EI1024/MT1024 "Programación Concurrente y Paralela" 2022–23	Entregable para Laboratorio
Nombre y apellidos (1):	
Nombre y apellidos (2):	la05_g
Tiempo empleado para tareas en casa en formato *h:mm* (obligatorio):	

Tema 07. *Thread Pools* e Interfaces Gráficas en Java

Tema 08. Concurrencia en Colecciones de Java

1 Se dispone del siguiente código que define una interfaz gráfica de tiro al blanco.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.lang.reflect.InvocationTargetException;
import java.util.*;
import java.util.concurrent.*;


// ================================================================
public class GUITiroAlBlanco {
  // ================================================================

  // Declaracion de constantes (para tamanyos de ventana).
  static final int  maxVentanaX = 800 ;
  static final int  maxVentanaY = 600 ;

  // Declaracion de variables.
  JFrame            jframe;
  JPanel            jpanel;
  CanvasCampoTiro   cnvCampoTiro;
  JTextField        txfInformacion;
  JTextField        txfVelocidadInicial;
  JTextField        txfAnguloInicial;
  JButton           btnDispara;
  Point             objetivo;

  // ----------------------------------------------------------------
  public static void main( String args[] ) {
    GUITiroAlBlanco gui = new GUITiroAlBlanco();
    gui.go();
  }

  // ----------------------------------------------------------------
  public void go() {
    SwingUtilities.invokeLater( new Runnable() {
      public void run() {
        generaGUI();
      }
    } );
  }

  // ----------------------------------------------------------------
```

```
public void generaGUI() {

  // Declaracion de variables locales.
  JPanel  tablero, informacion, controles, incdec;
  JButton btnVelInc100, btnVelDec100, btnVelInc5, btnVelDec5;
  double  velIni, angIni;
  Font    miFuenteP, miFuenteM, miFuenteG;

  // Crea el JFrame principal.
  jframe = new JFrame( "GUI Tiro Al Blanco " );
  jpanel = ( JPanel ) jframe.getContentPane();
  jpanel.setPreferredSize( new Dimension( maxVentanaX, maxVentanaY ) );
  jpanel.setLayout( new BorderLayout() );

  //
  // Creacion del canvas para el campo de tiro.
  //
  cnvCampoTiro = new CanvasCampoTiro();

  //
  // Creacion del panel de informacion (aciertos, fallos, etc.).
  //
  informacion = new JPanel();
  informacion.setLayout( new FlowLayout() );

  // Crea y anyade el campo de mensajes.
  JLabel labInformacion = new JLabel( "Informacion:" );
  miFuenteM = labInformacion.getFont().deriveFont( Font.PLAIN, 15.0F );
  labInformacion.setFont( miFuenteM );
  informacion.add( labInformacion );

  txfInformacion = new JTextField( 45 );
  txfInformacion.setFont( miFuenteM );
  txfInformacion.setEditable( false );
  txfInformacion.setHorizontalAlignment( JTextField.CENTER );
  informacion.add( txfInformacion );

  //
  // Creacion del panel de controles de disparo.
  //
  controles = new JPanel();
  controles.setLayout( new FlowLayout() );

  // Crea y anyade el control de velocidad inicial.
  JLabel labVelocidadInicial = new JLabel( "Velocidad: " );
  miFuenteG = labVelocidadInicial.getFont().deriveFont( Font.PLAIN, 18.0F );
  labVelocidadInicial.setFont( miFuenteG );
  controles.add( labVelocidadInicial );

  velIni = 100.0 * Math.round( 50.0 + Math.random() * 10.0 );
  txfVelocidadInicial  = new JTextField( String.valueOf( velIni ), 7 );
  txfVelocidadInicial.setFont( miFuenteG );
  controles.add( txfVelocidadInicial );

  // Creacion del minipanel de incrementos/decrementos.
  incdec = new JPanel();
  incdec.setLayout( new GridLayout( 2, 2 ) );

  // Crea y anyade el boton para incrementar la velocidad en 100.
  btnVelInc100 = new JButton( "+100" );
  miFuenteP = btnVelInc100.getFont().deriveFont( Font.PLAIN, 10.0F );
```

