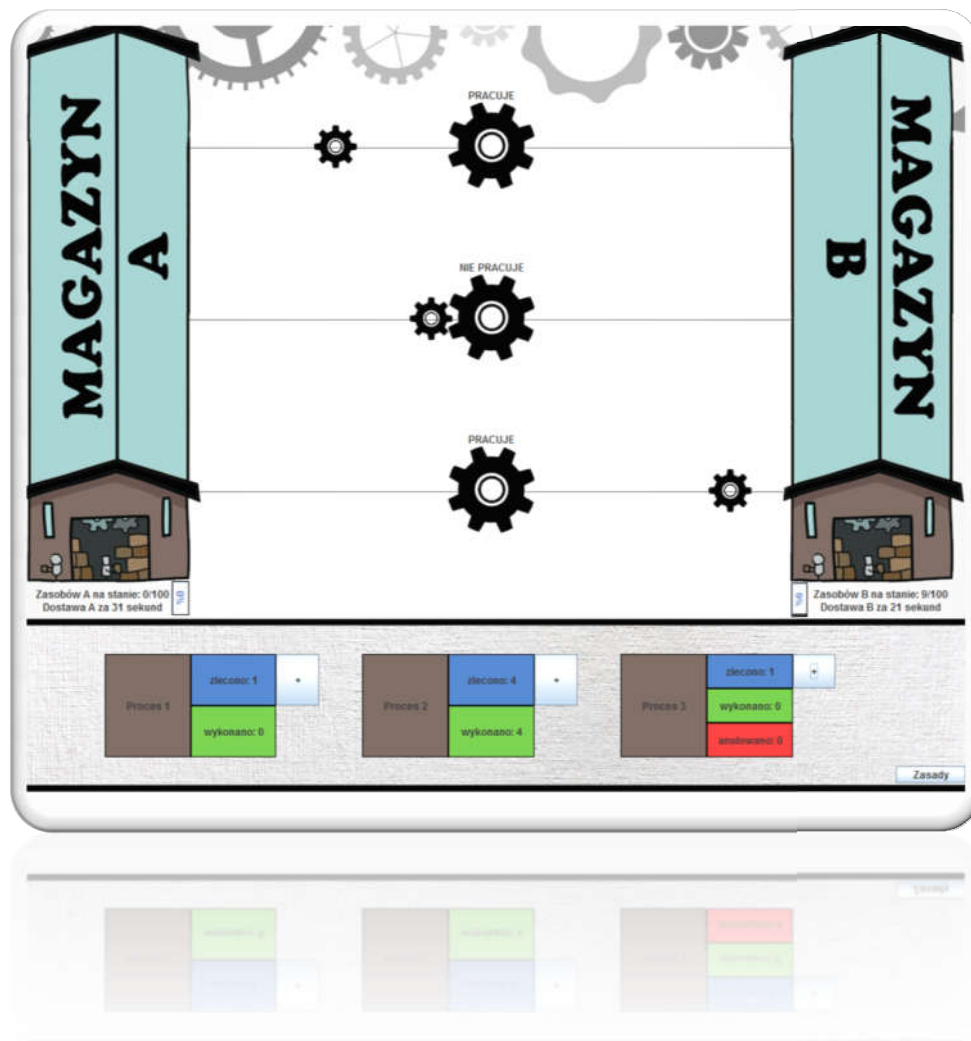


## PROCESY - projekt zaliczeniowy



Prowadzący: mgr Janusz Tuchowski

Uniwersytet Ekonomiczny w Krakowie,  
Kierunek: Informatyka Stosowana,  
Projekt wykonany w ramach zajęć z przedmiotu Programowanie współbieżne i równoległe.

