Politechnika
Wrocławska

Programowaniie Obiektowe

projekt wtorek 17:05

Symulacja Domu Towarowego

Autor:
Gabriel Malanowski 281081
Kamil Kondrat 281177

Prowadzący:
mgr inż. Tobiasz Puślecki

1 Wstęp

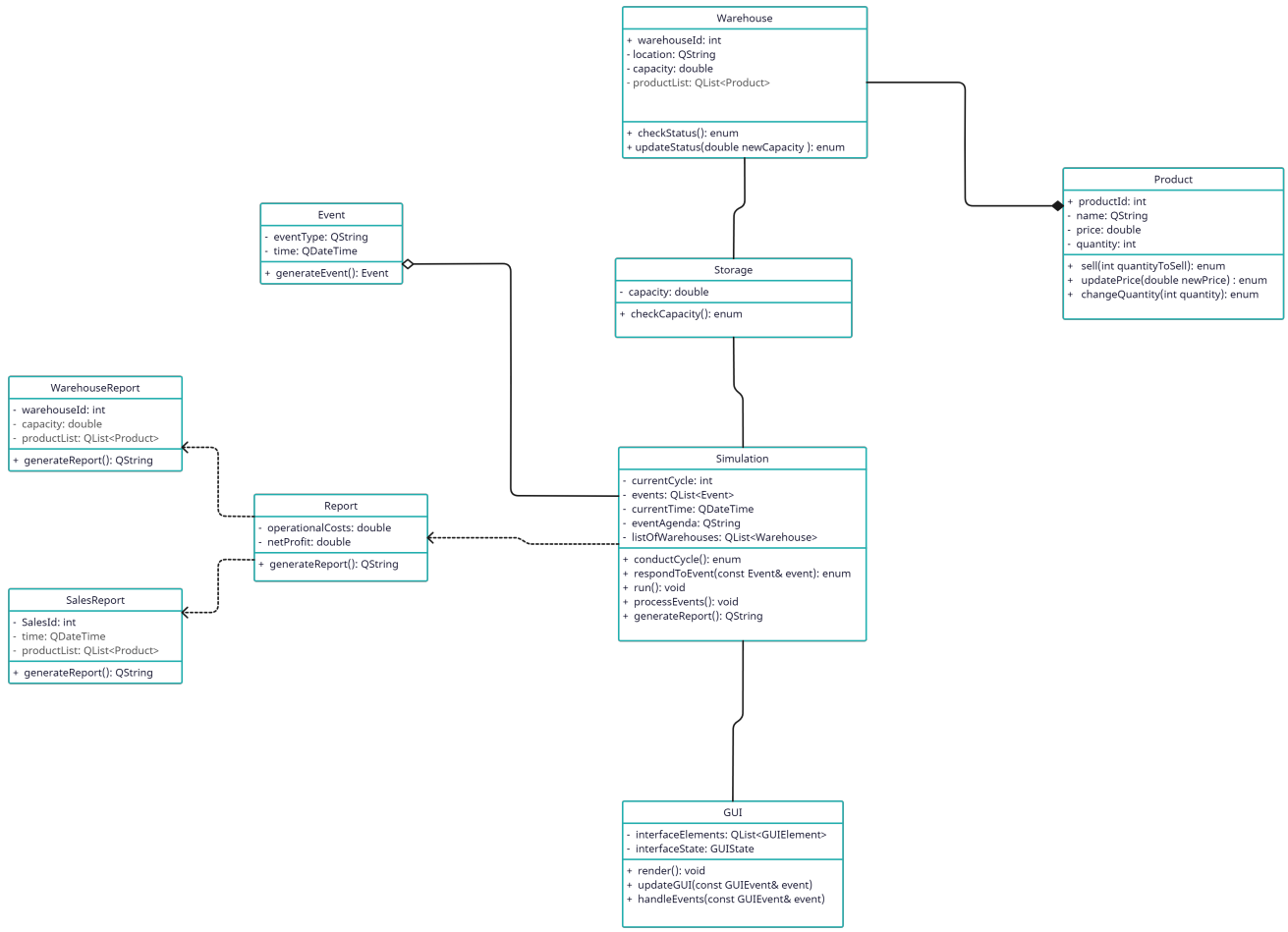
Projekt Symulacja Domu Towarowego to system wspomagający zarządzanie magazynem, który umożliwia symulowanie i optymalizowanie procesów magazynowych. Dzięki niemu użytkownicy mogą modelować różne scenariusze, analizować wyniki oraz podejmować decyzje w czasie rzeczywistym. Projekt integruje algorytmy symulacyjne z interaktywnym interfejsem użytkownika, co pozwala na dynamiczne monitorowanie stanu magazynu oraz szybką reakcję na zmiany w otoczeniu biznesowym. Główne cechy projektu obejmują definiowanie magazynów, produktów, atrybutów, symulację działania magazynu w pętli czasowej, generowanie zdarzeń, interwencję użytkownika oraz generowanie raportów z wynikami symulacji. Celem projektu jest dostarczenie narzędzia wspomagającego efektywne zarządzanie magazynem, które pozwoli firmom na zwiększenie efektywności operacyjnej i maksymalizację zysków.

