REPORTE – PRACTICA 2

Programación concurrente y paralela

14 DE FEBRERO DE 2025

ALUMNO: ENRIQUE HERNANDEZ LUNA
DOCENTE: DRA CARMEN CERON GARCIA

Introducción:

En esta práctica números 2 se realizaron los siguientes programas:

1. Realiza un programa de dos hilos para imprimir los números pares e impares usando los cambios de estado de los hilos, el hilo A imprime impares y el hilo B los pares y al final imprime la suma de los pares e impares.

2. Realizar de forma sincronizada y comunicada de forma alternada la cooperación de dos hilos para identificar que números son pares y que números son impares de una serie de 50 y escribe la suma de los pares e impares. Los números se generan de manera aleatoria.

3. Realizar de forma sincronizada y comunicada de forma alternada la cooperación de dos hilos para identificar que números son positivos y que números son negativos (forma aleatoria) de una serie de -100 a 100 y escribe la suma de cada hilo.

```java
package Practica2;

class Jardin {
  public synchronized void sembrar(String nombre) {
    for (int i = 1; i <= 10; i++) {
      System.out.println(nombre + " esta sembrando el arbol " + i);

      try {
          Thread.sleep(500);
      } catch (InterruptedException e) {
        e.printStackTrace();
      }

      System.out.println(nombre + " ha terminado de sembrar el arbol " + i);
    }
  }
}

class Hermano implements Runnable {
  private final Jardin jardin;
  private final String nombre;

  public Hermano(Jardin jardin, String nombre) {
    this.jardin = jardin;
    this.nombre = nombre;
  }

  @Override
  public void run() {
      jardin.sembrar(nombre);
  }
}

public class HermanoV1Runnable {
  public static void main(String[] args) {
    Jardin jardin = new Jardin();
    Hermano hermano1 = new Hermano(jardin, "Hermano 1");
    Hermano hermano2 = new Hermano(jardin, "Hermano 2");

    Thread t1 = new Thread(hermano1);
    Thread t2 = new Thread(hermano2);

    t1.start();
    try {
      t1.join();
    } catch (InterruptedException e) {
      e.printStackTrace();
    }

    t2.start();
    try {
      t2.join();
    } catch (InterruptedException e) {
      e.printStackTrace();
    }

    System.out.println("Ambos hermanos han terminado de sembrar los árboles");
  }
}

```

```java
package Practica2;

class Jardin {
  public synchronized void sembrar(String nombre) {
    for (int i = 1; i <= 10; i++) {
      System.out.println(nombre + " esta sembrando el arbol " + i);

      try {
        Thread.sleep(500);
      } catch (InterruptedException e) {
        e.printStackTrace();
      }

      System.out.println(nombre + " ha terminado de sembrar el arbol " + i);
    }
  }
}

class Hermano extends Thread {
  private final Jardin jardin;

  public Hermano(Jardin jardin, String nombre) {
    super(nombre);
    this.jardin = jardin;
  }

  @Override
  public void run() {
    jardin.sembrar(getName());
  }
}

public class HermanoV1Thread {
  public static void main(String[] args) {
    Jardin jardin = new Jardin();
    Hermano hermano1 = new Hermano(jardin, "Hermano 1");
    Hermano hermano2 = new Hermano(jardin, "Hermano 2");

    hermano1.start();

    try {
      hermano1.join();
    } catch (InterruptedException e) {
      e.printStackTrace();
    }

    hermano2.start();

    try {
      hermano2.join();
    } catch (InterruptedException e) {
      e.printStackTrace();
    }

    System.out.println("Ambos hermanos han terminado de sembrar los árboles");
  }
}