## Spis treści

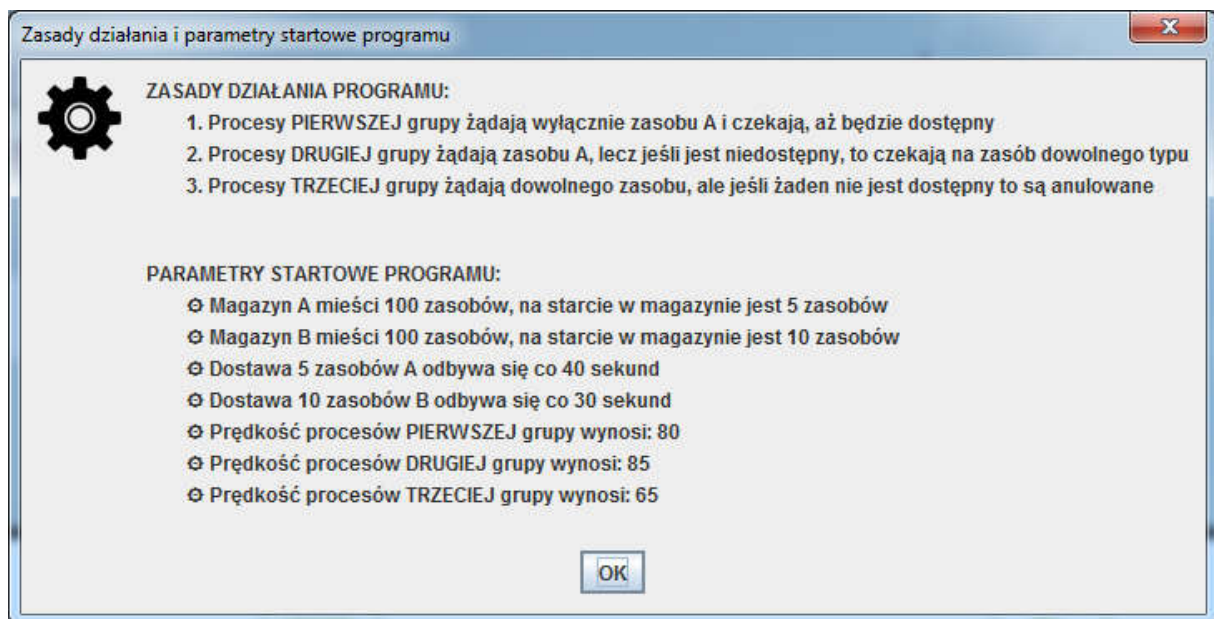
Tematyka projektu .....	3
Proponowane rozwiązanie .....	4
Wykorzystane klasy .....	4
Opis klas.....	5
Związki pomiędzy poszczególnymi komponentami .....	7
Sposób obsługi aplikacji.....	8
Struktura katalogów projektu .....	11

## Tematyka projektu

W systemie znajdują się dwa typy zasobów A i B, które są przechowywane w magazynach. Zasoby są wymienne, ale pierwszy z nich jest wygodniejszy niż drugi. Zasobów w magazynie B jest więcej niż zasobów w magazynie A. Dostawy do magazynów odbywają się cyklicznie. W systemie działają trzy grupy procesów, które różnią się sposobem zgłaszania zapotrzebowania na zasób:

- Procesy pierwszej grupy żądają wyłącznie zasobu A i czekają, aż będzie dostępny.
- Procesy drugiej grupy żądają zasobu A, lecz jeśli jest niedostępny, to czekają na zasób dowolnego typu.
- Procesy trzeciej grupy żądają dowolnego zasobu, ale jeśli żaden nie jest dostępny to są anulowane.

W prawym dolnym rogu aplikacji znajduje się przycisk "Zasady", który otwiera następujące okno wyjaśniające zasady działania programu oraz jego parametry startowe:



## Proponowane rozwiązanie

Proponowanym rozwiązaniem jest utworzenie trzech semaforów wykorzystywanych przez trzy różne grupy procesów. Następnie należy utworzyć 3 przyciski umożliwiające zlecenie procesu danej grupy do wykonania. W momencie kliknięcia w przycisk zostaje utworzony nowy wątek, który jeżeli semafor jest wolny uzyskuje dostęp do sekcji krytycznej aplikacji i się wykonuje, lub czeka na zwolnienie blokady przez semafor. Wykonywanie wątku polega na przebyciu przez zasób (małej zębatki) drogi z magazynu do procesu (dużej zębatki). Każdy wątek jest ponadto wstrzymywany jeżeli nie ma w magazynie zasobu, którego proces potrzebuje, aby się wykonać. Po wykonaniu zadania przez wątek, semafor oraz liczba wykonanych procesów danej grupy zostają podniesieni. Dostawy zasobów do magazynów A, B będą się odbywać co określony czas jako obiekty klasy Timer. W celu przedstawienia aktualnego stanu aplikacji niezbędne będzie utworzenie kilku etykiet. Dodatkowo w celu poprawy czytelności można utworzyć paski postępu odzwierciedlające stan magazynu w sposób graficzny.

