Pedagogická fakulta
Faculty of Education

Jihočeská univerzita v Českých Budějovicích
University of South Bohemia in České Budějovice

Jihočeská univerzita v ČB

Pedagogická fakulta

Katedra informatiky

Interaktivní datové struktury
Interactive Data Structures

Bakalářská práce

Vypracoval: Jan Pršala

Vedoucí práce: RNDr. Hana Havelková

České Budějovice

2019

Prohlášení

Prohlašuji, že svoji bakalářskou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

V Českých Budějovicích dne 22. dubna 2019.

....

Jan Pršala

Anotace

Cílem této práce je vytvoření jednoduché aplikace pro výuku datových struktur - převážně se zaměřuje na datové struktury frontu, zásobník, spojový seznam, binární vyhledávací strom a hash tabulku.

Každá datová struktura je obsažena ve vlastním modulu a je předvedena jak na příkladu z reálného života, tak i v podobě abstraktního znázornění, ve kterém může uživatel danou strukturu sám modelovat.

V samotné práci jsou výše vypsané datové struktury popsány a vysvětleny. Kromě toho je v ní i popsán způsob vytvoření nového modulu.

Na přiloženém CD se nachází jak samotná aplikace, včetně zdrojového kódu, tak i řešený příklad implementace každé z datových struktur.

Klíčová slova

datová struktura, Java, fronta, zásobník, spojový seznam, binární vyhledávací strom, hash tabulka

Abstract

The goal of this thesis is creating simple application for teaching dats structures - mostly focusing on data structures queue, stack, linked list, binary search tree and hash table.

Each data structure is contained in its own modul and is shown on an example from the real world and in an abstract example, which can be modified by the user.

This thesis itself contains the description of the data structures mentioned above. Instructions on how to add more modules are also there.

The application itself with all the source code and the example of implementation of each data structure is located on the enclosed CD.

Keywords

data structures, Java, queue, stack, linked list, binary search tree, hash table

Poděkování

Rád bych poděkoval RNDr. Haně Havelkové za ochotu při vedení mé bakalářské práce, za cenné rady, nápady a připomínky. Dále bych chtěl poděkovat všem, kteří se podíleli na testování aplikace, jmenovitě studentkám Pedagogické fakulty Jihočeské univerzity Denise Janů a Denise Šafářové a studentu Ondřeji Wyrwovi z Obchodně podnikatelské fakulty Slezské univerzity v Karviné.

Obsah

1 Úvod

Datové struktury jsou téma informatiky, se kterým jsem měl já i mnoho mých spolužáků problémy. Bylo pro nás těžké si některé ze složitějších struktur, jako je například spojový seznam nebo hash tabulka, představit, natož implementovat vlastní verzi.

Téma jsem si vybral právě z důvodu toto změnit. Věřím, že vytvořená aplikace půjde do výuky zakomponovat, převážně jako podpůrná vizualizace dané datové struktury. Tato práce se ale v žádném případě nesnaží nahradit dosavadní skripta a učebnice používané na školách. Naopak z nich čerpá a předpokládá, že uživatel je již s problematikou datových struktur seznámen. Pokud by čtenář chtěl prohloubit svoje teoretické znalosti těchto struktur, na konci práce se nachází seznam literatury, ze které jsem čerpal.

1.1 Cíle práce

Mým cílem bylo navrhnout a vytvořit jednoduchou aplikaci, která čtenáři poskytne snadnou cestu k vizualizaci a manipulování s vybranými datovými strukturami. Aplikace je naprogramována v Javě a je určena pro stolní počítače. Během vytváření jsem si pohrával s myšlenkou vytvoření i verze pro mobilní zařízení, protože ta jsou více aktuální, ale z časových důvodů jsem tento nápad zavrhl. Aplikaci jsem také navrhoval s ohledem na případné přidání dalších datových struktur. Každé datová struktura je obsažena v jednom modulu. Ty se mohou přidávat i odebírat dle libosti a bez vlivu na chod aplikace.

Dalším mým cílem bylo shrnout teoretické poznatky vybraných datových struktur. Snažil jsem být co nejvíce objektivní při porovnávání různých implementací, aby se mohl případně čtenář sám rozhodnout, kterou použije.

1.2 Metodika práce

Nejdříve jsem si nastudoval odbornou literaturu zabývající se problematikou datových struktur.

Následně jsem se musel rozhodnout, v jakém programovacím jazyce vytvořím aplikaci. Původně jsem chtěl použít Java Applet, ale ten se hodí spíše pro webové applety. Poté jsem zkoušel Java 2D, kde jsem měl problém s dosažením kýžené interaktivity mezi jednotlivými objekty na obrazovce. Proto jsem nakonec zvolil framework LibGDX, který je určen pro vývoj her. Má tedy už vyřešenou interakci mezi objekty a další potřebné operace, jako je mazání, přehrávání animací nebo měnění textur. Kromě toho s ním mám už zkušenosti a vím, jak s ním pracovat. Vzhledem k tomu, že nejsem grafik a s tvořením vlastní grafiky mám minimální zkušenosti, musel jsem si na internetu najít volně použitelnou grafiku. Tu jsem pak použil v příkladech z reálného života u všech struktur. Grafiku pro volné modelování jsem si vytvářel sám.

Aplikaci jsem navrhl následovně: Každá datová struktura je obsažena ve vlastním modulu a je možné další moduly přidávat. V každém modulu jsou pak dva příklady. První je příkladem z reálného života, na kterém je uživateli předvedeno fungování zvolené struktury. V druhém příkladu je uživateli umožněno tuto strukturu samostatně vytvořit a aplikace ho upozorní, pokud by postupoval chybně.

Nakonec jsem sepsal teoretické poznatky k probíraným datovým strukturám a vytvořil návod pro přidání nového modulu do aplikace.

2 Datové Struktury

Co to jsou datové struktury? Podle Oxfordského slovníku se jedná o formu, jakou je uspořádána určitá skupina dat tak, aby k nim byl snadný přístup a šlo s nimi efektivně manipulovat[1].

V této části budou postupně popsány vybrané datové struktury - fronta, zásobník, spojový seznam, binární vyhledávací strom a hashovací tabulka.

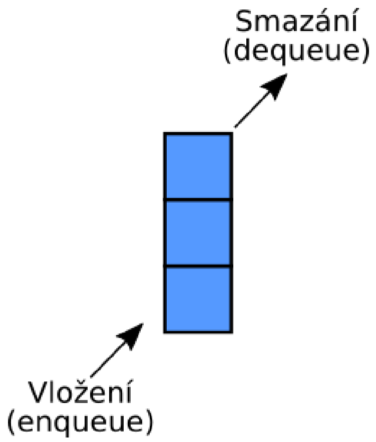
2.1 Fronta (Queue)

2.1.1 Definice

Fronta je jednou ze základních datových struktur. Řadí se mezi abstraktní datové struktury.

Slouží k ukládání dat principem FIFO (zkratka pro anglické sousloví First in, first out, česky první dovnitř, první ven). To znamená, že data se z fronty vybírají od nejstarších po nejmladší. Tím pádem může program pracovat jen s prvkem na prvním místě ve frontě.

S frontou se v reálném životě setkáváme každý den. Různé fronty v obchodech nebo na toalety jsou naprosto stejné jako vyjádření fronty jako datové struktury. V přiloženém programu je fronta předvedena na příkladu fronty ve školní jídelně.