```

```java
package Practica2;

import javax.swing.*;
import java.awt.*;

class Jardin {
    private final JTextArea textArea;

    public Jardin(JTextArea textArea) {
        this.textArea = textArea;
    }

    public synchronized void sembrar(String nombre) {
        for (int i = 1; i <= 10; i++) {
            textArea.append(nombre + " está sembrando el árbol " + i + "\n");

            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

            textArea.append(nombre + " ha terminado de sembrar el árbol " + i + "\n");
        }
    }
}

class Hermano extends Thread {
    private final Jardin jardin;

    public Hermano(Jardin jardin, String nombre) {
        super(nombre);
        this.jardin = jardin;
    }

    @Override
    public void run() {
        jardin.sembrar(getName());
    }
}

public class HermanosV1ThreadGUI {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Hermanos Sembrando Árboles");
        JTextArea textArea = new JTextArea(20, 40);
        textArea.setEditable(false);
        JScrollPane scrollPane = new JScrollPane(textArea);
        frame.add(scrollPane, BorderLayout.CENTER);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.pack();
        frame.setVisible(true);

        Jardin jardin = new Jardin(textArea);
        Hermano hermano1 = new Hermano(jardin, "Hermano 1");
        Hermano hermano2 = new Hermano(jardin, "Hermano 2");

        hermano1.start();
        try {
            hermano1.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        hermano2.start();
        try {
```

```
            hermano2.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        textArea.append("Ambos hermanos han terminado de sembrar los árboles\n");
    }
}

```

```
package Practica2;

import javax.swing.*;
import java.awt.*;

class Jardin {
    private boolean turnoHermano1 = true;
    private final JTextArea textArea;

    public Jardin(JTextArea textArea) {
        this.textArea = textArea;
    }

    public synchronized void sembrar(String nombre, boolean esHermano1) {
        for (int i = 1; i <= 10; i++) {
            synchronized (this) {
                while (turnoHermano1 != esHermano1) {
                    try {
                        wait();
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }

                textArea.append(nombre + " está sembrando el árbol " + i + "\n");

                try {
                    Thread.sleep(500);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }

                textArea.append(nombre + " ha terminado de sembrar el árbol " + i + "\n");

                turnoHermano1 = !esHermano1;
                notifyAll();
            }
        }
    }
}

class Hermano extends Thread {
    private final Jardin jardin;
    private final boolean esHermano1;

    public Hermano(Jardin jardin, String nombre, boolean esHermano1) {
        super(nombre);
        this.jardin = jardin;
        this.esHermano1 = esHermano1;
    }

    @Override
    public void run() {
```

```
        jardin.sembrar(getName(), esHermano1);
    }
}

public class HermanosV2RunnableGUI {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Hermanos Sembrando Árboles");
        JTextArea textArea = new JTextArea(20, 40);
        textArea.setEditable(false);
        JScrollPane scrollPane = new JScrollPane(textArea);
        frame.add(scrollPane, BorderLayout.CENTER);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.pack();
        frame.setVisible(true);

        Jardin jardin = new Jardin(textArea);
        Hermano hermano1 = new Hermano(jardin, "Hermano 1", true);
        Hermano hermano2 = new Hermano(jardin, "Hermano 2", false);

        hermano1.start();
        hermano2.start();

        try {
            hermano1.join();
            hermano2.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        textArea.append("Ambos hermanos han terminado de sembrar los árboles\n");
    }
}

```

```
package Practica2;

class Jardin {
  private boolean turnoHermano1 = true;

  public synchronized void sembrar(String nombre, boolean esHermano1) {
    for (int i = 1; i <= 10; i++) {
      synchronized (this) {
        while (turnoHermano1 != esHermano1) {
          try {
            wait();
          } catch (InterruptedException e) {
            e.printStackTrace();
          }
        }

        System.out.println(nombre + " esta sembrando el arbol " + i);

        try {
          Thread.sleep(500);
        } catch (InterruptedException e) {
          e.printStackTrace();
        }

        System.out.println(nombre + " ha terminado de sembrar el arbol " + i);

        turnoHermano1 = !esHermano1;
        notifyAll();
      }
```

```
    }
  }
}

class Hermano implements Runnable {
  private final Jardin jardin;
  private final String nombre;
  private final boolean esHermano1;

  public Hermano(Jardin jardin, String nombre, boolean esHermano1) {
    this.jardin = jardin;
    this.nombre = nombre;
    this.esHermano1 = esHermano1;
  }

  @Override
  public void run() {
    jardin.sembrar(nombre, esHermano1);
  }
}

public class HermanosV2Runnbale {
  public static void main(String[] args) {
    Jardin jardin = new Jardin();
    Hermano hermano1 = new Hermano(jardin, "Hermano 1", true);
    Hermano hermano2 = new Hermano(jardin, "Hermano 2", false);

    Thread t1 = new Thread(hermano1);
    Thread t2 = new Thread(hermano2);

    t1.start();
    t2.start();

    try {
      t1.join();
      t2.join();
    } catch (InterruptedException e) {
      e.printStackTrace();
    }

    System.out.println("Ambos hermanos han terminado de sembrar los árboles");
  }
}

