Fakulta riadenia a informatiky, Žilinská univerzita v Žiline

Algoritmy
a údajové štruktúry
2

2.Semestrálna práca

Zimný
semester

Jozef Staško

Št. skupina
5ZIB11

Obsah

1. Operácie a ich prenosová zložitosť

1.1 Operácia vyhľadania nehnuteľnosti podľa zadaného identifikačného čísla

Podľa zadaného kľúča teda v našom prípade identifikačného čísla, sa vypočíta pomocou hešovacej funkcie index v dynamickom poli blokov , následne sa na tento index pristúpi a prečíta sa celý blok do operačnej pamäte. Následne sa prejdú všetky záznamy a porovnajú sa. Ak sa nájde prvok tak operácia končí. Ak sa prvok nenájde, tak pokračujeme do prepĺňovacieho súboru, kde načítavame každý blok, prehľadáme a porovnáme záznamy, až dokedy nenatrafíme na koniec alebo nenatrafíme v niektorom bloku na hľadaný prvok.

Počet prenosov za predpokladu, že prvok je v hlavnej časti : 1

Počet prenosov za predpokladu, že prvok sa nenachádza v hlavnej časti : N, kde N je počet zreťazených blokov v preplňovacom súbore

1.2 Operácia pridávania nehnuteľnosti podľa zadaných dát

Táto operácia ma viacero výsledkov. Ako prvý krok sa pomocou hešovacej funkcie a identifikačného čísla vypočíta index bloku, na ktorý máme pristúpiť v dynamickom poli.

Následne ak blok ma voľné miesto , čo si zistíme podľa atribútu v operačnej pamäti, zapíšeme daný prvok do daného bloku a operácia končí. To znamená jeden prístup do súboru.

Avšak ak v danom bloku nemám voľné miesto, tak daný blok musím rozdeliť a následne prerozdeliť v ňom dáta.

Spočiatku si zistím, či hĺbka adresára je rovná hĺbke daného bloku, ktorý chcem rozdeliť. Ak áno tak zdvojnásobím adresár. Následne dochádza k rozdeleniu bloku a to tak, že alokujem si nový blok alebo použijem blok ktorý som nemohol uvoľniť a nenachádzajú sa tam žiadne dáta. Po alokovaní nového bloku a pridaní tohto bloku do adresára

(dynamického poľa) prečítam si všetky dáta z daného plného bloku a následne ich rozdelím medzi novo vytvorený blok a starý blok, z ktorého som vytváral nový. Následne tieto dva bloky zapíšem ak som do nich niečo vkladal. To znamená jeden až dva prístupy do súboru. Ak však som neúspešne vložil prvok, pretože som sa zase pokúšal vkladať prvok do plného bloku, tak opakujem postup od začiatku. To zapríčiní buď jeden navyše zápis alebo zase rozdelenie, až do kedy nenatrafíme na ďalšiu možnosť.

Touto možnosťou je, že sme prekročili stanovený limit hešovacej funkcie pre povolený počet bitov na výpočet indexu v adresári. V takom prípade namiesto toho aby sme rozdeľovali daný blok alebo zväčšovali adresár, tak prejdeme do preplňovacieho súboru. V súbore prechádzame zasa bloky v operačnej pamäti a zisťujeme či môžeme vložiť do daného bloku. Neuskutočňujeme načítanie bloku, keďže v operačnej pamäti si zaznamenávame počet prvok v bloku. Ak taký blok nenájdeme tak vytvoríme nový blok alebo znovu použijeme blok , ak taký máme, a vložíme do nového bloku, pričom nemôžeme zabudnúť na zreťazenie daného nového bloku alebo jeho zaznamenanie.

V každom z týchto prípadov sa operácia končí, až keď prvok úspešne vložíme.

Počet presunov :

- Blok v hlavnej štruktúre je voľný : 1
- Blok v hlavnej štruktúre musíme rozdeliť a nájdeme mu miesto : 3
- Blok v hlavnej štruktúre musíme rozdeliť a nenájdeme mu miesto : 2N + 1, pričom N je počet opakovaní ktoré musíme urobiť, aby sme našli voľné miesto
- Vkladanie do preplňovacieho súboru: 1

1.3 Operácia vymazania nehnuteľnosti podľa identifikačného čísla

Táto operácia podobne ako operácia pridanie má viacero možných zložitostí. Začína sa ako stále, výpočtom indexu pomocou hešovacej funkcie. Následne pristúpime na daný blok a prečítame z neho. Ak sa prvok v ňom nachádza, tak vymažeme ho. Ak sa prvok

nenachádza v hlavnej časti, tak smerujeme do preplňujúceho súboru a načítavame blok po bloku a nachádzame prvok. Ak sme našli prvok vymažeme a pokračujeme ďalej v algoritme ak však sme prvok nedokázali nájsť a vymazať je koniec.

