1 Se desea calcular el número π mediante integración numérica de la siguiente función:

$$\pi = \int_0^1 \frac{4}{1+x^2} dx.$$

Este método no es el más rápido para calcular el número π, pero sí uno de las más simples. Consiste en calcular la anterior integral mediante una aproximación numérica basada en el cálculo y acumulación del área de numerosos rectángulos pequeños.

Uno de los parámetros más importantes es el número de rectángulos cuya área se va a sumar. En este caso, este parámetro será pasado en la línea de argumentos, después del número de hebras.

El siguiente programa realiza el cálculo de forma secuencial. Con vistas a facilitar el desarrollo posterior de la versión paralela, este código secuencial contiene un fragmento de código comentado, además de la declaración e inicialización de la variables `numHebras` y `numRectangulos`. Ambas partes no son útiles en la versión secuencial, pero, la inclusión de este fragmento de código simplifica el desarrollo de la versión paralela.

1.1) Estudia el código anterior y paralelízalo mediante el uso de hebras con una `distribución cíclica`. Utiliza un objeto de la clase `Acumula` para almacenar el resultado.

En esta versión paralela cada vez que las hebras calculan el área de un rectángulo, deben acumularlo sobre el objeto compartido de la clase `Acumula`. Para un correcto manejo del programa, hay que asegurar que el acceso al objeto compartido sea *thread-safe*.

No crees un nuevo programa. Haz que esta implementación paralela se ejecute a continuación de la versión secuencial dentro del mismo programa. Ello permitirá obtener los tiempos y los incrementos de velocidad de forma más rápida y automatizada.

Escribe a continuación la parte de tu código que realiza esta tarea: la definición de la clase `MiHebraMultAcumulaciones` y el código incluido en el programa principal que permite gestionar los objetos de esta clase.

1.1)

```
class Acumula {
//
================================================================================
  double  suma;

  // ------------------------------------------------------------------------
  Acumula() {
    this.suma = 0;
  }

  // ------------------------------------------------------------------------
  synchronized void acumulaDato( double dato ) {
    this.suma +=dato;
  }

  // ------------------------------------------------------------------------
  synchronized double dameDato() {
    return this.suma;
  }
}
```

```
//
// ============================================================================
// =====
class MiHebraMultAcumulaciones extends Thread {
//
// ============================================================================
// =====
  int      miId, numHebras;
  long     numRectangulos;
  Acumula  a;

  // ---------------------------------------------------------------------------
  MiHebraMultAcumulaciones( int miId, int numHebras, long numRectangulos,Acumula a ) {
    this.miId=miId;
    this.numHebras=numHebras;
    this.numRectangulos=numRectangulos;
    this.a = a;
  }

  // ---------------------------------------------------------------------------
   public void run() {
    double baseRectangulo = 1.0 / ( ( double ) numRectangulos );
    for( long i = miId; i < numRectangulos; i+=numHebras ) {
      double x = baseRectangulo * ( ( ( double ) i ) + 0.5 );
      double suma = ( 4.0/( 1.0 + x*x ));
      a.acumulaDato(suma);
    }
  }
}
//
    // Calculo del numero PI de forma paralela:
    // Multiples acumulaciones por hebra.
    //
    System.out.println();
    System.out.print( "Comienzo del calculo paralelo: " );
    System.out.println( "Multiples acumulaciones por hebra." );
    t1 = System.nanoTime();
    a = new Acumula();
    vt = new MiHebraMultAcumulaciones[numHebras];
    for (int i = 0; i<numHebras;i++){
      MiHebraMultAcumulaciones v = new
MiHebraMultAcumulaciones(i,numHebras,numRectangulos,a);
      vt[i] = v;
      v.start();
    }
    for (MiHebraMultAcumulaciones miHebra : vt) {
      try {
```

```
      miHebra.join();
    } catch (InterruptedException ex) {
      ex.printStackTrace();
    }
  }
  pi=a.dameDato()*baseRectangulo;
  t2 = System.nanoTime();
  tPar = ( ( double ) ( t2 - t1 ) ) / 1.0e9;
  System.out.println( "Calculo del numero PI:   " + pi );
  System.out.println( "Tiempo ejecucion (s.):   " + tPar );
  System.out.println( "Incremento velocidad :   " + tSec/tPar  );
```

1.2) Modifica el programa anterior, de modo que en la versión paralela las hebras acumulen el área que han calculado en una variable local (`sumaL`), antes de sumarla al objeto compartido.