```

```
package Practica2;

class Jardin {
  private boolean turnoHermano1 = true;

  public synchronized void sembrar(String nombre, boolean esHermano1) {
    for (int i = 1; i <= 10; i++) {
      synchronized (this) {
        while (turnoHermano1 != esHermano1) {
            try {
              wait();
            } catch (InterruptedException e) {
              e.printStackTrace();
            }
        }

        System.out.println(nombre + " esta sembrando el arbol " + i);

```

```
        try {
          Thread.sleep(500);
        } catch (InterruptedException e) {
          e.printStackTrace();
        }

        System.out.println(nombre + " ha terminado de sembrar el arbol " + i);

        turnoHermano1 = !esHermano1;
        notifyAll();
      }
    }
  }
}

class Hermano extends Thread {
  private final Jardin jardin;
  private final boolean esHermano1;

  public Hermano(Jardin jardin, String nombre, boolean esHermano1) {
    super(nombre);
    this.jardin = jardin;
    this.esHermano1 = esHermano1;
  }

  @Override
  public void run() {
    jardin.sembrar(getName(), esHermano1);
  }
}

public class HermanosV2Thread {
  public static void main(String[] args) {
    Jardin jardin = new Jardin();
    Hermano hermano1 = new Hermano(jardin, "Hermano 1", true);
    Hermano hermano2 = new Hermano(jardin, "Hermano 2", false);

    hermano1.start();
    hermano2.start();

    try {
      hermano1.join();
      hermano2.join();
    } catch (InterruptedException e) {
      e.printStackTrace();
    }

    System.out.println("Ambos hermanos han terminado de sembrar los arboles");
  }
}

