Kolokwium #1 - Programowanie obiektowe - Zestaw W17

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 `Vegetable` (Warzywo) (13pkt max.)

A. (1pkt) Klasa `Vegetable` powinna być umieszczona w pakiecie `pl.vegetables`.

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

- `species`, (gatunek warzywa), typ `String`
- `origin`, (miejsce pochodzenia zawierające kraj i region), typ `String`
- `shelfLife`, (okres przydatności do spożycia w dniach), typ `int`

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

- miejsce pochodzenia nie może być puste (równe `""`) lub równe `null` - wtedy ustaw miejsce pochodzenia na `"Polska, Mazowsze"`
- okres przydatności do spożycia musi być liczbą dodatnią, w przeciwnym wypadku ustaw go na 7 dni.
- zwróć uwagę na wielkość znaków i znaki interpunkcyjne

D. (1pkt) 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. (3pkt) Nadpisz metodę `toString` tak, aby zwracała napis z reprezentacją obiektu. Na początku powinna być nazwa klasy - potem wartości wszystkich pól. Powinno to odbyć się według schematu (zwróć uwagę na wielkość znaków i znaki interpunkcyjne, wszystko w jednej linii):

```
[NazwaKlasy]: Species: [species]. Origin: [origin]. Shelf Life: [shelfLife] days.
```

lub jeśli gatunek nie jest ustalony (jest pustym napisem lub nullem):

```
[NazwaKlasy]: Origin: [origin]. Shelf Life: [shelfLife] days.
```

F. (2pkt) Nadpisz metodę `equals`. Dwa warzywa są sobie "równe" wtedy i tylko wtedy, gdy mają ten sam gatunek. Nadpisz metodę `hashCode()`, która generuje wartość hash dla odpowiedniego obiektu. Metoda ta powinna być zgodna z metodą `equals()`.

G. (2pkt) Napisz metodę (zwykłą) `ripen` (pol. dojrzewanie) z argumentem typu `int`. Metoda wydłuża pole `shelfLife` o wartość przekazaną przez argument. Jeśli po wydłużeniu pole `shelfLife` będzie większe niż 30, to ustaw je na 30.

Zadanie 2. Klasa `Tomato` (Pomidor) (13pkt max.)

A. (1pkt) Klasa `Tomato` powinna być umieszczona w pakiecie `pl.vegetables` w innym pliku niż klasa `Vegetable`.

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

- `color` typu `String` (np. kolor pomidora np. czerwony, zielony)
- `tasteProfile` typu `int` (skala smaku od 1 do 10, gdzie 1 jest najmniej smacznym)

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:

- kolor nie powinien być pusty (równy `""`) lub równy `null` - w przeciwnym wypadku ustaw go na `"czerwony"`
- profil smaku musi być liczbą z zakresu 1 do 10 - w przeciwnym wypadku ustaw go jako 5.

D. (1pkt) 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. (2pkt) Nadpisz metodę `toString` tak, aby zwracała napis z reprezentacją obiektu. Na początku powinna być nazwa klasy - potem wartości wszystkich pól. Powinno to odbyć się według schematu (zwróć uwagę na wielkość znaków i znaki interpunkcyjne, zwróć uwagę na łamanie linii):

```
[NazwaKlasy]: Species: [species]. Origin: [origin]. Shelf Life: [shelfLife] days.
Color: [color]. Taste Profile: [tasteProfile].
```

lub jeśli gatunek nie jest ustalony (jest pustym napisem lub nullem):

```
[NazwaKlasy]: Origin: [origin]. Shelf Life: [shelfLife] days.
Color: [color]. Taste Profile: [tasteProfile].
```

F. (2pkt) Nadpisz metodę (zwykłą) `ripen` z argumentem typu `int`. Metoda wydłuża pole `shelfLife` o wartość przekazaną przez argument. Jeśli po wydłużeniu pole `shelfLife` będzie większe niż 60, to ustaw je na 60. Dodatkowo zwiększ skalę smaku o 1 (pamiętając o ograniczeniu górnym).

G. (2pkt) Nadpisz metodę `equals`. Dwa pomidory są sobie "równe" wtedy i tylko wtedy, gdy mają ten sam gatunek, skalę smaku oraz taki sam kolor. Nadpisz metodę `hashCode()`, która generuje wartość hash dla odpowiedniego obiektu. Metoda ta powinna być zgodna z metodą `equals()`.

H. (1pkt) Zapewnij zgodność pozostałych metod z metodami z klasy bazowej.

Zadanie 3. Klasa `TestVegetable` (pol. klasa testująca dla warzywa) (9pkt max.)

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

B. (7pkt) W metodzie `main` stwórz 4 obiekty w typach definiowanych w zadaniu 1 i 2. Następnie sprawdź poprawność działania metody `equals` i `hashCode` na co najmniej 5 różnych sposobach.

Zadanie 4. Klasa `Game` (Gra) (5pkt max.)

A. (1pkt) Stwórz abstrakcyjną klasę `Game` zawierającą publiczną abstrakcyjną metodę `getRating()`, która nie przyjmuje argumentów i zwraca `double`. Klasę umieść w pakiecie `entertainment`.

B. (2pkt) Utwórz dwie klasy pochodne od `Game`: `StrategyGame` i `AdventureGame`. W obu klasach nadpisz metodę `getRating()`. Dla `StrategyGame` niech zwraca ocenę 8.5, a dla `AdventureGame` niech zwraca ocenę 7.3.

C. (2pkt) W klasie `TestGame` w pakiecie `entertainment` utwórz listę tabllicową typu `Game` i zainicjalizuj ją 5 instancjami `StrategyGame` i `AdventureGame`. Iteruj po liście tablicowej wywołując metodę `getRating()` dla każdej gry (wyświetl oceny na standardowym wyjściu).