Projektowanie obiektowe

Laboratorium 3
Wzorce projektowe

Kwiecień 2020

1 Opis

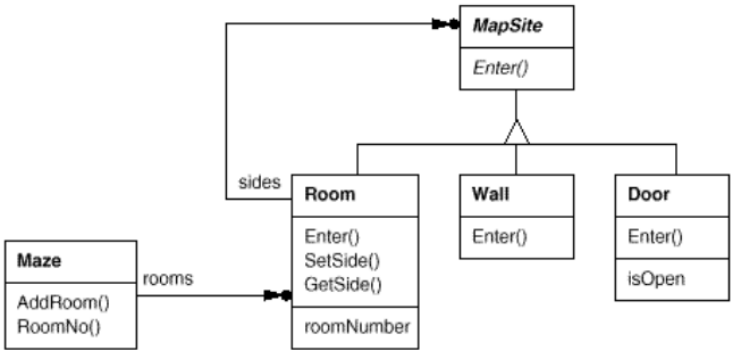
Celem laboratorium jest zapoznanie się z podstawowymi wzorcami projektowymi.

2 Przygotowanie do zajęć

Pobierz szkielet kodu, który jest dostarczony do laboratorium. Uruchom projekt w środowisku IntelliJ. Zapoznaj się z prezentacją dostarczoną jako materiał do laboratorium. Znajduje się tam opis kolejnych wzorców jakie będą używane w trakcie laboratorium.
Dostarczony kod można pod każdym kątem modyfikować pod siebie! To jest tylko bardzo prosty szkielet zależności. Co ważne - w razie modyfikacji proszę pisać dlaczego zostały dokonane i w czym to pomogło.

3 Wzorce konstrukcyjne

W kolejnych zadaniach wykorzystamy bardzo prosty przykład tworzenia labiryntu na potrzeby gry komputerowej. Pomijamy wiele szczegółów i głównie skupiamy się na akcji tworzenia labiryntu, które definiujemy jako zbiór pomieszczeń. Klasy Room, Door i Wall reprezentują komponenty labiryntu używane we wszystkich przykładach. Definiujemy tylko fragmenty tych klas potrzebne do utworzenia labiryntu. Na starcie ignorujemy gracza, wyświetlanie i poruszanie się po nim.



Rysunek 1: Diagram ilustrujący relacje pomiędzy wspomnianymi wyżej klasami.

4 Zadania

4.1 Builder

Zdefiniuj nową wersję funkcji składowej createMaze, która będzie przyjmować jako argument obiekt budujący klasy MazeBuilder.

1. Stwórz klasę MazeBuilder, która definiuje interfejs służący do tworzenia labiryntów. Co musi tam być zawarte? Wykorzystaj wiedzę nt. składowych, które są w labiryncie.

2. Po utworzeniu powyższego interfejsu zmodyfikuj funkcje składową tak, aby przyjmowała jako parametr obiekt tej klasy.

3. Prześledź i zinterpretuj co dały obecne zmiany (krótko opisz swoje spostrzeżenia).

4. Stwórz klasę StandardBuilderMaze będącą implementacją MazeBuildera. Powinna ona mieć zmienną currentMaze, w której jest zapisywany obecny stan labiryntu. Powinniśmy móc: tworzyć pomieszczenie i ściany w okół niego, tworzyć drzwi pomiędzy pomieszczeniami (czyli musimy wyszukać odpowiednie pokoje oraz ścianę, która je łączy). Dodaj tam dodatkowo metodę prywatną CommonWall, która określi kierunek standardowej ściany pomiędzy dwoma pomieszczeniami.

5. Utwórz labirynt przy pomocy operacji createMaze, gdzie parametrem będzie obiekt klasy StandardMazeBuilder.

6. Stwórz kolejną podklasę MazeBuildera o nazwie CountingMazeBuilder. Budowniczy tego obiektu w ogóle nie tworzy labiryntu, a jedynie zlicza utworzone komponenty różnych rodzajów. Powinien mieć metodę GetCounts, która zwraca ilość elementów.

4.2 Fabryka abstrakcyjna

1. Stwórz klasę MazeFactory, która służy do tworzenia elementów labiryntu. Można jej użyć w programie, który np. wczytuje labirynt z pliku .txt , czy generuje labirynt w sposób losowy.

2. Przeprowadź kolejną modyfikacje funkcji createMaze tak, aby jako parametr brała MazeFactory.

3. Stwórz klasę EnchantedMazeFactory (fabryka magicznych labiryntów), która dziedziczy z MazeFactory. Powinna przesłaniać kilka funkcji składowych i zwracać różne podklasy klas Room, Wall itd. (należy takie klasy również stworzyć).

4. Stwórz klasę BombedMazeFactory, która zapewnia, że ściany to obiekty klasy BombedWall, a pomieszczenia to obiekty klasy BombedRoom (teoretycznie wystarczy przesłonić jedynie 2 metody - MakeWall(...) / MakeRoom(...)).

4.3 Singleton

Wprowadź w powyżej stworzonej implementacji mechanizm, w którym MazeFactory będzie Singletonem. Powinien być on dostępny z pozycji kodu, który jest odpowiedzialny z tworzenie poszczególnych części labiryntu.

4.4 Rozszerzenie aplikacji labirynt

a) Korzystając z powyższych implementacji dodaj prosty mechanizm przemieszczania się po labiryncie. Po realizacji wcześniejszych zadań pozostaje stworzyć prostą klasę Player, która za pomocą np. strzałek + tekstu w konsoli będzie mogła zadecydować o kierunku chodzenia. Rozpatrz stosowne warianty rozgrywki (czy ściana ma drzwi przez które możemy przejść itp. itd.). Wprowadź elementy BombedRoom/BombedWall (rozwiązanie co się wtedy stanie zostawiam twórcy. Może być timer, który po 15s bez decyzji zabija gracza etc.).
b) Zademonstruj, że MazeFactory faktycznie jest Singletonem (najłatwiej stworzyć przykład, w którym się sprawdza, czy obiekt zwracany przy 2 konstrukcji to faktycznie ten sam, który został stworzony na początku).

4.5 Dla chętnych!

Dodanie prostej wizualizacji wykorzystując stosowne biblioteki Javy (JavaFX/Swing).