|  |
| --- |
| **ZÁPADOČESKÁ UNIVERZITA V PLZNI** |
| FAKULTA EKONOMICKÁ |
| Semestrální práce z předmětu KIV/PT |
| **Necháme to bloudovi s. r. o** |
| Petr Chrz, Radek Nolč  Plzeň 2022  . |

# Analýza problému

Z hlediska paměťové a časové náročnosti bylo jedním z hlavních problémů zvolení vhodných datových struktur, zvolení vhodného grafového algoritmu, protože se jedná hlavně o grafovou úlohu, a vhodné použití tříd.

Pro zpracování dat z textového souboru bylo zapotřebí zhodnotit možné varianty. Program nejdříve musí přečíst celý soubor se vstupem a s ním dále pracovat. Spojování řádků pomocí Stringů je v tomto ohledu neefektivní. V malém rozsahu je to uspokojivé, ve velkém rozsahu už silně časově neefektivní. Proto pro spojování řádků bylo na výběr mezi StringBuilderem a StringBufferem. Rozdíl mezi StringBuilderem a StringBufferem je hlavně v tom, zda je asynchronní nebo synchronní. V programu byl použit StringBuilder, protože se jedná o program pracující s jedním vláknem, tudíž nemůže nastat žádný problém.

Aby program mohl fungovat, je zapotřebí ukládat instance jednotlivých tříd do nějaké datové struktury. Vzhledem k tomu, že se jedná spíše o dynamické programování, nelze jednoduše použít pole. Proto pro ukládání instancí stěžejních tříd byl použit ArrayList, protože umožňuje ukládat prvky bez toho, abychom dopředu věděli, kolik jich bude.

Pro zjištění nejkratší cesty v grafu je možné použít 3 základní algoritmy. Dijkstrův algoritmus, Bellman-Fordův algoritmus a Floyd-Warshallův alogirtmus. Pro tento program se jeví nejvhodnější použít Floyd-Warshallův algoritmus, přestože se jedná o nejnáročnější. Floyd-Warshallův algoritmus byl použit právě kvůli jeho jedné vlastnosti, a to, že zjišťuje nejkratší cestu mezi všemi vrcholy, což je v našem programu žádoucí. Pokud by byl zvolen nějaký jiný algoritmus, přesto by musela být zjištěna nejkratší cesta mezi všemi vrcholy, pokud chceme mít program co nejefektivnější.

# Návrh programu

Nejdříve bylo zapotřebí zajistit, jak zavést do programu nějaké lokace. Pro tuto funkčnost byly vytvořeny 2 základní (abstraktní) třídy – Point (definující specifický bod v mapě) a Location (definující lokaci). Na třídě Location dále staví již klasické třídy, tedy Oasis (definující Oázu) a Storage (definující sklad).

Pro generování velbloudů bylo zapotřebí nejdříve definovat třídu CamelTemplate, která přijímá všechny atributy potenciálního velblouda a v programu se chová jako šablona. Následně je zde ještě třída Camel, která již definuje normálního, vytvořeného velblouda.

Dále bylo zapotřebí nějak určit cesty mezi těmito lokacemi. Pro tuto funkčnost je zde zajištěna třída Path, která přebírá v konstruktoru dvě lokace, mezi kterými jsou cesty.

Pro matematické výpočty je zde třída Calculator. Tato třída počítá vzdálenost mezi dvěmi lokacemi, čas cesty a rozdělení (rovnoměrné a normální).

Aby program mohl vygenerovat velbloudy s nějakým poměrem, je zavedena třída Factory, která s těmito poměry počítá. V této třídě je pomocí pole o 100 prvcích vybrán, a následně vytvořen velbloud ze šablony.

Pro čtení vstupu je zde třída DataReader. Tato třída se stará o čtení dat ze vstupu uživatele. Tato třída nejdříve načte kompletní soubor jako text a následně vymaže všechny nepotřebné informace – komentáře. Po vyčištění dat přichází na řadu zpracování, kde se již načítají data do programu.

