Programowanie obiektowe

**Projekt nr 3**

Wojciech Zieliński

Wtorek 10.15

Sala 09

Politechnika Warszawska

Wydział Elektroniki i Technik Informacyjnych

Prowadząca laboratorium: mgr inż. Anna Jusza

**1.Treść projektu**

Oprogramować graf modelujący abstrakcyjny system kolejkowy, w którym:

* węzeł grafu odpowiada stanowisku przetwarzania (np. malowanie karoserii),
* stanowisko produkuje obiekt danego typu (np. pomalowana karoseria),
* ze stanowiskiem związanych jest kilka (rozróżnialnych) kolejek, z których pobiera ono elementy, z których składa produkt swojego działania (np. stanowisko montujące silnik w karoserii ma dwie kolejki: karoserii i silników),
* skierowana krawędź odpowiada możliwości przekazania przez stanowisko źródłowe efektu swojej pracy do stanowiska docelowego.

Posługując się powyższym modelem zaprogramować symulator działania fabryki samochodów, w której są 4 stanowiska produkujące opony, 2 stanowiska produkujące karoserie, 2 stanowiska malujące karoserię, 2 stanowiska montujące opony do karoserii, 2 stanowiska produkujące silniki w karoseriach oraz 2 stanowiska montujące elektronikę w karoseriach z silnikami.

W symulacji przyjąć:

1. stanowisko zwraca efekt swojej pracy po jednostce czasu,
2. stanowisko przekazuje efekt swojej pracy temu następnemu stanowisku, przed którym jest najkrótsza kolejka
3. stanowisko wstrzymuje swoje działanie na jednostkę czasu, jeśli wszystkie kolejki w następnych stanowiskach są dłuższe niż 10

Oprogramować wizualizację symulacji. Ustalić eksperymentalnie gdzie robią się najdłuższe kolejki.

**2. Propozycja rozwiązania**

Do obsługi elementów utworzyłem klasę Element, po której dziedziczą klasy Opona, Karoseria oraz Silnik. Do obsługi stanowisk utworzyłem klasę Stanowisko, po której dziedziczą klasy ProdukcjaOpon, ProdukcjaKaroserii, ProdukcjaSilnikow, MalowanieKaroserii, OponyWKaroserii, SilnikWKaroserii oraz ElektronikaWKaroserii. Do określenia relacji między poszczególnymi stanowiskami użyłem klasy Krawedz. Do oprogramowania całej fabryki w postaci utworzyłem klasę Graf.

Poszczególne stanowiska wykonują określone operacje na elementach i mogą je przekazywać dalej po wykonaniu operacji. Założyłem, że w elementy Opona oraz Silnik są tylko przekazywane między odpowiednimi stanowiskami i ich obecność na odpowiednim stanowisku - OponyWKaroserii oraz SilnikWKaroserii jest konieczna do możliwości uruchomienia działania tych stanowisk. Element Karoseria po utworzeniu jest przekazywany między kolejnymi stanowiskami i odpowiednio w nich przetwarzany (zmieniana jest wartość logiczna pól określających pomalowanie karoserii, obecność opon, silnika oraz elektroniki). Każde stanowisko, które działa na karoserii musi mieć dostarczoną karoserię. Kiedy stanowisko wykona określone operacje na elemencie ten element jest przekazywany dalej i usuwany z tego stanowiska.

W klasie Krawedz ustalamy połączenie między dwoma konkretnymi stanowiskami. Jest to połączenie skierowane, więc musimy rozróżniać stanowisko poprzednie oraz następne. Mamy również w tej klasie kolejkę, zrealizowaną za pomocą kontenera z biblioteki STL – queue, w której są elementy oczekujące na przekazanie elementów ze stanowiska poprzedniego do następnego. Element ze stanowiska poprzedniego jest przekazywany na koniec kolejki a do stanowiska następnego pobieramy pierwszy element z kolejki. Każde stanowisko następne łączy się z kilkoma stanowiskami tego samego rodzaju.

W klasie Graf ustalamy całą strukturę grafu. Te informacje przechowujemy w dwóch wektorach, które są kontenerami z biblioteki STL – vector. Pierwszy z nich zawiera informacje o wszystkich stanowiskach. Drugi z nich zawiera informacje o wszystkich połączeniach między stanowiskami. W tej klasie są funkcje, które wykorzystują funkcje z innych klas, aby odpowiednio sterować działaniem programu.

