Programowanie - Java

Jakub Mazurek
kuba@fenix.club

12 grudnia 2016

Zadanie dodatkowe

Twoim zadaniem jest napisanie programu, który zamodeluje nam tworzenie składów pociągów w odpowiedni sposób. Celem tego zadania jest zademonstrowanie praktycznego wykorzystania podstawowych struktur danych w Javie.
Na tym etapie nasz program będzie miał dwie główne funkcjonalności:

- Składanie pociągów poprzez łączenie ze sobą różnych rodzajów wagonów i wyświetlanie zestawienia w konsoli (ta część będzie wykorzystywała klasę `LinkedList`).
- Definiowanie wymagań i ograniczeń dla stworzonych przez nas składów i sprawdzanie istniejących składów pod tym kątem (w tej części wykorzystamy klasę `HashMap`).

Specyfikacja zadania

1. Każdy rodzaj wagonu w naszym programie będzie miał nadrzędną klasę abstrakcyjną `Car`.
   a) Stwórz w tej klasie metodę abstrakcyjną `displayCar`, nie przyjmującą żadnych argumentów i zwracającą `String`, który później wykorzystamy do graficznego przedstawienia naszego wagonu.
   b) Stwórz drugą metodę abstrakcyjną o nazwie `getCarType`. Ta metoda będzie nam służyła do określenia typu wagonu i przyda się później w wykorzystaniu map. `getCarType` nie przyjmuje żadnych argumentów i zwraca `String`.
2. Stwórzmy teraz kilka klas konkretnych wagonów, będących podklasami `Car`.
   a) Dodaj klasę abstrakcyjną `PassengerCar`, która będzie przedstawiać wagon pasażerski. Ta klasa będzie dziedziczyć po klasie `Car`.
      i. Do klasy `PassengerCar` dodaj pole przedstawiające liczbę miejsc dla pasażerów w jednym wagonie (pojemność). Stwórz odpowiednie gettery i settery.
   b) Dodaj klasę nieabstrakcyjną `RestaurantCar`, która będzie przedstawiać wagon restauracyjny. Niech ta klasa dziedziczy po `PassengerCar`.
      i. Zaimplementuj metodę `drawCar`. Może ona zwracać coś w stylu: `[__WARS__]`, liczę na Twoją inwencję twórczą!
      ii. Zaimplementuj metodę `getCarType` w odpowiedni sposób. Możesz dodać do tej klasy **stałą** (słowo kluczowe `final`) typu `String`, której wartością będzie typ tego wagonu. W `getCarType` zwróć wartość tej stałej.
   c) Stwórz dwie nieabstrakcyjne klasy dziedziczące po `PassengerCar`. Zaimplementuj je analogicznie do poprzednich klas.
      i. Klasa `CompartmentCar` będzie przedstawiać wagony przedziałowe.
      ii. Klasa `NonCompartmentCar` będzie przedstawiać wagony bezprzedziałowe.
   d) Stwórz nieabstrakcyjną klasę `Locomotive`, dziedziczącą po klasie `Car`.
      i. Zaimplementuj odpowiednio metody abstrakcyjne z nadklasy.
      ii. Dodaj atrybuty przedstawiające moc i zasięg lokomotywy.

