Problem pięciu filozofów

# Historia:

W 1965 roku Edsger Dijkstra wymyślił zadanie egzaminacyjne polegające na tym że 5 komputerów próbuje uzyskać dostęp do pięciu współdzielonych dysków. Zadanie zostało na nowo sformułowane przez Charlesa Hoare'a jako problem ucztujących filozofów.

# Opis problemu:

Pięciu filozofów siedzi przy okrągłym stole, mają do dyspozycji pięć widelców z których każdy leży pomiędzy dwoma filozofami. Aby zacząć jeść filozof musi mieć dwa widelce, ale może sięgnąć tylko po te leżące obok niego, filozofowie nie mogą także komunikować się między sobą. Może tu wystąpić sytuacja zakleszczenia, na przykład w chwili gdy każdy z filozofów sięgnie po swój lewy widelec i będzie próbował zabrać prawy, który jednak będzie zabrany przez filozofa po prawej stronie. Jeżeli zostanie wzięta pod uwagę czas oczekiwania na dwa wolne widelce może też wystąpić sytuacja głodzenia.

# Możliwe rozwiązania:

#### Hierarchia zasobów

Widelce są ponumerowane od 1 do 5. Filozof najpierw sięga po widelec o niższym numerze, a potem o niższym. Z tego wynika że

* Po za ostatnim filozofem, wszyscy sięgają najpierw po lewy, a później prawy widelec.
* Ostatni filozof sięga najpierw po prawy, a później po lewy widelec.

Odkładać widelce należy w kolejności odwrotnej do kolejności brania.

#### Kelner (5 filozofów 4 miejsca)

Polega na ograniczeniu miejsc przy stole do 4. Filozof przed próbą uzyskania dostępu do widelców prosi kelnera o pozwolenie na zajęcie miejsca przy stole, kiedy uzyska może podjąć próbę uzyskania widelców (nie ma znaczenia czy filozofowie najpierw sięgają po lewy a potem po prawy, czy odwrotnie), po jedzeniu filozof odkłada widelce i wstaje od stołu zwalniając miejsce dla oczekującego filozofa.

#### Rozwiązanie Chandy/Misra

Dla każdej pary filozofów tworzymy widelec i przekazujemy go do filozofa z niższym numerem. Widelec może być czysty lub brudny, na początku wszystkie są brudne. Kiedy filozof chce zacząć jeść wysyła żądanie przekazania widelca. Filozof który otrzymuje żądanie, przekazuje widelec jeżeli ten jest brudny myjąc go przy tej okazji lub też zatrzymuje widelec dla siebie jeżeli ten jest czysty. Po jedzeniu oba widelce stają się brudne.

# Implementacje:

#### Deadlock

Zawiera kod z możliwością wystąpienia zakleszczenia.

Trzy klasy:

1. **Philosopher** dziedziczy po **Thread** - klasa reprezentująca wątek filozofa

**id** -stała liczba całkowita reprezentujaca numer filozofa

**leftFork**, **rightFork** - semafory binarne reprezentujące widelce

**eatCounter** - liczba całkowita reprezentująca licznik wywołań metody eat(), czyli w skrócie ile razy jadł, na początku przyjmuje wartość 0.

**Philosopher(int id, Semaphore leftFork, Semaphore rightFork) -**  konstruktor

**run()** - główna funkcja wątku. Zawiera pętle wykonującą się dopóki **eatCounter** nie osiągnie wartości 100. Najpierw podnosi widelec lewy, a potem prawy odkłada w tej samej kolejności.

**think()** - filozof myśli (ciało funkcji zakomentowane w celu zwiększenia przejrzystości wyświetlanego tekstu)

**eat()** - filozof je (zakomentowane wywołanie funkcji **TimeUnit. ... .sleep()** symulującej czas potrzebny na jedzenie, zrobione w celu przyspieszenia wykonania programu.