Nájdenie daného prvku nás stojí bez preplňujúceho súboru jedno načítanie a s preplňujúcim súborom N + 1 , kde N je počet blokov ktoré sme museli načítať až do konca a jednotka , pretože sme si museli najprv overiť hlavnú časť. Ak sme ale našli daný prvok, tak pri vymazaní z hlavnej časti, sú to celkovo dva prenosy (aj za následný zápis, na konci algoritmu) a pre preplňujúci súbor N a celkovo N + 3 (hlavná štruktúra a zápis) , kde N predstavuje počet prehliadnutých blokov a podobne ako pri hlavnej časti si uložíme dáta do dočasnej štruktúry ,pretože budeme ešte v ďalšej časti s tým blokom, z ktorého sme vymazávali, pracovať.

Po úspešnom vymazaní pokračuje algoritmus fázou strasenia. Ak sme prvok vymazali z hlavnej štruktúry, tak prehľadávame každý blok jeho následného zreťazenia a hľadáme či môžeme ušetriť blok v preplňujúcom súbore. Ak nemáme blok v preplňujúcom súbore, tento krok preskočíme. Ak však máme taký blok, postupne pomocou premenných pozeráme, či daný blok nezmestíme do bloku, z ktorého sme vymazali . Ak taký blok nájdeme, tak prečítame z neho dáta, a následne uložíme tieto dáta do pomocnej štruktúry(zapisujeme blok z hlavnej časti až na konci). Z daného bloku dáta odstránime, blok zaradíme medzi prázdne bloky a jeho potomka zreťazíme s predkom. Takto pokračujeme až do konca zreťazenia a teda výsledný počet prenosov je N , kde N je počet blokov, ktorých počet prvkov sa zmestil do bloku z ktorého sme vymazávali.

Ak sme prvok vymazali z preplňujúceho súboru algoritmus je podobný, avšak ak po vymazaní má blok počet prvkov nula, odovzdám ho manažmentu prázdnych blokov. Ak nie, tak pokračujem obdobným algoritmom, avšak nie od bloku v hlavnom súbore, ale od bloku, z ktorého sme mazali v preplňujúcom súbore a nachádzam bloky, z jeho reťazenia, ktoré sa zmestia do daného bloku. Následne ak som prešiel alebo našiel všetky takéto bloky, ešte nezapíšem daný blok. Pozriem sa ešte, či sa blok po zmene nezmestí do hlavného bloku. Ak nie, tak zapíšem blok. Ale ak áno, ušetrím zápis bloku a blok v preplňujúcom súbore a pridám všetky záznamy do hlavného bloku.

Teda, ak má blok počet prvkov nula , tak počet prenosov je nula, ak však idem reorganizovať, tak počet prenosov môžeme definovať ako N + 1, kde N je počet blokov, ktoré sa zmestia do daného bloku a jednotka predstavuje počet zápisov do bloku s tým, že do tejto časti algoritmu sme sa dostali tak, že sme mazali z preplňujúceho súboru a teda vieme, že prvky ktoré nám ostali, máme v pomocnej štruktúre taktiež. Vieme ešte, že viac už s preplňujúcim blokom pracovať nebudeme a teda zapíšeme všetky dáta, ak sme teda nemohli ho ešte spojiť do hlavného bloku a teda ušetriť jeden zápis navyše.

Po fáze strasenia ide fáza spájania. Začíname nájdením suseda hlavného bloku, ak sused neexistuje tak končíme , ak sused existuje, tak pokračujeme. Susedomom sa berie taký blok, ktorý má rovnakú hĺbku , hĺbka je väčšia ako 1 a prefix hešovania majú taký istý.

Ak sused existuje, zistíme, či tieto dva bloky môžeme zjednotiť, podľa počtu prvkov. Ak nie tak končíme. Ak však áno, tak dané bloky zjednotíme, kde obetujeme jeden prenos pri prečítaní zo suseda do bloku, ktorého si stále všetky prvky držíme v operačnej pamäti. Následne suseda odovzdáme manažmentu voľných blokov a nášmu bloku znížime hĺbku.

Ak hĺbka bola posledná daná hĺbka v adresári, tak znížime dĺžku adresára. Táto fáza pokračuje do vtedy, dokedy môžeme zjednocovať bloky.

