bloques o métodos synchronized

La funcionalidad se podía hacer fácilmente...

```
/**
  * Lo invoca un tren para conocer si est autorizado a circular por el tunel en
  * ese instante.
  */
private synchronized boolean autorizadoEntrar(TramoFerroviario.Tramo tramo)
{
    if (numCirculando>0)
    {
        return ((tramo==TramoFerroviario.Tramo.TRAMO_ROJO)&&autorizadosRojos)
               || ((tramo==TramoFerroviario.Tramo.TRAMO_AZUL)&&!autorizadosRojos
               && (rojosEstacionados==0)
              ) ;
    }
    else
    {
        return true;
    }
}
```

… aunque esperábamos otra cosa: diseño desde cero

Solución 4 estados: diagrama



Solución máquina de 4 estados: estados, eventos, acciones

- Estados

  LIBRE PASANDO_ROJOS

  PASANDO_AZULES ROJOS_ESPERANDO

- Eventos

  Entra ROJO Entra AZUL

  Sale ROJO Sale AZUL

- Variables de guarda (y acciones)

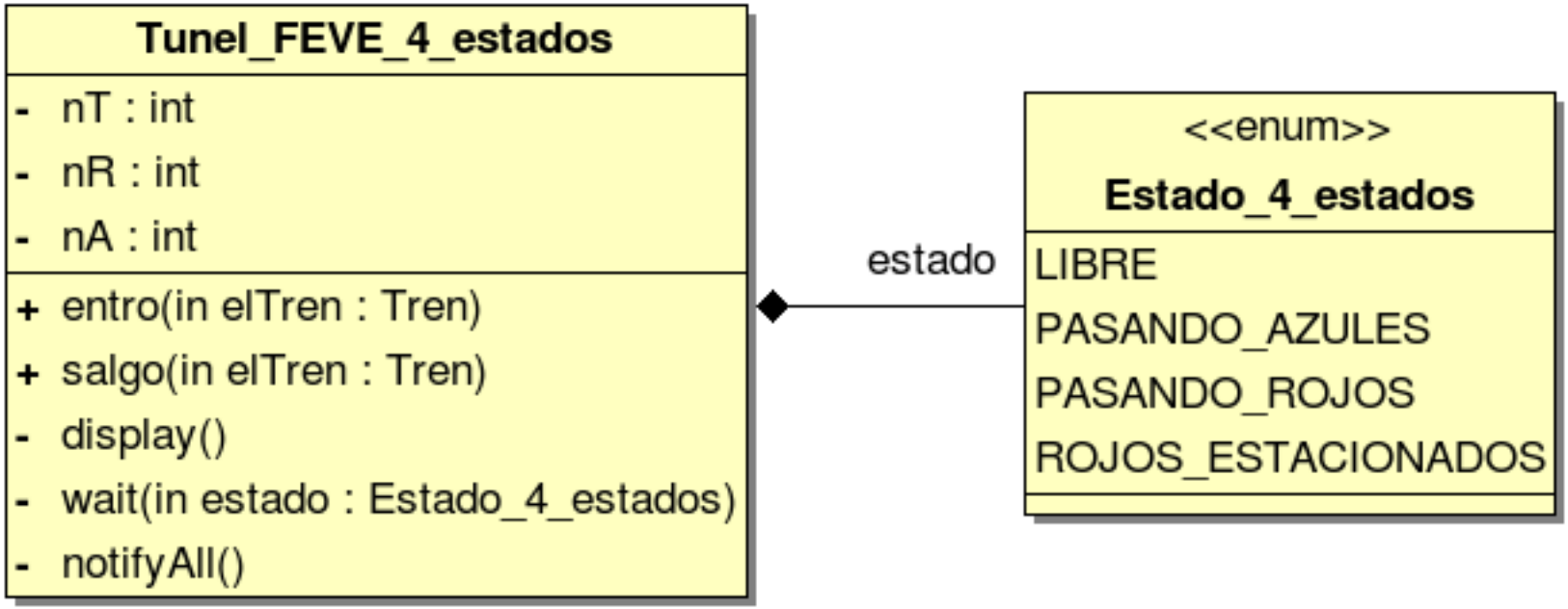
  nT (numCirculando) nA (azulesEstacionados) nR (rojosEstacionados)

- Acciones

  notifyAll() display() wait() (en el estado)

  nT++ nT-- nA++ nA-- nR++ nR--

Solución de 4 estados: clases y atributos



solución 4 estados: entro(AZUL)

```
public synchronized void entro(Tren elTren)
{
    if (elTren.tramo()==TramoFerroviario.Tramo.TRAMO_AZUL)
    {
        switch(estado)
        {
        case LIBRE:
            estado=EstadoTunel.PASANDO_AZULES;
        case PASANDO_AZULES:
            nT=nT+1;
            break;
        case PASANDO_ROJOS:
        case ESPERANDO_ROJOS:
            nA=nA+1;
            laRed.muestraEstacionados(nR, nA);
            while(estado!=EstadoTunel.PASANDO_AZULES){
                try{wait();}catch(InterruptedException e){}
            }
            nT=nT+1;
            nA=nA-1;
            laRed.muestraEstacionados(nR, nA);
        }
    }
}
```