Obrázek 1: Fronta

Speciálním případem fronty je tzv. prioritní fronta. V ní má každý prvek přiřazenou prioritu a prvky s větší prioritou se mohou zařadit před prvky s nižší prioritou.

Dalším speciálním případem fronty je cyklická fronta. Koncový prvek v této frontě odkazuje zpátky na první. Data jsou tedy uloženy v imaginárním kruhu, který se zvětšuje a zmenšuje s množstvím dat.

2.1.2 Metody

Fronta disponuje standardními metodami jako jsou metody pro vložení (enqueue) a odebrání (dequeue) prvku, zjištění, jaký prvek stojí ve frontě na začátku (peek) a velikost fronty (size)[2].

Mezi další metody, které může fronta obsahovat sama o sobě jsou například metody pro vyhledávání určitých prvků, výpis prvků nebo zjištění, zda je fronta prázdná.

2.1.3 Implementace

Fronta bývá nejčastěji v programu implementována jako pole nebo spojový seznam (který je také datovou strukturou a je popsán v samostatné kapitole).

Fronta implementovaná jako pole má mnoho výhod, ale i problémů. Vzhledem k tomu, že pole má předdefinovanou velikost, musí se vždy, když se dosáhne maximální kapacity, vytvořit nové větší pole a všechny dosavadní prvky do něj přesunout. Při mazání prvků se maže vždy jen první prvek z pole, proto je nejvýhodnější projít celé pole jednoduchým cyklem a přeuložit všechny prvky na předchozí pozici. Druhý prvek skočí na první pozici, třetí prvek na druhou a tímto způsobem se pokračuje až na konec pole. Velikost takto implementované fronty je stejná jako počet zaplněných políček pole a výpis probíhá přes jednoduchý cyklus procházející pole od začátku do konce.

S frontou implementovanou jako spojový seznam se pracuje lépe. Místo pole se na začátku inicializují proměnné určující počáteční a koncový prvek.

Je také potřeba vytvořit odkazy na následující prvky každého prvku. Při přidávání prvků do spojového seznamu se pak propojí nový prvek se současným konečným prvkem a z přidávaného prvku se stane nový koncový prvek. Při mazání se první prvek smaže tím způsobem, že se do proměnné počátečního prvku uloží druhý prvek. Pro zjištění velikosti a výpisu se musí celý spojový seznam projít od začátku do konce a velikost se musí uložit do speciální celočíselné proměnné. Alternativou je využít celočíselnou instanční proměnnou, která se bude měnit vždy při přidání nebo smazání prvku.

Při implementování prioritní fronty se musí kontrolovat hodnota priority při vkládání každého prvku. V případě verze s polem se musí v poli najít správná pozice, posunout všechny prvky za touto pozicí a nakonec na ni vložit nový prvek. Verze se spojovým seznamem je znovu mnohem méně náročná. V ní stačí najít správné místo a pak jen změnit odkazy na následující prvky. Pokud by více prvků mělo stejnou prioritu, uloží se za sebe v pořadí, ve kterém byly přidány.

2.1.4 Ukázka kódu

V této ukázce je fronta implementována jako spojový seznam, do kterého se ukládají instance třídy Prvek. Tyto prvky mají instanční proměnnou pro následující prvek. Ta je při vytvoření prvku nastavena na null.

Samotná fronta má dvě proměnné typu Prvek - *prvni* a *posledni*. Hodnota obou je při vytváření fronty nastavena na null, symbolizující prázdnost fronty.

Metoda pridej(Prvek pridavany) pak přidá nový prvek typu Prvek do fronty. Pokud je fronta před přidáním prázdná, tak je přidávaný prvek uložen do proměnných *prvni* i *posledni*. Při přidání dalšího prvku se z koncového prvku vytvoří odkaz na nový prvek a tímto prvkem se následně přepíše proměnná *posledni*.

```
public void pridej(Prvek pridavany){
    if(pridavany == null){
        return;
    }
   pridavany.setNasledujici(null);
   if(jePrazdna()){
      prvni = pridavany;
   } else{
      posledni.setNasledujici(pridavany);
   }
   posledni = pridavany;
}
```

Zdrojový kód 1: Fronta - vložení prvku

Smazání prvku je jednodušší proces. K mazání slouží metoda smaz(). Ta přiřadí status hlavičky prvku následujícího po současné hlavičce.

```
public void smaz(){
    if(jePrazdna()){
        return;
    }
    prvni = prvni.getNasledujici();
    if(jePrazdna()){
      posledni = null;
    }
}
```

Zdrojový kód 2: Fronta - smazání prvku

Přidáním celočíselné proměnné do třídy Prvek lze frontu lehce upravit na prioritní frontu.

Vložení nyní probíhá tak, že se najde poslední prvek se stejnou prioritou jako nový prvek a nový prvek se vloží za něj. Na ostatních metodách se nic nemění, ale je možné přidat nové metody, které například vrátí počet prvků s danou prioritou.

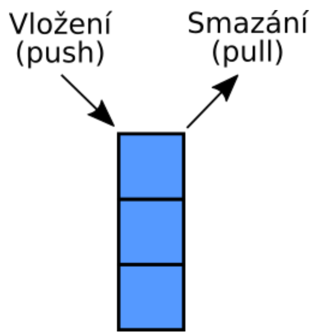
2.2 Zásobník (Stack)

2.2.1 Definice

Zásobník se v mnoha ohledech podobá frontě, ale zásadně se od ní liší přístupem k datům. Stejně jako fronta je to abstraktní datová struktura.

Data se do zásobníku ukládají způsobem LIFO (zkratka pro anglické sousloví Last in, first out, česky poslední dovnitř, první ven). To znamená, že nejnovější data jsou zpracována jako první. Tím se liší od fronty, která data ukládá způsobem FIFO.

Poslednímu přidanému prvku se říká vrchol.



Obrázek 2: Zásobník

Zásobník už se v reálném životě hledá hůř než fronta. Jako první všechny určitě napadne zásobník nábojů, ale dalším příkladem, se kterým se člověk setká téměř každý den je komínek špinavých talířů. Přesně takto je zásobník předveden i v přiloženém programu.

2.2.2 Metody

Nejzákladnější metody, které zásobník musí obsahovat, jsou: vložení (push) a smazání (pop) prvku, zjištění horního prvku (top) a počet prvků uvnitř zásobníku (size)[3].

Další operace, které může zásobník obsahovat patří například i metoda prázdnosti zásobníku, metoda pro vyhledávání uvnitř zásobníku nebo metoda, která smaže několik prvků najednou (tzv. Multipop).

2.2.3 Implementace

Zásobník se nejlépe implementuje jako pole nebo spojový seznam.

Pokud je zásobník implementovaný jako pole, má podobné problémy jako fronta implementovaná jako pole. Pokud se předem inicializované pole naplní, musí se vytvořit nové větší pole a data do něj přesunout. Může se stát, že pole bude zbytečně veliké a nikdy se nenaplní, čímž bude zbytečně zabírat místo v paměti. Při mazání se vybere poslední prvek z pole, smaže se, ale zůstane po něm prázdné místo v poli. Proto je potřeba plánovat dopředu a vytvořit optimálně velké pole pro řešený problém. Velikost je stejná jako počet zaplněných políček pole.

