|  |
| --- |
| Programowanie współbiezne |
| Laboratorium 5: Podstawowe problemy programowania współbieżnego |
| Rozszerzony problem Producenta i Konsumenta |
|  |
| **Tomasz Trębski – 165535**  **Karol Górecki - 165405** |
| **12/2/2013** |

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

Table of Contents

[Wprowadzenia 2](#_Toc373691022)

[Treść zadania 2](#_Toc373691023)

[Teoria 2](#_Toc373691024)

[Klasyczny problem producenta i konsumenta 2](#_Toc373691025)

[Implementacja 2](#_Toc373691026)

[Wnioski 3](#_Toc373691027)

# Wprowadzenia

Celem zadania było zaimplementowanie współbieżnego programu, który rozwiązuje postawiony przed nami problem.

# Treść zadania

Każdy z trzech dostawców oferuje inną parę spośród koniecznych składników (bibułka, tytoń   
i zapałki). Każdy z trzech palaczy ma jeden z wymienionych składników i potrzebuje dwóch składników oferowanych przez któregoś z dostawców.

# Teoria

## Klasyczny problem producenta i konsumenta

Problem producenta i konsumenta to klasyczny informatyczny problem synchronizacji.   
W problemie występują dwa rodzaje procesów: producent i konsument, którzy dzielą wspólny zasób   
- bufor dla produkowanych (konsumowanych) jednostek. Zadaniem producenta jest wytworzenie produktu, umieszczenie go w buforze i rozpoczęcie pracy od nowa. W tym samym czasie konsument ma pobrać produkt z bufora. Problemem jest taka synchronizacja procesów, żeby producent nie dodawał nowych jednostek, gdy bufor jest pełny, a konsument nie pobierał, gdy bufor jest pusty.

Rozwiązaniem dla producenta jest uśpienie procesu w momencie, gdy bufor jest pełny. Pierwszy konsument, który pobierze element z bufora budzi proces producenta, który uzupełnia bufor.   
W analogiczny sposób usypiany jest konsument próbujący pobrać z pustego bufora. Pierwszy producent, po dodaniu nowego produktu umożliwi dalsze działanie konsumentowi. Rozwiązanie wykorzystuje komunikację międzyprocesową z użyciem semaforów. Nieprawidłowe rozwiązanie może skutkować zakleszczeniem.

# Implementacja

Naszym zadaniem podczas tego laboratorium było zaimplementowanej rozszerzonej wersji tego problemu. Musieliśmy 3 wątki, każdy z nich produkował jeden z trzech produktów, a próbował pobrać dwa pozostałe. Za podstawę tych wątków posłużyła klasa **org.cp.last.producent.AbstractProducer** implementująca interfejs Runnable[[1]](#footnote-1). Zaś za bufor wykorzystaliśmy poszczególne rozszerzenia **org.cp.last.products.ProductPool** zawierające w sobie kolekcje określonego produktu. Pojedynczy blok krytyczny wygląda następująco:

1. Jeżeli bufor nie jest pełny stwórz określoną liczbę swoich produktów
2. Pobierz dwa pozostałe produkty

1. <http://docs.oracle.com/javase/7/docs/api/java/lang/Runnable.html> [↑](#footnote-ref-1)