**ŽILINSKÁ UNIVERZITA V ŽILINE**

Fakulta riadenia a informatiky

# Semestrálna práca 2 AaUŠ 2

Michael Holý

## Systém pre evidenciu návštev autoservisu

Cvičiaci : Ing. Peter Jankovič, Phd.

Žilina, 2024

**Obsah**

# 1 Zadanie semestrálnej práce

Cieľom semestrálnej práce bolo vytvoriť demonštračnú verziu softvéru pre informačný systém na evidenciu návštev autoservisu, pričom systém mal byť implementovaný s dôrazom na rýchlosť vyhľadávania a nízkou spotrebou pamäte RAM.

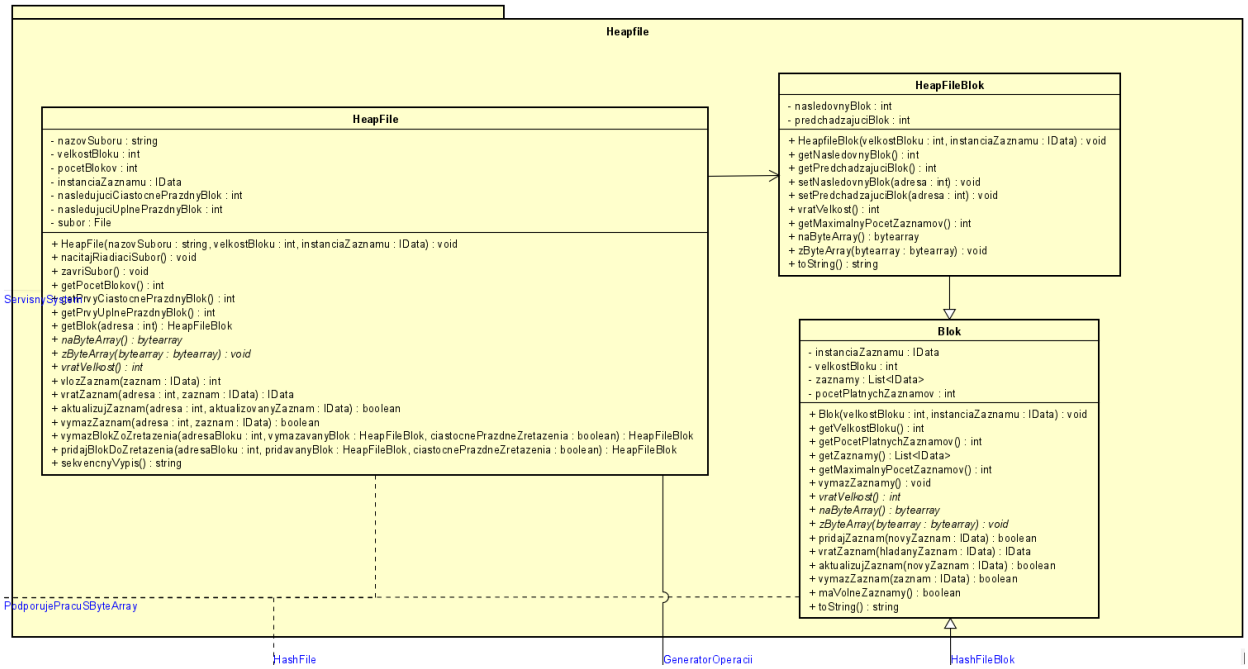
V rámci práce bolo potrebné:

- Implementovať neutriedený súbor dát na disku ako aj rozšíriteľné hešovanie.
- Vytvoriť aplikáciu umožňujúcu pridávanie, vyhľadávanie, úpravu a vymazávanie záznamov o návštevách autoservisu na základe zadaného jedinečného ID alebo ECV záznamu.
- Zabezpečiť aby bolo možné aplikáciu zavrieť a neskôr pokračovať v práci s uloženými dátami.

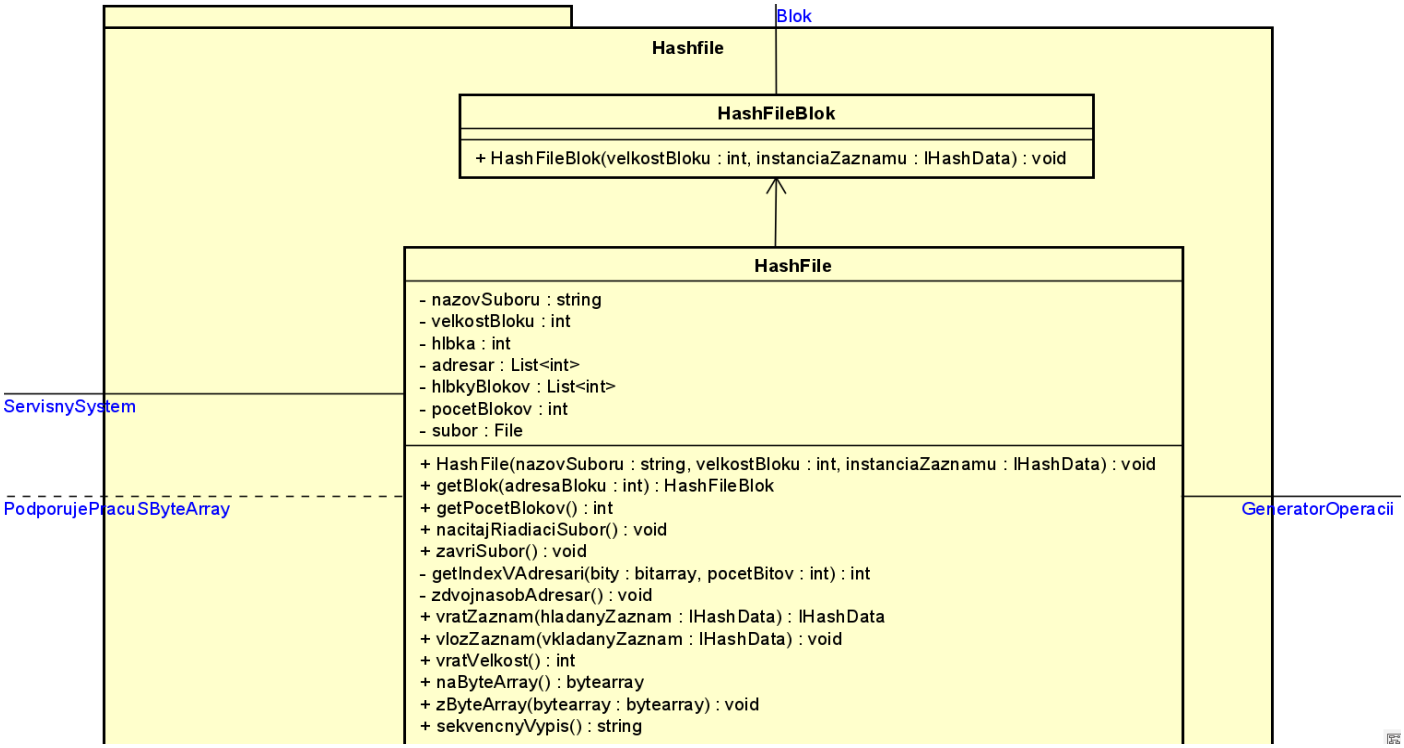
Počas implementácie neutriedeného súboru na disku a rozšíriteľného hešovania mal byť dôraz na čo najmenší počet prístupov k súboru počas jednotlivých operácií.