```

```
package Practica2;

import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

class Pelota {
    private int x = 7, xCambio = 7;
    private int y = 0, yCambio = 2;
    private int diametro = 10;
    private int rectIzqX = 0, rectDerX = 100;
```

```
12.     private int rectSupY = 0, rectInfY = 100;
13.
14.     public void mover() {
15.         // Verifica colisiones y cambia la dirección
16.         if (x + xCambio <= rectIzqX || x + xCambio >= rectDerX) {
17.             xCambio = -xCambio;
18.         }
19.         if (y + yCambio <= rectSupY || y + yCambio >= rectInfY) {
20.             yCambio = -yCambio;
21.         }
22.
23.         // Actualiza posición
24.         x += xCambio;
25.         y += yCambio;
26.     }
27.
28.     public void dibujar(Graphics g) {
29.         g.setColor(Color.red);
30.         g.fillOval(x, y, diametro, diametro);
31.     }
32. }
33.
34. public class NoHiloPelota extends JPanel implements ActionListener {
35.     private JButton iniciar;
36.     private Timer timer; // Usaremos un Timer en lugar de un Thread
37.     private Pelota pelota;
38.
39.     public NoHiloPelota() {
40.         iniciar = new JButton("Iniciar");
41.         add(iniciar);
42.         iniciar.addActionListener(this);
43.         pelota = new Pelota();
44.         timer = new Timer(50, e -> {
45.             pelota.mover();
46.             repaint(); // Redibuja la pelota con la nueva posición
47.         });
48.     }
49.
50.     @Override
51.     protected void paintComponent(Graphics g) {
52.         super.paintComponent(g);
53.         g.drawRect(0, 0, 110, 110); // Dibuja el área de rebote
54.         pelota.dibujar(g); // Dibuja la pelota
55.     }
56.
57.     public void actionPerformed(ActionEvent event) {
58.         if (!timer.isRunning()) {
59.             timer.start(); // Iniciar el Timer cuando se presiona el botón
60.         }
61.     }
62.
63.     public static void main(String[] args) {
64.         JFrame frame = new JFrame("Hilo Pelota");
65.         NoHiloPelota panel = new NoHiloPelota();
66.         frame.add(panel);
67.         frame.setSize(200, 200);
68.         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
69.         frame.setVisible(true);
70.     }
71. }
72.
```

```
1. package Practica2;
```

```java

class Contador {
  private int sumaPares = 0;
  private int sumaImpares = 0;

  public synchronized void sumarNumero(int num, boolean esPar) {
    if (esPar) {
      sumaPares += num;
    } else {
      sumaImpares += num;
    }
  }

  public void imprimirSumaTotal() {
    System.out.println("Suma total de pares: " + sumaPares);
    System.out.println("Suma total de impares: " + sumaImpares);
    System.out.println("Suma total de pares e impares: " + (sumaPares + sumaImpares));
  }
}

class HiloNumeros implements Runnable {
  private final Contador contador;
  private final boolean esPar;
  private final String nombreHilo;

  public HiloNumeros(Contador contador, boolean esPar, String nombreHilo) {
      this.contador = contador;
      this.esPar = esPar;
      this.nombreHilo = nombreHilo;
  }

  @Override
  public void run() {
      int inicio = esPar ? 2 : 1;
      for (int i = inicio; i <= 10; i += 2) {
          System.out.println(nombreHilo + " - " + (esPar ? "Par" : "Impar") + ": " + i);
          contador.sumarNumero(i, esPar);
          try {
            Thread.sleep(500); // Simula procesamiento
          } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
          }
      }
  }
}

public class NumerosEstadosRunnable {
  public static void main(String[] args) {
    Contador contador = new Contador();

    // Crear dos hilos con la misma clase, pero con configuraciones diferentes
    Thread hiloA = new Thread(new HiloNumeros(contador, false, "Hilo A")); // Imprime
impares
    Thread hiloB = new Thread(new HiloNumeros(contador, true, "Hilo B"));  // Imprime
pares

    // Iniciar hilos
    hiloA.start();
    hiloB.start();

    // Esperar a que los hilos terminen
    try {
      hiloA.join();
      hiloB.join();
    } catch (InterruptedException e) {
```

```
      System.out.println("Error en la sincronización de los hilos.");
    }

    // Imprimir la suma total
    contador.imprimirSumaTotal();
  }
}

```

```
package Practica2;

class Contador {
    private int sumaPares = 0;
    private int sumaImpares = 0;

    public synchronized void sumarNumero(int num, boolean esPar) {
        if (esPar) {
            sumaPares += num;
        } else {
            sumaImpares += num;
        }
    }

    public void imprimirSumaTotal() {
        System.out.println("Suma total de pares: " + sumaPares);
        System.out.println("Suma total de impares: " + sumaImpares);
        System.out.println("Suma total de pares e impares: " + (sumaPares + sumaImpares));
    }
}

class HiloNumeros extends Thread {
    private final Contador contador;
    private final boolean esPar;

    public HiloNumeros(Contador contador, boolean esPar, String nombreHilo) {
        super(nombreHilo);
        this.contador = contador;
        this.esPar = esPar;
    }

    @Override
    public void run() {
        int inicio = esPar ? 2 : 1;
        for (int i = inicio; i <= 10; i += 2) {
            System.out.println(getName() + " - " + (esPar ? "Par" : "Impar") + ": " + i);
            contador.sumarNumero(i, esPar);
            try {
                Thread.sleep(500); // Simula procesamiento
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }
    }
}

public class NumerosEstadosThread {
    public static void main(String[] args) {
        Contador contador = new Contador();

        // Crear dos hilos con la misma clase, pero con configuraciones diferentes
        HiloNumeros hiloA = new HiloNumeros(contador, false, "Hilo A"); // Imprime impares
        HiloNumeros hiloB = new HiloNumeros(contador, true, "Hilo B");  // Imprime pares