## Wykorzystane klasy

- Main
- Ramka
- Panel
- Procesy
- Proces1
- Proces2
- Proces3

## Opis klas

W klasie **Main** zawarta jest metoda `main()`, w ciele której tworzony jest nowy obiekt klasy **Ramka** oraz wyłączana jest możliwość zmieniania rozmiaru okna.

W klasie **Ramka** dziedziczącej z klasy **JFrame** znajduje się konstruktor, który wywołuje konstruktor nadklasy, nadając tytuł aplikacji, następnie tworzy nowy obiekt klasy **JPanel**, dodaje go do ramki, ustala domyślne zachowanie aplikacji w przypadku próby zamknięcia, tak, aby była ona zamykana. Następnie wyśrodkowuje ramkę oraz dopasowuje rozmiar ramki do jej zawartości, na koniec konstruktor uwidocznia ramkę.

W klasie **Panel** rozszerzającej **JPanel** definiowane są komponenty, obiekty odpowiedzialne za grafikę, etykiety, przyciski, paski postępu oraz tworzone są obiekty odzwierciedlające wykorzystywane kolory w aplikacji. W końcowym bloku są definiowane zmienne określające położenie X,Y zasobu na linii każdego procesu. W konstruktorze klasy **Panel** wczytywane są potrzebne grafiki z katalogu `src/resources/images/`. Następnie ustalane są rozmiary panelu na podstawie wymiarów grafiki w tle czyli `tloPanelu.png`. W kolejnych liniach kodu tworzone są etykiety wyświetlające status procesów, zasoby na stanie w magazynach, paski postępu, w przypadku obsługi etykiety odnoszącej się do czasu dostaw A i B tworzone są dodatkowo obiekty klasy **ActionListener** oraz przesłaniane metody `actionPerformed()`, których zadaniem w obiekcie klasy **Timer** jest obsługa czasu dostawy, oraz zmiana zasobów na stanie. W dalszej części konstruktora klasy **Panel** tworzone są etykiety odpowiedzialne za statystyki procesów oraz przyciski dla których dodane są **ActionListenery**, które w przypadku kliknięcia przycisku tworzą nowy wątek procesu danej grupy. Pod koniec kodu można odnaleźć obiekt odswieżaj klasy **ActionListener**, którego przesłonięta metoda `actionPerformed()` ma za zadanie obsługę pasków postępu, przypadku zapelnienia magazynu (zmiana koloru paska postępu) oraz przede wszystkim wywołanie metody `repaint()`. Obiekt odswieżaj jest wywoływany przez obiekt klasy **Timer**. Kolejny blok kodu odpowiedzialny jest za obsługę przycisku Zasady - poprzez wyświetlenie zasad działania programu oraz jego parametrów startowych. Poza konstruktorem w klasie **Panel** wywoływana jest metoda `paintComponent` w ciele której tworzony jest obiekt klasy **Graphics2D** za pomocą którego rysowane są obrazki.

W klasie **Procesy** tworzone są zmienne statyczne, które są wykorzystywane w aplikacji. Na początku są tworzone 3 Semaforey, po jednym dla każdego procesu, następnie statusy procesów odzwierciedlające stany w jakich mogą znajdować się procesy. W bloku kodu opatrzonego komentarzem "konfiguracja programu" znajdują się zmienne umożliwiające zarządzanie zachowaniem aplikacji, zmianę: rozmiaru magazynów, ilości zasobów znajdujących się w magazynach na starcie, wysokość dostaw, co ile dostawy mają się odbywać oraz prędkości wykorzystywania zasobów przez procesy. Wszystkie powyższe zmienne są stałymi. Kolejne zmienne już bez słówka final odzwierciedlają w aplikacji ilość zasobów na stanie w magazynach oraz ile procesów zostało zleconych, wykonanych i anulowanych. W końcowym bloku kodu klasy znajdują się zmienne logiczne odzwierciedlające czy dany proces pracuje oraz z którego magazynu pobrał zasób.