# 2 Návrh UML diagramu tried

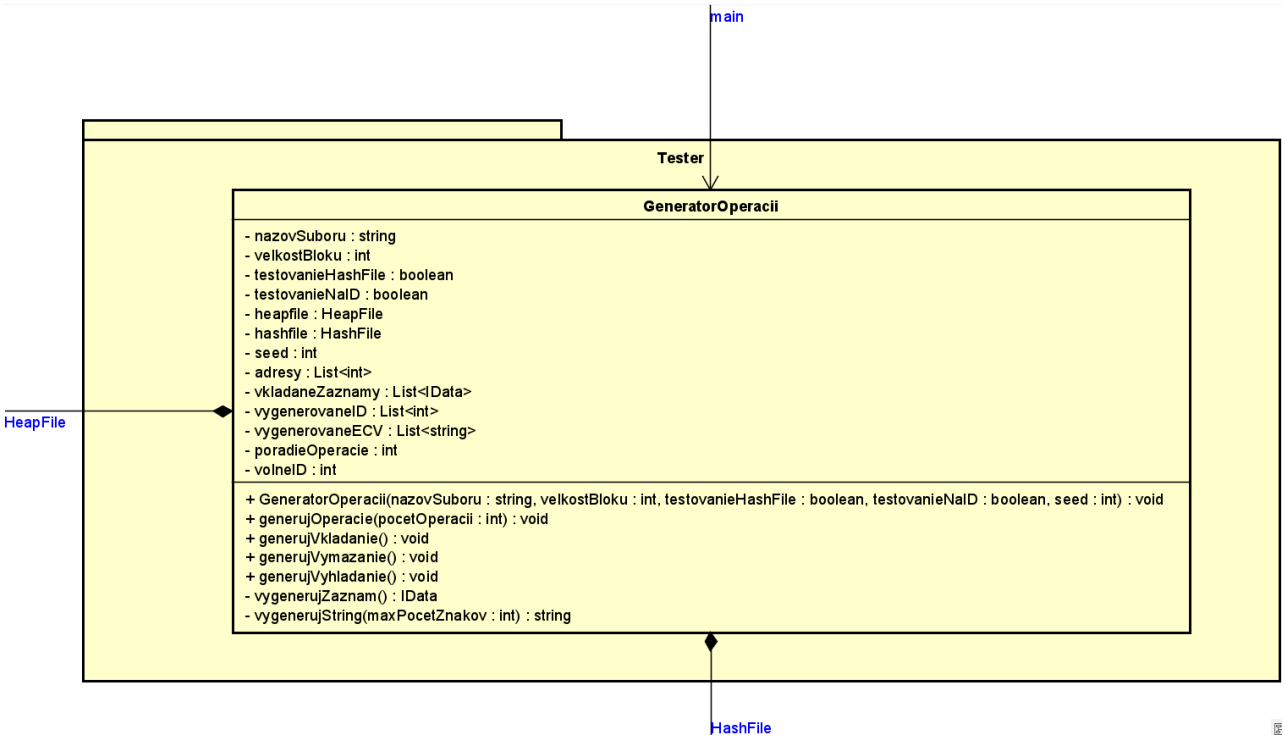
## 2.1 Balíček Heapfile



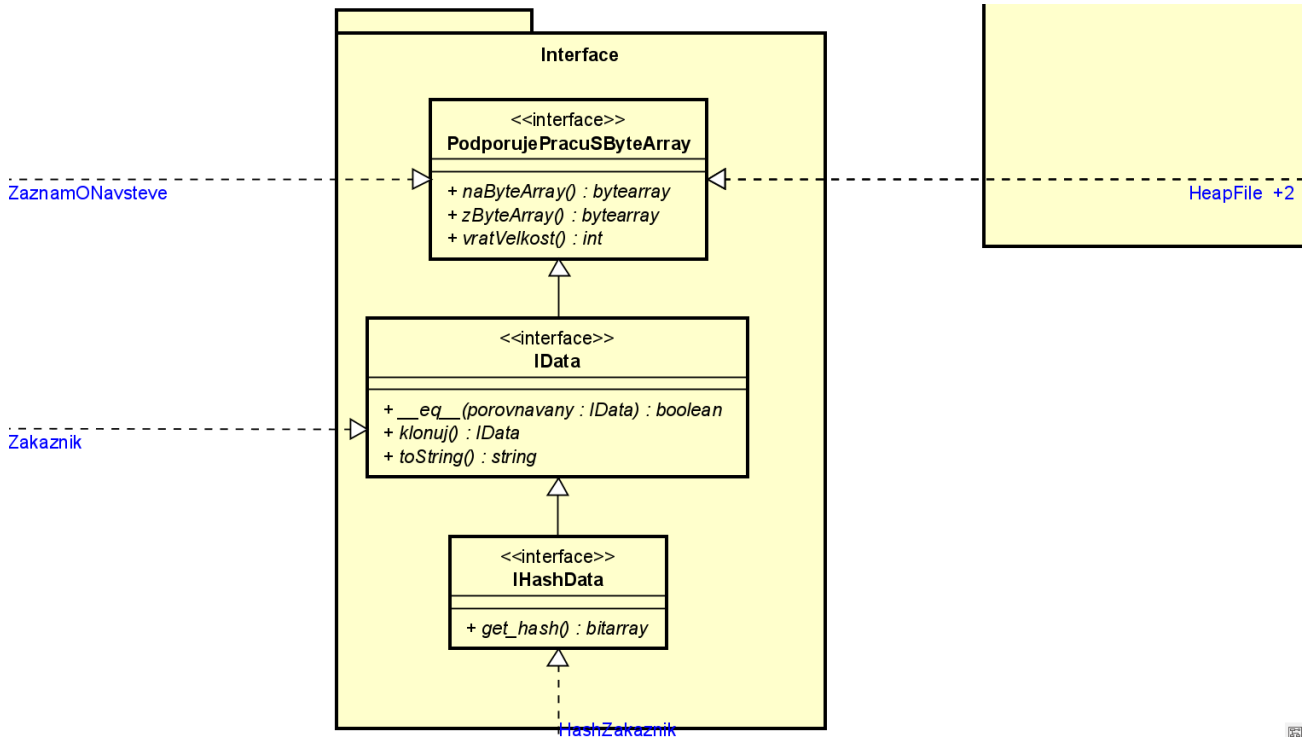

## 2.2 Balíček Hashfile



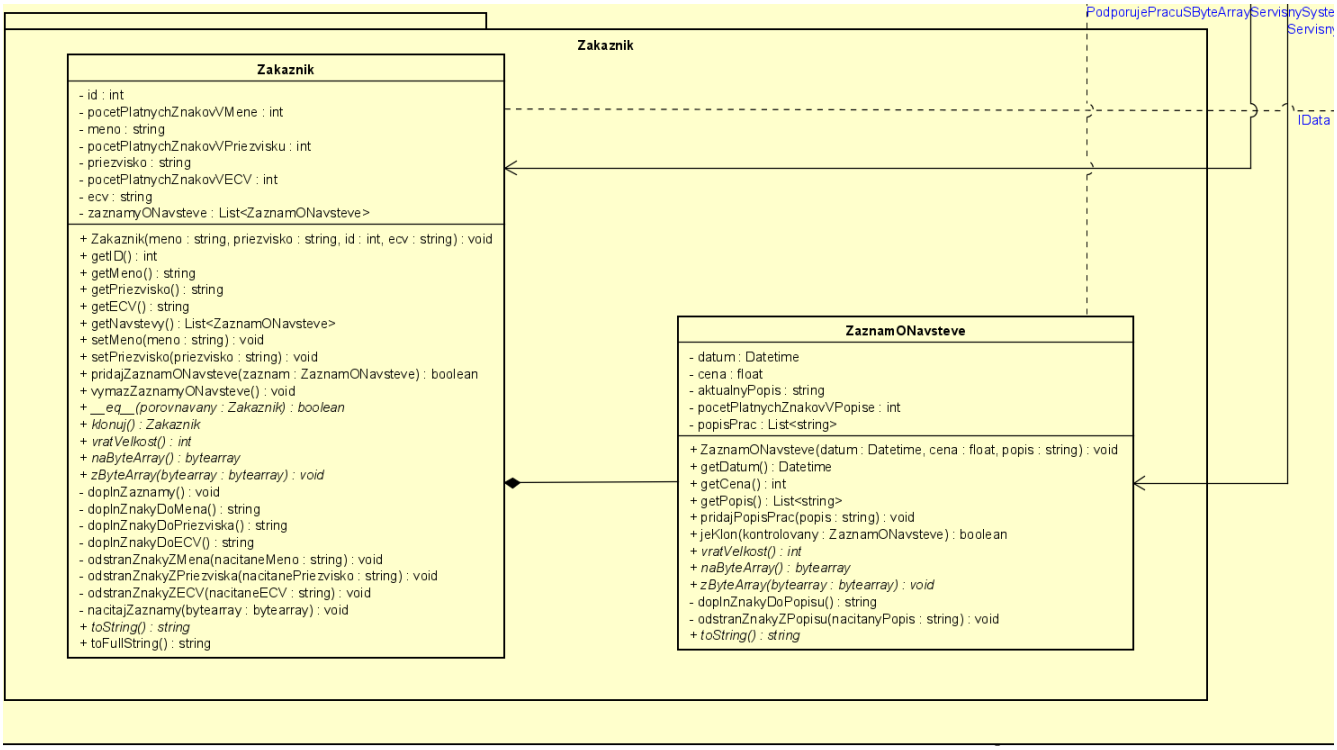
## 2.3 Balíček testera navrhnutých štruktúr



