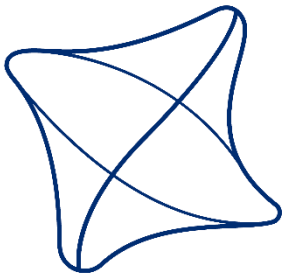
Žilinská univerzita v Žiline
Fakulta riadenia a informatiky



Semestrálna práca S2

Algoritmy a údajové štruktúry 2

Bc. Dominik Ježík

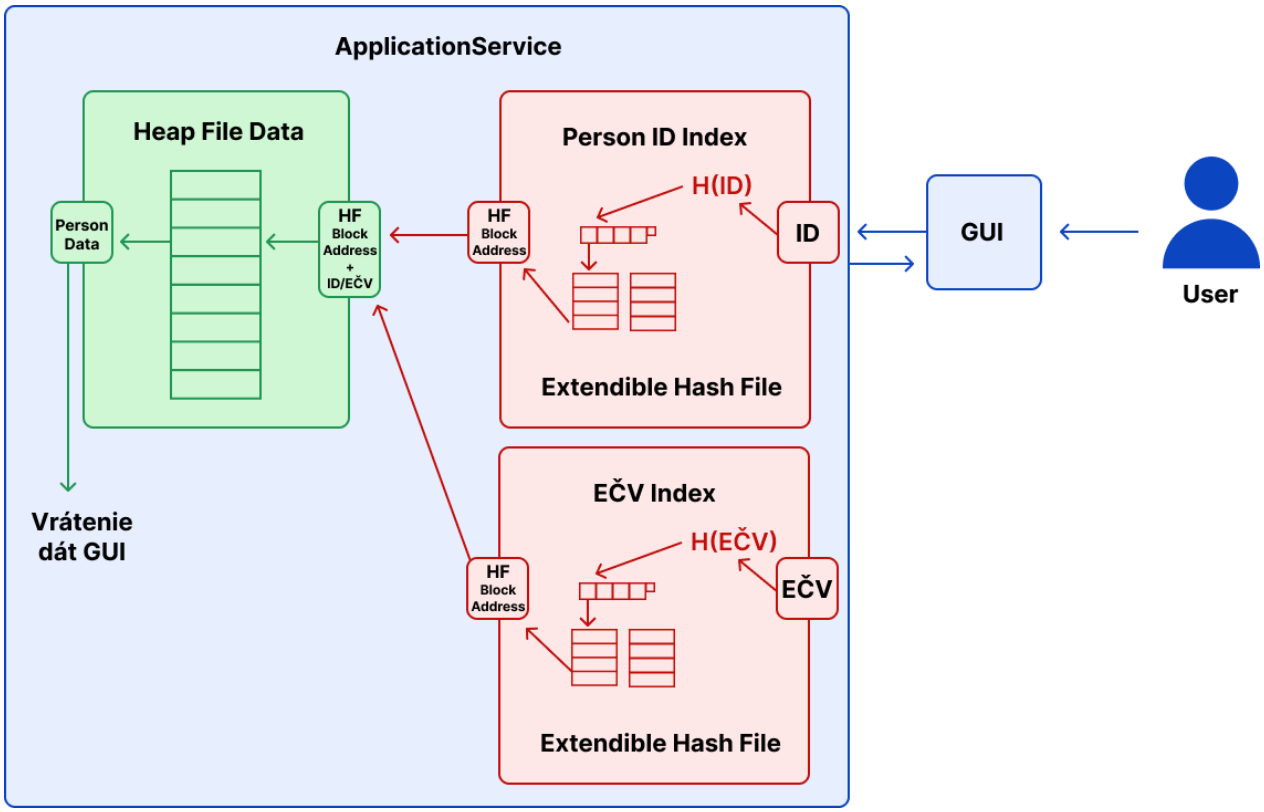
2024

Obsah

1 Použité údajové štruktúry

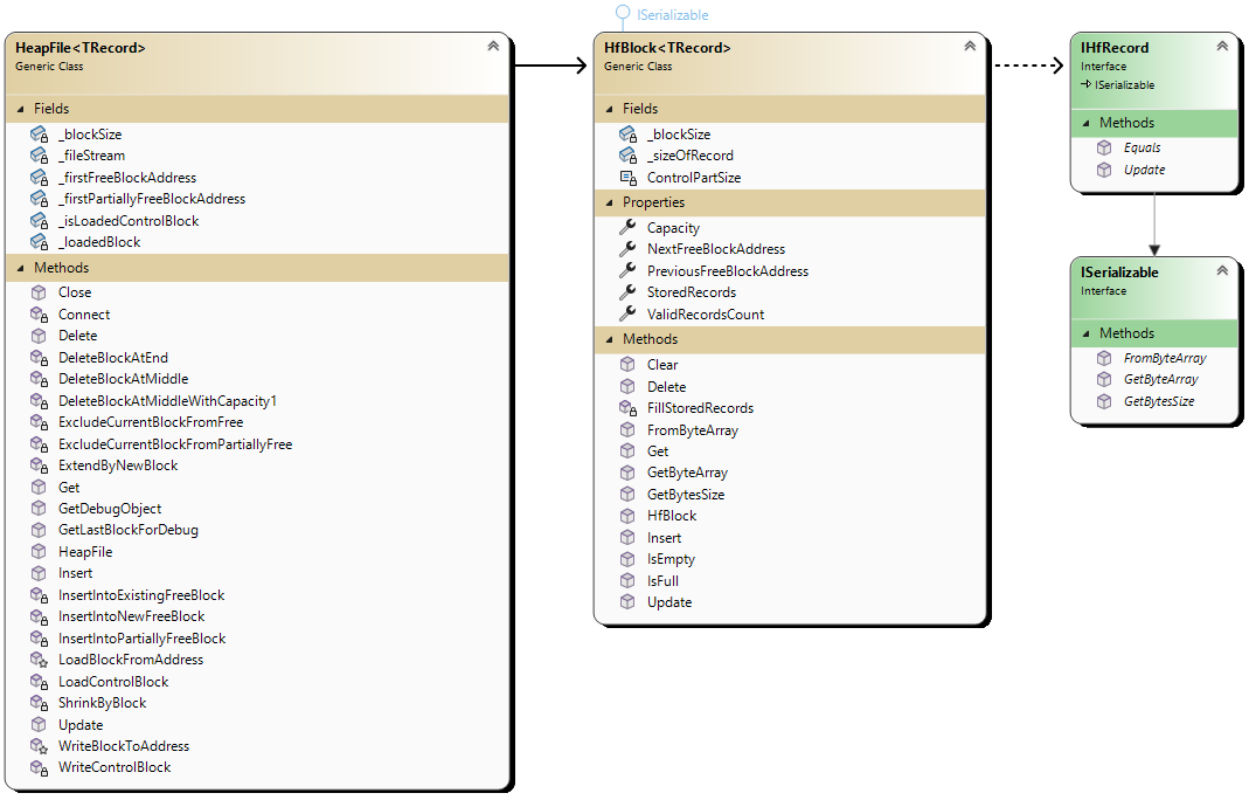
Jadro aplikácie tvoria 1 inštancia UŠ HeapFile a 2 inštancie UŠ ExtendibleHashFile. V štruktúre HeapFile sa nachádzajú dáta – informácie o zákazníkoch/autách a návštevách servisu. Po vložení záznamu do štruktúry programátor obdrží adresu bloku, kde sa záznam nachádza. Následne túto adresu bloku vložíme do oboch inštancií štruktúry ExtendibleHashFile, ktoré slúžia ako indexy pre rýchly prístup podľa kľúča. V jednej inštancií je kľúčom ID zákazníka a v druhej EČV, pričom obe sú unikátne.

Jednoduchý nákres zobrazuje základné väzby medzi použitými údajovými štruktúrami a používateľom cez GUI.



1.1 Triedy štruktúry HeapFile

Údajová štruktúra HeapFile pozostáva z hlavnej triedy *HeapFile*. Pri práci so súborom využíva ukladanie záznamov do blokov, ktorý je reprezentovaný triedou *HfBlock*. Obe triedy majú generický parameter *TRecord*, ktorý musí implementovať nami vytvorené rozhranie *IHfRecord*. Rozhranie má dve metódy – *Equals*, využívame pri porovnaní záznamu, keď potrebujeme získať, alebo odstrániť záznam zo súboru a Update, kde si programátor definuje, kód pre aktualizáciu požadovaných atribútov. *IHfRecord* navyše dedí z vytvoreného rozhrania *ISerializable*, ktoré definuje základné metódy, pre vlastnú serializáciu a deserializáciu záznamov na základe poľa bajtov.



