Ejercicio 6: Backtracking

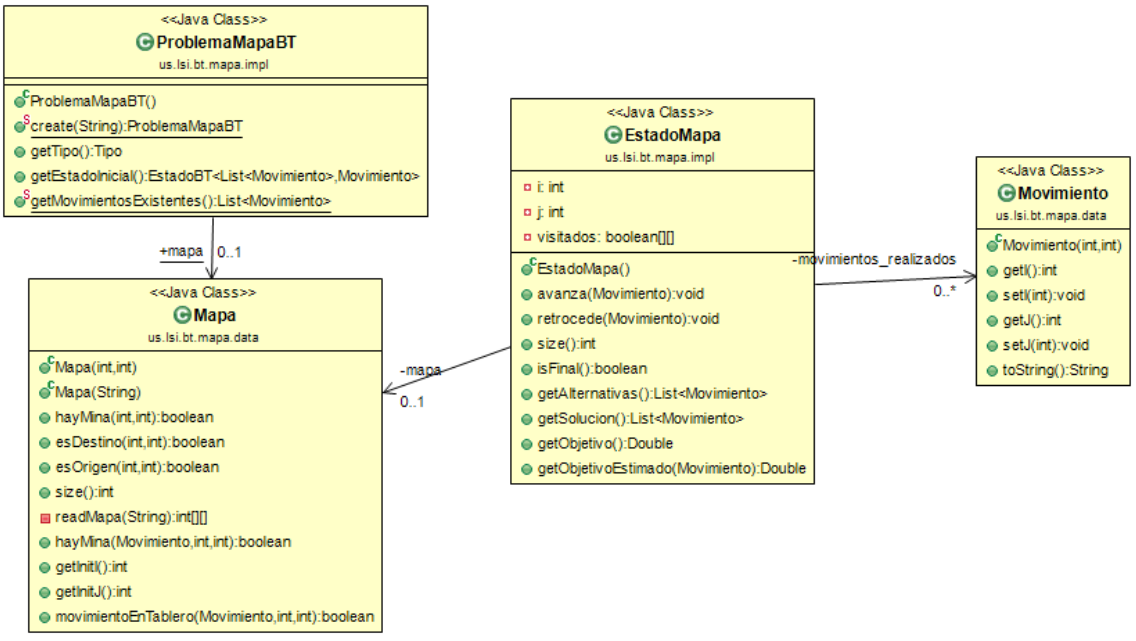
El negocio del Aragorn

Aragorn, quiere ir desde Gondor hasta La comarca. Para llegar a La comarca ha de atravesar el campo de minas donde se han marcado con un 1 aquellas posiciones con minas y con 0 las posiciones libres de ellas.

Por ejemplo, en el mapa presentado en la figura, para llegar desde Gondor a La comarca, un posible camino es ↓,↓,→,↓,→,→

Gondor	**0**	**1**	**0**
0	**1**	**1**	**1**
0	**0**	**1**	**1**
1	**0**	**0**	**La comarca**

Se pide diseñar un algoritmo de vuelta atrás para encontrar el camino más corto entre las ciudades de Gondor y La comarca. Para ello, se proveen las estructuras de datos presentadas en la siguiente figura.



Nota: La clase movimiento almacena las posiciones a sumar o restar para el eje x (i) y el eje y(j) del mapa.

Se pide:

1) Completar la tabla //TODO
2) Los métodos:
    a. void avanza(Movimiento a)
    b. List<Movimiento> getAlternativas()
    c. List<Movimiento> getSolucion()
    d. boolean isFinal()
    e. Double getObjetivo()

BT	
Problema Mapa	
Técnica: Vuelta atrás	
Propiedades Compartidas	mapa : Mapa // elemento que representa el mapa movimientosExistentes:List<Movimiento>//direcciones en las que mueve initI:int// posición I inicial initJ:int// posición J inicial finI: int //posición I final finJ: int //posición J final
Propiedades del estado	j,i:int // enteros representando la posición en el mapa visitados: boolean[][] // si se ha visitado la posición movimientos: List<Movimiento> // movimientos movimientos_realizados:List<Movimiento>// movimientos
Solución: List<Movimiento> // lista de movimientos a realizar para ir desde el origen al destino	
Inicial: //TODO	
Final: //TODO	
Alternativas://TODO	
Avanza(a): //TODO	

BT	
Problema Mapa	
Técnica: Vuelta atrás	
Propiedades Compartidas	mapa : Mapa // elemento que representa el mapa movimientosExistentes:List<Movimiento>//direcciones en las que mueve initI:int// posición I inicial initJ:int// posición J inicial finI: int //posición I final finJ: int //posición J final
Propiedades del estado	j,i:int // enteros representando la posición en el mapa visitados: boolean[][] // si se ha visitado la posición movimientos: List<Movimiento> // movimientos movimientos_realizados:List<Movimiento>// movimientos
Solución: List<Movimiento> // lista de movimientos a realizar para ir desde el origen al destino	
Inicial: visitados[0..n][0..m]=falso, i=initI, j= initJ, movimientos_realizados= []	
Final: i=finI ^ j=finJ	
Alternativas: $A_{i,j,mapa,visitados,movimientosExistentes} = \{mov: movimientosExistents\}, mov.getJ() + j \geq 0, nextI + i \geq 0, nextJ + mov.getJ() \leq mapa.sizeJ, nextI + mov.getI() \leq mapa.sizeI, !visitados(mov.getI() + nextI, mov.getJ() + nextJ), !mapa.hayMina(mov.getI() + nextI, mov.getJ() + nextJ)$	
Avanza(a): i+aI, j+aJ,visitados(i,j)=true, movimientos.add(i,j)	

Avanza

```
@Override
	public void avanza(Movimiento a) {
		this.i+=a.getI();
		this.j+=a.getJ();
		this.visitados[i][j]=true;
		movimientos.add(a);
	}
```

getAlternativa

```
@Override
public List<Movimiento> getAlternativas() {
	List<Movimiento> res = new LinkedList<Movimiento>();
	for (Movimiento mov : ProblemaMapaBT.getMovimientosExistentes()) {
		if (mapa.movimientoEnTablero(mov, this.i, this.j)
		&& !mapa.hayMina(mov, this.i, this.j)
		&& !this.visitados[this.i + mov.getI()][this.j + mov.getJ()]) {
			res.add(mov);
		}
	}

return res;
}
```

getSolucion

```
@Override
public List<Movimiento> getSolucion() {
	return new ArrayList<Movimiento>(movimientos);
}
```

isFinal

```
@Override
public boolean isFinal() {
	return mapa.esDestino(i, j);
}
```

getObjetivo

```
@Override
public Double getObjetivo() {
	return (double) movimientos.size();
}
```