Kolokwium #1 - Programowanie obiektowe - Zestaw W12

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 `Product` (pol. Produkt) (13pkt max.)

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

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

- `name`, (nazwa produktu), typ `String`
- `price`, (cena), typ `double`
- `quantity`, (ilość na stanie), 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 `name` nie może być null i nie może być pusty (`""`) - w przypadku błędnej wartości ustaw `"Apple"`.
- Cena `price` musi być liczbą dodatnią, w przeciwnym wypadku ustaw ją na 0.99.
- Ilość `quantity` musi być liczbą dodatnią lub równą 0, w przeciwnym wypadku ustaw ją na 5.

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

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

```
[Product]. Name: [name]. Price: [price]. Quantity: [quantity].
```

F. (2pkt) Nadpisz metodę `equals`. Dwa produkty są sobie "równe" wtedy i tylko wtedy, gdy mają tę samą nazwę i cenę. Nadpisz metodę `hashCode()` zgodnie z metodą `equals()`.

G. (1pkt) Napisz metodę (zwykłą) `restock` (pol. uzupełnij stan) z argumentem typu `int`. Metoda powiększa pole `quantity` o wartość przekazaną przez argument.

H. (2pkt) Napisz metodę statyczną `checkStock` (pol. sprawdź stan magazynowy), której argumentem jest obiekt typu `Product`. Metoda ma zwrócić aktualną ilość produktu na stanie.

Zadanie 2. Klasa `Electronics` (pol. Elektronika) (13pkt max.)

A. (1pkt) Klasa `Electronics` powinna być umieszczona w pakiecie `pl.edu.uwm.wmii.store` w innym pliku niż klasa `Product`.

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

- `warrantyPeriod`, typu `int` (okres gwarancji w miesiącach)
- `powerUsage`, typu `double` (zużycie energii w kWh)

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:

- Okres gwarancji `warrantyPeriod` musi być liczbą nieujemną - w przeciwnym wypadku ustaw ją na 24.
- Zużycie energii `powerUsage` musi być liczbą dodatnią, w przeciwnym wypadku ustaw je na 0.5.

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

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):

```
[Electronics]. Name: [name]. Price: [price]. Quantity: [quantity].
Warranty Period: [warrantyPeriod]. Power Usage: [powerUsage].
```

F. (2pkt) Nadpisz metodę `equals`. Dwa produkty elektroniczne są sobie "równe" wtedy i tylko wtedy, gdy mają tę samą nazwę, cenę i okres gwarancji. Nadpisz metodę `hashCode()` zgodnie z metodą `equals()`.

G. (2pkt) Nadpisz metodę `restock` tak, aby dodatkowo (poza tym co robi w klasie bazowej) zwiększała okres gwarancji o 1 miesiąc za każde nowo dodane 10 sztuk produktu.

H. (1pkt) Napisz metodę statyczną `checkStock` (pol. sprawdź stan magazynowy) której argumentem jest obiekt typu `Electronics`. Metoda ma zwrócić aktualną ilość produktu na stanie.

Zadanie 3. Klasa `TestProduct` (pol. klasa testująca dla produktu) (9pkt max.)

A. (2pkt) Klasę `TestProduct` 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 `Software` (pol. Oprogramowanie) (5pkt max.)

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

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

- `type` (typ oprogramowania), typu `String`.
- `version`, typu `String`.
- `licenseCount` (liczba licencji), typu `int`.

C. (1pkt) W klasie `Software`, zaimplementuj statyczną metodę `isLicenseAvailable` z argumentem typu `Software`. Metoda ma zwrócić wartość logiczną sprawdzającą czy liczba licencji oprogramowania przekazanego jako argument jest większa niż 3. Dla argumentu będącego `null`em zwróć `false`.

D. (1pkt) W klasie `Software`, zaimplementuj nie-statyczną metodę `isLicenseAvailable` bez argumentu. Metoda ma zwrócić wartość logiczną sprawdzającą czy liczba licencji bieżącego obiektu jest większa niż 3.

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