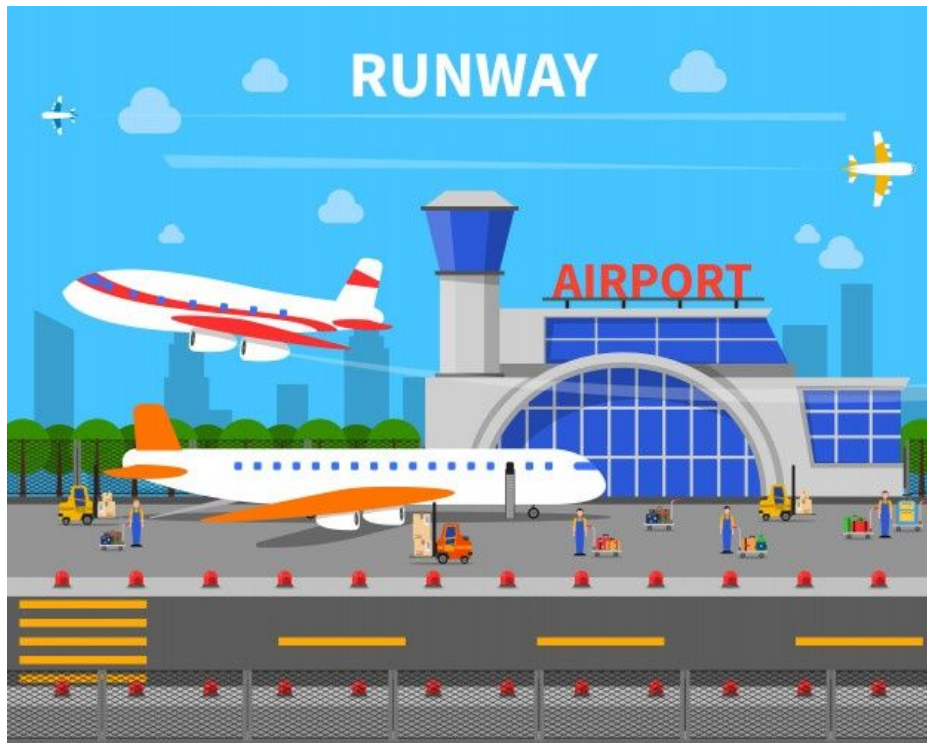


Trabajo final

Vuelo bonito



Alumno: Carlos Adolfo Amarante

Materia: Programación Concurrente

Año: 2019

Profesores: Silvia Amaro, Agustin Chiarotto,
Juan Pablo Orlando, Valeria Zoratto

Universidad Nacional Del Comahue

Facultad de informática

Índice

Índice	2
Introducción	6
Consigna	7
Análisis general de recursos activos	9
Perspectiva del pasajero	9
Perspectiva del vuelo	9
Perspectiva del tren	10
Perspectiva del tiempo	10
Análisis general de recursos pasivos	11
Aerolinea	11
Aeropuerto	11
CajaFreeShop	11
FreeShop	11
TorreDeControl	11
Clases	12
Aerolinea	12
Atributos	12
Métodos	12
Aerolinea (String nombre, int cantidadDePuestosDeAtencion)	12
getNombre()	12
setNombre(String nombre)	12
toString()	13
usarPuestoDeAtencion (Pasajero pasajero)	13
Aeropuerto	13
Atributos	13
Métodos	13
Aeropuerto (Tren tren, int cantidadDePuestosDeInformes,int capacidadFS, int cantCajasFS)	13
setTorre(AtomicInteger hora, Aerolinea[]aerolineas)	14
getTren()	14
pedirPuestoDeInforme(Pasajero pasajero)	14
pasarTiempo()	15
programarVuelosDelDia()	15
comenzarDiaLaboral()	15
terminarDiaLaboral()	15
usarTren(Pasajero pasajero)	15

comprarFreeShop(Pasajero pasajero)	16
verificarDespegues(int hora)	16
cantVuelosEnHora()	16
CajaFreeShop	16
Atributos	16
Métodos	16
CajaFreeShop(char terminal, String emoji2)	17
toString()	17
atenderPasajero(Pasajero pasajero)	17
FreeShop	17
Atributos	17
Métodos	17
FreeShop(int capacidad, char terminal, int cantCajas, String emoji)	18
toString()	18
comprar(Pasajero pasajero)	18
Pasajero	19
Atributos	19
Métodos	19
Pasajero(AtomicInteger hora, Aeropuerto aeropuerto)	19
setPasaje(Vuelo vuelo)	19
getHoraActual()	19
getAeropuerto()	20
getIdPasajero()	20
getPasaje()	20
toString()	20
run()	20
comprarEnFreeShop()	21
Tiempo	21
Atributos	21
Métodos	21
Tiempo (Aeropuerto aeropuerto, Aerolinea[] aerolineas, int capacidadTren, int factorHL, int factorHNL, boolean esTest)	21
getHora()	22
iniciarDiaLaboral()	22
terminarDiaLaboral()	22
hacerAndarTren()	22
run()	22
mensajeTest(boolean esHL)	23
pasarHoraLaboral()	24
pasarHoraNoLaboral()	24
generarPasajeros(boolean horaLaboral)	24
mensajeCambioHora()	24

TorreDeControl	25
Atributos	25
Métodos	25
TorreDeControl(AtomicInteger hora, Aerolinea[] aerolineas)	25
programasVuelosDelDia()	25
obtenerTerminal(int embarque)	26
cambioDeHora()	26
finalizarVentas(Vuelo vuelo)	26
ordenarDespegue(Vuelo vuelo)	27
verificarCantidadDespegues(int hora)	27
permitirAterrizaje(Vuelo vuelo)	27
actualizarVuelosPosibles()	27
obtenerPasaje (Pasajero pasajero)	28
cartelera()	28
cantVuelosEnHora()	28
ignorarVuelo(Vuelo vuelo)	28
Tren	29
Atributos	29
Métodos	29
Tren (int capacidad)	29
getCapacidad()	30
toString()	30
run()	30
volverAlHall()	30
dejarPasajeros()	31
visitarTerminal(char letra, String emojiT)	31
usarTren(Pasajero pasajero, char terminal)	31
subirAlTren(Pasajero pasajero, char terminal)	31
bajarDelTren(Pasajero pasajero, char terminal)	32
Vuelo	32
Atributos	32
Métodos	33
Vuelo(TorreDeControl torre, char terminal,int [] embarque_hora, Aerolinea aerolinea)	33
getCantidadPasajeros()	33
getId()	33
getEmbarque_hora()	33
getEmbarque()	33
getHora()	34
getTerminal()	34
getAerolinea()	34
toString()	34

sacarPasaje()	34
subirPasajero(Pasajero pasajero)	35
run()	35
embarcar()	35
Anexos	35
Tabla de emojis	35

Introducción

Este trabajo corresponde a la materia de “Programación Concurrente” dictada por la Facultad de Informática de la Universidad Nacional Del Comahue. En el mismo se trata de demostrar el conocimiento de los fundamentos de la materia.

Consigna

Se desea simular el funcionamiento del Aeropuerto "VIAJE BONITO", desde que el pasajero llega al mismo para tomar su vuelo hasta que sube al avión. El aeropuerto siempre tiene sus puertas abiertas, pero atiende a los pasajeros solo de 6.00 hrs. a 22.00 hrs.

Hay n aerolíneas, coexistiendo en dicho aeropuerto (por ejemplo Aerolíneas Argentinas, LAN, etc.), cada una de ellas con su puesto de atención de pasajeros.

Los pasajeros tienen su reserva en un vuelo de una de las aerolíneas del aeropuerto. Cuando un pasajero ingresa al aeropuerto llega a un puesto de informes, donde es atendido y desde allí es derivado al puesto de atención que corresponda según su vuelo y aerolínea para hacer el check-in. Cada puesto de atención tiene lugar para una cantidad max de pasajeros, que son atendidos por orden de llegada. Los pasajeros que llegan cuando el límite max está superado esperan en un hall central hasta que se haga lugar. Además, en los puestos, hay un guardia que se encarga de dar paso a los pasajeros que llegan, a medida que se va desocupando el lugar del puesto.

NO interesa simular el tema de las reservas en una aerolínea particular por lo que la misma, se generará de forma aleatoria al llegar al aeropuerto.