Do obsługi wyjątków związanych z tym, że stanowisko nie może działać, jeżeli nie przekazano do niego odpowiedniego elementu, utworzyłem klasę WyjatkiRzutowanie, w której jest odpowiednia funkcja do wypisywania komunikatu.

**3. Główne założenia**

W trakcie działania programu każde stanowisko wypisuje na ekran komunikat o tym, czy wykonało swoje działanie poprawnie, czy nie.

Jeżeli kolejka jest pusta, to nie możemy z niej pobrać żadnego elementu i wtedy również wypisywany jest odpowiedni komunikat.

Z ilości konkretnych stanowisk wynika, że każde stanowisko, które jest z czymś połączone ma dwa połączenia – lewe i prawe. Po wykonaniu swojego działania stanowisko przekazuje element do tego stanowiska, w którym jest mniejsza kolejka. W przypadku równych długości kolejek element jest najpierw przekazywany do lewej kolejki. Z tego wynika, że długości kolejek pochodzących od jednego stanowiska różnią się maksymalnie o 1.

Jeżeli obydwie kolejki wychodzące z danego stanowiska są równe 10, to wtedy stanowisko wstrzymuje swoją pracę i nic nie wykonuje. Wypisywany jest wtedy odpowiedni komunikat.

Stanowiska, które pobierają elementy, biorą je z tej kolejki, w której jest najwięcej elementów. W przypadku równej ilości pobierany jest element z lewej kolejki (oprocz stanowiska OponyWKaroserii).W przypadku stanowisk OponyWKaroserii oraz SilnikWKaroserii są w nich dwa rodzaje kolejek. Warto dodać, że stanowisko OponyWKaroserii pobiera po jednej oponie z każdego stanowiska produkującego opony. Przyjąłem takie założenie, ponieważ te stanowiska pracują tak samo szybko i z moich testów wyniknęło, że pracują niezawodnie, więc będą pobierały te elementy niezawodnie.

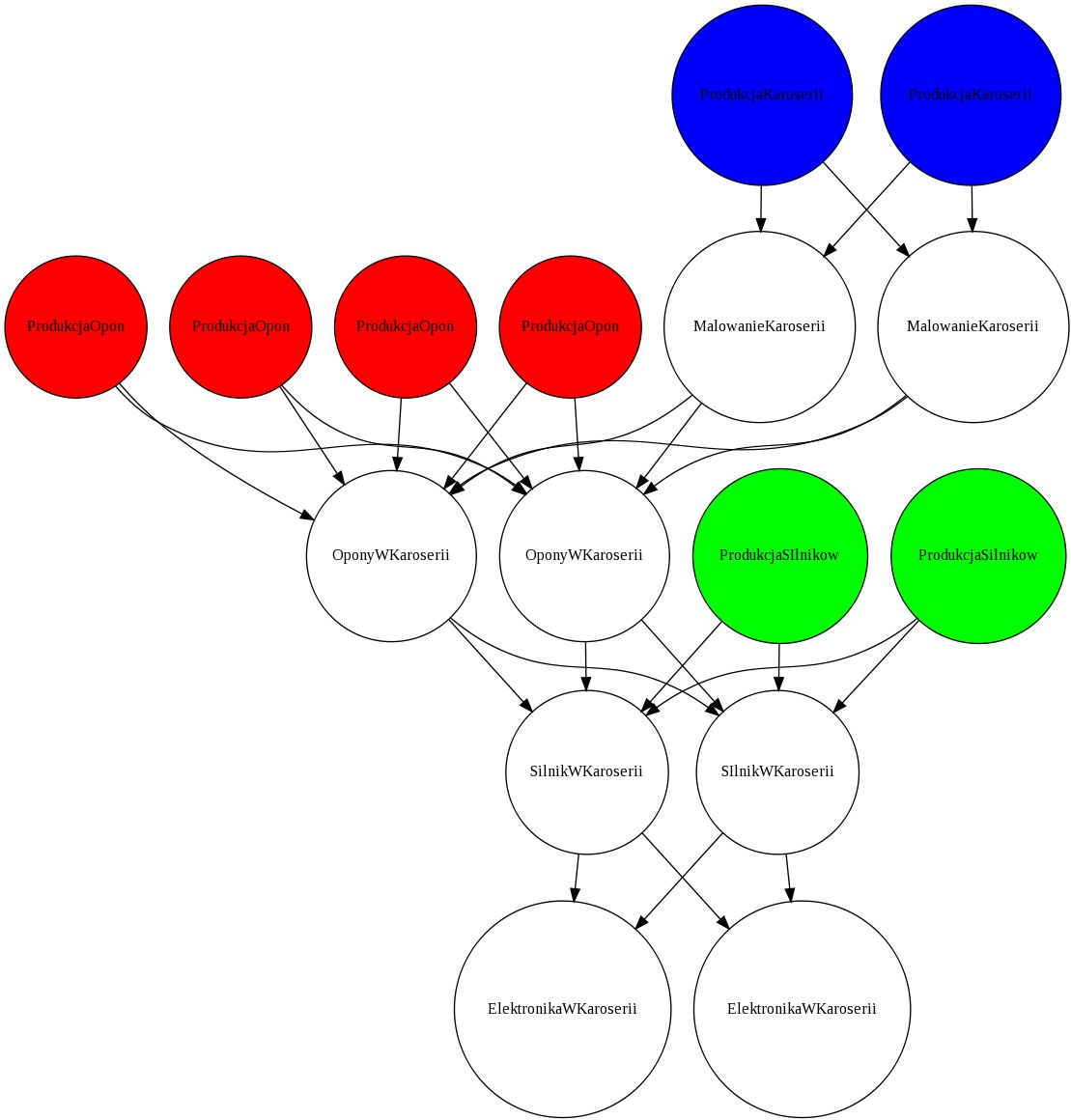
W przykładowej symulacji zakładam, że stanowiska są wywoływane po jednym razie „od góry” i działają tak określoną ilość razy. W takim przypadku nie robią się kolejki, ponieważ elementy są płynnie przekazywane i tylko kilka razy zdarza się, że stanowisko nie zadziałało, bo nie dostarczono do niego wystarczającej ilości elementów.

