Kolokwium #1 - Programowanie obiektowe - Zestaw W28

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 `Software` (pol. Oprogramowanie) (13pkt max.)

A. (1pkt) Klasa `Software` powinna być umieszczona w pakiecie `com.techacademy.software`.

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

- `name`, (nazwa oprogramowania), typ `String`
- `version`, (wersja oprogramowania zawierająca główną wersję, podwersję i numer poprawki), typ `String`
- `users`, (liczba użytkowników), typ `int`

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

- stringi nie mogą być nullami i nie mogą być puste (równe `""`) - wtedy ustaw wersję na `"1.0.0"` lub odpowiednio nazwę jako `"Generic Software"`.
- zwróć uwagę na wielkość znaków i znaki interpunkcyjne.

D. (2pkt) Napisz metody typu `getter` i `setter` dla wszystkich pól. Pamiętaj by sprawdzić kryteria podane w konstruktorze. W przypadku błędnych argumentów, metoda ma nic nie robić.

E. (1pkt) Nadpisz metodę `toString` tak, aby zwracała napis z reprezentacją obiektu. Na początku powinna być nazwa klasy - potem wartości wszystkich pól. Powinno odbyć się do według schematu:

```
[Software]. Name: [name]. Version: [version]. Number of users: [users].
```

Pomiń fragment dotyczący liczby użytkowników, gdy ta jest ujemna.

F. (2pkt) Nadpisz metodę `equals`. Dwa programy są sobie "równe" wtedy i tylko wtedy, gdy mają tą samą liczbę użytkowników. Nadpisz metodę `hashCode()`, która generuje kod hash dla odpowiedniego obiektu. Metoda ta powinna być zgodna z metodą `equals()`.

G. (1pkt) Napisz metodę (zwykłą) `addUsers` z argumentem typu `int`. Metoda powiększa pole `users` o wartość przekazaną przez argument. Jeśli po powiększeniu pole `users` będzie większe niż 100, to ustaw je na 100.

H. (2pkt) Napisz metodę statyczną `checkUserCapacity` której argumentem jest obiekt typu `Software`. Metoda ma zwrócić liczbę całkowitą ile zostało do pełnego limitu (różnica 1000 i liczby użytkowników). Metoda ma nic nie wyświetlać.

Zadanie 2. Klasa `TextEditor` (pol. edytor tekstu) (13pkt max.)

A. (1pkt) Klasa `TextEditor` powinna być umieszczona w pakiecie `com.techacademy.software` w innym pliku niż klasa `Software`.

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

- `fileFormat`, typu `String` (np. obsługiwany format pliku np. `.txt`, `.docx`)
- `features`, typu `int` (liczba dostępnych funkcji edytora)

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:

- format pliku nie może być nullem i nie może być pusty (równy `""`) - w przeciwnym wypadku ustaw `".txt"`
- liczba funkcji musi być liczbą nieujemną - w przeciwnym wypadku ustaw ją jako 10.

D. (2pkt) Napisz metody typu `getter` i `setter` dla wszystkich pól. Pamiętaj by sprawdzić kryteria podane w konstruktorze. W przypadku błędnych argumentów, metoda ma nic nie robić.

E. (1pkt) Nadpisz metodę `toString` tak, aby zwracała napis z reprezentacją obiektu. Na początku powinna być nazwa klasy - potem wartości wszystkich pól. Powinno odbyć się do według schematu:

```
[TextEditor]. Name: [name]. Version: [version]. Number of users: [users].
File Format: [fileFormat]. Number of features: [features].
```

Pomiń fragment dotyczący liczby użytkowników, gdy ta jest ujemna.

F. (2pkt) Nadpisz metodę (zwykłą) `addUsers` z argumentem typu `int`. Metoda powiększa pole `users` o wartość przekazaną przez argument. Jeśli po powiększeniu pole `users` będzie większe niż 2000, to ustaw je na 2000. Dodatkowo zwiększ pole `features` o dwukrotność przekazanego argumentu.

G. (2pkt) Nadpisz metodę `equals`. Dwa edytory tekstu są sobie "równe" wtedy i tylko wtedy, gdy mają tę samą liczbę użytkowników oraz obsługują ten sam format pliku. Nadpisz metodę `hashCode()`, która generuje kod hash dla odpowiedniego obiektu. Metoda ta powinna być zgodna z metodą `equals()`.

H. (1pkt) Napisz metodę statyczną `checkUserCapacity` której argumentem jest obiekt typu `TextEditor`. Metoda ma zwrócić liczbę całkowitą ile zostało do pełnego limitu (różnica 2000 i liczby użytkowników). Metoda ma nic nie wyświetlać.

Zadanie 3. Klasa `TestSoftware` (9pkt max.)

A. (2pkt) Klasę `TestSoftware` umieść bezpośrednio w katalogu `src` poza pakietami. Umieść w tej klasie tylko metodę `main`.

B. (7pkt) Wywołaj wszystkie metody z zadania 1 i 2 (np. zwykłe, statyczne, konstruktory).

Zadanie 4. Klasa `Store` (pol. sklep) (5pkt max.)

A. (1pkt) Klasa `Store` powinna być umieszczona w pakiecie `com.retailmanagement.store`.

B. (1pkt) Klasa powinna zawierać trzy prywatne atrybuty:

- `category` (kategoria sklepu), typu `String`.
- `address`, typu `String`.
- `staffCount` (liczba etatów pracowników), typu `double`.

C. (1pkt) W klasie `Store`, zaimplementuj statyczną metodę `createDepartmentStore(String category, String address, double staffCount)`. Metoda ma zwrócić nowy obiekt typu `Store`, którego pola ustawione są z argumentów metody.

D. (1pkt) W klasie `Store`, zaimplementuj nie-statyczną metodę `createSpecialtyStore(String category, String address, double staffCount)`. Metoda ma zwrócić nowy obiekt typu `Store`, którego pola ustawione są z argumentów metody.

E. (1pkt) Stwórz klasę `TestStore`, umieść ją w innym pliku w pakiecie `com.retailmanagement.store`. W klasie `TestStore` dodaj metodę `main`. Wywołaj w niej metody z punktu C i D.