

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 2, 842 16 Bratislava 4

SeaRoutes

Objektovo orientované programovanie

Matej Drienovský
Skupina: Streda 16:00
14.5.2023
ID:120777

Obsah

Zámer projektu a opis aplikácie.....	3
Štruktúra systému	3
Lode	3
Objednávky	4
Používatelia	4
Kapitán	5
Cesta.....	5
Kontrolery	5
Vlastná výnimka (OwnException).....	6
Grafické používateľské rozhranie	6
Vlastná generická trieda	7
Explicitné použitie RTTI.....	7
Lambda výrazy	8
Default method implementation	8
Serializácia	9
Návrhové vzory	9
Factory.....	9
Builder.....	9
Facade	9
Splnené kritéria.....	10
Databáza.....	10
GitHub.....	10
Návod k používaniu aplikácie	12
UML diagram.....	13

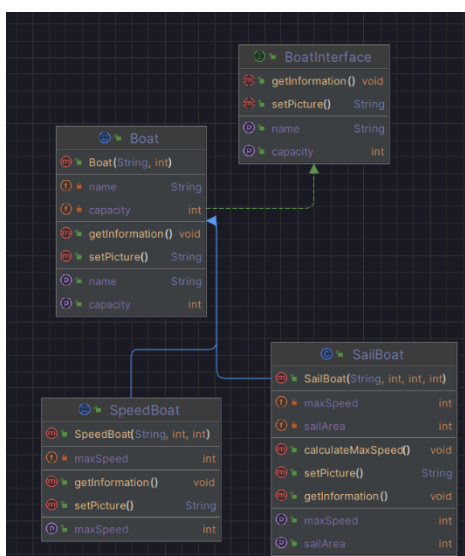
Zámer projektu a opis aplikácie

Zámerom projektu bolo vytvoriť aplikáciu SeaRoutes pre plánovanie lodných ciest s používateľským a správcovským rozhraním. Používateľ sa môže zaregistrovať alebo ak už má vytvorený účet, tak sa prihlási. Následne mu aplikácia ponúka možnosť kde si zvolí počiatočný prístav a taktiež cieľovú destináciu. Následne si zvolí loď aj kapitána, dátum plavby a pridá vek pre uplatnenie zľavy. Pri každom kapitánovi je hodnotenie, ktoré je vypočítané z recenzií od používateľov z predchádzajúcich plavieb. Následne bude vygenerovaný lodný lístok so všetkými potrebnými údajmi. Používateľ je navedený na stlačenie tlačidla pre cestu. Po stlačení sa spustí animácia plavby lode. Po úspešnom priplávaní do cieľového prístavu vyskočí pop-up okno v ktorom bude mať používateľ možnosť zanechať recenziu na kapitána. Po zrušení pop-up okna nás aplikácia presunie naspäť do úvodného menu pre tvorbu objednávky, kde môžeme vytvoriť novú trasu. Taktiež je v hlavnom menu je možnosť odhlásiť sa a prejsť do úvodnej scény. SeaRoutes taktiež obsahuje správcovské rozhranie pre admina. Admin po úspešnom zadaní hesla bude pripustený do správcovského rozhrania v ktorom má možnosti pridania a odoberania lodí, používateľov, kapitánov a taktiež admin vie pridať ďalšieho admina. Všetky tieto možnosti sú sprostredkované v intuitívnom GUI.