Además en el aeropuerto hay varias terminales. Cada terminal tiene varios puestos de embarque y una sala de embarque compartida.

Cuando un pasajero hace el check-in se le indica la terminal y puesto de embarque que corresponde a su vuelo.

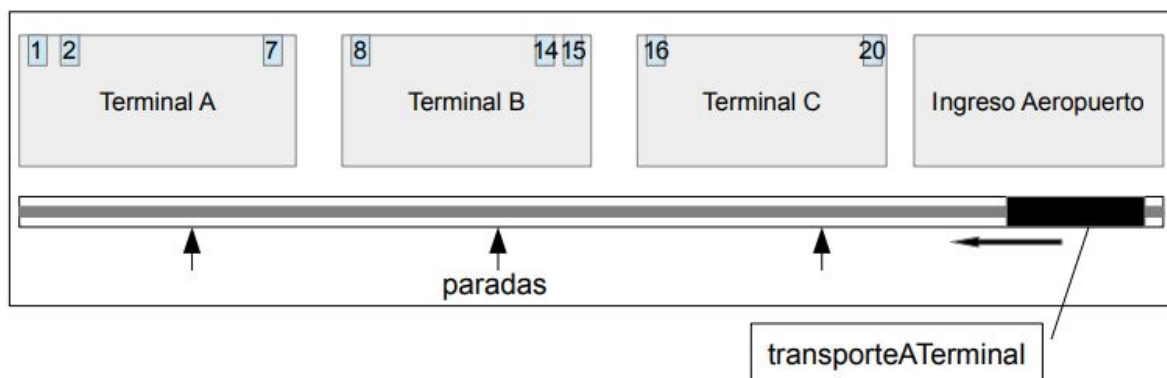
Entonces, los pasajeros que ya hicieron el check-in son trasladados a la terminal que les corresponde y permanecen allí hasta que se haga el llamado para embarcar. Por ejemplo, el aeropuerto "VIAJE BONITO" tiene 3 terminales: A, B y C. En la terminal A están los puestos de embarque 1 a 7, en la terminal B los puestos del 8 a 15 y en la terminal C del 16 a 20.

Por otro lado, en cada terminal hay un free-shop en el que los pasajeros que esperan para embarcar pueden hacer compras, o solo mirar los productos que se ofrecen. Por una cuestión de organización cada free-shop tiene una capacidad para "lugar" personas, es decir que si ya está cubierta la capacidad no podrán ingresar más personas hasta que alguna abandone el free shop y deje el lugar. Hay una puerta de ingreso y una puerta de egreso, y 2 cajas cercanas a la puerta de egreso.

Al llegar a la terminal, los pasajeros pueden ir al free-shop o solamente sentarse a esperar en la sala de embarque general el llamado para embarcar. Es importante tener en cuenta que solo pueden ir al free-shop cuando tengan tiempo suficiente antes de la hora de embarque.

En el aeropuerto hay un “people mover/transporteATerminal” que es como un pequeño tren interno que se utiliza para trasladar a los pasajeros hasta las distintas terminales, que se mueve hacia las terminales por una vía. A lo largo del trayecto encuentra las terminales. En cada terminal se detiene si algún pasajero lo solicita. Al llegar a la última terminal el tren debe quedar vacío, y vuelve al punto de inicio de su trayecto (o sea el ingreso al aeropuerto), allí espera hasta tener su capacidad completa para comenzar un nuevo recorrido.

Debe resolverse utilizando los mecanismos de sincronización vistos en la materia y provistos por el lenguaje: semáforos y monitores (obligatoriamente), Locks, CyclicBarrier, CountdownLatch, Exchanger., implementaciones de BlockingQueue.



Análisis general de recursos activos

Se considerarán recursos activos todos aquellos que serán hilos para este programa de simulación de aeropuerto

Perspectiva del pasajero

1. Un pasajero puede llegar a cualquier hora al aeropuerto, pero solo puede ser atendido de 6 a 22. Entonces al llegar podrán pasar dos situaciones:
 - a. Si es un horario de atención: Es atendido por un puesto de informe y guiado hacia los puestos de atención de la aerolínea que indique su pasaje
 - b. Si no es un horario de atención: Se deberá quedar esperando a que sea un horario de atención para realizar la acción A
2. Una vez que el pasajero sepa donde están los puestos de atención de su aerolínea deberá hacer la fila de esa aerolínea y esperar a ser llamado por un puesto que este libre para hacer el check-in
3. Una vez realizado el check-in deberá ir a la terminal donde aterrizará su vuelo, para hacer esto primero deberá ir a a formarse para subir al tren que lo llevara a su terminal
4. Cuando sea su turno, subirá al tren, una vez que el mismo esté lleno será dejado en su terminal
5. En la terminal, de dar los horarios podrá:
 - a. Ir al freeshop de la terminal: De no estar lleno podrá o no ingresar si lo decidiera, en el mismo elegirá un producto a comprar, formará una fila para pagar y luego se retirara
6. Espera su vuelo hasta que aterrice y abra sus puertas
7. Una vez abiertas las puertas de su vuelo ingresara y esperará que despegue.

Perspectiva del vuelo

1. El vuelo será creado a las 12 AM, asignándole una puerta de embarque y horario, el cual será informada a la torre de control,
2. Una hora antes de su aterrizaje se dejarán de vender pasajes para el vuelo, además cada vez que un pasajero saque un pasaje para el vuelo, el mismo deberá ser informado de tal hecho.
3. Cuando sea su hora, el vuelo pedirá pista a la torre de control, aterrizará y esperará a que suban todos sus pasajeros. En caso de que no haya pasajeros que hayan sacado pasajes para este vuelo, el mismo será enviado a otro aeropuerto, por lo que no aterrizará aquí.
4. Cuando suban todos los pasajeros el vuelo pedirá pista a la torre de control y despegará.

Perspectiva del tren

1. El tren estará esperando en el Hall a que suba su capacidad de pasajeros
2. Una vez que se su capacidad está llena irá desde la terminal A hasta la C dejando pasajeros donde corresponda y volverá al Hall para realizar esta misma acción

Perspectiva del tiempo

1. A las 12 PM programará todos los vuelos del día
2. De 22 a 24 y de 0 a 6 generará una cierta cantidad de pasajeros, menor a la cantidad de pasajeros del horario laboral, que deberá ser múltiplo de la capacidad de pasajeros del tren
3. De 6 a 22 generar generará una cierta cantidad de pasajeros, mayor a la cantidad de pasajeros del horario no laboral, que deberá ser múltiplo de la capacidad de pasajeros del tren
4. En el horario no laboral, cada vez que genere pasajeros luego pasara de hora (de ser la hora 24 reiniciará la hora para que sea 0, es decir que pasará de día)
5. En el horario laboral, primero informará a los aviones que corresponda para que dejen de vender vuelos o aterricen, luego de generar los pasajeros del horario laboral, finalmente deberá esperar a que despeguen todos los vuelos correspondientes a la hora actual que estaban programados previamente para poder pasar de hora.

Análisis general de recursos pasivos

Se considerarán recursos activos todos aquellos que serán los recursos compartidos por los hilos

Aerolinea

Una aerolínea podrá ser referenciada por vuelos, además tendrá puestos de atención que serán usados por los pasajeros para ser enviadas a las terminales que correspondan

Aeropuerto

El aeropuertos por un lado tendrá los puestos de informes que serán utilizados por los pasajeros para ser guiados a los puestos de atención de las aerolíneas que corresponda. Los puestos de informes sólo estarán habilitados en horario laboral, cuando el guardia de los puestos de informe esté presente (a partir de las 6 hasta las 22 inclusive).

CajaFreeShop

Las cajas del free shop atenderán a los clientes del free shop al cual pertenezcan cuando un pasajero lo solicite y estén disponibles

FreeShop

El mismo estará disponible si un pasajero desea acceder a él y él mismo no esté completo en su capacidad.

TorreDeControl

Por un lado la torre de control contendrá el cronograma de vuelos, que permitira a los usuarios sacar pasajes, y eliminar vuelos (cuando sean sacados de la tabla hash en la cual son ingresados al ser programados).

Por otro lado, esta recurso permitirá saber si la cantidad de vuelos programados para una determinada hora han despegado o no.

Por último, esta entidad permitirá a los vuelos aterrizar y despegar, de a uno, de esta forma se mantendrá la consistencia en la cantidad de vuelos por hora (los programados y los que han despegado).

