Empresa de viajes

Se trata de una empresa que coordina el transporte de sus clientes entre distintas localidades. Cada vez que un cliente quiere viajar a alguna localidad hace una reserva indicando:

- fecha del viaje
- localidad de origen
- localidad de destino

La reserva queda pendiente hasta el momento en que la empresa le asigna un transporte a dicho viaje.

Cada localidad pertenece a una zona determinada y sabe los transportes que están asociados a la empresa de viajes.
Cada zona conoce sus zonas cercanas y si tiene aeropuerto.
Cada vez que un transporte cumple el recorrido fijado vuelve al lugar de origen.

Los distintos transportes con los que trabaja la empresa en la actualidad son los siguientes:

- Los viajes en remís se hacen entre dos localidades no lejanas. Además, estos viajes se hacen en el día y pueden realizarse hasta 5 reservas para un remís en una misma fecha.
- Los viajes en micro pueden hacerse entre zonas cercanas y no cercanas, pero no se hacen dentro de una misma zona. Estos se hacen en el día si son entre zonas cercanas, y demoran dos días si las zonas no lo son. Un micro no puede ocuparse para más de un viaje en una misma fecha.
- Los viajes en avión se hacen exclusivamente entre zonas lejanas con aeropuerto. Los viajes en avión se hacen en el día, con una única reserva diaria. Solo pueden viajar a zonas que tengan aeropuerto.

Cuando un cliente hace una reserva, la misma puede satisfacerse si existe, en la zona y para la fecha solicitada, un transporte disponible que pueda realizar el viaje entre el origen y el destino.

Se pide: Codificar

1) **Transportes y localidades**

a) Saber si dos localidades son lejanas. Son lejanas cuando pertenecen a zonas distintas que no son cercanas entre sí.

b) Conocer los transportes de una localidad que son capaces de viajar a otra localidad especificada.

2) **Asignación del transporte a una reserva**

a) Saber si un transporte está ocupado en una determinada fecha, es

decir, si existe alguna reserva que provoque que no pueda utilizarse en esa fecha.

b) Saber si puede satisfacerse una reserva pendiente.

c) Asignar un transporte a una reserva, considerando las acciones necesarias para que los puntos anteriores funcionen correctamente. En caso de que no se pueda satisfacer la reserva, enviar un mensaje de error (self error: '... <mensaje de error explicando por qué no puede satisfacerse la reserva> ...')

3) **Estadísticas**

a) Saber las zonas preferidas de un cliente, que son aquellas hacia las que el cliente viajó (zona destino) más de 10 veces.

b) Conocer la zona "hot" de un cliente... es decir, la zona para la que más veces hizo una reserva (en base a la localidad de destino).

c) Saber si un cliente conoce una zona como la palma de su mano, que es cuando viajó a todas las localidades de la zona.

En todos los casos, indicar objeto receptor de cada mensaje.

4) Resolver **uno** de los siguientes 2 ítems:

a) Hacer dos diagramas de objetos en los cuales se puedan ver los cambios introducidos por la asignación del transporte a una reserva.

b) O bien, hacer un workspace en el cual se pruebe si una reserva puede satisfacerse.

5) **Genere el diagrama de clases de la solución adoptada**, en una hoja separada del resto.

6) Ahora la empresa nos pide agregar el helicóptero, el cual puede hacer viajes entre localidades no lejanas, y puede reservarse para un máximo de dos viajes por día.

a) ¿Qué cambiaría de lo ya realizado en el modelo? Codifique todos los métodos necesarios para que los 3 primeros puntos funcionen y haga las modificaciones pertinentes en el diagrama de clases.

b) ¿Qué concepto permite que el impacto en la solución existente sea menor? Justifique.

Ayuda: a una fecha se le puede enviar un mensaje addDays:. Ej: si Date today es 05/12/2009, si hacemos Date today addDays: 2 eso me devuelve 07/12/2009.

Posible solución:

(1.a)

```
#Localidad
esLejanaDe: unaLocalidad
  ^self zona esLejanaDe: unaLocalidad zona

#Zona
esLejanaDe: unaZona
  ^(self = unaZona) not and: [(self zonasCercanas includes: unaZona) not]
```

