Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológií

TravelNet

Peter Farkaš

Objektovo orientované programovanie

Cvičenie:
Streda 18:00-19:50

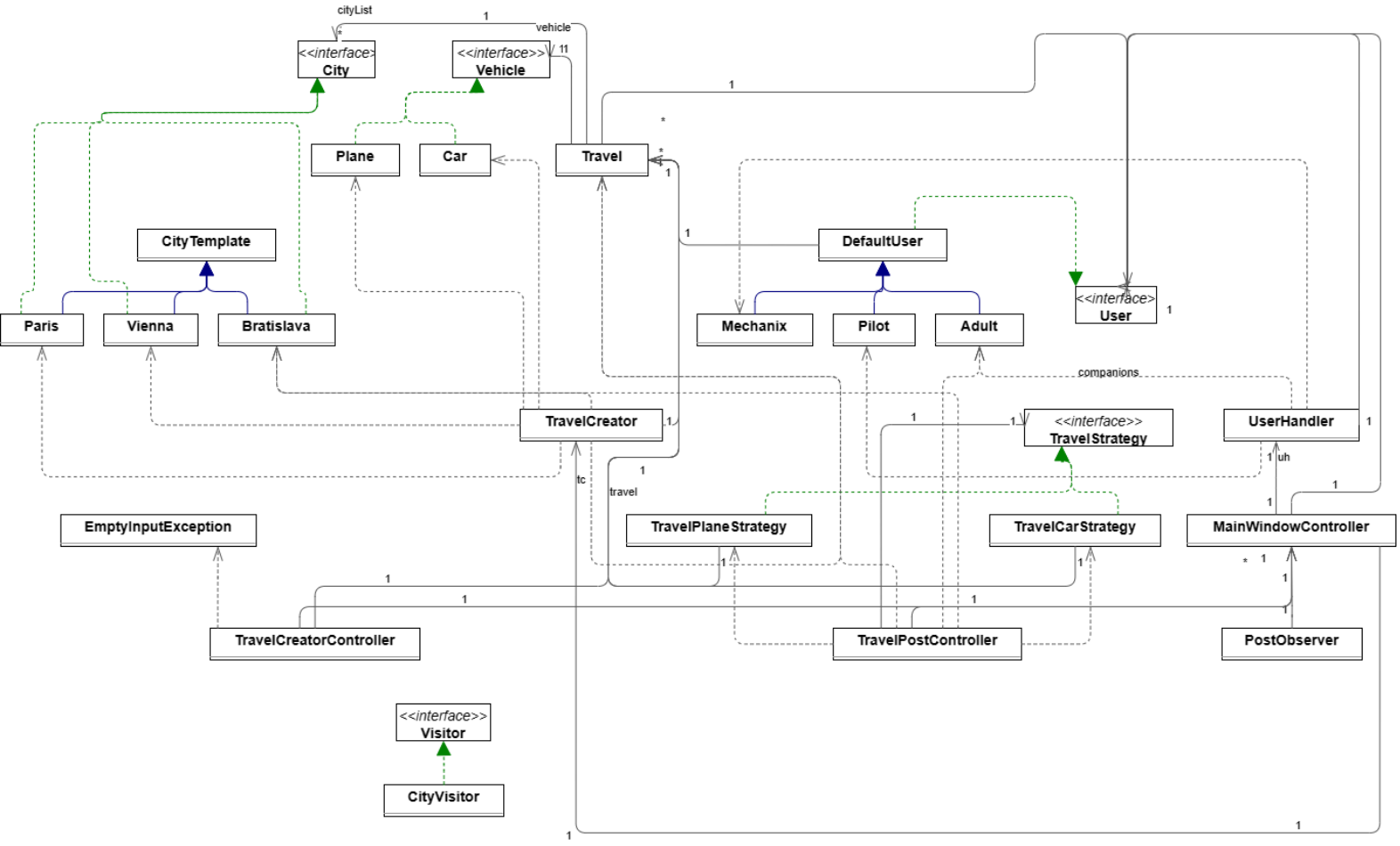
Cvičiaci: Ing.Tomáš Frťala, Phd.