V poslednej fáze si zapíšeme všetky prvky do vybraného bloku a ukončíme algoritmus tým, že necháme manažérov voľných pamätí sa vysporiadať s blokmi, ktoré vyčnievajú a môžeme skrátiť súbory.

Zhrnutie prenosov :

- Vymazanie z hlavnej štruktúry , bez preplňujúcich blokov a bez spájania : 2
- Vymazanie z hlavnej štruktúry s preplňujúcimi blokmi a bez spájania : 2 + N, kde N je počet blokov, ktoré prečítame z preplňujúcich súboru
- Vymazanie z hlavnej štruktúry s preplňujúcimi blokmi a so spájaním je : 2 + N + M, kde N je počet blokov, ktoré prečítame z preplňujúceho súboru a M je počet prečítaní suseda.

- Vymazanie z hlavnej štruktúry bez preplňujúceho súboru ale so spájaním je: 2 + N, kde N je počet spájaní
- Vymazanie z preplňujúceho súboru bez spájania a bez miesta v hlavnom bloku : 3 + N + M , kde N je počet blokov, ktoré je možne prečítať z preplňujúceho súboru a M je počet blokov potrebných na nájdenie prvku v preplňujúcom súbore.
- Vymazanie z preplňujúceho súboru so spájaním a bez miesta v hlavnom bloku: 3 + N + M + S, kde N je počet blokov, ktoré je možne prečítať z preplňujúceho súboru a M je počet blokov potrebných na nájdenie prvku v preplňujúcom súbore a S je počet prečítaní suseda pri spájaní
- Vymazanie z preplňujúceho súboru bez spájania a s miest v hlavnom bloku : 2 + N + M , kde N je počet blokov, ktoré je možne prečítať z preplňujúceho súboru a M je počet blokov potrebných na nájdenie prvku v preplňujúcom súbore.
- Vymazanie z preplňujúceho súboru so spájaním a s miestom v hlavnom bloku: 2 + N + M + S, kde N je počet blokov, ktoré je možne prečítať z preplňujúceho súboru a M je počet blokov potrebných na nájdenie prvku v preplňujúcom súbore a S je počet prečítaní suseda pri spájaní
-

1.4 Operácia editácie nehnuteľnosti

Operácia editácia je operácia kombinácie vymazania a následného pridania.

2. Použité štruktúry v systéme

V tomto systéme sa použilo tieto štruktúry :

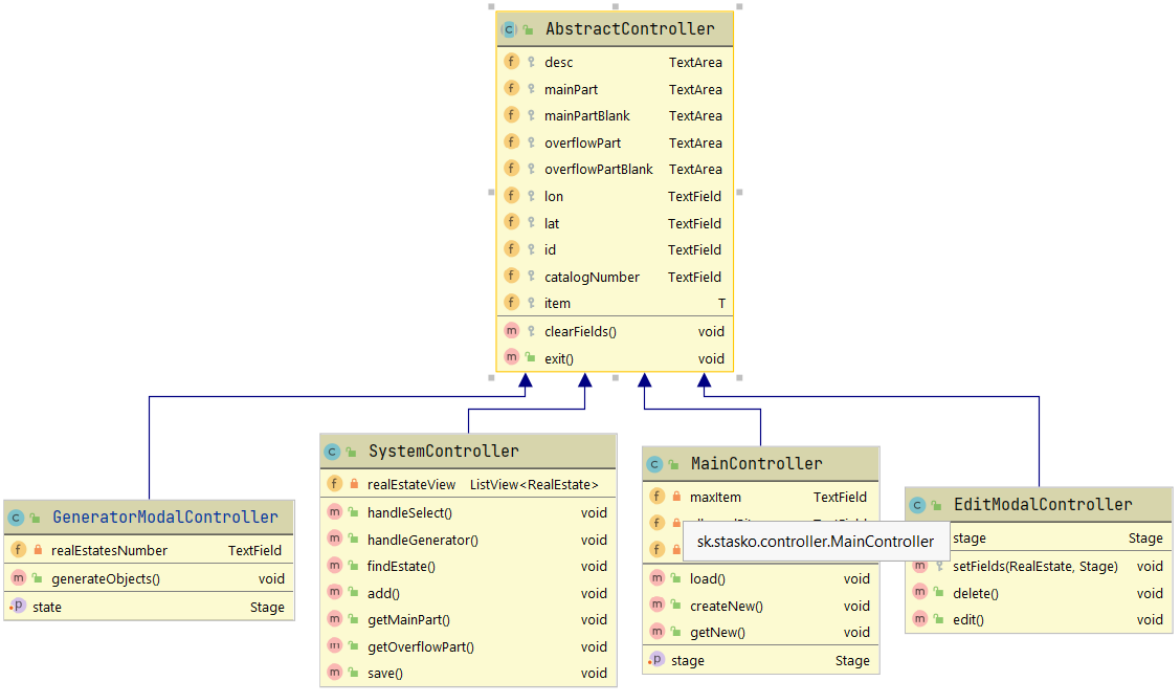
- ArrayList - tento druh listu som použil preto lebo potreboval som dynamické pole, kde som potreboval pristupovať na daný index prvku
- LinkedList – použil som ho preto lebo potreboval som ukladať informácie rovnakého druhu a vedieť ich všetky prehliadnuť a teda preto som si vybral zreťazený zoznam, a aj preto lebo pri týchto dátach nebola potrebná dynamická práca s poľom.
- Rozšíriteľné hešovanie – použil som to kvôli dátam ,ktoré sa ukladám na disk
- Preplňujúci súbor – použil som to pre riešenie kolízií v rozšíriteľnom hešovaní
- Front – FIFO – použil som to kvôli manažmentu prázdnych blokov

