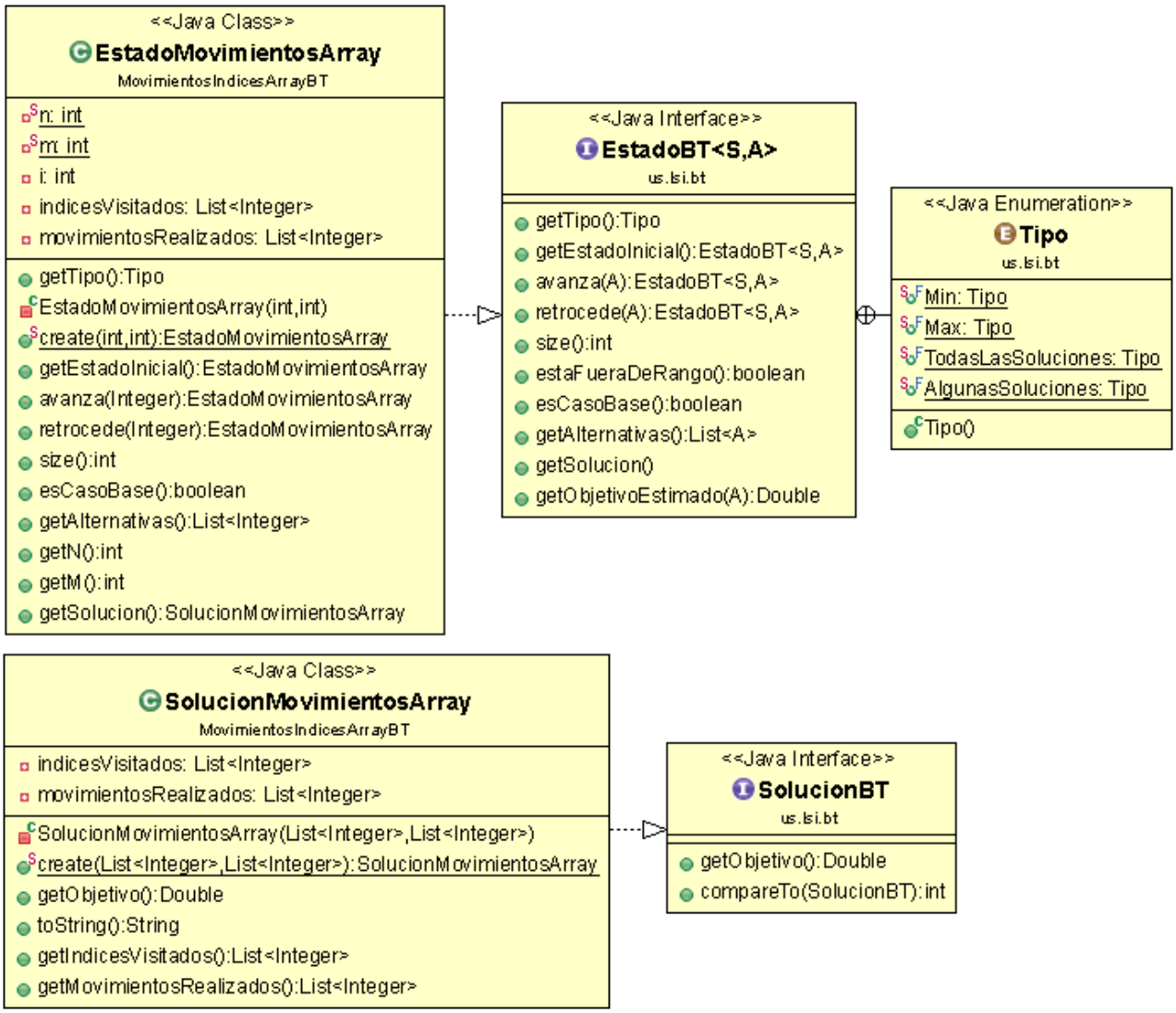
Ejercicio 5 – Backtracking

Se tiene un array de n elementos. Comenzando en la casilla número 0, se desea alcanzar la casilla n-1 realizando exactamente m movimientos permitidos. Los movimientos permitidos son saltar 1, 2 o 3 casillas hacia la derecha o hacia la izquierda.

