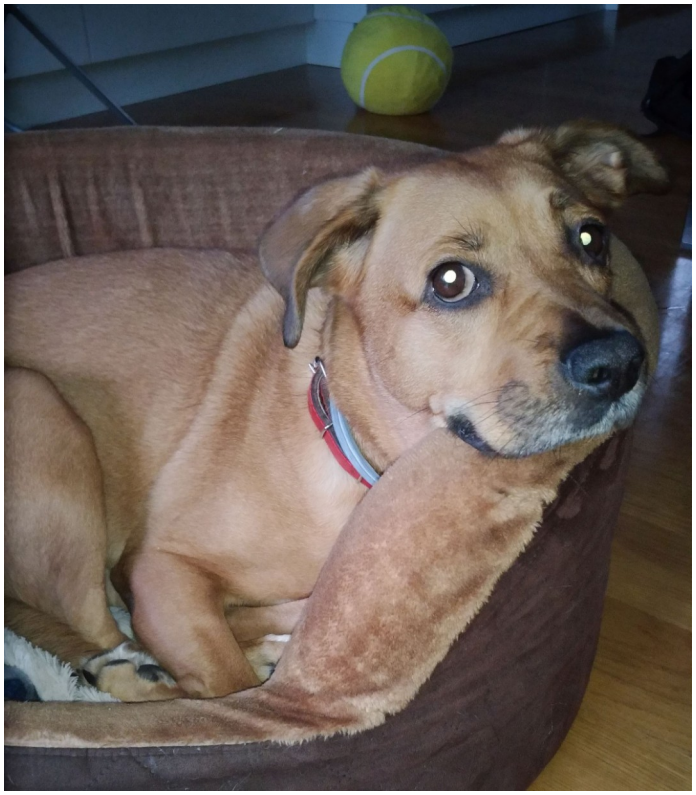
Zaawansowane Programowanie Obiektowe i Funkcyjne
Wyrażenia lambda
Zadanie oceniane nr 2b
14-11-2022

Dzisiaj będzie bez kodu wstępnego. Źródła powinny się znaleźć w Państwa katalogu roboczym (git) w podkatalogu zpoif_zadanie2b. Po zakończeniu pracy konieczne jest wgranie zmian.

"Psie pudełko"



Gdzieś na strychu znaleziono pudełko z akcesoriami dla psiego pupila. Należy stworzyć model jego zawartości wraz z towarzyszącą logiką bazującą na temacie wiodącym dzisiejszego zadania – czyli **wyrażeniach lambda**.

Miejmy gdzieś z tyłu głowy fakt, iż w porównaniu z klasą anonimową obiekt zdefiniowany za pomocą wyrażenia Lambda, można traktować jako metodę anonimową, czyli byt pozornie jeszcze bardziej "okrojony" z niepotrzebnego ciała niż klasa "bez nazwy" (w kórym to ciele zawsze można wstawić coś więcej niż tylko metodę). Używamy minimum tego co jest potrzebne (jeśli coś da się zrobić "lambdami", nie deklarujmy "anonimu"). Nie wymagam użycia wzorca Wizytor – dzisiaj można iść na skróty i do woli używać **instanceof**a. Nie używamy też strumieni – o nich będzie za tydzień. Tradycyjnie trzymamy się zasady: minimum zasięgu i minimum tego co potrzebujemy (chyba że w zadaniu wskazane będzie inaczej). Zamiast publicznych pól używamy setterów i getterów. Za ładną strukturę pakietów i formatowanie kodu – drobny extra bonus.

Prace do wykonania:

1. Stworzyć hierarchię klas reprezentującą następujące rodzaje psich akcesoriów:
   - Gryzak (nazwa producenta, atest* – tak/nie)
   - Gryzak piszczący (nazwa producenta, atest* – tak/nie, częstotliwość pisku w kHz* – 16/18/20/22/24) (jest to też gryzak, tylko że przy okazji piszczy)
   - Puszka dla psa (nazwa producenta, kaloryczność – 100-150)
   - Paczka karmy (nazwa producenta, kaloryczność – 300-400)

   * bazując na dotychczasowej wiedzy zaproponować sensowne typy dla tych pól.

   Inicjalizacji w/w pól dokonujemy w konstruktorach wykorzystując wartości dostarczone przez implementacje interfejsów funkcyjnych zwróconych przez metody klasy opisanej w kolejnym punkcie. Na całą hierarchię psich akcesoriów powinien przypadać tylko jeden taki obiekt (wyżej wymienionej klasy) i być umieszczony gdzieś w hierarchii klas należących do tej rodziny, aby konstruktory miały do niego dostęp. Konstruktor wywoływał będzie interesującą go metodę, a następnie pobierał wartość ze zwróconego obiektu (czyli inplementacji interfejsu funkcyjnego) i inicjował nią odpowiednie pole.