Clases

Aerolinea

Atributos

- De instancia
 - nombre (Tipo String): Nombre de la aerolínea
 - puestoDeAtencion (Tipo Semaphore): Simbolizan los puestos de atención de la aerolínea.
 - guardia (Tipo Lock): Simboliza el guardia de los puestos de atención, permitirá que los pasajeros se formen de manera ordenada.
- Estáticos:
 - emojis (Tipo String[]): Conjunto de emojis que serán usados de manera regular por distintos mensajes de la clase

Métodos

Aerolinea (String nombre, int cantidadDePuestosDeAtencion)

Método público de instancia.

Es el constructor de la clase, generará un guardia para la aerolínea generada, la cual deberá darse un nombre por parámetros y se deberá especificar la cantidad de puestos de atención que tendrá

```
public Aerolinea (String nombre, int cantidadDePuestosDeAtencion) {  
    this.nombre=nombre;  
    this.guardia=new ReentrantLock();  
    this.puestosDeAtencion=new Semaphore(cantidadDePuestosDeAtencion);  
}
```

getNombre()

Método público de instancia.

Permite obtener el nombre de la aerolínea

Retorna el nombre de la aerolínea como String.

```
public String getNombre() {  
    return this.nombre;  
}
```

setNombre(String nombre)

Método público de instancia.

Permite asignar un nombre a la aerolínea

No retorna nada

```
public void setNombre(String nombre) {  
    this.nombre=nombre;  
}
```

toString()

Método público de instancia.

Permite obtener el nombre de la aerolínea junto a la palabra "AEROLINEA"

Retorna un String

```
public String toString() {  
    return "AEROLINEA "+this.nombre;  
}
```

usarPuestoDeAtencion (Pasajero pasajero)

Método público de instancia.

Permite al pasajero enviado por parámetro usar un puesto de atención de la aerolínea.

No retorna nada.

```
public void usarPuestoDeAtencion (Pasajero pasajero) {  
    //el pasajero se forma en la fila  
    this.guardia.lock();  
    System.out.println("[ "+emojis[0]+" FILA ATENCIÓN "+this.nombre+": El "+pasajero.toString()+" se ha formado ");  
    this.guardia.unlock();  
  
    try {  
        //Aqui voy a ver si hay un puesto de atención disponible  
        this.puestosDeAtencion.acquire(); catch (InterruptedException e) {}  
        System.out.println("[ "+emojis[1]+" ATENCIÓN "+this.nombre+": El "+pasajero.toString()+" está siendo  
atendido");  
        try {  
            //Simulo la atencion  
            Thread.sleep(2000);  
        } catch (InterruptedException e) {}  
        System.out.println("[ "+emojis[2]+" ATENCIÓN "+this.nombre+": El "+pasajero.toString()+" es mandado al  
EMBARQUE "+pasajero.getPasaje().getEmbarque()+" (TERMINAL "+pasajero.getPasaje().getTerminal()+"");  
        //libero el puesto de atención  
        this.puestosDeAtencion.release();  
    }  
}
```

Aeropuerto

Atributos

De instancia:

- freeShops (Tipo FreeShop[]): Es el conjunto de los Free Shops del Aeropuerto, habiendo uno por cada terminal
- torre (Tipo TorreDeControl): La torre que coordinará la obtención de pasajes, los aterrizajes y despegues de los vuelos.
- puestosDeInformeDisponibles (Tipo Semaphore): Los mismos guiarán a los pasajeros a los puestos de atención de la aerolínea que corresponda según su pasaje
- turnoPuestosDeInforme(Tipo Semaphore): Este semaforo binario permitirá que los pasajeros usen los puestos de informes durante el horario laboral.

Métodos

Aeropuerto (Tren tren, int cantidadDePuestosDeInformes, int capacidadFS, int cantCajasFS)

Método público

Constructor de la clase

Método para crear una instancia de Aeropuerto, requiere un tren, que se especifique la cantidad de puestos de informes, la capacidad de los Free Shops y la cantidad de cajas que tendrán los FreeShops (asumimos que todos los Free Shops tienen la misma cantidad de cajas). Por la consigna, solo habrá 3 Free Shops, uno por cada terminal.

```
public Aeropuerto (Tren tren, int cantidadDePuestosDeInformes, int capacidadFS, int cantCajasFS) {
    this.tren=tren;
    this.puestosDeInformesDisponibles=new Semaphore(cantidadDePuestosDeInformes);
    this.turnoPuestoDeInforme=new Semaphore(0);
    this.freeShops=new FreeShop[3];

    this.freeShops[0]=new FreeShop(capacidadFS,'A',cantCajasFS,"\uD83C\uDD70");
    this.freeShops[1]=new FreeShop(capacidadFS,'B',cantCajasFS,"\uD83C\uDD71");
    this.freeShops[2]=new FreeShop(capacidadFS,'C',cantCajasFS,"\uD83C\uDD72");
}
```

setTorre(AtomicInteger hora, Aerolinea[] aerolineas)

Método público de instancia.

No retorna nada.

Asigna una torre de control nueva al aeropuerto

```
public void setTorre(AtomicInteger hora, Aerolinea[] aerolineas) {
    this.torre=new TorreDeControl(hora,aerolineas);
}
```

getTren()

Método público de instancia.

Retorna el tren del aeropuerto.

```
public Tren getTren() {
    return this.tren;
}
```

pedirPuestoDeInforme(Pasajero pasajero)

Método público de instancia.

No retorna nada.

Sirve para que un pasajero, enviado por parámetro, obtenga un pasaje de avión y sea designado a la aerolínea correspondiente a su vuelo.

```
public void pedirPuestoDeInforme(Pasajero pasajero) {
    //El pasajero se forma en la fila
    System.out.println("\uD83D\uDC6E FILA PUESTO INFORME: "+pasajero.toString()+" se ha formado");
    try {
        this.turnoPuestoDeInforme.acquire();} catch (InterruptedException e) {}
    //El pasajero es atendido por un puesto de informe
    try {
        this.puestosDeInformesDisponibles.acquire();} catch (InterruptedException e) {}
}
```

```

//Le digo al anterior pasajero que avance un lugar
this.turnoPuestoDelInforme.release();

System.out.println("\uD83D\uDEC9 \uD83D\uDEC8 "+pasajero.toString()+"- Esta siendo atendido por un puesto
de informes");
//Se me da un vuelo (pues no interesa modelar el tema de reserva)
try {
    //simulo atención
    Thread.sleep(1000);
} catch (InterruptedException e) {}
this.torre.obtenerPasaje(pasajero);
this.puestosDelInformesDisponibles.release();
}

```

pasarTiempo()

Método público de instancia.

Retorna vacío.

Permite actualizar los vuelos disponibles para dar pasajes.

```

public void pasarTiempo() {
    this.torre.actualizarVuelosPosibles();
}

```

programarVuelosDelDia()

Método público de instancia.

Retorna vacío.

Permite programar los vuelos del día.

```

public void programarVuelosDelDia() {
    this.torre.programasVuelosDelDia();
}

```

comenzarDiaLaboral()

Método público de instancia.

Retorna vacío.

Permite la atención de los puestos de informes, y por lo tanto que el aeropuerto comience con su flujo normal de trabajo durante horario laboral.

```

public void comenzarDiaLaboral() {
    //abrimos los puestos de informe, y habilitamos todo el proceso
    System.out.println("AEROPUERTO: Inicio del dia laboral, se abren los puestos de informe, atención y freeshops
al público, el conductor del tren llega a tiempo como siempre");
    this.turnoPuestoDelInforme.release();
}

```

terminarDiaLaboral()

Método público de instancia.

Retorna vacío.

No permite la atención de los puestos de informes, y por lo tanto que el aeropuerto interrumpa su ciclo de trabajo de horario laboral.

```

public void terminarDiaLaboral() {
    //cerramos el puesto de informes, de esta manera evitamos
    //el flujo de procesos y hacemos que todo cierre
}

```

```
try {this.turnoPuestoDelInforme.acquire();} catch (InterruptedException e) {}
System.out.println("AEROPUERTO: Todo el personal va a dormir luego de una larga jornada laboral");
}
```

usarTren(Pasajero pasajero)

Método público de instancia.

Retorna vacío.