3. UML Diagram tried

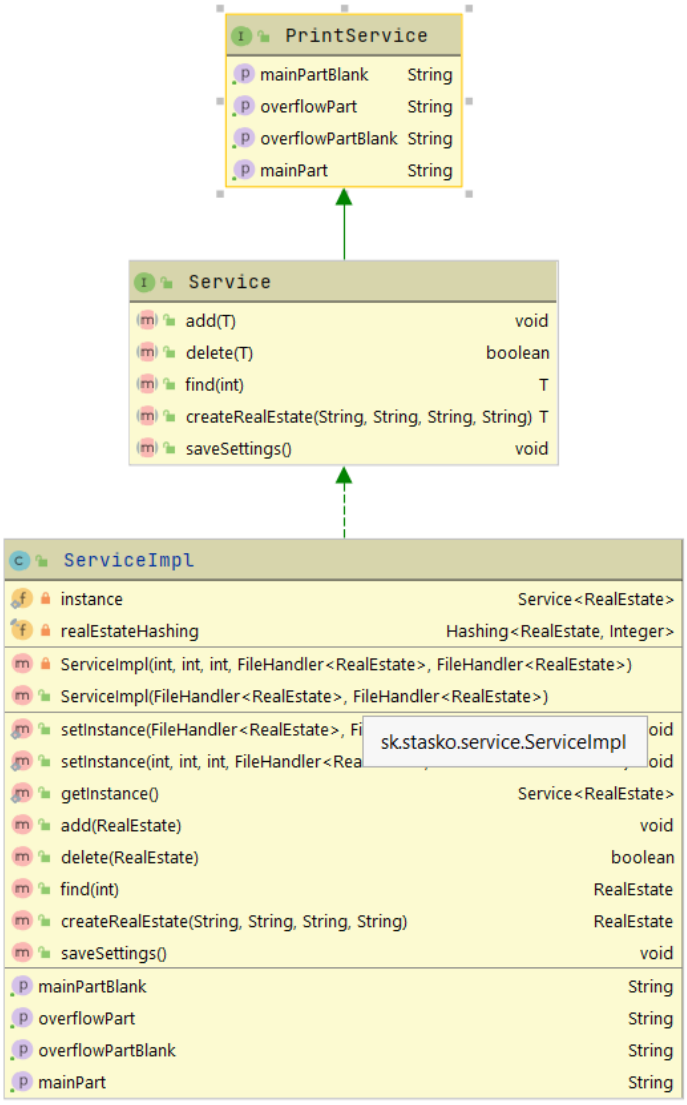
V tomto programe som sa rozhodol používať návrhový vzor MVP, teda Model-View-Presenter. Do môjho návrhu som však pridal jednu vrstvu navyše, ktorú som pomenoval Servis. Táto vrstva je vrstvou medzi presentérom/kontrolérom a modelom. Rozhodol som sa preto z troch dôvodov:

- Odčlenenie biznis logiky do samostatnej vrstvy
- Väčšia obrana vrstvy model – a teda z kontroléra do modelu sa vieme dostať len cez operácie vo vrstve servis a teda nieje tam priamy prístup
- Princíp „Loose coupling“ – a teda kontrolér vie len to čo on potrebuje vedieť na svoje fungovanie a to isté aj model

