Nombre y apellido: Enric Gil Gallen

Nombre y apellido: Victor Granados Segara

Tiempo en casa: 1:10

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

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?

- Se están realizando demasiados cálculos costosos en la hierba **event-dispatching**

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.

- No, porque cuando se arranca la hebra auxiliar se produce una sincronización implícita.

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

- No,ni de visibilidad y/o de atomicidad porque es un objeto local a la hebra trabajadora.

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

  - Sí hay un objeto gráfico por lo que hay que utilizar un método invoke para que su gestión la haga la event-dispatching.

- 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.

  - Sí hay un objeto gráfico por lo que hay que utilizar un método invoke para que su gestión la haga la event-dispatching

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 `go`.

```
package e5;

import javax.swing.*;
import java.awt.*;
import java.lang.reflect.InvocationTargetException;

public class MiHebraCalculadoraUnDisparo extends  Thread{
   CanvasCampoTiro4 canvas;
   JTextField mensajeJT;
   NuevoDisparo4 disparo;
   Point objetivo;

   public MiHebraCalculadoraUnDisparo(CanvasCampoTiro4 canvas, JTextField mensaje,
NuevoDisparo4 disparo, Point objetivo) {
       this.canvas = canvas;
       this.mensajeJT = mensaje;
       this.disparo = disparo;
       this.objetivo = objetivo;
   }

   @Override
   public void run() {
       Proyectil1a p;
       boolean      impactado;

       p = new Proyectil1a( disparo.velocidadInicial, disparo.anguloInicial, canvas );
       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.actualizaDibujoProyectil();

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

            duermeUnPoco( 1L );
        }
    }

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

    boolean determinaEstadoProyectil( Proyectil1a 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!!!";
            SwingUtilities.invokeLater(new Runnable() {
                @Override
                public void run() {
                    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 + ".";
            SwingUtilities.invokeLater(new Runnable() {
                @Override
                public void run() {
                    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;
       mensajeJT.setText( miMensaje );
   }

}


 // ---------------------------------------------------------------------------
 public void actualizaDibujoProyectil() {
   // 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 );
*/
     SwingUtilities.invokeLater(new Runnable() {
       @Override
       public void run() {
         cnvCampoTiro.dibujaProyectil( intPosX, intPosY,
                 intPosXOld, intPosYOld );
       }
     });
   }
 }


// 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..." );
       new MiHebraCalculadoraUnDisparo(cnvCampoTiro,txfInformacion,  new NuevoDisparo4(
vel, ang ), objetivo ).start();
     } else {
       txfInformacion.setText( "ERROR: Datos incorrectos." );
     }
   } catch( NumberFormatException ex ) {
     txfInformacion.setText( "ERROR: Numeros incorrectos." );
```

```
    }
  }
} );
```

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

- No, la pérdida de CPU interrumpe la trayectoria de un proyectil.
- El orden de los proyectiles podría cambiar.

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 (`lD`), 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?

- put (Espera a que tenga espacio)
- take (Recupera y elimina) **BLOQUEANTE**

La **segunda colección**, denominada lista de proyectiles (`lP`), 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?

- add()
- remove()

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 `1D`, 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 `1D`, 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.

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`.

```
package e5;

import javax.swing.*;
import java.awt.*;
import java.lang.reflect.InvocationTargetException;
import java.util.ArrayList;
import java.util.concurrent.LinkedBlockingQueue;

public class MiHebraCalculadoraVariosDisparoEje4 extends Thread {
   CanvasCampoTiro4 canvas;
   JTextField mensajeJT;
```

```
    NuevoDisparo4 disparo;
    Point objetivo;
    LinkedBlockingQueue<NuevoDisparo4> lD;
    ArrayList<Proyectil4> lP;
    boolean impactado;

    public MiHebraCalculadoraVariosDisparoEje4(CanvasCampoTiro4 canvas,
JTextField mensajeJT, Point objetivo, LinkedBlockingQueue<NuevoDisparo4> lD) {
        this.canvas = canvas;
        this.mensajeJT = mensajeJT;
        this.objetivo = objetivo;
        this.lD = lD;
        this.lP = new ArrayList<>();
    }

    public void run() {
        while (true) {
            while (lP.size() == 0 || !lD.isEmpty()) {
                try {
                    disparo = lD.take();
                    Proyectil4 proyectil4 = new
Proyectil4(disparo.velocidadInicial, disparo.anguloInicial, canvas);
                    lP.add(proyectil4);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }

            for (int i = 0; i < lP.size(); i++) {
                Proyectil4 p = lP.get(i);

                // Muestra en pantalla los datos del proyectil p.
                p.imprimeEstadoProyectilEnConsola();

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

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

                impactado = determinaEstadoProyectil(p);

                duermeUnPoco(1L);

                if (impactado) {
                    lP.remove(p);
                    i--;
                }

            }
        }
```

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

boolean determinaEstadoProyectil( Proyectil4 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!!!";
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                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 + ".";
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                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;
            mensajeJT.setText( miMensaje );
        }
    }
```

GUITiroAlBlanco4

Línea 26

```
LinkedBlockingQueue<NuevoDisparo4> lD;
```

Línea 43

```
//--------------------------------------------------------------------------
public void generaGUI() {
    lD = new LinkedBlockingQueue<NuevoDisparo4>();
```

Línea 270

```
MiHebraCalculadoraVariosDisparoEje4 hebra = new
MiHebraCalculadoraVariosDisparoEje4(cnvCampoTiro, txfInformacion, objetivo, lD);
hebra.start();
```

Línea 520

```
SwingUtilities.invokeLater(new Runnable() {
    @Override
    public void run() {
        cnvCampoTiro.dibujaProyectil(intPosX, intPosY,
                intPosXOld, intPosYOld);
    }
});
```