UD2 - A4: Generar Sucesivas Circunferencias con Clases Productora y Consumidora (Con y Sin Sincronización)

---

1. Objetivo de Aprendizaje:

El objetivo de esta actividad es que los alumnos comprendan cómo compartir recursos entre hilos en una aplicación multihilo, primero **sin sincronización**, para identificar los problemas de concurrencia, y luego **con sincronización**, para resolver esos problemas y garantizar un acceso seguro a los recursos compartidos. Los estudiantes aprenderán a detectar **zonas críticas** en el código y aplicar mecanismos de sincronización para evitar condiciones de carrera y conflictos derivados de la compartición de recursos.

---

2. Enunciado:

Crea un programa en Java que simule la interacción entre una **clase productora** y una **clase consumidora**. Ambas clases compartirán un recurso, que es una colección de radios de circunferencia. La clase productora generará radios y los almacenará en la colección, mientras que la clase consumidora utilizará esos radios para generar objetos `Circunferencia`.

Instrucciones:

1. Crea una clase **Productora** que genere radios de circunferencia y los añada a una colección compartida.

```java
package actividad;

import java.util.List;
import java.util.Random;

public class Productora implements Runnable {
    private List<Double> radios;

    public Productora(List<Double> radios) {
        this.radios = radios;
    }

    @Override
    public void run() {
        Random random = new Random();
        for (int i = 0; i < 10; i++) {
            double radio = 1 + (10 - 1) * random.nextDouble();
            System.out.println("Productora ha generado el radio: " + radio);
            radios.add(radio);
            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

2. Crea una clase **Consumidora** que tome los radios de la colección compartida y genere objetos `Circunferencia`.

```java
package actividad;

import java.util.List;

public class Consumidora implements Runnable {
    private List<Double> radios;

    public Consumidora(List<Double> radios) {
        this.radios = radios;
    }

    @Override
    public void run() {
        while (true) {
            if (!radios.isEmpty()) {
                double radio = radios.remove(0);
                Circunferencia circunferencia = new Circunferencia(radio);
                System.out.println("Consumidora ha creado una circunferencia con radio: " + radio + " y área: " + circunferencia.getArea());
            } else {
                try {
                    Thread.sleep(500);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

Circunferencia

```java
package actividad;

public class Circunferencia {
    private double radio;

    public Circunferencia(double radio) {
        this.radio = radio;
    }

    public double getArea() {
        return Math.PI * radio * radio;
    }
}
```

3. **Primero**, implementa el programa **sin sincronización** y observa los posibles problemas de concurrencia, como la clase consumidora intentando acceder a un radio que aún no ha sido generado por la clase productora.

Main

```java
package actividad;

import java.util.ArrayList;
import java.util.List;

public class Main {
    public static void main(String[] args) {
        List<Double> radios = new ArrayList<>();

        Productora productora = new Productora(radios);
        Consumidora consumidora = new Consumidora(radios);

        Thread hiloProductora = new Thread(productora);
        Thread hiloConsumidora = new Thread(consumidora);

        hiloProductora.start();
        hiloConsumidora.start();
    }
}
```

Resultado

```
Productora ha generado el radio: 1.4165024193710702
Consumidora ha creado una circunferencia con radio: 1.4165024193710702 y área: 6.303540012972023
Productora ha generado el radio: 2.1908297697134826
Productora ha generado el radio: 1.43377218882529
Consumidora ha creado una circunferencia con radio: 2.1908297697134826 y área: 15.07881246607429
Consumidora ha creado una circunferencia con radio: 1.43377218882529 y área: 6.458180467137327
Productora ha generado el radio: 8.124674929164103
Productora ha generado el radio: 7.444721376132083
Consumidora ha creado una circunferencia con radio: 8.124674929164103 y área: 207.37760770167736
Consumidora ha creado una circunferencia con radio: 7.444721376132083 y área: 174.1192428319254
Productora ha generado el radio: 2.223972760199371
Productora ha generado el radio: 6.128613969118643
Consumidora ha creado una circunferencia con radio: 2.223972760199371 y área: 15.538489543654885
Productora ha generado el radio: 9.654259046977522
Consumidora ha creado una circunferencia con radio: 6.128613969118643 y área: 117.99793475716692
Consumidora ha creado una circunferencia con radio: 9.654259046977522 y área: 292.81125655120667
Productora ha generado el radio: 4.575807250231643
Consumidora ha creado una circunferencia con radio: 4.575807250231643 y área: 65.77870465255657
Productora ha generado el radio: 2.4153945026054995
Consumidora ha creado una circunferencia con radio: 2.4153945026054995 y área: 18.3284618431495
```

