ZOO

Treść zadania

Stwórz:

- abstrakcyjną klasę `Animal`, która posiada 3 atrybuty: `int id`[1], `string name` oraz `float weight`. Klasa ma zawierać metodę `void introduce()`, która ma "przedstawiać dane zwierzę" (patrz oczekiwany output),
- interfejs `Canine` z jedną metodą `void bark()`,
- klasę `Bear`, która dziedziczy po klasie `Animal` i ma dodatkowy atrybut `int furLength`,
- klasę `Tiger`, która dziedziczy po klasie `Animal` i ma dodatkowy atrybut `int clawLength`,
- klasę `Wolf`, która dziedziczy po klasie `Animal`, implementuje interfejs `Canine` i wywołując jego metodę ma wypisać "BARK BARK BARK". Pondato, `Wolf` ma dodatkowy atrybut `int fangLength`,
- klasę `Dog`, która dziedziczy po klasie `Animal`, implementuje interfejs `Canine` i wywołując jego metodę ma wypisać "bark bark bark". Pondato, `Dog` ma dodatkową metodę `void sitPretty()`, która wypisuje na standardowe wyjście "(name)[2] sits pretty.".

Następnie, w metodzie `main` wykonaj następujące operacje:

- stwórz instancję klasy `Bear` - imię: Yogi, waga: 200, długość futra: 40,
- stwórz instancję klasy `Tiger` - imię: Jataka, waga: 150, długość pazura: 25,
- stwórz instancję klasy `Wolf` - imię: Howler, waga: 70, długość futra: 40,
- stwórz instancję klasy `Dog` - imię: Scooby, waga: 30,
- stwórz listę `animals` i dodaj tam wszystkie zwierzęta. Dla każdego obiektu wypisz jego `id`, a następnie ma się ono przedstawić,
- stwórz listę `howlers` i dodaj tam Scooby'ego oraz Howlera. Dla każdego obiektu w `howlers`, wypisz "My name is (name) and I am barking ", następnie wywołaj metodę `howl`. Potem, jeśli to możliwe, wywołaj metodę `sitPretty()` dla każdego obiektu.

Oczekiwany output:

```
1 : I'm a bear. My name is Yogi. I weigh 200.0 kg and my fur length is 40.
2 : I'm a tiger. My name is Jataka. I weigh 150.0 kg and my claw length is 25.
3 : I'm a wolf. My name is Howler. I weigh 70.0 kg and my fang length is 40.
4 : I'm a dog. My name is Scooby. I weigh 30.0 kg.
My name is Howler and I am barking: BARK BARK BARK
My name is Scooby and I am barking: bark bark bark
Scooby sits pretty.
```

Uwagi

1. Należy zadbać o podstawowe zasady programowania obiektowego - enkapsulację, gettery, settery, odpowiednią dostępność zmiennych... Te rzeczy nie są uwzględnione w treści zadania, ale są to dobre praktyki, które należy stosować.

[1] Odpowiedzialnością tej klasy jest przydzielanie każdemu nowo stworzonemu zwierzęciu id - żadna z klas pochodnych nie może go modyfikować

[2] (name) jest tutaj imieniem psa

Konta bankowe

Wprowadzenie

Jak już wiemy, **klasy wewnętrzne** w Javie przydają się na przykład wtedy, gdy chcemy by niektóre klasy były dostępne tylko w obrębie jednej, więszkej. Najczęściej mówimy o 3 typach klas wewnętrznych: statyczne/dynamiczne, lokalne oraz anonimowe.

Treść zadania

Zależy nam na symulowaniu przyrostu gotówki na naszych kontach bankowych. Każde konto bankowe ma swój aktualny balans oraz w jakis sposób korzysta ze swojego "zliczacza" implementującego `ActionListener`, nazywajmy go `BalanceCounter`. Jego zadaniem jest co sekundę zwiększać stan konta o zadane oprocentowanie.
Aplikacja ma wyświetlić proste okno utworzone w Swingu z guzikiem umożliwiającym jej zatrzymanie. Aby umożliwić równoległe naliczanie odsetek, w aplikacji ma chodzić `Timer`, który co sekundę odpali swojego listenera `BalanceCounter`.
W związku z tym stwórz 3 klasy:

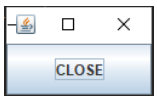
- `BankAccount1`, dla której `BalanceCounter` ma być **dynamiczną klasą wewnętrzną**, jej instancję stwórz własnoręcznie w mainie.
- `BankAccount2`, dla której `BalanceCounter` jest instancją **klasy lokalnej** wewnątrz metody `createBalanceCounter(double interestRate)`. Jej instancję otrzymaj w mainie przez użycie właśnie tej metody.
- `BankAccount3`, dla której `BalanceCounter` jest instancją **klasy anonimowej**, zwracaną przez metodę `createBalanceCounter()`. Jej instancję otrzymaj w mainie przez użycie właśnie tej metody.

