Práctica Programación 2 (Curso 2023/2024)
Parte 2: Hormiga de Langton Modificada.

- La práctica se debe entregar en **DeliverIt** donde se realizarán una serie de pruebas unitarias automáticas.
- Para que los archivos de código fuente compilen se debe respetar estrictamente los nombres de las clases y paquetes.
- Adicionalmente, el alumno puede comprobar el resultado del código fuente gráficamente completando los métodos de dibujado que sean necesarios.
- El dibujado **no** se comprueba en las pruebas automáticas.

1. Descripción general del problema

Se desea desarrollar varias clases necesarias para una aplicación que muestre la evolución de una versión **modificada** de la *Hormiga de Langton*. Para ello partiremos de las clases desarrolladas en la **Parte 1** y que se suponen resueltas. Las nuevas reglas de juego, incluidas las modificaciones son:

1. La hormiga se mueve en una cuadrícula bidimensional de casillas cuadradas, cada una de las cuales puede ser de tres colores: blanco, negro y rojo, codificados por los números 0, 1 y 2.
2. La hormiga siempre está orientada hacia arriba, abajo, izquierda o derecha.
3. En cada paso, la hormiga aplica las siguientes operaciones:
   a) Si está sobre un cuadrado blanco (color 0) gira noventa grados a la izquierda, cambia el color del cuadrado (a color 1) y avanza un cuadrado.
   b) Si está sobre un cuadrado negro (color 1) gira noventa grados a la derecha, cambia el color del cuadrado (a color 2) y avanza un cuadrado.
   c) Si está en una casilla de color rojo (color 2) gira 180 grados, cambia el color del cuadrado (a color 0) y avanza una casilla.
4. La cuadrícula de casillas tiene un tamaño de 11x11. Si una hormiga se sale de la cuadrícula da media vuelta y avanza una casilla.

Además de esta hormiga se va a hacer a otra todavía un poco más compleja: hormiga reiterativa. Esta hormiga repite la secuencia de movimientos periodicamente, es decir, primero se mueve como la hormiga modificada, pero cuando alcanza el número de movimientos dados repite toda la secuencia sin importar el color de las casillas.

Atención: Sólo se puede hacer la parte 2 si se ha completado la Parte 1.
Las clases de la parte 1 se utilizan en esta continuación.

2. Modelo de Datos

Para realizar el programa se ha decidido escribir las siguientes clases que constituyen las correspondientes abstracciones de datos. Se supone que se dispone de todas las clases e interfaces desarrolladas en la **Parte 1**.

Clases e interfaces para las Hormigas de Langton modificadas

CasillaTresColores. Hereda de Casilla y añade que puede tener un color adicional: el rojo. Cuando se construye una casilla su color inicial debe ser aleatorio generado con Random y con la semilla inicializada a 31416. Tiene la siguiente especificación:

```
// Consulta: https://docs.oracle.com/javase/8/docs/api/java/util/Random.html
static final Random r = new Random(31416);

/** Construye una casilla de color aleatorio:
 *
 * Se obtiene un numero aleatorio entre 0 y 99 (ambos inclusives) usando
 * el objeto 'r' y su metodo nextInt si aleatorio es menor estrictamente
```

```
 * que 20 el color de la casilla sera el 1, si esta entre 20 y 39 (ambos inclusives)
 * sera 2 y sera 0 en cualquier otro caso.
 */
public CasillaTresColores ()  ;

/** Cambia el color de 'esta' casilla.
 *
 * Si es 0 pasa a 1, si es 1 a 2 y si es 2 a 3.
 * Atencion: NO hace falta utilizar if para resolver esto.
 */
@Override
public void cambiarColor ()  ;
```

CuadriculaTresColores. La cuadrícula contiene un `array` de `CasillaTresColores`. Tiene un constructor para crear todas las casillas y un método para devolver la casilla en unas coordenadas dadas que lanza una excepción si se intenta acceder a una casilla fuera de la cuadrícula (la hormiga se ha salido de la cuadrícula). Esta clase debe implementar la interfaz `ICuadricula`. La especificación es:

```
// Tamanho de la matriz que guarda las casillas, DIMxDIM
public static final int DIM = 11;

// Array bidimensional de tamanho DIMxDIM que guarda las casillas
protected CasillaTresColores[][] casillas;

/** Construye e inicializa una nueva cuadricula
 */
public CuadriculaTresColores ()  ;

/** Devuelve la casilla  de 'esta' cuadricula en los indices de fila y columna dados por i y j.
 *  Se supone que los indices son las coordenadas (x, y) de la casilla en un plano 2D.
 * @throws IndexOutOfBoundsException cuando i o j no estan en el rango [0,Dim) */
@Override
public Casilla casilla (int i, int j)  ;
```

