Kolokwium #2 - Programowanie obiektowe - Zestaw A13

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. Stwórz klasę `Vehicle` w pakiecie `transport`, która powinna zawierać trzy pola:

- `model`: typu `String`, reprezentującego model pojazdu.
- `type`: typu `String`, reprezentującego typ pojazdu (np. samochód, motocykl).
- `range`: typu `int`, reprezentującego maksymalny zasięg pojazdu na pełnym zbiorniku paliwa w kilometrach.
- Zaimplementuj dwie klasy, które implementują generyczny interfejs `Comparator<Vehicle>`:
  - `RangeComparator`: porównuje obiekty klasy `Vehicle` według zasięgu (`range`), od najmniejszego do największego zasięgu.
  - `TypeModelComparator`: porównuje obiekty klasy `Vehicle` najpierw według typu (`type`), a w przypadku równości - według modelu (`model`). W obu przypadkach porządek powinien być leksykograficzny.

B. W klasie `TestVehicle` w tym samym pakiecie w metodzie `main`:

- Utwórz i posortuj tablicę obiektów `Vehicle` najpierw według zasięgu (używając `RangeComparator`).
- W przypadku, gdy dwa pojazdy mają ten sam zasięg, zastosuj `TypeModelComparator`, aby ustalić kolejność.
- Po zakończeniu sortowania wyświetl posortowaną listę, aby sprawdzić, czy sortowanie przebiegło prawidłowo i czy kolejność pojazdów jest zgodna z założeniami.

Zadanie 2. (15pkt max.)

A. Utwórz rekord `Book` w pakiecie `library`, który powinien zawierać trzy pola:

- `title`: typu `String`, reprezentującego tytuł książki.
- `author`: typu `String`, reprezentującego autora książki.
- `pageCount`: typu `int`, reprezentującego liczbę stron w książce.

B. Dodaj do rekordu `Book`:

- Kompaktowy konstruktor, który weryfikuje, czy liczba stron (`pageCount`) jest większa od 0. Jeśli `pageCount` jest mniejsza lub równa 0, konstruktor powinien rzucać wyjątek `IllegalArgumentException`.
- Metodę `isLongBook`, która zwraca `true`, jeśli liczba stron (`pageCount`) książki jest większa niż 300, i `false` w przeciwnym przypadku.
- Metodę `printDetails`, która wyświetla informacje o książce, włączając w to jej tytuł, autora i liczbę stron.

C. W pakiecie `library`, utwórz klasę testową `TestBook` z metodą `main`, w której:

- Utwórz dwa obiekty typu `Book` z różnymi danymi.
- Wywołaj metodę `printDetails` na każdym z obiektów, aby wyświetlić ich szczegóły.
- Sprawdź, które książki są długie, używając metody `isLongBook`.

Zadanie 3. (15pkt max.)

W pakiecie `sorting` zaimplementuj statyczną metodę generyczną `sortAscending`, która przyjmuje tablicę obiektów typu generycznego `T` i sortuje je w porządku rosnącym. Zakłada się, że typ `T` implementuje interfejs `Comparable<? super T>`. Metoda powinna sortować w miejscu, modyfikując przekazaną tablicę. Utwórz przypadek testowy dla tej metody, używając tablicy obiektów klasy `Product` z polami `name` (nazwa, typ `String`) oraz `price` (cena, typ `double`).

Zadanie 4. (15pkt max.)

W pakiecie `filtering`, zaimplementuj statyczną metodę generyczną `filterGreaterThan`, która przyjmuje `ArrayList<T>` i element typu `T`. Zakładając, że typ `T` implementuje interfejs `Comparable<T>`, metoda powinna zwracać nową listę tablicową elementów z wejściowej listy, które są większe od podanego elementu. Utwórz przypadek testowy dla tej metody.