2 Opis

2.1 Ogólny opis symulacji

Użytkownik podaje informacje o magazynach, produktach w magazynie itd. Każdy obiekt ma swoje atrybuty, takie jak pojemność magazynu, cena produktu. Symulacja działa w pętli czasowej, gdzie każdy cykl reprezentuje jednostkę czasu (np. godzinę, dzień). W każdym cyklu, symulacja sprawdza stan magazynu i podejmuje decyzje na podstawie zdefiniowanych reguł. Zdarzenia takie jak przyjęcie nowego towaru, sprzedaż produktu, lub zmiana zapotrzebowania są generowane losowo lub według określonego harmonogramu. Symulacja reaguje na te zdarzenia, aktualizując stan magazynu i inne powiązane obiekty. Na podstawie stanu magazynu i nadchodzących zdarzeń, symulacja podejmuje decyzje, takie jak zamówienie nowego towaru, przesunięcie zasobów, czy wysłanie powiadomień do administratorów. Wszystkie działania i zmiany są rejestrowane w systemie, co pozwala na analizę wyników symulacji i optymalizację procesów magazynowych. Użytkownik może interweniować w symulację, wprowadzając zmiany w strategii zarządzania magazynem lub reagując na generowane zdarzenia. Na koniec symulacji, użytkownik otrzymuje raport z wynikami, takimi jak koszty operacyjne, zysk netto itp.

2.2 Diagram klas



Opis diagramu klas dla symulacji domu towarowego w języku C++, który spełnia wymienione warunki:

1. Klasa **Simulation** :
   - **Atrybuty:** currentCycle, events, currentTime, eventAgenda, listOfWarehouses.
   - **Metody:** conductCycle(), respondToEvent(), run(), processEvents(), generateReport().
2. Klasa **Storage** :
   - **Atrybuty:** capacity.
   - **Metody:** checkCapacity().
3. Klasa **Warehouse** :
   - **Atrybuty:** warehouseId, location, capacity, productList.
   - **Metody:** checkStatus(), updateStatus(double newCapacity).
4. Klasa **Product** :
   - **Atrybuty:** productId, name, price, quantity.
   - **Metody:** sell(int quantityToSell), updatePrice(double newPrice) changeQuantity(int quantity).

5. Klasa **Event** :

    - **Atrybuty:** eventType, time.
    - **Metody:** generateEvent().

6. Klasa **Report** :

    - **Atrybuty:** operationalCosts, netProfit.
    - **Metody:** generateReport().

7. Klasa **WarehouseReport** :

    - **Atrybuty:** warehouseId, capacity, productList.
    - **Metody:** generateReport().

8. Klasa **SalesReport** :

    - **Atrybuty:** salesId, time, productList.
    - **Metody:** generateReport().

9. Klasa **GUI** :

    - **Atrybuty:** interfaceElements, interfaceState.
    - **Metody:** render(), updateGUI(const GUIEvent& event), handleEvents(const GUIEvent& event).

Hermetyzacja jest zastosowana poprzez ustawienie atrybutów jako prywatnych (*private*) i dostęp do nich poprzez publiczne metody (*public*).