Permite a los pasajeros usar el tren del aeropuerto

```
public void usarTren(Pasajero pasajero) {
    this.tren.usarTren(pasajero, pasajero.getPasaje().getTerminal());
}
```

comprarFreeShop(Pasajero pasajero)

Método público de instancia.

Retorna vacío.

Permite a los pasajeros comprar en el Free Shop de la terminal donde se encuentre.

```
public void comprarFreeShop(Pasajero pasajero) {
    char terminal=pasajero.getPasaje().getTerminal();
    this.freeShops[(int) terminal-65].comprar(pasajero);
}
```

verificarDespegues(int hora)

Método público de instancia.

Retorna vacío.

Permite a la torre de control verificar que todos si los vuelos de la hora enviada por parámetro que hubieran estado programados, efectivamente despegaron.

```
public void verificarDespegues(int hora) {
    if(hora>6&& hora<23) {
        this.torre.verificarCantidadDespegues(hora);
    }
}
```

cantVuelosEnHora()

Método público de instancia.

Retorna vacío.

Permite obtener la cantidad de vuelos que faltan despegar en un determinada hora (que la torre de control sabe).

```
public int cantVuelosEnHora() {
    return this.torre.cantVuelosEnHora();
}
```

CajaFreeShop

Atributos

De instancia

- id (tipo int): Identificador de la caja
- terminal (tipo char): Letra de la terminal donde se encuentra el freeshop

- emoji (tipo String): Cadena de caracteres que permite obtener el emoji de la caja
De clase:
- cant (Tipo AtomicInteger): contador de cantidad de cajas que tiene el aeropuerto.

Métodos

CajaFreeShop(char terminal, String emoji2)

Método público

Constructor de la clase

Método para crear una instancia de Caja de Free Shop, el emoji enviado por parámetro será la uno correspondiente a la letra de la terminal a la cual pertenece la caja.

```
public CajaFreeShop(char terminal, String emoji2) {
    this.id=cant.addAndGet(1);
    this.terminal=terminal;
    this.emoji="\ud83d\udecd\u201c"+emoji2;
}
```

toString()

Método público de instancia

Retorna un String

El String retornado permite identificar la caja y en qué terminal se encuentra.

```
public String toString() {
    return "CAJA "+this.id+ "(TERMINAL "+this.terminal+")";
}
```

atenderPasajero(Pasajero pasajero)

Método público de instancia.

No retorna nada.

Permite a la caja atender al pasajero enviado por parámetro.

```
public void atenderPasajero(Pasajero pasajero) {
    System.out.println("[ "+this.emoji+" "+this.toString()+"]: esta atendiendo al "+pasajero.toString());
    try {
        //simulo atención
        Thread.sleep(1000);
    } catch (InterruptedException e) {}
    System.out.println("[ "+this.emoji+" "+this.toString()+"]: termino de atender al "+pasajero.toString());
}
```

FreeShop

Atributos

De instancia:

- terminal (Tipo char): Letra de la terminal a la cual pertenece el Free Shop
- cajas (Tipo ArrayBlockingQueue<CajaFreeShop>): Cajas del Free Shop que atenderán a los pasajeros que decidan comprar
- permisosDeEntrada (Semaphore): Semáforo con una cantidad de permisos igual a la capacidad del Free Shop.
- emoji (Tipo String): Emoji para identificar mensajes por pantalla

De clase:

- probabilidadCompra (Tipo double): Probabilidad que un pasajero decida entrar a comprar al free shop si el mismo no tiene completa su capacidad

Métodos

FreeShop(int capacidad, char terminal, int cantCajas, String emoji)

Método público.

Constructor de la clase.

Método para crear una instancia de Free Shop, el emoji enviado por parámetro será la uno correspondiente a la letra de la terminal a la cual pertenece la caja, la capacidad sirve para determinar la cantidad de permisos que tendrá el semáforo de “permisosDeEntrada”, la cantidad de cajas servirá para crear la ArrayBlockingQueue<CajaFreeShop>.

```
public FreeShop(int capacidad, char terminal, int cantCajas, String emoji) {
    this.terminal=terminal;
    this.cajas=new ArrayBlockingQueue<CajaFreeShop>(cantCajas);
    this.permisosDeEntrada=new Semaphore(capacidad);
    this.emoji="\uD83D\uDED2 "+emoji;

    for(int i=0;i<cantCajas;i++) {
        try {
            this.cajas.put(new CajaFreeShop(this.terminal, emoji));
        } catch (InterruptedException e) {}
    }
}
```

toString()

Método público de instancia.

Retorna un String.

El String retornado informa a qué terminal pertenece el Free Shop.

```
public String toString() {
    return "FREE SHOP "+this.terminal;
}
```

comprar(Pasajero pasajero)

Método público de instancia.

No retorna nada.

Método que sirve para que el pasajero enviado por parámetro realice una compra en el Free Shop, siempre y cuando haya capacidad disponible.

```
public void comprar(Pasajero pasajero) {
    CajaFreeShop caja=null;
    if(Math.random()<=probabilidadCompra) {
        System.out.println("[ "+this.emoji+" "+this.toString()+"]: El "+pasajero.toString()+" trata de ingresar");
        if(this.permisosDeEntrada.tryAcquire()) {
            System.out.println("[ "+this.emoji+" "+this.toString()+"]: El "+pasajero.toString()+" ingreso tranquilamente");
            try {
                //Simulo la busqueda de productos
                Thread.sleep(1000);
                System.out.println("[ "+this.emoji+" \uD83D\uDE12 "+"FILA CAJAS "+this.toString()+"]: El "+pasajero.toString()+" eligió sus productos y fue a formarse en la fila");
                caja=this.cajas.take();
                caja.atenderPasajero(pasajero);
                System.out.println("[ "+this.emoji+" "+this.toString()+"]: El "+pasajero.toString()+" se retira del lugar para esperar su vuelo");
            }
        }
    }
}
```

```

        } catch (InterruptedException e) {}
        this.permisosDeEntrada.release();
        try {
            this.cajas.put(caja);
        } catch (InterruptedException e) {}
    } else {
        System.out.println("[ "+this.emoji+" "+this.toString()+"]: Capacidad agotada,
        "+pasajero.toString()+" decide irse a dormir una siesta hasta que llegue su vuelo");
    }
}
}

```

Pasajero

Atributos

De instancia:

- idPasajero (Tipo int): Identificador numérico del pasajero.
- pasaje (Tipo Vuelo): Vuelo el cual deberá abordar.
- aeropuerto (Tipo Aeropuerto): Aeropuerto donde deberá realizar todas las tareas necesarias para poder abordar su vuelo.
- hora (Tipo AtomicInteger): Hora del aeropuerto.

De clase:

- cant (AtomicInteger): Cantidad de pasajeros creados.

Métodos

Pasajero(AtomicInteger hora, Aeropuerto aeropuerto)

Método público.

Constructor de la clase.

Método para crear una instancia de Pasajero.

```

public Pasajero(AtomicInteger hora, Aeropuerto aeropuerto) {
    this.idPasajero=cant.incrementAndGet();
    this.hora=hora;
    this.aeropuerto=aeropuerto;
}

```

setPasaje(Vuelo vuelo)

Método público de instancia.

No retorna nada.

Permite asignar un vuelo a un pasajero.

```

public void setPasaje(Vuelo vuelo) {
    this.pasaje=vuelo;
}

```

getHoraActual()

Método público de instancia.

Retorna un int.

Retorna la hora actual.

```

public int getHoraActual() {

```

```
return this.hora.get();  
}
```

getAeropuerto()

Método público de instancia.

Retorna un Aeropuerto.

Retorna el aeropuerto donde se encuentra el pasajero.

```
public Aeropuerto getAeropuerto() {  
    return this.aeropuerto;  
}
```

getIdPasajero()

Método público de instancia.

Retorna un int.

Permite obtener el id del pasajero.

```
public int getIdPasajero() {  
    return this.idPasajero;  
}
```

getPasaje()

Método público de instancia.

Retorna un Vuelo.

Permite obtener el vuelo el cual designado al pasajero.

```
public Vuelo getPasaje() {  
    return this.pasaje;  
}
```

toString()

Método público de instancia.

Retorna un String.

Permite obtener una cadena de caracteres junto con el id del pasajero.

```
public String toString() {  
    return "PASAJERO "+this.idPasajero;  
}
```

run()

Método público de instancia.

No retorna nada.