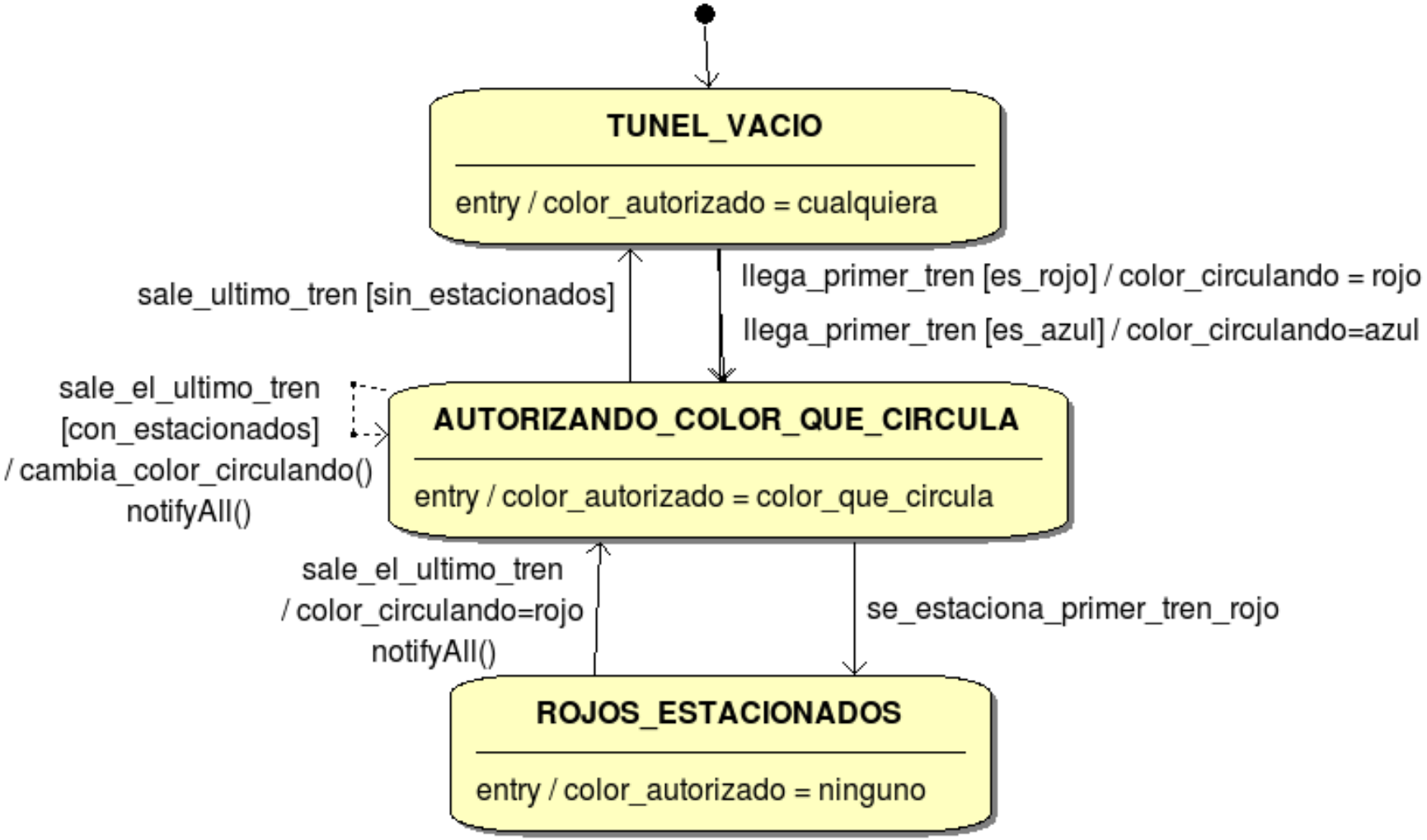
Solución 4 estados: entro(ROJO)

```
    else
    { //tramo=TramoFerroviario.Tramo.TRAMO_ROJO
        switch(estado)
        {
        case LIBRE:
            estado=EstadoTunel.PASANDO_ROJOS;
        case PASANDO_ROJOS:
            nT=nT+1;
            break;
        case PASANDO_AZULES:
            estado=EstadoTunel.ESPERANDO_ROJOS;
        case ESPERANDO_ROJOS:
            nR=nR+1;
            laRed.muestraEstacionados(nR, nA);
            while(estado!=EstadoTunel.PASANDO_ROJOS){
                try{wait();}catch(InterruptedException e){}
            }
            nT=nT+1;
            nR=nR-1;
            laRed.muestraEstacionados(nR, nA);
        }
    }
}
```

Solución 4 estados: salgo()

```
public synchronized void salgo(Tren elTren)
{
     nT=nT-1;
     switch (estado)
     {
     case PASANDO_AZULES:
          if (nT==0)estado=EstadoTunel.LIBRE;
          break;
     case PASANDO_ROJOS:
          if (nT==0)
               if (nA==0)
                    estado=EstadoTunel.LIBRE;
               else{
                    estado=EstadoTunel.PASANDO_AZULES;
                    notifyAll();
               }
          break;
     case ESPERANDO_ROJOS:
          if (nT==0)
          {
               estado=EstadoTunel.PASANDO_ROJOS;
               notifyAll();
          }
          break;
     }
}
```