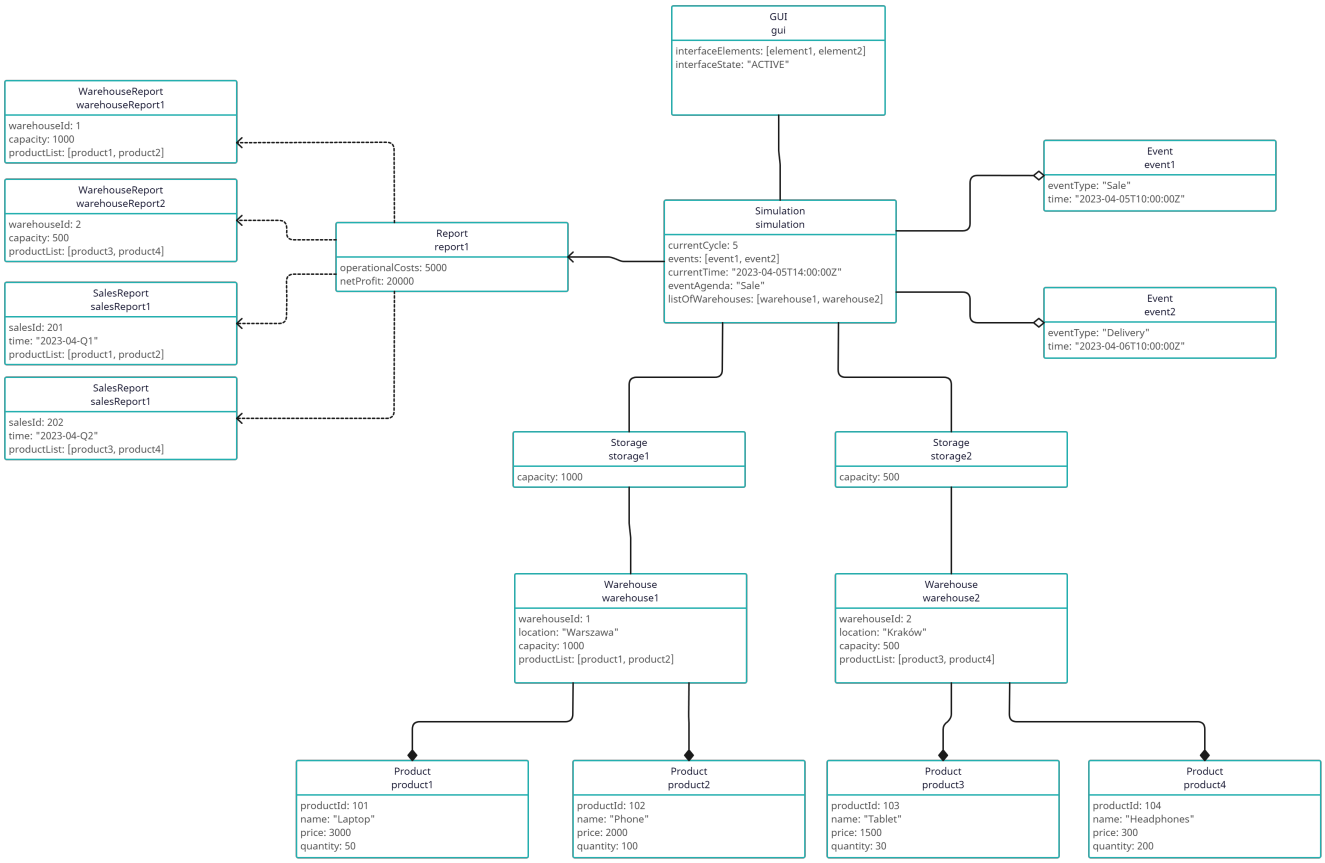
Dziedziczenie jest reprezentowane przez klasę *Warehouse*, która dziedziczy po klasie *Storage*.

Kompozycja występuje, gdy *Simulation* zawiera obiekty *Warehouse*, które z kolei zawierają obiekty *Product*..

Agregacja jest zastosowana w klasie *Simulation* dla klasy *Event*.

Polimorfizm może być reprezentowany przez różne typy zdarzeń, które są obsługiwane przez metodę *generateReport()* w klasie *Report*. Każde zdarzenie może mieć inną implementację tej metody, w zależności od jego typu.

2.3 Diagram obiektów



Opis diagramu obiektów dla przykładowej symulacji domu towarowego w języku C++.