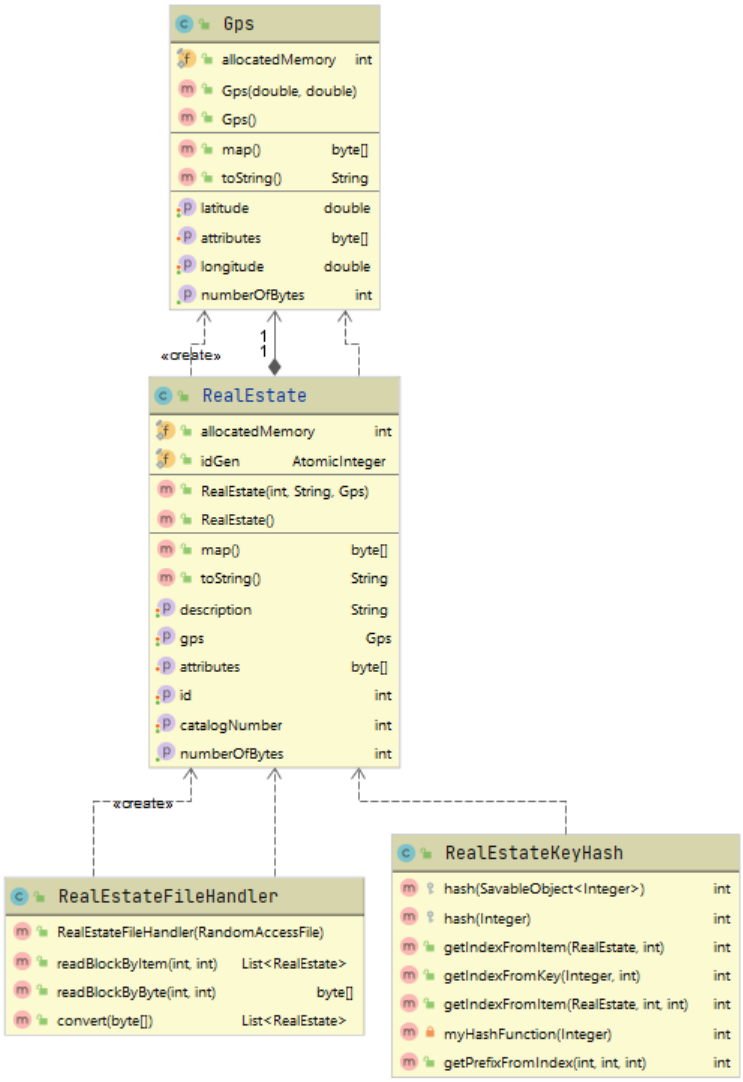
3.1 Presenter/Kontroler



3.2 Service


PrintService
mainPartBlank String
overflowPart String
overflowPartBlank String
mainPart String
Service
add(T) void
delete(T) boolean
find(int) T
createRealEstate(String, String, String, String) T
saveSettings() void
ServiceImpl
instance Service<RealEstate>
realEstateHashing Hashing<RealEstate, Integer>
ServiceImpl(int, int, int, FileHandler<RealEstate>, FileHandler<RealEstate>)
ServiceImpl(FileHandler<RealEstate>, FileHandler<RealEstate>)
sk.stasko.service.ServiceImpl
getInstance() Service<RealEstate>
add(RealEstate) void
delete(RealEstate) boolean
find(int) RealEstate
createRealEstate(String, String, String, String) RealEstate
saveSettings() void
mainPartBlank String
overflowPart String
overflowPartBlank String
mainPart String

