|  |
| --- |
| Programowanie współbiezne |
| Laboratium 3: Spotkania |
| Ada Rendezvous in Java |
|  |
| **Tomasz Trębski – 165535**  **Karol Górecki - 165405** |
| **11/18/2013** |

|  |
| --- |
| **Sprawozdanie na laboratoria z przedmiotu Programowanie współbieżne** |

Table of Contents

[Wprowadzenia 2](#_Toc372546899)

[Teoria 2](#_Toc372546900)

[Rendez-vous 2](#_Toc372546901)

[A co na to Java? 2](#_Toc372546902)

[Implementacja 2](#_Toc372546903)

[Wnioski 3](#_Toc372546904)

# Wprowadzenia

Celem zadania było zaimplementowanie współbieżnego programu implementującego mechanizm współbieżności znany z języka Ada – **spotkania.**

# Teoria

## Rendez-vous

**Rendez-vous** w Ada to sposób na uzyskanie współbieżności między n-procesami oraz ich synchronizacji. Głównym celem, jak i założeniem, spotkań był mechanizm jak najbardziej zbliżony do naturalnych sposobów wymiany danych między obiektami. Aby doszło do ich komunikacji i wymiany danych należy spełnić następujące warunki:

* Zadania wykonują się współbieżnie,
* Komunikacja możliwa jest jedynie podczas spotkania zadań,
* Istnieje pojęcie klienta oraz serwera,
* Klienci muszą znać nazwę/nazwy usług, które chcą wywołać na serwerze
* Serwer buforuje żądania klientów i wykonuje je, kiedy ma dostępne ku temu zasoby.

## A co na to Java?

**Java** nie posiada mechanizmu synchronizacji opartego o spotkanie w sensie stricte znanym z języka Ada. Do komunikacji między wątkami używa się, więc mechanizmów, które są zaimplementowane OOTB w samej Javie i dostępne bez konieczności ściągania kolejnych bibliotek. Tak naprawdę dowolny obiekt w Javie jest gotowy do bycia elementem przetwarzania wielowątkowego, z uwagi na metody **wait(),notify(),notifyAll()**. Które odpowiednio wywołują na monitorze obiektu na rzecz, którego wywołano funkcję -> czekaj, powiadom, powiadom Wszystkich. Niemniej nie możliwe jest, aby zaimplementować spotkania, jako asynchroniczne zdarzenie polegające na wywoływaniu funkcji klienta oraz serwera.

# Implementacja

W implementacji w Java postaraliśmy się zasymulować mechanizm spotkań, który działa na następujących zasadach:

1. Zadanie przyjmujące (**org.cp.jamaica.server.RServer**) posiada 3 kolejki wątków, z których każda kolejka przewidziana jest dla innego typu zadania
2. Zadanie wywołujące (**org.cp.jamaica.client.Client**) posiadają wiedzę o zadaniu przyjmującym oraz o liście udostępnianych przez niego zadaniach
3. Klienci wywołują serwer przekazując nazwę zadania oraz samego siebie.
   1. Jeśli dla danego żądania klient znajduje się już w kolejce nie zostanie tam ponownie dopuszczony i po pewnym czasie ponowi swoje żądanie
   2. Jeśli dla danego żądanie klient zostanie umieszczony w kolejce oczekujących przejdzie w stan oczekiwania, a jego wątek zostanie wznowiony w momencie, gdy serwer wywoła klienta (wykona usługę) i przekaże mu zasoby danego typu.
      1. **org.cp.monitor.resources.ResourceA**
      2. **org.cp.monitor.resources.ResourceB**
      3. **org.cp.monitor.resources.ResourceA** i **org.cp.monitor.resources.ResourceB**
   3. Usługa jest w tym wypadku trywialna I polega na zalogowaniu faktu otrzymania zasobów i uśpieniu wątku na pewien czas, co ma symulować wykonywania usługi
4. Klient po zakończeniu wywołania usługi jest zobowiązany do zwrotu zasobów przy kolejnym spotkaniu, aby umożliwić dostęp innym klientom
5. Serwer nadzoruje wywołania w sposób niedeterministyczny wybierając typ zadania do sprawdzania, następnie wybierając oczekującego klienta oraz przekazując mu zasoby – wywołując usługę.

# Wnioski

Najtrudniejsze w zadaniu była próba zrozumienia i przestawienie się na paradygmat współbieżności, którego nie spotkaliśmy do tej pory w języku Java. Niemniej korzystając z wbudowanych mechanizmów:

* Synchronizowanych metod (będących w naszym wypadku wywołaniem serwera oraz usługi klienta)
* Kolejek (celem buforować żądania klientów)

Byliśmy w stanie zasymulować spotkanie polegające na zsynchronizowaniu się serwera oraz klienta poprzez przekazania oraz zwrot zasobów.