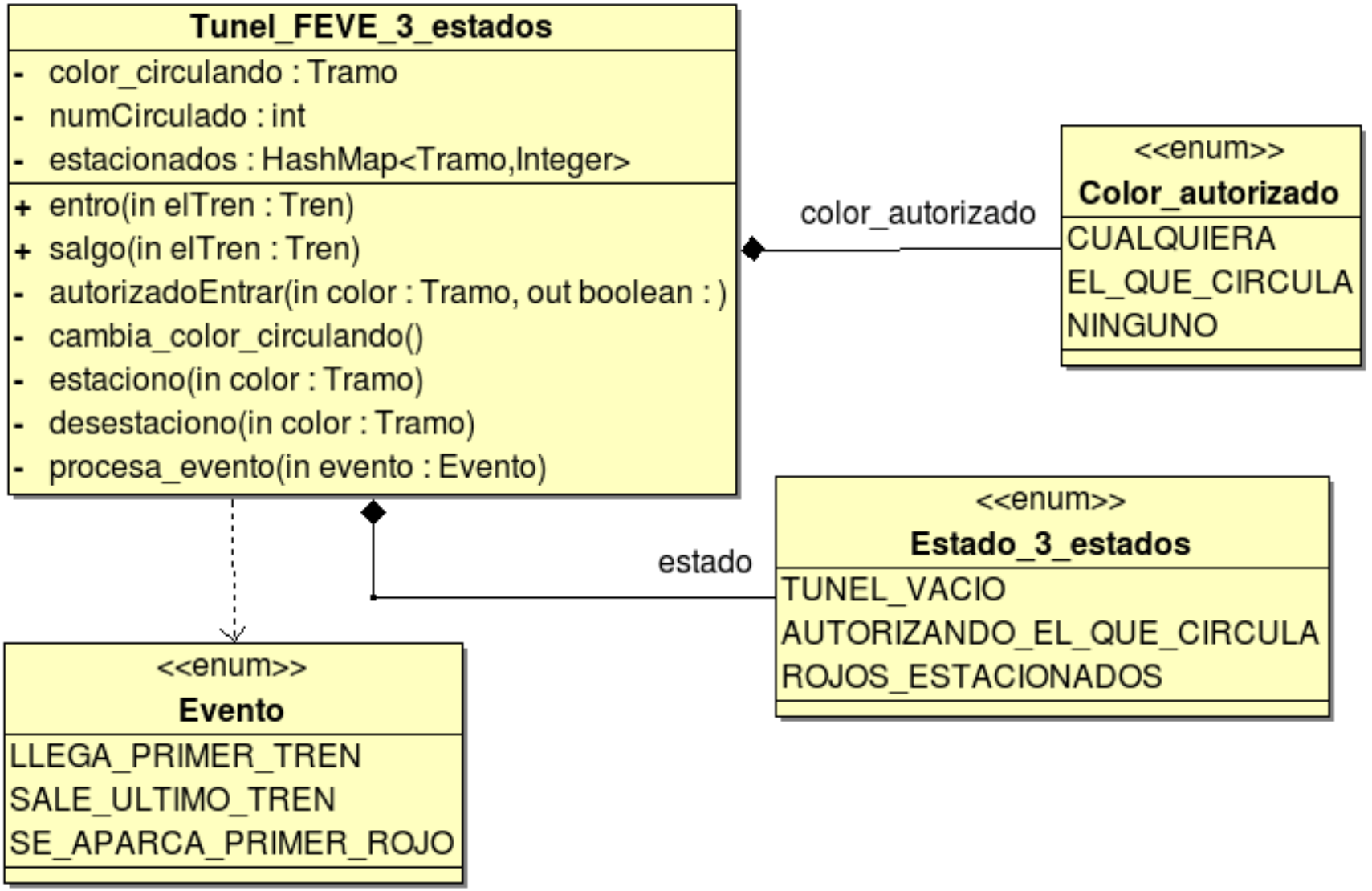
Solución con 3 estados: diagrama



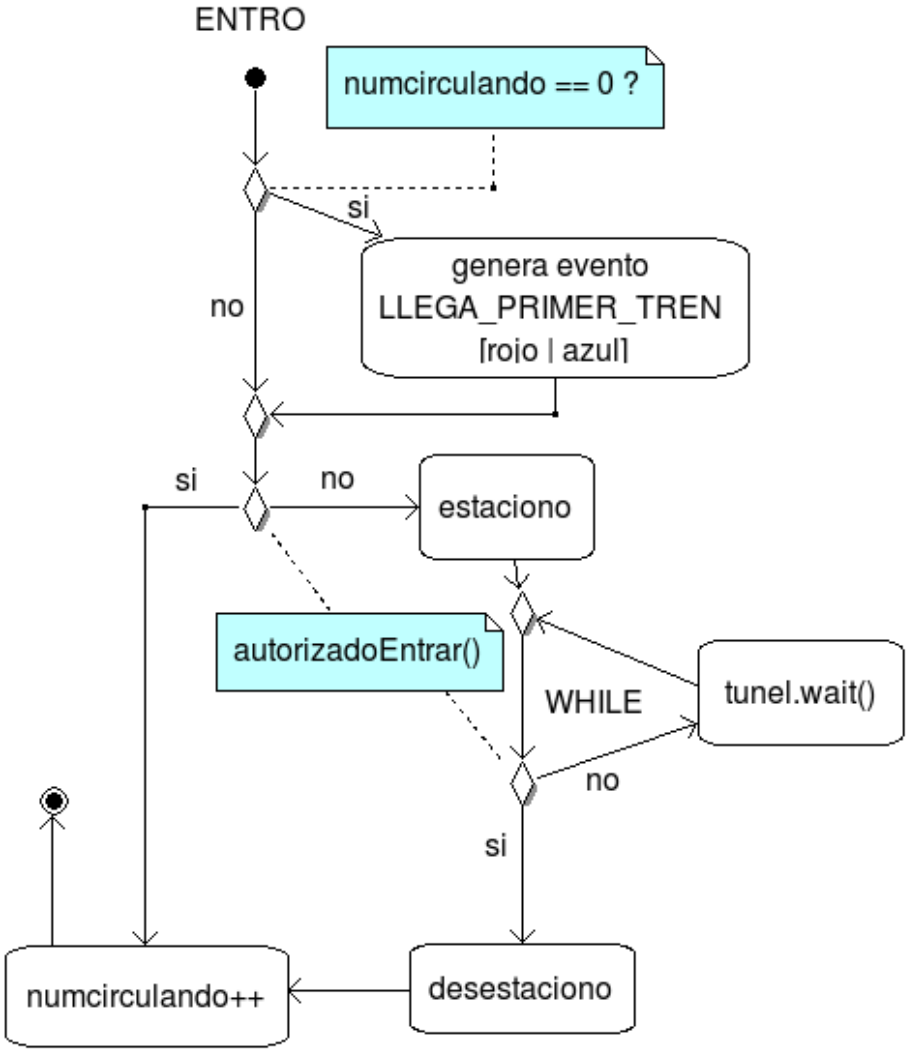
Solución con 3 estados: estados, eventos, acciones

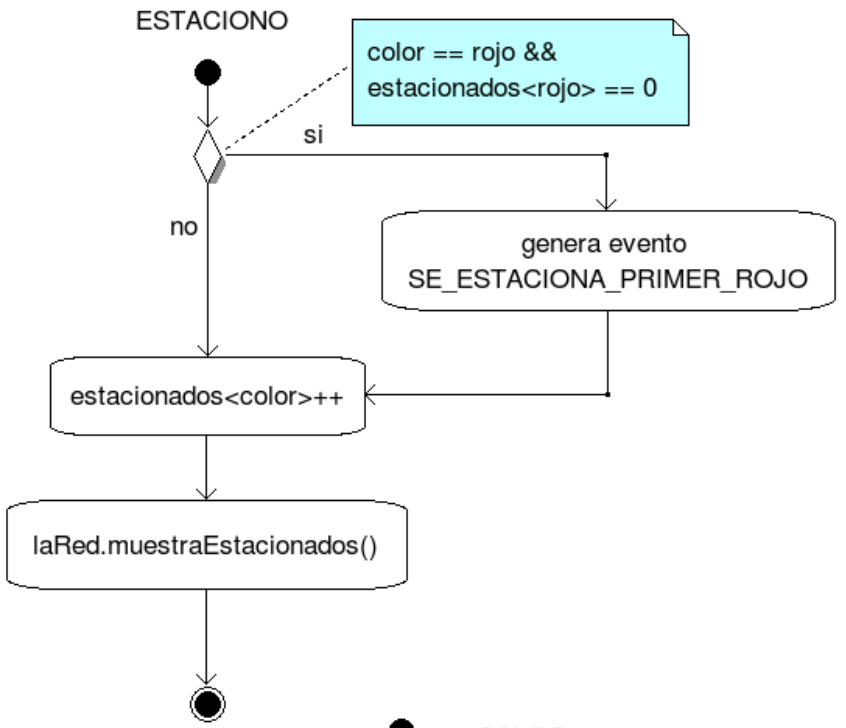
- Estados
  - TUNEL_VACIO
    - entry: color_autorizado = cualquiera
  - AUTORIZANDO_COLOR_QUE_CIRCULA
    - entry: color_autorizado = el_que_circula
  - ROJOS_ESTACIONADOS
    - entry: color_autorizado = ninguno
- Eventos
  - LLEGA_EL_PRIMER_TREN
    - condición de guarda: si es rojo o azul
  - SALE_EL_ULTIMO_TREN
    - condición de guarda: si hay aparcados o no
  - SE_ESTACIONA_PRIMER_TREN_ROJO
- Acciones
  - notifica_a_los_estacionados
  - color_cirulando = rojo | azul
  - cambia_color_circulando