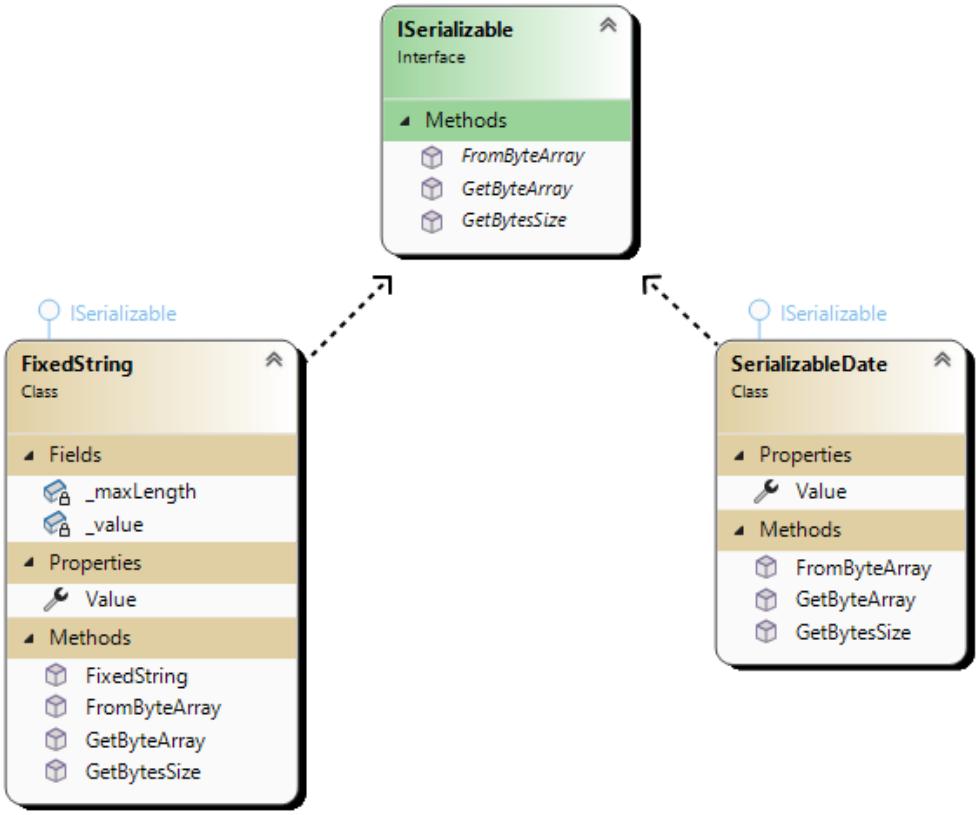
Dáta zapisujeme do jediného súboru, ktorý obsahuje riadiaci blok s adresou na prvý voľný blok v zreťazení voľných blokov a adresu na prvý čiastočne voľný blok v zreťazení čiastočne voľných blokov.

1.2 Pomocné triedy pre ukladanie dát

Pre ukladanie údajov o zákazníkovi a návšteve servisu používame pomocné triedy *FixedString* a *SerializableDate*, ktoré implementujú už spomínané rozhranie *ISerializable*.

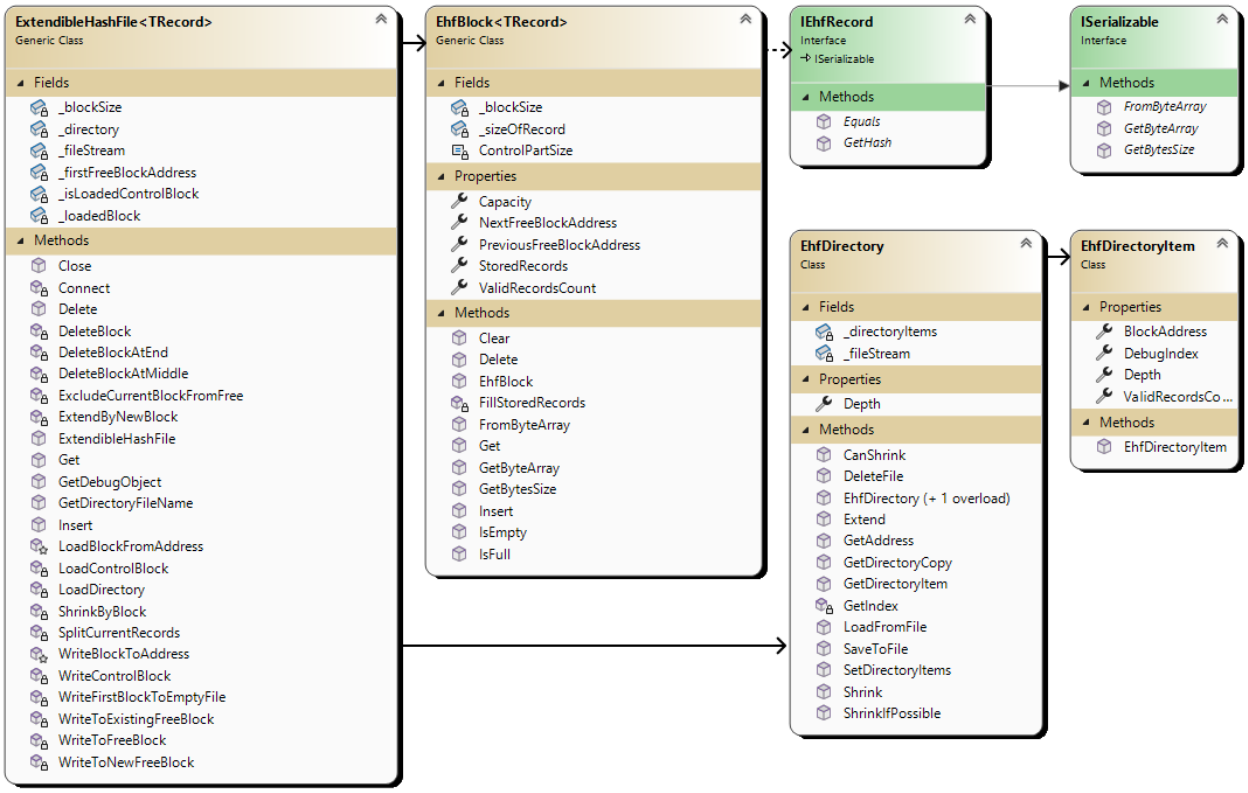
Keďže do súboru ukladáme reťazce s maximálnou dĺžkou, logika pre toto uloženie je zapuzdrená do triedy FixedString, kde je implementovaná serializácia/deserializácia. Používame kódovanie UTF-8, ktoré zakóduje znak do 1 až 4 bajtov. Z dôvodu tohto variabilného počtu je veľkosť stringu, ktorú zapisujeme do súboru vždy *4 * maximálnaDĺžka*. Pri serializácií na začiatku vždy zakódujeme do jedného bajtu počet bajtov, z koľkých pozostáva zadaný reťazec (zvyšné bajty sú vyplnené nulami).

Trieda SerializableDate obsahuje logiku pre ukladanie dátumov. Dátum uložíme s presnosťou na dni, ako číslo typu int. Číslo vyjadruje, koľko dní prešlo od dátumu 1.1.1970.



