Przykładowe kolokwium #2 - Zestaw Z14

Ostatnia aktualizacja pliku: 10.01.2024 21:58.

Imię i nazwisko, numer albumu .....

Informacje wstępne

- Łącznie do zdobycia max **60** punktów. Próg zaliczenia: 25 pkt (bez innych punktów).
- **Kolokwium należy wykonać na komputerach zamontowanych na stałe w pracowniach.**
- Student przesyłając rozwiązania oświadcza, że rozwiązał je samodzielnie.
- W trakcie kolokwium nie można korzystać z żadnych materiałów pomocniczych w żadnej formie. Wszelkie kody powinny być napisane manualnie bez wspomagania się dodatkami automatycznie generującymi kod (np. Copilot, chat GPT itp.).
- Publikowanie poleceń i rozwiązań w internecie jest zabronione do czasu napisania kolokwium przez wszystkie grupy ćw.
- Należy zwracać uwagę na właściwe umieszczenie kodu (luzem lub w pakiecie).
- Kod musi się kompilować, aby był sprawdzany.
- Należy oddzielać klasę z definicjami od klasy testującej (z main) zgodnie z poleceniami.
- Jeśli w poleceniu nie jest podany typ zmiennej, można go wybrać dowolnie.
- Jeśli w danej metodzie nie ma sprecyzowanej „walidacji", to można ją pominąć.
- Metody nie powinny wykonywać nadmiarowych, nielogicznych czynności.
- Poza zmiennymi/polami w klasie wymienionym w polecaniach zabronione jest tworzenie innych pól w klasie. Stworzenie dodatkowych metod jest dopuszczalne, ale nie należy tego nadużywać.
- Jeśli w poleceniu nie są sprecyzowane modyfikatory dostępu, należy dostępować zgodnie z zasadami hermetyzacji.
- **W rozwiązaniach należy uwzględniać dobre praktyki omawiane na wykładzie i ćwiczeniach, o ile polecenie nie mówi coś innego.**
- Rozwiązania (projekt z IntelliJ) należy w całości spakować jako archiwum zip. Następnie ustawić nazwę. Rozwiązania należy umieścić na pendrive przekazanym przez prowadzącego kolokwium.
- **Nazwa archiwum powinna być wg schematu NUMERZESTAWU_NUMERALBUMU.zip gdzie numer zestawu znajduje się na górze kartki z poleceniami. np. A23_123456.zip.**
- Archiwum powinno być bez hasła.
- Kod zakomentowany nie będzie sprawdzany.
- Zawartość pendrive będzie pusta. Udostępniony będzie tylko w celu zgrania rozwiązań. Umieszczenie poleceń na pendrive powinno odbyć się w czasie kolokwium. Rozwiązania po czasie mogą nie być sprawdzane.
- Jeśli w poleceniu pojawia się informacja o konieczności zachowania formatowania napisów (np. wielkość znaków, znaki interpunkcyjne), to należy to bezwzględnie wykonać.
- Podpunkty będą oceniane kaskadowo — wykonanie ich bez wykonania wcześniejszych podpunktów może oznaczać zero punktów.
- O ile nie zaznaczono w poleceniu inaczej, każdą z metod należy wywołać co najmniej jeden raz (może być bardzo trywialnie). Warto zwrócić uwagę, że samo tworzenie obiektów w każdym zdefiniowanym samodzielnie typie nie jest wymagane (chyba że polecenie tego wymaga).
- Należy zachowywać kolejność argumentów w konstruktorach i metodach. Należy dążyć do tego, że nazwy argumentów metod powinny pokrywać się z nazwami pól w klasie, gdzie to ma sens.
- Warto zwracać uwagę na typ zwracany metod — jeśli metoda ma „coś" zwrócić, będzie to wskazane w poleceniu.
- Po kartkach z poleceniami można pisać i traktować jako brudnopis.

Zadanie 1. (15pkt max.)