```

```
55.         // Iniciar hilos
56.         hiloA.start();
57.         hiloB.start();
58.
59.         // Esperar a que los hilos terminen
60.         try {
61.             hiloA.join();
62.             hiloB.join();
63.         } catch (InterruptedException e) {
64.             System.out.println("Error en la sincronización de los hilos.");
65.         }
66.
67.         // Imprimir la suma total
68.         contador.imprimirSumaTotal();
69.     }
70. }
71.
```

```
 1. package Practica2;
 2.
 3. import java.util.Random;
 4.
 5. class Contador {
 6.     private int sumaPares = 0;
 7.     private int sumaImpares = 0;
 8.     private final Object lock = new Object();
 9.     private boolean turnoPar = true;
10.
11.     public void sumarNumero(int num, boolean esPar, String nombreHilo) {
12.         synchronized (lock) {
13.             while (turnoPar != esPar) {
14.                 try {
15.                     lock.wait();
16.                 } catch (InterruptedException e) {
17.                     Thread.currentThread().interrupt();
18.                 }
19.             }
20.
21.             System.out.println(nombreHilo + " - " + (esPar ? "Par" : "Impar") + ": " +
num);
22.             if (esPar) {
23.                 sumaPares += num;
24.             } else {
25.                 sumaImpares += num;
26.             }
27.
28.             turnoPar = !turnoPar;
29.             lock.notifyAll();
30.         }
31.     }
32.
33.     public void imprimirSumaTotal() {
34.         System.out.println("Suma total de pares: " + sumaPares);
35.         System.out.println("Suma total de impares: " + sumaImpares);
36.         System.out.println("Suma total de pares e impares: " + (sumaPares + sumaImpares));
37.     }
38. }
39.
40. class HiloNumeros implements Runnable {
41.     private final Contador contador;
42.     private final boolean esPar;
43.     private final String nombreHilo;
44.     private final Random random = new Random();
```

```
45.
46.     public HiloNumeros(Contador contador, boolean esPar, String nombreHilo) {
47.         this.contador = contador;
48.         this.esPar = esPar;
49.         this.nombreHilo = nombreHilo;
50.     }
51.
52.     public boolean esPar(int num) {
53.         return num % 2 == 0;
54.     }
55.
56.     @Override
57.     public void run() {
58.         for (int i = 0; i < 25; i++) {
59.             int num = random.nextInt(100) + 1;
60.             contador.sumarNumero(num, esPar, nombreHilo);
61.             try {
62.                 Thread.sleep(500); // Simula procesamiento
63.             } catch (InterruptedException e) {
64.                 Thread.currentThread().interrupt();
65.             }
66.         }
67.     }
68. }
69.
70. public class NumerosSincronizadosRunnable {
71.     public static void main(String[] args) {
72.         Contador contador = new Contador();
73.
74.         // Crear dos hilos con la misma clase, pero con configuraciones diferentes
75.         Thread hiloA = new Thread(new HiloNumeros(contador, false, "Hilo A")); // Imprime
impares
76.         Thread hiloB = new Thread(new HiloNumeros(contador, true, "Hilo B"));  // Imprime
pares
77.
78.         // Iniciar hilos
79.         hiloA.start();
80.         hiloB.start();
81.
82.         // Esperar a que los hilos terminen
83.         try {
84.             hiloA.join();
85.             hiloB.join();
86.         } catch (InterruptedException e) {
87.             System.out.println("Error en la sincronización de los hilos.");
88.         }
89.
90.         // Imprimir la suma total
91.         contador.imprimirSumaTotal();
92.     }
93. }
94.
```

```
 1. package Practica2;
 2.
 3. import java.util.Random;
 4.
 5. class Contador {
 6.     private int sumaPares = 0;
 7.     private int sumaImpares = 0;
 8.     private final Object lock = new Object();
 9.     private boolean turnoPar = true;
10.
```

```
    public void sumarNumero(int num, boolean esPar, String nombreHilo) {
        synchronized (lock) {
            while (turnoPar != esPar) {
                try {
                    lock.wait();
                } catch (InterruptedException e) {
                    Thread.currentThread().interrupt();
                }
            }

            System.out.println(nombreHilo + " - " + (esPar ? "Par" : "Impar") + ": " +
num);
            if (esPar) {
                sumaPares += num;
            } else {
                sumaImpares += num;
            }

            turnoPar = !turnoPar;
            lock.notifyAll();
        }
    }

    public void imprimirSumaTotal() {
        System.out.println("Suma total de pares: " + sumaPares);
        System.out.println("Suma total de impares: " + sumaImpares);
        System.out.println("Suma total de pares e impares: " + (sumaPares + sumaImpares));
    }
}

class HiloNumeros extends Thread {
    private final Contador contador;
    private final boolean esPar;
    private final Random random = new Random();

    public HiloNumeros(Contador contador, boolean esPar, String nombreHilo) {
        super(nombreHilo);
        this.contador = contador;
        this.esPar = esPar;
    }

    public boolean esPar(int num) {
        return num % 2 == 0;
    }

    @Override
    public void run() {
        for (int i = 0; i < 25; i++) {
            int num = random.nextInt(100) + 1;
            contador.sumarNumero(num, esPar, getName());
            try {
                Thread.sleep(500); // Simula procesamiento
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }
    }
}

public class NumerosSincronizadosThread {
    public static void main(String[] args) {
        Contador contador = new Contador();

        // Crear dos hilos con la misma clase, pero con configuraciones diferentes
        HiloNumeros hiloA = new HiloNumeros(contador, false, "Hilo A"); // Imprime impares
```

```
        HiloNumeros hiloB = new HiloNumeros(contador, true, "Hilo B");  // Imprime pares

        // Iniciar hilos
        hiloA.start();
        hiloB.start();

        // Esperar a que los hilos terminen
        try {
            hiloA.join();
            hiloB.join();
        } catch (InterruptedException e) {
            System.out.println("Error en la sincronización de los hilos.");
        }

        // Imprimir la suma total
        contador.imprimirSumaTotal();
    }
}

