Alessandra Gonzalez

**Ejercicio 1**

Armar la lógica para que se ejecuten los procesos en la secuencia ABABAB.   
  
public class Main {

public static void main(String[] args) {

Semaphore semA = new Semaphore(1); // A empieza

Semaphore semB = new Semaphore(0); // B espera

new Thread(() -> imprimir('A', semA, semB)).start();

new Thread(() -> imprimir('B', semB, semA)).start();

}

private static void imprimir(char letra, Semaphore actual, Semaphore siguiente) {

for (int i = 0; i < 3; i++) {

try {

actual.acquire();

System.out.print(letra);

siguiente.release();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

}

—--------------------------------------------------------------------------------------

// Archivo: ImprimirA.java

import java.util.concurrent.Semaphore;

public class ImprimirA implements Runnable {

private final Semaphore semA;

private final Semaphore semB;

public ImprimirA(Semaphore semA, Semaphore semB) {

this.semA = semA;

this.semB = semB;

}

@Override

public void run() {

try {

// Imprimir 'A' tres veces

for (int i = 0; i < 3; i++) {

semA.acquire(); // Espera a que sea su turno

System.out.print("A");

semB.release(); // Permite que el hilo B continúe

}

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

}

}

}

—-------------------------------------------------------------------------------------------

// Archivo: ImprimirB.java

import java.util.concurrent.Semaphore;

public class ImprimirB implements Runnable {

private final Semaphore semA;

private final Semaphore semB;

public ImprimirB(Semaphore semA, Semaphore semB) {

this.semA = semA;

this.semB = semB;

}

@Override

public void run() {

try {

// Imprimir 'B' cinco veces

for (int i = 0; i < 5; i++) {

semB.acquire(); // Espera a que sea su turno

System.out.print("B");

semA.release(); // Permite que el hilo A continúe

}

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

}

}

}

**Ejercicio 2**

Armar la lógica para que se ejecuten los procesos en la secuencia AABB.

public class Main {

public static void main(String[] args) {

// Crear semáforos para controlar la secuencia

Semaphore semA = new Semaphore(2); // Permite que A imprima dos veces

Semaphore semB = new Semaphore(0); // B espera al inicio

// Crear e iniciar los hilos

Thread hiloA = new Thread(new ImprimirA(semA, semB));

Thread hiloB = new Thread(new ImprimirB(semA, semB));

hiloA.start();

hiloB.start();

}

}

class ImprimirA implements Runnable {

private final Semaphore semA;

private final Semaphore semB;

public ImprimirA(Semaphore semA, Semaphore semB) {

this.semA = semA;

this.semB = semB;

}

@Override

public void run() {

for (int i = 0; i < 2; i++) { // Imprimir A dos veces

try {

semA.acquire(); // Esperar a que se permita imprimir A

System.out.print("A");

semB.release(); // Liberar permiso para que B imprima

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

}

class ImprimirB implements Runnable {

private final Semaphore semA;

private final Semaphore semB;

public ImprimirB(Semaphore semA, Semaphore semB) {

this.semA = semA;

this.semB = semB;

}

@Override

public void run() {

for (int i = 0; i < 2; i++) { // Imprimir B dos veces

try {

semB.acquire(); // Esperar a que A imprima

System.out.print("B");

semA.release(); // Liberar permiso para que A imprima

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

}

**Ejercicio 3**

Armar la lógica para que se ejecuten los procesos en la secuencia ABBA.

Armar la lógica para que se ejecuten los procesos en la secuencia ABBA.

import java.util.concurrent.Semaphore; // Importar la clase Semaphore

public class Main {

public static void main(String[] args) {

// Crear semáforos para controlar la secuencia

Semaphore semA = new Semaphore(1); // A puede imprimir primero

Semaphore semB = new Semaphore(2); // B espera al inicio

// Crear e iniciar los hilos

Thread hiloA = new Thread(new ImprimirA(semA, semB));

Thread hiloB = new Thread(new ImprimirB(semA, semB));

hiloA.start();

hiloB.start();

}

}

import java.util.concurrent.Semaphore; // Importar la clase Semaphore

class ImprimirA implements Runnable {

private final Semaphore semA;

private final Semaphore semB;

public ImprimirA(Semaphore semA, Semaphore semB) {

this.semA = semA;

this.semB = semB;