Solución con 3 estados: diagrama de clases



Solución de 3 estados: diagrama actividad



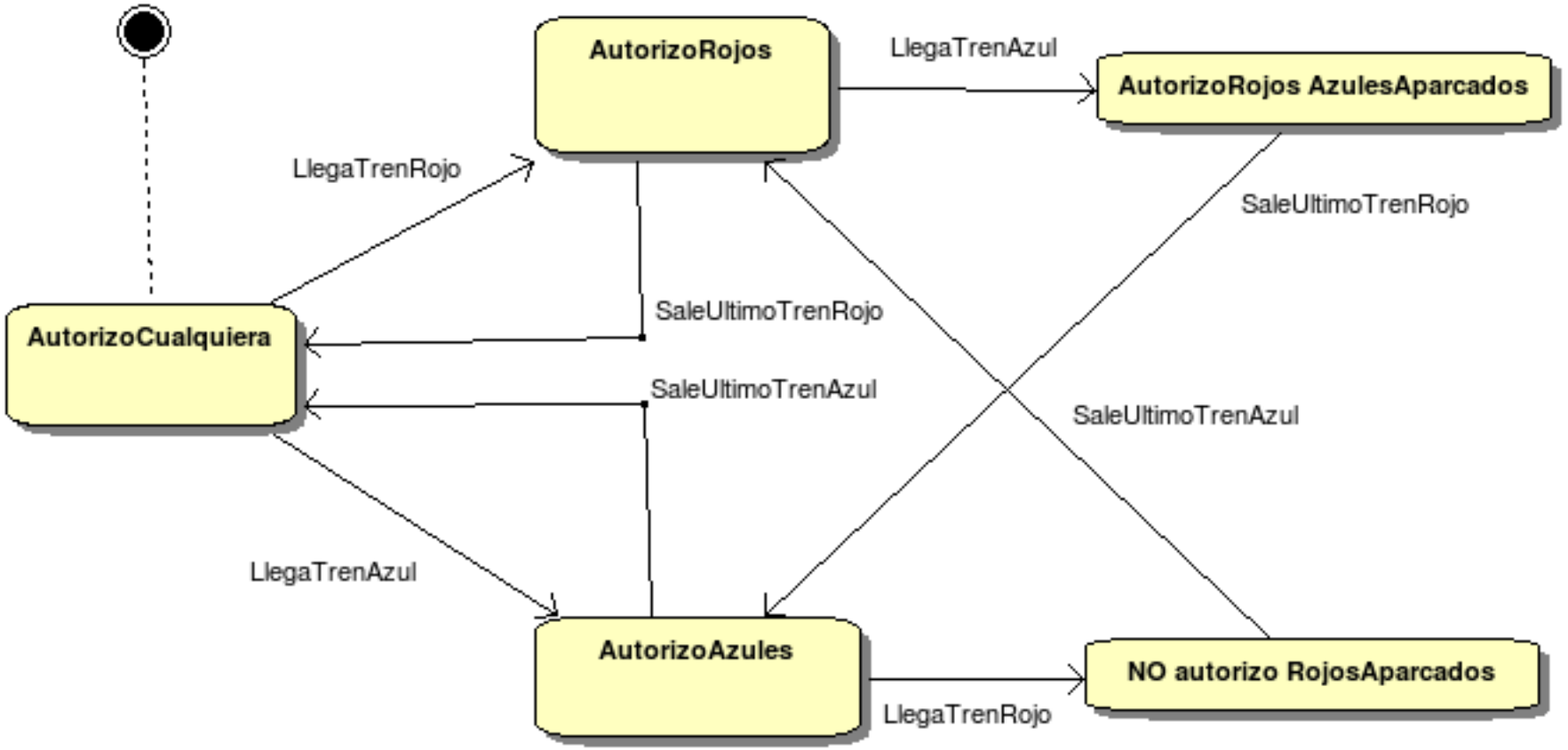


Solución de 3 estados: autorizadoEntrar

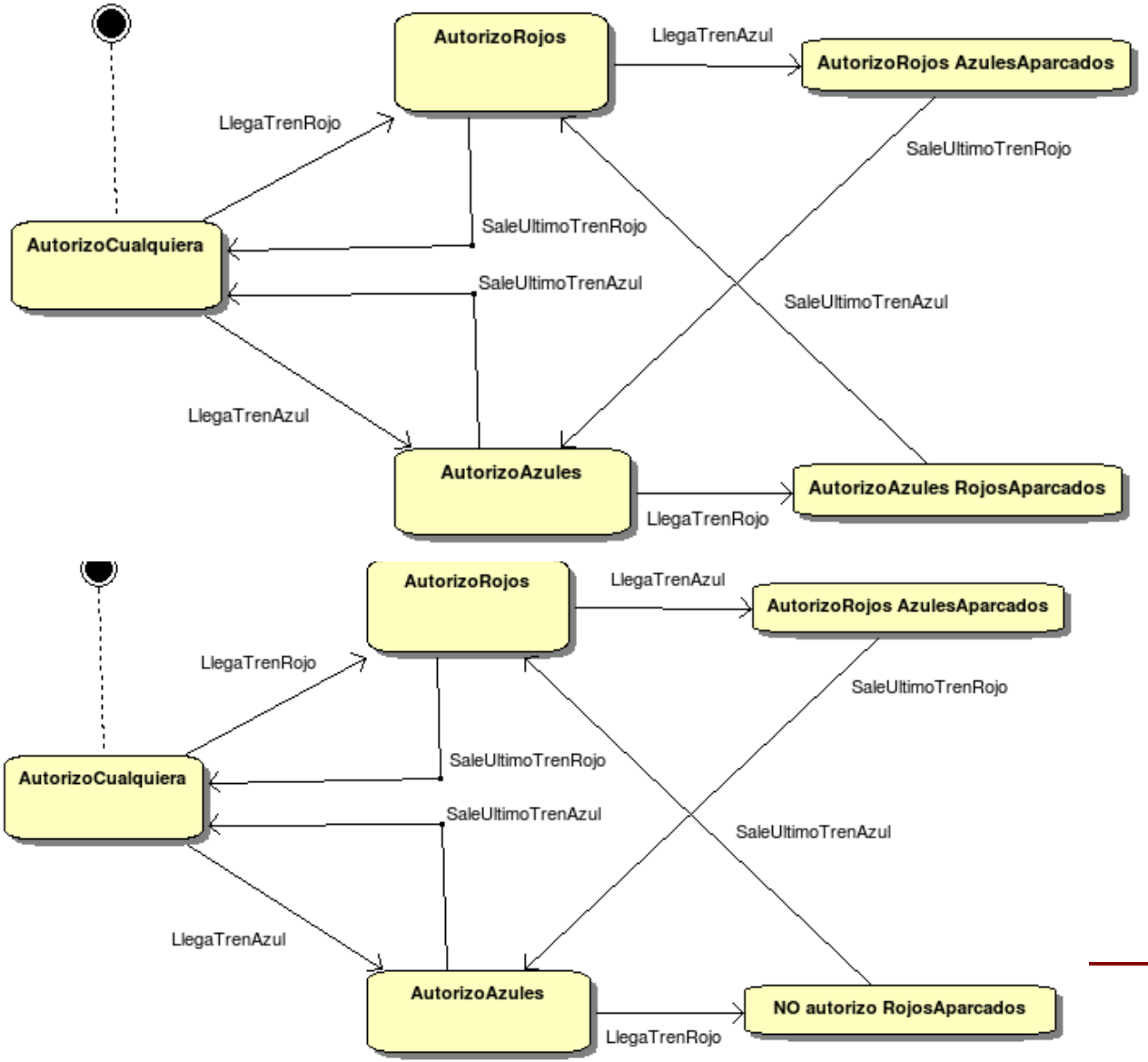
- El autorizado entrar sale directamente del enum color_autorizado (el cual está ligado al estado).

```
private boolean autorizadoEntrar(TramoFerroviario.Tramo tramo)
{
    switch (color_autorizado)
    {
    case AUTORIZO_CUALQUIERA:
        return true;

    case AUTORIZO_COLOR_CIRCULANDO:
        return tramo == color_circulando;

    case AUTORIZO_NINGUNO:
        return false;
    }

    return false;
}
```