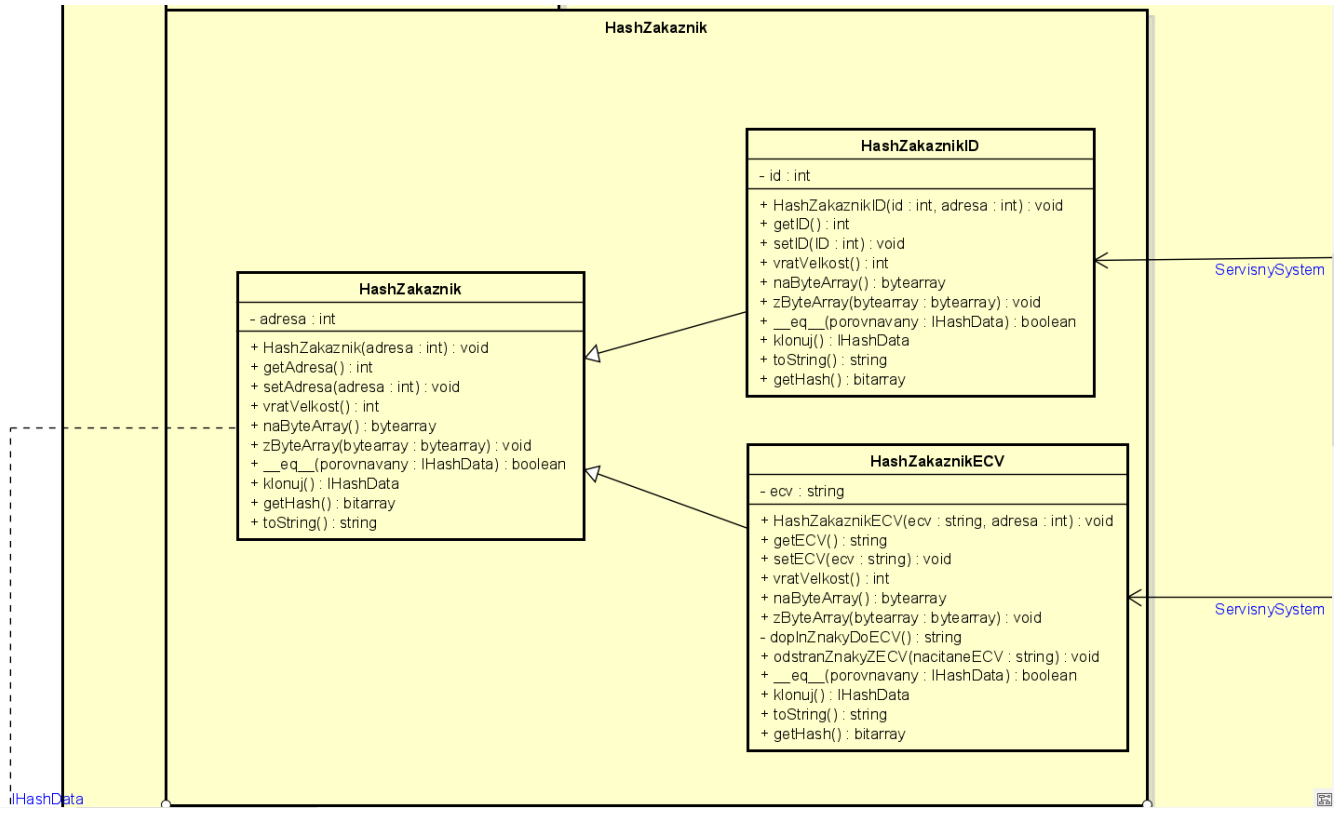

## 2.4 Balíček interfacov



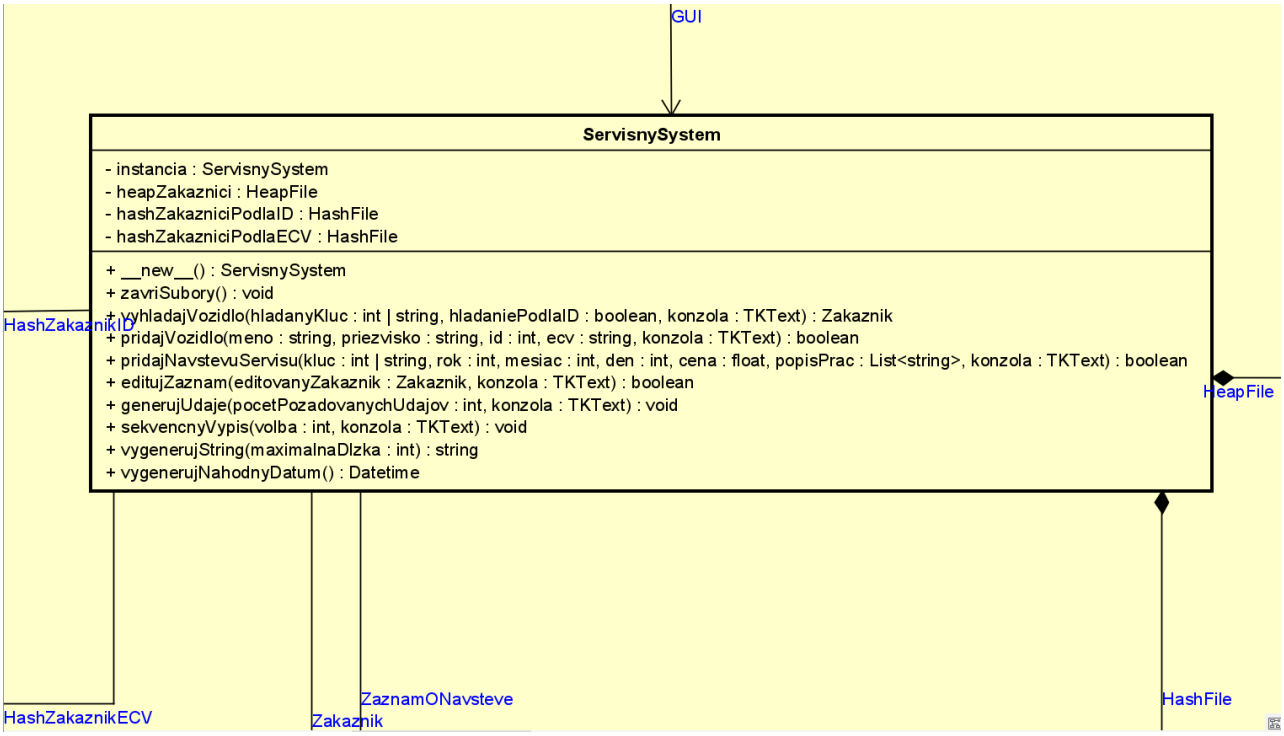
## 2.5 Balíček systém – triedy Zákazník, ZáznamONávšteve



