Przykładowe kolokwium #1 - Zestaw P12

Ostatnia aktualizacja pliku: 18.11.2023 23:26.

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`
- `address` (adres zawierający ulicę, numer posesji, kod pocztowy i miejscowość), typ `String`
- `patients` (liczba pacjentów), typ `int`

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

- stringi nie mogą być puste lub równe null - wtedy ustaw adres na `"ul. Zdrowia 100, 00-001 Warszawa"` lub odpowiednio nazwę jako `"Centralny Szpital Kliniczny"`.
- liczba pacjentów musi być liczbą dodatnią, w przeciwnym wypadku ustaw ją na 200.

D. (1pkt) Napisz metody typu `getter` i `setter` dla wszystkich pól. Powinny mieć taką samą walidację jak w konstruktorze.

E. (2pkt) Nadpisz metodę `toString`:

```
[NazwaKlasy]: Name: [name]. Address: [address]. Number of patients: [patients].
```

F. (2pkt) Nadpisz metodę `equals` i `hashCode()`. Dwa szpitale są równe, gdy mają wszystkie pola takie same.

G. (1pkt) Napisz metodę `admitPatient` (pol. przyjmij pacjenta) z argumentem `int`. Metoda zwiększa `patients` o wartość argumentu. Jeśli liczba pacjentów przekroczy 1000, ustaw ją na 1000.

H. (2pkt) Napisz statyczną metodę `checkCapacity` (pol. sprawdź pojemność) z argumentem typu `Hospital`. Metoda wyświetla aktualną liczbę pacjentów i ile zostało do limitu 1000.

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` (np. kardiologia, neurologia)
- `doctors` typu `int` (liczba lekarzy)

C. (2pkt) Napisz pięcio-argumentowy konstruktor tej klasy. Kolejność argumentów powinna być taka sama jak w punkcie B (najpierw z klasy bazowej, potem z pochodnej). Dodatkowe warunki sprawdzające poprawność:

- `specialization` nie może być pusta - w przeciwnym wypadku ustaw jako `"ogólna"`.
- liczba lekarzy musi być liczbą nieujemną - w przeciwnym wypadku ustaw ją na 20.

D. (1pkt) Napisz metody typu `getter` i `setter` dla wszystkich pól. Zapewnij walidację taką samą jak w konstruktorze.

E. (2pkt) Nadpisz metodę `toString`:

```
[NazwaKlasy]: Name: [name]. Address: [address]. Number of patients: [patients].
Specialization: [specialization]. Number of doctors: [doctors].
```

F. (2pkt) Nadpisz metodę `admitPatient` z argumentem `int`. Zwiększ `patients` o wartość argumentu. Jeśli liczba pacjentów przekroczy 500, ustaw ją na 500. Dodatkowo zwiększ liczbę lekarzy o 20 (niezależnie od wartości argumentu).

G. (2pkt) Nadpisz metodę `equals` i `hashCode()`. Dwa obiekty są równe, gdy mają wszystkie pola takie same.

H. (1pkt) Zapewnij zgodność pozostałych metod z metodami z klasy bazowej i z tzw. dobrymi praktykami.

Zadanie 3. Klasa `TestHospital` (pol. klasa testująca dla A i B) (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 `Animal` (pol. zwierzę) (5pkt max.)

A. (1pkt) Stwórz abstrakcyjną klasę `Animal` zawierającą publiczną abstrakcyjną metodę `makeSound()`, która nie przyjmuje argumentów i zwraca `String`. Klasę umieść w pakiecie `zoo`.

B. (2pkt) Utwórz dwie klasy pochodne od `Animal`: `Dog` i `Cat`. W obu klasach nadpisz metodę `makeSound()`. Dla `Dog` niech zwraca `"Bark"`, a dla `Cat` `"Meow"`.

C. (2pkt) W klasie `TestAnimal` w pakiecie `zoo` utwórz tablicę typu `Animal` i zainicjalizuj ją 5 instancjami `Dog` i `Cat`. Iteruj po tablicy wywołując metodę `makeSound()` dla każdego zwierzęcia.