```
btnVelInc100.setFont( miFuenteP );
incdec.add( btnVelInc100 );

// Anyade el codigo para procesar la pulsacion del boton "btnVelInc100".
btnVelInc100.addActionListener( new ActionListener() {
  public void actionPerformed( ActionEvent e ) {
    // En las llamadas a getText/setText de objetos graficos aqui no hace
    // falta el invokeLater dado que este codigo lo ejecuta la
    // hebra event-dispatching.
    double vel;
    try {
      vel = Double.parseDouble( txfVelocidadInicial.getText().trim() );
      vel += 100.0;
      txfVelocidadInicial.setText( String.valueOf( vel ) );
    } catch( NumberFormatException ex ) {
      txfInformacion.setText( "ERROR: Numeros incorrectos." );
    }
  }
} );

// Crea y anyade el boton para incrementar la velocidad en 5.
btnVelInc5 = new JButton( "+5" );
btnVelInc5.setFont( miFuenteP );
incdec.add( btnVelInc5 );

// Anyade el codigo para procesar la pulsacion del boton "btnVelInc5".
btnVelInc5.addActionListener( new ActionListener() {
  public void actionPerformed( ActionEvent e ) {
    // En las llamadas a getText/setText de objetos graficos aqui no hace
    // falta el invokeLater dado que este codigo lo ejecuta la
    // hebra event-dispatching.
    double vel;
    try {
      vel = Double.parseDouble( txfVelocidadInicial.getText().trim() );
      vel += 5.0;
      txfVelocidadInicial.setText( String.valueOf( vel ) );
    } catch( NumberFormatException ex ) {
      txfInformacion.setText( "ERROR: Numeros incorrectos." );
    }
  }
} );

// Crea y anyade el boton para decrementar la velocidad en 100.
btnVelDec100 = new JButton( "-100" );
btnVelDec100.setFont( miFuenteP );
incdec.add( btnVelDec100 );

// Anyade el codigo para procesar la pulsacion del boton "btnVelDec100".
btnVelDec100.addActionListener( new ActionListener() {
  public void actionPerformed( ActionEvent e ) {
    // En las llamadas a getText/setText de objetos graficos aqui no hace
    // falta el invokeLater dado que este codigo lo ejecuta la
    // hebra event-dispatching.
    double vel;
    try {
      vel = Double.parseDouble( txfVelocidadInicial.getText().trim() );
      vel -= 100.0;
      txfVelocidadInicial.setText( String.valueOf( vel ) );
    } catch( NumberFormatException ex ) {
      txfInformacion.setText( "ERROR: Numeros incorrectos." );
    }
```

```
    }
  } );

  // Crea y anyade el boton para decrementar la velocidad en 5.
  btnVelDec5 = new JButton( "-5" );
  btnVelDec5.setFont( miFuenteP );
  incdec.add( btnVelDec5 );

  // Anyade el codigo para procesar la pulsacion del boton "btnVelDec5".
  btnVelDec5.addActionListener( new ActionListener() {
    public void actionPerformed( ActionEvent e ) {
      // En las llamadas a getText/setText de objetos graficos aqui no hace
      // falta el invokeLater dado que este codigo lo ejecuta la
      // hebra event-dispatching.
      double vel;
      try {
        vel = Double.parseDouble( txfVelocidadInicial.getText().trim() );
        vel -= 5.0;
        txfVelocidadInicial.setText( String.valueOf( vel ) );
      } catch( NumberFormatException ex ) {
        txfInformacion.setText( "ERROR: Numeros incorrectos." );
      }
    }
  } );

  // Anyade el nuevo minipanel de incrementos/decrementos al panel de control.
  controles.add( incdec );

  // Crea y anyade un cierto espacio de separacion.
  JLabel labSeparacion1 = new JLabel( "     " );
  labSeparacion1.setFont( miFuenteG );
  controles.add( labSeparacion1 );

