Kolokwium #1 - Programowanie obiektowe - Zestaw W23

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 `MusicalInstrument` (Instrument Muzyczny) (13pkt max.)

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

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

- `type`, (typ instrumentu), typ `String`
- `brand`, (marka), typ `String`
- `price`, (cena), typ `double`

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

- String `type` nie może być null - w przypadku błędnej wartości ustaw `"Guitar"`.
- String `brand` nie może być null - w przypadku błędnej wartości ustaw `"Yamaha"`.
- Cena `price` musi być liczbą nieujemną, w przeciwnym wypadku ustaw ją na 50.0.

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):

```
[MusicalInstrument]. Type: [type]. Brand: [brand]. Price: [price].
```

Uwaga. Jeśli typ lub marka jest napisem pustym (równe `""`), to pomiń odpowiedni fragment napisu.

F. (2pkt) Nadpisz metodę `equals`. Dwa instrumenty są sobie "równe" wtedy i tylko wtedy, gdy mają ten sam typ (pole `type`) i cenę. Nadpisz metodę `hashCode()` zgodnie z metodą `equals()`.

G. (1pkt) Napisz metodę (zwykłą) `tune` (pol. strojenie) bez argumentów. Metoda zwiększa cenę o 10.23.

H. (2pkt) Napisz metodę statyczną `changePrice` (pol. zmień cenę), której argumentem jest obiekt typu `MusicalInstrument` i zmienna typu `double`. Metoda ma zmienić cenę 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 `Guitar` (Gitara) (13pkt max.)

A. (1pkt) Klasa `Guitar` powinna być umieszczona w pakiecie `pl.edu.uwm.wmii.music` w innym pliku niż klasa `MusicalInstrument`.

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

- `stringCount`, typu `int` (liczba strun)
- `fretCount`, typu `int` (liczba progów)

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 strun `stringCount` musi być liczbą dodatnią - w przeciwnym wypadku ustaw ją na 8.
- Liczba progów `fretCount` musi być liczbą dodatnią - w przeciwnym wypadku ustaw ją na 25.

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, łamanie linii i znaki interpunkcyjne):

```
[Guitar]. Type: [type]. Brand: [brand]. Price: [price].
String Count: [stringCount]. Fret Count: [fretCount].
```

Uwaga. Podobnie jak w klasie bazowej, jeśli typ lub marka jest napisem pustym (równe `""`), to pomiń odpowiedni fragment napisu.

F. (2pkt) Nadpisz metodę `equals`. Dwie gitary są sobie "równe" wtedy i tylko wtedy, gdy mają ten sam typ (pole `type`), cenę oraz liczbę strun. Nadpisz metodę `hashCode()` zgodnie z metodą `equals()`.

G. (1pkt) Nadpisz metodę `tune` tak, aby zmniejszała cenę o 30.

H. (2pkt) Napisz metodę statyczną `changePrice` (pol. zmień cenę), której argumentem jest obiekt typu `Guitar` i zmienna typu `double`. Metoda ma zmienić cenę 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 3. Klasa `TestMusicalInstrument` (pol. klasa testująca dla instrumentu muzycznego) (9pkt max.)

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

B. (7pkt) W metodzie `main` stwórz 5 obiektów w typach definiowanych w zadaniu 1 i 2. Następnie sprawdź poprawność działania metody `equals` na co najmniej 5 różnych sposobów.

Zadanie 4. Klasa `Building` (Budowla) (5pkt max.)

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

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

- `height` (wysokość budynku), typu `double`.
- `floorCount` (liczba pięter), typu `int`.
- `constructionYear` (rok budowy), typu `int`.

C. (1pkt) W klasie `Building`, zaimplementuj statyczną metodę `estimateMaintenanceCost` z argumentem typu `Building`. Metoda ma zwrócić szacowany roczny koszt utrzymania budynku (typu `double`) w złotych, obliczany jako iloczyn liczby pięter (`floorCount`) i stałego kosztu utrzymania na piętro (10000 zł za piętro). Dla argumentu będącego `null`em zwróć 0.0.

D. (1pkt) W klasie `Building`, zaimplementuj nie-statyczną metodę `calculateAge` bez argumentu. Metoda ma zwrócić wiek budynku (typu `int`), obliczany jako różnica między bieżącym rokiem (2023) a rokiem budowy (`constructionYear`).

E. (1pkt) Stwórz klasę `TestBuilding`, umieść ją w innym pliku w pakiecie `pl.edu.uwm.wmii.architecture`. W klasie `TestBuilding` dodaj metodę `main`. Wywołaj w niej metody `estimateMaintenanceCost` i `calculateAge` na przykładowych obiektach klasy `Building`.