Oczekiwany output dla początkowej wartości 1000 i oprocentowania 0.2 (liczenie aż do zatrzymania):

```
Changing account balance from 1000 to: 1200
Changing account balance from 1200 to: 1440
Changing account balance from 1440 to: 1728
Changing account balance from 1728 to: 2073
Changing account balance from 2073 to: 2487
Changing account balance from 2487 to: 2984
Changing account balance from 2984 to: 3580
```

Uwagi

1. Należy zadbać o podstawowe zasady programowania obiektowego - enkapsulację, gettery, settery, odpowiednią dostępność zmiennych... Te rzeczy nie są uwzględnione w treści zadania, ale są to dobre praktyki, które należy stosować.
2. Prezentację każdego z trzech zadań należy przeprowadzić oddzielnie. Podczas prezentacji można przyjąć, że każde konto zaczyna z balansem 1000, a każdy `BalanceCounter` będzie naliczał oprocentowanie 0.2. Oczywiście każda z tych wartości powinna być modyfikowalna.
3. Okno tworzone w Swingu powinno wyglądać podobnie do zaprezentowanego poniżej.



4. Wskazówka: jedynie `BankAccount3` przechowuje oprocentowanie jako atrybut, w `BankAccount2` oprocentowanie jest parametrem metody, a w przypadku pierwszym, oprocentowanie ma być trzymane jako atrybut klasy wewnętrznej.

Restauracja

Treść zadania

Celem zadania jest stworzenie symulacji prężnie działającej restauracji. Restauracja zaczyna z pewnym zasobem dań, co pewien interwał przygotowuje kolejne dania, a w międzyczasie klienci składają zamówienia. Problemem jest fakt, że głodny tłum składa zamówienia w nierównych odstępach czasu, a kucharz nie zawsze jest w stanie ugotować wystarczająco dużo posiłków. Wtedy restauracja traci pieniądze z niewykonanego zamówienia. Chcemy sprawdzić, jak dużo restauracja zarabia w stosunku do tego, ile traci na nieprzygotowanych daniach. Menu symulowanej restauracji wygląda następująco:

- Lody - 8pln,
- Gyros - 12pln,
- Tortilla - 13pln,
- Pizza - 18pln,
- Stek - 30pln.

Rozwiązanie wzorcowe składa się z 4 klas:

- `Restaurant`, która przechowuje informacje o aktualnych zasobach, zarobionych oraz straconych pieniądzach.
- `HungryCrowd`, która odpowiada za składanie zamówień do restauracji.
- `Dish`, która jest enumem reprezentującym odpowiednie danie z menu.
- `Main`, która tworzy obiekty restauracji, tłumu i uruchamia ich symulację.

Przykładowa konfiguracja:

- Restauracja zaczyna z 3 daniami każdego rodzaju. Co 5 sekund swojego cyklu przygotowuje po 1 daniu z każdego rodzaju.
- Tłum składa zamówienia co 1-4 sekund. Zamówienie to jeden rodzaj dania zamówiony 2-6 razy.

Przykładowy output:

```
Successfully ordered 5 Pizza
Couldn't order 2 Pizza
Money earned|lost: 90|36 Current stock: 6 Steak 6 Tortilla 1 Pizza 6 IceCream 6 Gyros
Successfully ordered 4 Gyros
Couldn't order 2 Pizza
Couldn't order 5 Pizza
Money earned|lost: 138|162 Current stock: 7 Steak 7 Tortilla 2 Pizza 7 IceCream 3 Gyros
Couldn't order 5 Gyros
Successfully ordered 2 Pizza
Couldn't order 3 Pizza
Money earned|lost: 174|276 Current stock: 8 Steak 8 Tortilla 1 Pizza 8 IceCream 4 Gyros
```

Uwagi

1. Rozwiązanie wzorcowe działa na 3 wątkach - głównym wątku aplikacji, wątku restauracji przygotowującej kolejne dania i wątku zamawiającego tłumu. Wskazówka: potrzeba 2 timerów.
2. Treść zadania podaje przykładowe wartości szybkości pracy kucharza oraz szybkości składania zamówień. Rozwiązanie wzorcowe daje możliwość konfigurowania tych wartości.
3. Przy nieodpowiedniej implementacji może dojść do kolizji wątków - 2 osobne wątki mogą mieć dostęp do modyfikacji/odczytywania danych o aktualnym stanie restauracji. Wskazówka: `synchronized`.

Blog

Treść zadania

Chcemy stworzyć architekturę dla bloga internetowego. Blog ma swoich użytkowników, którzy mogą tworzyć posty i komentować posty innych użytkowników. Potrzebujemy następujących klas:

- `Blog`, który zawiera listę użytkowników i postów. Ponadto ma metody umożliwiające dodanieg nowego posta, skomentowanie posta i wyświetlenie wszystkich wpisów dla danego użytkownika.
- `User`, który zawiera id, imię, nazwisko oraz nick użytkownika.
- `Post`, który zawiera id, datę dodania, autora, zawartość oraz listę komentarzy.
- `Comment`, który zawiera id, datę dodania, autora oraz zawartość.

Jako że `Post` różni się od `Comment` tylko tym, że posiada komentarze, to zgodnie z zasadą DRY chcemy, aby obie te klasy dziedziczyły z innej, abstrakcyjnej klasy.
Ponadto, id dla każdego użytkownika i wpisu ma być unikalne - stworzona architektura powinna ustawiać nowo tworzonym obiektom kolejne id.

