**ŽILINSKÁ UNIVERZITA V ŽILINE**

FAKULTA RIADENIA A INFORMATIKY

Návrhové vzory

MATEJ MAŽGÚT

**Semestrálna práca**

# Zadanie a popis problému

Hlavnou úlohou semestrálnej práce bolo vytvoriť elektronický zdravotný systém, v ktorom bude možné zakladať nové pobočky, ako sú nemocnice a evidovať o nich základné informácie o názve kapacite a zozname hospitalizovaných pacientov. Bude teda potrebné, aby používateľ bol schopný zaregistrovať nového pacienta a priradiť ho do konkrétnej nemocnice.

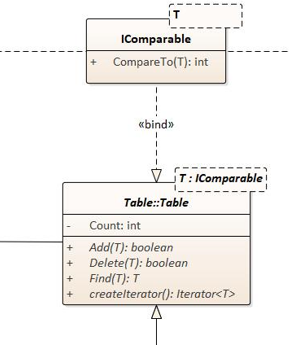
Ďalšou z požiadaviek bolo, aby si používateľ na začiatku mohol vybrať s akou konkrétnou implementáciou tabuľky, teda štruktúrou, bude aplikácia pracovať. A bolo možné ľahko pridávať nové implementácie. Ako prvé bolo potrebné implementovať štruktúru binárneho vyhľadávacieho stromu. Aby strom časom nezdegeneroval na lineárny zoznam, trepalo navrhnúť aj vlastný algoritmus na vyváženie stromu.

Stanovil som si ešte jeden ciel a to aby bolo možné aplikáciu jednoducho rozšíriť a využiť ju aj ako iný elektronický systém. Evidovať napríklad informácie o policajných staniciach a zadržaných osobách alebo školách a aktuálnych študentoch.

# Návrh

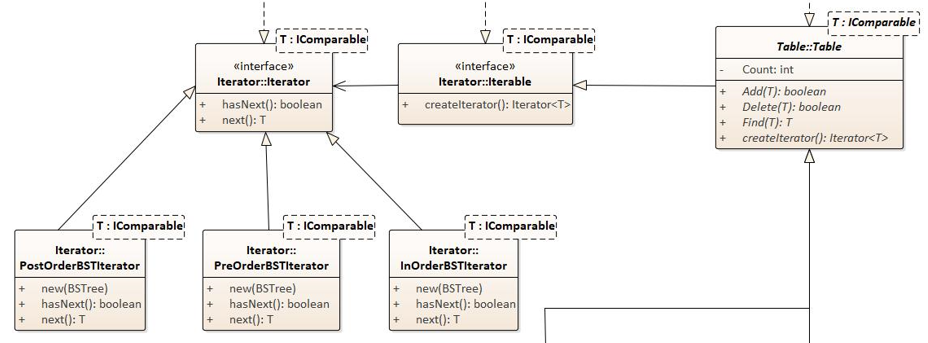
Ako prvé som si vytvoril návrh aplikácie v podobe UML diagramu, kde som použil viacero návrhových vzorov. Aplikáciu som si rozdelil do dvoch častí a to:

## DataStructure

Táto časť aplikácie bude slúžiť na jednoduché rozšírenie o ďalšiu konkrétnu implementáciu tabuľky. Predkom každej implementácie bude abstraktná trieda Table, ktorá je zobrazená na Obr. 1.

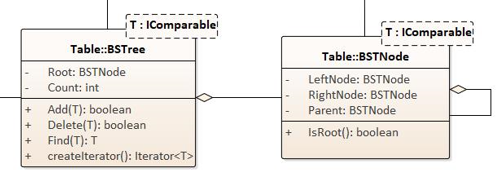
Obrázok 1 : Trieda Table

Trieda obsahuje 4 abstraktné metódy, ktoré musí každá implementácia, teda potom tabuľky prekryť. Šablónový parameter T bude musieť byť ešte potomkom triedy IComparable, aby bolo možne prvky v tabuľke porovnávať a tak ich umiestniť na správne miesto. Nie je potom potrebné evidovať si kľuč pre každý prvok tabuľky, ale stačí zavolať metódu CompareTo a porovnať dva prvky.

