

# PROGRAMACIÓN DE SERVICIOS Y PROCESOS

## CASO PRÁCTICO 1 - JOSE CRUCES APARICIO 2 ° DAM

### SECCIÓN CRÍTICA

Aquí muestro el código del main.java de un proyecto que gestiona una sección crítica en la que a la vez se le manda 3 precios y tiene que gestionar ese momento e incluir el mejor precio de los 3, esta muy comentado para explicar todo lo mas claro posible :

#### Main.java :

```
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.locks.ReentrantLock;

// Clase que representa un proveedor y su precio.
// Contiene los getters y setters para acceder o modificar sus atributos.
class PrecioProveedor {
    private String nombreProveedor;
    private double precio;

    public PrecioProveedor(String nombreProveedor, double precio) {
        this.nombreProveedor = nombreProveedor;
        this.precio = precio;
    }

    public String getNombreProveedor() {
        return nombreProveedor;
    }

    public void setNombreProveedor(String nombreProveedor) {
        this.nombreProveedor = nombreProveedor;
    }

    public double getPrecio() {
        return precio;
    }

    public void setPrecio(double precio) {
        this.precio = precio;
    }

    @Override
    public String toString() {
        return nombreProveedor + ": " + precio + " €";
    }
}

// Clase Producto: gestiona el conjunto de precios de un producto,
// controlando el acceso concurrente mediante un mecanismo de exclusión mutua (ReentrantLock).
class Producto {

    // Lista que almacena los precios de los distintos proveedores.
    private List<PrecioProveedor> precios = new ArrayList<>();

    // Objeto Lock que permite controlar la zona crítica entre varios hilos.
    private final ReentrantLock lock = new ReentrantLock();

    // Getter con control de concurrencia.
    // Se bloquea el acceso mientras se lee para evitar inconsistencias si otro hilo está escribiendo.
    public List<PrecioProveedor> getPrecios() {
        lock.lock();
        try {
            // Se devuelve una copia del ArrayList original para evitar modificaciones externas.
            return new ArrayList<>(precios);
        } finally {
            lock.unlock(); // Se libera el bloqueo aunque ocurra una excepción.
        }
    }

    // Método que añade un nuevo precio a la lista.
    // Representa la zona crítica, ya que varios hilos pueden intentar escribir simultáneamente.