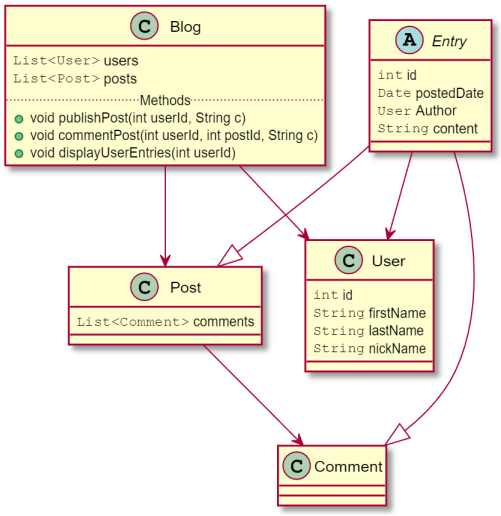
Przykładowy output:

```
janek95 posted just now: Żyrafa najwyższą formą życia.
xxmichuxx posted just now: Czas nie zawsze przemija w jednakowym rytmie. To my o tym decydujemy.
marian_dev posted just now: Jaki jest obiektowy sposób na zdobycie fortuny? Dziedzieczenie!
Cannot publish post - user of id 100 doesn't exist.

marian_dev commented janek95's post: Najwyższą formą życia jest żyrafa stojąca na taborecie!
janek95 commented xxmichuxx's post: To bardzo prawdziwe. Chyba cytat z Paulo Coelho.
xxmichuxx commented xxmichuxx's post: Mądre słowa.
janek95 commented marian_dev's post: Tak, a grupa krwi programisty to C++!
xxmichuxx commented marian_dev's post: Haha, usmialem sie!

janek95's entries:
(POST) Żyrafa najwyższą formą życia.
(COMMENT) To bardzo prawdziwe. Chyba cytat z Paulo Coelho.
(COMMENT) Tak, a grupa krwi programisty to C++!
```

Diagram UML



Uwagi

1. Należy zadbać o podstawowe zasady programowania obiektowego - enkapsulację, gettery, settery, odpowiednią dostępność zmiennych... Te rzeczy nie są uwzględnione w treści zadania ani diagramie, ale są to dobre praktyki, które należy stosować.

Wojna

Treść zadania

Chcemy zasymulować różne rodzaje gier karcianych, podobnych do klasycznej wojny. Każda karta ma 3 własności:

- **Wartość** - liczbę całkowitą z zakresu 1-10.
- **Kolor** - jeden spośród 4. Na potrzeby zadania przyjmiemy wartości kolorów: trefl - 1, karo - 2, kier - 3, pik - 4.
- **Rzadkość** - karta może być rzadka albo nie.

Każdy rodzaj wojny powinniśmy móc przeprowadzić ze śledzeniem jej przebiegu na żywo. Trudność zadania polega na tym, że prawdopodobnie w przyszłości powstanie dużo więcej rodzajów wojen (patrz wskazówka). Wynik wojny reprezentujemy przez 3 wartości: 1 gdy wygrała talia pierwsza, -1 gdy wygrała talia druga oraz 0 w przypadku remisu. Póki co, mamy 3 rodzaje wojen między taliami kart:

- `RarityBattle` - wygrywa ta talia, która posiada więcej kart rzadkich. W przypadku tej samej liczby w obu taliach, dochodzi do remisu.
- `ColorBattle` - wygrywa ta talia, której suma wartości kolorów wszystkich kart jest większa . W przypadku tej samej wartości w obu taliach, dochodzi do remisu.
- `NormalBattle` działa jak w typowej grze karcianej wojna: bierzemy pierwszą kartę z obu talii, a następnie porównujemy ich wartości. Kartę, która przegrała, odrzucamy, a wygrana będzie porównywana z następną kartą przeciwnej armii. Jeśli mieliśmy remis, obie karty są odrzucane (i bierzemy kolejne z obu talii). Wygrywa ta talia, w której na końcu zostanie choć jedna karta.

W zadaniu będziemy randomizować talie kart. Każda talia może być jednoznacznie opisana przez liczbę kart oraz ziarno losowania obiektu klasy `Random`. Talie powinny być randomizowane w następujący sposób[1]:

1. Tworzenie obiektu `Random` o zadanym seedzie.
2. Dla każdej karty:
   (a) Wylosowanie wartości z przedziału 1-10 przy użyciu stworzonego obiektu.
   (b) Wylosowanie koloru (spośród wcześniej wymienionych) przy użyciu stworzonego obiektu.
   (c) Wylosowanie `boolean`'a (odpowiadającego rzadkości karty) przy użyciu stworzonego obiektu.

Oczekiwany output:

```
Deck with 5 cards, seed 10: (4, 2, false) (1, 1, true) (8, 2, true) (5, 2, true) (2, 4, true)
Deck with 5 cards, seed 15: (2, 4, false) (8, 4, false) (10, 1, true) (1, 3, false) (6, 3, false)
Sizes 5, seeds 10 15, color battle result: -1
Sizes 5, seeds 15 10, color battle result: 1
Sizes 5, seeds 30 30, color battle result: 0
Sizes 5, seeds 10 15, rarity battle result: 1
Sizes 5, seeds 15 10, rarity battle result: -1
Sizes 5, seeds 30 30, rarity battle result: 0
Sizes 5, seeds 10 15, normal battle result: 1
Sizes 5, seeds 15 10, normal battle result: -1
Sizes 5, seeds 30 30, normal battle result: 0
```

Uwagi

1. Należy zadbać o podstawowe zasady programowania obiektowego - enkapsulację, gettery, settery, odpowiednią dostępność zmiennych... Te rzeczy nie są uwzględnione w treści zadania, ale są to dobre praktyki.
2. **Wskazówka**: użyć interfejsu.