V triede sa nachádzajú štyri abstraktné metódy a to tri pre prácu s prvkami tabuľky ako pridaj, vymaž a nájdi prvok a ešte metóda createIterator, ktorá vráti potomka triedy Iterator. Tu je použitý prvý návrhový vzor a to Iterator. Pomocou neho vieme prechádzať sekvenčne všetky prvky v tabuľke. Ak chceme aby to bolo možne musíme danú štruktúru definovať ako potomka triedy Iterable. Celý návrh vzoru Iterator je na Obr. 2.

Obrázok 2: Iterator

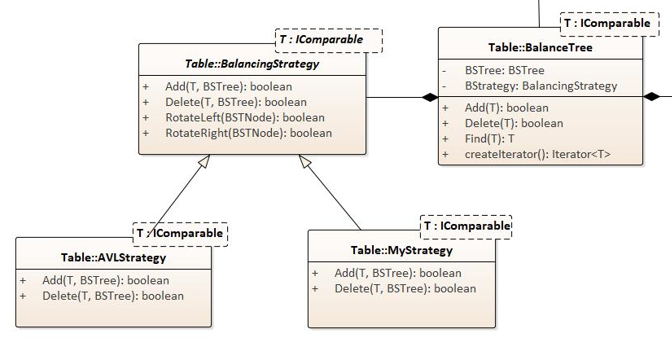
Každú tabuľku budeme môcť sekvenčné prechádzať, pretože predok Table je potomkom triedy Iterable. Konkrétna implementácia tabuľky vráti už konkrétnu implementáciu triedy Iterator. Trieda Iterator obsahuje iba dve metódy a to metódu na vrátenie nasledujúceho prvku a metódu na zistenie či existuje nasledujúci prvok.

Na ďalšom obrázku (Obrázku 3) môžete vidieť už konkrétnu implementáciu tabuľky a to Binárny vyhľadávací strom.

Obrázok 3:BSTree

V tomto prípade som využil návrhový vzor Composite, keďže sa každý strom skladá z koreňa typu BSTNode a každý BSTNode sa skladá ešte z ďalších BSTNode ako je pravý, ľavý syn a otec.

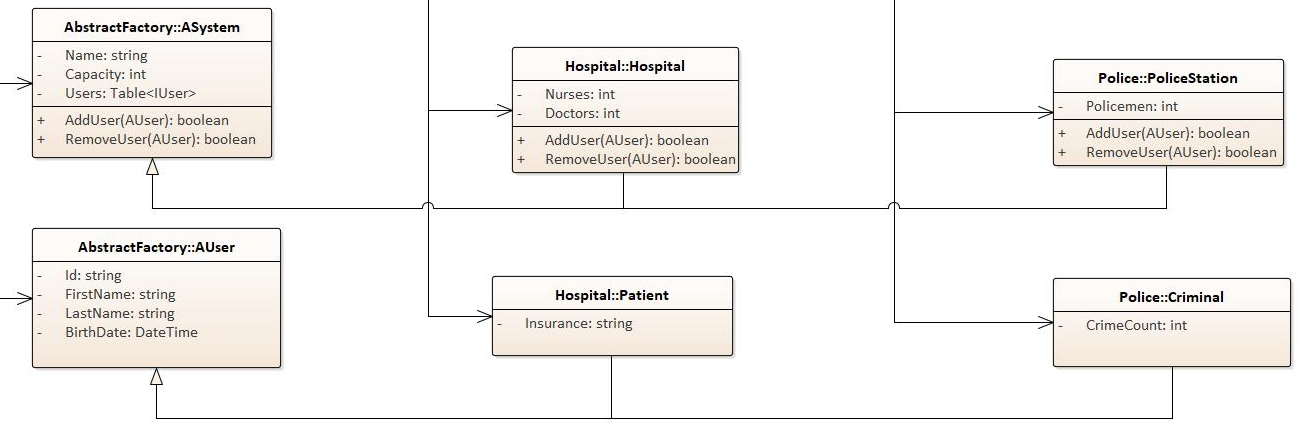
Ako ďalšie som musel vymyslieť akým spôsobom budem binárny vyhľadávací strom balancovať. Keďže existuje niekoľko algoritmov pre túto operáciu, použil som tu návrhový vzor Strategy, pre jednoduché doplnenie iného balancovacieho algoritmu. Implementácia je zobrazená na Obrázku 4.