No crees un nuevo programa. Haz que esta implementación paralela se ejecute a continuación de la versión paralela desarrollada en el apartado anterior. Ello permitirá obtener los tiempos y los incrementos de velocidad de forma más rápida y automatizada.

Escribe a continuación la parte de tu código que realiza esta tarea: la definición de la clase `MiHebraUnaAcumulacion` y el código incluido en el programa principal que permite gestionar los objetos de esta clase.

1.2)no se si se pide esto la verdad dudas

```
class MiHebraUnaAcumulacion extends Thread {
================================================================================
  int      miId, numHebras;
  long     numRectangulos;
  double sumaL;
  Acumula  a;

  MiHebraUnaAcumulacion( int miId, int numHebras, long numRectangulos,Acumula a ) {
    this.miId=miId;
    this.numHebras=numHebras;
    this.numRectangulos=numRectangulos;
    this.sumaL = 0.0;
    this.a = a;
  }
  public void run() {
    double baseRectangulo = 1.0 / ( ( double ) numRectangulos );
    for( long i = miId; i < numRectangulos; i+=numHebras ) {
      double x = baseRectangulo * ( ( ( double ) i ) + 0.5 );
      sumaL +=( 4.0/( 1.0 + x*x ) );
    }
    a.acumulaDato(sumaL);
  }

}
```

```
//
   // Calculo del numero PI de forma paralela:
   // Una acumulacion por hebra.
   //
   System.out.println();
   System.out.print( "Comienzo del calculo paralelo: " );
   System.out.println( "Una acumulacion por hebra." );
   t1 = System.nanoTime();
   a = new Acumula();
   MiHebraUnaAcumulacion[] vt2;
   vt2 = new MiHebraUnaAcumulacion[numHebras];
   for (int i = 0; i<numHebras;i++){
     MiHebraUnaAcumulacion v = new
MiHebraUnaAcumulacion(i,numHebras,numRectangulos,a);
     vt2[i] = v;
     v.start();
   }
   for (MiHebraUnaAcumulacion miHebra : vt2) {
     try {
       miHebra.join();
     } catch (InterruptedException ex) {
       ex.printStackTrace();
     }
   }
   pi=a.dameDato()*baseRectangulo;
   t2 = System.nanoTime();
   tPar = ( ( double ) ( t2 - t1 ) ) / 1.0e9;
   System.out.println( "Calculo del numero PI:   " + pi );
   System.out.println( "Tiempo ejecucion (s.):   " + tPar );
   System.out.println( "Incremento velocidad :   " + tSec/tPar );
```

1.3) En las **DOS** versiones paralelas anteriores se ha utilizado un objeto de la clase `Acumula` que permite acumular números reales con precisión doble de forma atómica, pero también se podría realizar empleando clases y operadores atómicos, como `DoubleAdder`.
Para ello, define las clases `MiHebraMultAcumulacionesAtomic` y `MiHebraUnaAcumulacionAtomic`. Estas clases deben manejar un objeto de la clase `DoubleAdder` y acumular los valores con el método `add`, mientras que el valor final se obtiene con el método `sum`.
Escribe a continuación los cambios realizados en el código.
Ten en cuenta que la mejor opción sería **eliminar completamente** la clase `Acumula` y en su lugar utilizar la clase atómica.