Štruktúra systému

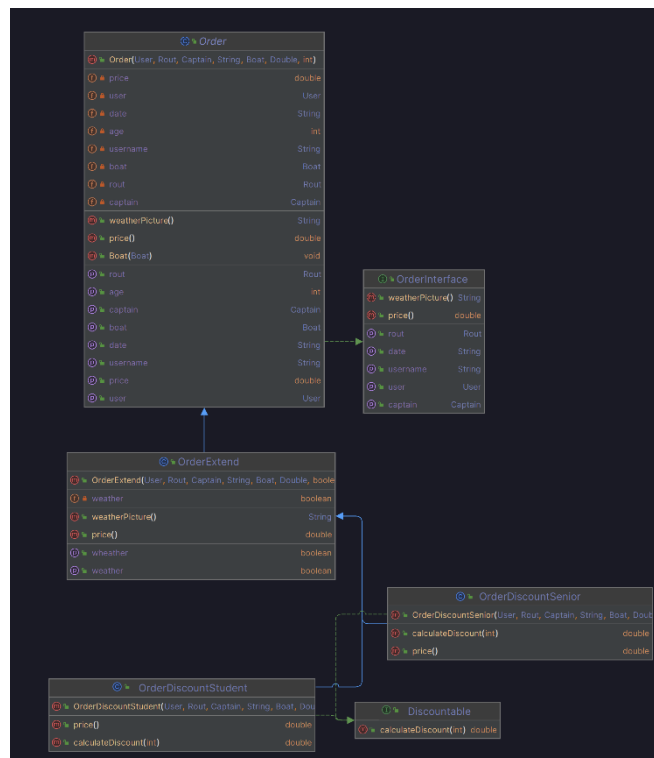
Lode

Dedenie je v tomto programe uplatnené viackrát ale tu je príklad pri lodiach. Použil som aj rozhranie BoatInterface a potom samotná trieda Boat ktorá obsahuje základné údaje ako názov a kapacitu. Ďalej triedy SpeedBoat a SailBoat, ktoré od základnej triedy Boat dedia a každá z týchto rozšírených tried má vlastné ďalšie údaje. Napríklad SpeedBoat obsahuje rýchlosť lode. Ako je vidieť na danom diagrame, v programe je použité zapúzdrenie a tak k jednotlivým údajom pristupujeme pomocou getterov alebo setterov. Taktiež pri lodiach využívam aj polymorfizmus a to konkrétne pri metódach setPicture a getInformation. Metóda setPicture slúži na priradenie obrázku danému typu lodi. A metóda getInformation slúži na vypísanie informácií o danej lodi, pri každom type vypíše iné informácie prislúchajúce danej lodi.



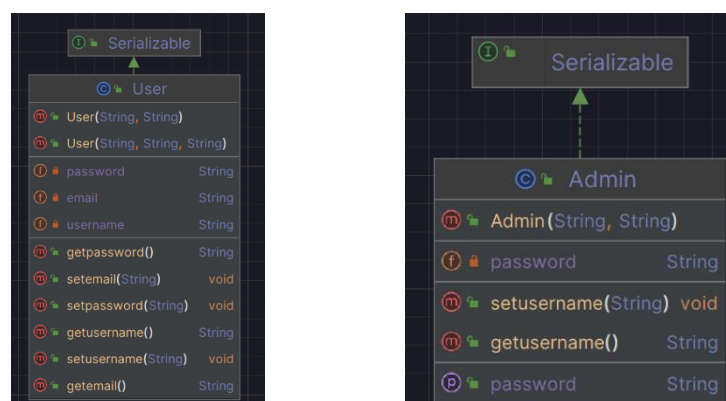
Objednávky

Dedenie pri objednávkach je pokročilejšie. Použil som aj rozhranie `OrderInterface`. Hlavná trieda `Order` je abstraktná trieda. Od tejto triedy dedí trieda `OrderExtend` ktorá obsahuje navyše aj počasie. A od tejto triedy dedia ďalšie dve triedy a to konkrétne `OrderDiscountSenior` a `OrderDiscountStudent`. V týchto triedach sa pomocou [implicitnej implementácie metód v rozhraní](#) prepočítava cena. Polymorfizmus pri objednávkach som využil pri metóde `price` a pri metóde `weatherPicture`. Metóda `price` slúži na prepočítavanie ceny, ktorú ovplyvňujú rôzne faktory a metóda `weatherPicture` slúži na priradenie obrázku podľa zvoleného počasia. Zapuzdrenie je využité taktiež v týchto triedach ako je vidieť na diagrame a k jednotlivým údajom pristupujem pomocou getterov a setterov. V triede `Order` využívam agregáciu. A to konkrétne pri `captain`, `user`, `boat`, `route`.



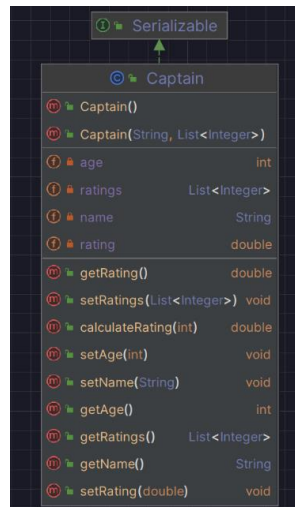
Používatelia

Triedy pre správcu a používateľa. Tieto triedy sú založené na pomerne jednoduchom princípe obsahujú iba pár údajov, ktoré sú samozrejme prívátne. Čiže je využité zapuzdrenie. Používam pri daných triedach aj serializáciu ako je vidieť v diagrame.