    public void agregarPrecio(PrecioProveedor precioProveedor) {
        lock.lock(); // Inicio de la sección crítica.
    }
}
```

```

    try {
        System.out.println(Thread.currentThread().getName() + " agregando " + precioProveedor);
        precios.add(precioProveedor);

        // Simulamos el tiempo de escritura para mostrar el efecto del bloqueo.
        Thread.sleep(500);
    } catch (InterruptedException e) {
        Thread.currentThread().interrupt();
    } finally {
        lock.unlock(); // Fin de la sección crítica (libera el lock).
    }
}

// Calcula y devuelve el mejor precio (el menor) entre todos los proveedores.
// También se protege con el bloqueo para asegurar una lectura consistente.
public double obtenerMejorPrecio() {
    lock.lock();
    try {
        return precios.stream()
            .mapToDouble(PrecioProveedor::getPrecio)
            .min()
            .orElse(Double.NaN);
    } finally {
        lock.unlock();
    }
}

// Clase principal: simula la ejecución concurrente de varios proveedores que actualizan precios.
public class Main {
    public static void main(String[] args) {
        Producto producto = new Producto();

        // Creamos varios hilos que intentarán escribir en la lista de precios simultáneamente.
        Thread proveedor1 = new Thread(() -> {
            producto.agregarPrecio(new PrecioProveedor("Proveedor A", 10.5));
        }, "Hilo-Proveedor-A");

        Thread proveedor2 = new Thread(() -> {
            producto.agregarPrecio(new PrecioProveedor("Proveedor B", 9.8));
        }, "Hilo-Proveedor-B");

        Thread proveedor3 = new Thread(() -> {
            producto.agregarPrecio(new PrecioProveedor("Proveedor C", 11.2));
        }, "Hilo-Proveedor-C");

        // Iniciamos los tres hilos.
        proveedor1.start();
        proveedor2.start();
        proveedor3.start();

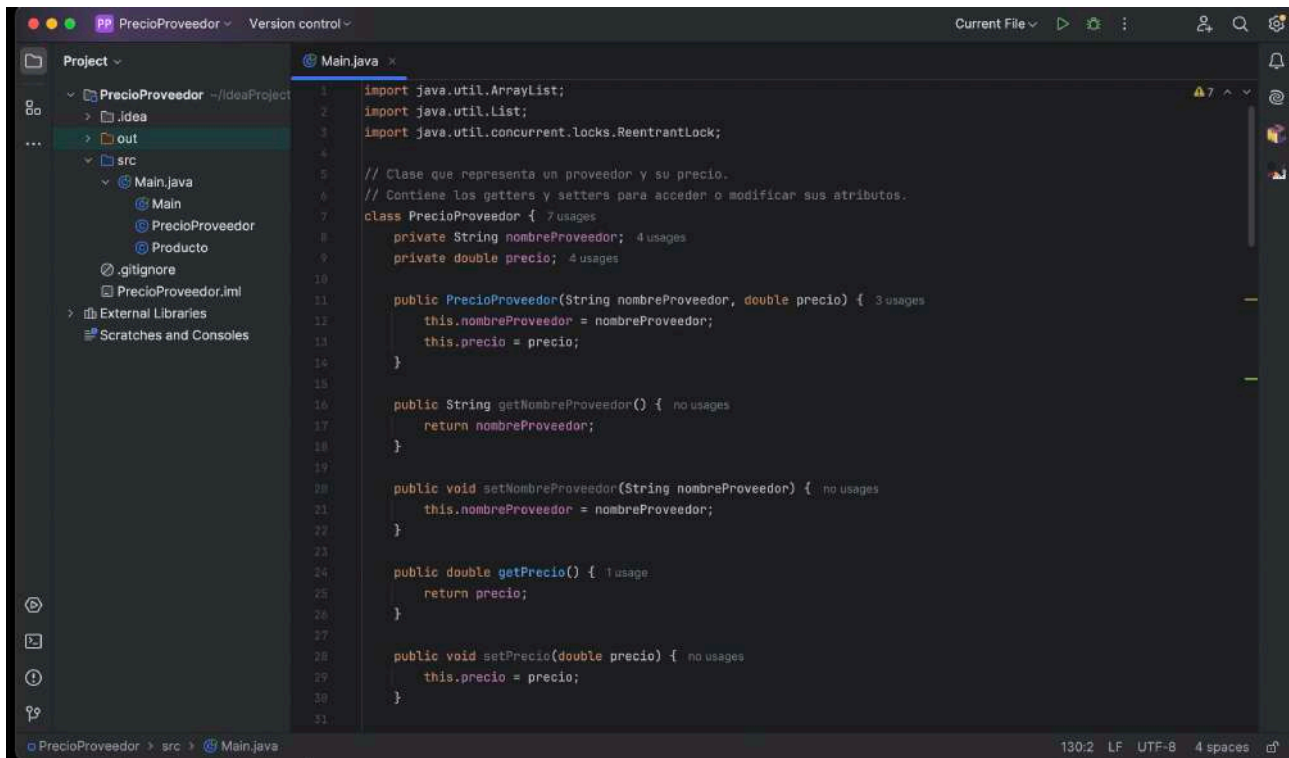
        // Con join() esperamos a que todos los hilos terminen antes de continuar.
        try {
            proveedor1.join();
            proveedor2.join();
            proveedor3.join();
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }

        // Mostramos los resultados finales.
        System.out.println("\nLista de precios final:");
        producto.getPrecios().forEach(System.out::println);

        System.out.println("\n Mejor precio: " + producto.obtenerMejorPrecio() + " €");
    }
}

