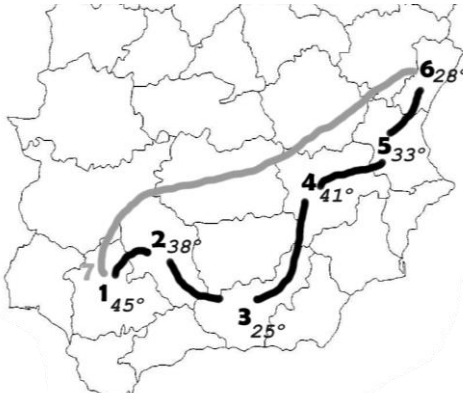
Nombre: _____Apellidos:_____

Ejercicio 2: El Fresquito

Descripción del problema

El grupo de amigos *El Fresquito*, planean su viaje de vacaciones. Disponen de varios días de vacaciones en los que recorrerán algunas ciudades de la geografía española. A ellos no les importan qué ciudades visitar, sólo se preocupan por no pasar calor.



Aunque todos los años suelen guiarse por las normalmente erróneas indicaciones de uno de los miembros del grupo, esta vez *El Fresquito* dispone de un pronóstico de la temperatura que hará durante todo el viaje en cada una de las ciudades de España.

El grupo **comienza su viaje desde Sevilla siempre**, donde pasan la primera noche y, cada día, **viajan a una ciudad vecina** y pasan allí el resto del día y la noche. Al día siguiente viajan a otra ciudad cercana diferente y así sucesivamente. El último día de vacaciones lo invierten viajando de vuelta a Sevilla (**la última ciudad puede estar alejada de Sevilla**).

En la imagen superior se muestra el recorrido que ha realizado *El Fresquito* para unas vacaciones de 7 días. Puede verse como son sólo 6 las ciudades visitadas. Puede observarse también la temperatura máxima que hizo en cada uno esos lugares. Este viaje tuvo **una temperatura máxima media** de (45+38+25+41+33+28)/6= 35º.

Se modelará lo arriba descrito para resolver el problema de este grupo, que consiste en planificar, de **manera voraz,** el viaje de forma que se **minimice la temperatura máxima media**.

Modelado:

Para simplificar el problema, se tendrán en cuenta las siguientes suposiciones:

1. Cada ciudad vendrá representada por un *String*.
2. La vecindad entre ciudades se representará como una propiedad compartida, ***vecinos***, de tipo *Map<String, List<String>>*. Para cada ciudad, amacena las ciudades a las que se pueden llegar, es decir, las **alternativas** de *El Fresquito*.
3. Existirá otra propiedad compartida, ***numDias***, de tipo *Integer* que indica cuántos días hay inicialmente de vacaciones.
4. Por último, se dispondrá de otra propiedad compartida, ***temperaturas***, de tipo *Map<String, List<Integer>>* que indicará la temperatura máxima pronosticada para cada ciudad en cada día del viaje. Por ejemplo, para saber la temperatura que hará en Sevilla el primer día de viaje, se usará la sentencia:

```
temperaturas.get("Sevilla").get(1-1);
```

5. En cada *Estado*, *El Fresquito* tiene que decidir a qué ciudad viajará de todas las vecinas que no haya visitado. Elegirá aquella ciudad que tenga menor ***temperatura*** ese día.
6. El problema terminará cuando no queden más días de vacaciones. **No considere el caso en que no queden más ciudades vecinas no visitadas.**
7. La solución al problema será de tipo *SolucionFresquito* con una propiedad, ***ciudadesRecorridas*** de tipo *List<String>*.

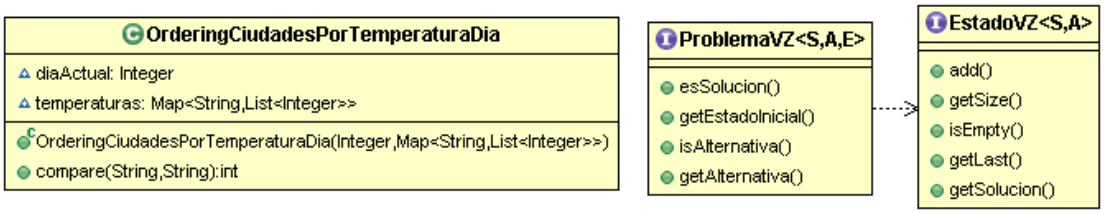
Ejemplo de ejecución:

Dados los datos de ***vecinos*** de la imagen anterior, teniendo como ***temperaturas*** la tabla de la derecha y un ***numDias*** igual a 4, la solución obtenida utilizando un esquema voraz, debería ser.