Explicado en el análisis general desde la perspectiva del pasajero.

```
public void run() {  
    //Llega al aeropuerto y va al puesto de informes (Allí se le asignara el vuelo)  
    System.out.println("\uD83D\uDEC9 "+this.toString()+" : Ha llegado al aeropuerto");  
    this.aeropuerto.pedirPuestoDeInforme(this);  
    //Va a su puesto de atención  
    if(this.pasaje!=null) {  
        //seria null si se equivoco de aeropuerto  
        this.pasaje.getAerolinea().usarPuestoDeAtencion(this);  
        //Va a espera el tren y va su terminal  
    }  
}
```

```

        this.aeropuerto.usarTren(this);

        this.comprarEnFreeShop();

        this.pasaje.subirPasajero(this);
        //Espera el avión y se va
    }
}

```

comprarEnFreeShop()

Método privado de instancia

No retorna nada.

Permite al pasajero ingresar al Free Shop a comprar.

```

private void comprarEnFreeShop() {
    //Trato de ingresar al Freeshop si me da el horario
    //En esta caso es si el pasajero tiene más de una hora de espera
    if(this.hora.get()<this.pasaje.getHora()) {
        this.aeropuerto.comprarFreeShop(this);
    }
}

```

Tiempo

Atributos

De instancia:

- capacidadTren (Tipo int): cantidad de pasajeros que puede tener el tren del Aeropuerto.
- factorHL (Tipo int): un número entero que permite crear pasajeros durante el horario laboral.
- factorHNL (Tipo int): un número entero que permite crear pasajeros durante el horario no laboral.
- hora (Tipo AtomicInteger): Entero que representara la hora.
- aeropuerto (Tipo Aeropuerto): Aeropuerto que será coordinado por la instancia de Tiempo.
- esTest (Tipo boolean): Booleano que permitirá saber si el hilo se está corriendo como Test o no. En caso que sea True, durante las hora laboral habrá una pausa en su inicio, que el usuario deberá introducir un carácter para poder continuar con la ejecución normal del hilo.

Métodos

Tiempo (Aeropuerto aeropuerto, Aerolinea[] aerolineas, int capacidadTren, int factorHL, int factorHNL, boolean esTest)

Método público.

Constructor de la clase.

Método para crear una instancia de Tiempo.

```

public Tiempo (Aeropuerto aeropuerto, Aerolinea[] aerolineas, int capacidadTren, int factorHL, int factorHNL,boolean
esTest) {
    this.hora=new AtomicInteger(0);
}

```

```
this.capacidadTren=capacidadTren;  
this.factorHL=factorHL;  
this.factorHNL=factorHNL;  
this.aeropuerto=aeropuerto;  
this.aeropuerto.setTorre(this.hora, aerolineas);  
this.esTest=esTest;  
}
```

getHora()

Método público de instancia.

Retorna un int.

Retorna la hora actual.

```
public int getHora() {  
    return this.hora.get();  
}
```

iniciarDiaLaboral()

Método privado de instancia.

No retorna nada.

Permite iniciar el horario laboral del aeropuerto.

```
private void iniciarDiaLaboral() {  
    this.aeropuerto.comenzarDiaLaboral();  
}
```

terminarDiaLaboral()

Método privado de instancia.

No retorna nada.

Permite finalizar el horario laboral del aeropuerto.

```
private void terminarDiaLaboral() {  
    this.aeropuerto.terminarDiaLaboral();  
}
```

hacerAndarTren()

Método privado de instancia.

No retorna nada.

Permite iniciar el hilo del Tren.

```
private void hacerAndarTren() {  
    (new Thread(this.aeropuerto.getTren(), "TREN")).start();  
}
```

run()

Método público de instancia.

No retorna nada.

1. Hace andar el tren
2. Establece la hora en 0
3. Programa los vuelos del día

4. El resto de lo que hace esta explicado en el análisis general desde la perspectiva del Tiempo.

```
public void run() {  
  
    this.hacerAndarTren();  
    while(true) {  
        this.hora.set(0);  
        this.aeropuerto.programarVuelosDelDia();  
        System.out.println();  
        while(this.hora.get()<25) {  
            this.mensajeCambioHora();  
            this.aeropuerto.pasarTiempo();  
            if(this.hora.get()<6 || this.hora.get()>22) {  
                if(this.esTest) {  
                    this.mensajeTest(false);  
                }  
                if(this.getHora()==23) {  
                    this.terminarDiaLaboral();  
                }  
                this.pasarHoraNoLaboral();  
            }else {  
                if(this.esTest) {  
                    this.mensajeTest(true);  
                }  
                if(this.getHora()==6) {  
                    this.iniciarDiaLaboral();  
                }  
                this.pasarHoraLaboral();  
            }  
            try {  
                Thread.sleep(10000);  
            } catch (InterruptedException e) {}  
        }  
  
        System.out.println("\n-----\n");  
        this.hora.addAndGet(1);  
    }  
}
```

mensajeTest(boolean esHL)

Método privado de instancia

No retorna nada.

Permite detener la ejecución del hilo hasta que el usuario que ejecuta el test desee continuar con la ejecución

```
private void mensajeTest(boolean esHL) {  
    char aux;  
    if(esHL) {  
        try {  
            Thread.sleep(5000);  
        } catch (InterruptedException e) {}  
    }else {  
        try {  
            Thread.sleep(1000);  
        } catch (InterruptedException e) {}  
    }  
  
    do {  
        System.out.println("Introduzca Y para continuar: ");  
        aux=TecladoIn.readNonwhiteChar();  
    }while(aux!='Y' && aux!='y');  
}
```

pasarHoraLaboral()

Método privado de instancia.

No retorna nada.

Permite generar pasajeros y verificar si todos los vuelos programados para la hora actual realmente han despegado.

```
private void pasarHoraLaboral() {
    this.generarPasajeros(true);
    //verifico que todos los vuelos programados
    //para esta hora hayan despegado
    this.aeropuerto.verificarDespegues(this.hora.get());
}
```

pasarHoraNoLaboral()

Método privado de instancia.

No retorna nada.

Permite generar pasajeros.

```
private void pasarHoraNoLaboral() {
    this.generarPasajeros(false);
}
```

generarPasajeros(boolean horaLaboral)

Método privado de instancia.

No retorna nada.

Permite generar pasajeros de forma aleatoria, siempre y cuando sean múltiplos de la capacidad de tren (para evitar deadlocks del hilo del tren, pues el mismo solo puede seguir su ejecución normal si el tren se encuentra lleno). La cantidad de pasajeros que puede generar varía según se trate de una hora laboral o no.

```
private void generarPasajeros(boolean horaLaboral) {
    Pasajero pasajero;
    int cantPasajeros;
    if(horaLaboral) {
        cantPasajeros=this.capacidadTren* ((int) (Math.random()*this.factorHL+1));
    }else {
        cantPasajeros=this.capacidadTren* ((int) (Math.random()*(1+this.factorHNL)));
    }
    System.out.println("Cantidad de pasajeros: "+cantPasajeros);
    if(cantPasajeros!=0) {
        for (int i=0; i<cantPasajeros;i++) {
            pasajero=new Pasajero (this.hora,this.aeropuerto);
            (new Thread (pasajero,pasajero.toString())).start();
        }
    }
}
```

mensajeCambioHora()

Método privado de instancia.

No retorna nada.

Imprime por consola la hora actual y la cantidad de vuelos programados para la hora impresa.


```
private void mensajeCambioHora() {
    System.out.println("HORA "+this.hora.get()+" (Cantidad de Vuelos: "+this.aeropuerto.cantVuelosEnHora()+");");
}
```

TorreDeControl

Atributos

De instancia:

- hora (AtomicInteger): Indica la hora actual.
- vuelosDelDia (HashMap<int[], Vuelo>): Simboliza el cronograma de los vuelos del día, donde la clave son el número de embarque y la hora de despegue.
- vuelosPosibles (Vuelo []): Son los vuelos para los cuales los pasajeros pueden sacar pasajes, se actualizan cada 1 hora
- aerolineas (Tipo Aerolineas[]): Aerolineas posibles para los vuelos
- cantVuelos (Tipo int[]): Cantidad de vuelos por hora pendientes de despegue.

De clase:

- probabilidadVuelo: probabilidad para que se genera un vuelo a una determinada hora en un determinado embarque

Métodos

TorreDeControl(AtomicInteger hora, Aerolineas[] aerolineas)

Método público.