```

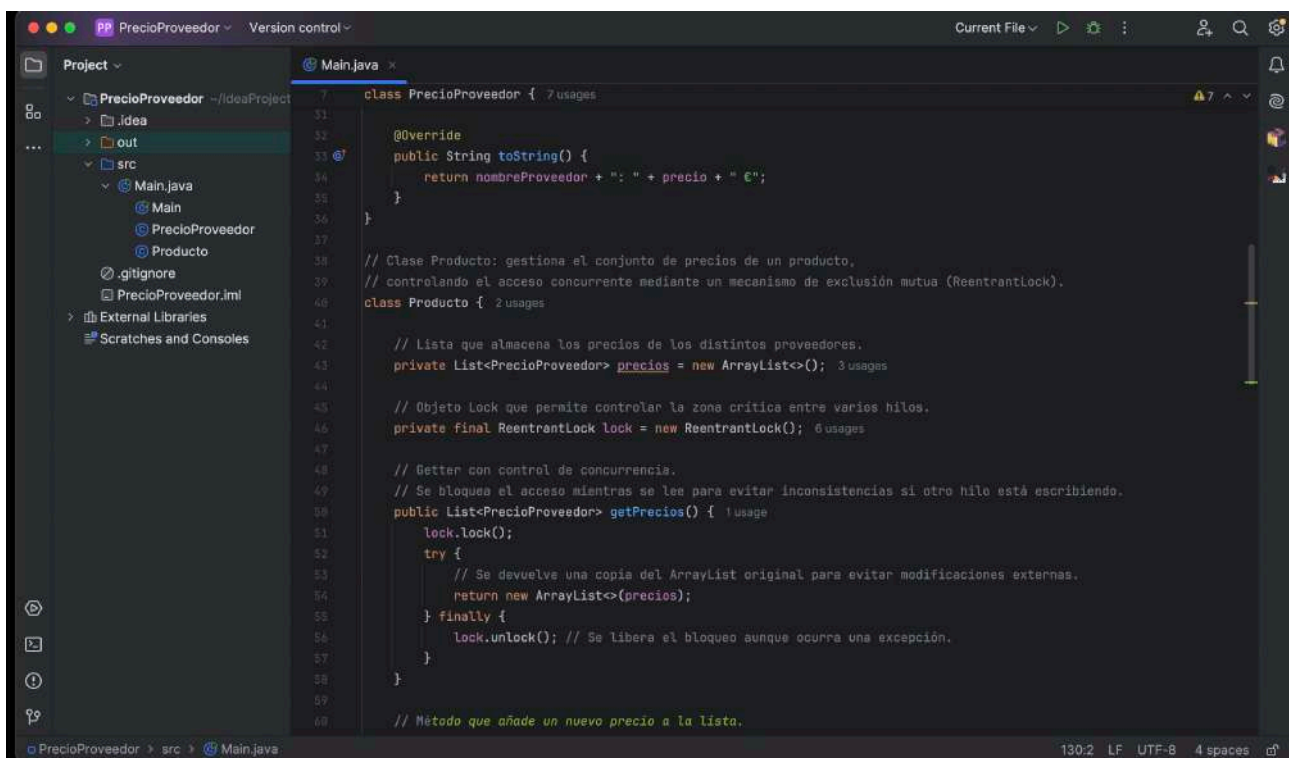
Adjunto capturas de pantalla para mostrar el código y después una captura de pantalla que como se tienen que mandar comprimidos los pdf no se ve del todo bien por eso está el código entero encima y entrego el archivo java también, pero además muestro el resultado del ejercicio viendo la gestión.



```

1  import java.util.ArrayList;
2  import java.util.List;
3  import java.util.concurrent.locks.ReentrantLock;
4
5  // Clase que representa un proveedor y su precio.
6  // Contiene los getters y setters para acceder o modificar sus atributos.
7  class PrecioProveedor { 7 usages
8      private String nombreProveedor; 4 usages
9      private double precio; 4 usages
10
11     public PrecioProveedor(String nombreProveedor, double precio) { 3 usages
12         this.nombreProveedor = nombreProveedor;
13         this.precio = precio;
14     }