HormigaModificada. Hereda de `Hormiga`. Esta hormiga implementa el movimiento según las reglas modificadas. **Solamente** hay que implementar el método `girar`, todo lo demás se reutiliza de la clase base. Su especificación es:

```
/** 'Esta' hormiga gira su orientacion segun el color de la casilla que ocupa
*
*  Segun las reglas del juego gira en un sentido u otro dependiendo del
*  color de la casilla,
*  @param cuadricula la cuadricula donde se encuentra la hormiga y que
*  se utiliza para obtener la casilla.
*  @throws IllegalStateException si el color de la casilla no es
*  uno de los que la hormiga reconoce en sus instrucciones de actuacion.
*/
@Override
public Giro girar (ICuadricula cuadricula)  ;
```

HormigaReiterativa. La hormiga reiterativa hereda de la `HormigaModificada`, las reglas de comportamiento de esta hormiga son:

1. Esta hormiga tiene dos estatus: moviendo y reiterando (se van a representar por un enumerado).
2. El estatus de la hormiga cambia cada 5 movimientos (se declara un atributo de clase para guardar este dato).
3. Cuando la hormiga está en el estatus *moviendo* almacena en la cola el giro que hace cada vez que se llama a `girar` y cuenta un movimiento. El giro es exactamente igual que el que realiza la `HormigaModificada`.
4. Cuando la hormiga está en el estatus *reiterando* se mira el primer giro que toca en la cola donde se almacenan los giros y se elimina. Si la hormiga está dentro de la cuadrícula se realiza el giro obtenido de la cola, en caso contrario se da *media vuelta*.

Su especificación es:

```
private static enum Estatus { MOVIENDO, REITERANDO }
private Estatus estatus;

// Numero de movimientos en el estatus MOVIENDO
private int movimientos = 0;

// Numero de movimientos para alternar entre estatus
// (para depurar se puede poner un valor mas bajo, por ejemplo: 3
public static int vecesParaAlternar = 5;

// La secuencia para guardar los giros
private final IQueue<Giro> giros = new LinkedQueue<>();

/** Construye la hormiga
 *
 * Inicializar el estatus a MOVIENDO.
 */
public HormigaReiterativa ()  ;

/** Comprueba si hay que cambiar el estatus y lo hace si fuese necesario.
 *
 * - Si el estatus es MOVIENDO y el numero de movimientos es ‘vecesParaAlternar‘
 *   se cambia el estatus, se fija ‘movimientos‘ a 0 y se pasa el estatus a
 *   REITERANDO.
 * - Si el estatus es REITERANDO y cola esta vacia se cambia el estatus
 *   a MOVIENDO.
 */
public void cambiarEstatus()  ;

/** Gira ‘esta‘ hormiga segun sus reglas.
*
*  En este metodo lo primero es llamar a ‘cambiarEstatus‘.
*  Importante: La implementacion es mas simple si se usa girar de
*  la clase _base_ (HormigaModificada).
*
*  @param cuadricula la cuadricula donde se encuentra la hormiga y que
*  se utiliza para obtener la casilla.
*  @throws IllegalStateException si el color de la casilla no es
*  uno de lo que la hormiga reconoce en sus instrucciones de actuacion.
*  Exactamente igual que en el caso de la HormigaModificada.
*/
@Override
public Giro girar (ICuadricula cuadricula)  ;

/** Ruta de la imagen a mostrar.
 * @return el archivo "res/ant_1.png" si ‘esta‘ se esta moviendo y "res/ant_2.png"
 * en caso contrario.
 */
@Override
public String rutaDeLaImagen ()  ;

/* ATENCION: Es necesario dejar este metodo exactamente asi.
 * Este se utiliza en las pruebas para comprobar el estado interno
 * de la hormiga (inaccesible de otra manera).
 * Si no se mantiene tal cual algunas pruebas no podran evaluar
 * correctamente el codigo fuente propuesto */
@Override
public String toString() {
  return String.format("st: %d, mov: %d, seq: %s, alt: %d",
      estatus.ordinal(), movimientos, giros.toString(), vecesParaAlternar);
}
```