**toString()** zwraca **String** - tekstowa reprezentacja obiektu

2. **Table -** reprezentuje stół

**forks** - **ArrayList<Semaphore>** zawiera semafory binarne (widelce)

**philosophers - ArrayList<Philosopher>** zawiera filozofów

**startDinner(int n) -** rozstawia **n** widelców i **n** filozofów na stole i startuje wątki, gdy n<2 przerywa wykonywanie funkcji

3. **Deadlock -** główna klasa zawierająca metodę **main()**

#### ResourceHierarchy

Implementacja rozwiązania **"Hierarchia zasobów"**

Cztery klasy:

1. **Philosopher** dziedziczy po **Thread** - klasa reprezentująca wątek filozofa

**id** - stała liczba całkowita reprezentujaca numer filozofa

**left**, **right** - typu **Fork** reprezentujące widelce

**eatCounter** - liczba całkowita reprezentująca licznik wywołań metody eat(), czyli w skrócie ile razy jadł, na początku przyjmuje wartość 0.

**Philosopher(int id, Forkleft, Fork right) -**  konstruktor

**run()** - główna funkcja wątku. Zawiera pętle wykonującą się dopóki **eatCounter** nie osiągnie wartości 100. Najpierw podnosi widelec o niższym numerze, a potem o wyższym, odkłada w odwrotnej kolejności.

**think()** - filozof myśli (ciało funkcji zakomentowane w celu zwiększenia przejrzystości wyświetlanego tekstu)

**eat()** - filozof je (zakomentowane wywołanie funkcji **TimeUnit. ... .sleep()** symulującej czas potrzebny na jedzenie, zrobione w celu przyspieszenia wykonania programu.

**toString()** zwraca **String** - tekstowa reprezentacja obiektu

2. **Fork** dziedziczy po **Semaphore** - klasa reprezentująca widelec

**id** - stała liczba całkowita reprezentujaca numer widelca

**Fork (int id) -** konstruktor, ustawia semafor na binarny

**getId()** zwraca **int**  - zwraca numer widelca

3. **Table -** reprezentuje stół

**forks** - **ArrayList<Fork>** zawiera semafory binarne (widelce)

**philosophers - ArrayList<Philosopher>** zawiera filozofów

**startDinner(int n) -** rozstawia **n** widelców i **n** filozofów na stole i startuje wątki, gdy n<2 przerywa wykonywanie funkcji

4. **ResourceHierarchy -** główna klasa zawierająca metodę **main()**

#### Waiter

Implementacja rozwiązania **"Kelner"**

Trzy klasy:

1. **Philosopher** dziedziczy po **Thread** - klasa reprezentująca wątek filozofa

**id** -stała liczba całkowita reprezentujaca numer filozofa

**waiter** - statyczny semafor o liczbie pozwoleń o 1 mniejszej od liczby filozofów, reprezentuje kelnera.

**left**, **right** - semafory binarne reprezentujące widelce

**eatCounter** - liczba całkowita reprezentująca licznik wywołań metody eat(), czyli w skrócie ile razy jadł, na początku przyjmuje wartość 0.

**Philosopher(int id, Semaphore left, Semaphore right) -**  konstruktor

**setWaiter(Semaphore waiter)** - statyczna funkcja ustawiająca semafor waiter

**run()** - główna funkcja wątku. Zawiera pętle wykonującą się dopóki **eatCounter** nie osiągnie wartości 100. Najpierw prosi kelnera o dostęp do stołu, gdy go dostanie najpierw podnosi lewy widelec, a potem prawy, odkłada je w odwrotnej kolejności następnie odchodzi od stołu robiąc miejsce dla oczekującego filozofa.

**think()** - filozof myśli (ciało funkcji zakomentowane w celu zwiększenia przejrzystości wyświetlanego tekstu)

**eat()** - filozof je (zakomentowane wywołanie funkcji **TimeUnit. ... .sleep()** symulującej czas potrzebny na jedzenie, zrobione w celu przyspieszenia wykonania programu.

**toString()** zwraca **String** - tekstowa reprezentacja obiektu

2. **Table -** reprezentuje stół

**forks** - **ArrayList<Semaphore>** zawiera semafory binarne (widelce)

**philosophers - ArrayList<Philosopher>** zawiera filozofów

**startDinner(int n) -** rozstawia **n** widelców i **n** filozofów na stole i startuje wątki, gdy n<2 przerywa wykonywanie funkcji

3. **Main -** główna klasa zawierająca metodę **main()**