Při implementování zásobníku jako spojového seznamu se ušetří čas na vytváření polí. Jediné proměnné, které jsou v tomto případě potřeba jsou proměnné pro počáteční a koncový prvek. Z prvního prvku v zásobníku existuje odkaz na další prvky. Při mazání se tento odkaz ruší. Pro zjištění velikosti takového zásobníku je potřeba ho projít cyklem a každý prvek započítat, nejlépe v samostatné metodě. Nebo je možné použít speciální celočíselnou instanční proměnnou a velikost měnit při vkládání/mazání prvků.

2.2.4 Ukázka kódu

Následující zásobník je implementován jako spojový seznam pracující s instancemi třídy Prvek. Na začátku jsou dvě proměnné *prvni* a *posledni*, obě typu Prvek. Při inicializaci zásobníku jsou nastaveny na hodnotu null symbolizující prázdnost zásobníku.

Metoda pridej(Prvek pridavany) přidává instanci třídy Prvek do zásobníku. Pokud je zásobník prázdný, uloží se tento prvek do proměnných *prvni* i *posledni*. Jinak se vytvoří odkaz z přidávaného prvku na první prvek.

```
public void pridej(Prvek pridavany){
     if(pridavany == null){
         return;
     }
    pridavany.setNasledujici(null);
    if(jePrazdny()){
       posledni = pridavany;
    } else{
       pridavany.setNasledujici(prvni);
    }
    prvni = pridavany;
}
```

Zdrojový kód 3: Zásobník - vložení prvku

Pro mazání posledního přidaného prvku se používá metoda smaz(). Maže poslední přidaný, protože zásobník je datová struktura typu LIFO. Vzhledem k tomu, že nejmladší prvek je na vrcholu zásobníku, stačí jen posunout proměnnou *prvni* na následující prvek.

```
public void smaz(){
     if(jePrazdny()){
         return;
     }
     prvni = prvni.getNasledujici();
     if(jePrazdny()){
       posledni = null;
     }
}
```

Zdrojový kód 4: Zásobník - smazání prvku

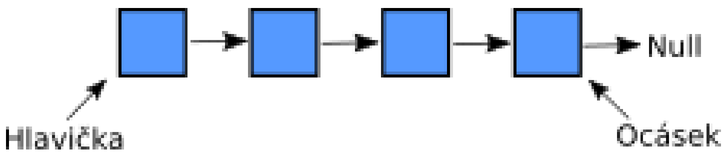
2.3 Spojový seznam (Linked list)

2.3.1 Definice

Spojový seznam, občas též nazývaný lineární seznam, patří mezi abstraktní datové struktury.

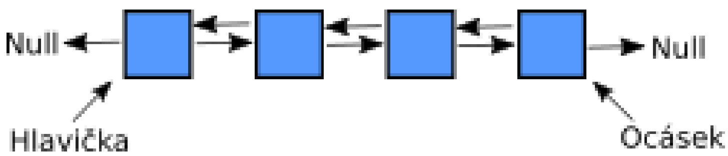
Často bývá přirovnáván k poli. Od něj se ale liší tím, že nemá předem stanovenou délku a není možné přímo přistupovat ke všem prvkům. Proto je výhodnější ho použít v případech, kdy není výsledná velikost známá, jako například při implementaci fronty a zásobníku. Nevýhodou ale je, že nejsou dostupné všechny prvky, ale jen první, popřípadě i poslední prvek a k dalším prvkům se musí dostat procházením celého seznamu.

Základní jednotkou spojového seznamu je uzel, ve kterém jsou vždy uložena data a odkaz na další prvek. Prvnímu uzlu se říká hlavička a poslednímu ocásek.



Obrázek 3: Jednosměrný spojový seznam

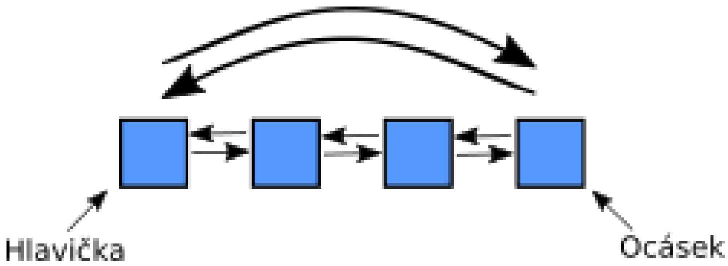
Existuje několik variant spojového seznamu. Nejčastější je jednoduchý jednosměrný zřetězený seznam. V něm každý uzel obsahuje odkaz jen na následující prvek a poslední prvek odkazuje na null.



Obrázek 4: Obousměrný spojový seznam

Dalším typem spojového seznamu je obousměrný zřetězený seznam. V něm každý uzel obsahuje krom odkazu na další prvek odkaz i na prvek předcházející. První i poslední prvek odkazují, kromě následující, respektive předchozího prvku, i na null. V tomto případě se může seznam procházet i odzadu, což může značně urychlit vyhledávání uvnitř spojového seznamu.

Pokud z posledního prvku vede odkaz zpátky na první prvek, vzniká cyklický (kruhový) spojový seznam.



Obrázek 5: Cyklický spojový seznam

V reálném životě odpovídá spojovému seznamu vlaková souprava. Lokomotiva symbolizuje hlavičku seznamu a vagony jsou jednotlivé uzly. Tento případ pak zpracovává přiložený program v podobě obousměrného spojového seznamu.

2.3.2 Metody

Všechny verze spojového seznamu obsahují následující operace: vložení (insert) na začátek, libovolnou pozici nebo konec, smazání (delete) libovolného prvku, zjištěním jaký prvek je na určité pozici (read) a zjištění velikosti (size).

2.3.3 Implementace

Implementace spojového seznamu probíhá přes vlastní třídu, která pracuje s prvky jiné třídy. Tyto prvky musí obsahovat informace o následujícím (a v případě obousměrného spojového seznamu i o předchozím) prvku.

Samotná třída spojového seznamu musí mít všechny již zmíněné metody.

2.3.4 Ukázka kódu

Následující ukázka obsahuje dva spojové seznamy. První je jednosměrný a druhý obousměrný. Oba pracují s instancemi třídy Prvek. Tyto prvky mají instanční proměnné *predchozi* a *nasledujici*, obě typu Prvek, které jsou při vytvoření nastaveny na null. Jednosměrný seznam využívá jen proměnné *predchozi*, zatímco obousměrný využívá obě instanční proměnné. Při inicializaci se musí jediné dvě proměnné typu Prvek *prvni* (hlavičku seznamu) a *posledni* (ocásek) nastavit na hodnotu null. To značí prázdnost spojového seznamu.

Metoda pridejNaZacatek(Prvek pridavany) přidá prvek na první místo spojového seznamu a udělá z něj tedy novou hlavičku. Pokud už jsou v seznamu prvky, vytvoří odkaz z přidávaného prvku na současnou hlavičku a následně přeuloží hodnotu hlavičky na nový prvek.

```
public void pridejNaZacatek(Prvek pridavany){
    if(pridavany == null){ return; }
    pridavany.setNasledujici(null);
    if(jePrazdny()){
        posledni = pridavany;
    } else{
       pridavany.setNasledujici(prvni);
    }
    prvni = pridavany;
}
```

Zdrojový kód 5: Jednosměrný spojový seznam - přidání na začátek

V obousměrném seznamu dojde v této metodě k jediné změně. Změna spočívá ve vytvoření zpätečního odkazu ze současné hlavičky na přidávaný prvek.

```
public void pridejNaZacatek(Prvek pridavany){
    if(pridavany == null){ return; }
    pridavany.setNasledujici(null);
    pridavany.setPredchozi(null);
    if(jePrazdny()){
        posledni = pridavany;
    } else{
      prvni.setPredchozi(pridavany);
      pridavany.setNasledujici(prvni);
    }
    prvni = pridavany;
}
```

Zdrojový kód 6: Obousměrný spojový seznam - přidání na začátek

Metoda smazPosledni() smaže ocásek. V jednosměrném spojovém seznamu se musí procházet celý spojový seznam až k předposlednímu prvku.

```
public void smazPosledni(){
    Prvek docasny = prvni;
    if(docasny == null){ return; }
   if(docasny.getNasledujici() == null){
      prvni = null;
      posledni = null;
      return;
   }
    while(docasny.getNasledujici() != posledni){
        docasny = docasny.getNasledujici();
    }
    docasny.setNasledujici(null);
    posledni = docasny;
}
```

Zdrojový kód 7: Jednosměrný spojový seznam - smazání posledního prvku

V obousměrném seznamu je tato operace jednodušší. Díky zpětným odkazům se nemusí celý seznam procházet, ale stačí posunout ocásek o jeden prvek zpátky.