[1] Jest to istotne, by móc porównywać output z oczekiwanym. Inne sposoby losowania są poprawne, lecz mogą dać inne wyniki

Trójbój siłowy

Treść zadania

Celem zadania jest symulacja treningu trójboistycznego. Jak wiemy, trójbój siłowy (powerlifting) składa się z 3 dyscyplin - przysiad ze sztangą (squat - SQ), wyciskanie na ławce płaskiej (bench press - BP) oraz martwy ciąg (deadlift - DL). Każdy trójboista w naszym zadaniu będzie reprezentowaney przez imię, nazwisko oraz wartość całkowitą odpowiadającą jego sile w każdym boju. Dla każdego trójboisty zaimplementuj metodę `void train(int time)`, która będzie odpowiadała za trening zawodnika w godzinach. Zawodnik zawsze chce trenować optymalnie - poprawiać słabe ogniwa. Każda godzina rutyny treningowej wygląda następująco:

1. Sprawdzenie, czy trójboista dozna kontuzji.
   (a) Jeśli nie, trening jest kontynuowany.
   (b) W przeciwnym przypadku - należy wypisać informację o zaistniałej kontuzji, a trwający trening jest przerwany.
2. Sprawdzenie, czy trójboista może jeszcze zwiększyć swoje punkty siły.
   (a) Jeśli nie, trening jest przerwany, a stosowna informacja wypisywana.
   (b) W przeciwnym przypadku - trening jest kontynuowany.
3. Sprawdzenie, w którym boju trójboista ma najmniej punktów siły.
4. Inkrementacja punktów siły w najsłabszym boju[1] o 1 oraz wypisanie odpowiedniej informacji.

W zadaniu chcemy mieć 2 typy trójboistów: `RawPowerlifter`, który wykonuje ćwiczenia bez dodatkowego sprzętu oraz `EquippedPowerlifter`, który korzysta ze sprzętu. Rozróżnienie jest tu istotne, ponieważ ćwiczenie bez sprzętu daje mniejsze wyniki siłowe, co przekłada się na większe przeciążenia w przypadku ćwiczenia ze sprzętem.

- `RawPowerlifter` może ćwiczyć każdy z bojów do 20 punktów siły, a szansa na kontuzję dla każdej godziny treningu to 5%.
- `EquippedPowerlifter` może ćwiczyć każdy z bojów aż do 45 punktów siły, ale szansa na kontuzję dla każdej godziny treningu to 15%.

W metodzie `main`:

1. Stwórz obiekty trójboistów obu typów (powinny być trzymane w jednej kolekcji).
2. Wywołaj dla każdego z nich metodę `void train(int time)`.

Przykładowy output dla Robert Piotrkowicz(15, 18, 12) oraz Blaine Summer (39, 45, 40):

```
Robert Piotrkowicz SQ: 15 BP: 18 DL: 13 trained deadlift.
Robert Piotrkowicz SQ: 15 BP: 18 DL: 14 trained deadlift.
Robert Piotrkowicz SQ: 15 BP: 18 DL: 15 trained deadlift.
Robert Piotrkowicz SQ: 16 BP: 18 DL: 15 trained squat.
Robert Piotrkowicz SQ: 16 BP: 18 DL: 16 trained deadlift.
Robert Piotrkowicz SQ: 17 BP: 18 DL: 16 trained squat.

Blaine Summer SQ: 40 BP: 45 DL: 40 trained squat.
Blaine Summer SQ: 40 BP: 45 DL: 40 sustained an injury and cannot train now.
```

[1] W przypadku kilku bojów o tej samej, najniższe wartości, przyjmujemy kolejność: przysiad, wyciskanie, martwy ciąg.

Uwagi

1. Należy zadbać o podstawowe zasady programowania obiektowego - enkapsulację, gettery, settery, odpowiednią dostępność zmiennych... Te rzeczy nie są uwzględnione w treści zadania, ale są to dobre praktyki, które należy stosować.
2. Rozwiązanie wzorcowe składa się z 4 klas - `Main`, abstrakcyjnej klasy `Powerlifter`, implementującej metodę treningu, oraz 2 klasach odpowiednio dziedziczących po niej.

Kantor wymiany walut

Treść zadania

Celem zadania jest stworzenie symulacji działania kantoru wymiany walut. Kantor w naszym zadaniu będzie miał pewien zasób każdej z 7 walut z klasy `Currency`. Każdy nowo powstały kantor ma tyle samo każdej z nich. Ponadto, będzie miał 2 metody:

- `void showFunds()`, która wypisuje dostępne środki dla wszystkich walut.
- `void exchange(double amount, Currency from, Currency to)`, która odpowiada za faktyczną wymianę: kantor wymienia `amount` waluty `from` na odpowiednią ilość waluty `to`. Jeszcze przed samą transakcją pobiera prowizję w wysokości 20%, czyli de facto wymienia tylko 0.8* `amount`.