1.3 Triedy štruktúry ExtendibleHashFile

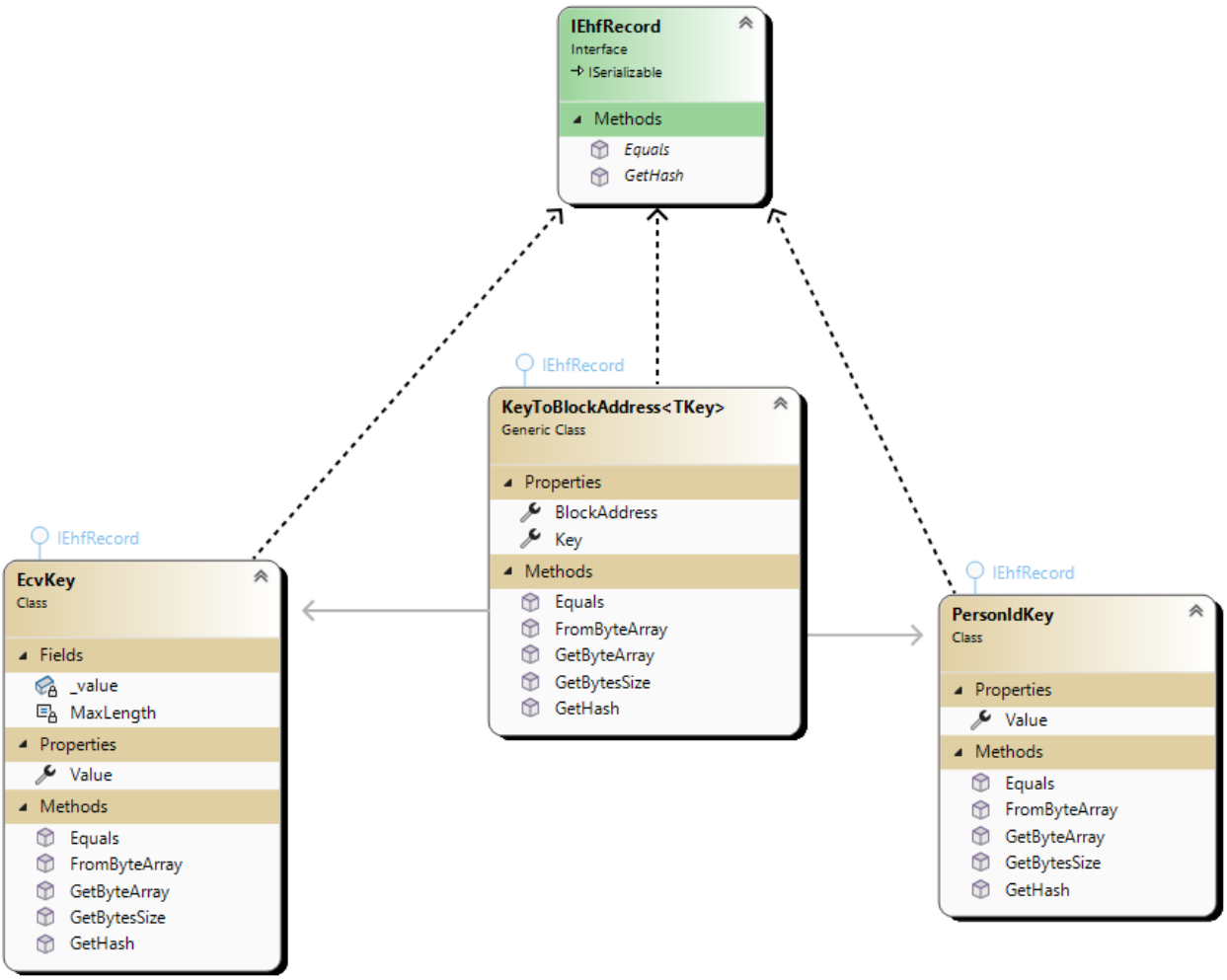
Rozšíriteľné hashovanie je implementované pomocou triedy *ExtendibleHashFile*. Podobne ako *HeapFile*, tiež ukladá záznamy do blokov. Využíva pri tom triedu *EhfBlock*. Vkladané záznamy musia implementovať triedu *IEhfRecord*, ktorá okrem metódy *Equals* pre nájdenie záznamov obsahuje aj *GetHash* metódu. Používame priame hashovanie, kde táto metóda vráti bitové pole. V rozšíriteľnom hashovaní sa využíva adresár, kde podľa indexu (zoberieme príslušný počet bitov z hashovacej funkcie) získame záznam adresára. Jeden záznam je tvorený atribútmi *BlockAddress* (adresa bloku v rozšíriteľnom hashovaní, *Depth* (hĺbka bloku) a *ValidRecordsCount* (počet platných záznamov v bloku). Adresár má tiež jeden atribút *Depth* (hĺba adresára).



V riadiacom bloku sa nachádza iba adresa na prvý voľný blok v zreťazení voľných blokov (nemá zmysel zreťazovať čiastočne voľné bloky). Bloky sú uložené v jednom súbore. Adresár, ktorý je počas existencie inštancie súboru v operačnej pamäti sa ukladá do samostatného súboru. Pri vytvorení inštancie sa adresár prečíta zo súboru a až pri ukončení (volanie *Close* metódy) sa zapíše do samostatného súboru.

1.4 Pomocné triedy používané s ExtendibleHashFile

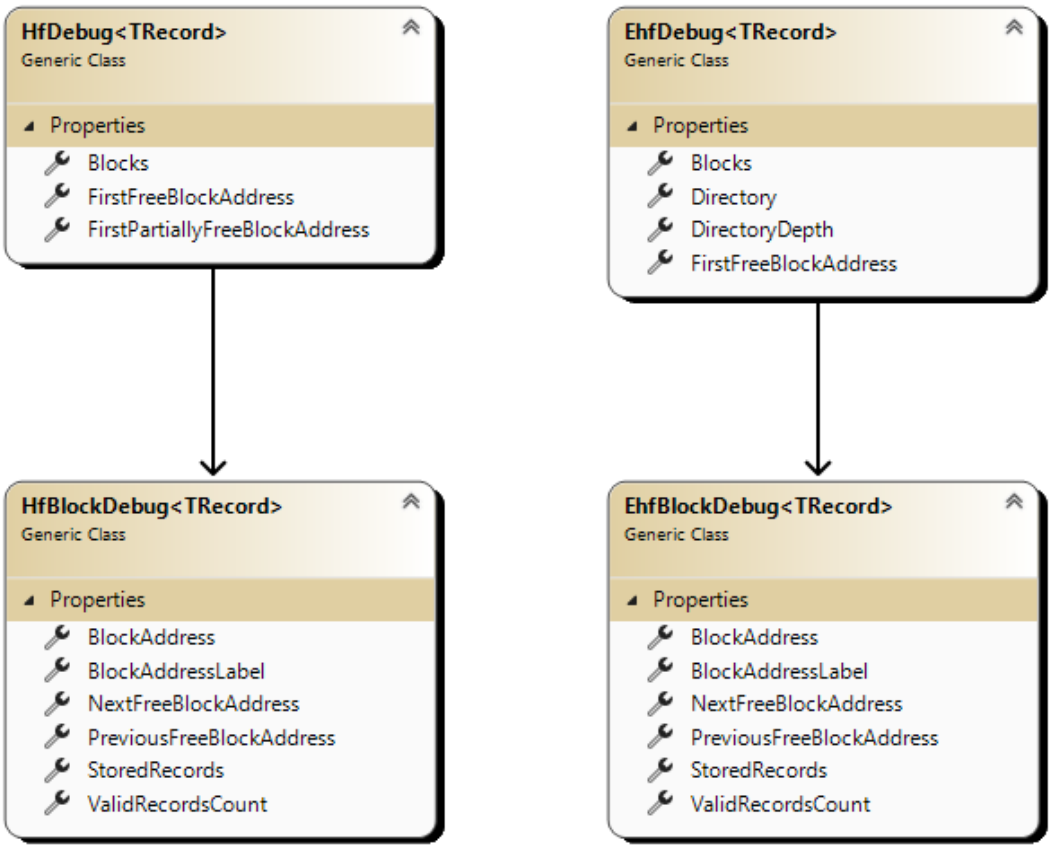
V aplikácií používame hashovacie súbory ako indexy, pre rýchly prístup podľa ID zákazníka alebo EČV. Vkladané prvky do týchto indexov vyžadujú aby implementovali nami vytvorené rozhranie, ktoré obsahuje metódu pre vrátenie hashu. Vytvorili sme preto jednu generickú obalovú triedu *KeyToBlockAddress*, ktorá obsahuje generický kľúč a adresu bloku (do HeapFilu). Potom sme pre oba typy kľúčov vytvorili obalové triedy (*PersonIdKey* a *EcvKey*), ktoré implementujú naše rozhranie pre prevod na pole bajtov (načítanie a zápis) a hlavne metódu pre výpočet hashu (priamym hashovaním vráti bitové pole).



Čiže ako záznam do hashovacích súborov idú typy KeyToBlockAddress<PersonalIdKey> a KeyToBlockAddress<EcvKey>.

1.5 Triedy pre ladenie údajových štruktúr

Keďže do súborov zapisujeme a čítame pole bajtov, nie je vždy počas implementácie úplne jasné aké dáta sa v súbore aktuálne nachádzajú. Za účelom ladenia údajových štruktúr, sú vytvorené triedy, ktorých inštancie vracajú obe spomínané štruktúry. Nachádzajú sa tu interné informácie vrátane riadiacich informácií, jednotlivých blokov a dát v blokoch. V prípade rozšíriteľného hashovania máme k dispozícii aj adresár z operačnej pamäte.



Tieto informácie je možné vypísať pomocou GUI okien integrovaných do výslednej aplikácie a tiež pomocou oddelených samostatných aplikácií.

2 Operácie štruktúry Heap File

2.1 Operácia Get

Operácia Get – získanie záznamu na základe adresy bloku vyžaduje na vstupe adresu bloku a inštanciu, v ktorej si programátor implementuje metódu *Equals* pre nájdenie záznamu v danom bloku. Metóda podľa adresy načíta daný blok do operačnej pamäte a vyhľadá záznam.

