Programowanie obiektowe - kolokwium 1, 26.04.2024

Podczas kolokwium wszystkie działania wykonywane na ekranie komputera są rejestrowane i mogą zostać odtworzone przez prowadzących. Po napisaniu kolokwium, katalog z projektem należy spakować do formatu zip lub tar.gz i wysłać go przy użyciu kampusu. Proszę nie wyłączać komputera.
Podczas kolokwium można korzystać ze wszystkich zasobów internetu, w tym z samodzielnie przygotowanych materiałów z wyłączeniem stron, aplikacji i pluginów opartych o sztuczną inteligencję. Podczas kolokwium zabroniona jest wzajemna komunikacja oraz publikowanie i dostęp do kodu stworzonego na potrzeby kolokwium. Zabronione jest korzystanie z innych urządzeń niż udostępniony komputer.
Przedstawione poniżej kroki stanowią propozycję kolejności rozwiązywania zadania. Po przeczytaniu całości można zdecydować o rozwiązywaniu w innej kolejności.

Krok 1.
Napisz klasę abstrakcyjną *Clock* przechowującą wewnętrznie stan zegara, posiadającą publiczne metody:

- *setCurrentTime*, która ustawia czas zegara na bieżącą godzinę zgodnie z zegarem systemowym.
- *setTime*, która przyjmuje trzy zmienne całkowite - godzinę, minutę i sekundę i ustawia zgodnie z nią czas zegara. Jeżeli dane nie są poprawne w kontekście zegara 24-godzinnego, należy rzucić wyjątek *IllegalArgumentException* i opisać przyczynę w wiadomości tego wyjątku (która ze zmiennych nie mieści się w jakim zakresie).
- *toString*, która zwraca napis zawierający godzinę w formacie hh:mm:ss.

Sposób przechowywania czasu w zegarze zaproponuj samodzielnie.

Krok 2.
Napisz klasę *DigitalClock*, dziedziczącą po *Clock*. W klasie *DigitalClock* utwórz publiczny typ wyliczeniowy pozwalający na rozróżnienie między zegarem 24-godzinnym i 12-godzinnym. Dodaj argument tego typu do konstruktora.
Jeżeli ustawiony jest tryb 24-godzinny, metoda *toString* powinna wywołać metodę *toString* klasy nadrzędnej. W trybie 12-godzinnym napis powinien ograniczać się do 12 godzin z dopiskiem AM lub PM i nie poprzedzać jednocyfrowej godziny zerem. Utwórz obiekt klasy DigitalClock i przetestuj przypadki z sąsiedniej tabeli.

Zapoznaj się z plikiem strefy.csv dołączonym do treści zadania. Plik zawiera letnie strefy czasowe wybranych miast oraz ich współrzędne geograficzne wyrażone stopniami kątowymi.

12 godzin	24 godziny
12:00:00 AM	00:00:00
12:01:00 AM	00:01:00
1:00:00 AM	01:00:00
11:59:00 AM	11:59:00
12:00:00 PM	12:00:00
1:00:00 PM	13:00:00
11:59:00 PM	23:59:00

Krok 3.
Napisz klasę *City* zawierającą wszystkie dane z pojedynczego wiersza pliku. Napisz w niej metody statyczne:

- prywatną *parseLine*, przyjmującą pojedynczą linię pliku i zwracającą obiekt *City*,
- publiczną *parseFile*, przyjmującą ścieżkę do tego pliku i zwracającą mapę zapełnioną danymi z pliku, w której kluczem jest nazwa miasta, a wartością obiekt City.

W razie potrzeby, do każdej z danych wczytanych do obiektu *City* można stworzyć publiczny akcesor. Wywołaj metodę *parseFile* przekazując jej ścieżkę do przykładowego pliku.

Krok 4.
W klasie *Clock* dodaj prywatne pole *City*. Zmodyfikuj konstruktor tak, aby przyjmował referencję na obiekt *City* i ustawiał ją w tym polu. W klasie *Clock* napisz publiczną metodę:

- *setCity*, która przyjmie referencję na obiekt *City* i zastąpi w obiekcie *Clock* dotychczasową referencję na *City*.

Zmiana miasta powinna spowodować zmianę wskazywanej godziny (np. zmiana Warszawy na Kijów powinna zwiększyć godzinę o 1 lub ewentualnie z 23 na 0). Wywołaj metodę *setCity*.