1. Obiekty klasy **Simulation**
   - **simulation**: { currentCycle: 5, events: [event1, event2], currentTime: ”2023-04-05T14:00:00Z”, eventAgenda: ”Sale”, listOfWarehouses: [warehouse1, warehouse2] }
2. Obiekty klasy **Storage**
   - **storage1**: { capacity: 1000 }
   - **storage2**: { capacity: 500 }
3. Obiekty klasy **Warehouse**
   - **warehouse1**: { warehouseId: 1, location: ”Warszawa”, capacity: 1000, productList: [product1, product2] }
   - **storage2**: { warehouseId: 2, location: ”Kraków”, capacity: 500, productList: [product3, product4] }
4. Obiekty klasy **Product**
   - **product1**: { productId: 101, name: ”Laptop”, price: 3000, quantity: 50 }
   - **product2**: { productId: 102, name: ”Phone”, price: 2000, quantity: 100 }
   - **product3**: { productId: 103, name: ”Tablet”, price: 1500, quantity: 30 }

- **product4**: { productId: 104, name: "Headphones", price: 300, quantity: 200 }

5. Obiekty klasy **Event**
   - **event1**: { eventType: "Sale", time: "2023-04-05T10:00:00Z" }
   - **event2**: { eventType: "Delivery", time: "2023-04-06T10:00:00Z" }
6. Obiekty klasy **Report**
   - **report**: { operationalCosts: 5000, netProfit: 20000 }
7. Obiekty klasy **WarehouseReport**
   - **warehouseReport1**: { warehouseId: 1, capacity: 1000, productList: [product1, product2] }
   - **warehouseReport2**: { warehouseId: 2, capacity: 500, productList: [product3, product4] }
8. Obiekty klasy **SalesReport**
   - **salesReport1**: { salesId: 201, time: "2023-04-Q1", productList: [product1, product2] }
   - **salesReport2**: { salesId: 202, time: "2023-04-Q2", productList: [product3, product4] }
9. Obiekty klasy **GUI**
   - **gui**: { interfaceElements: [element1, element2], interfaceState: "ACTIVE" }

2.4 Szczegółowy opis działania symulacji

Użytkownik rozpoczyna symulację poprzez interfejs użytkownika **GUI**, wywołując metodę *run()* klasy **Simulation**. Ta metoda inicjuje główną pętlę symulacji, która będzie się wykonywać przez określoną liczbę cykli reprezentujących jednostki czasu zawartą w zmiennej *currentCycle*. W każdym cyklu symulacji, metoda *conductCycle()* jest wywoływana. Odpowiada ona za przetwarzanie zdarzeń zaplanowanych na bieżący cykl, które są przechowywane w atrybucie *events*.

Zdarzenia takie jak przyjęcie nowego towaru, sprzedaż produktu lub zmiana zapotrzebowania są generowane losowo lub według harmonogramu. Są one tworzone przez metodę *generateEvent()* klasy **Event** i dodawane do kolejki zdarzeń w **Simulation**.

Metoda *respondToEvent()* klasy **Simulation** jest wywoływana, aby zareagować na każde zdarzenie. Może to obejmować aktualizację stanu magazynu, zamówienie nowego towaru, przesunięcie zasobów lub wysłanie powiadomień do administratorów.

Obiekty klasy **Warehouse**, które są częścią *listOfWarehouses* w **Simulation**, są aktualizowane w odpowiedzi na zdarzenia. Metody takie jak *checkStatus()* i *updateStatus()* są używane do monitorowania i modyfikacji stanu magazynu.

Produkty reprezentowane przez obiekty klasy **Product** są sprzedawane i zarządzane poprzez metody takie jak *sell(), updatePrice() i changeQuantity()*, które są wywoływane w odpowiedzi na zdarzenia sprzedaży.

Na koniec symulacji, metoda *generateReport()* klasy **Report** jest wywoływana, aby utworzyć raport z wynikami symulacji, takimi jak koszty operacyjne i zysk netto. Raporty mogą być szczegółowe dla magazynów **(WarehouseReport)** lub sprzedaży **(SalesReport)**.

Po zakończeniu określonej liczby cykli, symulacja kończy działanie, a użytkownik otrzymuje końcowy raport z wynikami.