Przykładowe kolokwium #1 - Zestaw P15

Ostatnia aktualizacja pliku: 19.11.2023 18:04.

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 `Hospital` (pol. Szpital) (13pkt max.)

A. (1pkt) Klasa `Hospital` powinna być umieszczona w pakiecie `healthcare`.

B. (1pkt) Klasa powinna posiadać prywatne pola: - `name`, (nazwa szpitala), typ `String` - `location`, (lokalizacja zawierająca ulicę, numer posesji, kod pocztowy i miejscowość), typ `String` - `capacity`, (pojemność szpitala, czyli liczba łóżek), typ `double`

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

- Lokalizacja nie może być pusta lub równa null (równa `""`) - w takim przypadku ustaw domyślną lokalizację, np. `"ul. Zdrowia 123, 00-999 Warszawa"`.
- Pojemność musi być liczbą dodatnią, w przeciwnym wypadku ustaw ją na 50.0.

D. (1pkt) Napisz metody typu `getter` i `setter` dla wszystkich pól. Pamiętaj, aby 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. Schemat:

```
[NazwaKlasy]: Name: [name]. Location: [location]. Capacity: [capacity].
```

lub jeśli nazwa nie jest ustalona (jest pustym napisem lub nullem):

```
[NazwaKlasy]: Location: [location]. Capacity: [capacity].
```

F. (2pkt) Nadpisz metodę `equals`. Dwa szpitale są sobie "równe" wtedy i tylko wtedy, gdy wszystkie ich pola są identyczne. Nadpisz metodę `hashCode()`, która generuje kod hash dla odpowiedniego obiektu. Metoda ta powinna być zgodna z metodą `equals()`.

G. (2pkt) Napisz metodę (zwykłą) `expand` (pol. rozbudowa) z argumentem typu `double`. Metoda powiększa pole `capacity` o wartość przekazaną przez argument. Jeśli po powiększeniu `capacity` będzie większe niż 1000.0, to ustaw je na 1000.0.

Zadanie 2. Klasa `Clinic` (pol. Klinika) (13pkt max.)

A. (1pkt) Klasa `Clinic` powinna być umieszczona w pakiecie `healthcare` w innym pliku niż klasa `Hospital`.

B. (2pkt) Klasa `Clinic` dziedziczy po klasie `Hospital`. Klasa powinna posiadać prywatne pola: - `specialization`, typu `String` (specjalizacja, np. kardiologia, ortopedia) - `rating`, typu `double` (ocena kliniki)

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:

- Specjalizacja nie powinna być pusta (równa `""`) i nullem - w przeciwnym wypadku ustaw na `"ogólna"`.
- Ocena musi być liczbą z zakresu od 0.0 do 5.0 - w przeciwnym wypadku ustaw ją jako 3.0.

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. Schemat:

`[NazwaKlasy]: Name: [name]. Location: [location]. Capacity: [capacity]. Specialization: [specialization]. Rating: [rating].`

lub jeśli nazwa nie jest ustalona (jest pustym napisem lub nullem):

`[NazwaKlasy]: Location: [location]. Capacity: [capacity]. Specialization: [specialization]. Rating: [rating].`

F. (2pkt) Nadpisz metodę (zwykłą) `expand` z argumentem typu `double`. Metoda powiększa pole `capacity` o wartość przekazaną przez argument. Jeśli po powiększeniu `capacity` będzie większe niż 500.0, to ustaw je na 500.0. Dodatkowo zwiększ ocenę o 0.5 (pamiętając ma maksymalnej możliwej górnej wartości).

G. (2pkt) Nadpisz metodę `equals`. Dwie kliniki są sobie "równe" wtedy i tylko wtedy, gdy mają takie same wszystkie pola. Nadpisz metodę `hashCode()`, która generuje kod 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 `TestHospital` (pol. klasa testująca) (9pkt max.)

A. (2pkt) Klasę `TestHospital` umieść w pakiecie tym samym co klasy z punktu w zadaniu 1 i 2, ale w innym pliku. 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 `Appliance` (pol. urządzenie) (5pkt max.)

A. (1pkt) Klasa `Appliance` powinna być umieszczona w pakiecie `electronics`.

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

- `brand` (marka), typu `String`.
- `model`, typu `String`.
- `powerUsage` (zużycie energii), typu `double` (w kilowatach).

C. (1pkt) W klasie `Appliance`, zaimplementuj statyczną metodę `createFridge(String brand, String model, double powerUsage)`. Metoda ma zwrócić nowy obiekt typu `Appliance`, którego pola ustawione są z argumentów metody.

D. (1pkt) W klasie `Appliance`, zaimplementuj nie-statyczną metodę `createAppliance(String brand, String model, double powerUsage)`. Metoda ma zwrócić nowy obiekt typu `Appliance`, którego pola ustawione są z argumentów metody.

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