15
16     public String getNombreProveedor() { no usages
17         return nombreProveedor;
18     }
19
20     public void setNombreProveedor(String nombreProveedor) { no usages
21         this.nombreProveedor = nombreProveedor;
22     }
23
24     public double getPrecio() { 1 usage
25         return precio;
26     }
27
28     public void setPrecio(double precio) { no usages
29         this.precio = precio;
30     }
31

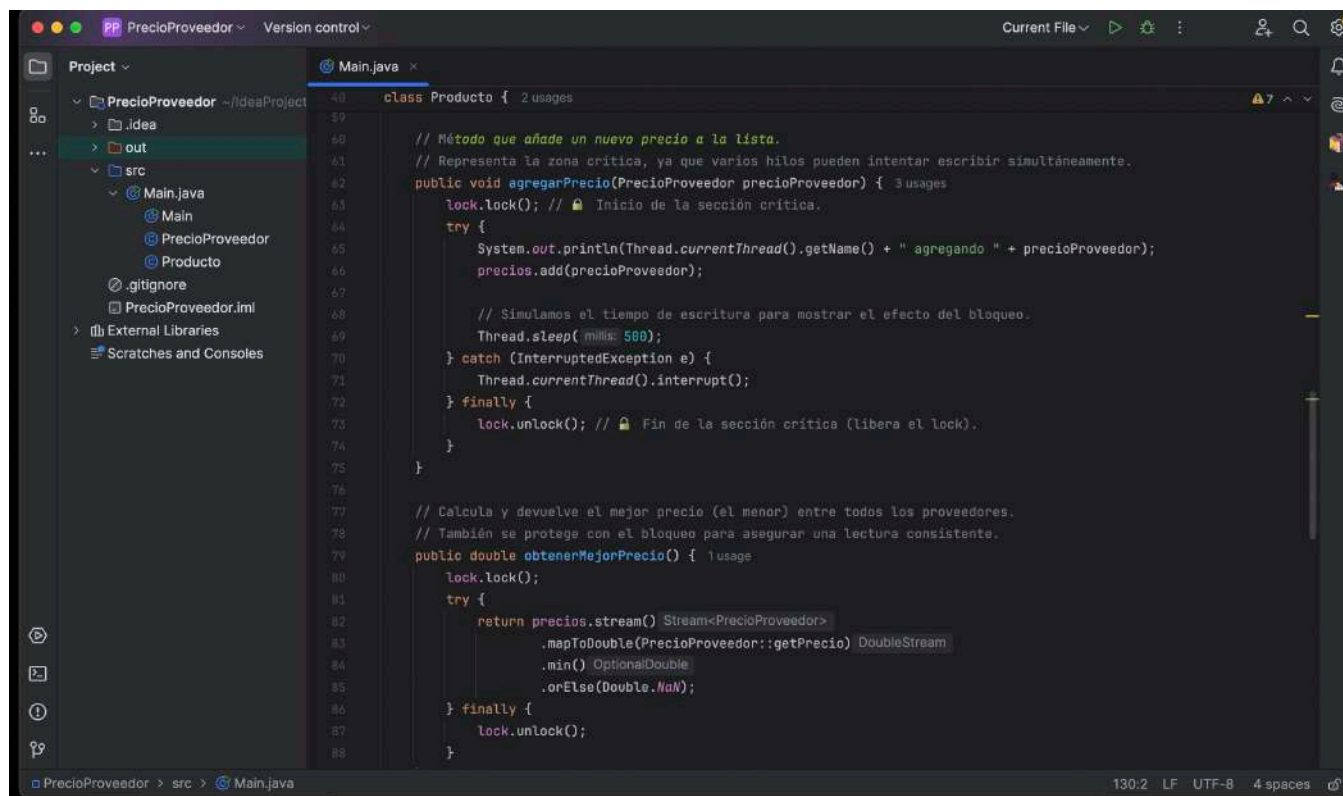
```



```

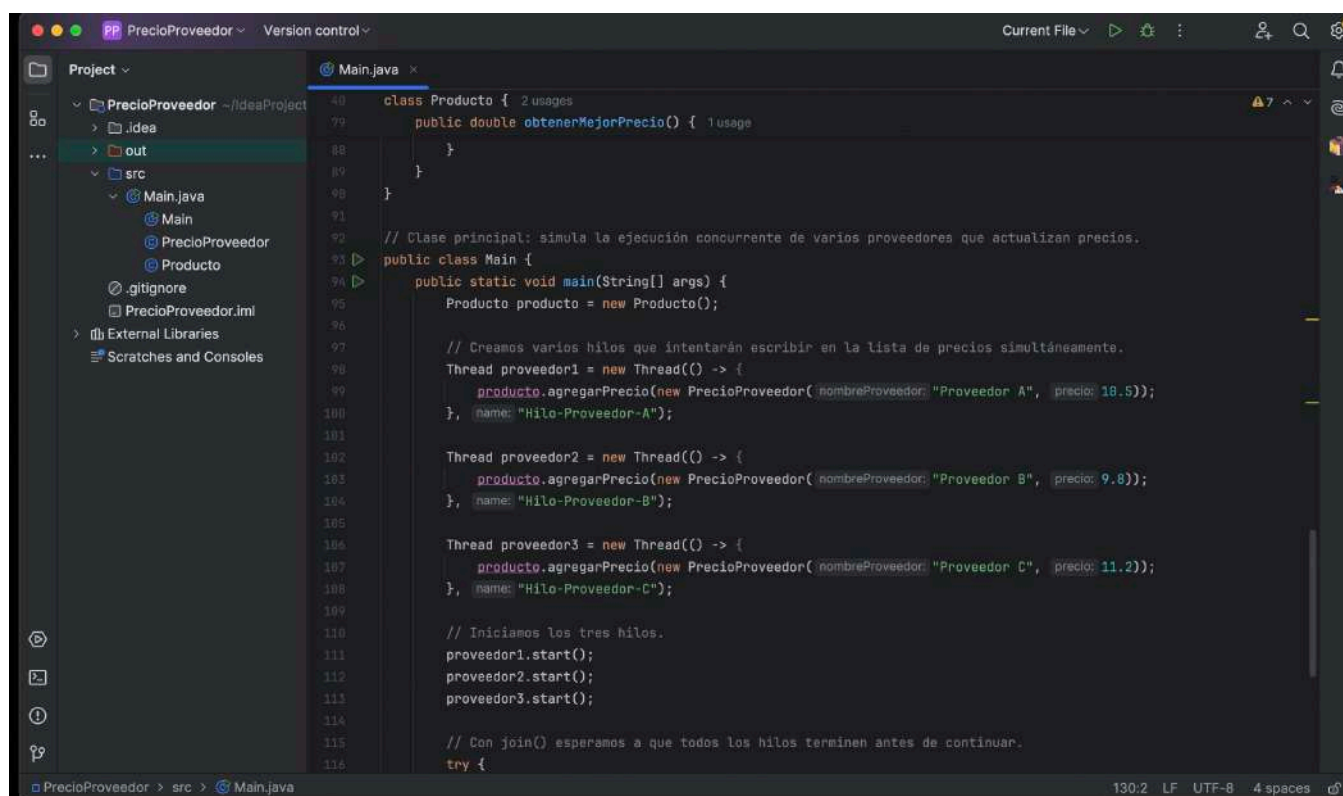
7  class PrecioProveedor { 7 usages
31
32     @Override
33     public String toString() {
34         return nombreProveedor + ": " + precio + " €";
35     }
36 }
37
38 // Clase Producto: gestiona el conjunto de precios de un producto,
39 // controlando el acceso concurrente mediante un mecanismo de exclusión mutua (ReentrantLock).
40 class Producto { 2 usages
41
42     // Lista que almacena los precios de los distintos proveedores.
43     private List<PrecioProveedor> precios = new ArrayList<>(); 3 usages
44
45     // Objeto Lock que permite controlar la zona crítica entre varios hilos.
46     private final ReentrantLock lock = new ReentrantLock(); 6 usages
47
48     // Getter con control de concurrencias.
49     // Se bloquea el acceso mientras se lee para evitar inconsistencias si otro hilo está escribiendo.
50     public List<PrecioProveedor> getPrecios() { 1 usage
51         lock.lock();
52         try {
53             // Se devuelve una copia del ArrayList original para evitar modificaciones externas.
54             return new ArrayList<>(precios);
55         } finally {
56             lock.unlock(); // Se libera el bloqueo aunque ocurra una excepción.
57         }
58     }
59
60     // Método que añade un nuevo precio a la lista.