Krok 5.
W klasie *City* napisz publiczną metodę:

- *localMeanTime*, która przyjmie czas zgodny ze swoją strefą czasową i zwróci czas miejscowy obliczony na podstawie długości geograficznej.

Długość geograficzna przyjmuje wartości od -180 do +180 stopni, a odpowiadające jej przesunięcie godzinowe wartości od -12 do +12 godzin. Przesunięcie czasu zmienia się wprost proporcjonalnie do długości geograficznej.
Na przykład, jeżeli w Lublinie, według strefy czasowej, jest godzina 12:00:00, według położenia geograficznego, będzie to 11:30:16. Wywołaj metodę *localMeanTime*.

Krok 6.
W klasie *City* napisz publiczną statyczną metodę:

- *worstTimezoneFit*, tak, aby mogła być użyta jako komparator dwóch miast.

Wynik sortowania przy użyciu tej metody powinien ustawić na początku kolekcji miasta, których różnica między czasem miejscowym a czasem wynikającym ze strefy czasowej jest największa. Wykonaj sortowanie listy obiektów *City* przy użyciu tej metody i wyświetl nazwy miast z posortowanej listy. Poprawna kolejność dla danych z pliku to:
Madryt, Paryż, Amsterdam, Ateny, Ryga, …, Lima, Rio de Janeiro, Sao Paulo, Kair, Sydney, Osaka.

Zapoznaj się z plikiem zegar.svg dołączonym do treści zadania.

Krok 7.
Napisz klasę *AnalogClock*, dziedziczącą po *Clock*. Napisz w niej metodę:

- *toSvg*, która przyjmie ścieżkę do pliku i zapisze w nim kod SVG przedstawiający tarczę zegara (bez wskazówek).

Krok 8.
Napisz klasę abstrakcyjną *ClockHand* reprezentującą wskazówkę zegara tarczowego. Klasa powinna posiadać publiczne, abstrakcyjne metody:

- *setTime*, która przyjmuje obiekt *LocalTime*,
- *toSvg*, która zwraca napis zawierający znacznik SVG wskazówki.

Krok 9.
Po klasie *ClockHand* podziedzicz klasę *SecondHand* - wskazówkę sekundową. Powinna implementować metody klasy nadrzędnej:

- *setTime*, która ustawia kąt wskazówki dyskretnie (skokowo), na podstawie składowej sekund przekazanego czasu,
- *toSvg*, która zwraca znacznik SVG przedstawiający wskazówkę jako odcinek o określonej długości, grubości i kolorze, obróconą odpowiednią liczbę stopni, wynikającą z ustawionego metodą *setTime* czasu.

Kąt w SVG jest reprezentowany za pomocą stopni kątowych, zgodnie z ruchem wskazówek zegara.

Krok 10.
Po klasie *ClockHand* podziedzicz jeszcze dwie klasy: *HourHand*, *MinuteHand* implementujące metody klasy nadrzędnej:

- *setTime*, która ustawia kąt wskazówki na podstawie czasu. Wskazówki powinny poruszać się ruchem ciągłym z dokładnością do jednej sekundy,
- *toSvg*, analogicznie jak w klasie *SecondHand* tak, aby wskazówki wizualnie odróżniały się od siebie.

Krok 11.
W klasie *AnalogClock* umieść prywatną, polimorficzną, finalną listę wskazówek, zawierającą po jednej wskazówce każdej klasy. Zmodyfikuj metodę *toSvg*, aby oprócz tarczy, narysowane zostały także wskazówki. Niech ustawienie godziny zegara metodami z kroku 1. (*setCurrentTime, setTime*) powoduje zmianę położenia wskazówek. Zaproponuj rozwiązanie, które nie będzie wymagało nadpisywania (overriding) tych dwóch metod.

Krok 12.
W klasie *City* napisz publiczną metodę statyczną *generateAnalogClocksSvg*, która przyjmie listę obiektów *City* oraz obiekt *AnalogClock*. Metoda powinna założyć katalog o nazwie będącej wynikiem funkcji *toString* obiektu zegara. W katalogu, dla każdego miasta z listy, należy utworzyć plik SVG o nazwie odpowiadającej nazwie miasta. Pliki powinny zawierać wynik działania metody *toSvg* zegara w kolejnych miastach. Wywołaj tę funkcję.