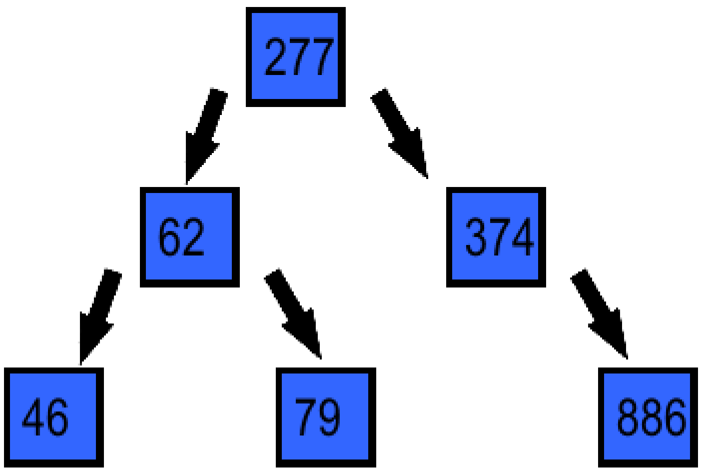
```
public void smazPosledni(){
   if(jePrazdny()){ return; }
    posledni = posledni.getPredchozi();
    if(posledni == null){
      prvni = null;
        return;
    }
    posledni.setNasledujici(null);
}
```

Zdrojový kód 8: Obousměrný spojový seznam - smazání posledního prvku

2.4 Binární vyhledávací strom (Binary Search Tree)

2.4.1 Definice

Binární vyhledávací strom je datová struktura, která slouží k ukládání porovnatelných dat (například čísel). Jedná se o hierarchickou strukturu. Z jednoho vrcholu, nazývaného kořen, vede cesta do dalších vrcholů. Každý vrchol (otec) má nejvýše 2 potomky. Pokud vrchol nemá potomka, nazývá se list. Jedná se o dynamickou datovou strukturu[4].



Obrázek 6: Binární vyhledávací strom

Potomci se dělí na levé a pravé. Z každého otce vede odkaz maximálně na jednoho levého a pravého potomka. Potomci nalevo od otce mají menší hodnotu a napravo od otce větší hodnotu než on.

Důležitým pojmem je také hloubka stromu. Pod tou se rozumí vzdálenost vrcholu od kořene. Kořen má tedy hloubku 0, jeho potomci hloubku 1, atd. Pokud mají všechny listy stejnou hloubku, jedná se o vyvážený binární strom.

Po odstranění libovolného prvku strom zůstane stromem.

Binární vyhledávací strom se v reálném životě hledá hůře, než předchozí struktury. Nejlepší příklad je vytvoření tzv. rozhodovacího stromu, kde je každý vrchol otázka, na kterou lze odpovědět jen ano nebo ne.

2.4.2 Metody

Nejzákladnější metody, které by měl každý binární vyhledávací strom obsahovat patří metody pro přidání nového vrcholu a odstranění libovolného vrcholu. Zatímco přidání je jednoduchá operace, která jen hledá správné místo uvnitř pro přidávaný prvek uvnitř stromu, u mazání je potřeba kontrolovat vyváženost stromu. List se může smazat bez problémů, ale pokud by se jednalo o otce s jedním potomkem, nahradí tento potomek svého otce. V případě, že má vrchol dva potomky, jeden z nich nahradí smazaný prvek.

Další častá metoda je metoda pro vyhledávání uvnitř stromu, zjištění hloubky nebo velikosti stromu.

2.4.3 Implementace

Binární vyhledávací strom se dá implementovat mnoha způsoby.

První způsob je přes jednoduché pole. Kořen je na pozici 0, jeho potomci na pozicích 1 a 2. Potomci prvku na pozici 1 jsou na pozicích 3 a 4 a potomci prvku na pozici 2 jsou na pozicích 5 a 6. Obecně se dá říct, že potomci prvku na pozici p jsou na pozicích *(2*p)+1* a *(2*p)+2*. Znovu zde nastává problém s použitím konečného pole pro teoreticky nekonečnou strukturu. Pokud bude pole plné, musí se vytvořit alespoň o 3 prvky větší pole. Velkou nevýhodou je, že pokud se budou přidávat prvky nevyváženě, může se skončit s velkým poloprázdným polem. Pokud by se například přidaly následující prvky v tomto pořadí: 5, 7, 11 a 13, všechny prvky se přidají do pravé větve a celá levá větev zůstane prázdná. Výhodou je přístup ke všem prvkům bez nutnosti procházet celý strom od kořene.

Lepší varianta je implementace přes speciální třídu podobnou spojovému seznamu. Znovu je potřeba samostatná třída pro prvky. V ní se nyní místo místo předchozího a následujícího prvku hlídají prvky v levé a pravé větvi. Podobně jako u spojového seznamu není potřeba řešit kapacitu. Nevýhodou je nutnost procházet celý strom při vyhledávání.

2.4.4 Ukázka kódu

Následující binární vyhledávací strom pracuje s instancemi třídy Vrchol. Každý tento vrchol má instanční proměnné pro pravého a levého potomka. Tito potomci jsou při vytvoření nastaveni na null. Strom má jedinou instanční proměnnou - *koren*. Všechny zmíněné metody využívají toho, že každý podstrom binárního vyhledávací stromu je také binární vyhledávací strom. To dovoluje naprogramovat metody pro vložení a smazání objektu rekurzivně.

```
private void pridejVrchol(Vrchol pridavany, Vrchol otec){
    if(pridavany == null || this.obsahuje(pridavany))
    { return; }
    if(otec == null){
        pridavany.setLeva(null);
        pridavany.setPrava(null);
        otec = pridavany;
        return; }
    if(pridavany.getHodnota() < otec.getHodnota()){
        if(otec.getLeva() == null){
            pridavany.setLeva(null);
            pridavany.setPrava(null);
            otec.setLeva(pridavany);
            return; }
        pridejVrchol(pridavany, otec.getLeva());
    } else if(pridavany.getHodnota() > otec.getHodnota()){
        if(otec.getPrava() == null){
            pridavany.setLeva(null);
            pridavany.setPrava(null);
            otec.setPrava(pridavany);
            return; }
        pridejVrchol(pridavany, otec.getPrava()); }}
```

Zdrojový kód 9: Binární vyhledávací strom - soukromá metoda pridejVrchol()

Tato soukromá metoda spravuje samotné přidávání nového vrcholu. Podle hodnoty tohoto vrcholu se pak ze celého stromu vybere odpovídající podstrom (buď levá nebo pravá strana stromu) a metoda se spustí znovu, tentokrát jen pro tento podstrom.