(1.b)

```
#Localidad
transportesQuePuedenViajarA: unaLocalidad
  self transportes select: [:unTransporte|unTransporte puedeViajarDesde: self a:
unaLocalidad]

#Remis
puedeViajarDesde: unaLocalidad a: otraLocalidad
  ^(unaLocalidad esLejanaDe: otraLocalidad) not

#Micro
puedeViajarDesde: unaLocalidad a: otraLocalidad
  ^(unaLocalidad esDeLaMismaZonaQue: otraLocalidad) not

#Localidad
esDeLaMismaZonaQue: unaLocalidad
  ^self zona = unaLocalidad zona

#Avion
puedeViajarDesde: unaLocalidad a: otraLocalidad
  ^unaLocalidad tieneAeropuerto &
   otraLocalidad tieneAeropuerto &
   (unaLocalidad esLejanaDe: otraLocalidad)
```

(2.a)

```
#Remis
estaOcupadoEn: unaFecha
  ^(self reservasQueOcupan: unaFecha) size >= 5

#Transporte
reservasQueOcupan: unaFecha
  ^self reservas select: [:unaReserva|unaReserva ocupa: unaFecha]

#Reserva
ocupa: unaFecha
  ^self transporte unViajeAPartirDe: self fecha ocupa: unaFecha

#Transporte
unViajeAPartirDe: unaFecha ocupa: otraFecha
  "--solo para el micro se redefine, el resto son viajes en el día—"
  ^unaFecha = otraFecha

cuandoVuelveSaliendoEn: unaFecha
  "--necesario en punto 2.b, redefinido para el micro--"
  ^unaFecha
```

```
estaOcupadoEn: unaFecha
  "--se redefine para el remis y el helicóptero--"
  ^(self reservasQueOcupan: unaFecha) notEmpty

#Micro
unViajeAPartirDe: fechaReserva ocupa: unaFecha
  ^unaFecha between: fechaReserva and: (self cuandoVuelveSaliendoEn:
fechaReserva)

cuandoVuelveSaliendoEn: unaFecha
  "--en método separado para usar también en 2.b--"
  ^unaFecha addDays: 1
```

(2.b)

```
#Reserva
puedeSatisfacerse
  ^self origen puedeSatisfacer: self

#Localidad
puedeSatisfacer: unaReserva
   ^(self transportesQueSatisfacen: unaReserva) notEmpty

transportesQueSatisfacen: unaReserva
  ^(self transportesQuePuedenViajarA: unaReserva destino)
     select: [:unTransporte|(unaReserva puedeHacerseEnFechaCon: unTransporte)]

#Reserva
puedeHacerseEnFechaCon: unTransporte
  |fechaFinReserva|
  fechaFinReserva := unTransporte cuandoVuelveSaliendoEn: self fecha.
  ^((unTransporte estaOcupadoEn: self fecha) or:
    [unTransporte estaOcupadoEn: fechaFinReserva]) not
```

(2.c)

```
#Reserva
asignarTransporte
  self puedeSatisfacerse ifFalse: [self error: 'No hay transportes disponibles que
puedan satisfacer la reserva'].
  transporte := (self origen transportesQueSatisfacen: self) anyOne.
  transporte agregarReserva: self.
```

Otra forma:

```
#Reserva
asignar: unTransporte
  self puedeSatisfacerse ifFalse: [self error: 'No hay transportes disponibles que
puedan satisfacer la reserva'].
  (self puedeHacerseEnFechaCon: unTransporte) ifFalse: [ self error: 'El transporte
no puede asignarse en esa fecha' ].
  unTransporte agregarReserva: self.

#Transporte
agregarReserva: unaReserva
  self reservas add: unaReserva
```

(3.a)

```
#Cliente
zonasPreferidas
  ^self zonasDestino select: [:unaZona|(self cantidadDeVisitasA: unaZona) > 10]

zonasDestino
  ^(self reservas collect: [:unaReserva|unaReserva destino zona]) asSet

cantidadDeVisitasA: unaZona
  ^(self reservas select: [:unaReserva|unaReserva destino zona = unaZona]) size
```

(3.b)

```
#Cliente
zonaHot
  ^(self zonasDestino asSortedCollection:
       [:unaZona :otraZona|(self cantidadDeVisitasA: unaZona) > (self
cantidadDeVisitasA: otraZona)]) first
```

(3.c)

```
#Cliente
conoceComoLaPalmaDeSuManoA: unaZona
  ^unaZona fueATodosLados: self

#Zona
fueATodosLados: unCliente
  ^self localidades allSatisfy: [:unaLocalidad|unCliente visito: unaLocalidad]

#Cliente
visito: unaLocalidad
  ^self reservas anySatisfy: [:unaReserva|unaReserva destino = unaLocalidad]
```

(6)

```
#Helicoptero
estaOcupadoEn: unaFecha
  ^(self reservasQueOcupan: unaFecha) size >= 2

puedeViajarDesde: unaLocalidad a: otraLocalidad
  ^(unaLocalidad esLejanaDe: otraLocalidad) not
```