}

@Override

public void run() {

try {

// Imprimir A

semA.acquire(); // Adquirir permiso para imprimir

System.out.print("A");

semB.release(); // Liberar a B

// Imprimir A nuevamente

semA.acquire(); // Esperar a que B imprima

System.out.print("A");

semB.release(); // Liberar a B

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

import java.util.concurrent.Semaphore; // Importar la clase Semaphore

class ImprimirB implements Runnable {

private final Semaphore semA;

private final Semaphore semB;

public ImprimirB(Semaphore semA, Semaphore semB) {

this.semA = semA;

this.semB = semB;

}

@Override

public void run() {

try {

// Esperar a que A imprima

semB.acquire();

System.out.print("B");

//semA.release(); // Permitir que A imprima

// Esperar a que A imprima nuevamente

semB.acquire();

System.out.print("B");

semA.release(); // Permitir que A imprima

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

**Ejercicio 4**

Armar la lógica para que se ejecuten los procesos en la secuencia ABABBA.  
  
public class Main {

public static void main(String[] args) {

// Crear semáforos para controlar la secuencia

Semaphore semA = new Semaphore(1); // A puede imprimir primero

Semaphore semB = new Semaphore(0); // B espera al inicio

Thread hiloA = new Thread(new ImprimirA(semA, semB));

Thread hiloB = new Thread(new ImprimirB(semA, semB));

hiloA.start();

hiloB.start();

}

}

import java.util.concurrent.Semaphore; // Importar la clase Semaphore

class ImprimirA implements Runnable {

private final Semaphore semA;

private final Semaphore semB;

public ImprimirA(Semaphore semA, Semaphore semB) {

this.semA = semA;

this.semB = semB;

}

@Override

public void run() {

try {

semA.acquire();

System.out.print("A");

semB.release();

semA.acquire();

System.out.print("A");

semB.release();

semA.acquire();

System.out.print("A");

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

import java.util.concurrent.Semaphore;

public class ImprimirB implements Runnable {

private final Semaphore semA;

private final Semaphore semB;

public ImprimirB(Semaphore semA, Semaphore semB) {

this.semA = semA;

this.semB = semB;

}

@Override

public void run() {

try {

semB.acquire();

System.out.print("B");

semA.release();

semB.acquire();

System.out.print("B");

System.out.print("B");

semA.release();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

**Ejercicio 5**

Armar la lógica para que se ejecuten los procesos en la secuencia AABBCC.

import java.util.concurrent.Semaphore;

public class Main {

public static void main(String[] args) {

// Crear semáforos para controlar la secuencia

Semaphore semA = new Semaphore(2);

Semaphore semB = new Semaphore(0);

Semaphore semC = new Semaphore(0);

Thread hiloA = new Thread(new ImprimirA(semA, semB));

Thread hiloB = new Thread(new ImprimirB(semB, semC));

Thread hiloC = new Thread(new ImprimirC(semC));

hiloA.start();

hiloB.start();

hiloC.start();

}

}

import java.util.concurrent.Semaphore;

class ImprimirA implements Runnable {

private final Semaphore semA;

private final Semaphore semB;

public ImprimirA(Semaphore semA, Semaphore semB) {

this.semA = semA;

this.semB = semB;

}

@Override

public void run() {

try {

semA.acquire();

System.out.print("A");

semB.release();

semA.acquire();

System.out.print("A");

semB.release();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

import java.util.concurrent.Semaphore;

class ImprimirB implements Runnable {

private final Semaphore semB;

private final Semaphore semC;

public ImprimirB(Semaphore semB, Semaphore semC) {

this.semB = semB;

this.semC = semC;

}

@Override

public void run() {

try {

semB.acquire();

System.out.print("B");

semC.release();

semB.acquire();

System.out.print("B");

semC.release();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

import java.util.concurrent.Semaphore;

class ImprimirC implements Runnable {

private final Semaphore semC;

public ImprimirC(Semaphore semC) {

this.semC = semC;

}

@Override

public void run() {

try {

semC.acquire();

System.out.print("C");

semC.acquire();

System.out.print("C");

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

**A partir de acá, deben pasar los ejercicios a código utilizando semáforos.**

**Ejercicio 6**

Armar la lógica para que se ejecuten los procesos en la secuencia ABABCC.  
  
import java.util.concurrent.Semaphore;

public class Main {

public static void main(String[] args) {

Semaphore semA = new Semaphore(1);

Semaphore semB = new Semaphore(0);

Semaphore semC = new Semaphore(0);

