ŽILINSKÁ UNIVERZITA V ŽILINE
FAKULTA RIADENIA A INFORMATIKY

ALGORITMY A ÚDAJOVÉ ŠTRUKTÚRY 2

Semestrálna práca 2

Jozef Chmelar
5ZZS12
2017/2018

ZADANIE

Pre pripravovaný projekt elektronickej zdravotnej karty malého štátu navrhnite a implementujte demonštračný systém. Tento systém bude umožňovať použitie základných administratívnych úkonov, ktoré budú neskôr poskytnuté vo finálnej verzii informačného systému zdravotnej dokumentácie obyvateľstva. Celá databáza bude centralizovaná (lokálne počítače tam budú realizovať svoje požiadavky) a bude sa nachádzať na dátovom úložisku (pevnom disku). V demonštračnej verzii sa budú evidovať iba pacienti a ich hospitalizácie. V demonštračnej verzii nie je potrebné zabezpečiť vzdialený prístup k centrálnej databáze, ale je potrebné umožniť výpis všetkých evidovaných údajov, tak aby bolo možné skontrolovať funkčnosť programu.

Pre každého pacienta evidujte nasledovné údaje:

- krstné meno (reťazec – max. 25 znakov)
- priezvisko (reťazec – max. 25 znakov)
- číslo preukazu (celé číslo)
- dátum narodenia
- záznamy o všetkých jeho hospitalizáciách (maximálne ich môže byť 100)

Pre každú hospitalizáciu pacienta evidujte:

- dátum začiatku hospitalizácie
- dátum konca hospitalizácie
- diagnózu s ktorou bol prijatý (reťazec – max. 40 znakov)

Informačný systém musí umožňovať tieto základné operácie (operácie sú zoradené podľa početnosti ich využívania):

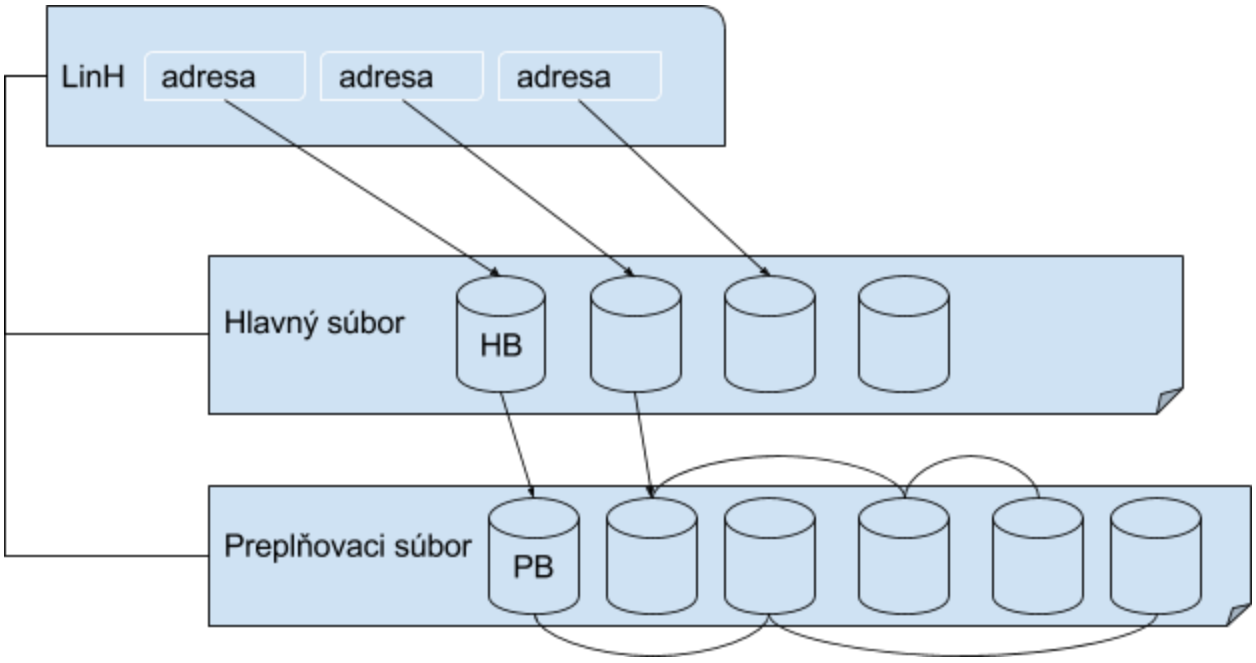
1. vyhľadanie záznamov pacienta (identifikovaný svojím číslom preukazu). Po nájdení pacienta je potrebné zobraziť všetky evidované údaje.
2. vykonanie záznamu o začiatku hospitalizácie pacienta (identifikovaný svojím číslom preukazu),
3. vykonanie záznamu o ukončení aktuálnej hospitalizácie pacienta (identifikovaný svojím číslom preukazu),
4. výpis pacientov, ktorých čísla preukazov sú so zadaného intervalu (od, do).
5. editácia údajov pacienta (identifikovaný svojím číslom preukazu), bez editácie údajovo hospitalizáciách, s možnosťou editácie čísla preukazu,
6. pridanie pacienta,
7. odstránenie pacienta (identifikovaný svojím číslom preukazu) spolu so všetkými jeho záznamami.