Jako że kursy walut są rzeczą, która zmienia się bardzo dynamicznie, będziemy łączyć się z Internetem, by pobierać możliwie aktualne. W tym celu:

1. Dodaj do projektu 2 dostarczone klasy: `CurrencyConverter` oraz `Currency`. Pamiętaj o zmianie paczki.
2. Stwórz klasę `ExchangeOffice` z funkcjonalnościami opisanymi wyżej. Sprawdzenie kursu danej waluty jest dostępne przez statyczną metodę `CurrencyConverter.convert(Currency from, Currency to)`.
3. Stwórz kantor z wartością początkową 1000 i dokonaj 3 wymian:
   - 1000 PLN → EUR
   - 250 PLN → RUB
   - 500 JPY → USD

Output w dniu 9.02.2019:

```
Exchanging 1000.0 PLN to 185.3744 EUR
Current funds: 1000.0 GBP 1000.0 JPY 814.6256 EUR 1000.0 CZK 1000.0 USD 2000.0 PLN 1000.0 RUB

Cannot exchange 250.0 PLN to 3438.7034 RUB : insufficient funds.
Current funds: 1000.0 GBP 1000.0 JPY 814.6256 EUR 1000.0 CZK 1000.0 USD 2000.0 PLN 1000.0 RUB

Exchanging 500.0 JPY to 3.6452 USD
Current funds: 1000.0 GBP 1500.0 JPY 814.6256 EUR 1000.0 CZK 996.3548 USD 2000.0 PLN 1000.0 RUB
```

Uwagi

1. Należy zadbać o podstawowe zasady programowania obiektowego - enkapsulację, gettery, settery, odpowiednią dostępność zmiennych... Te rzeczy nie są uwzględnione w treści zadania, ale są to dobre praktyki, które należy stosować.

Portale internetowe

Treść zadania

Celem zadania jest stworzenie aplikacji symulującej pracę reportera. Reporter pracuje dla róznych portali internetowych i wysyła do wszystkich z nich świeże newsy. Ten rynek jest bardzo konkurencyjny - różne typy portali publikują otrzymane newsy na swój sposób i zbierają za to punkty. Portale otrzymują punkty odpowiednio za:

- `VowelWebsite` - 1 punkt za samogłoskę.
- `ConsonantWebsite` - 1 punkt za spółgłoskę.
- `CustomWebsite` - 1 punkt za literę z zakresu a-k (zarówno wielką, jak i małą).

Wszystkie typy portali cechuje ich suma punktów oraz nazwa. Kiedy portal otrzyma jakąś wiadomość, powinien wyświetlić jej treść, a wszystkie punktowane dla niego litery powinny być kapitalizowane.

Klasa `Reporter` ma posiadać funkcjonalność `broadcastMessage(String msg)`, która przesyła wiadomość do wszystkich portali, dla których dany reporter pracuje. W klasie `Main` należy stworzyć obiekt reportera, który współpracuje z różnymi typami portali i wywołać dla niego metodę `broadcastMessage`.

Przykładowo, wartość zdania "Cows lose their jobs as milk prices drop" dla typów portali wymienionych powyżej to odpowiednio: 11, 29 i 14. Przykładowy output:

```
(Vowel news, 11) informs: cOws lOsE thEIr jObs As mIlk prIcEs drOp
(The Consonants, 29) informs: CoWS LoSe THeiR JoBS aS MiLK PRiCeS DRoP
(Custom Feed, 14) informs: Cows losE tHEIr JoBs As mIlK prICEs Drop

(Vowel news, 21) informs: sAfEty mEEtIng Ends In AccIdEnt
(The Consonants, 50) informs: SaFeTY MeeTiNG eNDS iN aCCiDeNT
(Custom Feed, 30) informs: sAFEty mEEtInG EnDs In ACCIDEnt
```

Uwagi

1. Należy zadbać o podstawowe zasady programowania obiektowego - enkapsulację, gettery, settery, odpowiednią dostępność zmiennych... Te rzeczy nie są uwzględnione w treści zadania, ale są to dobre praktyki, które należy stosować.
2. **Wskazówka**: użyj jednego z wzorców czynnościowych (behavioral patterns).

Wypożyczalnia rowerów

Treść zadania

Celem zadania jest zasymulowanie działania wypożyczalni rowerów. Wypożyczalnia przechowuje informację o aktualnym przychodzie i zwiększa jego wartość wraz z wypożyczanymi rowerami. Metoda `orderBike` w klasie wypożyczalni przyjmuje jako argumenty wszystkie 3 parametry charakteryzujące rower i długość okresu wypożyczenia. Metoda zwraca obiekt roweru z ustaloną już ceną. Wyznaczenie ceny roweru należy do obowiązków wypożyczalni.