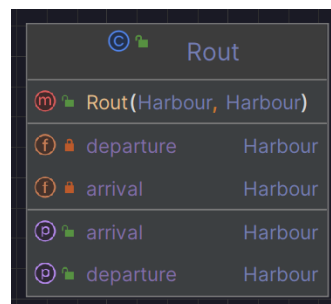
Kapitán

Trieda kapitán obsahuje základná informácie o danom objekte kapitán. Na prístupovanie využívam gettre a settre. Taktiež je využité zapuzdrenie. Trieda kapitán sa využíva pri tvorbe objednávky.



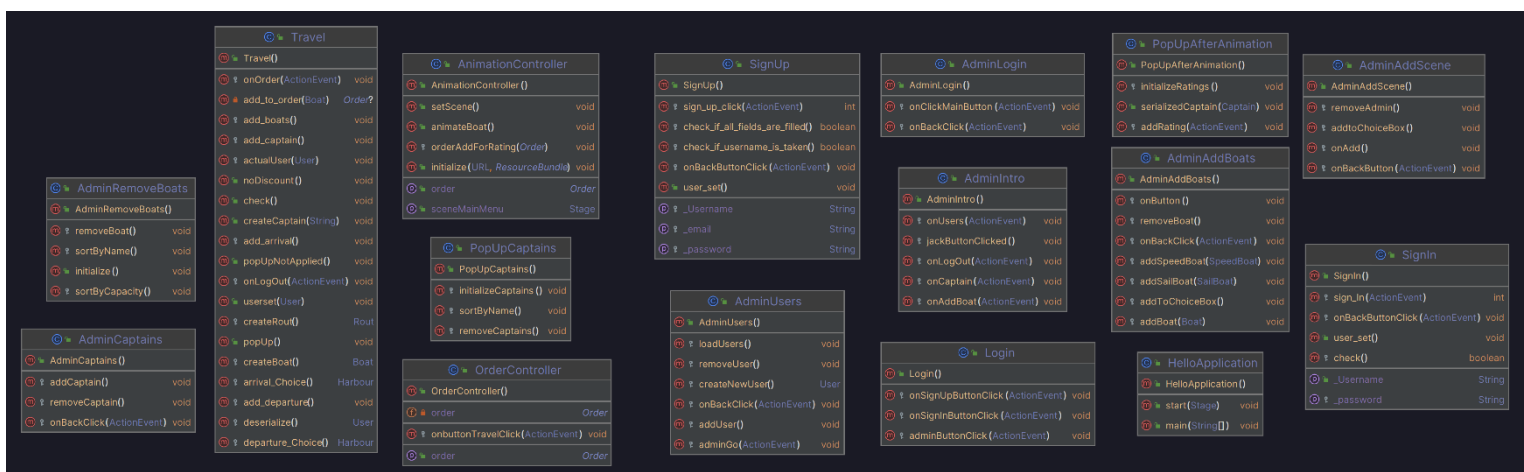
Cesta

Ďalší objekt ktorý obsahuje objednávka je cesta. Objekt cesta obsahuje objekt začínajúceho prístavu a objekt cieľového prístavu. Ospravedlňujem sa za menší preklep v názve. Všimol som si ho až pri odovzdávaní a radšej som ho už nemenil.



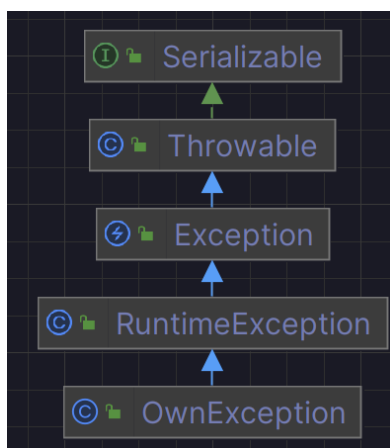
Kontrolery

Na manipuláciu s jednotlivými scénami som použil tieto triedy. Každá jedna trieda slúži pre jednu scénu v aplikácii. V jednotlivých triedach sú použité metódy pre vykonávanie jednotlivých akcií v danej scéne.