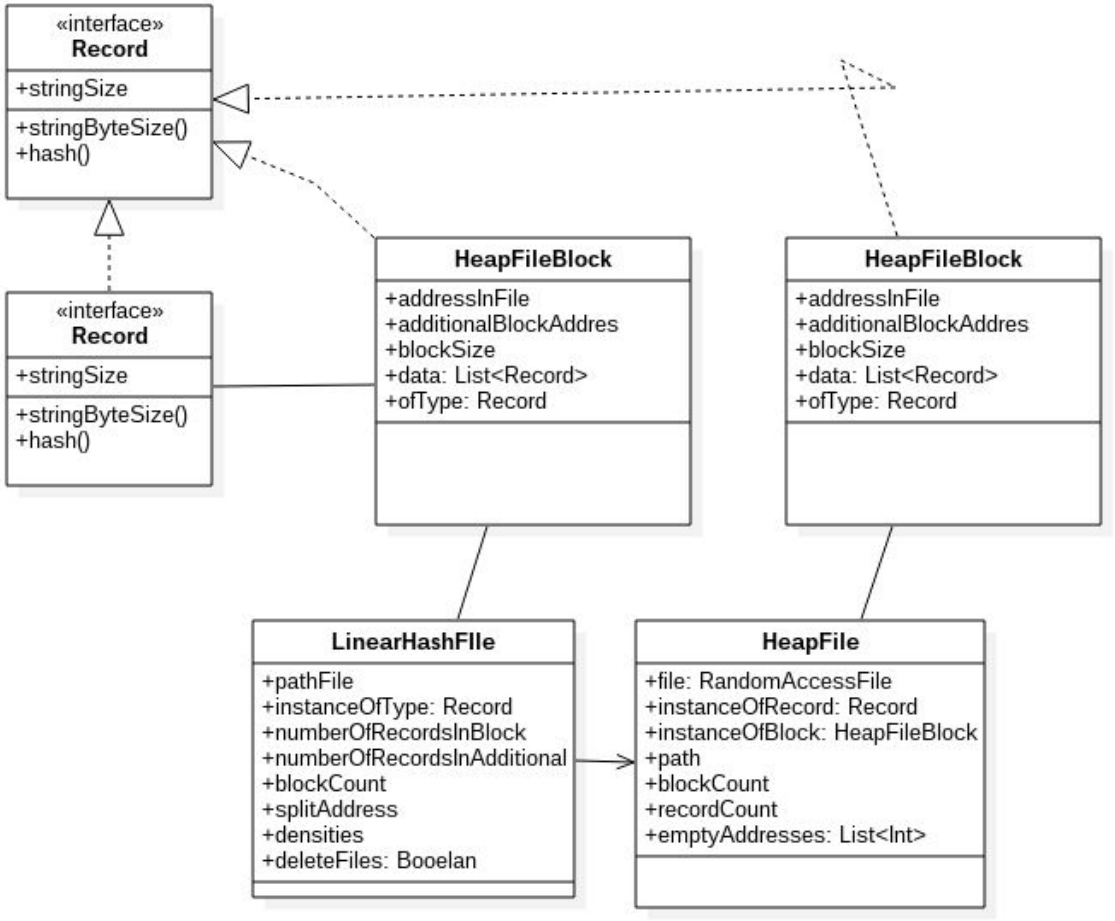
Pre účely testovania implementujte aj operáciu zobrazenia celého aktuálneho obsahu databázy (vrátane prepojenia blokov, preplňujúceho súboru, prípadne ďalších súborov) – obsah všetkých súborov aplikácie bude možné v aplikácií sekvenčne vypísať tak, aby bolo vidieť, čo jednotlivé bloky obsahujú (vrátane ich interných atribútov). Export do textového súboru nie je potrebné implementovať, avšak