Każdy rower jest identyfikowany przez swój kolor, grubość opony w milimetrach oraz ramę - może być stalowa lub aluminiowa. Koszt wypożyczenia roweru jest liczony zgodnie ze wzorem:

$$a \, \frac{b\left(c + \frac{d}{3}\right)}{20}$$

- a - okres wypożyczenia [dni]
- $b = \begin{cases} 1 & \text{dla roweru z ramą aluminiową,} \\ 2 & \text{dla roweru z ramą stalową} \end{cases}$
- $c = \begin{cases} 500 & \text{dla koloru czerwonego,} \\ 250 & \text{dla koloru niebieskiego,} \\ 100 & \text{dla koloru zielonego} \end{cases}$
- d - szerokość opony [milimetry]

Po utworzeniu odpowniej architektury, w klasie `Main`:

1. Stwórz obiekt wypożyczalni.
2. Wypożycz 3 rowery przy użyciu metody `orderBike` i wypisuj charakterystykę każdego z nich:
    - Czerwony z ramą aluminiową, szerokością opony 25mm, na 15 dni.
    - Zielony z ramą stalową, szerokością opony 85mm, na 40 dni.
    - Niebieski z ramą aluminiową, szerokością opony 43mm, na 20 dni.
3. Wypisz całkowity dochód wypożyczalni.

Oczekiwany output:

```
Ordered for 15 days: (RED, 25, true) -> 762.50
Ordered for 40 days: (GREEN, 85, false) -> 256.67
Ordered for 20 days: (BLUE, 43, true) -> 528.67
Rental income: 1547.83
```

Uwagi

1. Należy zadbać o podstawowe zasady programowania obiektowego - enkapsulację, gettery, settery, odpowiednią dostępność zmiennych... Te rzeczy nie są uwzględnione w treści zadania, ale są to dobre praktyki, które należy stosować.
2. Kolor roweru powinien być reprezentowany przez enum.

Biuro podróży

Treść zadania

Celem zadania jest zasymulowanie działania biura podróży. Biuro podróży ma swojej ofercie bazę wycieczek[1] i sprzedaje wycieczki klientom (sprzedana wycieczka jest usuwana z bazy dostępnych). Każda wycieczka jest reprezentowana przez:

- `float` Cenę za noc hotelową.
- `int` Liczbę nocy.
- `Continent` [2] Cel wycieczki.
- `boolean` Wartość wskazującą, czy na wycieczkę można zabrać ze sobą psa.

Klasa reprezentująca biuro podróży ma zawierać metodę `bookOffer`, przyjmującą 4 argumenty i zwracającą `boolean` informujący, czy udało się znaleźć odpowiewdnią ofertę. Argumenty dla metody `bookOffer` to:

- `float` Maksymalna cenę za noc hotelową, na którą może sobie pozwolić klient
- `int` Minimalna liczba nocy, którą klient chce spędzić w hotelu.
- `List<Continent>` Destynacje wycieczek, które zaakceptuje klient.
- `boolean` Wartość wskazującą, czy na klient chce wyjechać na wycieczkę z psem.

Procedura szukania ofert dla klienta przebiega następująco: biuro przeszukuje liniowo wszystkie dostępne oferty i gdy jedna z ofert spełnia wymogi, jest automatycznie usuwana z bazy i zostaje zwracana informacja o pomyślnym jej znalezieniu- `true`. Gdy jednak żadna z dostępnych ofert nie spełnia wymogów, zwracamy `false`.

Przykładowy output:

```
(219.99 * 7, EU, false) (349.99 * 10, SA, true) (290.99 * 14, AS, false) (199.99 * 8, AF, true)

Result for (200.00 * 6, [EU, AS, AF], false) is: true
(219.99 * 7, EU, false) (349.99 * 10, SA, true) (290.99 * 14, AS, false)

Result for (300.00 * 10, [EU, AS, AF], true) is: false
(219.99 * 7, EU, false) (349.99 * 10, SA, true) (290.99 * 14, AS, false)

Result for (500.00 * 10, [AF, SA], true) is: true
(219.99 * 7, EU, false) (290.99 * 14, AS, false)
```

Uwagi

1. Należy zadbać o podstawowe zasady programowania obiektowego - enkapsulację, gettery, settery, odpowiednią dostępność zmiennych... Te rzeczy nie są uwzględnione w treści zadania, ale są to dobre praktyki, które należy stosować.

[1] Zalecane jest stworzenie osobnej klasy `TripOffer` zawierającej informację o wycieczce.

[2] Typ wyliczeniowy reprezentujący poszczególne kontynenty. W naszej aplikacji dostępne są tylko 4 kontynenty: Europa(EU), Azja(AS), Afryka(AF) oraz Ameryka Południowa(SA).

Dealer samochodowy

Treść zadania

Przykładowy output:

```
asd
```

Uwagi

1. Należy zadbać o podstawowe zasady programowania obiektowego - enkapsulację, gettery, settery, odpowiednią dostępność zmiennych... Te rzeczy nie są uwzględnione w treści zadania, ale są to dobre praktyki, które należy stosować.