Constructor de la clase.

Método para crear una instancia de torre de control.

```
public TorreDeControl(AtomicInteger hora, Aerolineas[] aerolineas) {
    this.hora=hora;
    this.vuelosDelDia=new HashMap<int[],Vuelo>(15*25); //15 HL x 25 PE
    this.vuelosPosibles=null;
    this.aerolineas=aerolineas;
    this.cantVuelos=new int[16];

    for(int i=0;i<this.cantVuelos.length;i++) {
        this.cantVuelos[i]=0;
    }
}
```

programasVuelosDelDia()

Método público de instancia.

No retorna nada.

Permite programar de forma aleatoria los vuelos del día según la hora y en número de embarque.

```
public synchronized void programasVuelosDelDia() {
    //la clave sera embarque y hora
    Vuelo vuelo;
    char terminal;
    int[] e_h;
    for (int h=7; h<=22;h++) {
        for (int e=1; e<=25; e++) {
            if (Math.random()<=probabilidadVuelo) {
                terminal=this.obtenerTerminal(e);
            }
        }
    }
}
```

```

        e_h=new int[2];
        e_h[0]=e;
        e_h[1]=h;
        vuelo=new
Vuelo(this,terminal,e_h,this.aerolineas[(int)(Math.random()*this.aerolineas.length)]);
        this.vuelosDelDia.put(e_h, vuelo);
        this.cantVuelos[h-7]++;
        new Thread(vuelo,vuelo.toString()).start();
    }
}
this.cartelera();
}

```

obtenerTerminal(int embarque)

Método privado de instancia.

Retorna un char.

Permite obtener la terminal de un embarque, la misma es definida según lo dado en la consigna.

```

private char obtenerTerminal(int embarque) {
    char terminal='';
    if(embarque>=1 && embarque<=7) {
        terminal='A';
    }else if (embarque<=15){
        terminal='B';
    }else {
        terminal='C';
    }

    return terminal;
}

```

cambioDeHora()

Método privado de instancia.

No retorna nada.

Se envían notificaciones a los métodos del monitor.

```

private void cambioDeHora() {
    this.notifyAll();
}

```

finalizarVentas(Vuelo vuelo)

Método público de instancia.

No retorna nada.

Permite al vuelo enviado por parámetro finalizar sus ventas, esto implica que es eliminado del HashMap “vuelosDelDia”.

```

public synchronized void finalizarVentas(Vuelo vuelo) {
    //método para no permitir más ventas del
    //vuelo enviado por parametro
    while(this.hora.get()!=(vuelo.getEmbarque_hora()[1]-1)) {
        //Espero a la hora anterior para dejar de vender pasajes
        //Ejemplo: Si el vuelo sale a las 9, los pasajes se dejan
        //de vender a las 8
        try {
            wait();} catch (InterruptedException e) {}
        }
    this.vuelosDelDia.remove(vuelo.getEmbarque_hora());
    System.out.println("\u2708 \u274c "+vuelo.toString()+" : Ha finalizado sus ventas");
}

```

```
}
```

ordenarDespegue(Vuelo vuelo)

Método público de instancia.

No retorna nada.

Ordena el despegue del vuelo enviado por parámetro.

```
public synchronized void ordenarDespegue(Vuelo vuelo) {
    System.out.println("\uD83D\uDEE7 \uD83D\uDEEB DESPEGUE "+vuelo.toString()+"": ¡No vuelven más! (¡Bon Voyage!"));
    this.cantVuelos[vuelo.getHora()-7]--;
    if(this.cantVuelos[vuelo.getHora()-7]==0) {
        notifyAll();
    }
}
```

verificarCantidadDespegues(int hora)

Método público de instancia.

No retorna nada.

Verifica que en la hora enviada por parámetro hayan despegado todos los vuelos programados.

```
public synchronized void verificarCantidadDespegues(int hora) {
    //espero hasta que despeguen todos los vuelos
    while(this.cantVuelos[hora-7]!=0) {
        try {
            wait();
        } catch (InterruptedException e) {}
    }
    //Una vez que todos despegaron, podría pasar de hora
}
```

permitirAterrizaje(Vuelo vuelo)

Método público de instancia.

Retorna un boolean

Permite aterrizar al vuelo enviado por parámetro.

```
public synchronized boolean permitirAterrizaje(Vuelo vuelo) {
    //no permito aterrizar hasta que no sea la hora adecuada
    boolean exito;
    while(this.hora.get()!=vuelo.getHora()) {
        try {
            wait();
        } catch (InterruptedException e) {}
    }
    if(vuelo.getCantidadPasajeros()!=0) {
        System.out.println("\uD83D\uDEEC ATERRIZAJE "+vuelo.toString()+"": Ha aterrizado");
        exito=true;
    } else {
        this.ignorarVuelo(vuelo);
        exito=false;
    }
    return exito;
}
```

actualizarVuelosPosibles()

Método público de instancia.

No retorna nada.

Actualiza los vuelos posibles para que los pasajeros puedan obtener un pasaje.

```
public synchronized void actualizarVuelosPosibles() {  
    this.vuelosPosibles=new Vuelo[this.vuelosDelDia.size()];  
    this.vuelosDelDia.values().toArray(this.vuelosPosibles);  
    this.cambioDeHora();  
}
```

obtenerPasaje (Pasajero pasajero)

Método público de instancia.

No retorna nada.

Permite al pasajero enviado por parámetro obtener un pasaje.

```
public synchronized void obtenerPasaje (Pasajero pasajero) {  
    if(this.vuelosPosibles.length!=0) {  
        pasajero.setPasaje(this.vuelosPosibles[(int)(Math.random()*this.vuelosPosibles.length) ]);  
        pasajero.getPasaje().sacarPasaje();  
        System.out.println("\uD83D\uDEC9 \uD83D\uDEC8 \u2714 "+pasajero.toString()+" Ha sido derivado al  
puesto de atención correspondiente por tener el "+pasajero.getPasaje().toString());  
    }else {  
        System.out.println("\uD83D\uDEC9 \ud83d\udd02 "+pasajero.toString()+" se ha equivocado  
de aeropuerto y se retira muy apurado");  
        this.notifyAll();  
    }  
}
```

cartelera()

Método público de instancia.

No retorna nada.

Imprime por pantalla los vuelos programados.

```
public void cartelera() {  
    Vuelo[] cartelera=new Vuelo[this.vuelosDelDia.size()];  
    this.vuelosDelDia.values().toArray(cartelera);  
    for(int i=0;i<cartelera.length;i++) {  
        System.out.println("\u2708 \u2708 " +cartelera[i].toString());  
    }  
}
```

cantVuelosEnHora()

Método público de instancia.

Retorna int.

Permite saber cuántos vuelos están pendientes de despegue en la hora actual.

```
public synchronized int cantVuelosEnHora() {  
    int cant=0;  
    if(this.hora.get().>6 && this.hora.get().<23) {  
        cant=this.cantVuelos[this.hora.get()-7];  
    }  
    return cant;  
}
```

ignorarVuelo(Vuelo vuelo)

Método privado de instancia.

No retorna nada.

Este método permite al vuelo enviado por parámetro continuar su recorrido si nadie a sacado pasajes para el mismo.

```
private void ignorarVuelo(Vuelo vuelo) {
    System.out.println("\uD83D\uDEE7 \u274c CONTINUA "+vuelo.toString()+"]: Continua su camino porque nadie
ha sacado pasajes para este vuelo (" +vuelo.getCantidadPasajeros()+")");
    this.cantVuelos[vuelo.getHora()-7]--;
    if(this.cantVuelos[vuelo.getHora()-7]==0) {
        notifyAll();
    }
}
```

Tren

Atributos

De instancia:

- letraTerminal (Tipo char): Simboliza la letra de la terminal en la cual se encuentra el tren. ‘ ‘ representará al Hall Central.
- pasajeroPorTerminal (Tipo int[]): Un arreglo de 3 enteros, que llevará contadores de cuantos pasajeros faltan bajar en cada terminal (siendo el primero correspondiente a la terminal A, el segundo a la terminal B y el tercero a la terminal C).
- asientosOcupados (Tipo int): La cantidad de asientos ocupados por pasajeros del tren.
- capacidad (Tipo int): La cantidad de pasajeros que puedo llevar el tren
- guardiaTren (Tipo Lock): Simboliza un guardia que permite el ascenso y descenso de pasajeros.
- esperarHall (Tipo Condition): Permite a los pasajeros esperar a que el tren llegue al Hall Central y que puedan ascender si hay lugar o seguir esperando.
- esperarEnTren (Tipo Condition): Permite a los pasajeros esperar dentro del tren hasta llegar a su terminal destino.
- avanzarTren (Tipo Condition): Permite al tren avanzar y seguir su recorrido.

