Programowanie w języku JAVA - INF inż. - K.Warda - 2019/2020 - LATO - ARCH

Zestaw zadań z laboratorium.

Programowanie w języku Java

Witajcie Towarzysze

Przekraczając te wrota porzućcie wszelkie nadzieje.]:->

Skrolujecie w dół na własną odpowiedzialność gdyż tragedia wisi w powietrzu.

Poniżej znajdziecie serie zadań o kotkach, pieskach, telefonach i samochodach. Nie dajcie się jednak zwieźć, lekko nie będzie.

W razie potrzeby moją mało pomocną prezentację możecie znaleźć pod linkiem tutaj.

Także, jak coś to piszcie, pozdro.

Ogłoszenia

Podstawowe informacje o zajęciach

Zasady zaliczenia przedmiotu i oceniania

Ukryte przed studentami

Karta przedmiotu (sylabus)

Ukryte przed studentami

Ankieta

Ukryte przed studentami

Komunikacja

Aktualności - wiadomości od nauczyciela

Forum - dyskusje z nauczycielem i grupą

Adres e-mail nauczyciela

Stosuj netykietę

Blok pierwszy - podstawy

Przeszukałem trochę internety i jest sporo badziewnego wideo na temat javy.

Jeśli ktoś chce skumać podstawy Javy w mniej niż godzinę polecam

Jeśli ktoś chce zacząć całkiem od podstaw programowania polecam

Jeśli ktoś chce się na prawdę czegoś nauczyć polecam książkę JAVA. RUSZ GŁOWĄ! (eng Head First! Java)

Zadanie 0 - wstępna konfiguracja

1. Pobierz i zainstaluj
   - IntelliJ (może być w wersji community)
   - Java JDK (co najmniej 8)
   - Git
2. Załóż konto w serwisie github.com
3. Utwórz nowy projekt na podstawie szablonu 'Java Console Application' w IntelliJ
4. Utwórz lokalne repozytorium gita, i zdalne, publiczne repozytorium na githubie
5. Połącz lokalne repozytorium ze zdalnym na githubie
6. Utwórz klasy Human, Animal, Phone zgodnie z zasadami omawianymi na zajęciach
7. Dodaj klasy do repozytorium, wypchnij całość na github i wklej link to commit jako rozwiązanie zadania

Zadanie 1 - Pets feeding process

1. Dodaj pole Animal pet w klasie Human
2. Dodaj pola final String species i private Double weight w klasie Animal
3. Dodaj konstruktor w klasie Animal, który będzie ustawiał początkową wagę zwierzęcia na podstawie gatunku
4. Dodaj metody void feed() i void takeForAWalk() w klasie Animal. Karmienie zwierzęcia powinno zwiększać jego masę, wyprowadzenia na spacer zmniejszać. Jeśli masa zwierzęcia spadnie do 0 zwierze nie żyje. Kolejne próby karmienia lub wyprowadzania na spacer powinny skończyć odpowiednio pasywno-agresywnym komunikatem.
5. W Main wykonaj kilka razy obie metody żeby upewnić się że wszystko działa
5. Commit, push i link to githuba jako rozwiązanie

Zadanie 2: get yourself a car

1. Utwórz klasę Car, wybierz pola wg własnego uznania, ale nie zapomnij o finalnych polach String model i String producer
2. Utwórz a Main nowy samochód i przypisz go do utworzonego wcześniej człowieka
3. Wiem, że prostackie, ale samochód będzie nam później potrzebny (tzn po pandemii jak już będzie można w końcu gdzieś pojechać)
4. commit, push, github, wiadomo