Listy 1

Zadanie 1 - mieszkania

Stwórz klasę `Apartment`, która będzie zawierała informacje o **mieście**, w którym się znajduje, **powierzchni** w metrach kwadratowych, oraz **cenie za metr** mieszkania. Ponadto, klasa ma zawierać metodę `float getFullPrice()`, która zwraca cenę za mieszkanie przemnożoną przez 0.95 (rabat od dewelopera). Następnie w metodzie `main` stwórz listę mieszkań typu `LinkedList` i wypisz dla każdego z nich miasto, w którym się znajduje, oraz cenę od dewelopera. Wyznacz średnią cenę mieszkań. **Uwaga:** wszystkie wartości liczbowe w tym zadaniu wypisz, zaokrąglając do dwóch miejsc po przecinku. Przykładowy output:

```
1. Apartment in Warsaw costs 771875.00.
2. Apartment in Cracow costs 779000.00.
3. Apartment in Gdansk costs 855000.00.
Mean price is 801958.31
```

Zadanie 2 - Jacek

Chcemy wybrać najlepsze liceum dla Jacka. Jacek ma 180 punktów rekrutacyjnych i może pójść do szkoły, która będzie oddalona maksymalnie o 10km od jego domu. W tym celu, stwórz klasę `HighSchool`, która będzie zawierała **nazwę**, **próg punktowy** (wartość całkowita z przedziału 0-200) oraz **odległość** liceum od Jacka w kilometrach. Następnie w metodzie `main` stwórz listę liceów typu `ArrayList` i wypisz informacje o każdym z nich, używając pętli for each. Dopasuj najlepsze[1] liceum, do którego może pójść Jacek. W przypadku, gdy w liście nie znajduje się liceum, do którego Jacek może aplikować, wypisz stosowny komunikat. Przykładowy output:

```
LO im. Jana Zamoyskiego needs 173 pts and is 15 km away.
LO im. Mikołaja Kopernika needs 193 pts and is 7 km away.
LO im. Batalionu "Zośka" needs 122 pts and is 6 km away.
Apply for LO im. Batalionu "Zośka".
```

Zadanie 3 - książki

Stwórz klasę `Book`, która będzie zawierała informacje o **tytule**, **autorze**, **liczbie stron** oraz **dacie wydrukowania** książki. Ponadto, klasa ma zawierać metodę `long getDays()`, która zwraca wiek książki w dniach (obchodzi nas przedział czasowy od wydrukowania książki do dziś). Następnie w metodzie `main` stwórz listę książek typu `ArrayList` i wypisz informacje o każdej z nich oraz jej wiek w dniach. Wypisz tytuł najstarszej z nich (w przypadku kilku najstarszych, wypisz dowolną). Przykładowy output[2]:

```
1. book "Data science od podstaw. Analiza danych w Pythonie" is 142 days old.
2. book "HTML i CSS. Zaprojektuj i zbuduj witrynę WWW." is 536 days old.
3. book "MATEMATYKA ZADANIA MATURALNE I EGZAMINACYJNE" is 9080 days old.
The oldest book is "MATEMATYKA ZADANIA MATURALNE I EGZAMINACYJNE".
```

Uwagi

1. Należy zadbać o podstawowe zasady programowania obiektowego - enkapsulację, gettery, settery, odpowiednią dostępność zmiennych... Te rzeczy nie są uwzględnione w treści zadania, ale są to dobre praktyki, które należy stosować.

[1] Przez najlepsze liceum rozumiemy to o najwyższym progu punktowym.
[2] Przykładowy output nie zawiera wszystkich informacji o książkach ze względu na przejrzystość dokumentu.

Kolekcje 1

Zadanie 1 - podróżnik

Stwórz klasę `Traveler`, która będzie zawierała informacje o **nazwie** podróżnika oraz o **odwiedzonych przez niego miastach**. Kolekcja opisująca odwiedzone przez podróżnika miasta **nie może dopuszczać duplikatów** - każde miasto może tam występować tylko raz. Ponadto, klasa ma zawierać metodę `void visit(String city)`, która doda do kolekcji odwiedzonych miast to z argumentu (o ile nie było jeszcze odwiedzone). Następnie w metodzie `main` stwórz instancję `Traveler` i wywołaj na niej metodę `visit` wielokrotnie dla różnych miast. Wypisz wszystkie odwiedzone przez podróżnika miasta i upewnij się, czy nie ma tam duplikatów. Przykładowy output:

```
Robert Makłowicz has already been in:
* Warsaw
* New York
* Brasilia
* Moscow
```

Zadanie 2 - pamiętnik

Chcemy stworzyć bardzo prosty pamiętnik. W tym celu, stwórz klasę `DiaryNote`, która będzie zawierała informacje o **humorze**, **pogodzie** oraz **liczbie godzin**, przez które bawił się właściciel pamiętnika danego dnia. Następnie w metodzie `main` stwórz kolekcję, która będzie reprezentowała nasz dzienniczek. Ta kolekcja musi w łatwy sposób umożliwić dodawanie par[1] `<int, DiaryNote>`, w których pierwszy człon to dzień miesiąca, a drugi człon to kartka z pamiętnika stworzona danego dnia. W naszej kolekcji nie dopuszczamy wielu kartek dla jednego dnia. Przykładowy output:

```
3 : I was happy, because I played for 4h and it was cloudy.
6 : I was cranky, because I played for 2h and it was rainy.
7 : I was annoyed, because I played for 1h and it was sunny.
10 : I was joyful, because I played for 5h and it was cloudy.
```