```



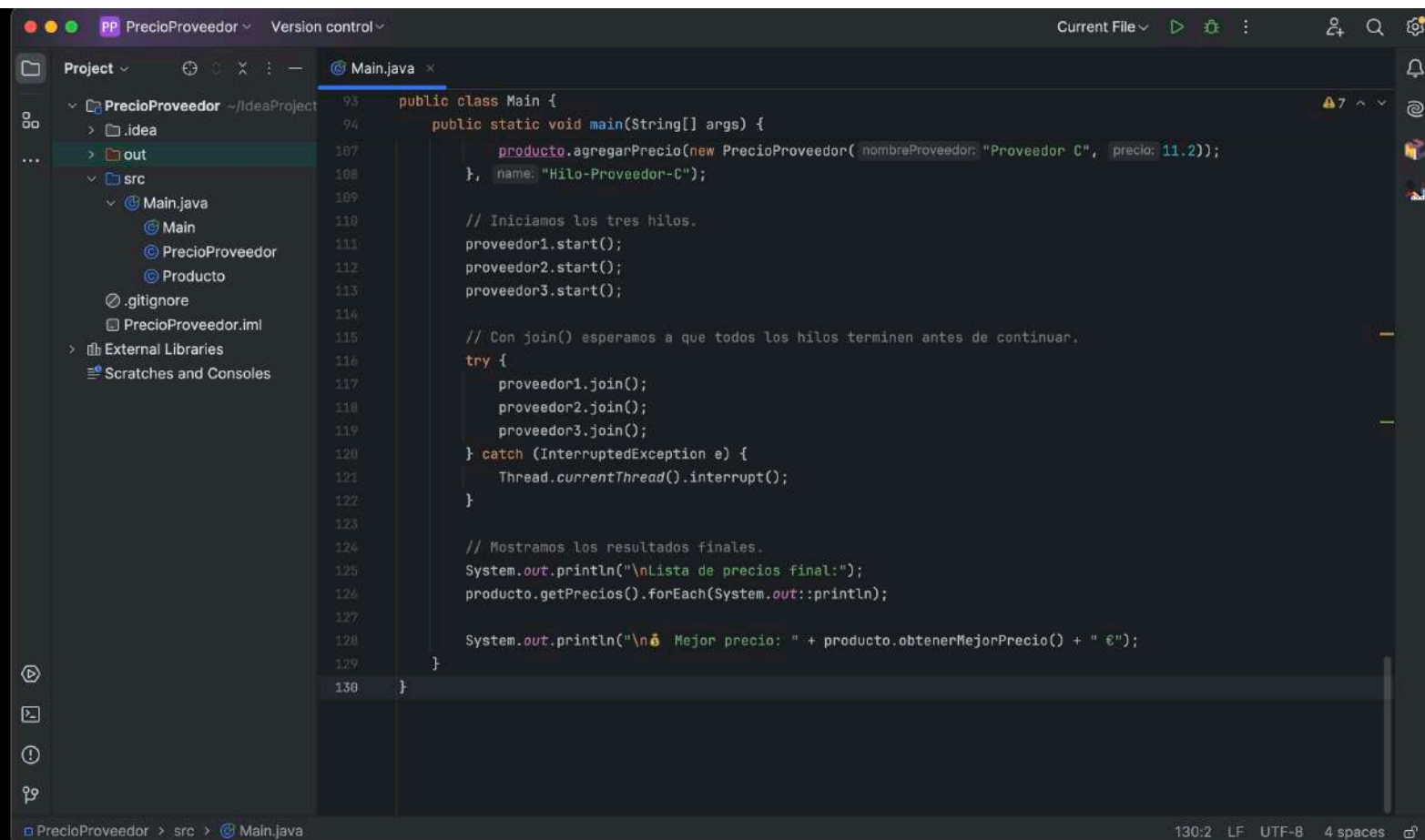
```

40 class Producto { 2 usages
41
42 // Método que añade un nuevo precio a la lista.
43 // Representa la zona crítica, ya que varios hilos pueden intentar escribir simultáneamente.
44 public void agregarPrecio(PrecioProveedor precioProveedor) { 3 usages
45     lock.lock(); // Inicio de la sección crítica.
46     try {
47         System.out.println(Thread.currentThread().getName() + " agregando " + precioProveedor);
48         precios.add(precioProveedor);
49
50         // Simulamos el tiempo de escritura para mostrar el efecto del bloqueo.
51         Thread.sleep(millis: 500);
52     } catch (InterruptedException e) {
53         Thread.currentThread().interrupt();
54     } finally {
55         lock.unlock(); // Fin de la sección crítica (libera el lock).
56     }
57 }
58
59 // Calcula y devuelve el mejor precio (el menor) entre todos los proveedores.
60 // También se protege con el bloqueo para asegurar una lectura consistente.
61 public double obtenerMejorPrecio() { 1 usage
62     lock.lock();
63     try {
64         return precios.stream().Stream<PrecioProveedor>
65             .mapToDouble(PrecioProveedor::getPrecio) DoubleStream
66             .min() OptionalDouble
67             .orElse(Double.NaN);
68     } finally {
69         lock.unlock();
70     }
71 }
72 }
  