- ***ciudadesRecorridas***: Sevilla, Cádiz, Málaga.

Su temperatura máxima media sería: (45+30+26)/3=33,67.

	1	2	3
Sevilla	**45**	43	40
Huelva	30	36	33
Córdoba	40	35	35
Cádiz	28	**30**	36
Málaga	29	31	**26**
Jaen	32	28	25
…			



Se pide:

1. Rellenar las partes de la ficha que no estén rellenas.
2. Implementar los métodos (Deberá corresponderse con lo que escriba en la ficha):
   a) `private void voraz()`.
   b) `public String getAlternativa(EstadoFresquito e)`. De *ProblemaFresquitoImpl*. **La clase "OrderingCiudadesPorTemperaturaDia" puede ser utilizada. Hereda the Ordering<String>. Su constructor recibe como parámetros el día actual y la propiedad *temperaturas*. El criterio de ordenación es primero el que menos temperatura tiene.**
   c) `public boolean isAlternativa(EstadoFresquito e)`.
   De *ProblemaFresquitoImpl*.
   d) `public boolean add(String c)`. De *EstadoFresquitoImpl*.

Problema El Fresquito	
Técnica: Voraz	
Propiedades Compartidas	***temperaturas**: Map<String, List<Integer>>* ***vecino**:Map<String,List<String>>* ***numDias**: Integer*
Propiedades del Estado	***ciudadesRecorridas**: List<String>* ***diasRestantes**: Integer*
Solucion: SolucionFresquito	
Alternativa: // Tendrá en cuenta las ciudades accesibles desde la última recorrida $A_{estado}=\{a_i \in (vecino.get(estado.ciudadesRecorridas.last) - estado.ciudadesRecorridas)$	
Elección: // Se tendrá en cuenta la propiedad temperatura del día actual *h(estado)= a_i, tal que:* $temperaturas.get(a_i).get(numDias - estado.diasRestantes)$ *sea minimo*	
Estado Inicial: //TODO: Habrá que considerar una ciudad inicial	
Estado Final: // Se tendrá que indicar cuándo El Fresquito no puede continuar *diasRestantes=0*	
Add(a_i): //TODO: Los valores del estado tendrán que actualizarse con la nueva ciudad	

SOLUCION PROPUESTA SOLUCION PROPUESTA

Problema El Fresquito	
Técnica: Voraz	
Propiedades Compartidas	***temperaturas**: Map<String, List<Integer>>* ***vecino**:Map<String,List<String>>* ***numDias**: Integer*
Propiedades del Estado	***ciudadesRecorridas**: List<String>* ***diasRestantes**: Integer*
Solucion: SolucionFresquito	
Alternativa: $A_{estado}=\{a_i \in (vecino.get(estado.ciudadesRecorridas.last) - estado.ciudadesRecorridas)$	
Elección: h(estado)= a_i *, tal que:* $temperaturas.get(a_i).get(numDias - estado.diasRestantes)$ *sea minimo*	
Estado Inicial: *ciudadesRecorridas={Sevilla}* *diasRestantes=numDias*	
Estado Final: *diasRestantes=0*	
Add(a_i*):* *ciudadesRecorridas+=* a_i *diasRestantes--*	

a)

```
        private void voraz() {
                while (problema.isAlternativa(estado)) {
                        String alternativa = problema.getAlternativa(estado);
                        estado.add(alternativa);
                }
                solucion = estado.getSolucion();
    }
```

b)

```
    public String getAlternativa(EstadoFresquito e) {

        Collection<String> vecinos=new ArrayList<String>(
                this.vecinos.get(e.getCiudadesRecorridas().get(
                        e.getCiudadesRecorridas().size()-1)));
        vecinos.removeAll(e.getCiudadesRecorridas());

        return  new OrderingCiudadesPorTemperaturaDia(
                        numDias-e.getDiasRestantes(),
                        this.temperaturas).min(vecinos);
    }
```

c)

```
    public boolean isAlternativa(EstadoFresquito e) {
        return e.getDiasRestantes()>0;
    }
```

d)

```
    public boolean add(String e) {
        boolean ret=false;
        if(this.diasRestantes>0){
            ret=true;
            ciudadesRecorridas.add(e);
            this.diasRestantes--;
        }
        return ret;
    }
```