Kolejki tworzą się jeżeli dane stanowisko w trakcie jednego cyklu działania fabryki zostaje uruchomione kilka razy. Największe kolejki robią się w stanowiskach ProdukcjaKaroserii oraz ProdukcjaSilnikow, bo te stanowiska muszą przekazać tylko jeden element dalej, aby kolejne stanowiska poprawnie działały i są one na samym dole linii produkcyjnej.

Do pliku zapisywane są efekty pracy ostatnich stanowisk (ElektronikaWKaroserii) w danym cyklu.

**4. Odstępstwa**

W przypadku dobrego ustawienia działania fabryki nie występują żadne poważne problemy. Jeżeli źle ustalimy kolejność działania stanowisk, to mogą się tworzyć duże kolejki i mało elementów będzie tworzonych. W moim programie nie zapewniłem, aby stanowiska uruchamiały się dokładnie w tym samym czasię, tylko są uruchamiane cyklicznie. Gdyby coś poszło nie tak z produkcją opon, w którymś stanowisku to opony zostałyby pobrane poprawnie dopiero gdy w którymś innym stanowisku były 4 w kolejce, ponieważ przyjąłem takie założenie z przekazywaniem opon.

**5. Schemat fabryki**

**6. Testowanie**

Testowanie prowadziłem przez cały proces pisania programu sprawdzając, czy tworzone funkcje spełniają przyjęte przeze mnie założenia. Końcowe testy doprowadziły mnie do stwierdzenia, że w przypadku poprawnej kolejności uruchamiania stanowisk, fabryka pracuje niezawodnie i w ciągu danej ilości cykli pracy tworzy tyle samochodów ile tych cykli było. Sprawdzałem również, czy dane zapisywane do pliku zgadzają się z tymi, które są wyświetlane na ekranie.

**7. Interfejs użytkownika**

W moim programie użytkownik ma możliwość monitorowania działania fabryki. W każdym cyklu są wyświetlane informacje o działaniu każdego stanowiska. Zastosowałem funkcje z biblioteki <time> do tego, aby po każdym cyklu nastąpiła krótka przerwa, która pozwala na obserwację działania fabryki i wyciąganie potrzebnych wniosków. Uznałem, że w przypadku tego projektu zbędna jest ingerencja użytkownika w schemat działania fabryki. Dostęp do tych zmian ma osoba pisząca program. W celu zmiany ilości stanowisk lub ich kolejności należy dokonać odpowiednich zmian w funkcji main.