Obsah

Zámer projektu

Cieľom je vytvoriť platformu sociálnej siete, ktorá bude zameraná na vytváranie cestovných plánov po svete, zdieľať ich s ostatnými užívateľmi a umožniť prijať spoločníkov do našej cesty. Každý užívateľ sa bude môcť registrovať do systému a následne sa prihlasovať do účtu. Pri registrácii užívateľ zadáva jazyky, ktoré ovláda, typy jedál, ktoré preferuje a podobné informácie, aby spolu cestovali ľudia, ktorí majú radi rovnaké veci a tým sa spríjemnila celá ich spoločná cesta. Práca so systémom bude nasledovná: užívateľ si bude môcť vytvoriť vlastné dobrodružstvo alebo hľadať v zozname existujúcich. Tvorba Dobrodružstva bude nasledovná: v grafickom prostredí systému sa pre klikáme do menu tvorby dobrodružstva. Tam si vyberieme, kde naša cesta začína, časové rozpätie cesty, následne začneme plánovať postupne miesta, ktoré navštívime a akými prostriedkami tie miesta navštívime. Ak sme cestu vytvorili, môžeme do nej pozvať svojich priateľov zo zoznamu priateľov alebo zverejniť naše dobrodružstvo do katalógu komunity, kde si na ňu môžu pridať žiadosť ľudia z platformy. Do aktívne dobrodružstvá sa celú dobu môžu pridávať fotky a komentáre z dobrodružstvá. Taktiež bude implementovaný bodovací systém, kde každý účastník dobrodružstva môže ohodnotiť celé dobrodružstvo a spolucestujúcom h individuálne. Podľa počtu dobrodružstiev a hodnotení sa užívateľom budú prideľovať ranky.

Uml Diagram najdôležitejších tried



Aby sa UML diagram zmestil do dokumentácie, vybrali sme iba najdôležitejšie triedy aplikačnej logiky, úpný diagram pridávam ako prílohu k dokumentácii. Z rozhrania City implementujú metódy trieda Paris, Bratislava a Vienna ktoré zároveň rozširujú triedu CityTemplate, ktorá je abstraktná. Tieto triedy, ktoré imprelemtujú rozhranie City sú v kompozícii s TravelCreatorom a samotnou Travel triedou. Travel trieda je najdôležitejšia trieda projektu, z tejto triedy sa odvíja správanie aplikcie na základe objektov ktoré sú v jeho kompozícii, rolu hrá pri vytváraní nových postov v rozhraní, pri simulácii cestovania aj pri správaní sa aplikácii po dorazení do ciela v CityWindowControlleri, táto trieda slúži zobrazovanie postov a ďalšiu interakciu s nimi. TravelCreator je singleton databaza Travel objektov, ktorá ich aj vytvára.Travel MainWindowController agreguje Travel tiredu, aj ked táto trieda má Travel ako atribút, ale zánikom MainWindowControlleru samotný object travel nezaniká, ale zostáva v databáze, to isté platí o TravelPostController. Triedy, ktoré implementujú rozhranie TravelStrategy sú kompozíciou v triedach TravelPostController a CityWindowController(vid rozšírený priložený diagram). Triedy, ktoré implementujú rozhranie Strategy slúžia ako algoritmy na určenie správania travel metód v TravelPostController a CityWindowController, tieto metódy objekty typu Strategy agragujú. MainWindowController je v obojsmernej agregácii s PostObserverom, navzájom si jeden do druhého vstupujú a volajú si svoje metódy. Triedy Mechanix, Pilot, Adult implementujú rozhranie User a rozširujú abstraktnú triedu DefaultUser. Slúžia na definovanie typu užívatela a zároveň definujú čo užívateľ môže robiť. CityVisitor implementuje rozhranie Visitor a slúži

na nastavovanie mapy pre jednotlivé mestá v CityWindowController, kde je kompozíciou tejto triedy.

Hlavné kritériá

Dedenie a polymorfizmus

V programe využívame viacero oddelených hierarchy dedenia, ako príklad by

som uviedol nasledovné :

Triedy Adult, Mechanix a Pilot rozširujú nadtyp DefaultUser, ktorý je abstraktná trieda, týmpádom sa jej konštruktor dá volať iba pomocou kľúčového slova super(). DefaultUser implementuje rozhranie User, týmpádom každá jedna metóda z tohto rozhrania je prekonaná v podtypoch triedy DefaultUser, čím sa zároveň uplatňuje polymorfizmus, pretože triedy rozširujúce DefaultUsera majú rôzne správanie pre metódy s rovnakou signatúrou.

Triedy TravelCarStrategy a TravelPlaneStrategy implementujú rozhranie TravelStrategy, čím dedia jeho metódy, a keďže sa jedná o rozhranie, metódy nadtypu musia byť prekonané preto ich správanie bude rozličné ako správania nadtypu, ergo sa jedná o ďalší prípad polymorfizmu. Rovnaký prístup sa uplatňuje pri triedach Car a Plane, ktoré implementujú rozhranie Vehicle.