V rámci snadné manipulace s daty je zavedena třída Map, který představuje mapu se všemi potřebnými informacemi. Tato třída tedy uchovává cesty mezi všemi lokacemi včetně jejich vzdáleností a zároveň i zjišťuje na základě požadavku nejkratší cestu mezi dvěma lokacemi.

Jako další z tříd je zavedena třída Request. Tato třída se stará o uchovávání požadavků a operacemi nad nimi.

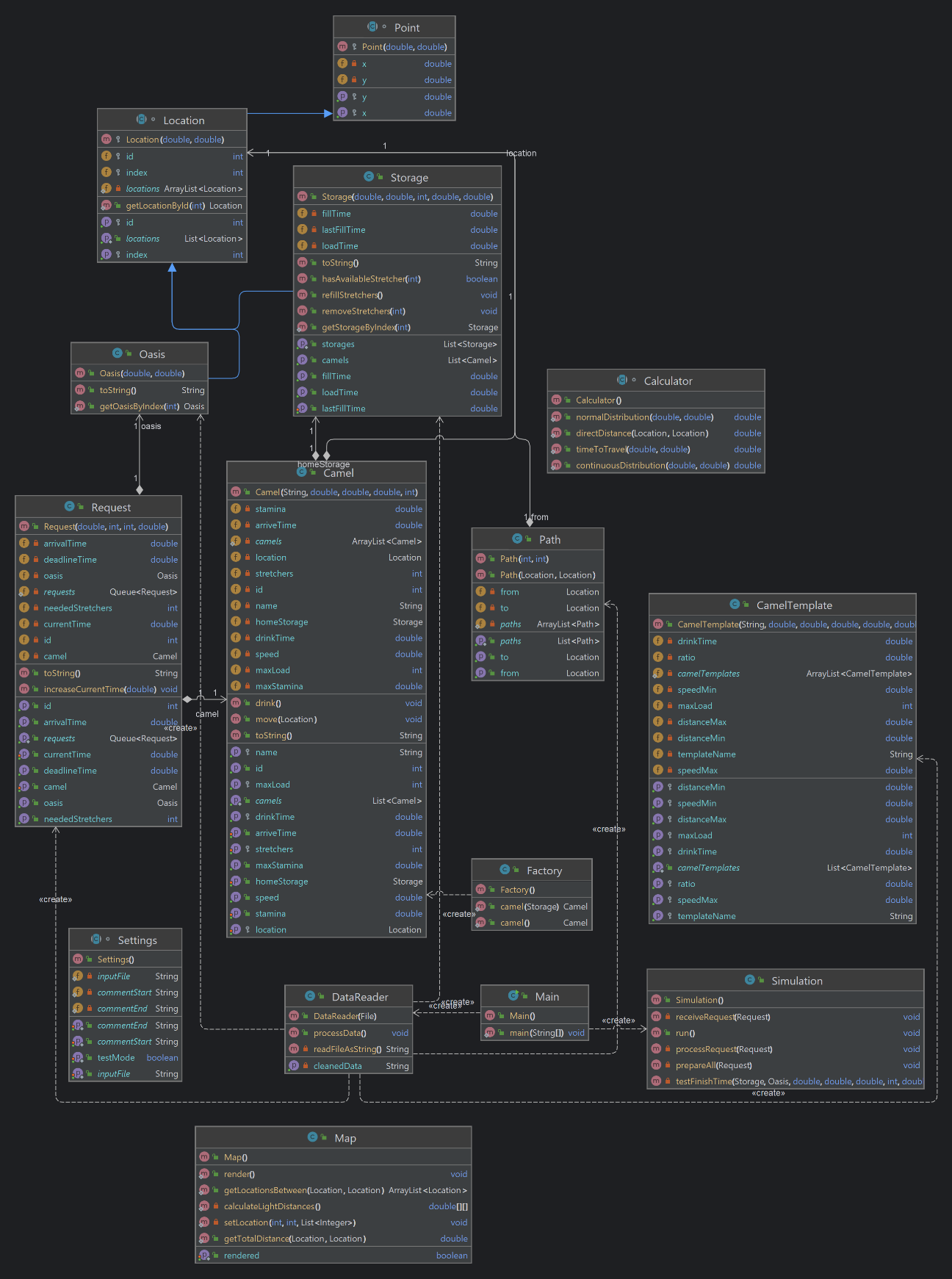
A jako poslední, nejdůležitější třída, je Simulation. Tato třída se stará o celý běh programu a bez této třídy by program nemohl fungovat tak, jak funguje.