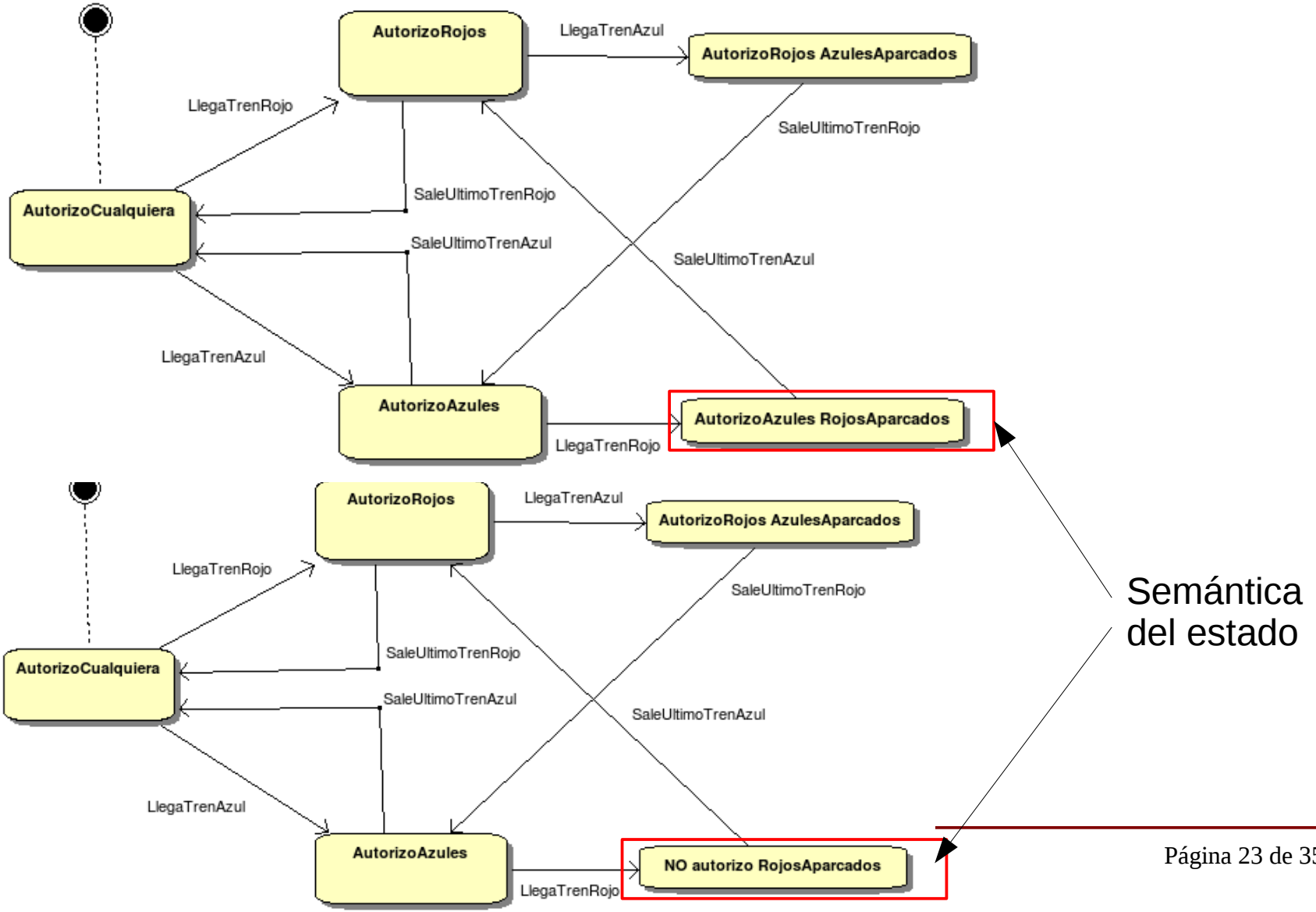
Máquina de 5 estados



Diferencias entre esta última y la de clase



Diferencias entre esta última y la de clase



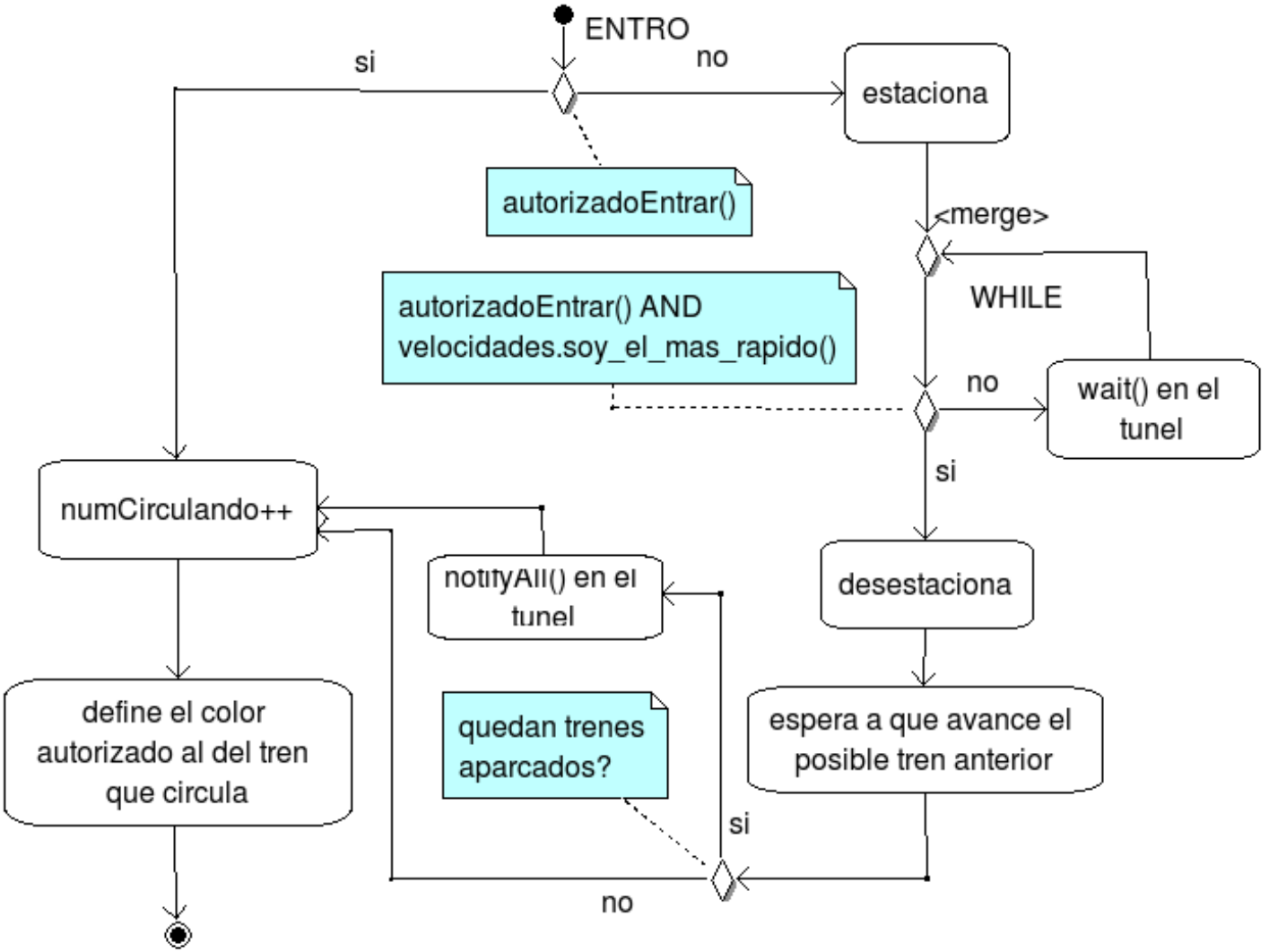
Examen del jueves

Política del jueves

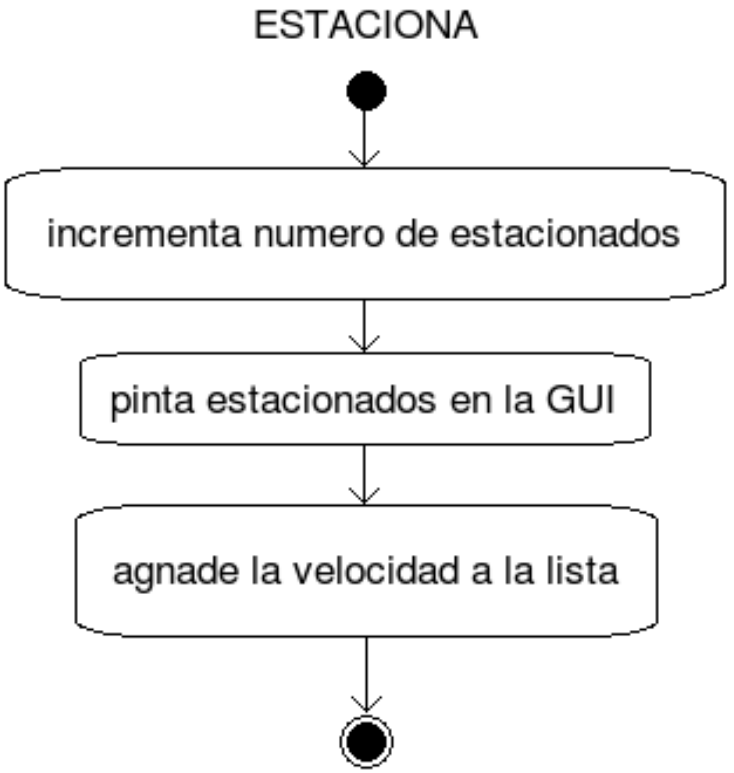
- Los trenes que están esperando en un extremo por estar el túnel ocupado, al ser permitida su entrada, entran en el túnel **ordenados según su velocidad**. Los trenes con velocidad más alta entran primero en el túnel.
  - Visualmente: Ningún tren que sale del estacionamiento “empuja” a otro.

- Se pide
  - La funcionalidad
  - Diagrama de actividad de entro()
  - Que la implementación siga ese diagrama
  - Justificar los bloques synchronized

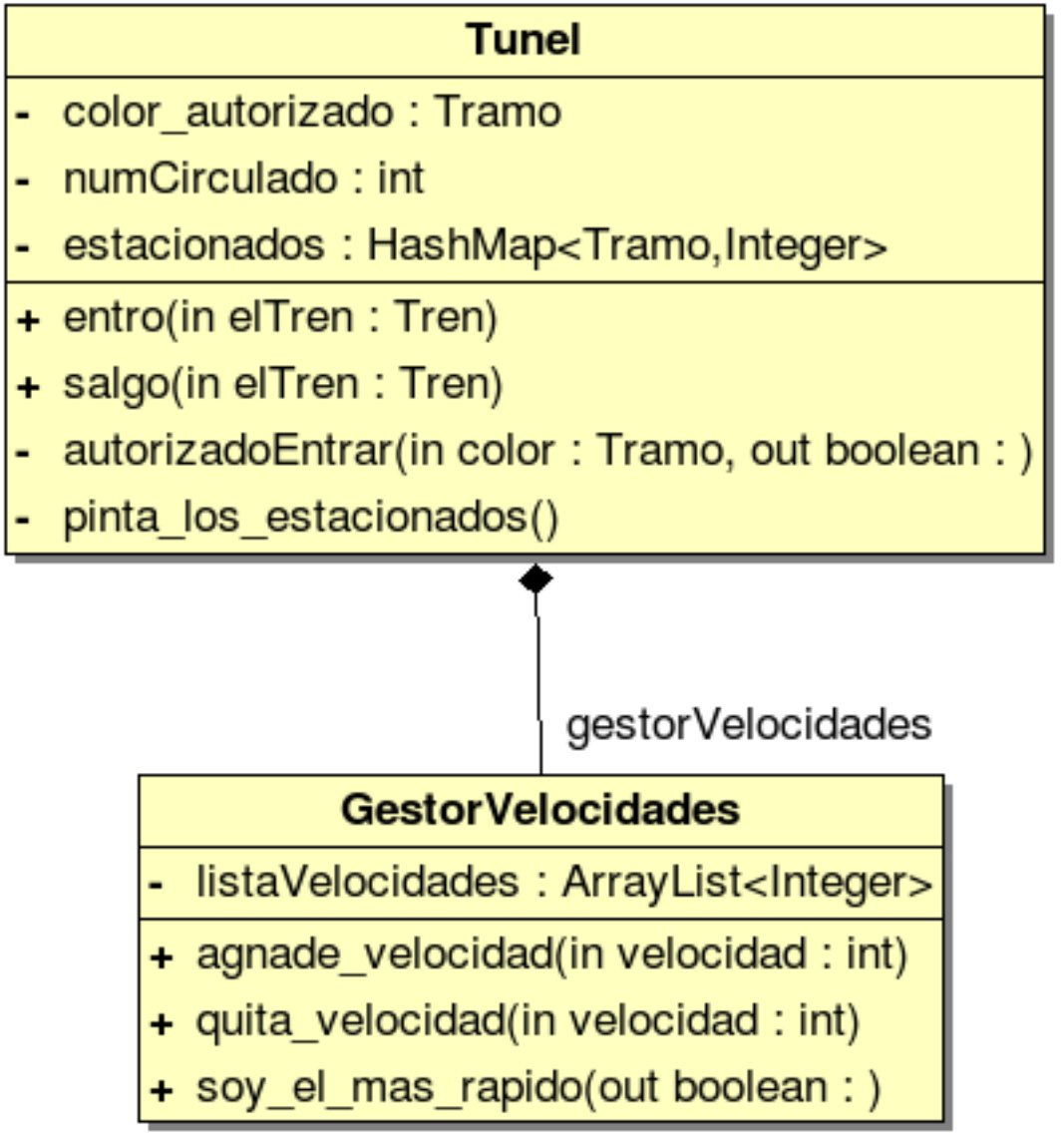
Solución sin estados: Diagrama actividad