databáza sa dá používať aj po opätovnom zapnutí programu. V semestrálnej práci je potrebné použiť lineárne hešovanie, alebo B+ strom. Operáciu č. 4 implementujete iba pri využití B+ stromu. Implementujte efektívny manažment prázdnych blokov v súbore. Dátumy neevidujte ako reťazce (použite vhodný dátový typ). V dokumentácii uveďte presný počet prístupov do súboru pri jednotlivých operáciách (v rôznych situáciách). Nezabudnite na všeobecné požiadavky semestrálnych prác (napr. generátor na naplnenie databázy...). Dôležitou časťou je dôsledné oddelenie jadra aplikácie od GUI.

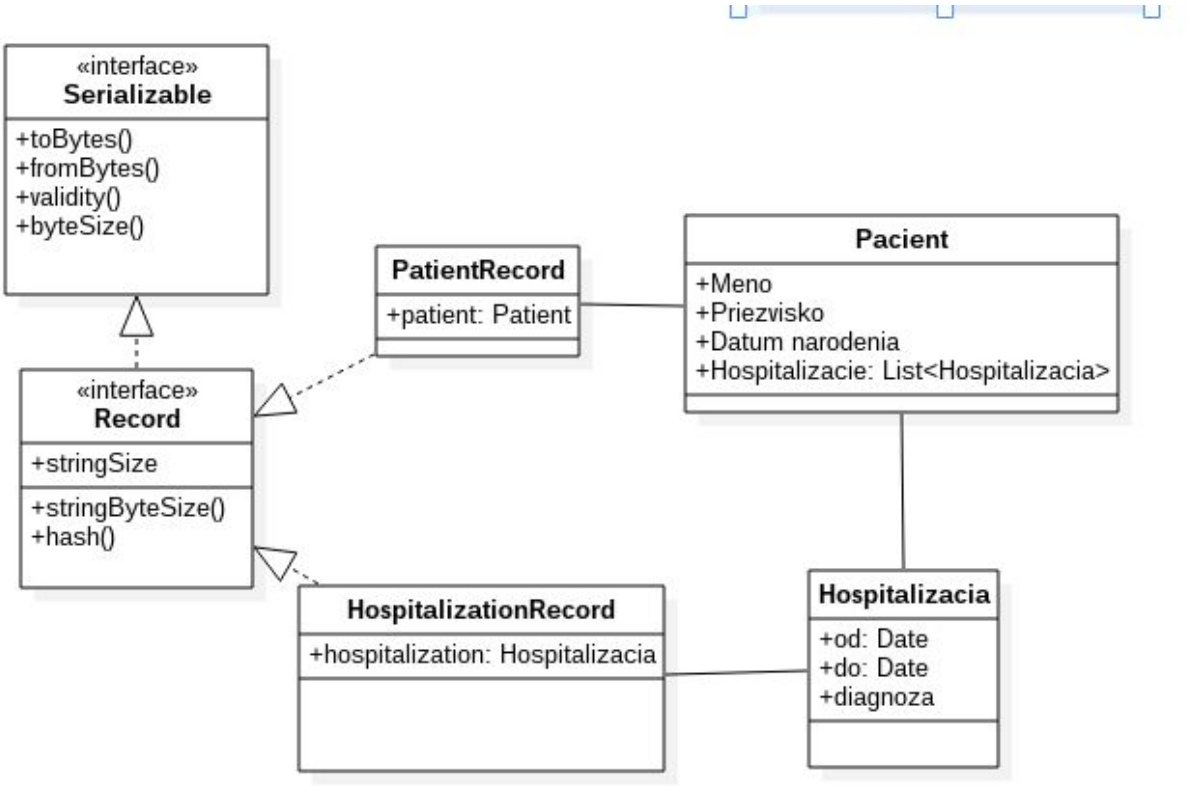
Riešenie

V semestrálnej práci je implementované lineárne hešovanie v jazyku Kotlin, ktoré využívam na organizáciu dát. Inštancia lineárneho hešovania (ďalej LinH) ktorá akceptuje triedy ktoré implementujú rozhranie *Record* ktoré sa stará o serializáciu objektov. Objekty sú zhlukované do blokov pevne danej veľkosti a pomocou hešovacej funkcie pridávanie na konkrétnu adresu. V prípade rovnakého výsledku funkcie a prekročení veľkosti bloku sa objekt uloží do preplňovacieho bloku ktorý sa nachádza v preplňovacom súbore.