```

```
package Practica2;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Random;

class Controlador {
    private boolean turnoPositivos = true; // Comienza con positivos
    private int sumaPositivos = 0;
    private int sumaNegativos = 0;
    private final List<Integer> numeros; // Lista de numeros generados aleatoriamente
    private int index = 0; // Indice para recorrer la lista de numeros

    public Controlador() {
        // Generar numeros aleatorios entre -100 y 100
        numeros = new ArrayList<>();
        Random random = new Random();
        for (int i = 0; i < 201; i++) {
            numeros.add(random.nextInt(201) - 100);
        }
        Collections.shuffle(numeros); // Mezclar los numeros aleatoriamente
    }

    public synchronized void procesarNumero(String nombre, boolean esPositivo) {
        while (index < numeros.size()) {
            while (turnoPositivos != esPositivo) { // Espera su turno
                try {
                    wait(); // Hilo espera hasta que sea su turno
                } catch (InterruptedException e) {
                    Thread.currentThread().interrupt();
                }
            }

            if (index >= numeros.size()) {
                break;
            }

            int num = numeros.get(index);
            index++;

            // Verifica si el numero es del tipo que debe procesar el hilo
            if ((num >= 0 && esPositivo) || (num < 0 && !esPositivo)) {
```

```
                System.out.println(nombre + " identifico: " + num);
                if (esPositivo) {
                    sumaPositivos += num;
                } else {
                    sumaNegativos += num;
                }

                // Cambia el turno y notifica a los otros hilos
                turnoPositivos = !turnoPositivos;
                notifyAll(); // Notifica al otro hilo
            }
        }
        notifyAll(); // Asegura que el otro hilo no se quede esperando
    }

    public void imprimirSumaTotal() {
        System.out.println("Suma de numeros positivos: " + sumaPositivos);
        System.out.println("Suma de numeros negativos: " + sumaNegativos);
    }
}

class HiloNumeros implements Runnable {
    private final Controlador controlador;
    private final boolean esPositivo;
    private final String nombreHilo;

    public HiloNumeros(Controlador controlador, boolean esPositivo, String nombreHilo) {
        this.controlador = controlador;
        this.esPositivo = esPositivo;
        this.nombreHilo = nombreHilo;
    }

    @Override
    public void run() {
        controlador.procesarNumero(nombreHilo, esPositivo);
    }
}

public class PosNegRunnable {
    public static void main(String[] args) {
        Controlador controlador = new Controlador();

        // Crear dos hilos con la misma clase, pero con configuraciones diferentes
        Thread hiloPositivos = new Thread(new HiloNumeros(controlador, true, "Hilo
Positivos"));
        Thread hiloNegativos = new Thread(new HiloNumeros(controlador, false, "Hilo
Negativos"));

        // Iniciar hilos
        hiloPositivos.start();
        hiloNegativos.start();

        // Esperar a que los hilos terminen
        try {
            hiloPositivos.join();
            hiloNegativos.join();
        } catch (InterruptedException e) {
            System.out.println("Error en la sincronizacion de los hilos.");
        }

        // Imprimir la suma total
        controlador.imprimirSumaTotal();
    }
}