3. Mając dodane klasy konkretnych wagonów możemy je teraz ze sobą łączyć w składy.
   a) Stwórz w swoim projekcie nieabstrakcyjną klasę `Train`. Jej obiekty będą przedstawiały składy pociągów.
   b) Każdy obiekt typu `Train` powinien zawierać w sobie **listę łączoną** wagonów (`LinkedList`). Zakładamy, że początek listy przedstawia początek całego pociągu.
   c) Do klasy `Train` dodaj teraz kilka potrzebnych metod:
      i. `addCar`, która doda wagon podany jako argument na koniec pociągu.
      ii. `removeCar`, która usunie dany wagon z pociągu. Jako argument możemy podać albo pozycję obiektu do usunięcia (`int`) albo odniesienie do samego obiektu (`Car`).
      iii. `swapCars`, która zamieni miejscami dwa wagony. Metoda ta powinna przyjmować dwie zmienne typu `int`, wyznaczające pozycje odpowiednich wagonów na liście.
      iv. `displayTrain`, która zwróci `String` przedstawiający graficznie wszystkie wagony wchodzące w skład pociągu (dla wagonów skorzystaj z metody `displayCar`). Możesz dodać tutaj rodzaj graficznego łącznika pomiędzy wagonami.
      v. Dodaj metodę, dzięki której możliwy będzie **niebezpośredni** dostęp do wagonów pociągu w **odpowiedniej kolejności**. Oznacza to, że nie chcemy zwracać tutaj bezpośrednio odwołania do wewnętrznej listy łączonej. Istnieje wiele możliwych sposobów, np. zwracanie tablicy elementów lub iteratora listy.

4. Stwórzmy teraz klasę o nazwie `TrainValidator`. Każdy obiekt tej klasy będzie mógł zawierać zbiór restrykcji dotyczących składu pociągu (np. pociąg powinien mieć dokładnie jedną lokomotywę i co najmniej jeden wagon pasażerski).
   a) Dodaj do klasy `TrainValidator` pole będące mapą, w której klucze będą typu `String`, a wartości będą parami liczb całkowitych. Ta mapa będzie przedstawiała ograniczenia - klucz będzie typem wagonu, którego dotyczy ograniczenie, a pary `int`ów będą minimalną i maksymalną liczbą wagonów.
      i. W celu przedstawienia par liczb możesz stworzyć własną klasę `Pair` albo skorzystać z gotowych rozwiązań (np. z paczki: `javafx.util.Pair`).
   b) Dodaj do klasy `TrainValidator` metodę o nazwie `addRestriction`, która doda nowe ograniczenie do mapy. Metoda ta powinna przyjmować trzy argumenty: typ wagonu, minimalną liczbę wagonów oraz maksymalną liczbę wagonów.
   c) Stwórz metodę `getTrainComposition`, która przyjmie jako argument obiekt typu `Train` i zwróci mapę zawierającą podliczenie wszystkich wagonów w tym pociągu. Mapa ta powinna mieć klucze typu `String`, zawierające rodzaje wagonów oraz wartości typu `Integer`, które będą mówiły ile jest w pociągu wagonów danego typu.
   d) Stwórz metodę `validate`, która korzystając z poprzednio dodanej `getTrainComposition` zweryfikuje pociąg podany jako argument. W tej metodzie wykorzystaj mapę zawierającą ograniczenia dla tego pociągu.

i. Jako wartość zwracaną przez tę metodę możesz wykorzystać albo zmienną logiczną `boolean` (wtedy będziemy jedynie wiedzieć, czy dany pociąg spełnia kryteria czy też nie), albo zbiór niespełnionych kryteriów.

5. Przetestuj swoje rozwiązanie w metodzie `main`.
   a) Stwórz kilka różnych pociągów o różnym zestawieniu. Wyświetl ich zestawienie w konsoli.
   b) Stwórz dwa różne walidatory, których zestawy ograniczeń będą się różniły (np. jeden z nich może sprawdzać, czy pociąg ma wagon restauracyjny oraz przynajmniej dwa wagony bezprzedziałowe).
   c) Sprawdź stworzone przez Ciebie składy za pomocą walidatorów. Wyświetl wyniki testów w konsoli.

Klasy w Twoim projekcie powinny być zorganizowanie w sensowne paczki (np. klasa z metodą `main` powinna znaleźć się w paczce o nazwie `program`).

Dla skrócenia opisu zadania pomijam dodawanie odpowiednich konstruktorów, getterów i setterów.

Swoje rozwiązania możecie wrzucić na GitHuba albo wysłać mi jako spakowany projekt. W razie pytań piszcie do mnie na Slacku (`mazurekk`) albo mailowo (`kuba@fenix.club`).