Zapuzdrenie

V tomto projekte ak chcem pristupovať k atribútom nejakej triedy mimo nej alebo z jej podtypu, tak používame gettre a setter. Atribúty každej triedy sú buď private alebo protected, kde na získavanie ich hodnôt používame get() metódy a na ich nastavovanie používame set() metódy. Priami prístup k týmto atribútom cez object.názovatribútu týmpádom nie sú povolené(to by bolo možné iba ak by tieto atribúty boli public, čo nieje ortodoxné). Ako príklad si vezmime triedu Default user, ktorý má viditeľnosť atribútov protected, týmpádom k nim triedy, ktoré ju rozširujú majú prístup ako Pilot, Adult a Mechanix ale ak chcem pristupovať k atribútom týchto tried mimo tried, ktoré by ich rotširovali, ako napríklad z triedy UserStatusController, tak musíme využívať get/set metódy.

Agregácia

Agregácia je, ak nejaký object vstupuje do iného objektu ako vstupný parameter a následne dokáže naďalej existovať, ak by trieda, do ktorej vsúpila zanikla. Agregáciu v tomto projekte sa využíva predovšetkým pri prenášani inštancií controller tried, keď z jedného controlleru musíme pracovať s inštanciou iného controllerov a týmpádom tak tvorili modulárne grafické užívateľské prostredie. Ale aj napríklad pri návrhovom vzore visitor, ktorý sa využíva tiež v tomto projekte, vstupnými parametrami sú objekty Bratislava, Paris alebo Vienna. Pri triede Mechanix metoda skill(Vehicle vehicle) agreguje objekt, implementujúci rozhranie Vehicle a podobne.

Oddelenie aplikačnejlogiky od rozhrania

Toto kritérium sme docielili použitím návrhového vzoru MVC to jest Model View Controller. Model reprezentuje data, v našom prípade objekty ako sú mestá, vozidlá, uživatelia a pod. Controller rieši dopyty od užívateľa, ergo reaguje na udalosti z

užívateľského rozhrania a na základe z nich pracuje s objektmi z modelu. Controller rieši aplikačnú logiku. View reprezentuje užívateľské rozhranie a prezentáciu dát z aplikačnej logiky.

Rozdelenie tried do balíkov

Keďže používame návrhový vzor MVC, naše rozdelenie do balíkov vyzerá nasledovne. Koreň nášho projektu je balík com. Potom balík s názvom projektu com.travelnet. Balík com.travelnet je vetvený na 3 balíky korešpondujúce s návrhovým vzorom MVC a to sú view, kďe máme triedy, ktoré sú zodpovedné za scény užívateľského rozhrania. V balíku model máme všetky triedy, s ktorými pracujeme v aplikačnej logike. Model je rozdelený na pod balíky pre mestá, exceptions, štratégiu, užívateľov, utility tried a vozidiel. A nazáver v balíku controller mám triedy, ktoré riadia aplikačnú logiku užívateľského rozhrania.

Ďaľšie kritériá

Návrhový vzor MVC

Ako bolo spomenuté vyššie pri opise odelenia aplikačnej logiky od užívateľského rozhrania, MVC je návrhový vzor ktorý nám rozdelí kód na prezentačnú, logickú a dátovú časť. View v našom prípade reprezentuje triedy jednotlivých scén, ktorými interaguje užívateľ a na tieto dopyty potom reagujú a spracuvávajú jednotlivé triedy controllerov týchto scén a ďalej pracujú s objektami z modelu ktoré sú v našom prípade POJO(Plain old java object).

Návrhový vzor Observer

Observera sme využili pri podpore viacerých užívateľov naraz konkrétne pri pridávaní a odoberaní postov v MainWindow scene. Aby sme docielili, že pri pridaní postu od jedného užívateľa sa post dynamicky ukáže aj v inštanciách iných prihlásených uzívateľov používame tento návrhový vzor tak, že zakaždým ak sa užívateľ prihlásy do aplikácie, tak jeho inštancia MainWindowControlleru sa pridá do arraylistu subjectov PostObservera. Následne, ak sa pridá nejaký post hociktorým užívateľom, tak sa zavolá metóda PostObservera notify(). Táto metóda potom v každej inštancii MainWindowController v arrayliste subov observera zavolá metódu MainWindowController upadate, ktorá pridá nový post pre všetky inštancie zo subov observera. Rovnaký spôsob sa využíva pri odoberaní postov, len na miesto notify sa zavolá notifyDelete metóda Observera.