K běžnému používání je pak určena stejně pojmenovaná, ale tentokrát veřejná metoda. Ta jen zavolá soukromou verzi a předloží jí přidávaný objekt a kořen stromu jako argumenty.

```
public void pridejVrchol(Vrchol pridavany){
    pridejVrchol(pridavany, koren);
}
```

Zdrojový kód 10: Binární vyhledávací strom - veřejná metoda pridejVrchol()

2.5 Hashovací tabulka (Hash Table)

2.5.1 Definice

Hashovací (čte se hašovací) tabulka patří mezi vyhledávací datové struktury. Z každé hodnoty přidané do tabulky se vypočte pomocí hashovací funkce tzv. hash. Zbytek po dělení tohoto hashe celkovou kapacitou tabulky se najde index, do kterého se hodnota uloží[5].

Pozice	Hodnota	Hash kód
1	Pán Prstenů	6489591
2	Plavec	54152
3	--	--
4	To	5124
5	--	--

Obrázek 7: Hashovací tabulka

Dokonalá hashovací funkce je rychlá a pro dvě různé hodnoty nevypočítá stejné číslo. Pokud by se tak stalo, došlo by ke kolizi, která se musí ošetřit - například nalezením prvního prázdného místa v tabulce za/před vypočtenou hodnotou. Příkladem perfektní hashovací funkce je Cormackovo hashování[6].

Příkladem hashovací tabulky v reálném životě je jednoduchý telefonní seznam, seznam rodných čísel nebo seznam knih v knihovně.

2.5.2 Metody

Kromě standardních metod jako je vložení a smazání prvku, vyhledávání uvnitř hashovací tabulky pomocí hodnoty nebo indexů a prázdnost tabulky, je i výše zmíněná hashovací funkce. Existuje mnoho způsobů, jak takovou funkci vytvořit, ale Java sama už obsahuje vlastní hashovací funkci pro všechny základní datové typy.

2.5.3 Implementace

Hashovací tabulka se dá implementovat několika způsoby.

Nejzákladnější je implementace pomocí jednoduchého pole. Přes hashovací funkci se zjistí hash. Ten se vydělí celkovou kapacitou hashovací tabulky a tím se dostane index, do kterého se vkládaný objekt uloží. Může se stát, že by hashovací funkce přiřadila dvěma různým objektům stejný index. Tento problém se dá elegantně vyřešit vytvořením spojového seznamu na každém indexu. Pokud tedy v jednom místě už bude uložen objekt a hashovací funkce přiřadí jinému objektu stejnou pozici, tak se z prvního objektu vytvoří odkaz na nově přidaný objekt.

2.5.4 Ukázka kódu

V tomto případě je hashovací tabulka implementovaná jako pole. Tyto Prvky musí obsahovat metody pro nastavení a zjištění odkazu na další prvky. Má dvě instanční proměnné. První je pole objektů jménem tabulka a druhá je statická proměnná určující kapacitu.

Hashovací funkce je v tomto případě obsažena ve třídě prvků a vrací hodnotu hashe. S ní se následně pracuje při přidání prvku do tabulky.

```
public int hashCode(){
    int soucet = 0;
    for(int i = 0; i < hodnota.length();i++){
        char c = hodnota.charAt(i);
        soucet += (int)c*Math.pow(31, hodnota.length()-i);
    }
    return soucet;
}
```

Zdrojový kód 11: Hashovací tabulka - hashovací funkce

V metodě pridej(Prvek pridavany) se pak vypočítá zbytek po dělení tohoto hashe kapacitou tabulky. Tím se získá pozice pro zařazení nového objektu do hashovací tabulky.

Následně se zjistí, jestli se už na pozici nachází nějaký jiný objekt. Pokud ano, vytvoří se z něj odkaz na nově přidaný prvek. Pokud je pozice prázdná, vloží se na ní nový objekt.

```
public void pridej(Prvek pridavany){
    if(pridavany == null)
    || this.obsahujeHodnotu(pridavany.getHodnota())){
        return;
    }

    int pozice = pridavany.hashCode()%KAPACITA;
    Prvek porovnavany = tabulka[pozice];
    if(porovnavany == null){
        tabulka[pozice] = pridavany;
    } else if(porovnavany.equals(pridavany)){
        return;
    }
    else{
        while(porovnavany.getNasledujici() != null){
            if(porovnavany.equals(pridavany)){
                return;
            }
            porovnavany = porovnavany.getNasledujici();
        }
        porovnavany.setNasledujici(pridavany);
    }
}
```

Zdrojový kód 12: Hashovací tabulka - přidání nového objektu

Při mazání prvku se musí postupně projít všechny možné indexy a najít mazanou hodnotu uvnitř tabulky. Následně dojde k přepsání odkazu z předchozího objektu na objekt za mazaným prvkem.

```
public void smazHodnotu(String hodnota){
    for(int i = 0; i < KAPACITA; i++){
        Prvek docasny = tabulka[i];
        if(docasny == null){
            continue;
        }
        if(docasny.getHodnota().equals(hodnota)){
            tabulka[i] = docasny.getNasledujici();
            return;
        }
      while(docasny.getNasledujici() != null){
         Prvek dalsi = docasny.getNasledujici()
         String docasnaHodnota = dalsi.getHodnota();
            if(docasnaHodnota.equals(hodnota)){
            PrvekM nasledujici = dalsi.getNasledujici();
                if(nasledujici != null){
                    docasny.setNasledujici(nasledujici);
                }
            }
            docasny = docasny.getNasledujici();
        }
    }
}
```

Zdrojový kód 13: Hashovací tabulka - smazání prvku

3 Aplikace Interaktivní datové struktury

3.1 Základní popis aplikace

Aplikace Interaktivní datové struktury slouží k výuce datových struktur popsaných výše. Je určena jak naprostým začátečníkům, jako jsou např. žáci ZŠ, tak i zkušenějším uživatelům. Ti mohou tuto aplikaci použít pro grafické znázornění a vizualizaci.

Samotná aplikace byla vytvořena v Javě za pomoci frameworku LibGDX, který primárně slouží k vytváření her. Tento framework byl použit proto, že jedním ze záměrů aplikace bylo, aby si uživatel s jednotlivými strukturami hrál, modeloval je a sledoval, jak se mění. Aplikace byla navíc navrhnuta s ohledem na případné následné přidání nových modulů pro další datové struktury nebo dokonce i témat mimo informatiku.

Obrázek 8: Hlavní menu aplikace

3.2 LibGDX

LibGDX je open-source knihovna určena pro tvorbu 2D i 3D her. Díky tomu není potřeba řešit, jak renderovat grafiku nebo interaktivitu mezi jednotlivými prvky, protože to už řeší sám LibGDX[7].

Vše na obrazovce (kromě pozadí) je odvozeno z třídy Actor, která v sobě má zabudovány metody pro řešení kolize, mazání instance této třídy nebo například nastavení textury, která se zobrazí na obrazovce a bude reprezentovat jednotlivé instance.

3.3 Spuštění a hlavní menu aplikace

Samotná aplikace se spouští souborem **ids.jar**, který lze najít na přiloženém CD. Není potřeba žádná instalace. Krom toho se na zmiňovaném CD nachází i zdrojový kód celé aplikace, který obsahuje mimo jiné i ukázkovou implementaci všech výše zmíněných datových struktur.