Zadanie 3 i 4: get your salary & give yourself a rise

1. Dodaj prywatne pole Double salary do człowieka
2. Dodaj getter, który będzie wypisywał na konsole informacje kiedy były pobierane dane o Twojej wypłacie i jaka była wtedy jej wartość przed zwróceniem wartości
3. Przeczytaj https://dzone.com/articles/java-getter-and-setter-basics-common-mistakes-and
4. Napisz setter, który symuluje prawdziwą operację zmiany wysokości wynagrodzenia czyli:
    - sprawdza czy ktoś nie próbuje przypisać ujemnej wartości wypłaty
    - wypisuje informacje o tym, że nowe dane zostały wysłane do systemu księgowego
    - wypisuje informację o konieczności odebrania aneksu do umowy od pani Hani z kadr
    - wypisuje info, że ZUS i US już wiedzą o zmianie wypłaty i nie ma sensu ukrywać dochodu
    - przypisuje nową wartość wypłaty do człowieka

Zadanie 5: check if you could buy it

1. zmień pole Car w człowieku na pole prywatne
2. dodaj pole wartość (ofkoz po angielsku) do samochodu
3. dodaj getter do człowieka, który zwraca samochód bez żadnej dodatkowej logiki
4. dodaj setter, który sprawdza, czy człowieka stać na ten samochód
    - jeśli zarobki człowieka są wyższe niż cena samochodu wypisujemy info, że udało się kupić za gotówkę i przypisujemy samochód
    - jeżeli zarobi są większe niż 1/12 wartości samochodu wypisujemy info, że udało się kupić na kredyt (no trudno) i przypisujemy samochód
    - w pozostałych wypadkach wypisujemy info, w stylu "zapisz się na studia i znajdź nową robotę albo idź po podwyżkę" i nie przypisujemy samochodu
5. link do commit na githubie pozostaje kluczem do szczęścia

Zadanie 6: one more thing

1. Spróbujcie skumać to https://www.javaworld.com/article/3305792/comparing-java-objects-with-equals-and-hashcode.html
2. Spróbujcie porównać dwa samochody o takich samych wartościach pól
3. Spróbujcie wypisać cały obiekt a nie jedno pole np System.out.println(dog), zobaczcie co tam widać, spróbujcie wypisać dwa samochody o takich samych wartościach pól
4. Spróbujcie skumać to https://www.javatpoint.com/understanding-toString()-method
5. Dodajcie toString do każdej klasy w projekcie. Użyjcie każdego z nich w Main
6. Spróbujcie skumać to https://www.tutorialspoint.com/java/java_packages.htm
7. Utwórzcie nową paczkę devices (piszemy to z małej litery i w jedynym słusznym języku) i przenieście do niej klasy Car i Phone (jeśli nie macie to wiadomo, trzeba sobie utworzyć)
8. Przejrzyjcie inne klasy i zobaczycie czy coś przestało działać, odpalcie program i sprawdźcie czy nie sypie błędami. Poprawcie błędy
9. => github link is my dream <=

Blok drugi - dziedziczenie

Jeżeli ktoś nie pamięta z zajęć o co chodzi z klasami abstrakcyjnymi i interface'ami to może sobie odświeżyć temat u tego kolegi:

Zadanie 7: create Device class

1. Utwórz abstrakcyjną klasę Device w paczce devices
2. Car i Phone powinny dziedziczyć po Device
3. Przenieś do Device pola producer, model, yearOfProduction
4. Dodaj implementacje klasy toString()
5. Dodaj abstrakcyjną metodę turnOn()
6. Zaimplementuj turnOn w podklasach
7. Przetestuj w Main czy to działa
8. wiadomo - github

Zadanie 8: real selling

1. Dodaj pole cash do Human
2. Utwórz interface salleable z jedną metodą sell przyjmującą jako parametry Human seller, Human buyer, Double price
3. Zaimplementuj tą metodę w zwierzętach i urządzeniach w taki sposób, że jej wykonanie spowoduje:
    - sprawdzenie, czy sprzedający rzeczywiście posiada zwierze, samochód lub telefon, który chciałby sprzedać
    - sprawdzenie, czy kupujący ma dość gotówki
    - ubytek pieniądza na koncie kupującego
    - przybytek pieniądza na koncie sprzedającego
    - zmianę właściciela sprzedawanego obiektu
    - wypisanie na konsole informacji o przeprowadzonej transakcji
4. Upewnij się, że handel ludźmi będzie niemożliwy w Twojej aplikacji
5. Przetestuj swoje pomysły w Main i wiadomo, github