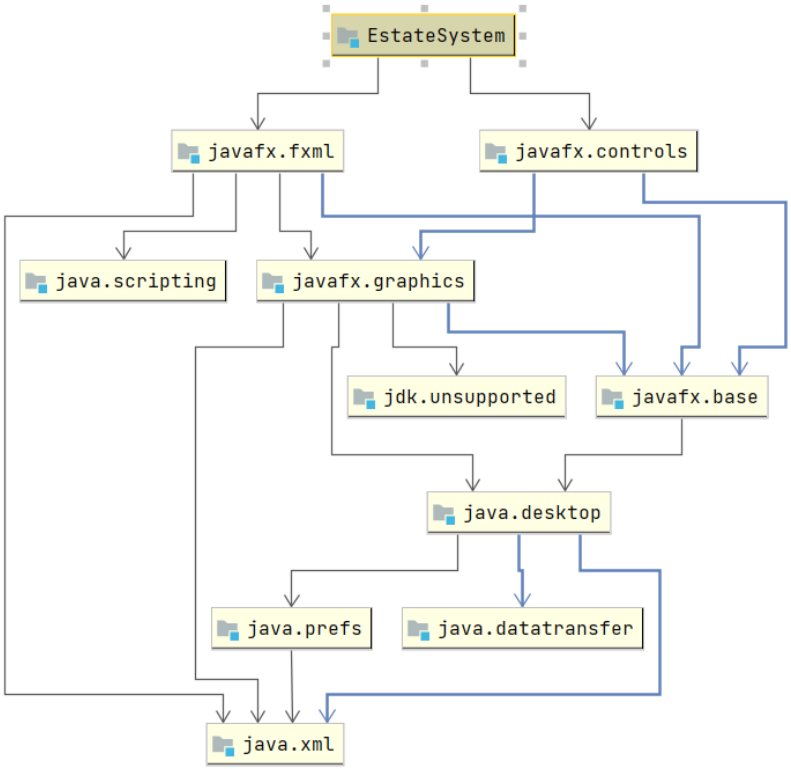
3.3 Model



3.4 View

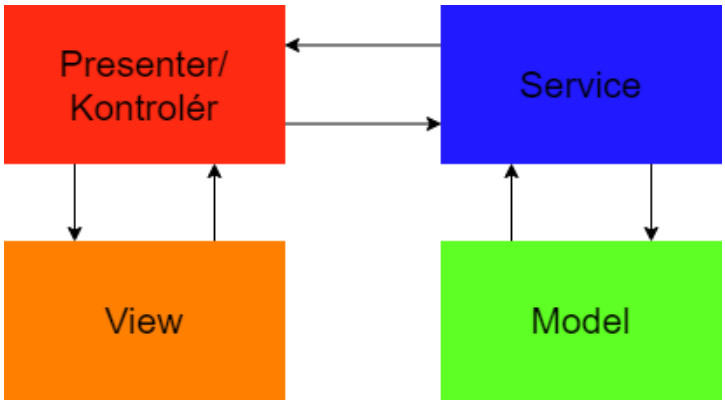
Táto časť je tvorená so súboromi s koncovkou .fxml . Je to jazyk, ktorý je založený na XML a poskytuje možnosť vybudovať používateľské rozhranie separátne od logiky aplikácie Na obrázku môžeme vidieť ako je aplikačná logika oddelená v package sk.stasko od javaFx.xml a aké závislosti a balíčky používa javaFX a jdkFX na to, aby vykreslila

pohľady, ktoré boli navrhnuté v .fxml súboroch. Tieto súbory sú v hierarchii uložené v priečinku resource



3.5 Celý systém MVP

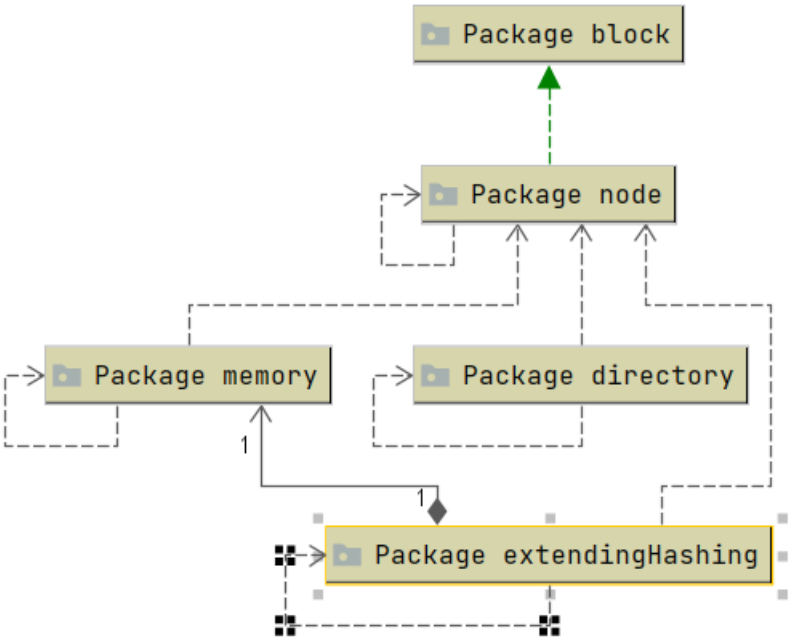
Na koniec si môžeme tieto komponenty poskladať do jedného systému a ukázať ako spolupracujú.