1.3)

```
class MiHebraMultAcumulacionAtomica extends Thread {
```

```
//
================================================================================
=====
class MiHebraMultAcumulacionAtomica extends Thread {
  DoubleAdder suma;
  int      miId, numHebras;
  long     numRectangulos;


  MiHebraMultAcumulacionAtomica( int miId, int numHebras, long
numRectangulos,DoubleAdder a ) {
    this.miId=miId;
    this.numHebras=numHebras;
    this.numRectangulos=numRectangulos;
    this.suma = a;
  }
//
================================================================================
=====
   public void run() {
    double baseRectangulo = 1.0 / ( ( double ) numRectangulos );
    for( long i = miId; i < numRectangulos; i+=numHebras ) {
      double x = baseRectangulo * ( ( ( double ) i ) + 0.5 );
      suma.add( 4.0/( 1.0 + x*x ) );
    }
    suma.sum();
  }
}

//
================================================================================
=====
class MiHebraUnaAcumulacionAtomica extends Thread {
  DoubleAdder suma;
  int      miId, numHebras;
  long     numRectangulos;
  MiHebraUnaAcumulacionAtomica( int miId, int numHebras, long
numRectangulos,DoubleAdder a ) {
    this.miId=miId;
    this.numHebras=numHebras;
    this.numRectangulos=numRectangulos;
    this.suma = a;
  }
  //
================================================================================
=====
  public void run() {
    double baseRectangulo = 1.0 / ( ( double ) numRectangulos );
```

```
    for( long i = miId; i < numRectangulos; i+=numHebras ) {
      double x = baseRectangulo * ( ( ( double ) i ) + 0.5 );
      suma.add( 4.0/( 1.0 + x*x ) );
    }
  }
//
=========================================================================
=====
// ...
}


//
    // Calculo del numero PI de forma paralela:
    // Multiples acumulaciones por hebra (Atomica)
    //
    System.out.println();
    System.out.print( "Comienzo del calculo paralelo: " );
    System.out.println( "Multiples acumulaciones por hebra (At)." );
    t1 = System.nanoTime();
    DoubleAdder b = new DoubleAdder();
    MiHebraMultAcumulacionAtomica[] vt3;
    vt3 = new MiHebraMultAcumulacionAtomica[numHebras];
    for (int i = 0; i<numHebras;i++){
      MiHebraMultAcumulacionAtomica v = new
MiHebraMultAcumulacionAtomica(i,numHebras,numRectangulos,b);
      vt3[i] = v;
      v.start();
    }
    for (MiHebraMultAcumulacionAtomica miHebra : vt3) {
      try {
        miHebra.join();
      } catch (InterruptedException ex) {
        ex.printStackTrace();
      }
    }
    pi=(b.doubleValue()*baseRectangulo);
    t2 = System.nanoTime();
    tPar = ( ( double ) ( t2 - t1 ) ) / 1.0e9;
    System.out.println( "Calculo del numero PI:   " + pi );
    System.out.println( "Tiempo ejecucion (s.):   " + tPar );
    System.out.println( "Incremento velocidad :   " + tSec/tPar );



    //
    // Calculo del numero PI de forma paralela:
    // Una acumulacion por hebra (Atomica).
```

```
    //
    System.out.println();
    System.out.print( "Comienzo del calculo paralelo: " );
    System.out.println( "Una acumulacion por hebra (At)." );
    t1 = System.nanoTime();
    b = new DoubleAdder();
    MiHebraUnaAcumulacionAtomica[] vt4;
    vt4 = new MiHebraUnaAcumulacionAtomica[numHebras];
    for (int i = 0; i<numHebras;i++){
      MiHebraUnaAcumulacionAtomica v = new
MiHebraUnaAcumulacionAtomica(i,numHebras,numRectangulos,b);
      vt4[i] = v;
      v.start();
    }
    for (MiHebraUnaAcumulacionAtomica miHebra : vt4) {
      try {
        miHebra.join();
      } catch (InterruptedException ex) {
        ex.printStackTrace();
      }
    }
    pi=(b.sum()*baseRectangulo);
    t2 = System.nanoTime();
    tPar = ( ( double ) ( t2 - t1 ) ) / 1.0e9;
    System.out.println( "Calculo del numero PI:   " + pi );
    System.out.println( "Tiempo ejecucion (s.):   " + tPar );
    System.out.println( "Incremento velocidad :   " + tSec/tPar );

    System.out.println();
    System.out.println( "Fin de programa." );
  }
```

2 Se dispone de una interfaz gráfica con un cuadro de texto y dos botones denominados `Comienza secuencia` y `Cancela secuencia`. Por el momento, la interfaz no hace nada cuando el usuario realiza alguna acción sobre los botones o sobre el cuadro de texto.