3.3.1 Hlavní menu

Hlavní menu aplikace obsahuje 3 položky - Start, Knihovna a Konec.

Položka Start vpustí uživatele do dalšího menu, kde si může vybrat jednu z datových struktur, se kterou bude chtít pracovat. Více se o každém modulu nachází níže.

Položka Knihovna dovede uživatele do sekce, kde se nachází krátké shrnutí o každé z datových struktur.

A nakonec položka Konec, jak už napovídá její jméno, ukončí celou aplikaci.

3.4 Obecné ovládání aplikace

Všechny moduly se ovládají levým, případně i pravým tlačítkem myši.

Navíc existuje několik klávesových zkratek. Po zmáčknutí klávesy **s** a následném kliknutím na jeden z prvků na obrazovce, dojde k jeho smazání za předpokladu, že je možné ho smazat. Klávesa **m** a tlačítko **X** vrátí uživatele zpět do hlavního menu. Klávesa **z** vyvolá na obrazovku popis současné datové struktury. Stejný popis též spouští tlačítko **?**, které se nachází nahoře na obrazovce u každého modulu.

Mezi jednotlivými modelovými příklady lze přepínat pomocí šipek **vpravo** a **vlevo** nebo pomocí tlačítek $<$ a $>$.

3.5 Modul Fronta

V tomto modulu se nachází příklady k modelování, které mají za cíl uživateli ukázat, jak funguje datová struktura fronta.

Po spuštění má uživatel možnost vybrat si mezi volným modelováním nebo příklady.

Obrázek 9: Reálný příklad fronty

Pro začátečníky je určen příklad **Školní jídelna**. Ten ukazuje jednoduchou frontu ve školní jídelně vytvořenou jen ze žáků. Uživatel zde nemá moc možností jak frontu vymodelovat, může jen přidat nebo odebrat jednoho žáka a sledovat chování fronty. Žáka uživatel přidá kliknutím na pravé tlačítko myši. Odstranění probíhá kliknutím na jeden z obědů, které se na obrazovce nacházejí. Krom toho je možné tlačítkem aktivovat prioritní frontu, které přidá do fronty i učitele. Ti do fronty chodí náhodně a místo toho, aby v ní stáli, tak všechny žáky předběhnou a sami si vezmou oběd.

Pro pokročilejší uživatele je určeno **Volné modelování**. Jak už název napovídá, zde si může uživatel vymodelovat svojí vlastní frontu. Prvek se do fronty přidává kliknutím na pravé tlačítko myši nejdříve na přidávaný prvek a následně na pozici, na kterou se má prvek přidat. V tomto režimu jsou jednotlivé prvky fronty znázorněny jako modré čtverečky, začátek fronty je pak červený čtvereček a konec fronty zelený čtvereček. Program hlídá validitu této fronty a upozorní uživatele na chyby, kterých se může dopustit.

3.6 Modul Zásobník

V tomto modulu se nachází interaktivní příklady, které uživateli pomohou pochopit datovou strukturu zásobník.

V příkladu **Školní kuchyně** se uživatel podívá dovnitř školní kuchyně a bude mít na starost mytí nádobí. Na uživatele už čeká předpřipravený zásobník špinavého nádobí. Nové nádobí se přidá kliknutím na pravé tlačítko myši, pokud se tedy do komínku ještě vejde. Umytím talíře ve dřezu, dojde k jeho odstranění ze zásobníku. S talíři se hýbe levým tlačítkem myši.

Druhé je **Volné modelování**, kde si uživatel může vytvořit vlastní zásobník. Prvky se do zásobníku přidávají kliknutím pravým tlačítkem myši na přidávaný prvek a následně na pozici, na kterou má být prvek přidán. Objekty jsou znázorněny jako modré čtverečky s tím, že z vrchního prvku se stane zelený čtvereček. Červený čtvereček pak znázorňuje nejstarší prvek v zásobníku.

Tento mód obsahuje i řádek, který dává uživateli přehled o tom, co udělal a dává mu vědět, jaké operace s řádkem nemůže provádět.

Obrázek 10: Volné modelování zásobníku

3.7 Modul Spojový seznam

V tomto modulu jsou příklady určené pro výuku spojového seznamu.

Jako první je doporučován příklad **Vlakové nádraží**. V něm jsou na vlakové soupravě vysvětleny základní pojmy spojového seznamu. Uživateli je umožněno poskládat svůj vlastní vlak a poslat ho do světa. V jednu chvíli ale na obrazovce může být jen jeden takový seznam. Uživatel může pravým tlačítkem označit dvě části vlaku, které se následně spojí ve vlak, pokud to jde. Poklikáním pravým tlačítkem na lokomotivu se vlak rozjede a odjede pryč z obrazovky. Tím také dojde k vyprázdnění spojového seznamu. Kromě toho mají hlavička i ocásek speciální texturu, aby je mohl uživatel rychle rozlišit. Hlavička je znázorněna jako lokomotiva a ocásek jako zelený vagón.

Volné modelování má 2 podoby. V jedné verzi pracuje uživatel s jednosměrným spojovým seznamem a v druhé s obousměrným. Uživatel může spojovat modré čtverečky a vazby mezi nimi jsou znázorněny šipkami. Prvky se do spojového seznamu přidají kliknutím pravým tlačítkem myši na přidávaný prvek a následně na pozici, na kterou se má přidat. Znovu se jedná o modely určené pro pokročilejší uživatele.

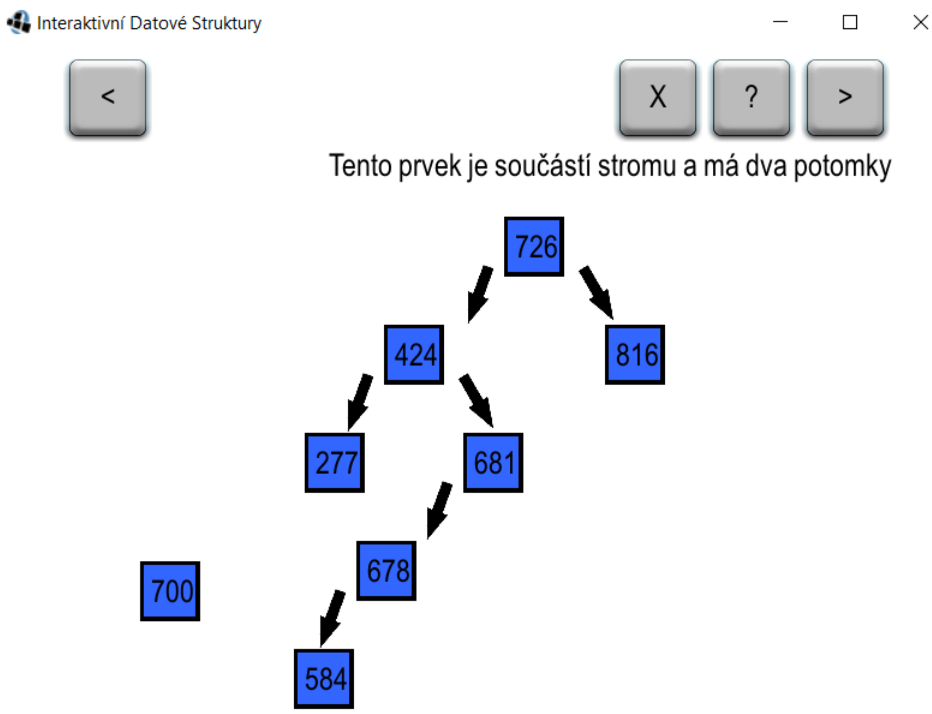
Obrázek 11: Reálný příklad spojového seznamu

3.8 Modul Binární vyhledávací strom

Tento modul slouží k výuce binárního vyhledávacího stromu. Vzhledem k obtížnému znázornění této datové struktury do reálného života je celý tento modul určen pro pokročilejší uživatele a předpokládá se, že si uživatel nejdříve přečetl oddíl této práce věnovaný binárním vyhledávacím stromům.