3.6 Core

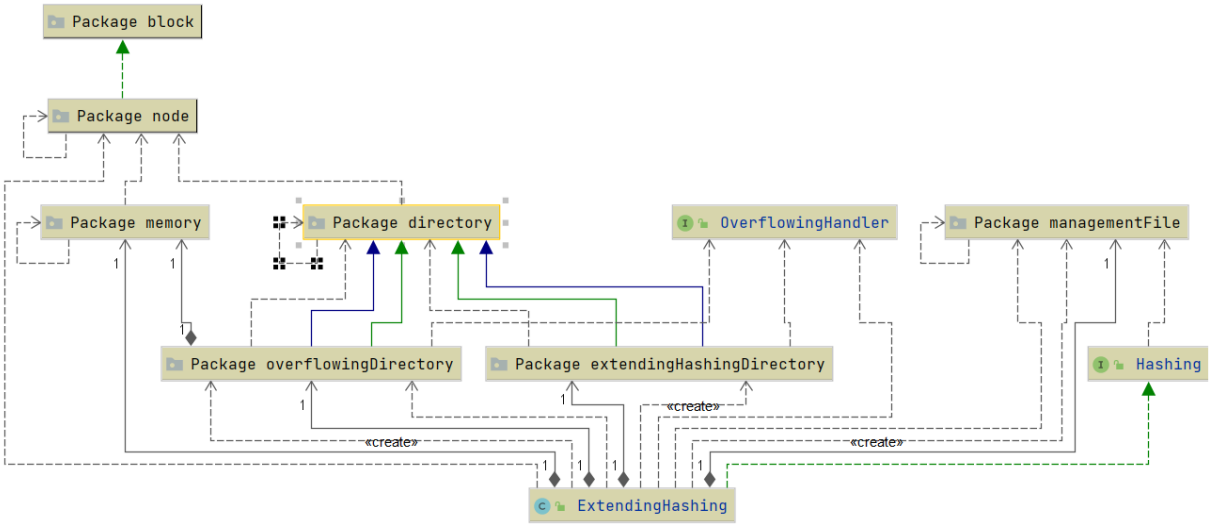
Táto časť by mala patriť do časti Model. Avšak v tomto projekt namiesto normálnej DB ako napríklad MongoDB alebo MySql sme využili štruktúru rozšíriteľné hešovanie s preťažujúcim súborom, aby sme dáta ukladali do súboru.

Teda tento balík obsahuje všetky prvky na to, aby užívateľ vedel vytvoriť svoje vlastné rozšíriteľné hešovanie alebo aby intuitívne a jednoducho použil svoje vlastné, respektíve vytvoril svoje vlastné iné hešovanie.

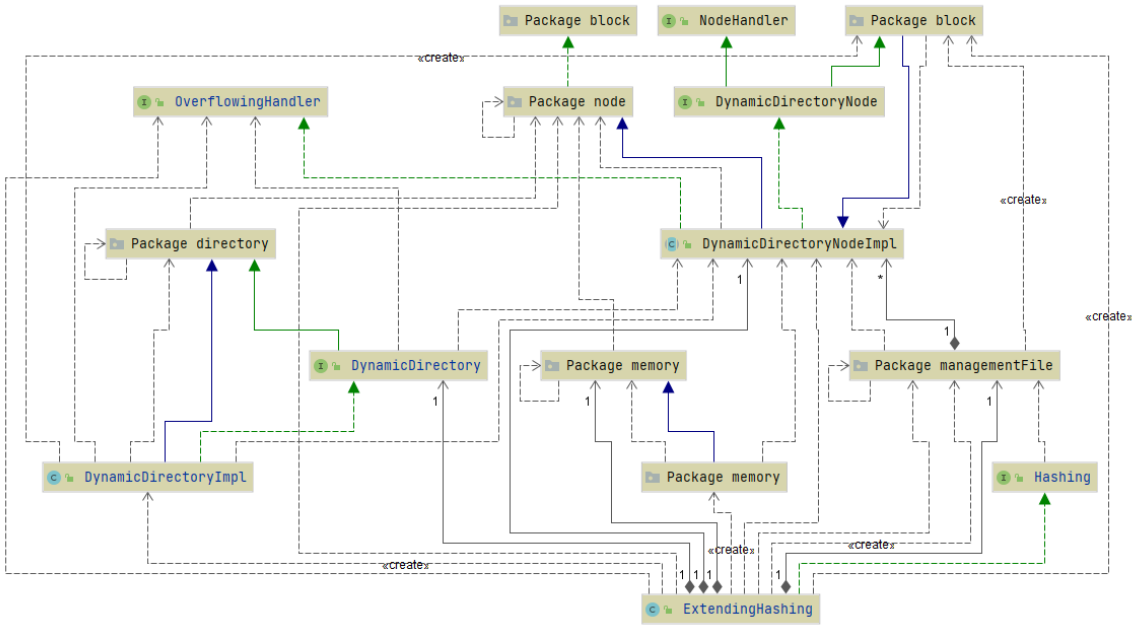


Tento obrázok nám jasne definuje čo musíme zmeniť , teda aké závislostí alebo čo implementovať, aby sme pohodlne nahradili rozšíriteľné hešovanie.