  // Crea y anyade el control del angulo inicial.
  JLabel labAnguloInicial = new JLabel( "angulo: " );
  labAnguloInicial.setFont( miFuenteG );
  controles.add( labAnguloInicial );

  angIni = Math.round( 45.0 + Math.random() * 15.0 );
  txfAnguloInicial = new JTextField( String.valueOf( angIni ), 5 );
  txfAnguloInicial.setFont( miFuenteG );
  controles.add( txfAnguloInicial );

  // Crea y anyade un cierto espacio de separacion.
  JLabel labSeparacion2 = new JLabel( "     " );
  labSeparacion2.setFont( miFuenteG );
  controles.add( labSeparacion2 );

  // Crea y anyade el boton de disparo.
  btnDispara = new JButton( "Dispara" );
  btnDispara.setFont( miFuenteG );
  controles.add( btnDispara );

  //
  // Creacion del panel de tablero que contiene los minipaneles de
  // informacion y controles.
  //
  tablero = new JPanel();
  tablero.setLayout( new BorderLayout() );
  tablero.add( "Center", informacion );
  tablero.add( "South", controles );
```

```
    //
    // Anyade el canvas y el tablero al panel principal.
    //
    jpanel.add( "Center", cnvCampoTiro );
    jpanel.add( "South", tablero );

    // Fija caracteristicas del frame.
    jframe.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
    jframe.pack();
    jframe.setResizable( false );
    jframe.setVisible( true );

    // Inicializa la posicion del objetivo.
    this.objetivo = generaCoordenadasDeObjetivo();
    System.out.println( "generaGUI. Coordenadas del objetivo: " +
              this.objetivo.x + "," + this.objetivo.y);
    cnvCampoTiro.guardaCoordenadasObjetivo( this.objetivo );

/* ================ CODIGO A MODIFICAR ========================== */
    // Anyade el codigo para procesar la pulsacion del boton "Dispara".
    btnDispara.addActionListener( new ActionListener() {
      public void actionPerformed( ActionEvent e ) {
        // En las llamadas a getText/setText de objetos graficos aqui no hace
        // falta el invokeLater dado que este codigo lo ejecuta la
        // hebra event-dispatching.
        double vel, ang;
        try {
          vel = Double.parseDouble( txfVelocidadInicial.getText().trim() ) / 100.0;
          ang = Double.parseDouble( txfAnguloInicial.getText().trim() );
          if( ( 0.0 <= ang )&&( ang < 90 )&&( vel > 0 ) ) {
            txfInformacion.setText( "Calculando y dibujando trayectoria..." );
            creaYMueveProyectil( new NuevoDisparo( vel, ang ) );
          } else {
            txfInformacion.setText( "ERROR: Datos incorrectos." );
          }
        } catch( NumberFormatException ex ) {
          txfInformacion.setText( "ERROR: Numeros incorrectos." );
        }
      }
    } );
/* ================ FIN CODIGO A MODIFICAR ====================== */
  }

  // ---------------------------------------------------------------------
  Point generaCoordenadasDeObjetivo() {
    int      maxDimX, maxDimY, distanciaAlBorde, objetivoX, objetivoY;
    double   mitadX, posicionX;

    // Obten las dimensiones del canvas.
    maxDimX = cnvCampoTiro.getWidth();
    maxDimY = cnvCampoTiro.getHeight();

    // Genera una posicion aleatoria en la segunda mitad.
    mitadX  = ( ( double ) ( maxDimX - 1 ) ) / 2.0 ;
    posicionX = Math.round( mitadX + Math.random() * mitadX );

    // Controla que el objetivo no esta muy cerca de los bordes.
    distanciaAlBorde = 50;
    objetivoX = Math.max( distanciaAlBorde,
              Math.min( maxDimX - distanciaAlBorde,
```

```
                    ( int ) posicionX ) );
  objetivoY = 0;

  return new Point( objetivoX, objetivoY );
}

//-----------------------------------------------------------------------
public void creaYMueveProyectil( NuevoDisparo d ) {
  Proyectil  p;
  boolean       impactado;

  p = new Proyectil( d.velocidadInicial, d.anguloInicial, cnvCampoTiro );
  impactado = false;
  while( ! impactado ) {
    // Muestra en pantalla los datos del proyectil p.
    p.imprimeEstadoProyectilEnConsola();