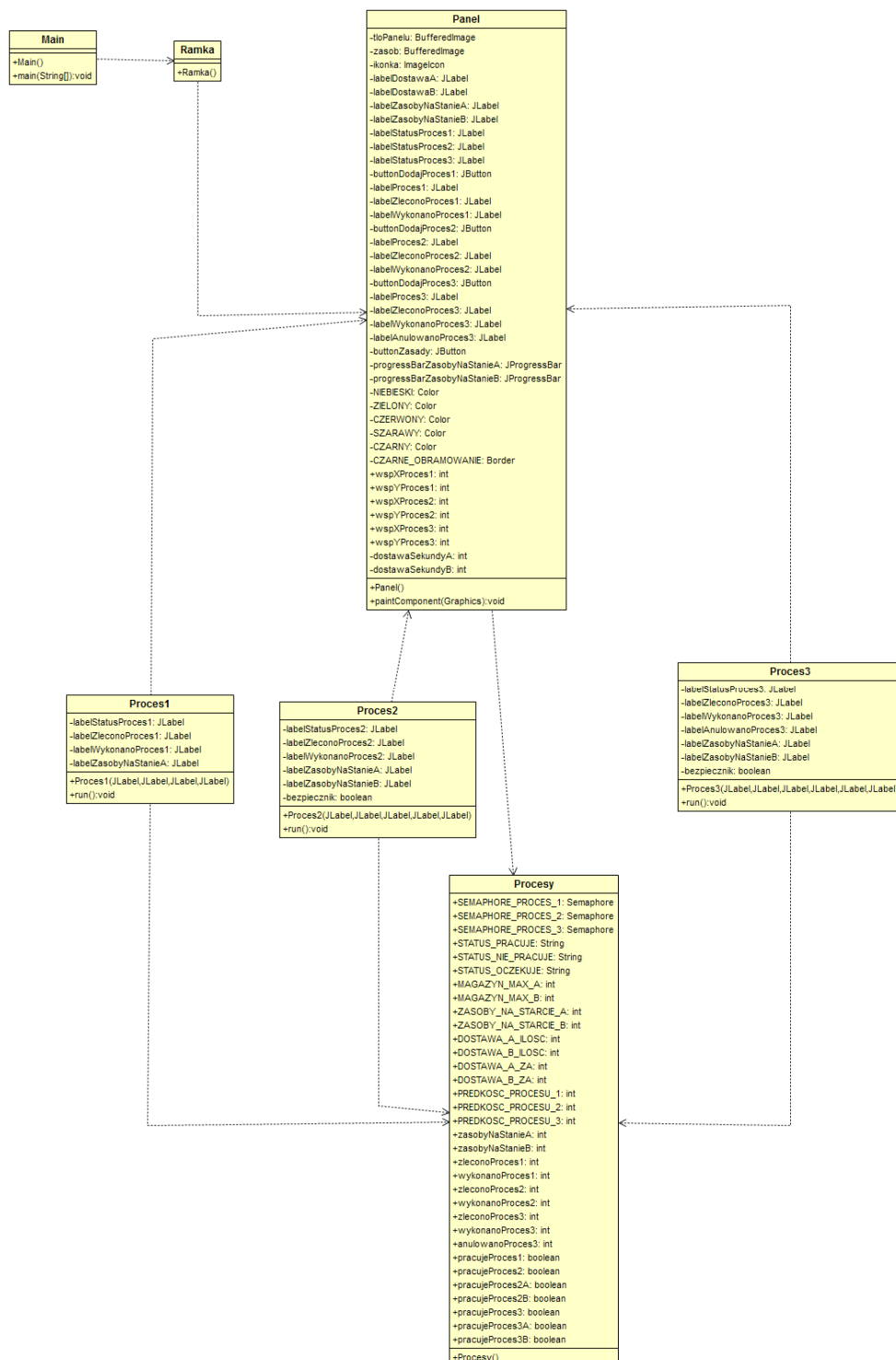
W klasie **Proces1** dziedziczącej z klasy Thread nadpisywana jest metoda run() umożliwiając obsługę procesów pierwszej grupy oraz aktualizację zmiennych przedstawiających stan aplikacji.

W klasie **Proces2** dziedziczącej z klasy Thread nadpisywana jest metoda run() umożliwiając obsługę procesów drugiej grupy oraz aktualizację zmiennych przedstawiających stan aplikacji.