Zadanie 3 - oceny

Stwórz klasę `Student`, która będzie zawierała informację o **id** ucznia oraz **kolekcję ocen**. Ta kolekcja musi w łatwy sposób umożliwić dodawanie par `<String, int`, w których pierwszy człon to nazwa przedmiotu, a drugi człon to ocena z teo przedmiotu. Ponadto, klasa ma zawierać 2 metody: `float calculateMean()` oraz `void addGrade(String s, int g)`. Pierwsza z nich wyznacza średnią arytmetyczną[2]. Druga metoda ma dodać ocenę z danego przedmiotu do kolekcji ocen ucznia. Następnie w metodzie `main` stwórz instancję `Student` i wywołaj na niej metodę `addGrade` wielokrotnie dla różnych ocen i przedmiotów. Następnie wypisz id studenta wraz z jego średnią. **Uwaga**: średnia powinna być wypisana w zaokrągleniu do dwóch miejsc po przecinku. Przykładowy output:

```
Student's id: 150
Student's mean: 4.50
```

Uwagi

1. Należy zadbać o podstawowe zasady programowania obiektowego - enkapsulację, gettery, settery, odpowiednią dostępność zmiennych... Te rzeczy nie są uwzględnione w treści zadania, ale są to dobre praktyki, które należy stosować.

[1] Chodzi o to, że nie zawsze będziemy mieli kartki z każdego dnia miesiąca, pamiętnik z dni 11, 17 i 20 jest już poprawnym pamiętnikiem. W związku z tym, nie powinniśmy tworzyć tablic o rozmiarze 31.

[2] W przypadku, gdy uczeń nie ma żadnych ocen, metoda zwraca -1.

Listy 2

Zadanie 1 - klasa

Stwórz klasę `Class`, która będzie zawierała **listę imion uczniów**. Ponadto, klasa ma zawierać 3 metody:

- `void addStudent(String name)`, która dodaje ucznia o imieniu `name` do listy studentów.
- `void displayStudents()`, która wypisuje wszystkich studentów w klasie po przecinku, kończąć kropką (tak jak w przykładowym outpucie). Jeśli klasa nie zawiera żadnych uczniów, należy wypisać stosowny komunikat.
- `int getGirlsCount()`, która zwraca liczbę dziewczyn w klasie. Na potrzeby zadania przyjmiemy, że każdy uczeń, którego imię kończy się na "a", jest dziewczyną.

Następnie w metodzie `main` stwórz instancję klasy `Class` i użyj metody `addStudent`, by dodać kilku studentów. Potem użyj metod `displayStudents` i `getGirlsCount`. Przykładowy output:

```
Students: Jacek, Ania, Marek, Grzesiek, Kasia.
This class has 2 girls.
```

Zadanie 2 - laptopy

Stwórz klasę `Laptop`, która będzie zawierała **nazwę** oraz **ocenę laptopa** - wartość całkowitą z przedziału $[0-50]$. Klasa ma zawierać metodę `void introduce()`, która wypisze komunikat postaci "Hi, I'm (nazwa)" oraz interpretację oceny. Dzielimy laptopy na 3 segmenty: z ocenami z przedziału $[0, 9]$, $[10, 24]$ oraz $[25, 50]$. Dla każdego z segmentów interpretacja oceny jest inna (patrz przykładowy output). Następnie w metodzie `main` stwórz listę laptopów typu `ArrayList` i dodaj do niej przykładowe laptopy. Dla każdego z nich wywołaj metodę `introduce`. Potem wypisz nazwę i ocenę wszystkich laptopów, które mają ocenę wyższą niż 20. Przykładowy output:

```
Hi, I'm ASUS NOVAGO TP370QL and I'm a very slow laptop.
Hi, I'm ACER PREDATOR 21 X and I'm a gaming machine!
Hi, I'm DELL LATITUDE 7390 and I'm a quite decent laptop.
Hi, I'm ALIENWARE 15 R3 and I'm a gaming machine!
ACER PREDATOR 21 X is rated 38.
ALIENWARE 15 R3 is rated 29.
```

Zadanie 3 - alkohole

Stwórz klasę `Alcohol`, która będzie zawierała **nazwę** oraz **moc** alkoholu - zawartość alkoholu (wartość z $[0-1]$). Klasa ma zawierać metodę `float calculatePerMil(int quantity, int weight, boolean male)`, która zwraca liczbę promili alkoholu we krwi po spożyciu danego alkoholu. Tę wartość liczymy ze wzoru Erika Widmarka. Następnie w metodzie `main` stwórz listę alkoholi typu `ArrayList` i dodaj do niej przykładowe alkohole. Dla każdego z nich wywołaj metodę `calculatePerMil` dla mężczyzny ważącego 80kg po spożyciu 100ml danego trunku. Output:

```
Man (80kg) will have 0.07‰ blood-alcohol content after drinking 100ml of Beer.
Man (80kg) will have 0.57‰ blood-alcohol content after drinking 100ml of Vodka.
Man (80kg) will have 0.20‰ blood-alcohol content after drinking 100ml of Wine.
Man (80kg) will have 1.00‰ blood-alcohol content after drinking 100ml of Hooch.
```

Uwagi

1. Należy zadbać o podstawowe zasady programowania obiektowego - enkapsulację, gettery, settery, odpowiednią dostępność zmiennych... Te rzeczy nie są uwzględnione w treści zadania, ale są to dobre praktyki, które należy stosować.