    // Mueve el proyectil p.
    p.mueveUnIncremental();

    // Dibuja el proyectil p.
    p.actualizaDibujoDeProyectil();

    // Comprueba si el proyectil p ha impactado o continua en vuelo.
    impactado = determinaEstadoProyectil( p );

    duermeUnPoco( 1L );
  }
}

//-----------------------------------------------------------------------
boolean determinaEstadoProyectil( Proyectil p ) {
  // Devuelve cierto si el proyectil ha impactado contra el suelo o contra
  // el objetivo.
  boolean  impactado;
  String   mensaje;

  if ( ( p.intPosX == objetivo.x )&&( p.intPosY == objetivo.y ) ) {
    // El proyectil ha acertado el objetivo.
    impactado = true;

    mensaje = " Destruido!!!";
    muestraMensajeEnCampoInformacion( mensaje );

  } else if( ( p.intPosY <= 0 )&&( p.velY < 0.0 ) ) {
    // El proyectil ha impactado contra el suelo, pero no ha acertado.
    impactado = true;

    mensaje = "Has fallado. Esta en " + objetivo.x + ". " +
    "Has disparado a " + p.intPosX + ".";
    muestraMensajeEnCampoInformacion( mensaje );
  } else {
    // El proyectil continua en vuelo.
    impactado = false;
  }
  return impactado;
}

//-----------------------------------------------------------------------
void muestraMensajeEnCampoInformacion( String mensaje ) {
  // Muestra mensaje en el cuadro de texto de informacion.
```

```
    String miMensaje = mensaje;
    txfInformacion.setText( miMensaje );
  }

  //-----------------------------------------------------------------------
  void duermeUnPoco( long millis ) {
    try {
      Thread.sleep( millis );
    } catch( InterruptedException ex ) {
      ex.printStackTrace();
    }
  }
}

//=========================================================================
class CanvasCampoTiro extends Canvas {
  //=======================================================================

  // Declaracion de constantes.
  static final int  tamProyectil  = 5;
  static final int  tamObjetivoX  = 20;
  static final int  tamObjetivoY  = 30;
  static final int  tamCanyonX  = 40;
  static final int  tamCanyonY  = 40;

  // Declaracion de variables.
  int objetivoX, objetivoY;

  //-----------------------------------------------------------------------
  public void paint( Graphics g ) {

    // Fija el color de fondo.
    this.setBackground( Color.gray );

    // Dibuja el borde.
    g.setColor( Color.black );
    g.drawRect( 0, 0, this.getWidth() - 1, this.getHeight() - 1 );

    // Dibuja el canyon y el objetivo.
    dibujaCanyon( 0, 0 );
    dibujaObjetivo( objetivoX, objetivoY );
  }

  //-----------------------------------------------------------------------
  public void dibujaProyectil( int x, int y, int xOld, int yOld ) {
    Graphics g = this.getGraphics();
    // Borra posicion anterior.
    g.setColor( Color.white );
    g.fillOval( coorX( xOld ), coorY( yOld ), tamProyectil, tamProyectil );
    // Dibuja posicion nueva.
    g.setColor( Color.red );
    g.fillOval( coorX( x ), coorY( y ), tamProyectil, tamProyectil );
  }

  //-----------------------------------------------------------------------
  public void dibujaCanyon( int x, int y ) {
    Graphics g = this.getGraphics();
    g.setColor( Color.green );
    g.fillOval( coorX( x ) - tamCanyonX / 2, coorY( y ) - tamCanyonY / 2,
          tamCanyonX, tamCanyonY );
```

```
  }

  // ------------------------------------------------------------------------
  public void dibujaObjetivo( int x, int y ) {
    Graphics g = this.getGraphics();
    g.setColor( Color.yellow );
    g.fillRect( coorX( x ) - tamObjetivoX / 2, coorY( y ) - tamObjetivoY / 2,
          tamObjetivoX, tamObjetivoY );
  }