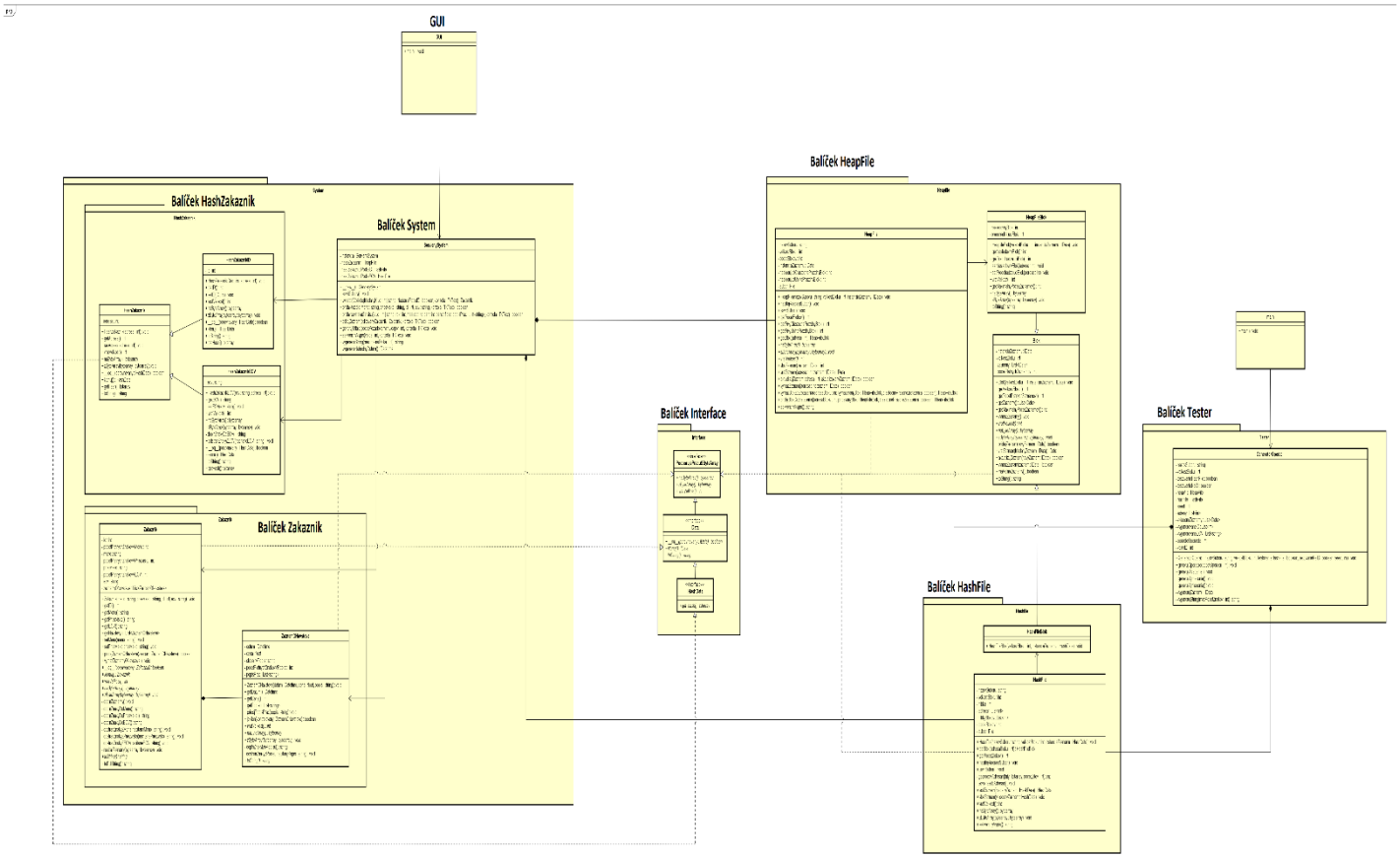

## 2.6 Balíček systém – triedy HashZakaznik, HashZ..ID, HashZ..ECV



## 2.7 Balíček systém – Trieda ServisnySystem



## 2.8 Celkový UML diagram



# 3 Rozbor návrhu štruktúry programu

## 3.1 Balíček Heapfile

Balíček **HeapFile** spravuje ukladanie a manipuláciu dát vo forme blokov v rámci neutriedeného súboru dát na disku.

Trieda **HeapFileBlok** reprezentuje jednotlivé bloky používané počas práce s neutriedeným súborom pričom dedí z triedy **Blok**, ktorá reprezentuje všeobecný blok dát, s ktorým je možné pracovať počas práce s binárnymi súbormi. Tieto bloky umožňujú správu ich obsahu, vrátane pridávania, aktualizácie či mazania záznamov.

Pre správnu funkčnosť štruktúry sa vyžaduje, aby všetky vkladané záznamy implementovali interface **IData**.

## 3.2 Balíček Hashfile

Balíček **HashFile** slúži na efektívne ukladanie a vyhľadávanie dát pomocou hešovania.

Trieda **HashFile** spravuje súbor, do ktorého sú zapisované bloky binárnych dát na základe výsledku hešovacej funkcie vkladaných záznamov.

Trieda **HashFileBlok** reprezentuje jednotlivé bloky obsahujúce ukladané záznamy a rovnako ako HeapFileBlok, dedí zo všeobecnej triedy Blok. Poskytuje operácie na vloženie, vyhľadávanie a odstránenie záznamov.

Pre správnu funkčnosť štruktúry sa vyžaduje, aby všetky vkladané záznamy implementovali interface **IHashData**.

## 3.3 Balíček Interface

Balíček obsahuje tri rozhrania:

- **PodporujePracuSByteArray** - na prácu s bajtovými poľami a konverziu objektov na / z bajtových polí.
- **IData** - na všeobecnú manipuláciu s vkladanými záznamami (porovnanie, klonovanie, textová reprezentácia) ako aj podporu práce s bajtovými poľami, keďže tento interface dedí z interfacu PodporujePracuSByteArray.
- **IHashData** na získavanie hešov dát vo forme bitového poľa, ako aj manipuláciu s vkladanými záznamami a prácu s bajtovými poľami, keďže dedí z vyššie spomenutých interfacov.

## 3.4 Balíček testera navrhnutých štruktúr

Tento balíček obsahuje komponenty pre testovanie navrhnutého Heapfile a Hashfile, pričom umožňuje voľbu medzi testovaním štruktúry Heapfile, alebo testovanie Hashfile pre 2 rôzne typy testovacích kľúčov – ID (unikátne celé číslo) alebo ECV (unikátny string).

## 3.5 Balíček systém

Balíček **System** sa skladá z troch podbalíčkov:

- **Zakaznik** - Tento podbalíček obsahuje triedu reprezentujúcu zákazníka a jeho návštevy servisu. Umožňuje manipuláciu s údajmi zákazníka, ako sú jeho meno, priezvisko, ID, ECV a jeho zoznam návštev, pričom každá návšteva obsahuje dátum, cenu a zoznam popisov prác.
- **HashZakaznik**: Tento podbalíček predstavuje objekty, ktoré sa využívajú na ukladanie informácií o zákazníkoch uložených v neutriedenom súbore a to najmä ich kľúč – ID alebo ECV, podľa ktorého sú vkladané a vyhľadávané v hešovacom súbore ako aj ich adresa, na ktorej sú uložený v neutriedenom súbore.
- **ServisnySystem**: Hlavný systém, ktorý spravuje zákazníkov a ich návštevy servisu. Poskytuje funkcie na pridávanie, úpravu, vyhľadávanie zákazníkov a návštev ako aj generovanie údajov do databázy a sekvenčný výpis štruktúr **HeapFile** a **HashFile**.

# 4 Rozbor implementácie použitých údajových štruktúr

## 4.1 Heapfile

### 4.1.1 Operácia vlož

Počas vykonávania tejto operácie môžu nastať tri rôzne situácie:

- **Vkladanie do prázdneho súboru**: Ak súbor neobsahuje žiadne bloky, vytvorí sa nový blok, záznam sa pridá, a ak má blok ešte voľné miesto, nastaví sa ako čiastočne prázdny.

```
def vloz_zaznam(self, zaznam : IData) -> int:
    Vlozi zaznam do suboru.
    """
    if isinstance(zaznam, type(self.__instancia_zaznamu)):

        if self.__pocet_blokov == 0:
            self.__pocet_blokov += 1
            blok = Blok(self.__velkost_bloku, self.__instancia_zaznamu)
            if blok.pridaj_zaznam(zaznam):
                if blok.ma_volne_zaznamy():
                    if blok.get_maximalny_pocet_zaznamov() != 1:
                        self.__nasledujuci_ciastocne_prazdny_blok = 0

                self.__subor.seek(0)
                self.__subor.write(blok.na_byte_array())

                print(f'Zaznam bol uspesne vlozeny na adrese 0 -> {zaznam.get_id()}')

                return 0
            else:
                return -1
```

- **Vkladanie na koniec súboru**: Ak nie sú k dispozícii prázdne ani čiastočne prázdne bloky, vytvorí sa nový blok na konci súboru a záznam sa do neho uloží.

```
def vloz_zaznam(self, zaznam : IData) -> int:

    #Vkladanie na koniec súboru kedže neexistuje čiastočne ani úplne prázdny blok
    elif self.__pocet_blokov > 0 and self.__nasledujuci_ciastocne_prazdny_blok == -1 and self.__nasledujuci_uplne_prazdny_blok == -1:
        adresa_noveho_bloku = self.__pocet_blokov * self.__velkost_bloku
        novy_blok = Blok(self.__velkost_bloku, self.__instancia_zaznamu)

        if novy_blok.pridaj_zaznam(zaznam):
            self.__pocet_blokov += 1
            if novy_blok.ma_volne_zaznamy():
                if novy_blok.get_maximalny_pocet_zaznamov() != 1:
                    novy_blok = self.__pridaj_blok_do_zretazenia(adresa_noveho_bloku, novy_blok, True)


            self.__subor.seek(adresa_noveho_bloku)
            self.__subor.write(novy_blok.na_byte_array())



            print(f'Zaznam bol uspesne vlozeny na adrese {adresa_noveho_bloku} -> {zaznam.get_id()}')
            return adresa_noveho_bloku
        else:
            return -1
```