Solución sin estados: Diagrama actividad



Solución sin estados (diagrama de clases)



```
public synchronized void entro(Tren elTren)
{
    TramoFerroviario.Tramo tramo = elTren.color();
    int velocidad = elTren.velocidad();

    if( !autorizadoEntrar(tramo) )
    {
        estaciona(elTren);
        while( !autorizadoEntrar(tramo) ||
             !gestorVelocidades.soy_el_mas_rapido(velocidad) )
        {
            try { wait();} catch (InterruptedException e) {e.printStackTrace();}
        }
        desestaciona(elTren);

        // Si quedan trenes estacionados
        if (estacionados.get(tramo) > 0)
        {
            notifyAll();  // Puede que se hubieran dormido por no ser el mas rapido
        }

        try { // Esperamos por si hay un rezagado que ha salido sin avanzar
            Thread.sleep( (RENFEPrincipal.TIEMPO_ACTUALIZACION_MS * 2) / );
        } catch (InterruptedException e) {e.printStackTrace();}
    }
    numCirculando=numCirculando+1;
    color_autorizado = tramo;
}
```

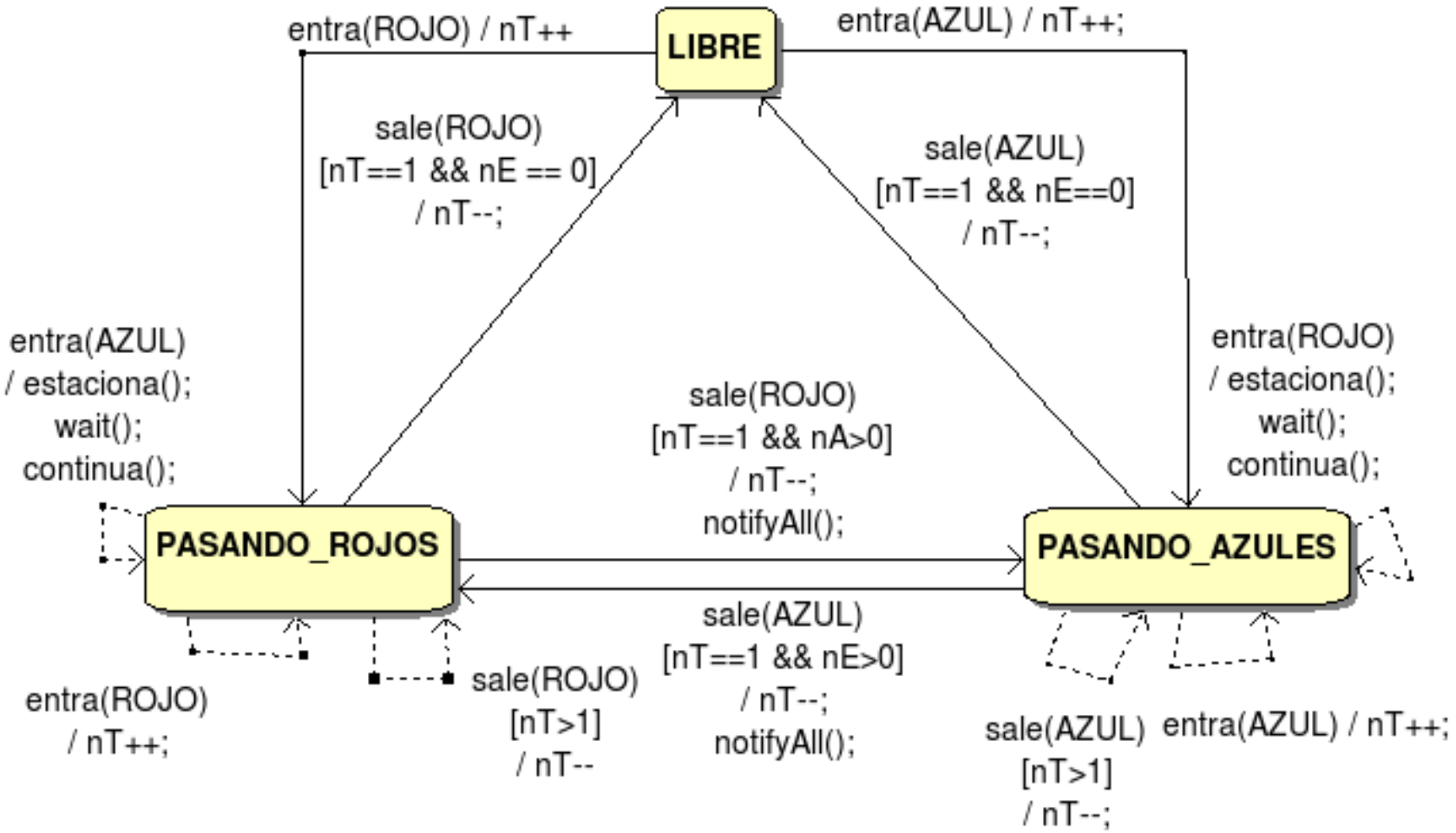
estaciona() y desestaciona()

```
private void estaciona(Tren elTren)
{
    int velocidad = elTren.velocidad();
    TramoFerroviario.Tramo tramo = elTren.color();

    estacionados.put(tramo, estacionados.get(tramo) + 1);
    pinta_los_estacionados();
    gestorVelocidades.agnade_velocidad(velocidad);
}

private void desestaciona(Tren elTren)
{
    int velocidad = elTren.velocidad();
    TramoFerroviario.Tramo tramo = elTren.color();

    estacionados.put(tramo, estacionados.get(tramo) -1);
    pinta_los_estacionados();
    gestorVelocidades.quita_la_velocidad(velocidad);
}
```

