Kolokwium #1 - Programowanie obiektowe - Zestaw W14

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 `Beverage` (pol. Napój) (13pkt max.)

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

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

- `flavor`, (smak napoju), typ `String`
- `temperature`, (temperatura podania), typ `double`
- `volume`, (objętość), 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 `flavor` nie może być null i nie może być pusty (`""`) - w przypadku błędnej wartości ustaw `"Lemon"`.
- Temperatura `temperature` musi być liczbą dodatnią, w przeciwnym wypadku ustaw ją na 20.0
- Objętość `volume` musi być liczbą dodatnią większą niż 50, w przeciwnym wypadku ustaw ją na 250.

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 zgodnie z następującym schematem (zwróć uwagę na wielkość znaków i znaki interpunkcyjne):

```
[Beverage]. Flavor: [flavor]. Temperature: [temperature]. Volume: [volume].
```

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

G. (1pkt) Napisz metodę (zwykłą) `refill` (pol. uzupełnij) z argumentem typu `int`. Metoda powiększa pole `volume` o wartość przekazaną przez argument (ale tylko gdy argument metody jest liczbą dodatnią).

H. (2pkt) Napisz metodę statyczną `checkVolume` (pol. sprawdź objętość) której argumentem jest obiekt typu `Beverage`. Metoda ma zwrócić aktualną objętość napoju.

Zadanie 2. Klasa `Tea` (pol. Herbata) (13pkt max.)

A. (1pkt) Klasa `Tea` powinna być umieszczona w pakiecie `pl.edu.uwm.wmii.drinks` w innym pliku niż klasa `Beverage`.

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

- `type`, typu `String` (rodzaj herbaty)
- `brewTime`, typu `int` (czas parzenia w minutach)

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 herbaty `type` nie może być null i nie może być pusty (`""`) - w przeciwnym wypadku ustaw go na `"Green"`.
- Czas parzenia `brewTime` musi być większą niż 2 - w przeciwnym wypadku ustaw go na 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 zgodnie z następującym schematem (zwróć uwagę na wielkość znaków, znaki interpunkcyjne i łamanie linii):

```
[Tea]. Flavor: [flavor]. Temperature: [temperature]. Volume: [volume].
Type: [type]. Brew Time: [brewTime].
```

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

G. (2pkt) Nadpisz metodę `refill` tak, aby dodatkowo (poza tym co robi w klasie bazowej) skracała czas parzenia o 1 minutę za każde 100 ml dodanej objętości.

H. (1pkt) Napisz metodę statyczną `checkTeaType` (pol. sprawdź rodzaj herbaty) której argumentem jest obiekt typu `Tea`. Metoda ma zwrócić aktualny rodzaj herbaty.

Zadanie 3. Klasa `TestBeverage` (pol. klasa testująca dla napoju) (9pkt max.)

A. (2pkt) Klasę `TestBeverage` 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 `BuildingMaterial` (pol. Materiał Budowlany) (5pkt max.)

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

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

- `materialType` (typ materiału), typu `String`.
- `qualityGrade`, typu `String`.
- `quantityInStock` (ilość na stanie), typu `int`.

C. (1pkt) W klasie `BuildingMaterial`, zaimplementuj statyczną metodę `isSufficientQuantity` z argumentem typu `BuildingMaterial`. Metoda ma zwrócić wartość logiczną sprawdzającą czy ilość materiału na stanie przekazanego jako argument jest większa niż 100. Dla argumentu będącego `null`em zwróć `false`.

D. (1pkt) W klasie `BuildingMaterial`, zaimplementuj nie-statyczną metodę `isSufficientQuantity` bez argumentu. Metoda ma zwrócić wartość logiczną sprawdzającą czy ilość materiału na stanie bieżącego obiektu jest większa niż 100.

E. (1pkt) Stwórz klasę `TestBuildingMaterial`, umieść ją w innym pliku w pakiecie `pl.edu.uwm.wmii.construction`. W klasie `TestBuildingMaterial` dodaj metodę `main`. Wywołaj w niej metody z punktu C i D.