Kolokwium #2 - Programowanie obiektowe - Zestaw A23

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 `ImmutableColor` w pakiecie `graphics`, która powinna zawierać trzy prywatne i finalne pola:

- `red`: typu `int`, reprezentującego wartość czerwonego koloru.
- `green`: typu `int`, reprezentującego wartość zielonego koloru.
- `blue`: typu `int`, reprezentującego wartość niebieskiego koloru.

B. Dodaj do klasy `ImmutableColor`:

- Konstruktor parametryczny do inicjalizacji wszystkich pól `(red,green,blue)`.
- Publiczne metody dostępowe `getRed`, `getGreen`, `getBlue` 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 koloru w formacie `"(red, green, blue)"`.
  - `equals`: powinna porównywać obiekty `ImmutableColor` na podstawie wartości pól `red`, `green` i `blue`.
  - `hashCode`: powinna generować wartość hash na podstawie pól `red`, `green` i `blue`.

C. W pakiecie `graphics`, utwórz klasę testową `TestImmutableColor` z metodą `main`, w której:

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

Zadanie 2. (15pkt max.)

- Poniższe czynności wykonaj w pakiecie `company`.
- Napisz klasę `Employee` z polami `employeeId` (typu int), `name` (typu String) oraz `salary` (typu double). Zaimplementuj dwie klasy implementujące generyczny interfejs `Comparator`: `SalaryComparator` do porównywania obiektów po polu `salary` (od najwyższego do najniższego wynagrodzenia) oraz `EmployeeIdComparator` do porównywania obiektów po polu `employeeId` (od najniższego do najwyższego identyfikatora pracownika). Stwórz tablicę 5 obiektów klasy `Employee` i posortuj ją zgodnie z oboma kryteriami (najpierw po wynagrodzeniu, a następnie przy równości po identyfikatorze pracownika).

Zadanie 3. (15pkt max.)

- Poniższe czynności wykonaj w pakiecie `finding`.
- Utwórz statyczną metodę generyczną `findFirstNonNull`. Metoda ta przyjmuje tablicę obiektów tego samego typu generycznego `T` i zwraca pierwszy element z listy, który nie jest `null`. Jeśli wszystkie elementy są null, metoda zwraca `null`. Stwórz przypadek testowy.

Zadanie 4. (15pkt max.)

- Poniższe czynności wykonaj w pakiecie `algorithm`.
- Napisz statyczną metodę generyczną `findSmallestKey`, która przyjmuje `TreeMap<K, V>` jako argument i zwraca najmniejszy klucz w mapie. Metoda powinna być w stanie obsłużyć dowolny typ klucza, który implementuje interfejs `Comparable<K>`. Stwórz przypadek testowy dla metody.