  // ------------------------------------------------------------------------
  int coorX( int x ) {
    return x;
  }

  // ------------------------------------------------------------------------
  int coorY( int y ) {
    return ( this.getHeight() - 1 - y );
  }

  // ------------------------------------------------------------------------
  void guardaCoordenadasObjetivo( Point objetivo ) {
    this.objetivoX = objetivo.x;
    this.objetivoY = objetivo.y;
  }
}

// ============================================================================
class NuevoDisparo {
  // ==========================================================================

  final double velocidadInicial, anguloInicial;

  // ------------------------------------------------------------------------
  public NuevoDisparo( double velocidadInicial, double anguloInicial ) {
    this.velocidadInicial = velocidadInicial;
    this.anguloInicial  = anguloInicial;
  }
}

// ============================================================================
class Proyectil {
  // ==========================================================================

  // Declaracion de constantes.
  static final double GRAVITY = 9.8;
  static final double TO_RAD  = ( 2.0 * Math.PI ) / 360.0;
  static final double DELTA_T = 5.0E-3;

  // Declaracion de variables.
  CanvasCampoTiro cnvCampoTiro;
  // Posiciones, angulo y velocidades con precision doble.
  double      posX, posY;
  double      anguloRad, velX, velY;
  // Posiciones exactas enteras.
  int         intPosX, intPosY, intPosXOld, intPosYOld;

  // ------------------------------------------------------------------------
  Proyectil( double velocidadInicial, double anguloInicial,
        CanvasCampoTiro cnvCampoTiro ) {
    this.posX     = 0.0;
```

```
    this.posY      = 0.0;
    this.anguloRad  = anguloInicial * TO_RAD;
    this.velX      = Math.cos( anguloRad ) * velocidadInicial;
    this.velY      = Math.sin( anguloRad ) * velocidadInicial;
    this.cnvCampoTiro = cnvCampoTiro;
  }

  //-------------------------------------------------------------------------
  void mueveUnIncremental() {
    // Actualiza la posicion y la velocidad.
    this.posX += this.velX * DELTA_T;
    this.posY += this.velY * DELTA_T;
    //// this.velX = this.velX;  Esta velocidad no cambia.
    this.velY -= GRAVITY * DELTA_T;

    // Guarda la anterior posicion entera.
    this.intPosXOld = intPosX;
    this.intPosYOld = intPosY;

    // Calcula la nueva posicion entera.
    this.intPosX = ( int ) posX;
    this.intPosY = ( int ) posY;
  }

  //-------------------------------------------------------------------------
  void imprimeEstadoProyectilEnConsola() {
    System.out.format( "  Pos:( %6.2f %6.2f )" +
              " Vel:( %6.2f %6.2f )" + " IntPos:( %4d %4d )%n",
              this.posX, this.posY,
              this.velX, this.velY, this.intPosX, this.intPosY );
  }