Vlastná výnimka (OwnException)

Na ošetrenie mimoriadnych stavov som vytvoril triedu OwnException. Táto trieda slúži na vytvorenie vlastného typu výnimky (exception) pre program. Trieda má jediný konštruktor, ktorý prijíma reťazec (String) ako parameter. Tento reťazec obsahuje správu, ktorá popisuje výnimku, ktorá nastala. Konštruktor volá konštruktor triedy "RuntimeException" so zadanou správou, aby inicializoval novú výnimku s touto správou. Výnimku som využil napríklad pri pripájaní do databázy. Ak sa programu nepodarí pripojiť na databázu vypíše chybovú správu s použitím vlastnej výnimky s textom "Error connecting to database". Taktiež som uplatnil vlastnú výnimku v prípade ak admin pridáva nového používateľa alebo používateľ sa registruje a zadá email v nekorrektnnej forme. A v hlavnom menu ak používateľ zadá rovnaké destinácie vyskočí alert box s vlastnou výnimkou.

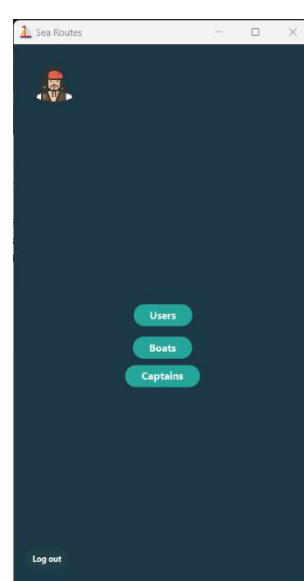
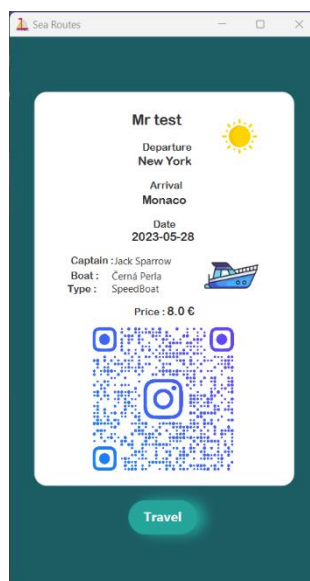
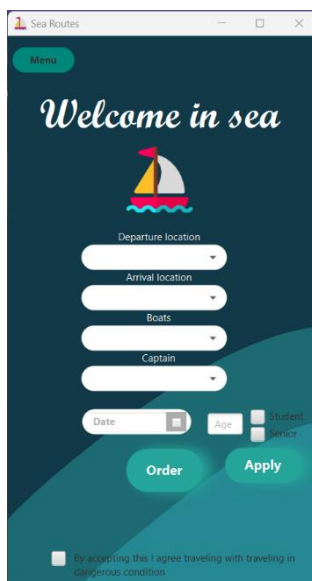


```
public class OwnException extends RuntimeException{  
    /**  
     * Constructor for OwnException  
     * own exception for the program  
     * @param message  
     */  
    public OwnException(String message) {  
        super(message);  
    }  
}
```

```
public void connect(String uri, String databaseName, String collectionName)  
{  
    try{  
        client = MongoClient.create(uri);  
        database = client.getDatabase(databaseName);  
        collection = database.getCollection(collectionName);  
    } catch (Exception e) {  
        throw new OwnException("Error connecting to database");  
    }  
}
```

Grafické používateľské rozhranie

Vďaka intuitívnemu grafickému používateľskému rozhraniu (GUI), je aplikácia jednoduchá a pohodlná na používanie pre používateľov. S prehľadným rozložením a logickým usporiadaním prvkov na obrazovke, používatelia sa môžu ľahko navigovať cez aplikáciu a vykonávať úlohy s minimálnym úsilím. GUI je oddelené od aplikačnej logiky a jednotlivé scény sú kontrolované cez [kontrolery](#). Jednotlivé scény sú navrhnuté pomocou aplikácie SceneBuilder a dizajnované pomocou CSS. Ukážka pár scén z aplikácie.



