Ejercicio 3: Grafos Virtuales

El problema del Laberinto-NR consiste en un tablero cuadrado de tamaño NxN donde las casillas pueden tener dos colores Rojo o Negro. El problema consiste en partiendo desde una casilla inicial (entrada) llegar a una casilla final (salida) realizando el menor número de movimientos, teniendo en cuenta que si se pasa por una casilla roja se penaliza en 10 el número de movimientos. Los movimientos posibles solo se pueden hacer entre casillas adyacentes horizontales o verticales.

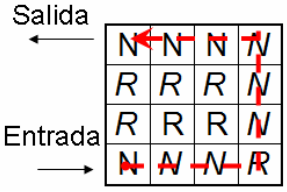
Se pide:

1. Termine de rellenar los campos **// TODO** de la siguiente ficha.

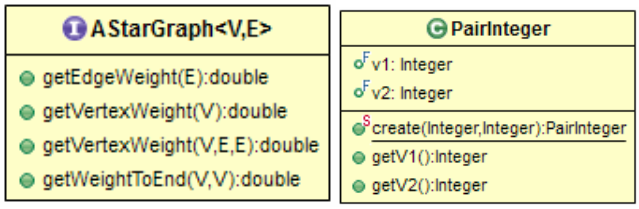
Problema del Puzzle: Grafo Virtual y A*	
Propiedades Compartidas	*CE=(xe,ye), Casilla de entrada* *CS=(xs,ys), Casilla de salida* *N, tamaño del laberinto, derivada* *Movs, Conjunto<PairInteger>,* {(0,1),(0,-1),(1,0),(-1,0)} Tablero, Integer[0..N-1][0..N-1] *1-* Casilla negra 2- Casilla roja
Props del vértice	***// TODO***
Aristas**: **// TODO ***Vecinos:*** $\{(i0+a.i,\ j0+a.j)\ \ \forall a \in A\}$ ***Peso de la arista(e): // TODO*** ***Peso del vértice(v): // TODO*** ***Peso de la trayectoria(v,ein,eout): // TODO*** ***Heurística(v, vf): // TODO , Distancia Manhatan entre v y vf*** ***Vértice Origen:*** CE ***Vértice Objetivo***: CS ***Solución (lv):*** lv	

2. Dentro de la clase que corresponda a sus vértices implemente el método *getNeighborListOf*() que debe devolver un *Set* con los vértices vecinos del objeto vértice actual.
3. Dentro de la clase que corresponda a su grafo, implemente los métodos que corresponden con la interfaz *AStarGraph*.

Ejemplo:



El camino óptimo desde la casilla (3,0) a la casilla (0,0) tiene 9 movimientos y su peso es de 19.



Solución:

1. **Respuesta:**

Problema del Puzzle: Grafo Virtual y A*	
Propiedades Compartidas	*CE=(xe,ye), Casilla de entrada* *CS=(xs,ys), Casilla de salida* *N, tamaño del laberinto, derivada* *Movs, Conjunto<PairInteger>,* {(0,1),(0,-1),(1,0),(-1,0)} Tablero, Integer[0..N-1][0..N-1] 1- Casilla negra 2- Casilla roja
Propiedades del vértice	*i0, entero en [0,N), coordenada i de la casilla actual* *j0, entero en [0,N), coordenada j de la casilla actual*
Aristas**: La arista a representa el posible movimiento en i,j de la casilla actual.* $A=\{a\in Movs/(i,j)=((i0,j0)+a)\in[0,N)x[0,N)\}$ ***Vecinos: $\{(i0+a.i, j0+a.j)\ \forall a\in A\}$ ***Peso de la arista(e):*** 1 ***Peso del vértice(v):*** $tablero(v.i0, v.j0)=2?10:0$ ***Peso de la trayectoria(v,ein,eout):*** 0 ***Heurística(v, vf):*** $\lvert v.i0-vf.i0\rvert+\lvert v.j0-vf.j0\rvert$ ***Vértice Origen:*** CE ***Vértice Objetivo***: CS ***Solución (lv):*** lv	

2. **Respuesta:**

```
public Set<Casilla> getNeighborListOf() {
    return movs.stream()
      .filter(x -> x.v1 + i0 >= 0 && x.v1 + i0 < ProblemaLaberinto.N)
      .filter(x -> x.v2 + j0 >= 0 && x.v2 + j0 < ProblemaLaberinto.N)
      .map(x ->getVecino(x.v1, x.v2)))
      .collect(Collectors.toSet());
}

public Casilla getVecino(int incx, int incy){
    int f = this.i0+incx;
    int c = this.j0+incy;
    if(f<0 || f>=ProblemaLaberinto.N || c<0 ||c>=ProblemaLaberinto.N)
          throw new IllegalArgumentException();
   return new Casilla(f, c);
}
```

3. **Respuesta:**

```
public double getEdgeWeight(SimpleEdge<Casilla>  edge) {
      return 1.;
}

public double getVertexWeight(Casilla vertex) {
  return ProblemaLaberintoRN.tablero[vertex.i0][vertex.j0]==2?10:0;
}

public double getVertexWeight(Casilla vertex,
      SimpleEdge<Casilla> edgeIn,
      SimpleEdge<Casilla> edgeOut) {
   return 0;
}

public double getWeightToEnd(Casilla startVertex, Casilla endVertex) {
   if(startVertex==null || endVertex==null)
      throw new IllegalArgumentException();
   return Math.abs(startVertex.i0-endVertex.i0) +
          Math.abs(startVertex.j0-endVertex.j0);
}
```