- **Vkladanie do existujúceho bloku**: Ak existuje čiastočne alebo úplne prázdny blok, tento blok sa načíta, pričom sa prednostne berú najprv čiastočne prázdne bloky a záznam sa do neho vloží. Na základe stavu, do akého sa blok po vložení dostane (stane sa napr. čiastočne prázdnym) sa taktiež upravia aj potrebné zreťazenia blokov.

```
class HeapFile(PodporujePracuSByteArray):
    def vloz_zaznam(self, zaznam : IData) -> int:
            else:
                #ak je počet blokov viac ako 0 a existuje čiastočne prázdny blok alebo úplne prázdny blok
                if self.__nasledujuci_ciastocne_prazdny_blok != -1:
                    adresa_bloku = self.__nasledujuci_ciastocne_prazdny_blok
                elif self.__nasledujuci_uplne_prazdny_blok != -1:
                    adresa_bloku = self.__nasledujuci_uplne_prazdny_blok


                self.__subor.seek(adresa_bloku)
                blok = Blok(self.__velkost_bloku, self.__instancia_zaznamu)
                blok.z_byte_array(self.__subor.read(self.__velkost_bloku))

                if blok.pridaj_zaznam(zaznam):

                    if adresa_bloku == self.__nasledujuci_ciastocne_prazdny_blok:
                        if not blok.ma_volne_zaznamy():
                            blok = self.__vymaz_blok_zo_zretazenia(adresa_bloku, blok, True)

                        self.__subor.seek(adresa_bloku)
                        self.__subor.write(blok.na_byte_array())



                    else:
                        blok = self.__vymaz_blok_zo_zretazenia(adresa_bloku, blok, False)
                        if blok.get_maximalny_pocet_zaznamov() != 1:
                            blok = self.__pridaj_blok_do_zretazenia(adresa_bloku, blok, True)

                        self.__subor.seek(adresa_bloku)
                        self.__subor.write(blok.na_byte_array())

                else:
                    return -1
```

Taktiež sa kontroluje počas úpravy zreťazenia blokov, či sa nejedná o špeciálny prípad, kedy sa do bloku zmestí iba jediný záznam (čiastočné zreťazenie nepotrebné).

### 4.1.2 Operácia nájdi

Operácia nájdi (vráť záznam) má nasledovný priebeh:

- **Kontrola adresy**: Overí, či je adresa platná , ak nie, vráti None.
- **Načítanie bloku**: Načíta blok zo zadanej adresy.
- **Vrátenie záznamu**: Záznam sa vyhľadá v načítanom bloku pomocou komparátora a pokiaľ sa v danom bloku záznam nachádza, vráti sa.

```
def vrat_zaznam(self, adresa, zaznam)-> IData:
    """
    Vráti záznam zo zadanej adresy, pokiaľ sa na danej adrese nájde.
    """
    #Ak sú zadané neplatné adresy
    if adresa > (self.__pocet_blokov - 1) * self.__velkost_bloku:
        return None
    elif adresa % self.__velkost_bloku != 0:
        return None

    self.__subor.seek(adresa)
    blok = Blok(self.__velkost_bloku, self.__instancia_zaznamu)
    blok.z_byte_array(self.__subor.read(self.__velkost_bloku))
    return blok.vrat_zaznam(zaznam)
```

### 4.1.3 Operácia aktualizuj

Operácia aktualizuj má podobný priebeh ako operácia nájdi, najprv sa skontroluje platnosť adresy, následne sa načíta blok na zadanej adrese a ako posledné sa vyhľadá záznam, ktorý sa má **aktualizovať**, pokiaľ sa nájde, tak sa tento záznam prepíše na aktualizovaný záznam.

```
def aktualizuj_zaznam(self, adresa, aktualizovany_zaznam: IData) -> bool:
    """
    Aktualizuje záznam na zadanej adrese, pokiaľ sa na danej adrese nájde.
    """
    #Ak sú zadané neplatné adresy
    if adresa > (self.__pocet_blokov - 1) * self.__velkost_bloku:
        return False
    elif adresa % self.__velkost_bloku != 0:
        return False

    self.__subor.seek(adresa)
    blok = Blok(self.__velkost_bloku, self.__instancia_zaznamu)
    blok.z_byte_array(self.__subor.read(self.__velkost_bloku))
    if blok.aktualizuj_zaznam(aktualizovany_zaznam):
        self.__subor.seek(adresa)
        self.__subor.write(blok.na_byte_array())
        return True
    else:
        return False
```

### 4.1.4 Operácia vymaž

Operácia vymaž má nasledovný priebeh:

Ako prvé sa **skontroluje adresa** – overí sa, či je adresa platná , ak nie, vráti False.

Nasleduje **načítanie bloku** - načíta blok na zadanej adrese a pokúsi sa z neho vymazať záznam, pokiaľ je toto úspešné, tak sa môže blok dostať do nasledovných stavov po vymazaní:

**Ak je blok úplne prázdny:**

- Ak je posledným blokom, odstráni ho a kontroluje či sa za ním nenachádzajú ďalšie úplne prázdne bloky. Súbor sa prípadne skráti na posledný ne-prázdny blok alebo až na začiatok súboru, v prípade, že súbor nie je skrátený až na začiatok, postupne sa vymažú zo zreťazení všetky za sebou uložené prázdne bloky, ktoré sa skrátením súboru vymazali.

```
# Ak je blok posledný v súbore
if adresa == (self.__pocet_blokov - 1) * self.__velkost_bloku:

    if blok.get_maximalny_pocet_zaznamov() != 1:
        blok = self.__vymaz_blok_zo_zretazenia(adresa, blok, True)
    self.__pocet_blokov -= 1
    adresa_kontrolovaneho_bloku = adresa - self.__velkost_bloku
    kontrolovany_blok = Blok(self.__velkost_bloku, self.__instancia_zaznamu)

    adresy_blokov_na_vymazavanie_zo_zretazenia = []
    vymazavane_bloky_zo_zretazenia = []

    while True:
        if self.__pocet_blokov > 0:
            self.__subor.seek(adresa_kontrolovaneho_bloku)
            kontrolovany_blok.z_byte_array(self.__subor.read(self.__velkost_bloku))

            if kontrolovany_blok.get_pocet_platnych_zaznamov() != 0:
                break

            adresy_blokov_na_vymazavanie_zo_zretazenia.append(adresa_kontrolovaneho_bloku)
            vymazavane_bloky_zo_zretazenia.append(kontrolovany_blok)

            self.__pocet_blokov -= 1
            adresa_kontrolovaneho_bloku -= self.__velkost_bloku
            kontrolovany_blok = Blok(self.__velkost_bloku, self.__instancia_zaznamu)
        else:
            break

    #Pokiaľ sa súbor neskráti až na začiatok, tak sa vymažú všetky prázdne bloky zo zreťazenia, ktoré sa skrátením vymažú
    #Pokiaľ sa súbor skráti až na začiatok, čiže sa vymažú všetky bloky, nie je potrebné pristupovať k súboru navyše tým že sa postupne vymažú zo zreťazenia
    if self.__pocet_blokov != 0:
        for i in range(len(adresy_blokov_na_vymazavanie_zo_zretazenia)):
            kontrolovany_blok = self.__vymaz_blok_zo_zretazenia(adresy_blokov_na_vymazavanie_zo_zretazenia[i], vymazavane_bloky_zo_zretazenia[i], False)
    if self.__pocet_blokov == 0:
        self.__nasledujuci_ciastocne_prazdny_blok = -1
        self.__nasledujuci_uplne_prazdny_blok = -1
    self.__subor.truncate(adresa_kontrolovaneho_bloku + self.__velkost_bloku)

    return True
```

- Ak nie je posledným blokom, upravia sa potrebné zreťazenia.

```
# Ak blok nie je posledný v súbore
else:
    if blok.get_maximalny_pocet_zaznamov() != 1:
        blok = self.__vymaz_blok_zo_zretazenia(adresa, blok, True) #Vymazanie z čiastočného zreťazenia, stane sa prázdnym blokom

    blok = self.__pridaj_blok_do_zretazenia(adresa, blok, False) #Pridanie do úplného zreťazenia
    self.__subor.seek(adresa)
    self.__subor.write(blok.na_byte_array())
    return True
```

**Ak je blok čiastočne prázdny:**

Pokiaľ bol blok pred vymazaním plný, pridá sa do zreťazenia čiastočne prázdnych blokov.

```
        # Blok sa stal čiastočne prázdnym
        else:

            if blok.get_pocet_platnych_zaznamov() == blok.get_maximalny_pocet_zaznamov() - 1 and blok.get_maximalny_pocet_zaznamov() != 1:
                blok = self.__pridaj_blok_do_zretazenia(adresa, blok, True)

            self.__subor.seek(adresa)
            self.__subor.write(blok.na_byte_array())
            return True


    else:
        return False
```

V oboch prípadoch sa pri úprave zreťazení kontroluje špeciálna situácia, kedy sa do bloku zmestí iba jediný záznam (čiastočné zreťazenie nepotrebné).