```



```

79 class Producto { 2 usages
80     public double obtenerMejorPrecio() { 1 usage
81     }
82 }
83
84 // Clase principal: simula la ejecución concurrente de varios proveedores que actualizan precios.
85 public class Main {
86     public static void main(String[] args) {
87         Producto producto = new Producto();
88
89         // Creamos varios hilos que intentarán escribir en la lista de precios simultáneamente.
90         Thread proveedor1 = new Thread(() -> {
91             producto.agregarPrecio(new PrecioProveedor( nombreProveedor: "Proveedor A", precio: 10.5));
92         }, name: "Hilo-Proveedor-A");
93
94         Thread proveedor2 = new Thread(() -> {
95             producto.agregarPrecio(new PrecioProveedor( nombreProveedor: "Proveedor B", precio: 9.8));
96         }, name: "Hilo-Proveedor-B");
97
98         Thread proveedor3 = new Thread(() -> {
99             producto.agregarPrecio(new PrecioProveedor( nombreProveedor: "Proveedor C", precio: 11.2));
100         }, name: "Hilo-Proveedor-C");
101
102         // Iniciamos los tres hilos.
103         proveedor1.start();
104         proveedor2.start();
105         proveedor3.start();
106
107         // Con join() esperamos a que todos los hilos terminen antes de continuar.
108         try {
109
110
111
112
113
114
115
116
  