Počet prístupov do súboru je 1 (Čítanie)

2.2 Operácia Insert

V operácií Insert sa najskôr skontroluje či bol vôbec doposiaľ načítaný riadiaci blok (mohli sa od spustenia programu používať operácie Get, ktoré nepotrebujú riadiace informácie súboru). Ak nebol, načíta sa zo súboru. Toto načítanie riadiaceho bloku sa vykonáva vždy maximálne jeden krát počas existencie inštancie. Následne kontrolujeme či môžeme záznam vložiť do čiastočne voľného bloku alebo do voľného bloku alebo potrebujeme súbor rozšíriť o nový blok.

Prístup do súboru za účelom <u>jednorazového</u> načítania riadiaceho bloku je 1 (Čítanie)

2.2.1 Vloženie do čiastočne voľného bloku

Na základe atribútu z riadiaceho bloku sme zistili, že máme k dispozícií čiastočne voľný blok. Tento blok načítame do operačnej pamäte a záznam sem vložíme. Ak blok nie je plný, zapíšeme ho späť do súboru.

V tomto prípade je počet prístupov do súboru 1 + 1 (Čítanie + Zápis)

Pokiaľ je ale blok plný (nenachádza sa v ňom miesto pre ďalší záznam), je potrebné ho odobrať z reťazenia čiastočne voľných blokov. Poznačíme si adresu nasledujúceho čiastočne voľného bloku z reťazenia, aktuálnemu bloku nastavíme *next* adresu na -1 (čím ho odpojíme) a blok zapíšeme na disk. V operačnej pamäti aktualizujeme riadiacu časť, aby adresa čiastočne voľného bloku bola

nastavená na nasledujúci (túto adresu sme si poznačili). Ak je táto adresa rôzna od -1 (čiže blok existuje), načítame ho do pamäte, nastavíme *previous* na -1 (čím sme ho odpojili od prvého bloku) a zapíšeme späť do súboru. Ak by adresa bola rovná -1 ušetríme 1 pár čítania a zápisu (1 + 1).

V tomto prípade je maximálny počet prístupov do súboru 2 + 2 (Čítanie + Zápis)

2.2.2 Vloženie do voľného bloku

Ak nebol k dispozícii čiastočne voľný blok ale voľný blok áno, načítame ho a vložíme doň záznam. Nastavíme adresu nasledujúceho voľného bloku z atribútu *next* do riadiacej časti súboru (*_firstFreeBlockAddress*). Načítaný blok zaznačíme do riadiacej časti súboru ako prvý čiastočne voľný blok (*_firstPartiallyFreeBlockAddress*) (ak je zvolený blokovací faktor 1, potom nemusíme značiť čiastočne voľný blok, lebo čiastočne voľné bloky neexistujú). Blok zapíšeme do súboru. Ešte potrebujeme opraviť adresu aktuálne prvého voľného bloku, pretože *previous* adresa je stále nastavená na už neprázdny blok (sem sme vložili záznam), preto ho načítame, opravíme adresu na -1 a zapíšeme do súboru. Podobne ako v predchádzajúcej situácií sa dokáže ušetriť jeden pár prístupov do súboru ak je v zreťazení voľných blokov iba jeden blok čiže o 1 + 1 (Čítanie a Zápis).

V tomto prípade je maximálny počet prístupov do súboru 2 + 2 (Čítanie + Zápis)

2.2.3 Vloženie do nového voľného bloku

V poslednej možnej situácií súbor rozšírime o jeden nový blok na konci súboru. Vložíme doň záznam a blok zapíšeme. Ak by bol blok po vložení plný (čiže iba vtedy, keď je blokovací faktor 1), blok nemusíme značiť do čiastočne voľných blokov. V opačnom prípade nastavíme v riadiacej časti (*_firstPartiallyFreeBlockAddress*) na adresu tohto bloku.

Počet prístupov do súboru je 1 (Zápis)

2.3 Operácia delete

Na vstupe programátor zadá adresu bloku a inštanciu záznamu, ktorá sa použije na vyhľadanie požadovaného záznamu na zmazanie (cez *Equals*). Podobne ako v operácií insert sa najskôr skontroluje, či sme od momentu vytvorenia inštancie už načítali riadiaci blok súboru. Ak nie načítame ho.

Prístup do súboru za účelom jednorazového načítania riadiaceho bloku je 1 (Čítanie)

Následne podľa zadanej adresy načítame požadovaný blok a odstránime záznam. Teraz môžu vzniknúť nasledovné situácie.

2.3.1 Blok je prázdny a je na konci súboru

V prípade, že blokovací faktor je viac ako 1, blok vyradíme z čiastočne voľných blokov – je teda potrebné načítať, spojiť a zapísať predchodcu a nasledovníka tohto bloku. Súbor zmenšíme od konca o veľkosť bloku. Momentálne je ďalšia kontrola možných voľných blokov na konci súboru riešená postupným načítaním, skontrolovaním stavu bloku (počet záznamov). V prípade voľného bloku je odobratý zo zreťazenia voľných blokov (načítanie, spojenie a zápis predchodcu a nasledovníka) a zmenšením súboru o veľkosť bloku.

Maximálny počet prístupov do súboru pri blokovacom faktore > 1 je:

(3 + 2) + (N+1)(1 + 0) + N(2 + 2) (Čítanie + Zápis),

kde N je počet ďalších prázdnych blokov od konca súboru

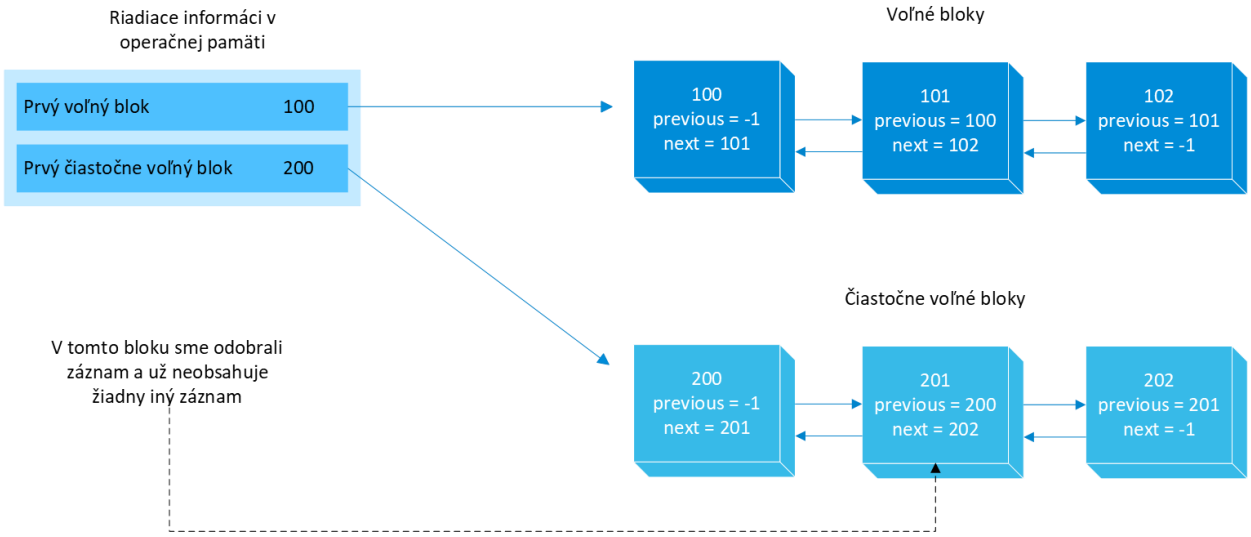
Poznámka vysvetlenie vzorca:
– prvá časť (3+2) = Načítanie bloku (1+0), zreťazenie predchodcu a nasledovníka (2 + 2)
– druhá časť (N+1)(1 + 0) = Načítanie posledného bloku a kontrola či je prázdny
– tretia časť N(2 + 2) = Zreťazenie predchodcu a nasledovníka (2 + 2)

Maximálny počet prístupov do súboru pri blokovacom faktore = 1 je:

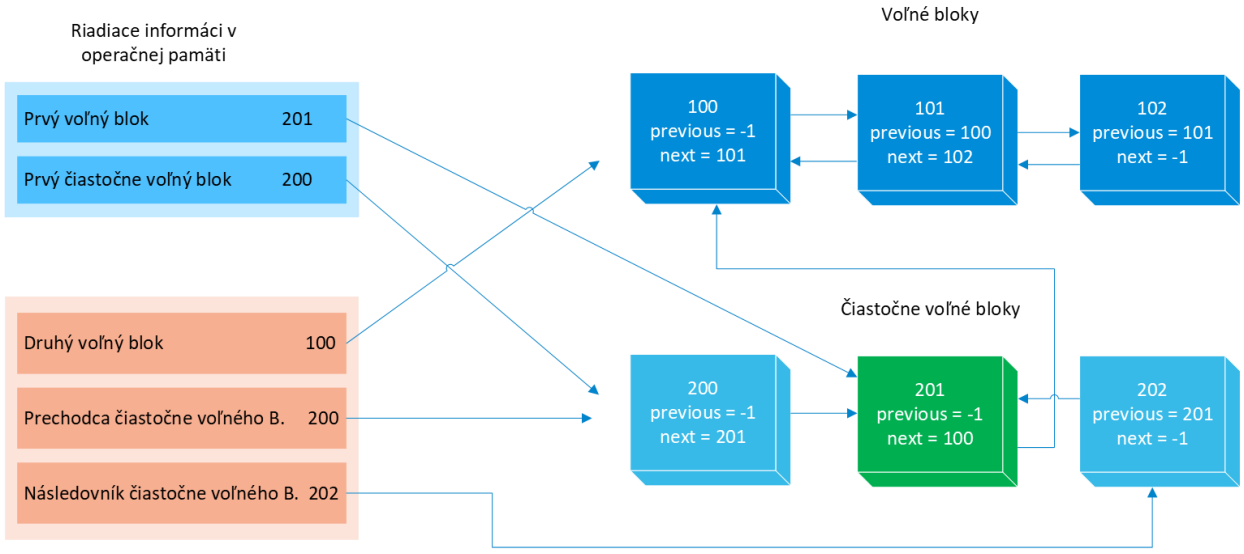
(predchádzajúci výraz - (2 + 2)) keďže blok nemusíme vyradiť z čiastočne voľných blokov

2.3.2 Blok je prázdny ale nie je na konci súboru

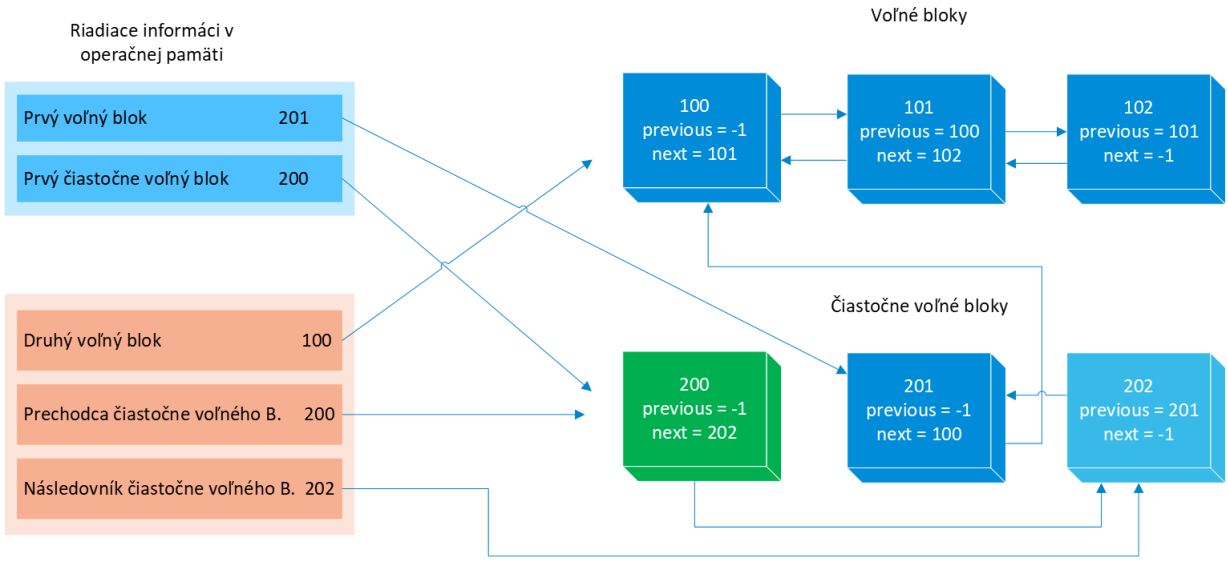
Najskôr predpokladáme, že blokovací faktor je viac ako 1, preto blok musel byť pred vymazaním záznamu v zreťazení čiastočne voľných blokov.

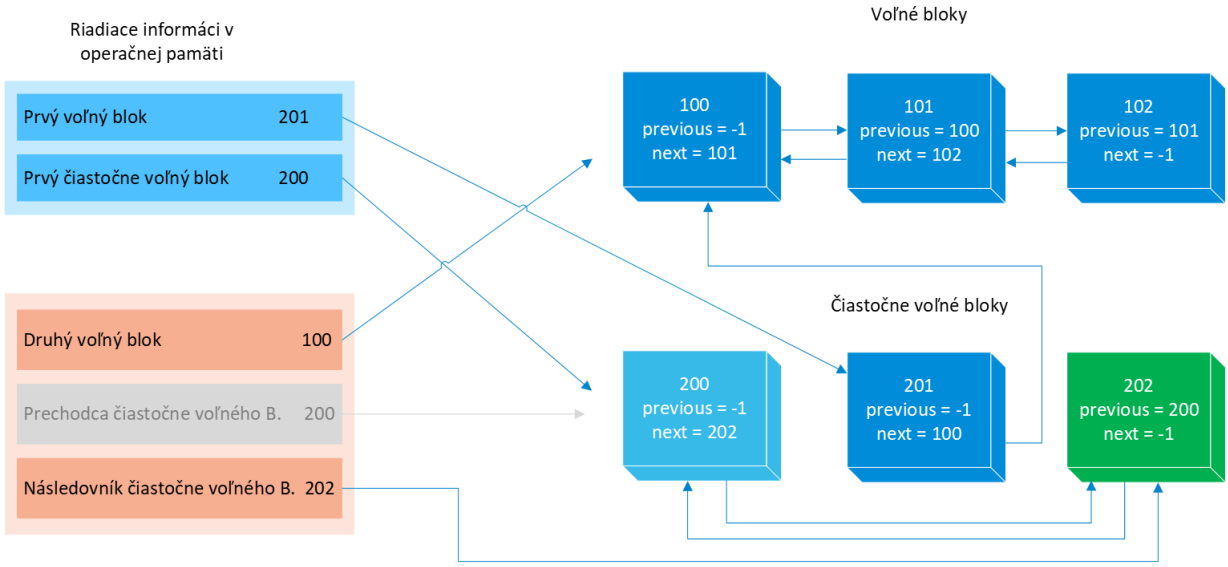


Poznačíme si preto adresy na predchodcu a nasledovníka z tohto zreťazenia. Poznačíme si aj adresu aktuálne prvého voľného bloku (stane sa druhým voľným blokom). Načítanému bloku (ktorý sa stane prvým voľným blokom) nastavíme *previous* na -1 a *next* na prvý voľný blok (ten sa stane druhým voľným blokom). V operačnej pamäti aktualizujeme adresu prvého voľného bloku. Blok zapíšeme do súboru. Ak bol náš načítaný blok prvým blokom v zreťazení čiastočne voľných blokov aktualizujeme túto adresu na druhý čiastočne voľný blok.

