Kolokwium #1 - Programowanie obiektowe - Zestaw W13

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 `Game` (pol. Gra) (13pkt max.)

A. (1pkt) Klasa `Game` powinna być umieszczona w pakiecie `com.uwm.wmii.gaming`.

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

- `title`, (tytuł gry), typ `String`
- `genre`, (gatunek gry), typ `String`
- `rating`, (ocena gry), typ `double`

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

- Stringi `title` i `genre` nie mogą być nullami i nie mogą być puste - wtedy ustaw domyślnie `"Unknown Game"` dla tytułu i `"Miscellaneous"` dla gatunku.
- Ocena `rating` musi być liczbą w zakresie od 0.0 do 10.0 (bez wartości granicznych), w przeciwnym wypadku ustaw ją na 5.0.

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

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

```
[Game]. Title: [title]. Genre: [genre]. Rating: [rating].
```

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

G. (1pkt) Napisz metodę `updateRating` z argumentem typu `double`. Metoda powiększa pole `rating` o wartość przekazaną przez argument. Jeśli nowa wartość oceny wyjdzie poza zakres 0.0-10.0 (bez wartości granicznych), to nie rób żadnej zmiany.

H. (2pkt) Napisz metodę statyczną `compareRating` z argumentem typu `Game`. Metoda ma zwracać różnicę w ocenach między przekazaną grą a średnią oceną (przyjmij, że to 7.0).

Zadanie 2. Klasa `BoardGame` (pol. Gra Planszowa) (13pkt max.)

A. (1pkt) Klasa `BoardGame` powinna być umieszczona w pakiecie `com.uwm.wmii.gaming` w innym pliku niż klasa `Game`.

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

- `numberOfPlayers`, typu `int` (liczba graczy)
- `averagePlaytime`, typu `double` (średni czas gry 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:

- Liczba graczy `numberOfPlayers` musi być liczbą dodatnią - w przeciwnym wypadku ustaw ją na 2.
- Średni czas gry `averagePlaytime` musi być liczbą dodatnią - w przeciwnym wypadku ustaw go na 30.0.

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

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

```
[BoardGame]. Title: [title]. Genre: [genre]. Rating: [rating].
Number of Players: [numberOfPlayers]. Average Playtime: [averagePlaytime].
```

F. (2pkt) Nadpisz metodę `equals`. Dwie gry planszowe są sobie "równe" wtedy i tylko wtedy, gdy mają ten sam tytuł, gatunek i liczbę graczy. Nadpisz metodę `hashCode()` zgodnie z metodą `equals()`.

G. (2pkt) Nadpisz metodę `updateRating`. Dodatkowo do zmiany oceny (według klasy bazowej), metoda powinna również zwiększać średni czas gry o 5 minut za każdy pełny punkt oceny powyżej 7.0.

H. (1pkt) Napisz metodę statyczną `compareRating` z argumentem typu `BoardGame`. Metoda ma zwracać różnicę w ocenach między przekazaną grą a średnią oceną (przyjmij, że to 8.0).

Zadanie 3. Klasa `TestGame` (pol. klasa testująca dla gry) (9pkt max.)

A. (2pkt) Klasę `TestGame` 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). Wywołanie `getter`-ów i `setter`-ów nie jest obowiązkowe.

Zadanie 4. Klasa `Vegetable` (pol. Warzywo) (5pkt max.)

A. (1pkt) Klasa `Vegetable` powinna być umieszczona w pakiecie `com.uwm.wmii.agriculture`.

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

- `type` (typ warzywa), typu `String`.
- `origin`, typu `String` (kraj pochodzenia).
- `shelfLife` (okres przydatności do spożycia), typu `int` (w dniach).

C. (1pkt) W klasie `Vegetable`, zaimplementuj statyczną metodę `createTomato(String type, String origin, int shelfLife)`. Metoda ma zwrócić nowy obiekt typu `Vegetable`, którego pola ustawione są z argumentów metody.

D. (1pkt) W klasie `Vegetable`, zaimplementuj nie-statyczną metodę `growVegetable(String type, String origin, int shelfLife)`. Metoda ma zwrócić nowy obiekt typu `Vegetable`, którego pola ustawione są z argumentów metody.

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