Vlastná generická trieda

Trieda `GenericFilter` slúži na filtrovanie zoznamov objektov na základe podmienky zadanej pomocou `Predicate` rozhrania. Trieda obsahuje jednu premennú `predicate`, ktorá je typu `Predicate<T>`. Táto premenná uchováva podmienku, podľa ktorej sa bude zoznam filtrovať. Metóda `filter` triedy prijíma zoznam objektov typu `List<T>` a vráti nový zoznam, ktorý obsahuje len tie prvky pôvodného zoznamu, ktoré spĺňajú podmienku uloženú v premennej `predicate`. Metóda využíva `stream()` na iteráciu zoznamu a následne `filter()` na vyfiltrovanie prvkov podľa zadaného predikátu. Výsledný zoznam je potom vrátený pomocou `collect()` a `Collectors.toList()`. Vlastnú generickú triedu používam v aplikácii na usporiadanie jednotlivých prvkov v `Choice Boxoch`. Napríklad v admin menu môže admin zoradiť objekty lodí podľa kapacity alebo podľa názvu. Taktiež vlastnú generickú triedu využívam pri zoradení objektov `User` v admin menu podľa názvu.

```
/**
 * GenericFilter class
 * using for sorting list of objects in my program
 * @param <T>
 * @return list of objects sorted
 */
public class GenericFilter<T> {
    private Predicate<T> predicate;

    public GenericFilter(Predicate<T> predicate) {
        this.predicate = predicate;
    }

    public List<T> filter(List<T> list) {
        return list.stream()
            .filter(predicate)
            .collect(Collectors.toList());
    }
}
```

Explicitné použitie RTTI

Program používa RTTI (Run-Time Type Information), čo znamená, že umožňuje identifikovať typ objektu počas behu programu. Táto funkcia umožňuje programu dynamicky spracovávať objekty a prispôbovať ich správanie na základe ich typu. RTTI som použil napríklad v nasledujúcich dvoch príkladoch. V prvom prípade sa používa operátor `instanceof`, ktorý slúži na overenie, či je určitý objekt inštanciou určitého typu triedy. V tomto prípade sa overuje, či objekt `orderforAnimation` je inštanciou triedy `OrderExtend` a ak áno vykonajú sa ďalšie operácie. Ak by sme sa totiž snažili získať názov obrázka z objektu, ktorý nie je typu `OrderExtend`, mohlo by dôjsť k chybám a nežiadúcemu správaniu programu. V druhom prípade používam metódu `getClass()` umožňuje získať informácie o type objektu počas behu programu, čo umožňuje dynamické spracovanie objektov v závislosti na ich type. A následne vykoná nadväzujúce operácie. V tomto prípade skryje label v GUI.

```
if (orderforAnimation instanceof OrderExtend) {
    String weather = orderforAnimation.weatherPicture();
    weatherPic.setImage(new Image(Objects.requireNonNull(getClass().getResource(weather)).openStream()));
}
```

```
if (order.getBoat().getClass().equals(Boat.class)) {
    type.setVisible(false);
    boatTypeLabel.setVisible(false);
}
```

Lambda výrazy

Lambda výrazy v mojom programe sú využívané v dostatočne veľkej miere najmä pri usporadúvaní. Na prvom a druhom obrázku kódu lambda výraz definuje predikát pre [vlastnú generickú triedu](#). Na treťom obrázku lambda výraz definuje komparátor, ktorý porovnáva mená dvoch používateľov podľa abecedy bez ohľadu na veľkosť písmen. Na štvrtom obrázku sa lambda výraz používa na definovanie funkcie, ktorá sa použije na porovnanie dvoch prvkov Listu v rámci metódy sorted(). V mojom prípade zoradím lode podľa ich kapacity.

```
// Filter using predicate
Predicate<Document> alphabeticalIdPredicate = (doc) -> {
    String[] words = doc.get("_id").toString().split( regex: "\\s+" );
    if (words.length > 0) {
        String firstWord = words[0];
        return firstWord.matches( regex: "\\p{L}+" );
    }
    return false;
};
```

Obrázok 1

```
// Create predicate to filter by alphabetical order of name
Predicate<User> alphabeticalNamePredicate = (user) -> user.getUsername().matches( regex: "[A-Za-z]+" );
```

Obrázok 2

```
// Create comparator to sort by name
Comparator<User> nameComparator = (u1, u2) -> u1.getUsername().compareToIgnoreCase(u2.getUsername());
```