## 4.2 Hashfile

### 4.2.1 Operácia vlož

Operácia vlož má nasledovný priebeh:

- **Výpočet indexu**: Na základe hash-u kľúča vkladaného záznamu sa vypočíta index v adresári a zistí sa adresa bloku, do ktorého by mal byť vložený.

```
def vloz_zaznam(self, vkladany_zaznam: IHashData):
    """
    Vloží záznam do súboru.
    """
    hash_kluca = vkladany_zaznam.get_hash()
    zaznam_je_vlozeny = False
    while not zaznam_je_vlozeny:
        index_bloku_v_adresari = self.__get_index_v_adresari(hash_kluca, self.__hlbka)
        adresa_bloku = self.__adresar[index_bloku_v_adresari]
```

- **Pridanie do nového bloku**: Ak blok na danom indexe v adresári neexistuje, vytvorí sa nový blok a zapíše sa jeho adresa do adresára. V prípade, že hĺbka bloku je menšia ako hĺbka súboru, adresa nového bloku sa v adresári nastaví pre všetky bloky, ktoré sú z daného rozsahu hĺbky bloku.

```
if adresa_bloku == -1:
    novy_blok = HashFileBlok(self.__velkost_bloku, self.__instancia_zaznamu)
    novy_blok.pridaj_zaznam(vkladany_zaznam)
    adresa_noveho_bloku = self.__pocet_blokov * self.__velkost_bloku
    # Pokiaľ je hĺbka bloku menšia ako hĺbka súboru, tak sa nová adresa bloku zapíše po celom rozsahu blokov
    if self.__hlbky_blokov[index_bloku_v_adresari] < self.__hlbka:
        rozsah = 2 ** (self.__hlbka - self.__hlbky_blokov[index_bloku_v_adresari])
        index_bloku = index_bloku_v_adresari
        while True:
            if index_bloku % rozsah == 0:
                zaciatok_splitu = index_bloku
                break
            else:
                index_bloku -= 1

        koniec_splitu = zaciatok_splitu + rozsah
        index_po_splite = int((zaciatok_splitu + koniec_splitu) / 2)
        if index_bloku_v_adresari < index_po_splite:
            for i in range (zaciatok_splitu, index_po_splite):
                self.__adresar[i] = adresa_noveho_bloku
                self.__hlbky_blokov[i] += 1
        else:
            for i in range(index_po_splite, koniec_splitu):
                self.__adresar[i] = adresa_noveho_bloku
                self.__hlbky_blokov[i] += 1

    else:
        self.__adresar[index_bloku_v_adresari] = adresa_noveho_bloku
        self.__hlbky_blokov[index_bloku_v_adresari] = self.__hlbka


    self.__subor.seek(adresa_noveho_bloku)
    self.__subor.write(novy_blok.na_byte_array())
    self.__pocet_blokov += 1
    zaznam_je_vlozeny = True
```

- **Spracovanie plného bloku**: Ak je blok plný a hĺbka bloku je rovná hĺbke súboru, vykoná sa **zdvojnásobenie adresára**.

```
else:
    blok = HashFileBlok(self.__velkost_bloku, self.__instancia_zaznamu)
    self.__subor.seek(adresa_bloku)
    blok.z_byte_array(self.__subor.read(self.__velkost_bloku))

    #Zdvojnásobenie adresára
    if not blok.ma_volne_zaznamy():
        if self.__hlbky_blokov[index_bloku_v_adresari] == self.__hlbka:
            self.__zdvojnasob_adresar()
```

Následne sa vždy vykoná **split** bloku – čiže jeho rozdelenie. Blok sa rozdelí na dva nové bloky a jeho pôvodné záznamy sa následne presúvajú medzi tieto bloky podľa ich prepočítaných indexov na základe aktuálnej hĺbky súboru.

```
#SPLIT
pridavany_blok = HashFileBlok(self.__velkost_bloku, self.__instancia_zaznamu)
adresa_noveho_bloku = self.__pocet_blokov * self.__velkost_bloku
index_bloku_v_splite = self.__get_index_v_adresari(hash_kluca, self.__hlbka)
rozsah_splitovania = 2 ** (self.__hlbka - self.__hlbky_blokov[index_bloku_v_splite])

index_bloku = index_bloku_v_splite
while True:
    if index_bloku % rozsah_splitovania == 0:
        zaciatok_splitu = index_bloku
        break
    else:
        index_bloku -= 1
koniec_splitu = zaciatok_splitu + rozsah_splitovania
index_po_splite = int((zaciatok_splitu + koniec_splitu) / 2)
zaznamy: List[IHashData] = blok.get_zaznamy()
pocet_platnych_zaznamov = blok.get_pocet_platnych_zaznamov()
presuvane_zaznamy: List[IHashData] = []
for i in range(pocet_platnych_zaznamov):
    presuvane_zaznamy.append(zaznamy[i])
blok.vymaz_zaznamy()


for zaznam in presuvane_zaznamy:
    hash_zaznamu = zaznam.get_hash()
    index_presuvaneho_zaznamu = self.__get_index_v_adresari(hash_zaznamu, self.__hlbka)
    if index_presuvaneho_zaznamu < index_po_splite:
        blok.pridaj_zaznam(zaznam)
    else:
        pridavany_blok.pridaj_zaznam(zaznam)
```

Po vykonaní splitu sa aktualizuje adresár a hĺbky príslušných blokov, pričom v prípade rozdelenia záznamov medzi oba vytvorené bloky sa nový blok zapíše do súboru.

```
        if blok.get_pocet_platnych_zaznamov() != 0 and pridavany_blok.get_pocet_platnych_zaznamov() != 0:
            for i in range(zaciatok_splitu, koniec_splitu):
                if i < index_po_splite:
                    self.__adresar[i] = adresa_bloku
                    self.__hlbky_blokov[i] += 1
                else:
                    self.__adresar[i] = adresa_noveho_bloku
                    self.__hlbky_blokov[i] += 1

            self.__subor.seek(adresa_bloku)
            self.__subor.write(blok.na_byte_array())
            self.__subor.seek(adresa_noveho_bloku)
            self.__subor.write(pridavany_blok.na_byte_array())
            self.__pocet_blokov += 1

        else:
            if blok.get_pocet_platnych_zaznamov() != 0:
                for i in range(zaciatok_splitu, koniec_splitu):
                    if i < index_po_splite:
                        self.__adresar[i] = adresa_bloku
                        self.__hlbky_blokov[i] += 1
                    else:
                        self.__adresar[i] = -1
                        self.__hlbky_blokov[i] += 1



            else:
                for i in range(zaciatok_splitu, koniec_splitu):
                    if i < index_po_splite:
                        self.__adresar[i] = -1
                        self.__hlbky_blokov[i] += 1
                    else:
                        self.__adresar[i] = adresa_bloku
                        self.__hlbky_blokov[i] += 1



    else:
        blok.pridaj_zaznam(vkladany_zaznam)
        self.__subor.seek(adresa_bloku)
        self.__subor.write(blok.na_byte_array())
        zaznam_je_vlozeny = True
```

- **Pridanie do existujúceho bloku**:Ak má blok voľné miesto, záznam sa pridá a blok sa uloží späť na disk.

Tieto procesy sa vykonávajú až dokým nie je záznam vložený.