```

```java
package Practica2;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Random;

class Controlador {
    private boolean turnoPositivos = true; // Comienza con positivos
    private int sumaPositivos = 0;
    private int sumaNegativos = 0;
    private final List<Integer> numeros; // Lista de numeros generados aleatoriamente
    private int index = 0; // Indice para recorrer la lista de numeros

    public Controlador() {
        // Generar numeros aleatorios entre -100 y 100
        numeros = new ArrayList<>();
        Random random = new Random();
        for (int i = 0; i < 201; i++) {
            numeros.add(random.nextInt(201) - 100);
        }
        Collections.shuffle(numeros); // Mezclar los numeros aleatoriamente
    }

    public synchronized void procesarNumero(String nombre, boolean esPositivo) {
        while (index < numeros.size()) {
            while (turnoPositivos != esPositivo) { // Espera su turno
                try {
                    wait(); // Hilo espera hasta que sea su turno
                } catch (InterruptedException e) {
                    Thread.currentThread().interrupt();
                }
            }

            if (index >= numeros.size()) {
                break;
            }

            int num = numeros.get(index);
            index++;

            // Verifica si el numero es del tipo que debe procesar el hilo
            if ((num >= 0 && esPositivo) || (num < 0 && !esPositivo)) {
                System.out.println(nombre + " identifico: " + num);
                if (esPositivo) {
                    sumaPositivos += num;
                } else {
                    sumaNegativos += num;
                }

                // Cambia el turno y notifica a los otros hilos
                turnoPositivos = !turnoPositivos;
                notifyAll(); // Notifica al otro hilo
            }
        }
        notifyAll(); // Asegura que el otro hilo no se quede esperando
    }

    public void imprimirSumaTotal() {
        System.out.println("Suma de numeros positivos: " + sumaPositivos);
        System.out.println("Suma de numeros negativos: " + sumaNegativos);
    }
}

class HiloNumeros implements Runnable {
```

```
66.     private final Controlador controlador;
67.     private final boolean esPositivo;
68.     private final String nombreHilo;
69. 
70.     public HiloNumeros(Controlador controlador, boolean esPositivo, String nombreHilo) {
71.         this.controlador = controlador;
72.         this.esPositivo = esPositivo;
73.         this.nombreHilo = nombreHilo;
74.     }
75. 
76.     @Override
77.     public void run() {
78.         controlador.procesarNumero(nombreHilo, esPositivo);
79.     }
80. }
81. 
82. public class PosNegThread {
83.     public static void main(String[] args) {
84.         Controlador controlador = new Controlador();
85. 
86.         // Crear dos hilos con la misma clase, pero con configuraciones diferentes
87.         Thread hiloPositivos = new Thread(new HiloNumeros(controlador, true, "Hilo
Positivos"));
88.         Thread hiloNegativos = new Thread(new HiloNumeros(controlador, false, "Hilo
Negativos"));
89. 
90.         // Iniciar hilos
91.         hiloPositivos.start();
92.         hiloNegativos.start();
93. 
94.         // Esperar a que los hilos terminen