Obrázok 4:Balance Tree

Vytvoril som triedu BalnceTree, ktorá obsahuje dva atribúty a je potomkom triedy Table. Jeden atribút predstavuje samotný binárny vyhľadávací strom a druhý stratégiu, akou bude prebiehať balancovanie pri pridávaní a odstraňovaní prvkov. V abstraktnej triede BalanceStrategy sa nachádzajú dve abstraktné metódy pre pridanie prvku a odstránenie prvku, ktoré musia potomkovia prekryť. Takmer každá stratégia využíva pre balancovanie rotácie a tak som ich implementoval už v abstraktnom predkovi.

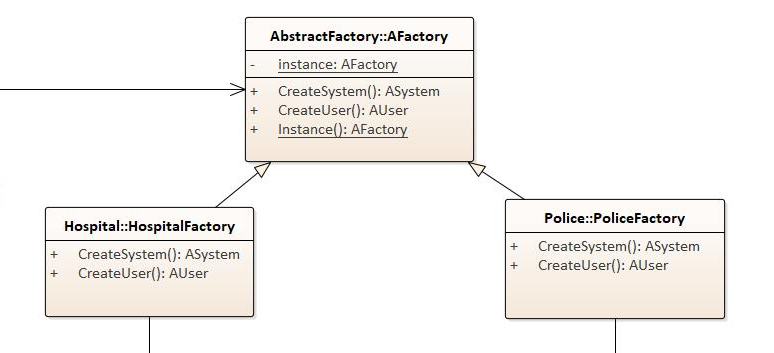
## App

Druhá časť programu je samotná aplikácia. Aby som použiteľnosť aplikácie rozšíril a nebol to len systém pre správu nemocníc a pacientov, ale aj iné podobné systémy, využil som tu návrhový vzor Abstarct Factory. Rodinu súvisiacich produktov tvoria dve základné triedy a to ASystem a AUser zobrazené na Obrázku 5.

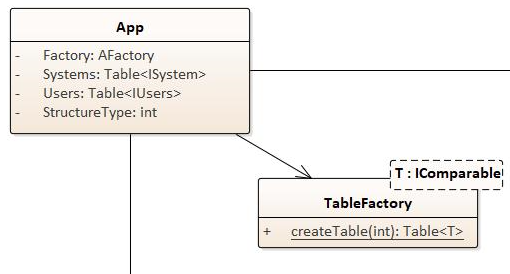
Obrázok 5:Abstract Factory

Obrázok 6

Konkrétny potomkami týchto tried už budú nemocnica, policajná stanica a pacient, kriminálnik. Každá skupina objektov bude mať ešte vlastnú továreň ako je na Obrázku 6.

Predkom každej továrne bude trieda AFactory, ktorá ma metódy pre vytvorenie správnej množiny produktov. Tu som implementoval ako Singleton, keďže v aplikácii budem potrebovať iba jednu, podľa toho s akým systémom pracujem.