Obrázok 3

```
// Sort using Java Streams by 'capacity' field
List<Document> sortedList = docsList.stream()
    .sorted(Comparator.comparing(d -> (int) d.get("capacity")))
    .collect(Collectors.toList());
```

Obrázok 4

Default method implementation

Default method implementation som použil v rozhraní s názvom Discountable, ktoré má jednu metódu calculateDiscount s implicitnou implementáciou. Metóda calculateDiscount vypočíta zľavu na základe veku zákazníka. V prípade, že zákazník je mladší ako 18 rokov, aplikuje sa 20% zľava. Ak zákazník má 60 alebo viac rokov, aplikuje sa 15% zľava. V opačnom prípade sa neaplikuje žiadna zľava. Použitie implicitnej implementácie metódy v rozhraní Discountable znamená, že ak trieda neposkytuje vlastnú implementáciu tejto metódy, bude použitá predvolená implementácia, ktorá je poskytnutá v rozhraní. To znamená, že trieda nemusí explicitne implementovať metódu calculateDiscount, ale môže ju použiť priamo vo svojej implementácii. V mojom programe je využitá pri vypočítavaní zľavy na lístok ak používateľ zapadá do danej kategórie.

```
public interface Discountable {
    /**
     * Calculates the discount based on the age of the customer
     * Default implementation is provided
     * @param age
     * @return
     */
    default double calculateDiscount(int age) {
        if (age <= 18) {
            // Apply 20% discount for customers under 18
            System.out.println("Student discount applied");
            return 0.8;
        } else if (age >= 60) {
            // Apply 15% discount for senior citizens
            System.out.println("Senior citizen discount applied");
            return 0.75;
        } else {
            // No discount applied
            System.out.println("No discount applied");
            return 0;
        }
    }
}
```


Serializácia

Serializácia je v mojom programe využitá viackrát. Napríklad do serializovaného súboru ukladám objekty adminov. Adminov ukladám do súboru s názvom serialization.ser z ktorého potom pri prihlasovaní admina kontrolujem korektnosť zadaného hesla. Ďalej používam serializáciu na ukladanie aktuálny objekt používateľa alebo kapitána pri prechode medzi scénami. Ďalšie uplatnenie serializácie v mojej aplikácii je využité na hodnotenie kapitánov. Do serializovaného súboru captains.ser ukladám všetkých kapitánov a tak uchovávam aj hodnotenie z prechádzajúcich plavieb on iných používateľov čo mi umožňuje výpočet ratingu kapitána. Na základe ratingu si môže používateľ vybrať spoľahlivejšieho kapitána.

Návrhové vzory

Factory

Factory pattern umožňuje vytvárať objekty bez toho, aby sme poznali presnú triedu objektu. Môj program obsahuje metódu ktorá vytvára objekty triedy Boat, ale môže vytvárať aj podtypy tejto triedy - SailBoat a SpeedBoat - na základe zadaných parametrov. Takto implementovaný Factory pattern umožňuje jednoduché vytváranie rôznych typov lodí pomocou jednej metódy, čo zjednodušuje kód a zvyšuje jeho prehľadnosť a flexibilitu.

Builder

Tento návrhový vzor umožňuje vytvárať objekty s množstvom rôznych vlastností pomocou jedného a toho istého objektu builderu. Tento objekt builderu má rôzne metódy na nastavenie jednotlivých vlastností objektu a na konci vracia hotový objekt. V mojom programe som využíval pri vytváraní jednotlivých objektov či už objednávky, lode, užívateľa. Napríklad pri implementovaní návrhového vzoru [Factory](#) trieda BoatBuilder slúži ako základ pre SailBoatBuilder a SpeedBoatBuilder, ktoré implementujú špecifické vlastnosti pre dané typy lodí. Konkrétny typ člna sa určuje na základe parametrov metódy createBoat triedy Factory. Vytvorenie objektu SailBoat a SpeedBoat sa potom uskutočňuje pomocou príslušného builderu, ktorý sa používa na nastavenie vlastností objektu a nakoniec vráti hotový objekt.

Facade

Tento typ návrhového vzoru som použil na zjednodušenie prístupu k MongoDB databáze. Trieda DatabaseFacade slúži ako rozhranie pre klienta na pripojenie k databáze a získanie kolekcie. Týmto spôsobom je skrytá komplexita spojená s pripojením a manipuláciou s databázou od klienta.