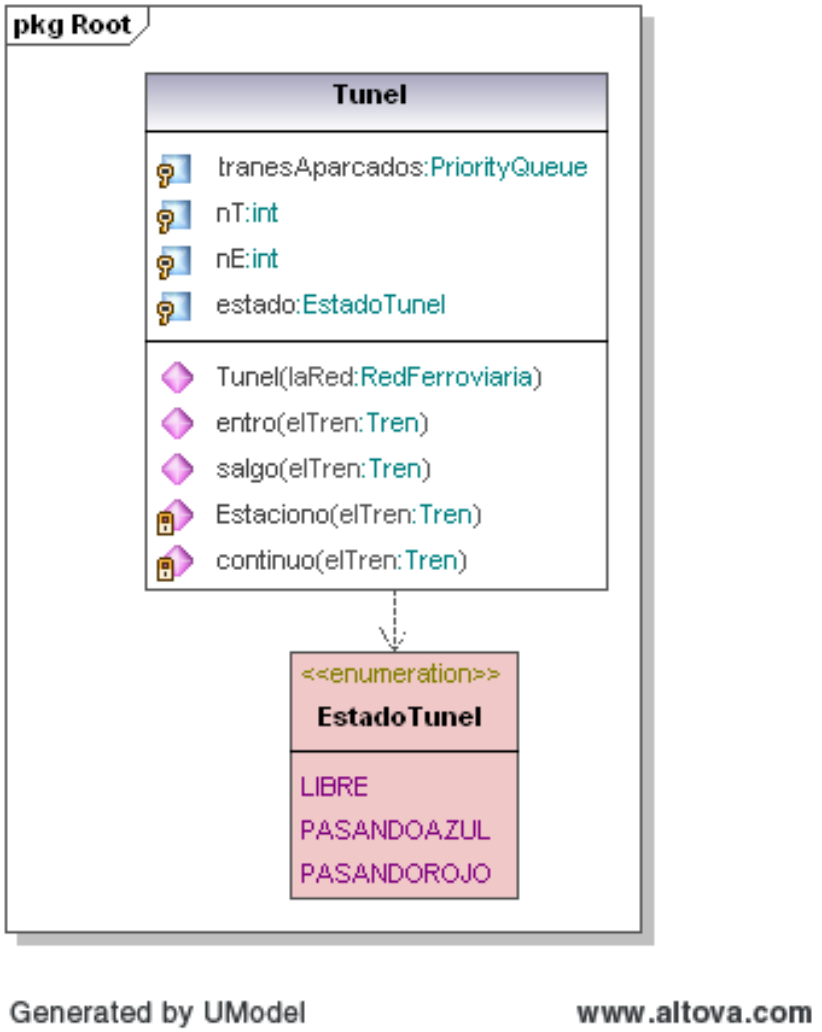
Implementación del gestor velocidades

```
public class GestorVelocidades
{
    private PriorityQueue<Integer> lasVelocidades =
                new PriorityQueue<Integer>(5,new GestorVelocidades.Comparador());

    public class Comparador implements Comparator<Integer>
    {
      public int compare(Integer v1, Integer v2){ return v2 - v1; }
    }

    void agnade_velocidad(int velocidad)
    {
        lasVelocidades.add(velocidad);
    }

    boolean soy_el_mas_rapido(int velocidad)
    {
        return velocidad == lasVelocidades.peek();
    }

    void quita_la_velocidad(int velocidad)
    {
        lasVelocidades.remove(velocidad);
    }
}
```

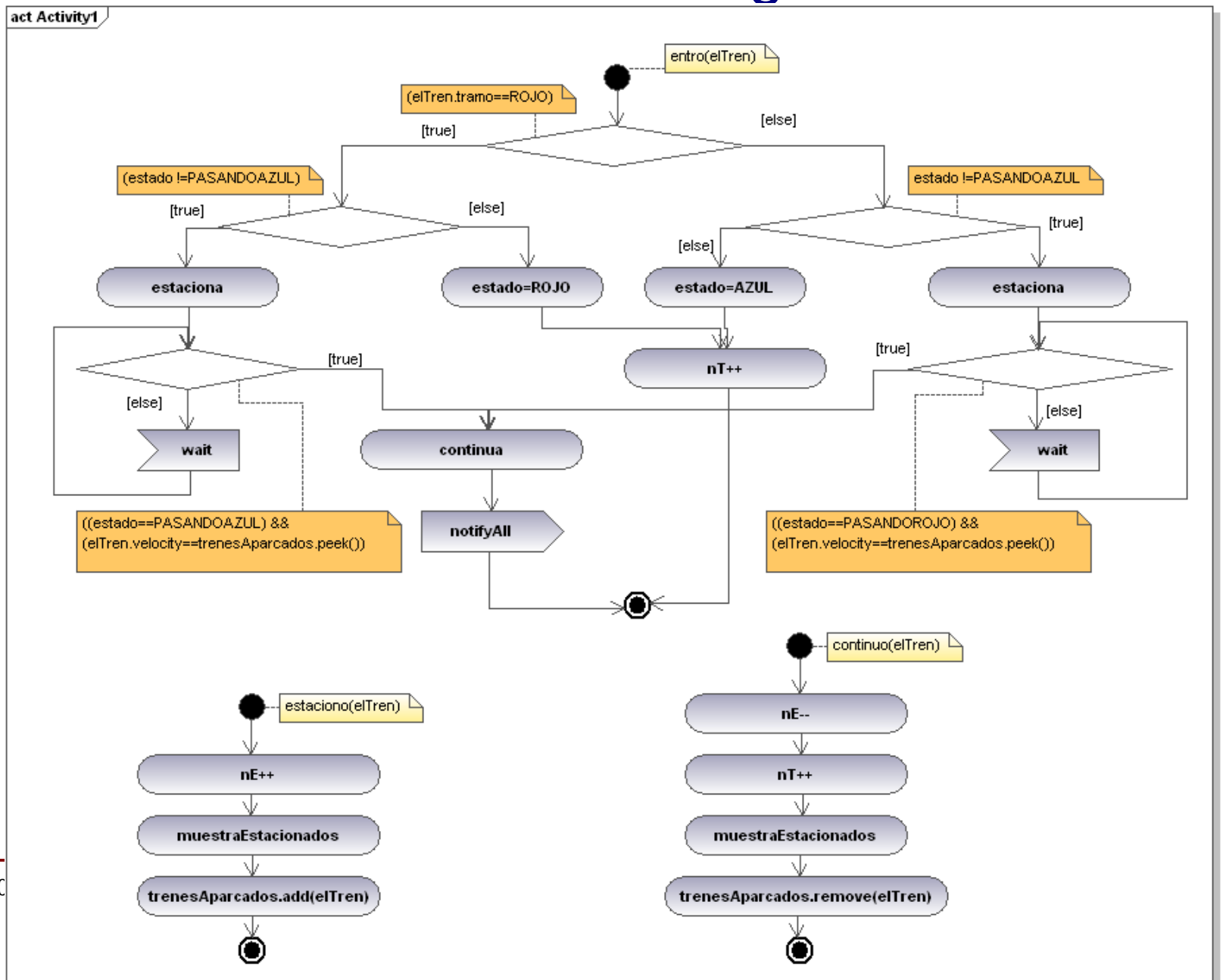
Solución con estados: Diagrama de estados



Solución con estados: diagrama de clases



Solución con estados: diagrama de actividad



Implementación de entro (con estados)

```
public synchronized void entro(Tren elTren)
{
    if (elTren.tramo()==TramoFerroviario.Tramo.TRAMO_AZUL)
    {
        switch(estado)
        {
        case LIBRE:
            estado=EstadoTunel.PASANDO_AZULES;
        case PASANDO_AZULES:
            nT++;
            break;

        case PASANDO_ROJOS:
            estaciono(elTren);
            while((estado==EstadoTunel.PASANDO_ROJOS)||
                  (elTren != trenesAparcados.peek() ) )
            {
                try{wait();}catch(InterruptedException e){}
            }
            continuo(elTren);
            notifyAll();
            try {
                Thread.sleep(200);
            } catch (InterruptedException e) {e.printStackTrace();
            System.out.println(elTren.velocidad());
        }
    }else{ // El tres es ROJO se trata de manera similar}
}
```