«interface»
Record
+stringSize
+stringByteSize()
+hash()
«interface»
Record
+stringSize
+stringByteSize()
+hash()
HeapFileBlock
+addressInFile
+additionalBlockAddres
+blockSize
+data: List<Record>
+ofType: Record
HeapFileBlock
+addressInFile
+additionalBlockAddres
+blockSize
+data: List<Record>
+ofType: Record
LinearHashFIle
+pathFile
+instanceOfType: Record
+numberOfRecordsInBlock
+numberOfRecordsInAdditional
+blockCount
+splitAddress
+densities
+deleteFiles: Booelan
HeapFile
+file: RandomAccessFile
+instanceOfRecord: Record
+instanceOfBlock: HeapFileBlock
+path
+blockCount
+recordCount
+emptyAddresses: List<Int>

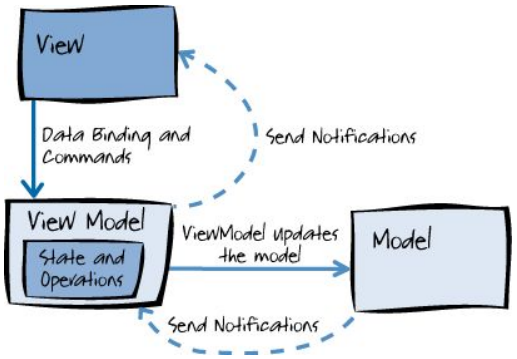
Triedy *Pacient a Hospitalizácia* používaju na serializáciu wrappere PatientRecord a HospitalizaionRecord ktoré implementujú rozhranie *Recordable* pričom kvôli fixnej veľkosti bloku je pevne nastavený maximálny počet hospitalizácii (100) ktoré má pacient a dĺžka reťazcov ako meno (25) , priezvisko(25) a diagnóza (40).



Kompletné UML je v v prílohe

GUI

Na grafické rozhranie som využil framework TornadoFX. Oddelenie od jadra aplikácie som zabezpečil wrapperemi nad modelovými triedami a implementáciou MVVM architektúry.



Výpočtová zložitosť

Vlož

1. Prečítaj blok na adrese hešu
2. Má voľné miesto
    a. Nájdi prvé voľné miesto a vlož
    b. Zapíš
    c. Ak treba rozdeľovať, rozdeľuj
    d. Koniec
3. Nájdi prepľňovací blok
    a. Nájdi prvé voľné miesto
        i. Vlož
        ii. Zapíš
        iii. Vráť hlavnému bloku správu o vložení
        iv. Zapíš hlavný blok
        v. Koniec
    b. Chod 3

Zápis hlavného bloku + poradie preplňovacieho bloku + zápis preplňovacieho

Rozdeľuj

1. Prečítaj blok ktorý rozdeľujem (z)
2. Alokuj a prečitaj blok kam rozdeľujem (do)
3. Prečítaj všetky záznamy (z)
4. Prečítaj všetky záznamy (do)
5. Pridaj (z)/(do)
    a. Ak má miesto pridaj záznam
    b. Rozdeľ načítane bloky na kúsky o veľkosti preplňovacieho bloku
    c. Vkladaj hlavnému bloku celé preplňovacie bloky
        i. Zapíš sa
        ii. Vráť správu o vložení
        iii. Aktualizuj metadáta hlavného bloku
    d. Zapíš hlavný blok
    e. Skús skrátiť preplňovací blok od konca
    f. Koniec

1 + 2 * Počet preplňovacích + Počet kúskov blokov * Poradie prelňovacieho + 1 + prázdne bloky na konci súboru

Nájdi

1. Prečítaj blok na adrese
2. Ak sa nachádza v bloku, vráť záznam,koniec
3. Daj další preplňovací
    a. Ak nemá, koniec
    b. Choď 1
        1 + poradie preplňovacieho bloku v ktorom sa blok nachádza

