Kolokwium #1 - Programowanie obiektowe - Zestaw W25

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 `Document` (Dokument) (13pkt max.)

A. (1pkt) Klasa `Document` powinna być umieszczona w pakiecie `com.example.legal`.

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

- `title`, (tytuł), typ `String`
- `author`, (autor), typ `String`
- `pageCount`, (liczba stron), typ `int`

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

- String `title` nie może być null - w przypadku błędnej wartości ustaw `"Untitled Document"`.
- String `author` nie może być null - w przypadku błędnej wartości ustaw `"Anonymous"`.
- Liczba stron `pageCount` musi być liczbą dodatnią, w przeciwnym wypadku ustaw ją na 1.

D. (2pkt) Napisz metody typu `getter` i `setter` dla wszystkich pól. Powinny mieć taką samą walidację jak w konstruktorze.

E. (1pkt) Nadpisz metodę `toString` tak, aby zwracała napis z reprezentacją obiektu zgodnie z następującym schematem (zwróć uwagę na wielkość znaków i znaki interpunkcyjne):

```
[Document]. Title: [title]. Author: [author]. Page Count: [pageCount].
```

Uwaga. Jeśli tytuł lub autor jest napisem pustym (równy `""`), to pomiń odpowiedni fragment napisu.

F. (2pkt) Nadpisz metodę `equals`. Dwa dokumenty są sobie "równe" wtedy i tylko wtedy, gdy mają ten sam tytuł i autora. Nadpisz metodę `hashCode()` zgodnie z metodą `equals()`.

G. (1pkt) Napisz metodę (zwykłą) `addPage` bez argumentów. Metoda zwiększa liczbę stron o 1.

H. (2pkt) Napisz metodę statyczną `changePageCount`, której argumentem jest obiekt typu `Document` i zmienna typu `int`. Metoda ma zmienić liczbę stron w przekazanym jako argument obiekcie na wartość przekazaną przez drugi argument metody. Jeśli drugi argument jest liczbą ujemną, to metoda ma nic nie robić. Jeśli pierwszy argument metody jest null, to metoda też ma nic nie robić.

Zadanie 2. Klasa `Contract` (Umowa) (13pkt max.)

A. (1pkt) Klasa `Contract` powinna być umieszczona w pakiecie `com.example.legal` w innym pliku niż klasa `Document`.

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

- `contractType`, typu `String` (rodzaj umowy)
- `expirationDate`, typu `String` (data wygaśnięcia)

C. (2pkt) Napisz pięcio-argumentowy konstruktor tej klasy. Kolejność argumentów powinna być taka sama jak w punkcie B (najpierw z klasy bazowej, potem pochodnej). Zapewnij niezależnie warunki sprawdzające poprawność dodatkowo:

- Rodzaj umowy `contractType` nie może być null - w przeciwnym wypadku ustaw ją na `"General Agreement"`.
- Data wygaśnięcia `expirationDate` nie może być null - w przeciwnym wypadku ustaw ją na `"Indefinite"`.

D. (2pkt) Napisz metody typu `getter` i `setter` dla wszystkich pól. Powinny mieć taką samą walidację jak w konstruktorze.

E. (1pkt) Nadpisz metodę `toString` tak, aby zwracała napis z reprezentacją obiektu zgodnie z następującym schematem (zwróć uwagę na wielkość znaków, łamanie linii i znaki interpunkcyjne):

```
[Contract]. Title: [title]. Author: [author]. Page Count: [pageCount].
Contract Type: [contractType]. Expiration Date: [expirationDate].
```

Uwaga. Podobnie jak w klasie bazowej, jeśli tytuł lub autor jest napisem pustym (równy `""`), to pomiń odpowiedni fragment napisu.

F. (2pkt) Nadpisz metodę `equals`. Dwie umowy są sobie "równe" wtedy i tylko wtedy, gdy mają ten sam tytuł, autora oraz rodzaj umowy. Nadpisz metodę `hashCode()` zgodnie z metodą `equals()`.

G. (1pkt) Nadpisz metodę `addPage` tak, aby zmniejszała liczbę stron o 1.

H. (2pkt) Napisz metodę statyczną `changePageCount`, której argumentem jest obiekt typu `Contract` i zmienna typu `int`. Metoda ma zmienić liczbę stron w przekazanym jako argument obiekcie na wartość przekazaną przez drugi argument metody. Jeśli drugi argument jest liczbą dodatnią, to metoda ma nic nie robić. Jeśli pierwszy argument metody jest null, to metoda też ma nic nie robić.

Zadanie 3. Klasa `TestDocument` (pol. klasa testująca dla dokumentu) (9pkt max.)

A. (2pkt) Klasę `TestDocument` umieść w tym samym pakiecie co klasy z zadania 1 i 2. Umieść w tej klasie tylko metodę `main`.

B. (7pkt) Wywołaj wszystkie metody z zadania 1 i 2. (np. zwykłe, statyczne, konstruktory). Wywołanie `getter`-ów i `setter`-ów nie jest obowiązkowe.

Zadanie 4. Klasa `Clothing` (Ubranie) (5pkt max.)

A. (1pkt) Klasa `Clothing` powinna być umieszczona w pakiecie `pl.edu.uwm.wmii.clothing`.

B. (1pkt) Utwórz abstrakcyjną klasę bazową `Clothing`. Klasa ta powinna zawierać następujące pola chronione:

- `material` (materiał), typu `String`
- `size` (rozmiar), typu `String`
- `color` (kolor), typu `String`

Utwórz konstruktor w klasie `Clothing`, który przyjmuje wszystkie trzy atrybuty jako parametry. Konstruktor powinien przypisywać wartości parametrów do odpowiednich pól klasy.

C. (1pkt) Zdefiniuj w klasie `Clothing` dwie metody abstrakcyjne:

- `public abstract void wear()` - metoda nie zwraca wartości i nie przyjmuje żadnych argumentów, ma symulować proces zakładania ubrania.
- `public abstract String getMaintenanceInstructions()` - metoda zwraca instrukcje dotyczące konserwacji ubrania (np. pranie, prasowanie), nie przyjmuje żadnych argumentów.

D. (1pkt) W tym samym pakiecie w różnych nowych plikach stwórz klasy pochodne `Shirt` (Koszula) i `Pants` (Spodnie) dziedziczące po klasie `Clothing`. Dodaj dowolną implementację (niepustą), aby spełnione były zasady kompilacji i kod wykonywał się bez błędów.

E. (1pkt) Stwórz klasę `TestClothing`, umieść ją w innym pliku w pakiecie `pl.edu.uwm.wmii.clothing`. W tej klasie dodaj metodę `main`. Wywołaj w niej metody z punkty D.