De clase:

- emoji (Tipo String): Emoji que será usado para emitir mensajes del tren por consola.

Métodos

Tren (int capacidad)

Método público.

Constructor de la clase.

Método para crear una instancia de Tren.

```
public Tren (int capacidad) {
    this.letraTerminal=' ';

    this.pasajerosPorTerminal=new int[3];

    this.asientosOcupados=0;
    this.capacidad=capacidad;

    this.guardiaTren=new ReentrantLock();
    this.esperarHall=this.guardiaTren.newCondition();
    this.esperarEnElTren=this.guardiaTren.newCondition();
}
```

```

        this.avanzarTren=this.guardiaTren.newCondition();

        for (int i=0;i<this.pasajerosPorTerminal.length;i++) {
            this.pasajerosPorTerminal[i]=0;
        }
    }

```

getCapacidad()

Método público de instancia

Retorna un int.

Permite saber la capacidad de pasajeros del tren.

```

public int getCapacidad() {
    return this.capacidad;
}

```

toString()

Método público de instancia

Retorna un int.

Permite obtener la cadena "TREN".

```

public String toString() {
    return "TREN";
}

```

run()

Método público de instancia

No retorna nada.

Permite al tren realizar su tarea, la cual es explicada en el análisis general desde la perspectiva del tren.

```

public void run() {
    while (true) {
        this.guardiaTren.lock();
        System.out.println("[ "+emoji+" \uD83C\uDD37 "+this.toString()+"]: Aguarda pasajeros");
        if (this.letraTerminal==' ' && this.asientosOcupados!=10) {
            this.esperarHall.signalAll();
        }
        if(this.asientosOcupados!=10) {
            try {
                this.avanzarTren.await();
            } catch (InterruptedException e) {}
        }
        this.dejarPasajeros();
        this.volverAlHall();
    }
}

```

volverAlHall()

Método privado de instancia

No retorna nada.

Permite al tren volver al Hall.

```

private void volverAlHall() {
    System.out.println("[ "+emoji+" \u2B31 TREN]: vuelve al Hall central");
    this.letraTerminal=' ';
}

```

```
}
```

dejarPasajeros()

Método privado de instancia

No retorna nada.

Permite al tren recorrer las terminales e ir dejando a los pasajeros donde corresponda.

```
private void dejarPasajeros() {  
    this.visitarTerminal('A', "\uD83C\uDD70");  
    this.visitarTerminal('B', "\uD83C\uDD71");  
    this.visitarTerminal('C', "\uD83C\uDD72");  
}
```

visitarTerminal(char letra, String emojiT)

Método privado de instancia

No retorna nada.

El tren avisa a sus pasajeros que ha llevado a una nueva terminal y los que deban bajarse se bajan, luego el tren continúa su recorrido.

```
private void visitarTerminal(char letra, String emojiT) {  
    System.out.println("[ "+emoji+" "+emojiT+" TREN]: Ha llegado a la terminal "+letra+" (Se bajan  
    "+this.pasajerosPorTerminal[(int) letra-65]+")");  
    this.letraTerminal=letra;  
    this.esperarEnElTren.signalAll();  
    while(this.pasajerosPorTerminal[(int) letra-65]!=0) {  
        try {  
            this.avanzarTren.await();  
        } catch (InterruptedException e) {}  
    }  
    System.out.println("[ "+emoji+" \u2B8A TREN]: continua su recorrido");  
}
```

usarTren(Pasajero pasajero, char terminal)

Método público de instancia

No retorna nada.

Permite a los pasajeros enviados por parámetro usar el tren y bajarse en la terminal que enviada por parámetro.

```
public void usarTren(Pasajero pasajero, char terminal) {  
    this.subirAlTren(pasajero, terminal);  
  
    if(this.asientosOcupados==this.capacidad) {  
        this.avanzarTren.signalAll();  
    }  
  
    while(this.letraTerminal!=terminal) {  
        try {  
            this.esperarEnElTren.await();  
        } catch (InterruptedException e) {}  
    }  
    this.bajarDelTren(pasajero, terminal);  
}
```

subirAlTren(Pasajero pasajero, char terminal)

Método privado de instancia

No retorna nada.

Permite a los pasajeros enviados por parámetro subir al tren y esperar a llegar a la terminal enviada por parámetro.

```
private void subirAlTren(Pasajero pasajero, char terminal) {
    this.guardiaTren.lock();
    while(this.asientosOcupados==10 && this.letraTerminal!='') {
        //si el tren esta ocupados me quedo esperandolo
        try {
            this.esperarHall.await();
        } catch (InterruptedException e) {}
    }
    this.asientosOcupados++;
    System.out.println("[ "+emoji+" \uD83E\uDC45 "+this.toString()+" RECOGE]: Subió el "+pasajero.toString()+"
(Asientos ocupados: "+this.asientosOcupados+"");
    //anotamos en que terminal hay que bajarnos
    this.pasajerosPorTerminal[(int) terminal - 65]++;
}
```

bajarDelTren(Pasajero pasajero, char terminal)

Método privado de instancia

No retorna nada.

Permite a los pasajeros enviados por parámetro bajar del tren en la terminal enviada por parámetro.

```
private void bajarDelTren(Pasajero pasajero, char terminal) {
    //anoto que se bajo alguien en esta terminal
    this.pasajerosPorTerminal[(int) terminal - 65]--;
    this.asientosOcupados--;
    System.out.println("[ "+emoji+" \uD83E\uDC47 "+this.toString()+" BAJA]: El
"+pasajero.toString()+" se ha bajado del TREN (Asientos ocupados:
"+this.asientosOcupados+"");
    //le aviso al tren que ya nos bajamos
    this.avanzarTren.signalAll();
    this.guardiaTren.unlock();
}
```

Vuelo

Atributos

De instancia:

- id (Tipo int): identificador numérico del vuelo.
- cantidadPasajeros (Tipo int): cantidad de pasajeros que han sacado pasajes para el vuelo.
- aerolinea (Tipo Aerolinea): Aerolinea del vuelo.
- embarque_hora (int[]): Arreglo de dos dimensiones, cuyo primer elemento será el número de embarque donde deberá ir, y su segundo parámetro la hora de aterrizaje.
- terminal (Tipo char): letra de la terminal donde se encuentra el embarque donde aterrizará.
- torre (Tipo TorreDeControl): Torre de control donde el vuelo deberá pedir permisos para aterrizar y despegar.

- abordaje (Tipo Semaphore): Semáforo que permite saber si ya todos los pasajeros han subido
- oficialAbordo (Tipo Semaphore): Semáforo binario que permite que los pasajeros suban de manera coordinada al vuelo.

De clase:

- cont (Tipo AtomicInteger): Cantidad de vuelos creados

Métodos

Vuelo(TorreDeControl torre, char terminal,int [] embarque_hora, Aerolinea aerolinea)

Método público.

Constructor de la clase.

Método para crear una instancia de Vuelo.

```
public Vuelo(TorreDeControl torre, char terminal,int [] embarque_hora, Aerolinea aerolinea) {
    this.id=cont++;
    this.terminal=terminal;
    this.embarque_hora=embarque_hora;
    this.aerolinea=aerolinea;
    this.cantidadPasajeros=0;
    this.torre=torre;
    this.oficialDeAbordo=new Semaphore(0);
    this.abordajes=new Semaphore(0);
}
```

getCantidadPasajeros()

Método público de instancia.

Retorna un int.

Concretamente retorna la cantidad de pasajeros que han sacado un pasaje para el vuelo.

```
public int getCantidadPasajeros() {
    return this.cantidadPasajeros;
}
```

getId()

Método público de instancia.

Retorna un int.

Concretamente retorna el identificador numérico del vuelo.

```
public int getId() {
    return this.id;
}
```

getEmbarque_hora()

Método público de instancia.

Retorna un arreglo de int.

