Dokumentacja dla pozostałych klas

1. Klasa WorldMarket.cs

Klasa odpowiedzialna za wyznaczanie cen produktów na światowym rynku.

Elementy do uwzględnienia:

- publiczna statyczna metoda *int GetInitialPricePerKg( )*, która zwróci początkową cenę za 1t produktu w formie losowej liczby całkowitej z przedziału od 100 do 1000
- publiczna statyczna metoda *int GetNewPricePerKg(int)*, która zmodyfikuje cenę za 1t produktu na podstawie ceny dotychczasowej (podawanej jako argument). Modyfikacja powinna nastąpić o losową liczbę całkowitą z przedziału od -50 do 200, a metoda powinna zwrócić na koniec nową cenę.

W celu napisania powyższych metod proszę samodzielnie zapoznać się z oficjalną dokumentacją języka lub z innym wybranym przez siebie źródłem. Pytanie do rozważenia: czemu może służyć zadeklarowanie powyższych metod jako statycznych?

Dokumentacja: https://docs.microsoft.com/pl-pl/dotnet/api/system.random.next?view=netcore-3.1

2. Klasa Product.cs

Klasa odpowiedzialna za przechowywanie informacji o konkretnym produkcie.

Elementy do uwzględnienia:

- int *currentPrice* przechowujący aktualną cenę produktu
- string *Name* z nazwą produktu (możliwy do odczytania, ale nie do ustawiania)
- int *Mass* z masą produktu w tonach (możliwy do odczytania i ustawiania, jednak nigdy nie może być ujemny)
- konstruktor parametryczny pozwalający ustawić nazwę produktu i domyślnie ustawiający masę na zero
- konstruktor parametryczny pozwalający ustawić jednocześnie nazwę i masę produktu
- publiczna metoda *int GetCurrentValue( )*, która zwróci cenę sprzedaży zgromadzonego produktu, uwzględniając jego masę w tonach i aktualną cenę za jedną tonę podawaną przez klasę *WorldMarket.*

Klasa powinna korzystać z obydwu metod statycznych umieszczonych w klasie *WorldMarket* – pierwsza z nich powinna zostać wywołana przy tworzeniu nowego produktu, druga za każdym razem, gdy chcemy sprawdzić bieżącą cenę.

3. Klasa Ship.cs

Centralna klasa całego programu, która odpowiada za przechowywanie informacji o statku i jego działanie.

Elementy do uwzględnienia:

- obiekt klasy *Engine* reprezentujący silnik (możliwy do odczytania i ustawiania)
- prywatna zmienna int przechowująca masę niezaładowanego statku (bez produktów)
- konstruktor domyślny proponujący jakieś wartości dla powyższych zmiennych
- konstruktor parametryczny pozwalający ustawić powyższe zmienne (masa statku musi być dodatnia)
- publiczna metoda *bool TravelOffer(Destination, Product, Product)*, która wczyta ofertę podróży do wybranego portu docelowego (pierwszy argument) wraz z przewozem dwóch wybranych produktów (drugi i trzeci argument). Dla uproszczenia zakładamy, że statek zawsze będzie przewoził dokładnie dwa obiekty typu *Product.* Metoda powinna wykonać następujące czynności:
    - obliczyć całkowitą cenę sprzedaży proponowanego ładunku
    - obliczyć całkowity koszt podróży do proponowanego portu, korzystając z metody zawartej w dołączonej klasie *Engine*
    - zdecydować, czy oferta jest opłacalna – zakładamy, że podróż opłaci się tylko wtedy, gdy cena sprzedaży przewyższy koszty o co najmniej 1000
    - jeśli oferta jest opłacalna, wypisać na ekran czytelne informacje o akceptacji oferty, zawartości i masie ładunku, czasie podróży (pomocna może być metoda z klasy *Engine*), całkowitej cenie sprzedaży i koszcie podróży
    - jeśli oferta jest nieopłacalna, wystarczy wyświetlić informację o odrzuceniu oferty
    - w obydwu wypadkach należy zwrócić bool informujący o akceptacji (lub nie) oferty
- opcjonalnie także inne, prywatne metody – jeśli uznają Państwo, że mogą być przydatne

4. Klasa Destination.cs

Dokumentacja do tej klasy uległa zagubieniu – proszę ją skonstruować samodzielnie na podstawie pozostałych elementów programu (podpowiedź: jest to prosta klasa z dwiema właściwościami i konstruktorem parametrycznym)

5. Typ enumeracyjny Fuel

Dokumentacja do tego typu uległa zagubieniu – proszę go skonstruować samodzielnie na podstawie pozostałych elementów programu (podpowiedź: wystarczą trzy wartości)