Kolokwium #1 - Programowanie obiektowe - Zestaw W15

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 `SportsFacility` (13pkt max.)

A. (1pkt) Klasa `SportsFacility` powinna być umieszczona w pakiecie `pl.sports`.

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

- `name`, (nazwa obiektu sportowego), typ `String`
- `location`, (lokalizacja zawierająca ulicę, numer posesji, kod pocztowy i miejscowość), typ `String`
- `capacity`, (pojemność), typ `int`

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 (równa `""`) lub równa `null` - wtedy ustaw lokalizację na `"ul. Sportowa 10, 00-001 Warszawa"`
- pojemność musi być liczbą dodatnią, w przeciwnym wypadku ustaw ją na 50.
- 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]: 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 obiekty sportowe są sobie "równe" wtedy i tylko wtedy, gdy mają tę samą lokalizację. Nadpisz metodę `hashCode()`, która generuje wartość hash dla odpowiedniego obiektu. Metoda ta powinna być zgodna z metodą `equals()`,

G. (2pkt) Napisz metodę (zwykłą) `expand` (pol. rozbudowa) z argumentem typu `int`. Metoda powiększa pole `capacity` o wartość przekazaną przez argument. Jeśli po powiększeniu pole `capacity` będzie większe niż 1000, to ustaw je na 1000.

Zadanie 2. Klasa `Stadium` (pol. boisko) (13pkt max.)

A. (1pkt) Klasa `Stadium` powinna być umieszczona w pakiecie `pl.sports` w innym pliku niż klasa `SportsFacility`.

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

- `surfaceType` typu `String` (np. rodzaj nawierzchni np. trawiasta, sztuczna)
- `sportsTypes` typu `int` (liczba dyscyplin sportowych, które można uprawiać na boisku)

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 nawierzchni nie powinien być pusty (równy `""`) - w przeciwnym wypadku ustaw ją `"trawiasta"`
- liczba dyscyplin sportowych musi być liczbą nieujemną - w przeciwnym wypadku ustaw ją jako 1.

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]: Name: [name]. Location: [location]. Capacity: [capacity].
Surface Type: [surfaceType]. Number of sports: [sportsTypes].
```

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

```
[NazwaKlasy]: Location: [location]. Capacity: [capacity].
Surface Type: [surfaceType]. Number of sports: [sportsTypes].
```

F. (2pkt) Nadpisz metodę (zwykłą) `expand` z argumentem typu `int`. Metoda powiększa pole `capacity` o wartość przekazaną przez argument. Jeśli po powiększeniu pole `capacity` będzie większe niż 1500, to ustaw je na 1500. Dodatkowo zwiększ liczbę dyscyplin sportowych o 5.

G. (2pkt) Nadpisz metodę `equals`. Dwa boiska są sobie "równe" wtedy i tylko wtedy, gdy mają tę samą lokalizację oraz tę samą liczbę dyscyplin sportowych. 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 `TestSportsFacility` (pol. klasa testująca dla obiektu sportowego) (9pkt max.)

A. (2pkt) Klasę `TestSportsFacility` 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 sposobów.

Zadanie 4. Klasa `Product` (pol. produkt) (5pkt max.)

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

B. (2pkt) Utwórz dwie klasy pochodne od `Product`: `Book` i `Clothing`. W obu klasach nadpisz metodę `getPrice()`. Dla `Book` niech zwraca cenę 29.99, a dla `Clothing` niech zwraca cenę 59.99.

C. (2pkt) W klasie `TestProduct` w pakiecie `store` utwórz tablicę typu `Product` i zainicjalizuj ją 5 instancjami `Book` i `Clothing`. Iteruj po tablicy wywołując metodę `getPrice()` dla każdego produktu (wyświetl ceny na standardowym wyjściu).