Por ejemplo, para n = 6 y m = 5, una posible solución sería:

```
Índices visitados: [1, 0, 1, 2, 5]
Movimientos realizados: [1, -1, 1, 1, 3]
```

Se desea resolver este problema mediante Backtracking. La solución debe proporcionar tanto la secuencia de índices visitados como la de movimientos realizados. Para ello, se han modelado las clases EstadoMovimientosArray y SolucionMovimientosArray que se muestran en el siguiente diagrama:



SE PIDE:

1. Completar la ficha que se proporciona
2. Implementar los siguientes métodos de la clase EstadoMovimientosArray:

a) `public EstadoMovimientosArray avanza(Integer a)`
b) `public List<Integer> getAlternativas()`
c) `public SolucionMovimientosArray getSolucion()`

<table>
<tr><th colspan="2">Problema de los movimientos en un array: BT</th></tr>
<tr><td colspan="2">Tipos

• S – SolucionMovimientosArray (puede ver los detalles de esta clase en el UML que se proporciona)
• A - Integer</td></tr>
<tr><td>Propiedades Compartidas</td><td>• n, Integer, tamaño del array
• m, Integer, número de movimientos</td></tr>
<tr><td>Propiedades del Estado</td><td>• i, entero en [0,n-1], índice actual en el array
• movimientosRealizados, lista de enteros que contiene los movimientos realizados hasta el momento
• indicesVisitados, lista de enteros que contiene los índices visitados hasta el momento, derivada</td></tr>
<tr><td colspan="2">Tamaño: //TODO</td></tr>
<tr><td colspan="2">Alternativas: //TODO</td></tr>
<tr><td colspan="2">Instanciación (estado inicial) //TODO</td></tr>
<tr><td colspan="2">Estado Final: //TODO

Avanza: //TODO

Retrocede: //TODO</td></tr>
</table>

Solución:

1)

<table>
<tr><th colspan="2">Problema de los movimientos en un array: BT</th></tr>
<tr><td colspan="2">Tipos

• S – SolucionMovimientosArray
• A - Integer</td></tr>
<tr><td>Propiedades Compartidas</td><td>n, Integer, tamaño del array

m, Integer, número de movimientos</td></tr>
<tr><td>Propiedades del Estado</td><td>i, entero en [0,n-1], índice actual en el array

movimientosRealizados, lista de enteros que contiene los movimentos realizados hasta el momento

indicesVisitados, lista de enterros que contiene los índices visitados hasta el momento, derivada</td></tr>
<tr><td colspan="2">Tamaño: m – movimientosRealizados.size() (ó m – indicesVisitados.size())</td></tr>
<tr><td colspan="2">Alternativas: $A = \{a: a \in [-3 \dots 3], a \neq 0, 0 \leq i + a \leq n - 1|\}$,

La alternativa a representa qué movimiento realizamos desde la posición i</td></tr>
<tr><td colspan="2">Instanciación Estado Inicial $= (0, [], [])$</td></tr>
<tr><td colspan="2">Estado Final
m == movimientosRealizados.size() (ó m == indicesVisitados.size())

Avanza:
$(i, movimientosRealizados, indicesVisitados) \rightarrow^{a} (i + a, movimientosRealizados + a, indicesVisitados + (i + a))$

Retrocede:
$(i, movimientosRealizados, indicesVisitados) \rightarrow^{a} (i - a, movimientosRealizados - a, indicesVisitados - i)$

//tanto en el avanza como en el retrocede puede no incluirse indicesVisitados ya que es una prop. derivada
$\perp$</td></tr>
</table>

2)

```
@Override
public EstadoMovimientosArray avanza(Integer a) {
	i += a;
	indicesVisitados.add(i);
	movimientosRealizados.add(a);
	return this;
}

@Override
```

```
	@Override
	public List<Integer> getAlternativas() {
		List<Integer> movAlt =
				IntStream.rangeClosed(-3, 3)
				.boxed()
				.filter(v -> (i + v) >= 0 && (i + v) <= n-1 &&
v != 0)
				.collect(Collectors.toList());
		return movAlt;
	}

	@Override
	public SolucionMovimientosArray getSolucion() {
		SolucionMovimientosArray res = null;
		if(i == n-1)
			res =
SolucionMovimientosArray.create(indicesVisitados,
movimientosRealizados);
		return res;
	}
```