

Sprawozdanie: Kelner

Uczestnicy:

1. Kacper Feliks
2. Filip Malejki
3. Michał Szymocha

1. Opis tematu zadania

Zadanie polegało na zaimplementowaniu monitora *Kelner*, który steruje dostępem do dwuosobowego stolika. W systemie występuje N par (kobieta i mężczyzna o tym samym numerze j).

Wymagania:

- Algorytm dla każdej osoby (kobiety i mężczyzny) wygląda następująco:

```
repeat
    własne_sprawy;
    KELNER.CHCE_STOLIK(j);
    randka;
    KELNER.ZWALNIAM;
forever;
```
- Stolik jest przydzielany parze j (kobiecie j i mężczyźnie j) tylko wtedy, gdy oboje zgłoszą chęć zajęcia stolika.
- Zwalnianie stolika nie musi być jednoczesne – osoby mogą odejść od stołu w różnym czasie, ale stolik staje się wolny dla innej pary dopiero, gdy obie osoby go opuszczają.

2. Kod implementacji

Poniżej znajduje się kod klasy *Kelner* zaimplementowany w języku Java przy użyciu mechanizmów *ReentrantLock* oraz *Condition*.

```
package lab6;

import java.util.concurrent.locks.Condition;
import java.util.concurrent.locks.ReentrantLock;
import java.util.concurrent.ThreadLocalRandom;

public class Kelner {
    private final ReentrantLock lock = new ReentrantLock();
    private final Condition[] czekajNaPare;
    private final Condition czekajNaStol;
    private final boolean[] someoneWaitingForPair;
```

```

private int przyStole = 0;
private final int N;

public Kelner(int N) {
    this.N = N;
    czekajNaPare = new Condition[N];
    someoneWaitingForPair = new boolean[N];
    for (int i = 0; i < N; i++) {
        czekajNaPare[i] = lock.newCondition();
        someoneWaitingForPair[i] = false;
    }
    czekajNaStol = lock.newCondition();
}

public void chceStolik(int j) throws InterruptedException {
    lock.lock();
    try {
        int id = j;
        if (!someoneWaitingForPair[id]) {
            someoneWaitingForPair[id] = true;
            // Czekamy na partnera.
            // Dopiero jak ustawi flagę na false, to zaczynamy randkę
            while (someoneWaitingForPair[id]) {
                czekajNaPare[id].await();
            }
        } else {
            // Jestem drugą osobą z pary.
            // Muszę zdobyć stolik (czekam aż przyStole == 0).
            while (przyStole > 0) {
                czekajNaStol.await();
            }
            // Zajmujemy stolik
            przyStole = 2;

            // Informujemy partnera, że stolik jest gotowy
            someoneWaitingForPair[id] = false;
            czekajNaPare[id].signal();
        }
    } finally {
        lock.unlock();
    }
}

public void zwalniam() {
    lock.lock();
    try {

```

```

        przyStole--;
        if (przyStole == 0) {
            czekajNaStol.signal();
        }
    } finally {
        lock.unlock();
    }
}

public static void main(String[] args) {
    final int N = 3;
    final Kelner kelner = new Kelner(N);

    for (int id = 0; id < N; id++) {
        final int pairId = id;
        new Thread(() -> osoba("Mężczyzna", pairId, kelner)).start();
        new Thread(() -> osoba("Kobieta", pairId, kelner)).start();
    }
}

private static void osoba(String role, int j, Kelner kelner) {
    while (true) {
        try {
            Thread.sleep(ThreadLocalRandom.current().nextInt(200, 800));
            System.out.printf("%s %d: chcę stolik\n", role, j);
            kelner.chceStolik(j);
            System.out.printf("%s %d: SIEDZIMY (randka)\n", role, j);

            Thread.sleep(ThreadLocalRandom.current().nextInt(300, 1000));

            System.out.printf("%s %d: odchodzę\n", role, j);
            kelner.zwalniam();

            Thread.sleep(ThreadLocalRandom.current().nextInt(100, 400));
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
            return;
        }
    }
}
}

```

3. Opis rozwiązania

Rozwiązanie wykorzystuje jeden lock (ReentrantLock) do ochrony sekcji krytycznych oraz zestaw zmiennych warunkowych (Condition):

1. Synchronizacja pary:

- Tablica `someoneWaitingForPair` przechowuje informację, czy dla danej pary `j` jedna osoba już czeka.
- Pierwsza osoba, która wywoła `chceStolik(j)`, ustawia `someoneWaitingForPair[j] = true` i zasypia na zmiennej warunkowej `czekajNaPare[j]`.
- Druga osoba widzi, że `someoneWaitingForPair[j]` jest `true`, co oznacza, że para jest w komplecie.

2. Dostęp do stolika:

- Druga osoba z pary (ta, która przysła później) jest odpowiedzialna za rezerwację stolika.
- Sprawdza zmienną `przyStole`. Jeśli `przyStole > 0`, oznacza to, że inna para zajmuje stół (lub jedna osoba z innej pary jeszcze nie odeszła). Wtedy wątek czeka na zmiennej `czekajNaStol`.
- Gdy `przyStole == 0`, druga osoba ustawia `przyStole = 2`, budzi swojego partnera (`czekajNaPare[j].signal()`) i oboje wchodzi do sekcji krytycznej (randka).

3. Zwalnianie stolika:

- Każda osoba wywołująca `zwalniam()` zmniejsza licznik `przyStole`.
- Dopiero gdy `przyStole` spadnie do 0 (ostatnia osoba z pary odeszła), wysyłany jest sygnał `czekajNaStol.signal()`, który budzi ewentualną kolejną parę czekającą na stół.

Dzięki temu mechanizmowi zapewnione jest, że: 1) Para siada tylko razem, 2) Stół jest zajmowany przez jedną parę na raz, 3) Stół jest zwalniany dopiero po odejściu obu osób.