Návrhový vzor Visitor

Tento návrhový vzor funguje na báze polymorfizmu a preťažovania. To znamená že objekt bude meniť svoje správanie na základe vstupného parametra. Tento návrhový vzor používame pri nastavení wigetu mapy v CityWindowController. Do visitora vstupuje objekt Typu City ktorému pomocou RTTI zístíme inštanciu a castneme ho do nej. Potom tento objekt pošleme do visitora a na základe či je objekt typu Bratislava, Vienna alebo Paris sa nám v CityWindowController nastavý mapa.

Návrhový vzor Strategy

Návrhový vzor strategy využívame pri simulácii cestovania z jedného mesta do druhého. Vtriedach TravelPostController a CityWindowController sa nachádza metóda travel, ktorá má vstupný parameter typu TravelStrategy. Pri volaní tejto triedy sa správanie mení či vstupným parametrom bude TravelPlaneStrategy alebo TravelCarStrategy podla typu vozidla v aktualnom Travel objekte. Pomocou tohto paternu sme schopný meniť správanie algoritmu Tracel metód počas runtime program. TravelCarStrategy a TravelPlaneStrategy slúžia ako algoritmy na vykonanie Travel metody. Implementuju rozhranie Strategy.

Vlastná výnimka

V projekte využívame vlastnú výnimku EmptyInputException, ktorý vyhadzujeme zakaždým ak užívateľ nezvolí povinný vstup. Ošetrujeme výnimku tak, že pri zachytení vypíšeme do užívateľského rozhrania užívateľovi, čo zabudol vyplniť.

Grafické Rozhranie

Grafické rozhranie je poskytnuté pomocou frameworku JavaFx, aplikačná logika je oddelená od užívateľského rozhrania pomocou controller tried ktoré v sebe majú spracovávateľov udalostí napríklad na kliknutie myšou na tlačidlo. Rozhranie bolo navrhnuté v programe SceneBuilder.

Viacniťovosť

Na viacniťovosť sa v projekte využíva API Service. Viacniťovosť sa využíva na simulácie cestovanie, kde po odkliknutí cestovanie do nejakého mesta v Gui, na novom vlákne sa spustí delay metoda delayCounter, ktorá vypočíta a odpočítava dobu trvania cesty do destinácie. Tento časovač sa vypisuje aj do užívateľského prostredia dynamicky, bez toho aby Gui stratilo responzivitu, nezasekne sa. Po splnení tasku na novom vlákne, ergo ak časovač dovŕši nulu. Tak sa zobrazí okno s destináciou.

RTTI

RTTI sa využíva pri zistení inštancie Objektov City a Vehicle. Pri výbere štratégie alebo v prípade objektov City pri downcastovaní pred vstupom do Visitora.

Vhniezdená trieda

TravelCarStrategy a TravelPlaneStrategy majú vhniezdenú triedu Service, ktorá slúži na spustenie delay metody delayCounter, ktorá bola spomínaná pri bode Viacniťovosti.

Lambda

Lambda výrazy boli použité na updateovanie zostávajúceho času z procesu cestovania z iného tredu, na ktorom beží javaFx. Celkovo v tomto projekte boli lambda výrazy použité na navigáciu medzi niťmi program. Hlavne v hniezdených triedach z API Service patriace triedam TravelCarStrategy a TravelPlaneStrategy.

Serializácia

Serializácia sa používa pri registrácii užívateľa, kde zakaždým, ak sa zaregistuje nový užívateľ, tak sa stav objektov v zozname užívateľov zo singleton databázy UserHandler serializuje a pri každom spustení aplikácie sa tieto objekty deserializujú a pridajú sa naspať do databázy užívateľov.

Verziovanie

1. -aggregation – vytvorenie základnej agregácie, kompozície, dedenia a Gui
2. -travel-creator gui implemented – vytvorenie logiky a rozhrania pre pridávanie postov
3. -better polymorphism,better inheritance – zlepšenie polimorfizmu a dedenia tried
4. -observer implemented and multiuser simulation – pridanie observer návrhového vzoru a simulácia viacerých užívateľov zabezpečená
5. -login register – pridane možnosti registácie a prihlasovania užívateľov
6. -city window – pridanie rozhrania pre mestá
7. -multicultithreading and nested classes – pridanie viacniťovosti
8. -custom exception handling – pridanie vlastných výnimiek
9. -user serialization – pridanie serializácie
10. -Strategy implementation – pridanie Strategy vzoru
11. -travel to next city – pridanie možnosti cestovania z mesta do mesta a Visitor implementovaný