### 4.2.2 Operácia nájdi

Operácia nájdi má nasledovný priebeh:

- **Výpočet indexu**: Na základe hash-u kľúča hľadaného záznamu sa vypočíta index v adresári a zistí sa adresa bloku, do ktorého by mal byť vložený, pokiaľ je táto adresa -1 , čiže k danému výsledku hešovacej funkcie nie je priradení žiadny existujúci blok, vráti None.
- **Vyhľadanie záznamu**: Na základe adresy bloku v adresári sa načíta príslušný blok a hľadaný záznam sa v ňom vyhľadá. Pokiaľ sa nájde, tento záznam sa vráti, inak sa vráti None.

```
def vrat_zaznam(self, hladany_zaznam: IHashData) -> IHashData:
    """
    Vráti záznam na základe záznamu, ktorý sa hľadá, pokiaľ sa nachádza v súbore.
    """
    hash_kluca = hladany_zaznam.get_hash()
    index_v_adresari = self.__get_index_v_adresari(hash_kluca, self.__hlbka)
    adresa_bloku = self.__adresar[index_v_adresari]
    if adresa_bloku != -1:
        blok = HashFileBlok(self.__velkost_bloku, self.__instancia_zaznamu)
        self.__subor.seek(adresa_bloku)
        blok.z_byte_array(self.__subor.read(self.__velkost_bloku))
        return blok.vrat_zaznam(hladany_zaznam)

    return None
```

# 5 Počet prístupov do súboru operácií implementovaných štruktúr

## 5.1 Operácie štruktúry Heapfile

### 5.1.1 Vlož záznam

V prípade operácie vlož závisí počet prístupov do súboru od konkrétnej situácie, ktorá nastane počas vkladania záznamu:

**Vkladanie záznamu do prázdneho súboru**

V tomto prípade je potrebné vytvorenie nového bloku, vloženie záznamu do bloku. V prípade voľných záznamov sa blok vloží do zreťazenia a **zapíše** sa do súboru

**Počet prístupov = 1 (W)**

**Vkladanie záznamu na koniec súboru**

V tomto prípade neexistujú čiastočne ani úplne voľné bloky, preto sa vytvorí nový blok, vloží sa do neho záznam a v prípade voľných záznamov sa pridá do zreťazenia. Následne sa **zapíše** na koniec súboru.

**Počet prístupov = 1 (W)**

**Vkladanie záznamu na základe čiastočne voľného zreťazenia**

V tomto prípade je uložený aspoň 1 blok a existuje čiastočne voľný blok, do ktorého sa vloží záznam prioritne. Najprv sa blok **načíta,** pridá sa do neho záznam. Pokiaľ sa vložením blok naplní, odstráni sa z čiastočného zreťazenia.

- Ak je v zreťazení iba jeden blok(ten, ktorý sa naplnil po vložení), nie sú potrebné ďalšie prístupy do súboru, adresa prvého bloku v zreťazení sa zruší.
- Ak je v zreťazení viacero blokov, **načíta** sa nasledujúci blok v zreťazení, upraví sa jeho predchodca a zároveň sa nastaví ako prvý blok v zreťazení, **zapíše** sa naspäť do súboru.

Pri všetkých prípadoch sa nakoniec blok, do ktorého sa vložil záznam **zapíše** do súboru.

## Počty prístupov:

- **Blok sa po vložení záznamu nenaplní = 2 = (R+W)**
- **Blok sa po vložení naplní – je jediný v zreťazení = 2 = (R+W)**
- **Blok sa po vložení naplní – je ich viacej v zreťazení = 4 = (2*R+2*W)**

**Vkladanie záznamu na základe úplne voľného zreťazenia**

Rovnako ako v prípade čiastočného zreťazenia sa daný blok najprv **načíta,** vloží sa do neho záznam a následne sa odstráni z úplne voľného zreťazenia (už nie je prázdny).

Pri odstránení môžu nastať analogicky dve rovnaké situácie ako pri čiastočnom zreťazení, že blok je jediný blok v zreťazení alebo je na začiatku a má aj nasledovníka.

Následne sa blok pridá do čiastočného zreťazenia.

- Ak v zreťazení ešte nie je žiadny blok, nie sú potrebné ďalšie prístupy do súboru.
- Ak je v zreťazení N blokov, postupne sa **N*načítajú** cez nasledovníkov až po posledný blok v zreťazení, tomu sa pridá vkladaný blok ako nasledovník a teraz už „predposledný“ blok v zreťazení sa **zapíše.**
- Ak nastane špeciálny prípad, kedy sa do bloku zmestí vždy iba jeden záznam, čiastočné zreťazenie sa nikdy nepoužije, v takomto prípade nie je pridanie do zreťazenia potrebné a tým pádom ani ďalšie prístupy do súboru.

Pri všetkých prípadoch sa nakoniec blok, do ktorého sa vložil záznam **zapíše** do súboru.

## Počty prístupov:

**Blok je jediný v úplnom zreťazení, v čiastočnom nie je žiadny blok = 2 = (R+W)**

**Blok je jediný v úplnom zreťazení, v čiastočnom je N blokov = 3+N = [R+(N*R)+2*W]**

**Blok má nasledovníka, v čiastočnom nie je žiadny blok = 4 = (2*R+2*W)**

**Blok má nasledovníka, v čiastočnom je N blokov = 5 + N = [2*R+W+(N*R)+2*W]**

**Blok je jediný v úplnom zreťazení, špeciálny prípad Max 1 záznam = 2 = (R+W)**

**Blok má nasledovníka, špeciálny prípad Max 1 záznam = 4 = (2*R+2*W)**

### 5.1.2 Nájdi záznam

V prípade operácie nájdi je počet prístupov do súboru fixný, keďže sa podľa zadanej adresy **načíta** požadovaný blok a z tohto bloku sa následne vráti záznam, pokiaľ sa nájde.

**Počet prístupov = 1 = (R)**

### 5.1.3 Aktualizuj záznam

V prípade operácie aktualizuj závisí počet prístupov do súboru od samotného úspechu operácie. Každopádne je potrebné takmer vždy **načítať** požadovaný blok, kde sa vykonáva zmena, v prípade, že bola zadaná platná adresa. Po načítaní bloku sa vykonáva výmena záznamu za nový – aktualizovaný záznam, v prípade, že aktualizácia prebehla úspešne je zmenený blok naspäť **zapísaný** do súboru, v opačnom prípade nie je zapisovanie potrebné, keďže aktualizácia neprešla.

**Počet prístupov = 2 = (R+W)**

### 5.1.4 Vymaž záznam

V prípade operácie vymaž závisí počet prístupov do súboru od konkrétnej situácie, ktorá nastane počas vymazávania záznamu:

**Blok sa po vymazaní záznamu stal prázdnym**

Blok, z ktorého sa vymazáva sa najprv **načíta** a daný záznam sa vymaže, blok sa stane prázdnym.

- Ak je daný blok posledný v súbore, vymaže sa z čiastočného zreťazenia. Pri odstránení môžu nastať analogicky dve rovnaké situácie ako predtým + situácia, kedy je blok zreťazení medzi dvoma blokmi (je potrebné **načítať** predka aj nasledovníka bloku, vzájomne ich prepojiť a naspäť **zapísať**). Následne sa musí skontrolovať, či pred posledným blokom (ktorý je prázdny) sa nenachádza **N prázdnych blokov**, po ktoré je potrebné súbor **skrátiť**. Preto sa postupne **N+1*načíta** predchádzajúci blok, v prípade že sa nájde neprázdny blok alebo prehľadávanie objaví prázdne bloky až po začiatok súboru, prehľadávanie skončí. Počas prehľadávania sa v prípade prázdneho bloku ešte vymaže z úplného zreťazenia, toto sa už ale **nevykoná** pokiaľ je blok neprázdny, alebo prehľadávanie príde až na začiatok súboru. Čiže v konečnom dôsledku sa ešte **N*odstráni blok z úplného zreťazenia** (keďže sa budú skracovať), **toto ale nenastane pokiaľ sa súbor skráti až na začiatok** (úspora prístupov do súboru). Následne sa ako posledné súbor **skráti** na danú dĺžku.

- Ak daný blok nie je posledný v súbore, vymaže sa z čiastočného zreťazenia a následne sa pridá do úplného zreťazenia, pričom oba tieto operácie majú situácie analogické tým predošlým. Následne sa ešte blok **zapíše**.
- Taktiež treba myslieť na situáciu, kedy je maximálny počet záznamov v bloku = 1, v tom prípade sa **čiastočné zreťazenie nepoužíva**.