Zadanie 9: farmin'

1. Rozszerz klasę Animal o bardziej szczegółowe klasy: FarmAnimal i Pet
2. Użyj super w konstruktorach podklas
3. Dodaj interface Edbile z metodą beEaten(), zaimplementuj ten interface w FarmAnimal
4. Utwórz paczkę creatures i przenieś do niej wszystkie klasy i interface'y, które mają coś wspólnego ze zwierzętami
5. Zmień Animal w klasę abstrakcyjną
6. Utwórz interface Feedable z metodami feed() i feed(foodWeight)
7. Zaimplementuj interface w Animal
8. Popraw Main, przetestuj kod, wstaw na github

Zadanie 10: apps instalation & electric cars

1. Do klasy Phone dodaj kilka wersji metody installAnnApp, które powinny różnić się przyjmowanymi parametrami, poszczególne wersje powinny przyjmować
    - nazwę aplikacji do zainstalowanie
    - nazwę i wersję
    - nazwę, wersję i adres serwera, z którego należy zainstalować aplikację
    - listę nazw aplikacji do zainstalowania
    - obiekt kasy URL, który powinien odwoływać się do konkretnej aplikacji w konkretnej wersji na konkretnym serwerze
2. W klasie Phone dodaj stałe (static final) z domyślnym adresem serwera z aplikacjami, domyślnym protokołem i domyślną nazwą wersji
3. Postaraj się zrobić to w taki sposób, żeby różniło się od kodu, który przerabialiśmy jako przykład na zajęciach ;)
4. Dodaj trzy klasy dziedziczące z Car: Electric, Disel i LPG
5. Dodaj abstrakcyjną metodę refuel() do Car
6. Napraw kod, który przestanie działać po tej zmianie
7. Przetestuj wszystko w Main i ślij na github.

Blok trzeci - tablice i kolekcje

Podstawy odnośnie tablic możecie znaleźć np tu:

jeżeli chcecie oglądać, ale jeżeli chcecie poczytać to lepszy tutorial jest w w3schools https://www.w3schools.com/java/java_arrays.asp

Listy:

https://www.w3schools.com/java/java_arraylist.asp

Zbiory i iterator:

http://tutorials.jenkov.com/java-collections/set.html

Sortowanie na różne sposoby:

Myszojeleń po niemiecku:

Zadanie 11 - garage

1. Usuń pole Car car z Human i na jego miejsce wstaw tablicę samochodów o nazwie garage.
2. Popraw konstruktor w Human tak, aby nowo utworzony człowiek posiadał garaż z domyślną liczą pojazdów.
3. Dodaj nowy konstruktor w którym rozmiar garażu będzie (kolejnym?) parametrem.
4. Popraw metody getCar i setCar, żeby przyjmowały jako parametr numer miejsca parkingowego w garażu z którego ma być pobrany samochód lub na które ma być wstawiony samochód.
5. Utwórz metodę, która zwraca sumę wartości pojazdów w garażu. Pole Double value powinno znajdować się w klasie Device.
6. Utwórz metodę sortującą samochody w garażu po roku produkcji od najstarszych do najmłodszych.
7. Zmień metodę Car.sell(Human seller, Human buyer, Double price) tak, aby zawierała:
   - sprawdzenie, czy sprzedawca posiada samochód w garażu (jeżeli nie, metoda powinna rzucić wyjątek)
   - sprawdzenie, czy kupujący posiada wolne miejsce w garażu (jeżeli nie, metoda powinna rzucić wyjątek)
   - sprawdzenie, czy kupujący posiada wystarczającą ilość gotówki (jeżeli nie, metoda powinna rzucić wyjątek)
   - usunięcie pojazdu z garażu sprzedawcy
   - dodanie pojazdu w pierwsze wolne miejsce w garażu kupującego
   - wymianę gotówki
   - wypisanie informacji jeżeli transakcja zakończyła się sukcesem
8. Sprawdzenie w main wszystkich utworzonych metod, commit, push, link do github.

