Przykładowe kolokwium #1 - Zestaw P14

Ostatnia aktualizacja pliku: 19.11.2023 17:50.

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 `School` (pol. Szkoła) (13pkt max.)

A. (1pkt) Klasa `School` powinna być umieszczona w pakiecie `education`.

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

- `name`, (nazwa szkoły), typ `String`
- `address`, (adres zawierający ulicę, numer posesji, kod pocztowy i miejscowość), typ `String`
- `students`, (liczba uczniów), typ `int`

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

- adres nie może być pusty lub równy null (równe `""`) - wtedy ustaw adres WMII czyli `"ul. Słoneczna 54, 10-710 Olsztyn`
- liczba uczniów musi być liczbą dodatnią, w przeciwnym wypadku ustaw ją na 100.
- 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 błędny 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 odbyć się do według schematu (zwróć uwagę na wielkość znaków i znaki interpunkcyjne, wszystko w jednej linii):

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

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

```
[NazwaKlasy]: Address: [address]. Number of students: [students].
```

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

G. (2pkt) Napisz metodę (zwykłą) `recruitment` (pol. rekrutacja) z argumentem typu `int`. Metoda powiększa pole `students` o wartość przekazaną przez argument. Jeśli po powiększeniu pole `students` będzie większe niż 500, to ustaw je na 500.

Zadanie 2. Klasa `University` (pol. uniwersytet) (13pkt max.)

A. (1pkt) Klasa `University` powinna być umieszczona w pakiecie `education` w innym pliku niż klasa `School`.

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

- `type` typu `String` (np. rodzaj np. rolniczy - agricultural, politechnika - university of technology, itp)
- `studies` typu `int` (liczba kierunków)

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:

- typ powinien nie być pusty (równy `""`) - w przeciwnym wypadku ustaw `"university of technology"`

- liczba kierunków musi być liczbą nieujemną - w przeciwnym wypadku ustaw ją jako 10.

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 odbyć się do według schematu (zwróć uwagę na wielkość znaków i znaki interpunkcyjne, zwróć uwagę na łamanie linii):

```
[NazwaKlasy]: Name: [name]. Address: [address]. Number of students: [students].
Type: [type]. Number of fields of study: [studies].
```

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

```
[NazwaKlasy]: Address: [address]. Number of students: [students].
Type: [type]. Number of fields of study: [studies].
```

F. (2pkt) Nadpisz metodę (zwykłą) `recruitment` z argumentem typu `int`. Metoda powiększa pole `students` o wartość przekazaną przez argument. Jeśli po powiększeniu pole `students` będzie większe niż 500, to ustaw je na 500. Dodatkowo zwiększ liczbę kierunków o 1/10 przekazanego argumentu (w zaokrągleniu lub obcięciu do liczby całkowitej).

G. (2pkt) Nadpisz metodę `equals`. Dwa obiekty są sobie "równe" wtedy i tylko wtedy, gdy mają ten sam adres oraz tą samą liczbę kierunków. 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 `TestSchool` (pol. klasa testująca dla szkoły) (9pkt max.)

A. (2pkt) Klasę `TestSchool` umieść bezpośrednio w katalogu `src` poza pakietami. 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 `Vehicle` (pol. pojazd) (5pkt max.)

A. (1pkt) Klasa `Vehicle` powinna być umieszczona w pakiecie `moto`.

B. (1pkt) Klasa powinna zawierać przynajmniej trzy atrybuty: `brand` (marka) typu `String`, `model` typu `String`, i `year` (rok produkcji) typu `int`.

C. (1pkt) W klasie `Vehicle`, zaimplementuj statyczną metodę `createCar(String brand, String model, int year)`. Metoda ma zwrócić nowy obiekt typu `Vehicle`, którego pola ustawione są z argumentów metody.

D. (1pkt) W klasie `Vehicle`, zaimplementuj nie-statyczną metodę `createVehicle(String brand, String model, int year)`. Metoda ma zwrócić nowy obiekt typu `Vehicle`, którego pola ustawione są z argumentów metody.

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