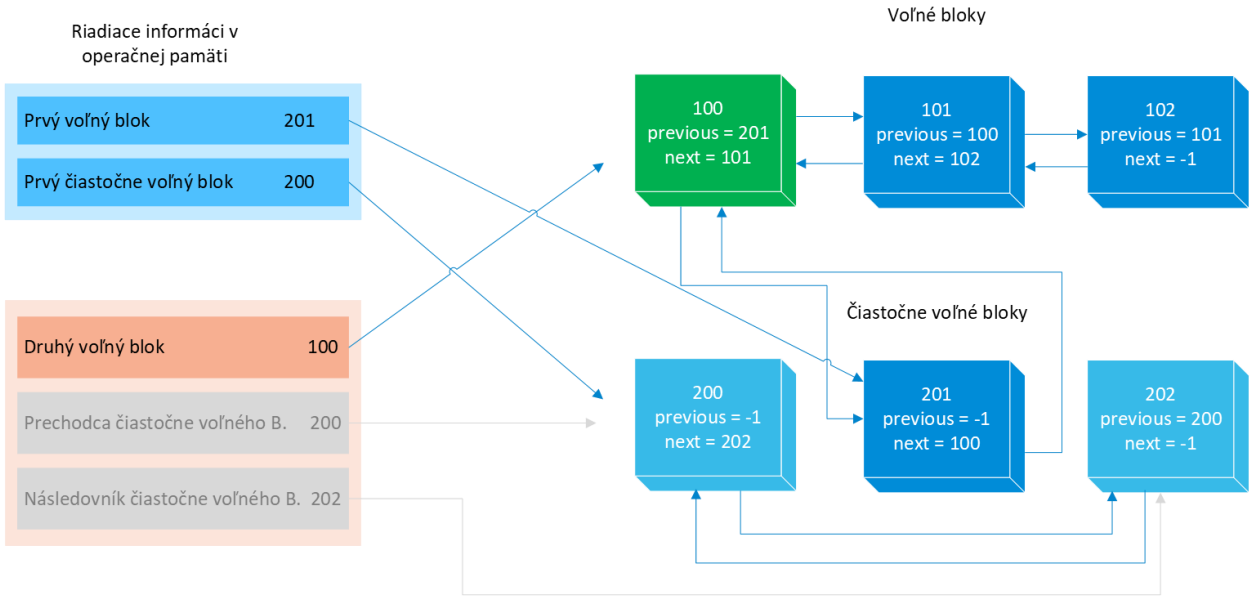


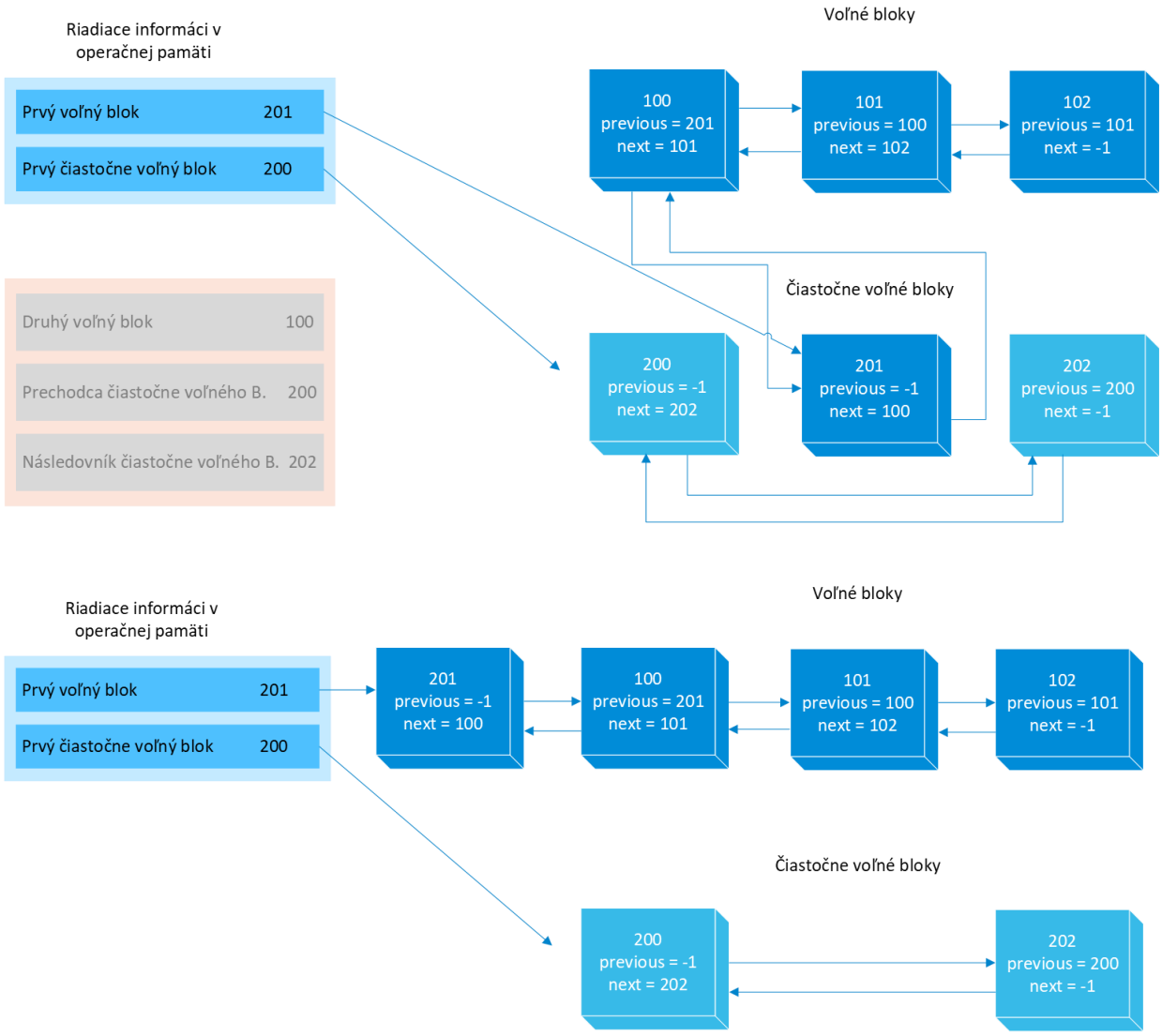
Následne je treba spojiť poznačených predchodcu a nasledovníka v čiastočne voľných blokoch. Oba načítame aktualizujeme príslušný *next* a *previous* a zapíšeme oba do súboru.





Na záver je potrebné opraviť *previous* adresu druhého voľného bloku na prvý voľný blok.





Maximálny počet prístupov do súboru 4 + 4 (Čítanie + Zápis)

V prípade, že blokovací faktor je 1, blok bol pred zmazaním záznamu plný (neexistujú čiastočne voľné bloky) a stačí tento blok iba zaradiť do voľných blokov.

Počet prístupov do súboru je v takomto prípade 2 + 2 (Čítanie + Zápis)

2.3.3 Blok nie je prázdny ale už bol v čiastočne voľných blokoch

Čiže pred zmazaním záznamu, bolo v bloku ešte miesto. V tomto prípade blok naďalej ostáva v čiastočne voľných blokoch.

Počet prístupov do súboru 1 + 1 (Čítanie + Zápis)

2.3.4 Blok nie je prázdny ale pred zmazaním nebol v čiastočne voľných blokoch

To znamená, že blok bol predtým plný a nenachádzal sa v žiadnom zreťazení. Blok musíme zaradiť do čiastočne voľných blokov. Poznačíme si adresu aktuálne prvého čiastočne voľného bloku a aktuálnemu bloku nastavíme *next* na túto adresu. Blok zapíšeme a načítame druhý čiastočne voľný blok. Opravíme mu adresu na *previous* a spätne zapíšeme do súboru.

Maximálny počet prístupov do súboru 2 + 2 (Čítanie + Zápis)

3 Operácie štruktúry Extendible Hash File

Pri vytvorení inštancie súboru rozšíriteľného hashovania sa načíta adresár zo samostatného súboru – prečíta sa celý obsah ako pole bajtov. Z poľa sa zistí hĺbka adresára a prečítajú sa jednotlivé adresy, hĺbky blokov a počet platných evidovaných záznamov, ktoré sú v adresári uložené (je ich 2^hĺbkaAdresára).

3.1 Operácia Get

Operácia Get – získanie záznamu na základe kľúča. Metóda získa hash podľa vstupného objektu, v ktorom programátor implementuje metódu *GetHash*. Metóda vráti bitové pole, ktoré reprezentuje hash. Hash odošleme do adresára, ktorý si z neho zoberie prvých depth-bitov zľava, kde depth je aktuálna hĺbka adresára. Túto časť bitov prevedieme na desiatkové číslo slúžiace ako index do adresára. Na tomto indexe sa nachádza adresa bloku s hľadaným záznamom. Blok načítame do operačnej pamäte, vyhľadáme záznam a ak sa našiel, záznam vrátime.

Počet prístupov do súboru je 1 (Čítanie)

3.2 Operácia Insert

V operácií Insert sa najskôr skontroluje či bol vôbec doposiaľ načítaný riadiaci blok, kde sa nachádza adresa na prvý voľný blok v zreťazení (mohli sa od spustenia programu používať operácie Get, ktoré nepotrebujú riadiace informácie súboru). Ak nebol, načíta sa zo súboru. Toto načítanie riadiaceho bloku sa vykonáva vždy maximálne jeden krát počas existencie inštancie.

Prístup do súboru za účelom jednorazového načítania riadiaceho bloku je 1 (Čítanie)

Pokiaľ sa v súbore nenachádza žiadny blok, zapíšeme prvý blok a adresu tohto bloku poznačíme do prvej položky adresáru (teraz je hĺbka adresára 0, keďže sme ešte nepoužili žiadny bit z výsledku hashovacej funkcie).