Obrázok 6: AFactory

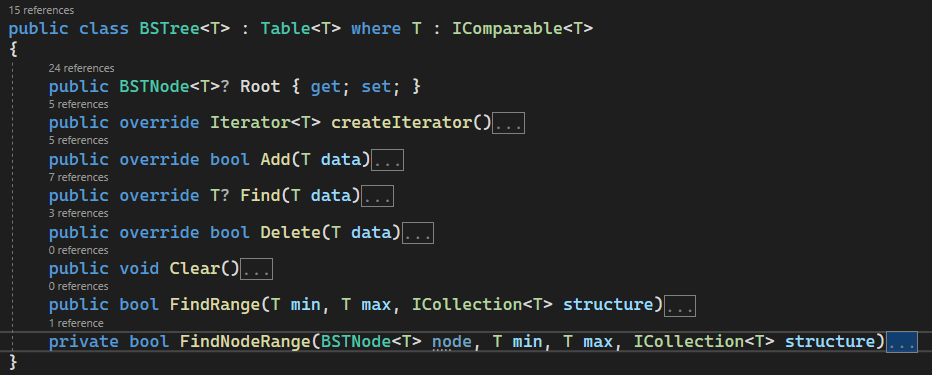
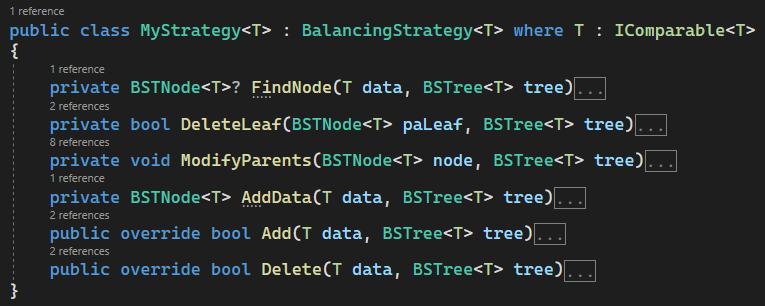
Jadrom celej aplikácie bude trieda App.

Obrázok 7: App

Tá obsahuje atribúty ako konkrétnu továreň pre tvorbu nových Systémov a Používateľov a tiež tabuľky pre ich ukladanie. Ešte tu je atribút StructureType, ktorý určuje s akým typom alebo konkrétnou implementáciou tabuľky bude systém pracovať. Pre ich tvorbu sa tu nachádza ešte trieda TableFactory, slúžiaca na tvorbu tabuliek konkrétneho typu. Tú som implementoval podľa vzoru SimpleFactory method.

# Implementácia a rozšírenia

Aplikáciu som implementoval v programovacom jazyku C#. Vytvoril som si LibraryClass aplikáciu, kde som naprogramoval všetky potrebne veci tykajúce sa časti DataStructure. Ak bude chcieť používateľ pridať novú štruktúru, s ktorou bude aplikácia pracovať, stačí ju implementovať ako potomka triedy Table. Príklad môžete vidieť na Obrázku 8, kde som vytvoril Binárny vyhľadávací strom.

Je tu implementovaná aj štruktúra BalanceTree, kde sa ako parameter konštruktora zadáva algoritmus balancovania. Ten je v aplikácií zatiaľ iba jeden nazvaný ako MyStrategy. Je to potomok triedy BalanceStrategy, a podobne ako je zobrazený na Obrázku 8 vieme pridávať nové stratégie pre balancovanie.

Obrázok 9:MyStrategy

Obrázok 8:BSTRee implementacia

Druhú časť aplikácie som vytvoril vo web frameworku Blazor. Tu som si najskôr naprogramoval celú abstraktnú továreň podľa návrhu a tiež jej konkrétnu implementáciu HospitalFactory. Základ továrne a teda triedu Abstract Factory možete vidiet na Obrázku 10.

Text

Description automatically generatedAk by sme chceli aplikáciu rozšíriť o novú továreň musíme vytvoriť potomka tejto triedy a tiež všetky konkrétne produkty, ktoré systém obsahuje. Následne je potrebné priradiť do atribútu instance konkrétnu továreň, s ktorou bude chcieť používateľ pracovať.

Graphical user interface, text, application, email

Description automatically generatedPosledná trieda, ktorú som implementoval bolo jadro aplikácie a to trieda App. Táto trieda pracuje ešte so všetkými podstránkami aplikácie. Ak chce používateľ pridať, odobrať používateľa zo systému alebo vytvoriť nový systém pracuje s touto triedou a atribútmi s tabuľkami. Výsledná aplikácia potom vyzerá nasledovne:

V jej ľavej časti sa nachádza menu a všetky konkrétne operácie, ktoré môže používateľ vykonať. Na začiatku je zobrazená domovská stránka, kde je potrebné zvoliť štruktúru, s ktorou bude aplikácia pracovať. Bez zvolenia štruktúry nebude možne vykonať žiadnu s funkcionalít.