Konkrétne, trieda DatabaseFacade poskytuje jednoduché metódy pre pripojenie k databáze cez metódu connect, následne získanie kolekcie s dokumentami pomocou getCollection, a vykonávanie dotazov pomocou find. Vďaka návrhovému vzoru Facade, ktorý je použitý v tejto triede, je jednoduché používať MongoDB databázu bez nutnosti poznania podrobností o komplexnom pripojení a manipulácii s databázou.

Splnené kritéria

Tak ako je už spomenuté v predchádzajúcej časti tak pri triedach jednotlivých objektov je využité zapúzdrenie, dedenie, polymorfizmus a pri triede pre [objednávku](#) (Order) je využitá agregácia. Taktiež na týchto triedach sú aplikované rozhrania ako je vidieť z [diagramu](#). Základná trieda Order je abstraktná trieda. Taktiež som dbal na oddelenie aplikačnej logiky od používateľského rozhrania. Jednotlivé scény sú kontrolované pomocou tried kontrolerov. Celý program je prehľadne usporiadaný v balíkoch, ktoré sú prehľadne usporiadané. Jednotlivé scény pre [GUI](#) sú taktiež usporiadané v balíkoch aj s obrázkami a CSS pre dané scény. K bonusovým kritériám. Ako je vidieť v [štruktúre systému](#) do môjho programu som implementoval [návrhové vzory](#), [vlastnú výnimku](#), [grafické používateľské rozhranie](#), [vlastnú generickú triedu](#), [RTTI](#), [lambda výrazy](#), [default method implementation](#), [serializáciu](#). Funkcionalita a presné uplatnenie je bližšie špecifikované [štruktúre systému](#).

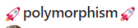





































































Databáza

V aplikácii je využívaná databáza MongoDB, z ktorej sú nahrávané dáta o kapitánoch, lodiach a prístavoch, s ktorými aplikácia pracuje. Databáza obsahuje informácie o používateľoch aplikácie, ktoré sú tiež používané v rámci funkcionality aplikácie. Z databázy sa údaje načítavajú počas behu aplikácie podľa toho ktoré dáta potrebujeme z nej dostať. Tento spôsob ukladania dát mi prišiel najjednoduchší aj najpraktickejší. S databázou sa pracuje jednoducho a rýchlo. I keď na začiatku bolo mierne komplikovanejšie nastaviť pripojenie k databáze ale ďalšie procesy už boli intuitívne a ľahké na pochopenie.

GitHub

Verzie programu som commitoval do GitHub classroomu. Ukladal som veľmi často, či už po drobných zmenách alebo väčších úpravách. Každý commit obsahuje názov čo som zmenil, pridal, odstránil. Keďže na projekte som pracoval dlho a pomerne často, verzii mojej aplikácií na GitHube je pomerne dosť takže popíšem len tie hlavné. Pod každým obrázkom sa nachádza stručný popis čo sa pridalo do danej verzii.

 first commit  matejdrienovsky committed on Mar 15	 079d1bf <>
Inicializovanie repozitáru	
 sign in  matejdrienovsky committed on Mar 20	 2bdf59 <>
login page  matejdrienovsky committed on Mar 20	 b2c7216 <>
Scéna pre prihlasovanie do aplikácie a samotné prihlasovanie	
sign-up database  matejdrienovsky committed on Mar 21	 3eeeb63 <>
database  matejdrienovsky committed on Mar 21	 222394f <>
Registrácia a nastavenie databázy	
import boats and captains database  matejdrienovsky committed on Mar 23	 a5b95be <>
Pridanie databázy s loďami a kapitánmi	
create order  matejdrienovsky committed on Mar 27	 96e123e <>
Vytvorenie objednávky	
inheritance  matejdrienovsky committed on Mar 30	 cb983ad <>
Pridanie dedenia	