2. Stworzyć klasę InitHelper której instancja posiada **raz** zainicjowane pole klasy Random z którego korzystają trzy niestatyczne metody zwracające implementacje interfejsu **java.util.function.Supplier** (jeżeli pasuje on i uda się go wykorzystać) lub zdefiniowanego przez siebie (jeśli Supplier nie wystarczy). Będą one stworzone rzecz jasna tylko za pomocą wyrażeń Lambda. Zwrócone obiekty mają zostać użyte przez konstruktory psich gadżetów (z poprzedniego punktu) w celu uzyskania wylosowanej wartości do inicjalizacji pól (nazwa producenta, atest, częstotliwość, kaloryczność). Metody te są następujące:

   - provideRandomProducerNameGenerator()
     Zwraca obiekt dostarczający wartość dla pola "nazwa producenta" losowany spośród nazw takich jak: "DINGO", "FAFIK", "My pet", "LAPA", "CERBER", "raBIES".
   - provideRandomFrequencyGenerator()
     Zwraca obiekt dostarczający wartość wylosowanej częstotliwości.
   - provideRandomAttestationGenerator(boolean alwaysTrue)
     Zwraca obiekt dostarczający wartość dla pola "atest". Jeśli flaga alwaysTrue metody provideRandomAttestationGenerator ma wartość true, to "tak" jest zwracane zawsze. W p.p. "tak" wypada z P=0.95. Flaga z metody zostanie przekazana do zwróconej implementacji interfejsu funkcyjnego, tak że konstruktor pobierając wartość ze zwróconego obiektu nie będzie musiał jej podawać.
   - provideRandomCaloriesValue()
     Zwraca obiekt (implementację interfejsu funkcyjnego) dostarczający wartość dla pola "kaloryczność". Posiada on metodę zwracającą losowaną wartość kaloryczną spomiędzy przyjętych parametrów (int x, int y).

3. Stworzyć klasę MyDogBox posiadającą listę losowo pomieszanych gadżetów, która zawiera po 20 gryzaków (na każdy typ) oraz po 40 puszek i 50 paczek z karmą.

   Klasa zawiera poniżej wylistowane metody niestatyczne, z których każda wykonuje swoje zadanie bazując na iterowaniu wspomnianej listy za pomocą forEach() i dostarczając dla tej metody pożądaną implementację interfejsu java.util.function.Consumer. Kod consumera **nie może** się odnosić do żadnych

elementów na zewnątrz jego bloku**. Jeżeli jest możliwe żeby użyć tutaj wyrażenia lambda to trzeba to zrobić za jego pomocą. W p.p. użyć klasy anonimowej, a jeżeli i to będzie za mało to wewnętrznej lub normalnej. Wszelkim wykrywaniem, kolekcjonowaniem danych i ich udostępnianiem metodzie zajmuje się kod Consumera.**

- detectNonAttestationChew() – jeśli podczas iteracji natrafi na gryzak bez atestu, to wypisuje ten fakt na konsoli ("Brak atestu!!!").
- detectCaloricFood() – wypisuje na konsoli "Uwaga na kalorie", gdy natrafi na wartość kaloryczną większą nić 350 kalorii, pod warunkiem że ilość tego typu przypadków jest nie większa niż 31. Powyżej takiej ilości napotkanych przypadków już nie wypisuje.
- getSummarizedDryFoodCalories() – uzyskuje informację o sumarycznej liczbie kalorii dla wszystkich paczek z suchą karmą i wypisuje ją raz na konsoli.

4. Stworzyć podklasę klasy MyDogBox, której niestatyczne metody również działają na zainicjowanej w nadklasie liście. One także wykonują swoje działania bazując na iterowaniu metodą forEach dziedziczonej kolekcji, z tą różnicą że tym razem kod konsumera **może** się odnosić do elementów na zewnątrz bloku ale za to musi być definiowany w momencie przekazywania jako argument i nie wcześniej. Przykład: forEach((x) -> {implementacja.}). **Tutaj używamy tylko wyrażeń Lambda.**
   - upgradeProducerName(newName) – podczas iteracji w przypadku natrafienia na nazwę producenta nie dłuższą niż 5 liter, następuje wstawienie w jej miejsce wartości newName przekazaej w parametrze.
   - upgradeFrequency4All() – podmienia wszystkie częstotliwości w napotkanym gryzaku na wartość wylosowaną poza blokiem wyrażenia lambda (w ciele metody w którym ono zostało zdefiniowane)
   - getAverageWetFoodCalories() - ustala średnią wartość kaloryczną karmy w napotkanych puszkach i wypisuje ją na konsolę.

5. Stworzyć klasę MyFunnyTools zawierającą:
   - własny wewnętrzny interfejs funkcyjny zawierający metodę String myConcatenation(String, String), która zwraca dwa połączone ze sobą argumenty
   - metodę createMyConcatenation zwracającą implementację w/w interfejsu **z wykorzystaniem mechanizmu referencji**, który to zapożyczy ją od istniejącej znanej metody w klasie String. Chodzi o "podwójne dwukropki" ;)