Kolokwium #2 - Programowanie obiektowe - Zestaw A24

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. Klasa `BusStation` w pakiecie `transport` z prywatnymi polami:

- `name`: typu `String`, reprezentujący nazwę dworca autobusowego.
- `city`: typu `String`, reprezentujący miasto, w którym znajduje się dworzec.
- `buses`: typu `ArrayList<String>`, lista przechowująca nazwy autobusów.

B. Metody w klasie `BusStation`:

- Metoda `addBus(String bus)`: dodaje autobus do listy `buses`.
- Metoda `removeBus(String bus)`: usuwa autobus z listy `buses`.
- Konstruktory, gettery, settery, `toString()`, `equals()` i `hashCode()`.
- Pamiętaj o odpowiedniej kopii dla pola będącego listą tablicową.

C. Klasa `IntercityBusStation`, dziedzicząca po `BusStation` w tym samym pakiecie, z dodatkowym prywatnym polem `numberOfPlatforms`: typu `int`, reprezentujący liczbę peronów na dworcu.

D. Metody w klasie `IntercityBusStation`:

- Konstruktory, gettery i settery dla `numberOfPlatforms`.
- Nadpisane metody `toString()`, `equals()` i `hashCode()`.

E. Napisz klasę testującą `TestBusStation` w tym samym pakiecie:

- W metodzie `main` utwórz obiekty klasy `BusStation` i `IntercityBusStation`.
- Testuj działanie metod dodawania i usuwania autobusów.
- Wyświetl informacje o obu dworcach, aby sprawdzić poprawność działania metod.

Zadanie 2. (15pkt max.)

- Wykonaj czynności w pakiecie `farm`.
- Napisz klasę `Farmer`, która zawiera pola: `name` (typu `String`), `yieldPerAcre` (typu `double`) i `yearOfBirth` (typu `int`). Zaimplementuj interfejs `Comparable` w taki sposób, aby obiekty klasy `Farmer` były sortowane malejąco według wydajności upraw na akr (`yieldPerAcre`). Stwórz tablicę 5 obiektów klasy `Farmer` i posortuj ją według sprecyzowanego kryterium.

Zadanie 3. (15pkt max.)

- Wykonaj poniższe czynności w pakiecie `house`.
- Zdefiniuj klasy `Furniture` i `Chair`, gdzie `Chair` dziedziczy po `Furniture`. Napisz statyczną metodę generyczną `areSameFurnitureType`, która przyjmuje dwa argumenty: `furniture1` i `furniture2` typu `extends Furniture`. Metoda ma zwracać wartość `true`, jeśli obiekty są tego samego typu (np. oba są krzesłami), w przeciwnym wypadku `false`. Użyj metody `getClass()` do porównania klas obiektów. Stwórz przypadek testowy.

Zadanie 4. (15pkt max.)

W pakiecie `algorithm`, zaimplementuj statyczną metodę `mapToString(HashMap<K, V> map)`, która zwraca `String` reprezentujący wszystkie pary klucz-wartość w podanej mapie w formacie `"klucz: wartość"`. Każda para powinna być oddzielona przecinkiem i spacją. Metoda ta powinna być odpowiednia dla map przechowujących dowolny typ kluczy i wartości. Stwórz przypadek testowy.