Uprav

1. Nájdi
2. Prepíš nájdený

Nájdi + 1

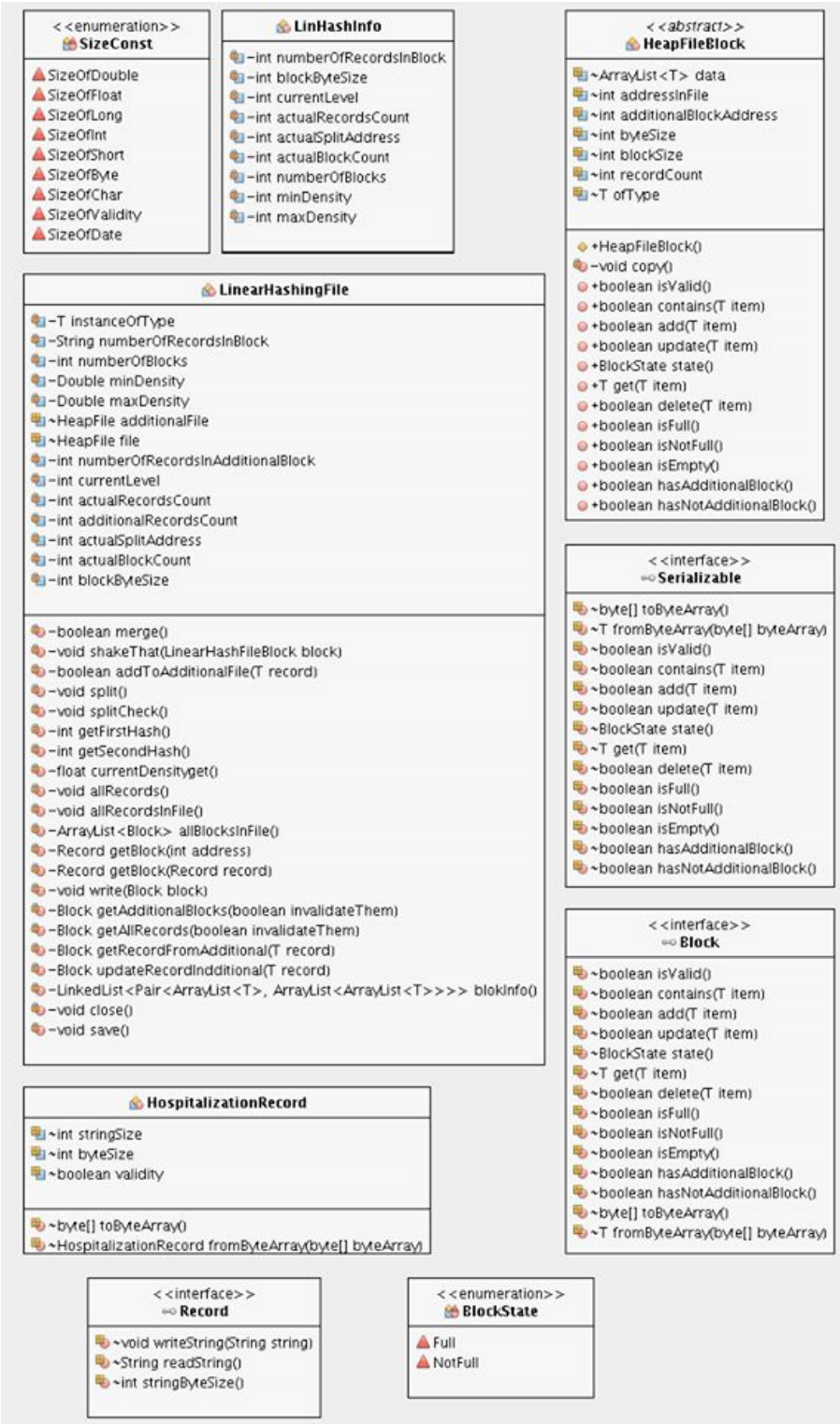
Vymaž

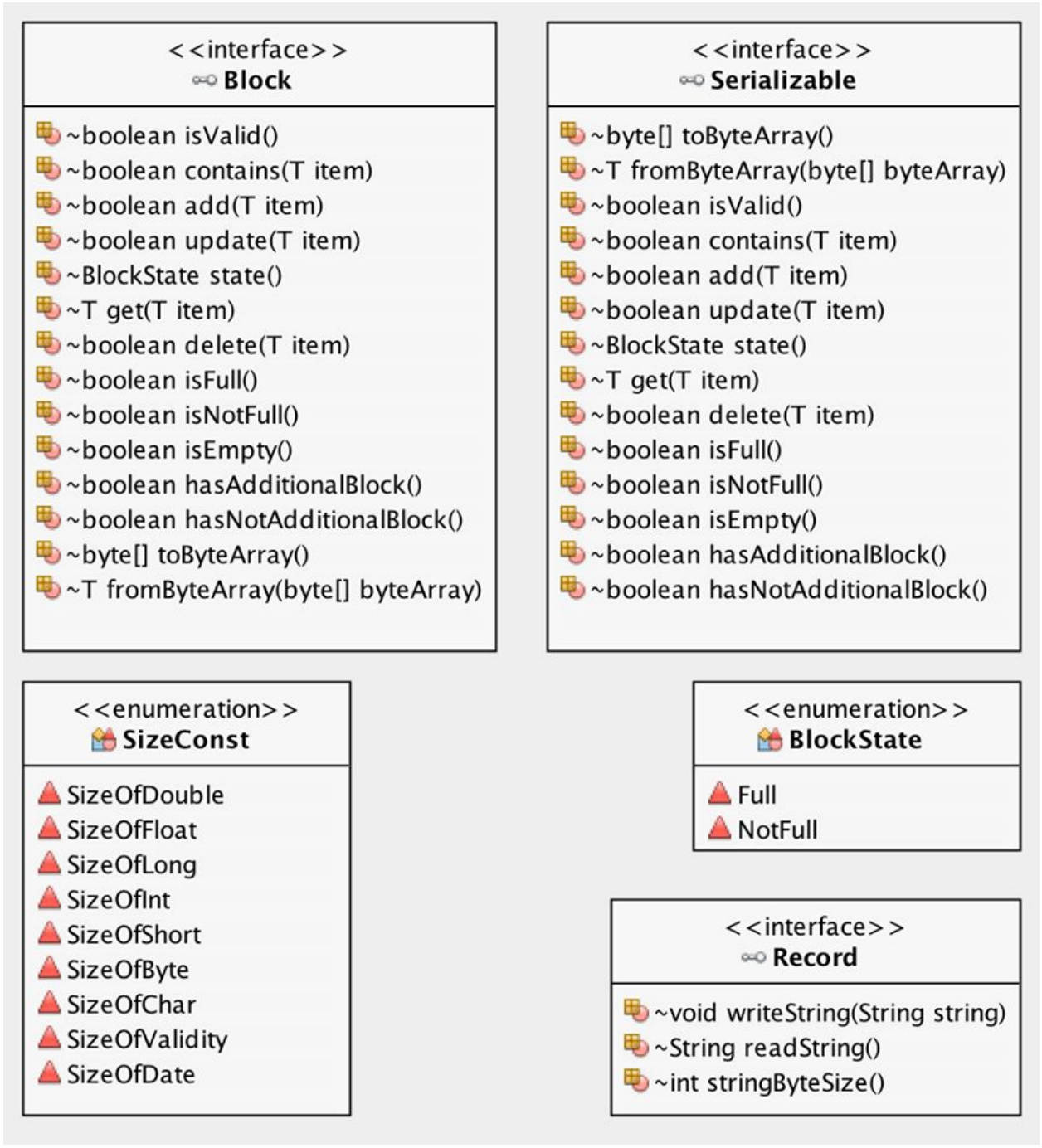
1. Nájdi
2. Označ ako neplatný
3. Zapíš
4. Ak môžem zlúčit, zlúč
5. Koniec

Zlúč

1. Prečítaj poslednú skupinu
2. Zober všetky jej záznamy
3. Prečítaj skupinu do ktorej vkladám
    a. Skús do nej vložiť záznamy
    b. Rozdeľ načítane bloky na kúsky o veľkosti preplňovacieho bloku
    c. Vkladaj hlavnému bloku celé preplňovacie bloky
        i. Zapíš sa
        ii. Vráť správu o vložení
        iii. Aktualizuj metadáta hlavného bloku
    d. Zapíš hlavný blok
    e. Skús hlavný blok od konca
    f. Koniec