Příklad **Morseova abeceda** využívá binární vyhledávací strom k uspořádání písmen abecedy tak, aby se pohybem po stromu ke každému písmenu vytvořil odpovídající kód Morseovy abecedy. Pohyb doleva znamená tečku a pohyb doprava čárku. Písmena jsou rozmístěna tak, že ke každému vede nejkratší možná cesta. Uživatel pak může z jednotlivých písmen sestavit slova a věty.

Následně je v této části také **Volné modelování**. V něm si uživatel vymodeluje svůj vlastní strom, zatímco ho program kontroluje. Zde uživatel pracuje v očíslovanými modrými čtverečky, které může pomocí pravého tlačítka propojit. Propojení proběhne jen tehdy, je-li jeden z prvků uvnitř stromu a druhý splňuje všechny podmínky pro propojení s tímto prvkem.



Obrázek 12: Volné modelování binárního vyhledávacího stromu

3.9 Modul Hash tabulka

V tomto modulu jsou uživateli k dispozici cvičení pro výuku hashovací tabulky.

V příkladu **Knihovna** je uživateli předložen seznam knih strukturovaný jako hashovací tabulka a je mu umožněno si s ním hrát a předělávat ho. Může do něj vložit nové knížky, ale přidávání se drží předem dané hashovací funkce.

Ve **Volném modelování** si může uživatel vybrat jednu z několika předpřipravených hashovacích funkcí a aplikovat ji na seznam hodnot. Hashovací funkci může měnit za běhu programu a tak sledovat, jak se hashovací tabulka mění.

Obrázek 13: Reálný příklad hashovací tabulky

4 Vytvoření nového modulu

4.1 Úvod

Celá aplikace Interaktivní datové struktury byla navržena tak, aby do ní bylo možné přidávat další moduly. V této kapitole bude na příkladu popsán krátký postup jak takový modul správně vytvořit. Návod ale nemá za úkol naučit uživatele jazyk Java a předpokládá se jeho předchozí znalost. Stejně tak se předpokládá základní znalost frameworku LibGDX, ale vzhledem k tomu, že by s ním běžný programátor Javy nemusel být seznámen, bude jeho základní fungování popsáno.

Kompletní zdrojový kód aplikace obsahující i kód z této kapitoly je k dispozici na přiloženém CD. Stejně tak se na CD nachází i veškerá grafika použita v tomto případu.

4.2 Množina

Jako příklad bude přidání abstraktní datové struktury jménem Množina. Nejdříve by bylo dobré si tuto strukturu popsat stejně jako všechny struktury v první části této práce.

4.2.1 Definice

Jak už bylo řečeno, množina je abstraktní datová struktura. Prvky v ní se nijak neřadí, nemají tedy jasné pořadí, což může být problém při procházení množiny. Na druhou stranu jsou všechny prvky uvnitř množiny vždy přístupné a může se s nimi manipulovat. Kromě toho může množina obsahovat různé druhy dat, např. čísla a textové řetězce.

4.2.2 Metody

Pro práci s množinou bude potřeba vytvořit několik metod. Pro názornou ukázku budou stačit metody pro přidání (add) a smazání (remove) určitého prvku, zjištění velikosti a prázdnosti množiny a nakonec vyhledání prvku uvnitř množiny.

4.2.3 Implementace

Množina se dá implementovat mnoha způsoby. Jako každá datová struktura, i množinu lze implementovat jako jednoduché pole. Toto řešení se hodí při vytváření množin s předem danou délkou.

Další možnost pro implementaci množiny je Hash mapa. V tomto případě se jedná o Hash mapu, která má jen klíče. Výhodou je neomezená kapacita.

Množinu je též možné implementovat jako strom.

4.2.4 Ukázka kódu

Tato množina je implementována jako pole a pracuje s instancemi třídy Prvek.

Při přidání prvku dojde ke kontrole kapacity množiny. Pokud je už množina plná, zvětší se její kapacita a dojde k zapsání přidávaného prvku na první volnou pozici.

```
public void pridej(Prvek prvek){
    if(prvek == null){ return; }
    if(obsahuje(prvek)){ return; }
    if(velikost() == kapacita){
        zmenKapacitu(kapacita*2);
    }
    for(int i = 0; i < kapacita; i++){
        if(mnozina[i] == null){
            mnozina[i] = prvek;
            return;
        }}}
```

Zdrojový kód 14: Množina - přidání prvku

4.3 Návrh příkladu

V tomto návodu bude jako příklad vytvořeno modelové prostředí pro množinu. Uživateli bude předložena jedna množina s omezenou kapacitou (měnitelnou uživatelem), do které bude moci vkládat nové instance třídy Prvek, zde znázorněny jako čtverečky s čísly uvnitř. Samotná množina pak bude vypadat jako jednoduchý kruh.

Objekty uvnitř množiny budou mít zelenou barvu, zatímco objekty mimo budou modré.

4.4 Hlavní třída

Prvním krokem je vytvoření třídy. Tato třída musí být odvozena od abstraktní třídy BaseScreen. Ta zajišťuje správné vykreslení objektů na obrazovku.

Vzhledem k zdědění všeho potřebného stačí v konstruktoru zavolat konstruktor třídy BaseScreen. Narozdíl od normální Javovské třídy v konstruktoru ale nedochází k inicializaci globálních proměnných.

4.4.1 Metoda create()

Aplikace vytvořené pomocí frameworku LibGDX využívají tvz. herního cyklu (anglicky Game Loop), během kterého dojde k aktualizaci celé obrazovky programu.

Ještě než se ale tento cyklus spustí, dojde k zavolání funkce create(). Ta, jak už název napovídá, má za úkol vytvořit počáteční pozice všech objektů na obrazovce. Dochází zde v vytvoření pozadí a základních objektů, případně k inicializaci proměnných.

V tomto případě se zde vytvoří pozadí a prázdná instance třídy Mnozina.

4.4.2 Metoda update()

Tato metoda se spouští pořád dokola v nekonečném cyklu, dokud je aplikace spuštěná. V hlavní aplikaci je tato metoda využita převážně k aktualizaci stavového řádku a k úplnému vymazání objektů, které už na obrazovce nejsou.

Zde se bude kontrolovat a nastavovat správná barva jednotlivých objektů. Kromě toho se zde bude i aktualizovat stavový řádek s informací o prázdnosti a plnosti množiny.

4.4.3 Vytvoření objektů

Jedna z nejdůležitějších metod je samotné vytvoření objektů, se kterými může uživatel pracovat.

Musí se vytvořit nová instance třídy Prvek, nastavit její texturu, šířku, výšku, souřadnice a nakonec se musí tato instance přidat na obrazovku.

V této metodě je také dobré přiřadit každé instanci InputListener, popřípadě ClickListener. Ty především hlídají, jestli uživatel na objekt klikl nebo s ním hnul a provedou určitou akci.

