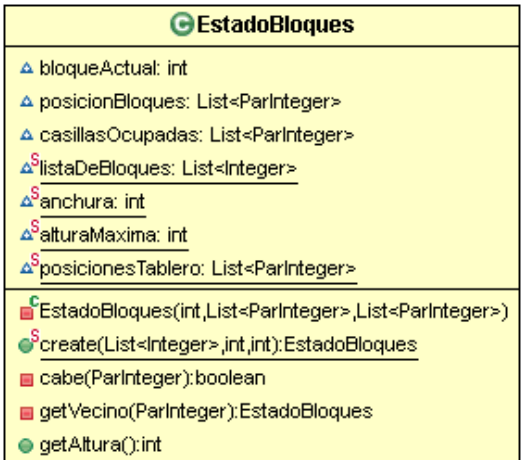
PROBLEMA 2: Grafos Virtuales

Se quiere obtener la posición óptima que han de tener diferentes **bloques cuadrados** dentro de un tablero de ancho fijo para que, colocándose **todos los bloques**, **su altura sea la mínima** posible. Para ello, considere lo siguiente:

1. El tamaño de los bloques cuadrados, así como el ancho del tablero y su altura máxima serán conocidos a priori.
2. La posición de cada bloque vendrá identificada por su coordenada inferior izquierda.
3. Los bloques no pueden solaparse.
4. Tiene disponible una clase *EstadoBloques* con los siguientes atributos y métodos:
   a. static List<ParInteger> posicionesTablero: propiedad estática que contiene todas las posiciones del tablero, desde la (0,0), hasta la (*anchura-1*, *alturaMaxima-1*)
   b. boolean cabe(ParInteger): devuelve true si la al colocar *bloqueActual* en la casilla pasada como parámetro, no solapa con otros bloques y no excede los límites del tablero.
   c. EstadoBloque getVecino(ParInteger): *EstadoBloque* resultante al colocar el *bloqueActual* en la casilla pasada como parámetro.

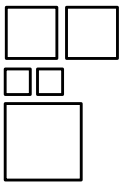


Ejemplo:

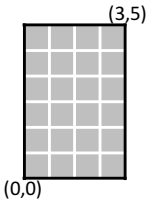
Para los siguientes datos:
Anchura: **4**
Altura máxima: **6**
Lista de Bloques: **[2,2,1,1,3]**

EstadoBloques inicial:
Bloque actual: **0**
Posición de bloques: **[]**

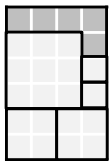
Bloques



Tablero



Una solución sería la siguiente:



altura=5

EstadoBloques final:
Bloque actual: **6**
Posición de bloques:
[(0,0), (0,2), (2,3), (4,3), (3,3), (2,0)]

Para resolver este problema usando Grafos Virtuales **se pide:**

1. ¿Qué atributo tienen los vértices? **Justifíquelo.** *Pista: Mirar el UML*
2. ¿Qué elementos tienen pesos y qué valor tienen? **Justifíquelo**
3. **Justifique** cuántos vecinos tiene el vértice inicial del ejemplo e indique el valor de sus atributos.
4. ¿Usaría el algoritmo de Dijkstra o A*? **Justifíquelo**
5. Implemente los siguientes métodos:
   a. `public Set<EstadoBloques> getNeighborListOf()`
   b. `public Set<SimpleEdge<EstadoBloques>> edgesOf()`

1. Los especificados en el UML ☺

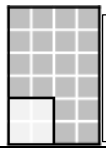
bloqueActual que apunta a un bloque de la lista de bloques. Cada estado considera la colocación de un bloque en lista posicionBloques.

posicionBloques que guarda las posiciones seleccionadas para cada bloque.

casillasOcupadas se deriva de posicionBloques e indica todas las posiciones

2. OPCIÓN 1: Pesos sólo en el último vértice, la altura total. El resto de los elementos peso=0.
OPCIÓN 2: Pesos en las aristas, la diferencia de altura entre el destino y el origen. El resto 0.

3. Tiene 15 vecinos ya que el primer bloque, de 2x2, puede colocarse en las 3 primeras casillas de las 5 primeras filas. Ejemplo:



bloqueActual=1
posicionesBloques=[(0,0)]
casillasOcupadas=[(0,0),(0,1),(1,0),(1,1)]

4. Si los pesos están las aristas y no aplico heurística, Dijstra. A* en otro caso. Se puede justificar A* por la condición de parada.

5a. OPCIÓN 1

```
public Set<EstadoBloques> getNeighborListOf() {
	return edgesOf().stream()
		.map(edge-> edge.getTarget())
		.collect(Collectors.toSet());
}
```

OPCIÓN 2

```
public Set<EstadoBloques> getNeighborListOf() {
	if(bloqueActual >=listaDeBloques.size()){
		return new HashSet<>();
	}
	int tamBloqueActual=listaDeBloques.get(bloqueActual);
	return EstadoBloques.posicionesTablero.stream()
		//No exceda el tablero
		.filter(posicion -> posicion.p1+tamBloqueActual<=anchura
				&& posicion.p2+tamBloqueActual<=alturaMaxima)
		//No solape con otro bloque
		.filter(posicion -> noSolapa(posicion))
		.map(posicion -> getVecino(posicion))
		.collect(Collectors.toSet());
}
```

5b. OPCIÓN 1

```
public Set<SimpleEdge<EstadoBloques>> edgesOf() {
	return getNeighborListOf().stream()
		.map(v-> SimpleEdge.create(this, v))
		.collect(Collectors.toSet());
}
```

OPCIÓN 2

```
public Set<SimpleEdge<EstadoBloques>> edgesOf() {
	if(bloqueActual >=listaDeBloques.size()){
		return new HashSet<>();
	}
	int tamBloqueActual=listaDeBloques.get(bloqueActual);
	return EstadoBloques.posicionesTablero.stream()
		//No exceda el tablero
		.filter(posicion -> posicion.p1+tamBloqueActual<=anchura
				&& posicion.p2+tamBloqueActual<=alturaMaxima)
		//No solape con otro bloque
		.filter(posicion -> noSolapa(posicion))
		.map(posicion -> {
			EstadoBloques vecino= getVecino(posicion);
			return SimpleEdge.create(this, vecino,
					vecino.getAltura()-getAltura());
			})
		.collect(Collectors.toSet());
}
```