  //-------------------------------------------------------------------------
  public void actualizaDibujoDeProyectil() {
    // Dibuja la nueva posicion del proyectil solo si la nueva posicion es
    // distinta de la anterior.
    if( ( this.intPosX != this.intPosXOld )||
        ( this.intPosY != this.intPosYOld ) ) {
/*
      final int finalIntPosX  = this.intPosX;
      final int finalIntPosY  = this.intPosY;
      final int finalIntPosXOld = this.intPosXOld;
      final int finalIntPosYOld = this.intPosYOld;
      cnvCampoTiro.dibujaProyectil( finalIntPosX, finalIntPosY,
                    finalIntPosXOld, finalIntPosYOld );
*/
      cnvCampoTiro.dibujaProyectil( intPosX, intPosY,
                      intPosXOld, intPosYOld );
    }
  }
}
```

Compila el código y pruébalo. Comprueba que una vez has disparado y mientras el proyectil se encuentra en movimiento, la aplicación no responde a ninguna acción (como por ejemplo intentar cambiar la velocidad, intentar cambiar el ángulo e intentar realizar otro disparo).

¿Cuál es la causa del problema?

.....

.....

2 El objetivo de este apartado es modificar la aplicación anterior para que la interfaz gráfica sea más interactiva.

Realiza una primera implementación en la que cada disparo sea movido por una hebra nueva.

Para ello, debes definir una nueva clase de hebras (subclase de la clase `Thread`) que se podría denominar, por ejemplo, `MiHebraCalculadoraUnDisparo`. El constructor de esta clase debe recibir cuatro argumentos: **el canvas**, **el cuadro de texto de mensajes**, **el nuevo disparo** y **el objetivo**. Por su parte, el código del método `run` de la hebra debe crear un proyectil a partir del disparo recibido y moverlo hasta que alcance el suelo (ver método `creaYMueveProyectil`).

Así, cada vez que el usuario pulse el botón de disparo, la hebra *event-dispatching* debe comprobar los parámetros, y, si son correctos, debe crear un nuevo disparo (`d`). Seguidamente debe crear una hebra auxiliar (`t`) que se encargue de mover el proyectil asociado al disparo.

En la descripción anterior, aparecen situaciones que pueden generar problemas de visibilidad y/o de atomicidad, cuya resolución puede modificar tu implementación. Con este fin, hay que analizar el código, localizar qué líneas pueden ser problemáticas, y actuar en consecuencia.

Seguidamente se plantean cuestiones que ayudan a detectar y resolver estos problemas:

- Cuando se crea y se arranca la hebra auxilar. ¿Se producen problemas de visibilidad y/o de atomicidad? Razona tu respuesta.

- Cuando se crea el proyectil. ¿Se producen problemas de visibilidad y/o de atomicidad? Razona tu respuesta.

- Cuando se actualiza la trayectoria del proyectil (método `actualizaDibujoDeProyectil`). ¿Hay un objeto gráfico? ¿Quién debe actualizarlo? ¿Se producen problemas de visibilidad y/o de atomicidad? Razona tu respuesta.

- Cuando se imprime el estado de un proyectil en el cuadro de texto de mensajes (método `muestraMensajeEnCampoInformacion`). ¿Hay un objeto gráfico? ¿Quién debe actualizarlo? ¿Se producen problemas de visibilidad y/o de atomicidad? Razona tu respuesta.

Escribe a continuación la parte de tu código que realiza la tarea descrita: la definición de la clase `MiHebraCalculadora` y los cambios a introducir en el código del método `generaGUI`.

3 ¿Piensas que es realista la implementación? ¿Qué pasaría si varias hebras estuviesen moviendo diferentes proyectiles y una de ellas pierde la CPU?

. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .

4 El objetivo de este apartado es modificar la aplicación para que la interfaz gráfica sea más interactiva y también más realista.

Para lograrlo, todos los proyectiles deben ser movidos por **una única hebra auxiliar** que es creada junto con el interfaz gráfico, Esta hebra se debe bloquear mientras no haya ningún proyectil en el aire, es decir, no se puede emplear espera activa.

Con este objetivo, se pretende que la hebra auxiliar trabaja con dos colecciones.

Una **primera colección**, denominada lista de disparos (`listaD`), es utilizada por la hebra gráfica para comunicar a la hebra auxiliar los datos de los nuevos disparos (objetos de la clase `NuevoDisparo`) que el usuario ha producido. Esta colección debe ser *thread-safe* porque tanto la hebra gráfica como la hebra auxiliar accederán a la información que contiene.

Como la hebra auxiliar puede bloquearse a la espera de nuevos disparos (si no hay ningún proyectil en el aire), se puede emplear un objeto de la clase `LinkedBlockingQueue`. ¿Qué métodos de la clase `LinkedBlockingQueue` permiten realizar una inserción bloqueante y una extracción bloqueante en un objeto de esta clase?

. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .

La **segunda colección**, denominada lista de proyectiles (`listaP`), debe ser local a la hebra auxiliar, por lo que no necesita ser *thread-safe*. En esta lista, la hebra auxiliar guarda todos los proyectiles que están en el aire, y cuando un proyectil llega al suelo, debe eliminarlo de la lista. Periódicamente, la hebra auxiliar debe consultar la lista de disparos para comprobar si no está vacía, y en tal caso, coger los disparos, crear los proyectiles asociados e insertarlos en su lista local. Esta segunda colección debe ser de una clase que permita eliminar cualquier componente de la colección, puesto que no se conoce de antemano la posición de los proyectiles que llegan al suelo.

Una opción sería emplear, la clase `ArrayList`. ¿Qué métodos de la clase `ArrayList` permiten realizar una inserción y un borrado en un objeto de esta clase?

. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .

ATENCIÓN: Los ejercicios anteriores deben realizarse en casa. Los siguientes, en el aula.

A continuación se describe con más detalle el proceso iterativo que debe realizar la hebra auxiliar:

1. La hebra auxiliar, antes de proceder a mover todos los proyectiles que se encuentran en vuelo, debe comprobar si la lista de disparos no está vacía.

   Si la hebra gráfica ha dejado uno o varios nuevos disparos en la `listaD`, la hebra auxiliar debe extraerlos, crear los proyectiles e insertarlos en su lista local de proyectiles en el aire.

   Si la hebra gráfica no ha dejado trabajo y la lista local de proyectiles en el aire está vacía, la hebra auxiliar no puede hacer nada. En tal caso, la hebra debe bloquearse a la espera de recibir nuevos disparos en `listaD`, evitando realizar una espera activa, para lo cual debe utilizar el método adecuado.
2. Tras vaciar la lista de disparos, y en el caso que existan proyectiles en el aire, la hebra auxiliar debe mover **todos los proyectiles** que están en su lista de proyectiles en el aire como si hubiese transcurrido **un incremental de tiempo**.
3. Si algún proyectil de los que están en vuelo alcanza el suelo (y estalla), la hebra auxiliar debe eliminarlo de la lista local de proyectiles en el aire, utilizando el procedimiento adecuado.
4. Repetir los pasos anteriores hasta que la interfaz gráfica termine. Para ello, las acciones anteriores deben estar en un bucle infinito.

A continuación se muestra el algoritmo del cuerpo de la hebra en pseudo-código:

```
// Bucle infinito en el cuerpo de la hebra.
while( true )
  // Bucle para coger todos los nuevos disparos dejados por la hebra grafica.
  while( ( hayan nuevos disparos en la listaD ) || ( listaP este vacia ) ) {
    Tomar un nuevo disparo, bloqueandose si no hubiera.
    Crear el nuevo proyectil a partir del nuevo disparo.
    Anyadir el nuevo proyectil a listaP.
  }
  // Procesado de la lista local de proyectiles.
  for( todos los proyectiles de listaP) {
    Mostrar datos del proyectil en pantalla
    Mover un incremental de tiempo el proyectil actual.
    Actualizar la posicion del proyectil actual.
    Comprueba si el proyectil actual ha impactado en el suelo.
    if( Ha impactado sobre el suelo el proyectil actual ) {
      Eliminar el proyectil actual de listaP.
    }
  }
}
```

Una vez terminado el código, realiza las siguientes comprobaciones:

1. Comprueba que el nuevo código mueve simultáneamente varios disparos en el aire.
2. Comprueba que no hay espera activa: Ejecuta el comando `top` en unix o similar en *Windows*. Comprueba la carga mientras no hay proyectiles en el aire. Comprueba la carga mientras hay varios proyectiles en el aire.
3. Comprueba que la hebra auxiliar no acceda a ningún método de un objeto gráfico (excepto si estos aparecen dentro de: `invokeAndWait`, `invokeLater`, etc.).
4. Comprueba que la hebra auxiliar no acceda a ningún dato modificado por la hebra gráfica que no esté protegido con `synchronized` o con `volatile`, o que no sea `final`.
5. Comprueba que la hebra gráfica no acceda a ningún dato modificado por la hebra auxiliar que no esté protegido con `synchronized` o con `volatile`, o que no sea `final`.

Escribe a continuación la parte de tu código que realiza tal tarea: la definición de la clase `MiHebraCalculadora` y los cambios a introducir en el código del método `go`.