V tomto případě se bude přes InputListener hlídat pohyb objektu a jeho případné smazání.

4.4.4 Mazání objektů

Mazání objektů je potřeba hlídat na několika místech. Jednak je potřeba správně smazat objekt z obrazovky a pak i množiny.

Všechny třídy odvozené od třídy Actor dědí její metodu remove(). Díky ní lze smazat objekty těchto tříd z obrazovky.

Pořád ale tyto objekty zůstanou uložené v množině. Nebudou naoko vidět, ale jejich přítomnost se odrazí při počítání velikosti nebo zpětném vyhledávání těchto objektů. Proto je potřeba ještě tyto objekty odstranit z množiny přes výše vytvořenou metodu smaz().

Alternativní přístup je přidání nové proměnné jedné z datových struktury, do které se budou ukládat všechny objekty určené ke smazání a obsah celé struktury pak mazat v metodě update(). Tento postup se hodí pokud by objekty měly opustit hranice obrazovky.

4.4.5 Tlačítka a UI

Tlačítka jsou ve frameworku LibGDX vytvořena přes třídu TextButton a využívají InputListener k zjištění, jestli na ně bylo kliknuto. Při vytváření tohoto případu není potřeba vytvářet moc tlačítek. Bude stačit jedno pro vrácení zpátky do menu (v hlavním programu označeno křížkem) a jedno, které zobrazí popis množiny (v hlavním programu značeno jako ?).

Popis je defaultně nastavena tak, aby nebyl vidět, ale po zmáčknutí tlačítka ? se vše na původní obrazovce zastaví a následně se tento popis zobrazí. Při vytváření množiny se ještě hodí napsat její celková kapacita, počet prvků v množině a jejich výčet.

Poslední důležitou částí UI je stavový řádek, který dá uživateli vědět základní informace o tom, co právě na obrazovce probíhá. To je například přidání nebo odebrání prvků z množiny, hlídání nebo jestli je množina plná nebo úplně prázdná. Tento řádek je vytvořen jako instance třídy Label a je nadále aktualizován při každém kliknutí na objekt. Plnost a prázdnost množiny se kontroluje v metodě update().

Všechny tyto komponenty se přidají do proměnné uiTable. Ta před obrazovku vytvoří pomyslnou tabulku a do každé její kolonky se přidá jeden z objektů. Počet sloupců a řádků je měnitelný a každá kolonka se dá následně do jisté míry upravit. Je možno zvětšit okraje, sloučit více buněk do sebe nebo nastavit zarovnání textu uvnitř buňky.

4.5 Menu

Po úspěšném vytvoření interaktivního příkladu je potřeba k němu vytvořit cestu. Hlavní menu aplikace se obsaženo ve třídě Menu a výběrové menu ve třídě VyberoveMenu. Právě v něm je potřeba přidat tlačítko spouštějící tento příklad. V hlavní aplikaci má každá datová struktura ještě svoje vlastní menu (třídy FrontaMenu, ZasobnikMenu, SeznamMenu, StromMenu, TabulkaMenu), které obsahuje odkaz na jednotlivé příklady daných struktur. Ale vzhledem k tomu, že tento návod operuje jen s jedním takovým příkladem, tak není takové mezimenu potřeba.

4.6 Knihovna

Nakonec je ještě třeba vytvořit odpovídající odkaz v Knihovně, vytvořené přes třídu Knihovna. Zde jsou tlačítka a text také naskládány na obrazovku přes uiTable. Ten je rozdělen na dvě části. V jedné jsou uloženy odkazy v podobě tlačítek, které aktualizují proměnnou popis (typu Label) v druhé části uiTable.

Pro vytvoření záznamu o množině stačí přidat tlačítko, které přepíše proměnnou popis na popis množiny.

4.7 Shrnutí

Při vytváření nového modulu je potřeba vytvořit novou třídu, která bude spravovat dění na obrazovce, jako je vytvoření pozadí a objektů, se kterými může uživatel pracovat. Stejně tak je důležité navrhnout takové uživatelské rozhraní, které dovolí uživateli návrat do hlavní nabídky a bude uživatele informovat o dění na obrazovce.

Do každého modulu by se uživatel měl dostat přes hlavní menu a každý modul by měl být také krátce charakterizován v Knihovně.

5 Závěr

V práci jsem shrnul teoretické poznatky o frontě, zásobníku, spojovém seznamu, binárním vyhledávacím stromě a hashovací tabulce. Nakonec jsem ještě na konci přidal popis další datové struktury - množiny. Kromě toho jsem sepsal návod jak do aplikace případně přidat další moduly.

Současná verze aplikace podporuje výuku datových struktur fronty, zásobníku, spojového seznamu, binárního vyhledávacího stromu a hashovací tabulky. Všechny tyto struktury obsahují po jednom příkladu z reálného života a možnost volného modelování. Kromě toho též aplikace obsahuje i krátký popis jednotlivých struktur v sekci Knihovna a v režimu volného modelování. Tím pádem byly všechny mnou vytyčené cíle splněny.

Z časových důvodů bohužel nebyla aplikace řádně testována na cílové skupině uživatelů, ale na menší skupině, které obsahovala jak informatiky, tak i studenty z jiných oborů, převážně ekonomiky. Největší úspěch sklidil reálný příklad u spojového seznamu - vlakové nádraží. Na něm se uživatelům líbila animace odjíždějícího vlaku. Nejméně se líbil reálný příklad u zásobníku, který graficky nezapadal do zbytku programu. Také jsem díky těmto testům narazil na několik, převážně grafických bugů, které bylo potřeba opravit.

Ve vývoji aplikace bych rád pokračoval i nadále, protože si myslím, že má smysl do ní přidat i další datové struktury. Také zatím není nikde zveřejněna, což je další věc, kterou bych rád do budoucna změnil.

Seznam použité literatury a zdrojů

[1] Definition of data structure. English Oxford Living Dictionaries [online]. Oxford: Oxford University, 2019 [cit. 2019-04-10]. Dostupné z: <https://en.oxforddictionaries.com/definition/data_structure>

[2] GOODRICH, Michael T. a Roberto TAMASSIA. Data structures and algorithms in Java. 4th ed. Hoboken, NJ: Wiley, c2006, s. 285. ISBN 0-471-73884-0

[3] GOODRICH, Michael T. a Roberto TAMASSIA. Data structures and algorithms in Java. 4th ed. Hoboken, NJ: Wiley, c2006, s. 261-262. ISBN 0-471-73884-0.

[4] MEHTA, Dinesh P. a Sartaj SAHNI. Handbook of data structures and applications. Boca Raton, Fla.: Chapman & Hall/CRC, c2005, s. 66. ISBN 1-58488-435-5.

[5] SEDGEWICK, Robert a Kevin Daniel WAYNE. Algorithms. 4th ed. Upper Saddle River, NJ: Addison-Wesley, c2011, s. 459. ISBN 978-0-321-57351-3.

[6] G. V. Cormack, R. N. S. Horspool, M. Kaiserswerth, Practical Perfect Hashing, The Computer Journal, Volume 28, Issue 1, 1985, Pages 54–58, https://doi.org/10.1093/comjnl/28.1.54

[7] STEMKOSKI, Lee. Beginning Java game development with LibGDX. Berkeley, CA: Apress, [2015], s. 11. ISBN 978-1-4842-1501-2.

Seznam obrázků

Seznam zdrojových kódů

A Příloha 1

CD/DVD