## Počty prístupov:

**Blok je posledný, prvý v čiastočnom zreťazení bez nasledovníka, prázdne bloky sú až po začiatok súboru**

**= 2 + (N+1)= (R+T) + (N+1*R)**

**Blok je posledný, prvý v čiastočnom zreťazení bez nasledovníka, prázdne bloky nie sú až po začiatok súboru**

**= Min 2+(N+1) a Max 2+(N+1)+ (N-1*2R2W)= (R+T) +[(N+1*R)+(N*?)]**

**Blok je posledný, prvý v čiastočnom zreťazení s nasledovníkom, prázdne bloky nie sú až po začiatok súboru**

**= Min 4+(N+1) a Max 4+(N+1)+ (N-1*2R2W)= (R+T) +(R+W)+[(N+1*R)+(N*?)]**

**Blok je posledný, posledný v čiastočnom zreťazení, prázdne bloky nie sú až po začiatok súboru**

**= Min 4+(N+1) a Max 4+(N+1)+ (N-1*2R2W)= (R+T) +(R+W)+[(N+1*R)+(N*?)]**

**Blok je posledný, zreťazení medzi dvoma blokmi, prázdne bloky nie sú až po začiatok súboru**

**=Min6+(N+1) a Max 6+(N+1)+ (N-1*2R2W)=(R+T) +(2*R+2*W)+[(N+1*R)+(N*?)]**

**Blok je posledný, Max 1 záznam, prázdne bloky sú až po začiatok súboru**

**= 2 + (N+1) + = R + (N+1*R) + T**

**Blok je posledný, Max 1 záznam, prázdne bloky nie sú až po začiatok súboru**

**= Min 2 + (N+1) a Max 2 + (N+1) + (N-1* 2R2W) = (R+T) + [(N+1*R) + (N*?)]**

**Blok nie je posledný, prvý v čiastočnom zreťazení s nasledovníkom, v úplnom nie je žiadny blok**

**= 4 = (R+W) + (R+W)**

**Blok nie je posledný, prvý v čiastočnom zreťazení s nasledovníkom, v úplnom je M blokov**

**= 5 + M = (R+W) + (R+W) + [(M*R)+W]**

**Blok nie je posledný, prvý v čiastočnom zreťazení bez nasledovníka, v úplnom nie je žiadny blok**

**= 2 =(R+W)**

**Blok nie je posledný, prvý v čiastočnom zreťazení bez nasledovníka, v úplnom je M blokov**

**= 3 + M= (R+W) + [(M*R)+W]**

**Blok nie je posledný, posledný v čiastočnom zreťazení, v úplnom nie je žiadny blok**

**= 4 = (R+W) + (R + W)**

**Blok nie je posledný, posledný v čiastočnom zreťazení, v úplnom je M blokov**

**= 5 + M = (R+W) + (R+W) + [(M*R)+W]**

**Blok nie je posledný, zreťazení medzi dvoma blokmi, v úplnom nie je žiadny blok**

**= 6 = (R+W) + (2*R + 2*W)**

**Blok nie je posledný, zreťazení medzi dvoma blokmi, v úplnom je M blokov**

**= 7 + M = (R+W) + (2*R + 2*W) + [(M*R)+W]**

**Blok sa po vymazaní záznamu stal čiastočne prázdnym**

Blok, z ktorého sa vymazáva sa najprv **načíta** a daný záznam sa vymaže, blok sa stane čiastočne prázdnym.

- Ak bol daný blok predtým plný (stal sa práve čiastočne prázdnym), pridá sa do čiastočného zreťazenia, kde môžu nastať analogické situácie počas vkladania do zreťazenia.
- Taktiež treba myslieť na situáciu, , kedy je maximálny počet záznamov v bloku = 1, v tom prípade sa **čiastočné zreťazenie nepoužíva**.

**Počty prístupov**

**Ak bol predtým plný, v čiastočnom nie je žiadny blok = 2 = (R+W)**

**Ak bol predtým plný, v čiastočnom je N blokov = 3 + N = (R+W) + (N*R + W)**

**Ak bol predtým plný, Max 1 záznam = 2 = (R+W)**

**Ak bol predtým čiastočne prázdny = 2 = (R+W)**

## 5.2 Operácie štruktúry Hashfile

### 5.2.1 Vlož záznam

V prípade operácie vlož závisí počet prístupov do súboru od konkrétnej situácie, ktorá nastane počas vkladania záznamu:

**Vkladanie záznamu do prázdneho miesta v adresári**

V tomto prípade nie je v adresári poznačená žiadna adresa už alokovaného bloku, ktorý by prislúchal pre daný výsledok hashovacej funkcie. V takomto prípade je potrebné alokovať nový blok, do tohto bloku vkladaný záznam vložiť a ako posledné je daný blok potrebné **zapísať**.

**Počet prístupov = 1 (W)**

**Vkladanie záznamu do bloku, ktorý má voľné miesto**

V tomto prípade, je pre daný výsledok hešovacej funkcie poznačená v adresári adresa už alokovaného bloku, tento blok je preto potrebné **načítať** a v prípade, že daný blok nebude plný, vkladaný záznam sa do neho vloží a upravený blok sa následne znovu **zapíše** do súboru na adrese, ktorá bola poznačená v adresári.

**Počet prístupov = 2 (R+W)**

**Vkladanie záznamu do bloku, ktorý je už plný**

V tomto prípade, je analogicky ako v predošlom prípade v adresári už poznačená adresa pre daný výsledok hešovacej funkcie. Tento blok je opäť potrebné **načítať** a v prípade, že je daný blok už plný, nasleduje buď proces zdvojenia adresára alebo proces splitovania – snaha o rozdelenie bloku na dva a zároveň uvoľnenie miesta pre vkladaný prvok. Počas procesu zdvojenia adresára sa nevykonáva **žiadny prístup do súboru.**. Následne sa vytvára dočasný nový blok, ktorý môže byť ale aj nemusí byť po dokončení splitu **zapísaný** ako nový blok do súboru, v závislosti od toho, či do neho boli presunuté všetky záznamy z pôvodného bloku alebo iba časť týchto záznamov a zvyšné ostali v pôvodnom bloku. V prípade, že sa z pôvodného bloku presunuli všetky jeho záznamy, iba do jedného z dvoch vytvorených blokov, nevyžaduje táto operácia splitu **žiadny dodatočný prístup do súboru**, keďže sú záznamy stále fyzicky zapísané iba v jednom bloku a to v pôvodnom, preto stačí iba aktualizovať adresár tak, aby obsahoval adresu pôvodného bloku na správnych indexoch v adresári. V prípade, že sa záznamy rozdelili do oboch z napĺňaných blokov, je potrebné **zapísať obe** tieto bloky.

Tieto dva procesy (prípadné zdvojnásobenie adresára a split) sa následne vykonávajú, dokým nie je uvoľnené miesto pre nový vkladaný záznam. Preto je pri **N** opakovaných operáciách splitovania potrebné **N * načítať** a prerozdeliť splitovaný blok. V prípade, že sa v **M** operáciách splitovania rozdelia záznamy medzi oba vytvorené bloky bude potrebné aj **M * zapísanie oboch blokov**.

**Počet prístupov = 2 + N + (M*2) = R + (N*R) + (M *2W) + W**

### 5.2.2 Nájdi záznam

V prípade operácie nájdi je počet prístupov, podobne ako pri štruktúre heapfile, do súboru fixný, keďže sa podľa zadaného záznamu a jeho kľúča **načíta** požadovaný blok, v ktorom by sa mal záznam s daným kľúčom nachádzať. Z tohto bloku sa následne vráti požadovaný záznam, pokiaľ sa nájde.

**Počet prístupov = 1 = (R)**

# 6 Záver

V tejto semestrálnej práci sa mi podarilo úspešne implementovať neutriedený súbor dát na disku a rozšíriteľné hešovanie, ktoré tvoria základ systému na evidenciu návštev autoservisu. V prípade rozšíriteľného hešovania nepodporujem operáciu vymazávania a aktualizácie vložených záznamov, čo poskytuje možné rozšírenie do budúcnosti.

Na základe tejto implementácie som vytvoril aplikáciu, ktorá umožňuje používateľom pridávať, vyhľadávať a upravovať záznamy o návštevách na základe jedinečného ID alebo ECV vozidla. Zároveň aplikácia podporuje ukladanie údajov do súboru, čím zabezpečuje ich trvalé uchovanie a umožňuje pokračovanie v práci s dátami po reštarte systému.