W klasie **Proces3** dziedziczącej z klasy Thread nadpisywana jest metoda run() umożliwiając obsługę procesów trzeciej grupy oraz aktualizację zmiennych przedstawiających stan aplikacji.

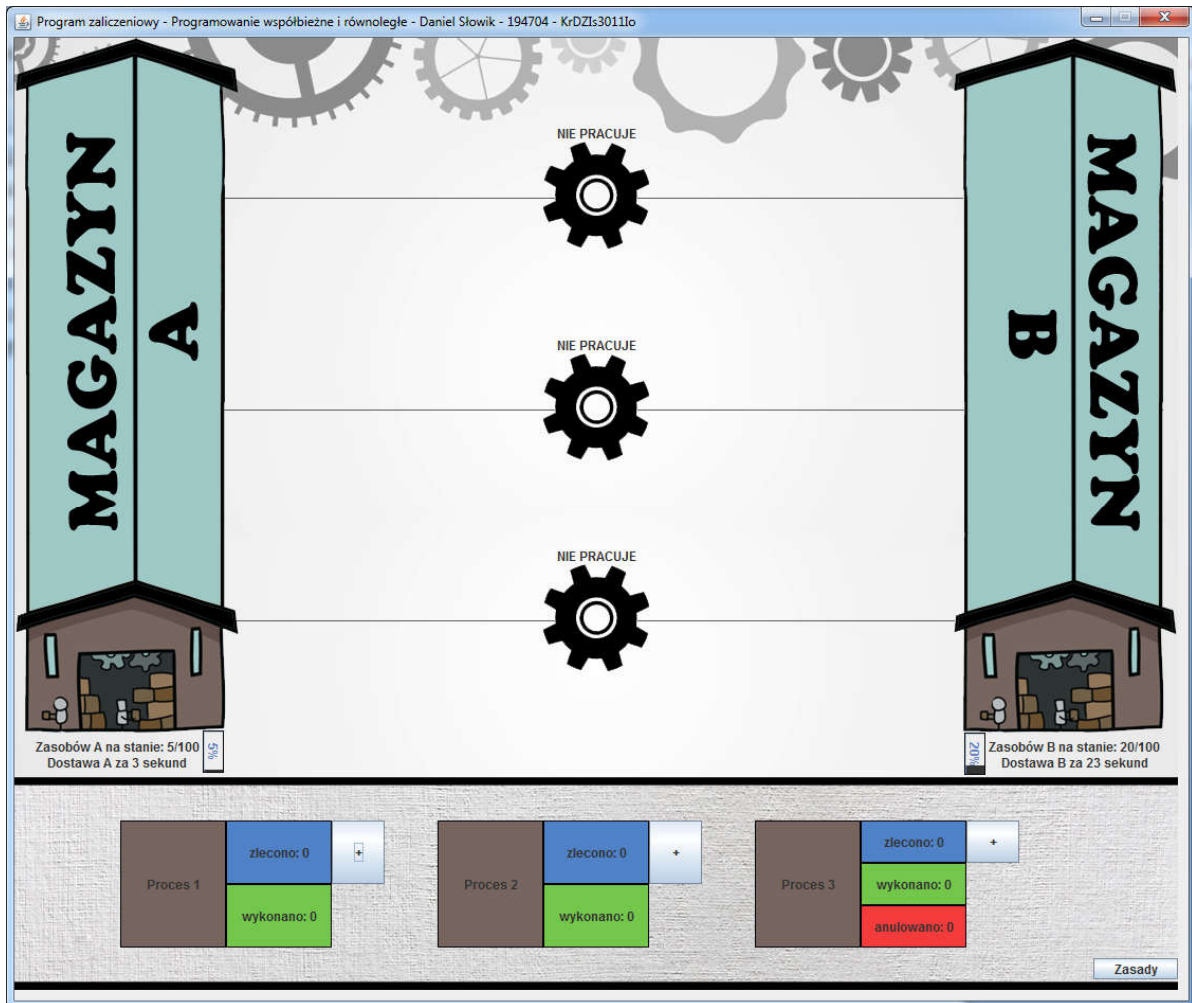
## Związki pomiędzy poszczególnymi komponentami

Diagram UML został stworzony przy użyciu "The ObjectAid UML Explorer for Eclipse"



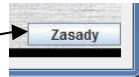
Uniwersytet Ekonomiczny w Krakowie,  
Kierunek: Informatyka Stosowana,  
Projekt wykonany w ramach zajęć z przedmiotu Programowanie współbieżne i równoległe.

