Przykładowe kolokwium #2 - Zestaw Z12

Ostatnia aktualizacja pliku: 07.01.2024 22:35.

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. Wykonaj poniższe czynności:

- Stwórz klasę `Athlete`, która powinna być częścią odpowiedniego pakietu `sports`.
- Klasa `Athlete` powinna posiadać dwa pola:
  - `name`: typu `String`, reprezentującego nazwę sportowca.
  - `times`: tablica pięciu zmiennych typu `double`, reprezentująca czasy (w sekundach) potrzebne na przebiegnięcie 100 metrów podczas różnych prób.
- Zaimplementuj w klasie `Athlete` interfejs `Cloneable`.
- Nadpisz metodę `clone` z interfejsu `Cloneable`, aby umożliwić klonowanie obiektów klasy `Athlete`.
- W zadaniu uwzględnij głębokie kopiowanie dla pola będącego tablicą.

B. Wykonaj poniższe czynności:

- Napisz metodę `main` w klasie `TestAthlete` w tym samym pakiecie, a w niej
  - Utwórz obiekt klasy `Athlete`.
  - Sklonuj utworzony obiekt `Athlete`.
  - Zmień czas na pozycji trzeciej w tablicy `times` oryginalnego sportowca (stwórz w tym celu odpowiednią metodę).
  - Wyświetl czasy obu sportowców (oryginału i jego klona), aby sprawdzić, czy zmiany w jednym obiekcie nie wpływają na drugi, świadcząc o ich niezależności.

Zadanie 2. (15pkt max.)

- Poniższe czynności wykonaj w pakiecie vacation‘.
- Stwórz interfejs `TravelPlanner` z następującymi elementami:
  - Metodą abstrakcyjną `void planRoute(String destination)`.
  - Metodą domyślną `cancelTrip()`, która wyświetla informację “Trip Cancelled”.
  - Metodą statyczną `getTravelMode()` zwracającą `String` “Travel Mode”.
- Stwórz klasy `RoadTripPlanner` i `FlightPlanner`, które implementują `TravelPlanner`. Metoda `planRoute(String destination)` w `RoadTripPlanner` powinna wyświetlać `"Planning Road Trip to [destination]"`, a w `FlightPlanner` - `"Planning Flight to [destination]"`.
- Stwórz klasę testującą `TravelTester`. Utwórz obiekty `RoadTripPlanner` i `FlightPlanner`, wywołaj dla nich `planRoute(String destination)` i `cancelTrip()`, oraz wywołaj statycznie `TravelPlanner.getTravelMode()`.

Zadanie 3. (15pkt max.)

- Poniższe czynności wykonaj w pakiecie `utilities`.
- Napisz statyczną metodę generyczną `appendFromEnd`.
- Metoda `appendFromEnd` dopisuje wszystkie elementy z jednej tablicy typu ArrayList do drugiej od końca.
- W implementacji tej metody należy użyć symbolu wieloznacznego `<? super E>`.
- Przetestuj działanie metody na dowolnym przykładzie.

Zadanie 4. (15pkt max.)

- Poniższe czynności wykonaj w pakiecie `pair`.

- Utwórz klasę generyczną `Pair`, z dwoma typami generycznymi `K` i `V` reprezentującymi klucz i wartość pary. Klasa powinna mieć dwa pola odpowiadające tym typom. Dodaj konstruktor inicjujący oba elementy oraz metody `getFirst()`, która zwraca klucz, i `getSecond()`, która zwraca wartość.