Na ďalšom obrázku vidíme čo obnáša rozšíriteľné hešovanie a ako sa zapracuje pomocou implementácií s ostatnými balíčkami

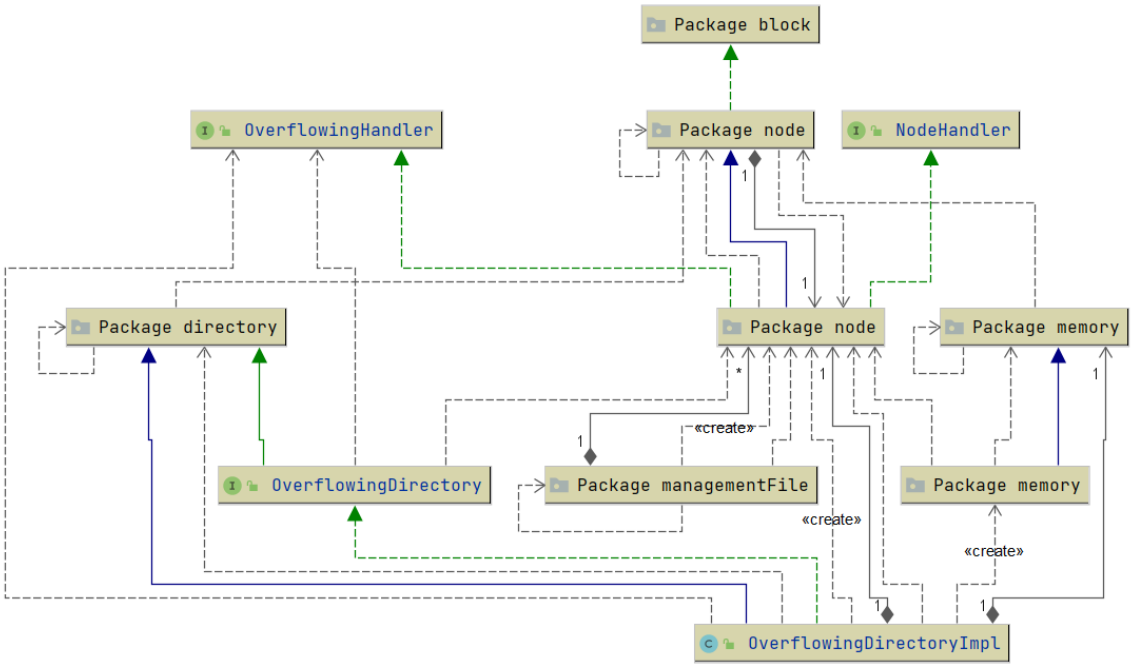


Nasledujúce obrázky budú predstavovať diagramy, pre zvlášť naše štruktúry



Ako prvý vidíme rozšíriteľné hešovanie. Od bežnej implementácie si je vhodné všimnúť interface OverflowingHandler, ktorý nám umožňuje prácu s preplňujúcim súborom. Za

zmienku stojí aj to, že pekne vidno na diagrame to, že v rozšíriteľnom hešovaní využívame dynamické pole.



Na poslednom UML diagrame vidíme to, že preplňujúci súbor dokáže vystupovať ako samostatná štruktúra, ale ako vidíme vyššie vie fungovať aj v kombinácií s inou štruktúrou.

4. Záver

Real Estate System je aplikácia, ktorá ukladá dáta do súboru a umožňuje v jednoduchom a intuitívnom rozhraní pre používateľa vykonávať jednoduché operácie pridaj, odober, vyhľadaj a uprav. Pre takýto typ ukladania dát som vybral štruktúru rozšíriteľné hešovanie s preplňujúcim súborom. Systém používa návrhový vzor MVP. Po štarte aplikácie užívateľ má na výber aj vygenerovanie si dát ale aj načítanie si konfigurácii zo súboru, ako aj ich uloženie. Následne si môže vybrať, aké operácie chce vykonávať.