Počet prístupov do súboru je 0 + 1 (Čítanie + Zápis)

Následne v cykle sa snažíme záznam vložiť do bloku. Vypočítame hash, získame adresu záznamu z adresára. Načítame blok a pokiaľ blok nie je plný záznam doň vložíme a blok zapíšeme.

Počet prístupov do súboru je v tomto prípade 1 + 1 (Čítanie + Zápis)

Môže nastať situácia, že v adresári je poznačená adresa -1, čo znamená že pri niektorom z minulých vkladaní bol blok plný a záznamy sa nepodarilo prerozdeliť medzi pôvodný blok a nový blok. V tom prípade sme nezapísali prázdny blok na disk ale uvoľnili ho iba z operačnej pamäte (nebol žiadny prístup do súboru spojený s prázdnym blokom). Preto na miesto -1, vytvoríme nový blok, adresu poznačíme do adresára a zapíšeme ho na disk.

V takejto situácií je počet prístupov do súboru 0 + 1 (Čítanie + Zápis)

Pokiaľ je ale blok plný vzniknú ďalšie prístupy do súboru. Najskôr sa skontroluje, či je potrebné zdvojnásobiť adresár (keď sa hĺba adresáru rovná hĺbke plného bloku). Vytvoríme v operačnej pamäti nový blok a záznamy z plného bloku si odložíme. Odložíme aj hĺbku bloku a zvýšime ju o 1. Plný blok vyčistíme a pre každý záznam vypočítame hash (prehashovanie) a podľa n-tého bitu (kde n je zvýšená odložená hĺbka) záznamy prerozdelíme medzi tieto bloky. Potom sa pokúsime vložiť aj nový záznam, pokiaľ blok nie je plný (znova podľa n-tého bitu). Aktualizujeme hĺbku blokov a prvý blok zapíšeme. Následne si vyžiadame nový blok zo súboru (buď sa rozšíri od konca alebo sa použije voľný blok zo zreťazenia), zapíšeme ho a získame adresu tohto bloku. Túto adresu ešte treba vložiť do adresára. Adresáru odošleme hash záznamu z tohto bloku, hĺbku bloku (depth) a adresu pre nastavenie. Adresár si z hashu zoberie prvých depth-bitov a vytvorí dva nové hashe (rozsahy v adresári) – ponechá depth-bitov nezmenené a ostatné bity nastaví na samé 0 respektíve na samé 1. Potom si z hashov zoberie n-bitov (kde n je hĺba adresára), prevedie ich na desiatkové čísla a tým získa rozsah indexov. Pre tento rozsah (sú to indexy do adresára) nastaví adresu bloku. Ak sa bloky nedokázali rozdeliť cyklus sa opakuje.

Poznámka: ak sa záznamy neprerozdelili, plný blok zbytočne nezapisujeme do súboru ale udržujeme ho v operačnej pamäti. V ďalšej iterácií, tento blok budeme znova potrebovať, keď sa záznamy pokúsime prerozdeliť podľa ďalšieho bitu. Poznačíme si preto do pomocnej premennej

fullBlockOfRecordsIsAlreadyLoaded na hodnotu true a tým pádom v ďalšej iterácii už blok zbytočne nemusíme načítať zo súboru.

Ak bol blok plný, počet prístupov do súboru sa štandardne pohybuje od:

(1 + 2) (Čítanie + Zápis) – *Ak nemáme existujúce voľné bloky ale iba rozširujeme súbor, čiže (1 + 0) prečítame plný blok. Keď sa v niektorej z iterácií podarí záznamy prerozdeliť. zapíšeme pôvodný blok (0 + 1) a zapíšeme aj nový blok (0 + 1), ktorý vznikol rozšírení od konca súboru.*

do

(3 + 3) (Čítanie + Zápis) – A*k použijeme existujúci voľný blok. Čiže (1 + 0) prečítame plný blok. Potom keď sa v niektorej z iterácií podarí záznamy prerozdeliť vyberieme zo zreťazenia voľných blokov prvý voľný a prečítame ho (1 + 0), odoberieme ho zo zreťazenia, tým že načítame druhý voľný a zrušíme mu napojenie na prvý (1 + 1). Zapíšeme pôvodný blok (0 + 1) a aj nový blok (0 + 1) – čiže záznamy sú prerozdelené medzi tieto dva bloky.*

3.3 Operácia Delete

Rovnako ako pri operácií Insert sa najskôr skontroluje či bol vôbec doposiaľ načítaný riadiaci blok. Ak nebol, načíta sa zo súboru. Toto načítanie riadiaceho bloku sa vykonáva vždy maximálne jeden krát počas existencie inštancie.

Prístup do súboru za účelom jednorazového načítania riadiaceho bloku je 1 (Čítanie)

Programátor vloží ako vstupný parameter metódy znova inštanciu záznamu pre zmazanie, kde je metóda *GetHash* a *Equals*. Následne získame hash a podľa neho položku z adresára. V položke máme uloženú adresu bloku – tento blok načítame a záznam z bloku odstránime. Aktualizujeme adresár, keďže sa o jedna znížil počet validných záznamov bloku. Následne vstúpime do cyklu za účelom zlúčiť čo najviac susedných blokov pokiaľ je to možné. Tiež sa snažíme čo najviac zmenšiť adresár o polovice pokiaľ je to možné. Na začiatku si vždy z adresára vytiahneme aktuálnu položku

na základe hashu. V prípade že je hĺbka adresára 0, nie je možné adresár ďalej zmenšovať. Preto buď blok zapíšeme späť na disk, alebo ak už neexistuje žiadna položka odstránime tento blok.

V takejto situácií je počet prístupov do súboru (1 + 1) (Čítanie + Zápis)

Pokiaľ je ale hĺbka bloku iná ako 0, je potrebné identifikovať či vieme zlúčiť načítaný blok so susedným. Preto si skopírujeme už vypočítaný hash a v tejto kópií zmeníme bit (ktorý je zľava na pozícií hĺbky bloku) na opačný. Na základe tohto zmeneného hashu získame susednú položku z adresára. Pokiaľ je hĺbka aktuálneho bloku a hĺbka susedného rôzna, nie je bloky možné zlúčiť a rovnako ako v predchádzajúcej situácií buď zapíšeme blok alebo ho odstránime ak je prázdny (v tomto prípade sa aj aktualizuje adresa bloku v adresári na -1).

V takejto situácií je počet prístupov do súboru tiež (1 + 1) (Čítanie + Zápis)

Ak je hĺbka susedného bloku rovnaká rozlišujeme 3 rôzne situácie.

Ak je adresa susedného bloku -1, blok vieme zlúčiť. Tento prípad nikdy nenastane v prvej iterácií cyklu ale môže nastať v ďalších (prípad keď sa nám pri operácií insert neprerozdelili záznamy a nastavili sme adresu -1 miesto zápisu prázdneho bloku). Bloky zlúčime tým, že aktualizujeme adresár s hĺbkou bloku menšou o jedna. Tiež skontrolujeme, či je možné adresár zmenšiť o polovicu – toto sa skontroluje cyklicky, keďže by mohlo nastať viacnásobné zmenšenie adresára o polovicu. Hlavný cyklus sa zopakuje.

V prípade, že adresa susedného bloku nie je -1, skontrolujeme či sa záznamy z aktuálneho a susedné bloku zmestia do jedného bloku. Na túto kontrolu nenačítavame zbytočne susedný blok pre zistenie platného počtu záznamu, ale túto informáciu máme uloženú v adresári. Ak sa záznamy zmestia do jedného bloku, odložíme si aktuálne načítaný blok a načítame si aj susedný blok. Odložíme si záznamy zo susedného bloku a tento susedný blok odstránime (buď zaradíme do zreťazania voľných blokov alebo ak je na konci tak zmenšíme súbor a prípadne cyklicky ďalšie voľné bloky na konci súboru). Záznamy pridáme k záznamov do pôvodného bloku, aktualizujeme adresár (počet záznamov a hĺbka bloku je o jedna menšia). Rovnako ako v minulom prípade skontrolujeme či je možné adresár zmenšiť o polovicu cyklicky. Hlavný cyklus sa zopakuje.

Ak sa bloky nezmestia do jedného bloku, cyklus končí – blok zapíšeme do súboru. Rovnako ako pri operácií insert, načítaný blok načítame iba jeden krát a zapíšeme ho až pri ukončení cyklu.