A. Utwórz finalną klasę `ImmutableDate` w pakiecie `date`, która powinna zawierać trzy prywatne i finalne pola:

- `year`: typu `int`, reprezentującego rok.
- `month`: typu `int`, reprezentującego miesiąc.
- `day`: typu `int`, reprezentującego dzień.

B. Dodaj do klasy `ImmutableDate`:

- Konstruktor parametryczny do inicjalizacji wszystkich pól (`year`, `month`, `day`).
- Publiczne metody dostępowe `getYear`, `getMonth`, `getDay` do pobierania wartości pól. Upewnij się, że klasy nie zawierają metod umożliwiających modyfikację tych pól.
- Zaimplementuj metody `toString`, `equals` i `hashCode`:
  - `toString`: powinna zwracać reprezentację daty w formacie "YYYY-MM-DD".
  - `equals`: powinna porównywać obiekty `ImmutableDate` na podstawie wartości pól `year`, `month` i `day`.
  - `hashCode`: powinna generować wartość hash na podstawie pól `year`, `month` i `day`.

C. W pakiecie `date`, utwórz klasę testową `TestImmutableDate` z metodą `main`, w której:

- Utwórz dwa obiekty klasy `ImmutableDate` z różnymi datami.
- Wyświetl te obiekty, używając metody `toString`.
- Przetestuj porównywanie obiektów za pomocą metody `equals`.
- Wyświetl wartości hash obiektów, używając metody `hashCode`.

Zadanie 2. (15pkt max.)

A. Utwórz rekord `StudentRecord` w pakiecie `student`, który powinien zawierać trzy pola:

- `name`: typu `String`, reprezentującego imię studenta.
- `id`: typu `String`, reprezentującego identyfikator studenta.
- `gpa`: typu `double`, reprezentującego średnią ocen studenta.

B. Dodaj do rekordu `StudentRecord`:

- Kompaktowy konstruktor, który weryfikuje, czy średnia ocen (`gpa`) jest w przedziale od 0.0 do 4.0. Jeśli `gpa` jest poza tym zakresem, konstruktor powinien rzucać wyjątek `IllegalArgumentException`.
- Metodę `isHonorStudent`, która zwraca `true`, jeśli średnia ocen (`gpa`) studenta jest wyższa lub równa 3.5, i `false` w przeciwnym przypadku.
- Metodę `printDetails`, która wyświetla informacje o studencie, włączając w to jego imię, identyfikator i średnią ocen.

C. W pakiecie `student`, utwórz klasę testową `TestStudentRecord` z metodą `main`, w której:

- Utwórz dwa obiekty typu `StudentRecord` z różnymi danymi.
- Wywołaj metodę `printDetails` na każdym z obiektów, aby wyświetlić ich szczegóły.
- Sprawdź, którzy studenci są studentami z wyróżnieniem, używając metody `isHonorStudent`.

Zadanie 3. (15pkt max.)

- Poniższe czynności wykonaj w pakiecie `create`.

- Napisz statyczną metodę generyczną `createArray`, która akceptuje dwa argumenty typu generycznego `T` oraz dwuelementową tablicę tego samego typu. Zadaniem metody jest zapisanie dwóch pierwszych argumentów kolejno do tablicy będącej trzecim argumentem. Upewnij się, że metoda weryfikuje, czy tablica ma dokładnie dwa elementy. Stwórz przypadek testowy.

Zadanie 4. (15pkt max.)

- Poniższe czynności wykonaj w pakiecie `algorithm`.
- Napisz statyczną metodę generyczną `mergeMaps`, która przyjmuje jako argumenty dwie mapy typu `HashMap<K, V>`. Metoda ma zwrócić nową mapę, która jest połączeniem obu map wejściowych. W przypadku wystąpienia tego samego klucza w obu mapach, wartością w mapie wynikowej powinna być wartość z drugiej mapy.