Przykładowe kolokwium #1 - Zestaw P13

Ostatnia aktualizacja pliku: 18.11.2023 23:43.

Imię i nazwisko, numer albumu .....

Informacje wstępne

- Łącznie do zdobycia max 40 punktów. Próg zaliczenia: 25 pkt (bez innych punktów).
- **Czas: 90 minut. Po zakończeniu można wyjść, dalszych zajęć nie będzie.**
- **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.
- W pierwszym kolokwium nie występują zagnieżdżone klasy w żadnym z poleceń.
- Jeśli w poleceniu nie są sprecyzowane modyfikatory dostępu, należy dostępować zgodnie z zasadami hermetyzacji (pola prywatne, przy metodach najmniejszy z możliwych, klasy – dostęp pakietowy).
- 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. Klasa `Library` (pol. Biblioteka) (13pkt max.)

A. (1pkt) Klasa `Library` powinna być umieszczona w pakiecie `education`.

B. (1pkt) Klasa powinna posiadać prywatne pola:

- `name` (nazwa biblioteki), typ `String`
- `location` (adres zawierający ulicę, numer posesji, kod pocztowy i miejscowość), typ `String`
- `books` (liczba książek), typ `int`

C. (3pkt) Napisz trzyargumentowy konstruktor tej klasy. Kolejność argumentów powinna być taka sama jak w punkcie B. Warunki sprawdzające poprawność:

- stringi nie mogą być puste lub równe null - wtedy ustaw lokalizację na `"ul. Wiedzy 123, 00-001 Miasteczko"` lub odpowiednio nazwę jako `"Biblioteka Miejska"`.
- liczba książek musi być liczbą dodatnią, w przeciwnym wypadku ustaw ją na 1000.

D. (1pkt) Napisz metody typu `getter` i `setter` dla wszystkich pól. Zapewnij walidację jak w konstruktorze dla setterów, jednak w przypadku błędnych argumentów pozostaw wartości pól bez zmian.

E. (2pkt) Nadpisz metodę `toString`:

```
[NazwaKlasy]: Name: [name]. Location: [location]. Number of books: [books].
```

F. (2pkt) Nadpisz metodę `equals` i `hashCode()`. Dwie bibliotki są równe, gdy mają ten sam adres.

G. (1pkt) Napisz metodę `addBooks` (pol. dodaj książki) z argumentem `int`. Metoda zwiększa `books` o wartość argumentu. Jeśli liczba książek przekroczy 5000, ustaw ją na 5000.

H. (2pkt) Napisz statyczną metodę `checkBookCapacity` (pol. sprawdź pojemność książek) z argumentem typu `Library`. Metoda wyświetla aktualną liczbę książek i ile zostało do limitu 5000.

Zadanie 2. Klasa `SchoolLibrary` (pol. Biblioteka Szkolna) (13pkt max.)

A. (1pkt) Klasa `SchoolLibrary` powinna być umieszczona w pakiecie `education`, w innym pliku niż klasa `Library`.

B. (2pkt) Klasa `SchoolLibrary` dziedziczy po klasie `Library`. Klasa powinna posiadać prywatne pola:

- `schoolType` typu `String` (np. podstawowa, średnia)
- `librarians` typu `int` (liczba bibliotekarzy)

C. (2pkt) Napisz pięcio-argumentowy konstruktor tej klasy. Kolejność argumentów powinna być taka sama jak w punkcie B. Dodatkowe warunki sprawdzające poprawność:

- `schoolType` nie może być pusty lub być nullem - w przeciwnym wypadku ustaw jako `"średnia"`.
- liczba bibliotekarzy musi być większa niż 2 - w przeciwnym wypadku ustaw ją na 3.

D. (1pkt) Napisz metody typu `getter` i `setter` dla wszystkich pól. Zapewnij walidację jak w konstruktorze dla setterów, jednak w przypadku błędnych argumentów pozostaw wartości pól bez zmian.

E. (2pkt) Nadpisz metodę `toString`:

```
[NazwaKlasy]: Name: [name]. Location: [location]. Number of books: [books].
School Type: [schoolType]. Number of librarians: [librarians].
```

F. (2pkt) Nadpisz metodę `addBooks` z argumentem `int`. Zwiększ `books` o wartość argumentu. Jeśli liczba książek przekroczy 3000, ustaw ją na 3000.

G. (2pkt) Nadpisz metodę `equals` i `hashCode()`. Dwa obiekty są równe, gdy mają ten sam adres i typ szkoły.

H. (1pkt) Zapewnij zgodność pozostałych metod z metodami z klasy bazowej i z tzw. dobrymi praktykami.

Zadanie 3. Klasa `TestLibrary` (pol. klasa testująca dla zad 1 i 2) (9pkt max.)

A. (2pkt) Klasę `TestLibrary` umieść w pakiecie tym samym co klasy z punktu w zadaniu 1 i 2, ale w innym pliku. Umieść w tej klasie tylko metodę `main`.

B. (7pkt) W metodzie `main` stwórz 5 obiektów w typach definiowanych w zadaniu 1 i 2. Dodaj je na listę tablicową. Następnie iterując po niej wywołaj nadpisaną metodę `toString` oraz `addBooks`.

Zadanie 4. Klasa `Employee` (pol. pracownik) (5pkt max.)

A. (1pkt) Stwórz klasę `Employee` reprezentującą pracownika. Klasę umieść w pakiecie `pl.com.corporation`. Klasa powinna zawierać:

- Prywatne pola: `name` (typu `String`), `salary` (typu `double`).
- Publiczny konstruktor przyjmujący `name` i `salary` jako argumenty i ustawiający te pola.
- Metody `getter` dla obu pól.

B. (2pkt) Dodaj statyczne prywatne pole `totalEmployeeCount` typu `int` w klasie `Employee`, które będzie śledzić łączną liczbę utworzonych pracowników. Zmodyfikuj konstruktor klasy `Employee`, aby każde utworzenie nowego obiektu `Employee` inkrementowało `totalEmployeeCount`.

C. (1pkt) Dodaj publiczną statyczną metodę `getTotalEmployeeCount`, która zwraca wartość pola `totalEmployeeCount`.

D. (1pkt) W klasie `TestEmployee` (inny plik w tym samym pakiecie) w metodzie `main` utwórz 5 obiektów `Employee` Wyświetl łączną liczbę stworzonych pracowników.