Zadanie 12 - Niemiec płakał jak sprzedawał

1. Do klasy Car dodaj listę właścicieli pojazdu.
2. Ostatni element listy powinien wskazywać na aktualnego właściciela pojazdu.
3. Każda transakcja sprzedaży powinna dodawać nowy element do listy.
4. Podczas sprzedaży powinniśmy sprawdzać nie tylko czy samochód jest w garażu sprzedawcy, ale także czy sprzedający jest ostatnim właścicielem pojazdu wpisanym w liście właścicieli.
5. Dodaj do Car metodę sprawdzającą czy jakiś człowiek był właścicielem pojazdu.
6. Dodaj do Car metodę sprawdzającą czy człowiek A sprzedał samochód człowiekowi B.
7. Dodaj do Car metodę zwracającą liczbę transakcji sprzedaży w których uczestniczył samochód.
8*. Zamiast zapisywać listę właścicieli utwórz listę obiektów zbierających informację na temat każdej transakcji (kupujący, sprzedający, cena, data sprzedaży) - dla chętnych.
9. wiadomo przetestuj całość w main, git-> github -> moodle

Zadanie 13 - apki

1. Utwórz nową klasę Application, która będzie przechowywała nazwę, wersję i cenę aplikacji.
2. Dodaj do klasy Phone zbiór aplikacji.
3. Dodaj w Phone metodę instalującą nową aplikację w której:
   - sprawdzamy czy właściciel telefonu ma dość pieniędzy,
   - dodajemy zainstalowaną aplikację do zbioru,
   - zmniejszamy ilość gotówki na koncie kupującego.
4. Dodaj w Phone metody umożliwiające:
   - sprawdzenie czy aplikacja jest zainstalowana przyjmującą w parametrze obiekt klasy Application,
   - sprawdzenie czy aplikacja jest zainstalowana przyjmująca w parametrze nazwę aplikacji,
   - wypisanie wszystkich bezpłatnych aplikacji,
   - zwracającą wartość wszystkich zainstalowanych aplikacji,
   - wypisanie nazw wszystkich zainstalowanych aplikacji w kolejności alfabetycznej,
   - wypisane nazw wszystkich zainstalowanych aplikacji od najtańszych do najdroższych.
5. Utwórz kilka aplikacji i przetestuj wszystko w main.
6. github itp.

Blok czwarty - projekt

Każdy student powinien przygotować jeden z trzech projektów. Przypisanie do projektu zależy od reszty z dzielenia numeru indeksu przez 3:

- reszta = 1 - appstore - symulator firmy IT
- reszta = 2 - autokomis - symulator handlu samochodami
- reszta = 0 - farma - no wiadomo ;)

Każdy powinien realizować swój projekt jako nowy projekt w IntelliJ, z osobnym repozytorium na github. Repozytorium projektu powinno posiadać historię commitów pokazujących historię pracy nad projektem. Polecam samodzielnie podzielić sobie projekt na mniejsze zadania i realizować je po kolei.

W przypadku wątpliwości co do samodzielnego wykonania projektu konieczne będzie kilkuminutowe spotkanie online w celu wspólnego omówienia kodu. Oddanie nieswojego projektu = niezaliczenie przedmiotu.

Możliwe jest oddanie części projektu i uzyskanie częściowej liczby punktów, pod warunkiem, że oddana część będzie działać.

Możliwe jest zaliczenie tej części przedmiotu przez publiczne udostępnienie innego, własnego, repozytorium z kodem o co najmniej takim samym poziomie złożoności. Taka forma zaliczenia będzie wymagała kilkuminutowej rozmowy w celu wspólnego omówienia kodu.

powodzenia :]

 Projekt