Kolokwium #2 - Programowanie obiektowe - Zestaw A14

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 `BusDepot` w pakiecie `transport` z prywatnymi polami:

- `name`: typu `String`, reprezentujący nazwę zajezdni autobusowej.
- `city`: typu `String`, reprezentujący miasto, w którym znajduje się zajezdnia.
- `buses`: typu `ArrayList<String>`, lista przechowująca numery rejestracyjne autobusów.

B. Metody w klasie `BusDepot`:

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

C. Klasa `CityBusDepot`, dziedzicząca po `BusDepot` w tym samym pakiecie, z dodatkowym prywatnym polem `numberOfRoutes`: typu `int`, reprezentujący liczbę tras obsługiwanych przez zajezdnię.

D. Metody w klasie `CityBusDepot`:

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

E. Napisz klasę testującą Klasa `TestBusDepot` w tym samym pakiecie:

- W metodzie `main` utwórz obiekty klasy `BusDepot` i `CityBusDepot`.
- Testuj działanie metod dodawania i usuwania numerów rejestracyjnych autobusów.
- Wyświetl informacje o obu zajezdniach, aby sprawdzić poprawność działania metod.

Zadanie 2. (15pkt max.)

- Poniższe czynności wykonaj w pakiecie `shopping`.
- Napisz klasę `Product`, która zawiera pola: `name` (typu `String`): reprezentuje nazwę produktu; `category` (typu `String`): reprezentuje kategorię produktu; `price` (typu `double`): reprezentuje cenę produktu. Zaimplementuj generyczny interfejs `Comparable` w taki sposób, aby obiekty klasy `Product` były porównywane rosnąco według ceny. Stwórz listę tablicową (`ArrayList`) 4 obiektów klasy `Product` i posortuj ją według sprecyzowanego kryterium.

Zadanie 3. (15pkt max.)

- Poniższe czynności wykonaj w pakiecie `checking`.
- Utwórz statyczną metodę generyczną `isEitherNull`. Metoda ta przyjmuje dwa argumenty tego samego typu generycznego `T` i zwraca `true`, jeśli przynajmniej jeden z argumentów jest `null`. Metoda ma sprawdzać, czy którykolwiek z argumentów nie jest zainicjalizowany. Na przykład, `isEitherNull(object1, object2)` zwróci `true`, jeśli `object1` lub `object2` (lub oba) są `null`. Stwórz przypadek testowy dla tej metody.

Zadanie 4. (15pkt max.)

- Poniższe czynności wykonaj w pakiecie `algorithm`.
- Stwórz statyczną metodę generyczną `clearIfContains`, która przyjmuje kolekcję elementów typu `T` oraz pojedynczy element tego samego typu. Metoda powinna sprawdzać, czy dany element znajduje się w kolekcji. Jeśli tak, to cała kolekcja powinna zostać wyczyszczona. Metoda modyfikuje istniejącą kolekcję, a nie tworzy nową. Dodaj zabezpieczenie, aby metoda nie mogła być wywołana z kolekcją o wartości `null`. Zaimplementuj przypadek testowy dla tej metody.