Príloha - UML


<<enumeration>>
SizeConst
SizeOfDouble
SizeOfFloat
SizeOfLong
SizeOfInt
SizeOfShort
SizeOfByte
SizeOfChar
SizeOfValidity
SizeOfDate
LinHashInfo
-int numberOfRecordsInBlock
-int blockByteSize
-int currentLevel
-int actualRecordsCount
-int actualSplitAddress
-int actualBlockCount
-int numberOfBlocks
-int minDensity
-int maxDensity
<<abstract>>
HeapFileBlock
~ArrayList<T> data
~int addressInFile
~int additionalBlockAddress
~int byteSize
~int blockSize
~int recordCount
~T ofType
+HeapFileBlock()
-void copy()
+boolean isValid()
+boolean contains(T item)
+boolean add(T item)
+boolean update(T item)
+BlockState state()
+T get(T item)
+boolean delete(T item)
+boolean isFull()
+boolean isNotFull()
+boolean isEmpty()
+boolean hasAdditionalBlock()
+boolean hasNotAdditionalBlock()
LinearHashingFile
-T instanceOfType
-String numberOfRecordsInBlock
-int numberOfBlocks
-Double minDensity
-Double maxDensity
~HeapFile additionalFile
~HeapFile file
-int numberOfRecordsInAdditionalBlock
-int currentLevel
-int actualRecordsCount
-int additionalRecordsCount
-int actualSplitAddress
-int actualBlockCount
-int blockByteSize
-boolean merge()
-void shakeThat(LinearHashFileBlock block)
-boolean addToAdditionalFile(T record)
-void split()
-void splitCheck()
-int getFirstHash()
-int getSecondHash()
-float currentDensityget()
-void allRecords()
-void allRecordsInFile()
-ArrayList<Block> allBlocksInFile()
-Record getBlock(int address)
-Record getBlock(Record record)
-void write(Block block)
-Block getAdditionalBlocks(boolean invalidateThem)
-Block getAllRecords(boolean invalidateThem)
-Block getRecordFromAdditional(T record)
-Block updateRecordIndditional(T record)
-LinkedList<Pair<ArrayList<T>, ArrayList<ArrayList<T>>>> blokInfo()
-void close()
-void save()
<<interface>>
Serializable
~byte[] toByteArray()
~T fromByteArray(byte[] byteArray)
~boolean isValid()
~boolean contains(T item)
~boolean add(T item)
~boolean update(T item)
~BlockState state()
~T get(T item)
~boolean delete(T item)
~boolean isFull()
~boolean isNotFull()
~boolean isEmpty()
~boolean hasAdditionalBlock()
~boolean hasNotAdditionalBlock()
<<interface>>
Block
~boolean isValid()
~boolean contains(T item)
~boolean add(T item)
~boolean update(T item)
~BlockState state()
~T get(T item)
~boolean delete(T item)
~boolean isFull()
~boolean isNotFull()
~boolean isEmpty()
~boolean hasAdditionalBlock()
~boolean hasNotAdditionalBlock()
~byte[] toByteArray()
~T fromByteArray(byte[] byteArray)
HospitalizationRecord
~int stringSize
~int byteSize
~boolean validity
~byte[] toByteArray()
~HospitalizationRecord fromByteArray(byte[] byteArray)
<<interface>>
Record
~void writeString(String string)
~String readString()
~int stringByteSize()
<<enumeration>>
BlockState
Full
NotFull


<<interface>>
Block
~boolean isValid()
~boolean contains(T item)
~boolean add(T item)
~boolean update(T item)
~BlockState state()
~T get(T item)
~boolean delete(T item)
~boolean isFull()
~boolean isNotFull()
~boolean isEmpty()
~boolean hasAdditionalBlock()
~boolean hasNotAdditionalBlock()
~byte[] toByteArray()
~T fromByteArray(byte[] byteArray)
<<interface>>
Serializable
~byte[] toByteArray()
~T fromByteArray(byte[] byteArray)
~boolean isValid()
~boolean contains(T item)
~boolean add(T item)
~boolean update(T item)
~BlockState state()
~T get(T item)
~boolean delete(T item)
~boolean isFull()
~boolean isNotFull()
~boolean isEmpty()
~boolean hasAdditionalBlock()
~boolean hasNotAdditionalBlock()
<<enumeration>>
SizeConst
SizeOfDouble
SizeOfFloat
SizeOfLong
SizeOfInt
SizeOfShort
SizeOfByte
SizeOfChar
SizeOfValidity
SizeOfDate
<<enumeration>>
BlockState
Full
NotFull
<<interface>>
Record
~void writeString(String string)
~String readString()
~int stringByteSize()