4. **Después**, implementa el programa **con sincronización**, utilizando mecanismos como `synchronized` o semáforos para evitar los problemas de concurrencia detectados.

Productora

```java
package actividad;

import java.util.List;
import java.util.Random;

public class Productora implements Runnable {
    private List<Double> radios;

    public Productora(List<Double> radios) {
        this.radios = radios;
    }

    @Override
    public void run() {
        Random random = new Random();
        for (int i = 0; i < 10; i++) {
            double radio = 1 + (10 - 1) * random.nextDouble();
            synchronized (radios) {
                System.out.println("Productora ha generado el radio: " + radio);
                radios.add(radio);
                radios.notify();
            }
            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Consumidora

```java
package actividad;

import java.util.List;

public class Consumidora implements Runnable {
    private List<Double> radios;

    public Consumidora(List<Double> radios) {
        this.radios = radios;
    }

    @Override
    public void run() {
        while (true) {
            synchronized (radios) {
                while (radios.isEmpty()) {
                    try {
                        radios.wait();
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
                double radio = radios.remove(0);
                Circunferencia circunferencia = new Circunferencia(radio);
                System.out.println("Consumidora ha creado una circunferencia con radio: " + radio + " y área: " + circunferencia.getArea());
            }
        }
    }
}
```

Resultado

```
Productora ha generado el radio: 9.796616263454649
Consumidora ha creado una circunferencia con radio: 9.796616263454649 y área: 301.5102401122702
Productora ha generado el radio: 1.634941188125553
Consumidora ha creado una circunferencia con radio: 1.634941188125553 y área: 8.397579857403478
Productora ha generado el radio: 8.439246221715278
Consumidora ha creado una circunferencia con radio: 8.439246221715278 y área: 223.74698330799873
Productora ha generado el radio: 2.131877418896652
Consumidora ha creado una circunferencia con radio: 2.131877418896652 y área: 14.278228627109764
Productora ha generado el radio: 3.9230485121254173
Consumidora ha creado una circunferencia con radio: 3.9230485121254173 y área: 48.350083665334715
Productora ha generado el radio: 3.580796413056362
Consumidora ha creado una circunferencia con radio: 3.580796413056362 y área: 40.28182443681276
Productora ha generado el radio: 6.781760885988169
Consumidora ha creado una circunferencia con radio: 6.781760885988169 y área: 144.48901121520086
Productora ha generado el radio: 1.5765212817060072
Consumidora ha creado una circunferencia con radio: 1.5765212817060072 y área: 7.808175176302511
Productora ha generado el radio: 2.696352288549549
Consumidora ha creado una circunferencia con radio: 2.696352288549549 y área: 22.840370279195607
Productora ha generado el radio: 4.978476391334386
Consumidora ha creado una circunferencia con radio: 4.978476391334386 y área: 77.86508762333249
```

5. Identifica y explica las zonas críticas en el código que provocan problemas de compartición de recursos.

Al ejecutar el programa sin sincronización, es posible que la clase Consumidora intente acceder a un radio que aún no ha sido generado por la clase Productora, lo que puede causar errores o comportamientos inesperados.