La interfaz está definida por el siguiente código:

2.1) Modifica la interfaz gráfica para que los botones `Comienza secuencia` y `Cancela secuencia` se activen y desactiven (`setEnabled`) de acuerdo a la lógica de funcionamiento de la aplicación :

- Inicialmente el botón `Comienza secuencia` debe estar activado y el botón `Cancela secuencia` debe estar desactivado (modificar método `go`).
- Cuando se presione el botón `Comienza secuencia`, éste se desactiva y se activa el botón `Cancela secuencia` (modificar `ActionListener` del primero).
- Cuando se presione el botón `Cancela secuencia`, éste se desactiva y se activa el botón `Comienza secuencia` (modificar `ActionListener` del primero).

2.1)

```
// Anyade codigo para procesar el evento del boton de Comienza secuencia.
   btnComienzaSecuencia.addActionListener( new ActionListener() {
      public void actionPerformed( ActionEvent e ) {
        btnComienzaSecuencia.setEnabled(false);
        btnCancelaSecuencia.setEnabled(true);

        // ...
      }
   } );

   // Anyade codigo para procesar el evento del boton de Cancela secuencia.
   btnCancelaSecuencia.addActionListener( new ActionListener() {
      public void actionPerformed( ActionEvent e ) {
        btnCancelaSecuencia.setEnabled(false);
        btnComienzaSecuencia.setEnabled( true );
        // ...
      }
```

2.2) Modifica la anterior interfaz de tal forma que en cuanto el usuario pulse el botón `Comienza secuencia` el programa muestre la secuencia de números primos (comenzando por el 2, 3, 5, 7, 11, etc.) en el cuadro de texto. Para ello, la hebra *event-dispatching* deberá crear una hebra trabajadora (`t`) en la que delegará dicho trabajo.

En cuanto el usuario pulse el botón `Cancela secuencia`, la generación de la secuencia debe terminar. Para detener la hebra, se fijará un valor especial en un atributo de la hebra (`fin`), cuyo valor será revisado por ésta cada vez que se genere un nuevo número primo.

Además, cada vez que se presione el botón `Comienza secuencia` la secuencia se inicia desde el principio.

Seguidamente se muestra la estructura del cuerpo de la hebra.

```
// Estructura del cuerpo de la hebra
long i = 1L;
while ( ! fin ) {
  if ( esPrimo ( i ) ) {
    // imprime ( i );
  }
  i++;
}
```

Una hebra trabajadora no puede llamar directamente a ningún método de ningún objeto gráfico, ya que éstos sólo pueden ser manejados por la *event-dispatching*. Por tanto, cuando la hebra trabajadora desee realizar alguna escritura sobre el cuadro de texto (`txfMensajes`), debe utilizar los métodos `invokeAndWait` o `invokeLater`, que indican a la *event-dispatching* que labores debe realizar.

Estos métodos ejecutan un objeto Runnable que reciben como parámetro de entrada, el primer método bloquea a la hebras hasta que la *event-dispatching* finaliza, por lo que es necesario gestionar dos excepciones, mientras que el segundo no bloquea a la hebra.