  matejdrienovsky committed on Mar 31	 b973e28 
ticket  matejdrienovsky committed on Mar 31	 48430e2 
<i>Scéna pre listok a pridanie polymorfizmu</i>	
animation  matejdrienovsky committed on Mar 31	 de996f1 
<i>Pridanie animácie pohybu loďky</i>	
boats speed  matejdrienovsky committed on Apr 1	 f75240b 
<i>Upravenie animácie tak, že bude záležať na rýchlosti loďky</i>	
senior discount update  matejdrienovsky committed on Apr 6	 e0b8594 
<i>Rozšírenie triedy order o triedu so zľavami</i>	
double check age discount  matejdrienovsky committed 2 weeks ago	 a72ab92 
<i>Pridanie default method implementation</i>	
OwnException  matejdrienovsky committed last week	 e34e4d5 
<i>Pridanie vlastnej výnimky</i>	
generic  matejdrienovsky committed last week	 1d82858 
sorting boats by capacity  matejdrienovsky committed last week	 1d1b888 
removing users from database  matejdrienovsky committed last week	 fcd2053 
Lambda sorting  matejdrienovsky committed last week	 4ea2862 
<i>V týchto verziách boli pridané lambda výrazy, v admin menu vymazávanie používateľov, usporadúvanie lodí podľa kapacity a vlastná generická trieda</i>	
 serialization  matejdrienovsky committed last week	 88c9c1b 
<i>Pridanie serializácie</i>	
design patterns  matejdrienovsky committed 4 days ago	 bcb29a5 
<i>Pridanie návrhových vzorov</i>	
CSS updates  matejdrienovsky committed 4 days ago	 0e62091 
abstract  matejdrienovsky committed 4 days ago	 2e3a936 
<i>Pridanie abstraktnej triedy a upravenie dizajnu aplikácie</i>	
facade to connecting  matejdrienovsky committed 3 days ago	 1201995 
<i>Pridanie návrhového vzoru Facade</i>	
ratings of captains  matejdrienovsky committed 3 days ago	 da7071d 
<i>Pridanie hodnotenia kapitánov</i>	
after travel you can create new order  matejdrienovsky committed 2 days ago	 3713711 
<i>Opravenie plynulosti aplikácie(po dokončení plavby nebolo možné prejsť znova na ďalšiu problém bol so setnutím usera)</i>	
remove some prints  matejdrienovsky committed 2 days ago	 fd26e55 
<i>Odstránenie zbytočných printov v terminály</i>	
comments   matejdrienovsky committed 2 days ago	 d472e9f 
<i>Pridanie komentárov</i>	
JavaDoc   matejdrienovsky committed now	 2b9ea6d 
<i>Pridanie JavaDoc</i>	
The last dance  matejdrienovsky committed now	 4d93440 
<i>Finálna verzia programu</i>	

Návod k používaniu aplikácie

Pre spustenie programu je potrebné pripojenie na internet z dôvodu pripojenia k [databáze](#). Program sa spúšťa cez triedu v priečinku `com.example.searoutes` a názov triedy `HelloApplication`. Po spustení programu sa spustí intuitívne GUI kde je možnosť prihlásiť/zaregistrovať sa alebo po kliknutí na panáčika v ľavom dolnom rohu spustiť správcovské (admin) menu. Z každej scény je možné sa vrátiť späť šípkami v ľavom dolnom rohu okrem scény kde je vytlačený lístok tam už treba postupovať ďalej. Pri menu kde sa vyberajú možnosti trasy je v ľavom hornom rohu rolovacie menu na ktoré keď kliknete ukáže sa možnosť pre odhlásenie z užívateľského menu. Pri overovaní veku treba dávať pozor na text ktorý vypíše pop-up okno. V prípade , že vek nie je správne overený (vek nespadá pod danú kategóriu zľavy) je potrebné odznačiť checkbox alebo zmeniť vek. Taktiež ak sa plavba skončí vyskočí pop-up okno pre hodnotenie kapitána, ak toto okno zavriete program prejde znova do menu pre výber trasy. V správcovskom menu je možné sa odhlásiť pomocou tlačidla vľavo dole a taktiež z nasledujúcich scén prejsť naspäť pomocou šípok vľavo dole. Po zvolení tlačidla `Users` v správcovskom menu je v nasledujúcej scéne vpravo dole tlačidlo `Admins` ktoré nás presmeruje do časti kde je možno pridávať/mazať správcov. Dúfam, že z dokumentácie je všetko jasné. Prajem príjemné používanie aplikácie `SeaRoutes` a želám vám príjemné plavby - nech vás oceán zavedie tam, kam vaše srdce túži.

Prihlasovacie údaje používateľa

Prihlasovacie meno: **test**

Heslo: **test**

Prihlasovacie údaje správcu

Heslo: **test**

UML diagram