Počet prístupov do súboru sa pohybuje

od (1 + 1) (Čítanie + Zápis) – *ak by sa v cykle opakovala situácia, kde má susedný blok adresu -1.*

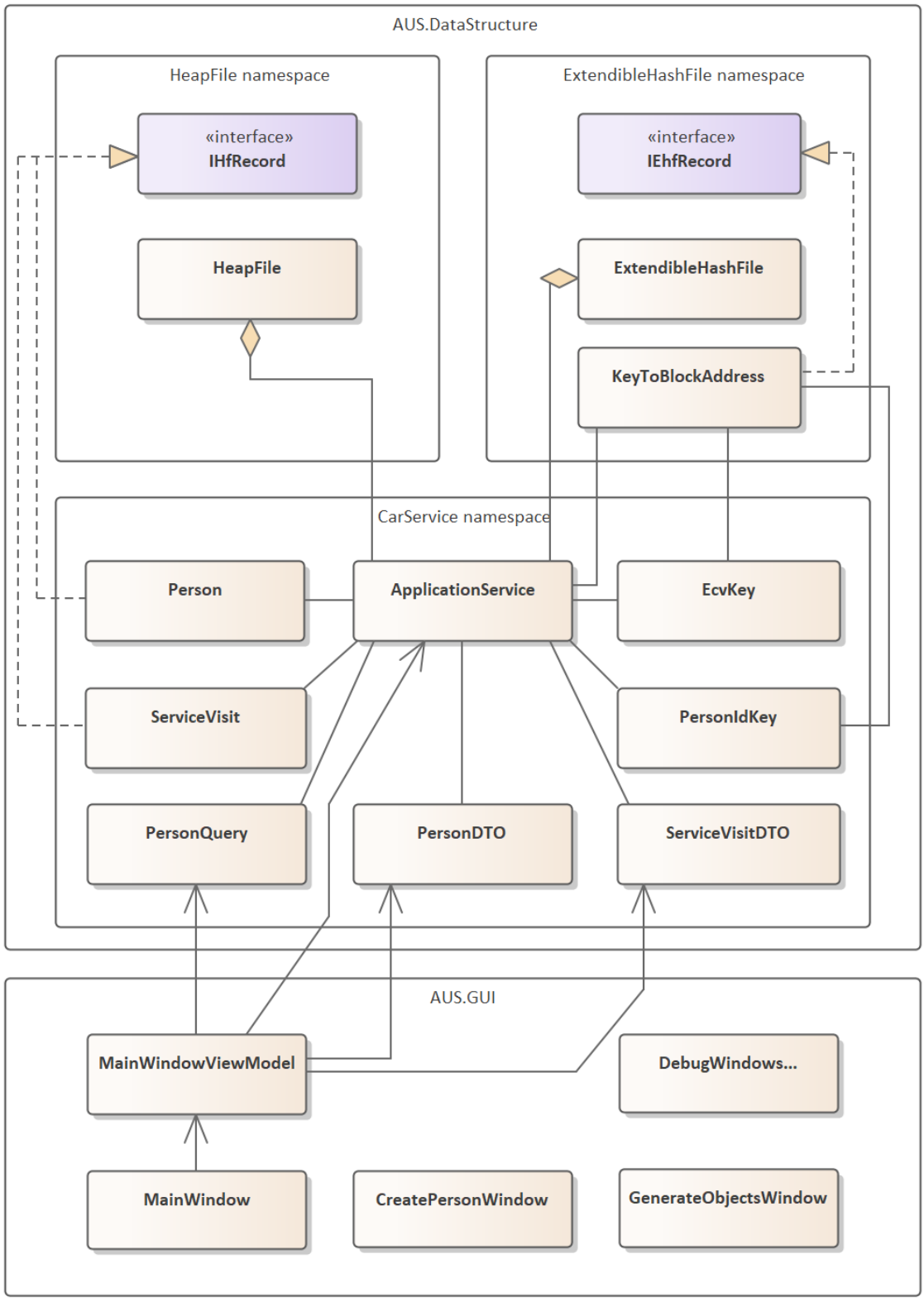
respektíve

od (1 + 1) + N(2 + 2) (Čítanie + Zápis) – *ak by sa v cykle opakovala situácia N krát, kde vieme zlúčiť susedný blok s aktuálnym, pričom ale platí, že susedný blok, ktorý odstraňujeme je v strede súboru – nie je na konci, čiže zaradzujeme do zreťazenia voľných blokov. (1 + 1) je prečítanie a zápis pôvodného bloku a potom N krát susedný blok zaradzujeme do zreťazenia voľných blokov (2 + 2).*

respektíve

od (1 + 1) + N((1 + 0) + M) – *ak by sa v cykle opakovala situácia N krát, kde vieme zlúčiť susedný blok s aktuálnym, pričom ale platí, že susedný blok, ktorý odstraňujeme je na konci súboru – je potrebné súbor zmenšiť o veľkosť bloku a prípadne cyklicky odstraňovať voľné bloky od konca súboru. (1 + 1) je prečítanie a zápis pôvodného bloku a potom N krát susedný blok prečítame (pre získanie záznamov) (1 + 0). Následne je blok potrebné odstrániť od konca súboru a cyklicky aj ďalšie voľné bloky na konci súboru. Táto zložitosť je označená ako **M**, ktoré vyjadruje presne rovnakú zložitosť ako v operácií delete v heap file pri blokovacom faktore 1 [kapitola 2.3.1].*

4 Architektúra aplikácie



Architektúra aplikácie oddeľuje jadro (databázu) od používateľského rozhrania (okná implementované vo frameworku Avalonia). V knižnici AUS.DataStructures, obsahujeme menné priestory pre HeapFile, ExtendibleHashFile a CarService. Hlavnou triedou v jadre aplikácie je trieda ApplicationService, ktorú používa GUI. Táto trieda obsahuje inštanciu heapfilu a dve inštancie rozšíriteľného hashovania ako index. Entity Person a ServiceVisit sa mapujú na svoje DTO triedy a GUI pracuje iba s týmito triedami – z jadra nie je možné priamo k entitám uloženým v databáze.

5 Funkcie aplikácie

Podľa zadania semestrálnej práce je v aplikácií možné realizovať týchto 6 základných funkcií pričom sa používajú operácie nad údajovými štruktúrami:

1. Vyhľadanie všetkých evidovaných údajov o vozidle (vozidlo sa vyhľadá podľa id zákazníka alebo podľa EČV – vyberie si užívateľ).
   Najskôr sa vyhľadá adresa bloku do HeapFilu, pričom sa použije operácia get nad zvoleným indexom (hashovacím súborom). Po získaní adresy sa vykoná operácia get nad HeapFilom.

2. Pridanie vozidla – na základe zadaných údajov zaradí vozidlo do evidencie (pozor na ošetrenie unikátnosti niektorých údajov).
   Najskôr sa pomocou oboch indexov overí, že vkladáme unikátneho zákazníka (ID aj EČV musia byť unikátne). Následne vložíme zákazníka do HeapFilu, ktorý vráti adresu bloku. Tú potom vložíme do oboch indexov.

3. Pridanie návštevy servisu - na základe zadaných údajov pridá návštevu servisu do evidencie (vozidlo sa vyhľadá podľa id zákazníka alebo podľa EČV – vyberie si užívateľ).
   Funkcia je súčasťou nasledujúcej funkcie Zmeny.

4. Zmena – umožní zmeniť akékoľvek evidované údaje (vozidlo sa vyhľadá podľa id zákazníka alebo podľa EČV – vyberie si užívateľ), pozor na zmenu kľúčových a nekľúčových atribútov.
   Najskôr skontrolujeme či došlo k zmene kľúčových atribútov a overíme, že nové kľúčové atribúty nenarušujú vlastnosť unikátnosti. Potom podľa pôvodného ID vyhľadáme v index adresu bloku do HeapFilu. Po získaní adresy vykonáme operáciu Update nad HeapFilom, ktorý načíta blok, vyhľadá záznam a spustí definovanú operáciu Update nad osobou (rozhranie IHfRecord definuje, že záznamy musia mať túto metódu). V Update metóde sa podľa vstupného parametra – aktualizovaná inštancia osoby, aktualizujú atribúty vrátane návštev servisu. Po update sa blok zapíše späť do súboru.

5. Zmazanie návštevy servisu – umožní zmazať akékoľvek evidované údaje (vozidlo sa vyhľadá podľa id zákazníka alebo podľa EČV – vyberie si užívateľ).
   Funkcia je súčasťou predchádzajúcej funkcie Zmeny.

6. Zmazanie vozidla – umožní zmazať všetky údaje o vozidle (vozidlo sa vyhľadá podľa id zákazníka alebo podľa EČV – vyberie si užívateľ).
   Vyhľadáme adresu bloku do HeapFilu cez index ID. Záznam z HeapFilu odstránime a následne odstránime podľa ID a podľa EČV záznamy z indexov.