95.         try {
96.             hiloPositivos.join();
97.             hiloNegativos.join();
98.         } catch (InterruptedException e) {
99.             System.out.println("Error en la sincronizacion de los hilos.");
100.        }
101. 
102.        // Imprimir la suma total
103.        controlador.imprimirSumaTotal();
104.    }
105. }
106. 
```

```
1. package Practica2;
2. 
3. import java.awt.*;
4. import javax.swing.*;
5. import java.awt.event.*;
6. 
7. class Pelota implements Runnable { // Implementar Runnable para usar hilos
8.     private Graphics g;
9.     private int x = 7, xCambio = 7;
10.    private int y = 0, yCambio = 2;
11.    private int diametro = 10;
12.    private int rectIzqX = 0, rectDerX = 100;
13.    private int rectSupY = 0, rectInfY = 100;
14.    private boolean running = true; // Control para detener el hilo
15. 
16.    public Pelota(Graphics graficos) {
17.        g = graficos;
18.    }
19. 
```

```
20.      public void detener() {
21.          running = false; // Método para detener la pelota si es necesario
22.      }
23.
24.      @Override
25.      public void run() {
26.          g.drawRect(rectIzqX, rectSupY, rectDerX - rectIzqX + 10, rectInfY - rectSupY +
10);
27.
28.          for (int n = 1; n < 1000 && running; n++) { // Se ejecuta mientras "running" sea
true
29.              g.setColor(Color.white);
30.              g.fillOval(x, y, diametro, diametro);
31.
32.              if (x + xCambio <= rectIzqX || x + xCambio >= rectDerX) {
33.                  xCambio = -xCambio;
34.              }
35.              if (y + yCambio <= rectSupY || y + yCambio >= rectInfY) {
36.                  yCambio = -yCambio;
37.              }
38.
39.              x += xCambio;
40.              y += yCambio;
41.              g.setColor(Color.red);
42.              g.fillOval(x, y, diametro, diametro);
43.
44.              try {
45.                  Thread.sleep(50); // Pequeña pausa para ver el movimiento
46.              } catch (InterruptedException e) {
47.                  System.err.println("Excepcion de inactividad");
48.                  Thread.currentThread().interrupt();
49.              }
50.          }
51.      }
52. }
53.
54. public class hiloPelota extends JPanel implements ActionListener {
55.      private JButton iniciar;
56.      private Thread hiloPelota; // Hilo para manejar la animación de la pelota
57.
58.      public hiloPelota() {
59.          iniciar = new JButton("Iniciar");
60.          add(iniciar);
61.          iniciar.addActionListener(this);
62.      }
63.
64.      public void actionPerformed(ActionEvent event) {
65.          if (event.getSource() == iniciar) {
66.              Graphics g = getGraphics();
67.              Pelota pelota = new Pelota(g);
68.              hiloPelota = new Thread(pelota); // Crear hilo con la pelota
69.              hiloPelota.start(); // Iniciar hilo
70.          }
71.      }
72.
73.      public static void main(String[] args) {
74.          JFrame frame = new JFrame("Hilo Pelota");
75.          hiloPelota panel = new hiloPelota();
76.          frame.add(panel);
77.          frame.setSize(200, 200);
78.          frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
79.          frame.setVisible(true);
80.      }
81. }
```