```

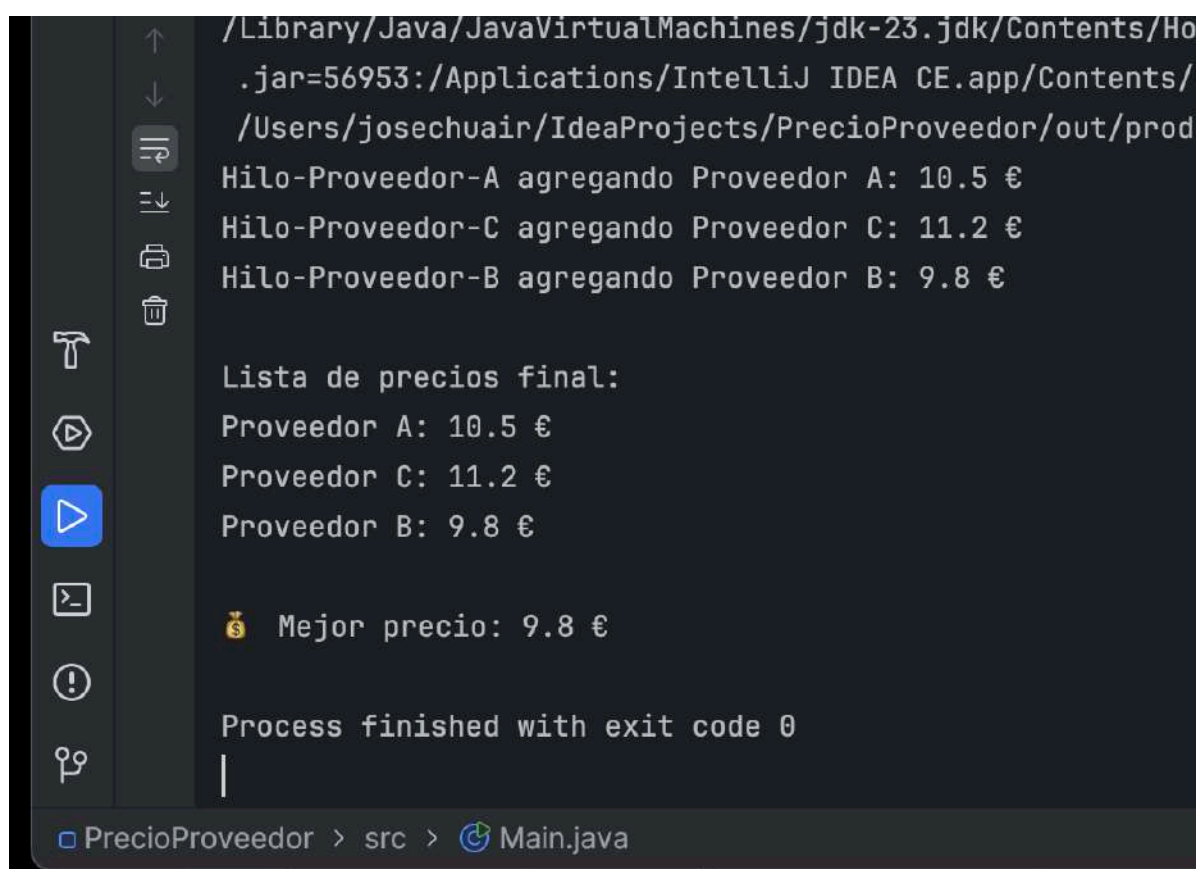


```

93 public class Main {
94     public static void main(String[] args) {
107         producto.agregarPrecio(new PrecioProveedor( nombreProveedor: "Proveedor C", precio: 11.2));
108         }, name: "Hilo-Proveedor-C");
109
110         // Iniciamos los tres hilos.
111         proveedor1.start();
112         proveedor2.start();
113         proveedor3.start();
114
115         // Con join() esperamos a que todos los hilos terminen antes de continuar.
116         try {
117             proveedor1.join();
118             proveedor2.join();
119             proveedor3.join();
120         } catch (InterruptedException e) {
121             Thread.currentThread().interrupt();
122         }
123
124         // Mostramos los resultados finales.
125         System.out.println("\nLista de precios final:");
126         producto.getPrecios().forEach(System.out::println);
127
128         System.out.println("\n💰 Mejor precio: " + producto.obtenerMejorPrecio() + " €");
129     }
130 }

```

Aqui muestro la ejecución del programa y el resultado en la terminal :



```

/Library/Java/JavaVirtualMachines/jdk-23.jdk/Contents/Home
.jar=56953:/Applications/IntelliJ IDEA CE.app/Contents/
/Users/josechuair/IdeaProjects/PrecioProveedor/out/prod
Hilo-Proveedor-A agregando Proveedor A: 10.5 €
Hilo-Proveedor-C agregando Proveedor C: 11.2 €
Hilo-Proveedor-B agregando Proveedor B: 9.8 €

Lista de precios final:
Proveedor A: 10.5 €
Proveedor C: 11.2 €
Proveedor B: 9.8 €

💰 Mejor precio: 9.8 €

Process finished with exit code 0

```



## Notas

La implementación de exclusión mutua mediante `ReentrantLock` garantiza la integridad de los datos en entornos multihilo.

Gracias a este mecanismo, se evita que varios procesos escriban simultáneamente en el mismo recurso, eliminando las condiciones de carrera.

Este tipo de control es esencial en sistemas concurrentes, especialmente en contextos empresariales donde se manejan precios, inventarios o información crítica compartida.

## Documentacion :

Material de la Unidad 1

<https://webprogramacion.com/exclusion-mutua/>

Youtube

[https://es.wikipedia.org/wiki/Exclusi%C3%B3n\\_mutua\\_\(inform%C3%A1tica\)](https://es.wikipedia.org/wiki/Exclusi%C3%B3n_mutua_(inform%C3%A1tica))

<https://docs.oracle.com/en/java/javase/25/docs/api/java.base/java/util/concurrent/locks/ReentrantLock.html>

<https://es.scribd.com/document/832852201/Exclusion-Mutua>