Concretamente retorna el embarque y la hora de aterrizaje.

```
public int [] getEmbarque_hora() {
    return this.embarque_hora;
}
```

getEmbarque()

Método público de instancia.

Retorna un int.

Permite saber el embarque donde aterrizará el vuelo

```
public int getEmbarque() {  
    return this.embarque_hora[0];  
}
```

getHora()

Método público de instancia.

Retorna un int.

Permite saber la hora de aterrizaje del vuelo.

```
public int getHora() {  
    return this.embarque_hora[1];  
}
```

getTerminal()

Método público de instancia.

Retorna un char.

Permite saber la terminal donde deberá ir el vuelo al aterrizar.

```
public char getTerminal() {  
    return this.terminal;  
}
```

getAerolinea()

Método público de instancia.

Retorna una Aerolinea.

Permite saber la aerolínea del vuelo.

```
public Aerolinea getAerolinea() {  
    return this.aerolinea;  
}
```

toString()

Método público de instancia.

Retorna una String

Permite obtener una cadena de caracteres que informa el n° de vuelo, su aerolínea, la terminal, el embarque y la hora en la que aterrizara.

```
public String toString() {  
    return ("VUELO "+this.id+ "(Aerolinea: "+this.aerolinea.getNombre()+"; Terminal: "+this.terminal+"; Embarque: "+this.embarque_hora[0]+"; Hora: "+this.embarque_hora[1]+")");  
}
```

sacarPasaje()

Método público de instancia.

No retorna nada.

Avisa al vuelo que han sacado pasajes para el mismo.

```
public synchronized void sacarPasaje() {
    this.cantidadPasajeros++;
    System.out.println("\u2708 \ud83d\udcb5 VUELO "+this.id+": pasajes reservados: "+this.cantidadPasajeros);
}
```

subirPasajero(Pasajero pasajero)

Método público de instancia.

No retorna nada

Permite al pasajero subido por parámetro subir al avión.

```
public void subirPasajero(Pasajero pasajero) {
    System.out.println("\ud83d\udec9 \ud83d\ude0e "+pasajero.toString()+" Espera a que llegue el "+this.toString());
    //el pasajero trata de subir al avion
    try {
        this.officialDeAbordo.acquire();
    } catch (InterruptedException e) {}
    System.out.println("\ud83e\udd42 EMBARQUE "+this.toString()+" Ha subido el "+pasajero.toString());
    //Informo que me subí
    this.abordajes.release();
    //se deja el paso al siguiente pasajero
    this.officialDeAbordo.release();
}
```

run()

Método público de instancia.

No retorna nada

Permite al vuelo ejecutar su flujo de trabajo explicado en el análisis general desde la perspectiva del Vuelo

```
public void run() {
    //primero debemos cerrar las ventas de pasajes
    this.torre.finalizarVentas(this);
    //luego pedimos aterrizar
    if(this.torre.permitirAterrizaje(this)) {
        //embarcamos
        this.embarcar();
        try {Thread.sleep(2000);} catch (InterruptedException e) {}
        //pedimos despegar
        this.torre.ordenarDespegue(this);
    }
}
```

embarcar()

Método privado de instancia.

No retorna nada


















Permite al vuelo iniciar y terminar su embarque



















```
private void embarcar() {
    System.out.println("\u2708 \ud83d\udd11 EMBARQUE: "+this.toString()+" empiezan a subir pasajeros");
    //permiso el abordaje
    this.officialDeAbordo.release();
    try {
        //controlo que suban todos
        this.abordajes.acquire(this.cantidadPasajeros);
    } catch (InterruptedException e) {}
    System.out.println("\u2708 \ud83d\udd12 EMBARQUE: "+this.toString()+" finalizó su embarque");
}
```








}

Anexos

Tabla de emojis

Emoji	Código	Significado
	\uD83D\uDE86\uD83C\uDD37	El tren se encuentra en el Hall
	\uD83D\uDEC9	Un pasajero ha llegado al aeropuerto
	\uD83D\uDC6E	Un pasajero se ha formado en el puesto de informes
	\u2708 \u274c	Un vuelo ha dejado de vender pasajes
	\uD83D\uDEC9 \uD83D\uDEC8	Un pasajero esta siendo atendido por un puesto de informes
	\uD83D\uDEC9 \uD83D\uDEC8 \u2714	Un pasajero ha sido atendido por un puesto de infrome
	\u2708 \ud83d\udcb5	Se ha vendido un pasaje para un vuelo
	\uD83D\uDC82	Un pasajero se ha formado en el puesto de atención de una aerolínea
	\uD83D\uDC69	Un pasajero está siendo atendido por un puesto de atención de una aerolínea
	\uD83D\uDC69 \u2714	Un pasajero fue atendido por un puesto de atención de una aerolínea
	\uD83D\uDE86 \uD83E\uDC45	Un pasajero sube al tren
	\uD83D\uDE86 \uD83E\uDC47	Un pasajero baja del tren
	\uD83D\uDE86 \uD83C\uDD70	El tren llega a la terminal A
	\uD83D\uDE86 \uD83C\uDD71	El tren llega a la terminal B
	\uD83D\uDE86 \uD83C\uDD72	El tren llega a la terminal C
	\uD83D\uDE86 \u2B8A	El tren continúa su recorrido
	\uD83D\uDE86 \u2B31	El tren está volviendo al Hall Central

 A	\uD83D\uDED2 \uD83C\uDD70	Un pasajero trata de ingresar al Free Shop de la estación A
 B	\uD83D\uDED2 \uD83C\uDD71	Un pasajero trata de ingresar al Free Shop de la estación B
 C	\uD83D\uDED2 \uD83C\uDD72	Un pasajero trata de ingresar al Free Shop de la estación C
 A 😊	\uD83D\uDED2 \uD83C\uDD70 \ud83d\u2013	Un pasajero ingresó al Free Shop de la estación A
 B 😊	\uD83D\uDED2 \uD83C\uDD71 \ud83d\u2013	Un pasajero ingresó al Free Shop de la estación B
 C 😊	\uD83D\uDED2 \uD83C\uDD72 \ud83d\u2013	Un pasajero ingresó al Free Shop de la estación C
 A 😞	\uD83D\uDED2 \uD83C\uDD70 \ud83d\u2013	Un pasajero quiere usar una caja del Free Shop A
 B 😞	\uD83D\uDED2 \uD83C\uDD71 \ud83d\u2013	Un pasajero quiere usar una caja del Free Shop B
 C 😞	\uD83D\uDED2 \uD83C\uDD72 \ud83d\u2013	Un pasajero quiere usar una caja del Free Shop C
 A 😊	\uD83D\uDED2 \uD83C\uDD70 \ud83d\u2013	Un pasajero se retira del Free Shop de la estación A
 B 😊	\uD83D\uDED2 \uD83C\uDD71 \ud83d\u2013	Un pasajero se retira del Free Shop de la estación B
 C 😊	\uD83D\uDED2 \uD83C\uDD72 \ud83d\u2013	Un pasajero se retira del Free Shop de la estación C
 A	\ud83d\uddcd\u2013 \uD83C\uDD70	Un pasajero está siendo atendido por una caja del Free Shop A
 B	\ud83d\uddcd\u2013 \uD83C\uDD71	Un pasajero está siendo atendido por una caja del Free Shop B
 C	\ud83d\uddcd\u2013 \uD83C\uDD72	Un pasajero está siendo atendido por una caja del Free Shop C
 A 😊	\ud83d\uddcd\u2013 \uD83C\uDD70 \ud83d\u2013	Un pasajero fue atendido por una caja del Free Shop A
 B 😊	\ud83d\uddcd\u2013 \uD83C\uDD71 \ud83d\u2013	Un pasajero fue atendido por una caja del Free Shop B
 C 😊	\ud83d\uddcd\u2013 \uD83C\uDD72 \ud83d\u2013	Un pasajero fue atendido por una caja del Free Shop C

	\uD83D\uDEC9 \ud83d\uDE0E	Un pasajero espera en su terminal a que llegue su vuelo
	\uD83D\uDEEC	Un vuelo ha aterrizado
	\uD83D\uDEE7 \uD83D\uDEEB	Un vuelo ha despegado
	\u2708 \ud83d\udd11	Un vuelo ha comenzado su embarque
	\u2708 \ud83d\udd12	Un vuelo ha finalizado su embarque
		Un pasajero ha subido a su vuelo
		Una persona se ha equivocado de aeropuerto