Escribe a continuación la parte de tu código que realiza tal tarea: la definición de la clase `HebraTrabajadora` y el código a incluir en el programa principal que permite gestionar los objetos de esta clase.

```
public class HebraCalculadora extends Thread{
        private JTextField  txfMensajes;
        private AtomicBoolean fin;

        public HebraCalculadora(JTextField txfMensajes, AtomicBoolean fin) {
        this.txfMensajes = txfMensajes;
        this.fin = fin;
        }

        @Override
        public void run() {
        long i = 1L ;
        while (!fin.get()) {
        if (esPrimo(i)) {
                final long fnum = i;
                try {
                SwingUtilities.invokeAndWait(new Runnable() {
                public void run() {
                        txfMensajes.setText(Long.valueOf(fnum).toString());
```

```
                }
                });
                } catch (InterruptedException | InvocationTargetException e) {
                throw new RuntimeException(e);
                }
        }
        i ++;
        }
        }

        static boolean esPrimo( long num ) {
        boolean primo;
        if( num < 2 ) {
        primo = false;
        } else {
        primo = true;
        long i = 2;
        while( ( i < num )&&( primo ) ) {
                primo = ( num % i != 0 );
                i++;
        }
        }
        return( primo );
        }
}

HebraCalculadora  t;
AtomicBoolean fin = new AtomicBoolean(false);

 // Anyade codigo para procesar el evento del boton de Comienza secuencia.
        btnComienzaSecuencia.addActionListener( new ActionListener() {
        public void actionPerformed( ActionEvent e ) {
        fin.getAndSet(false);
        btnComienzaSecuencia.setEnabled(false);
        btnCancelaSecuencia.setEnabled(true);
        t = new HebraCalculadora(txfMensajes, fin);
        t.start();
        }
        } );

        // Anyade codigo para procesar el evento del boton de Cancela secuencia.
        btnCancelaSecuencia.addActionListener( new ActionListener() {
        public void actionPerformed( ActionEvent e ) {
        btnCancelaSecuencia.setEnabled(false);
        btnComienzaSecuencia.setEnabled( true );
        fin.getAndSet(true);
        }
        } );
```

2.3) Modifica el programa anterior de tal forma que se muestre en pantalla una barra de deslizamiento horizontal (`JSlider`) con la que el usuario pueda determinar la velocidad de generación de números primos (ver código inicial).
Si la barra está en un extremo, la hebra deberá generar números primos intercalando una demora (método `sleep`) de un segundo **tras la impresión en el cuadro de texto**. Si la barra está en el otro extremo, la hebra deberá generar números primos sin ninguna demora.
Se recomienda definir y emplear una nueva clase denominada `ZonaIntercambio`, a través de la cual se comunicarán la hebra gráfica y la hebra calculadora. La hebra gráfica escribirá valores en un objeto de dicha clase y la hebra calculadora tomará valores de dicho objeto.
Por último comentar que el tiempo de espera se expresa en milisegundos, y que el valor inicial definido en el código es 500.
Escribe a continuación la parte de tu código que realiza tal tarea: la definición de la clase `ZonaIntercambio`, el código para la gestión de la barra de desplazamiento, y los cambios en la clase `HebraTrabajadora`.

2.4) Se desea sustituir la barra de deslizamiento por dos botones adicionales: Un botón añadirá 0,1 segundos al tiempo de espera, mientras que el otro botón restará 0,1 segundos al tiempo de espera.
No hagas ninguna implementación, pero responde a la siguiente pregunta. ¿Se podría realizar dicha modificación sólo con el operador `volatile` o habría que recurrir al modificador `synchronized`? Justifica la respuesta.

3 Este ejercicio es una continuación del ejercicio 1.

3.1) Completa la siguiente tabla para 500 000 000 de rectángulos. Obtén los resultados para 4 hebras en el ordenador del aula. Obtén los resultados para 16 hebras en patan. Redondea los tiempos dejando sólo tres decimales y redondea los incrementos dejando dos decimales. Los resultados utilizando la clase atómica son **optativos**. Para obtenerlos, debes sustituir el objeto de la clase `Acumula` por un objeto de la clase atómica propia de Java 8, y utilizar sus métodos en la actualización.
Justifica los resultados obtenidos.

Ejecución con 500 000 000 rectángulos				
	4 hebras		16 hebras	
	Tiempo	Incremento	Tiempo	Incremento
Secuencial		—		—
Paralela: Múltiples acumul.				
Paralela: Una única acumul.				
Paralela: Múltiples acumul. (clase atom.)				
Paralela: Una única acumul. (clase atom.)				

	4 hebras	4 hebras	16 hebras	16 hebras
	tiempo	incremento	tiempo	incremento
secuencial	4.646		2.012	
Paralela multiples acumul.	20.340	0.228	105.926	0.019
Paralela una unica acumul	1.401	3.315	0.290	6.941
Paralela multiples acumul.(atom)	1.195	3.887	0.399	5.041
Paralela una unica acumul.(atom)	1.181	3.934	0.524	3.849