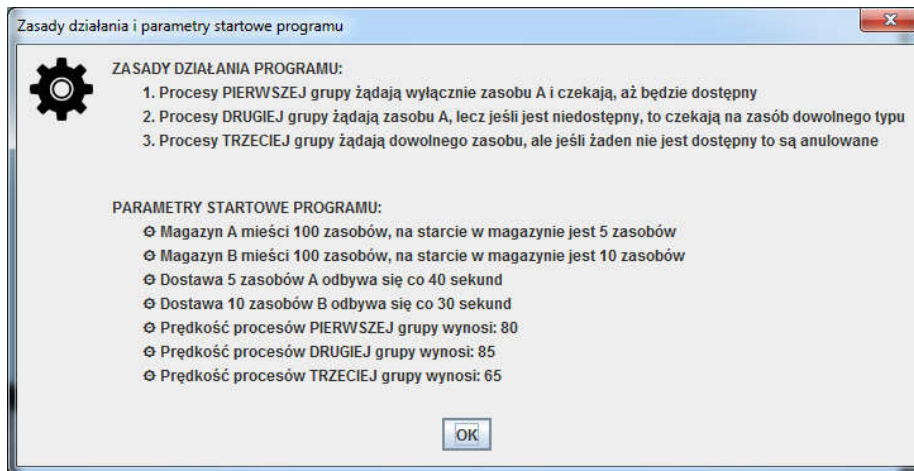
## Sposób obsługi aplikacji



Uniwersytet Ekonomiczny w Krakowie,  
Kierunek: Informatyka Stosowana,  
Projekt wykonany w ramach zajęć z przedmiotu Programowanie współbieżne i równoległe.



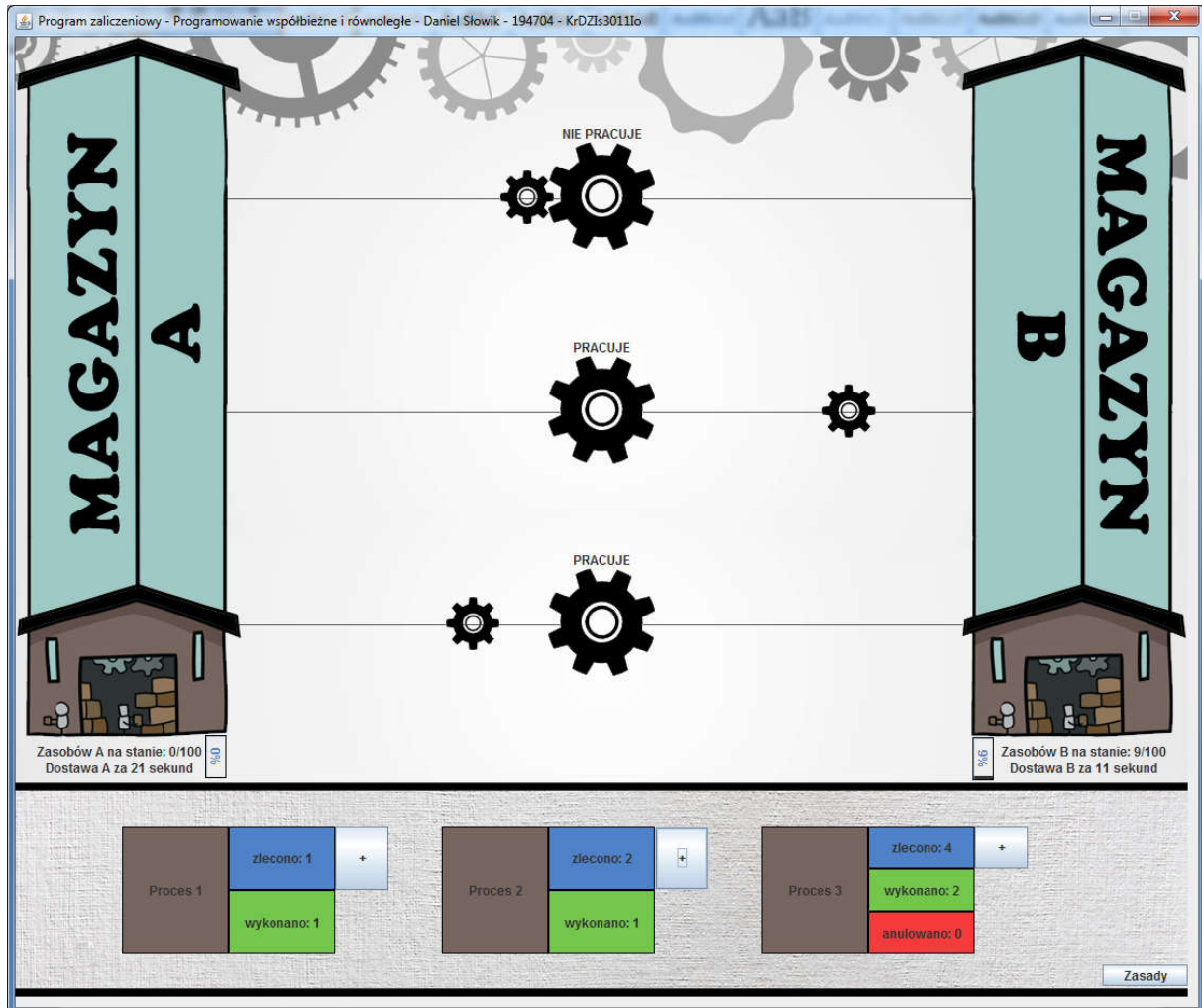
Obsługa aplikacji sprowadza się do kliknięcia przycisku "Zasady"  w celu wyświetlenia poniższego okna:



Zlecanie procesów do wykonania odbywa się poprzez kliknięcie przycisku "+" przy procesie, który chcemy zlecić:

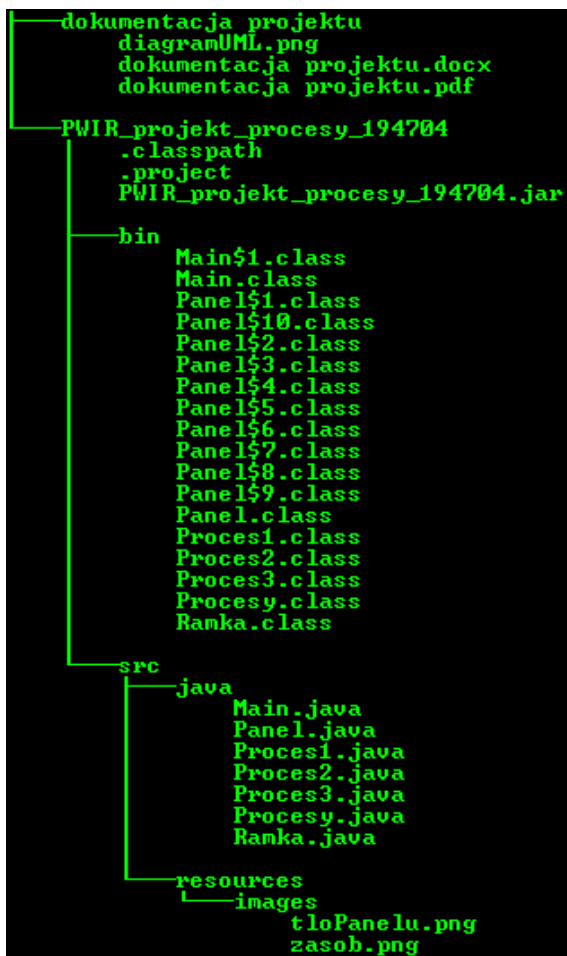


co powoduje utworzenie nowego wątku procesu danej grupy. Przykładowy efekt jest widoczny poniżej:



Uniwersytet Ekonomiczny w Krakowie,  
Kierunek: Informatyka Stosowana,  
Projekt wykonany w ramach zajęć z przedmiotu Programowanie współbieżne i równoległe.

## Struktura katalogów projektu



Uniwersytet Ekonomiczny w Krakowie,  
Kierunek: Informatyka Stosowana,  
Projekt wykonany w ramach zajęć z przedmiotu Programowanie współbieżne i równoległe.