Thread hiloA = new Thread(new ImprimirA(semA, semB));

Thread hiloB = new Thread(new ImprimirB(semB, semC, semA));

Thread hiloC = new Thread(new ImprimirC(semC, semA));

hiloA.start();

hiloB.start();

hiloC.start();

}

}

import java.util.concurrent.Semaphore;

class ImprimirA implements Runnable {

private final Semaphore semA;

private final Semaphore semB;

public ImprimirA(Semaphore semA, Semaphore semB) {

this.semA = semA;

this.semB = semB;

}

@Override

public void run() {

try {

for (int i = 0; i < 2; i++) {

semA.acquire();

System.out.print("A");

semB.release();

}

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

import java.util.concurrent.Semaphore;

class ImprimirB implements Runnable {

private final Semaphore semB;

private final Semaphore semC;

private final Semaphore semA;

public ImprimirB(Semaphore semB, Semaphore semC, Semaphore semA) {

this.semB = semB;

this.semC = semC;

this.semA = semA;

}

@Override

public void run() {

try {

for (int i = 0; i < 2; i++) {

semC.release();

semB.acquire();

System.out.print("B");

if (i < 1) {

semA.release();

}

}

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

import java.util.concurrent.Semaphore;

class ImprimirC implements Runnable {

private final Semaphore semC;

private final Semaphore semA;

public ImprimirC(Semaphore semC, Semaphore semA) {

this.semC = semC;

this.semA = semA;

}

@Override

public void run() {

try {

for (int i = 0; i < 2; i++) {

semC.acquire();

System.out.print("C");

if (i < 1) {

semA.release(); }

}

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

**Ejercicio 7**

Armar la lógica para que se ejecuten los procesos en la secuencia ABCCBA.

import java.util.concurrent.Semaphore; // Importar la clase Semaphore

public class Main {

public static void main(String[] args) {

Semaphore semA = new Semaphore(1);

Semaphore semB = new Semaphore(0);

Semaphore semC = new Semaphore(0);

Semaphore semReverseA = new Semaphore(0);

Semaphore semReverseB = new Semaphore(0);

Thread hiloA = new Thread(new ImprimirA(semA, semB, semReverseA));

Thread hiloB = new Thread(new ImprimirB(semB, semC, semReverseB, semReverseA));

Thread hiloC = new Thread(new ImprimirC(semC, semReverseB));

hiloA.start();

hiloB.start();

hiloC.start();

}

}

import java.util.concurrent.Semaphore;

class ImprimirA implements Runnable {

private final Semaphore semA;

private final Semaphore semB;

private final Semaphore semReverseA;

public ImprimirA(Semaphore semA, Semaphore semB, Semaphore semReverseA) {

this.semA = semA;

this.semB = semB;

this.semReverseA = semReverseA;

}

@Override

public void run() {

try {

semA.acquire();

System.out.print("A");

semB.release();

// Esperar a la secuencia inversa

semReverseA.acquire(); // <-- Espera hasta que el hilo C termine

System.out.print("A"); // <-- Imprime A al final

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

import java.util.concurrent.Semaphore;

class ImprimirB implements Runnable {

private final Semaphore semB;

private final Semaphore semC;

private final Semaphore semReverseB;

private final Semaphore semReverseA;

public ImprimirB(Semaphore semB, Semaphore semC, Semaphore semReverseB, Semaphore semReverseA) {

this.semB = semB;

this.semC = semC;

this.semReverseB = semReverseB;

this.semReverseA = semReverseA;

}

@Override

public void run() {

try {

// Imprimir B después de A

semB.acquire();

System.out.print("B");

semC.release(); // <-- Permitir que C imprima

// Esperar a la secuencia inversa

semReverseB.acquire(); // <-- Espera hasta que el hilo C termine

System.out.print("B"); // <-- Imprime B antes de A en la secuencia inversa

semReverseA.release(); // <-- Permite que A imprima después de B

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

import java.util.concurrent.Semaphore;

class ImprimirC implements Runnable {

private final Semaphore semC;

private final Semaphore semReverseB;

public ImprimirC(Semaphore semC, Semaphore semReverseB) {

this.semC = semC;

this.semReverseB = semReverseB;

}

@Override

public void run() {

try {

semC.acquire();

System.out.print("C");

System.out.print("C");

// Iniciar la secuencia inversa

semReverseB.release(); // <-- Liberar el semáforo para que B imprima de nuevo

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}