## UML diagramy

Pro lepší orientaci v návrhu programu byly vytvořeny dva UML diagramy. První UML diagram je zjednodušený, přehledný. Druhý UML diagram již detailnější, s použitými metodami.

Obsah obrázku text, černá, snímek obrazovky, obrazovka

Popis byl vytvořen automatickyObrázek 1 Zjednodušený UML diagram

Obrázek 2 Detailní UML diagram

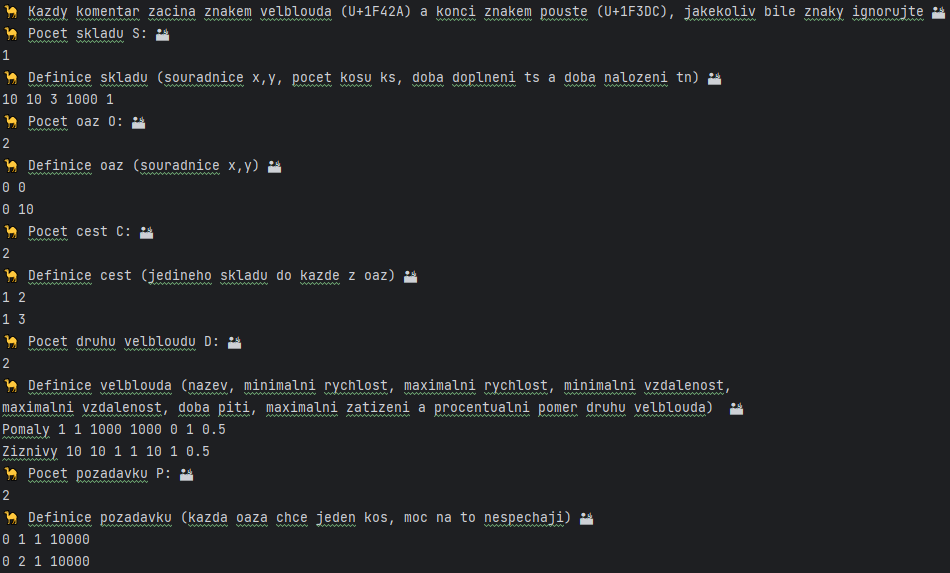
# Uživatelská dokumentace

## O programu

Program slouží pro simulaci efektivní přepravy košů pomocí velbloudů mezi lokacemi. Vstupem je textový soubor s daty. Po zpracování dojde k postupnému výpisu. Tento výpis obsahuje informace o požadavku. Který velbloud se stará o dokončení požadavku, kde se aktuálně nachází, případně co aktuálně dělá.

## Příprava dat

Před samotným spuštěním je zapotřebí si připravit nějaká vstupní data. Tato data musí mít určitou strukturu, jinak program nebude fungovat správně. Strukturu dat včetně popisu můžete nalézt na obrázku č. 3.

Soubor se vstupními daty se musí jmenovat input.txt a musí se nacházet ve stejné složce jako soubor app.jar.

Obrázek 3 Ukázka vstupních dat

## Spuštění programu

Před spuštěním se ujistěte, že máte připravený soubor s daty, tento soubor se jmenuje input.txt a je ve stejné složce jako app.jar.

Pro spuštění programu můžete využit více způsobů, přes vývojové prostředí, přes shell script anebo přes terminál.

### Spuštění přes vývojové prostředí

Program lze spustit přes vývojové prostředí, například přes IntelliJ. Nejdříve je potřeba si toto vývojové prostředí otevřít, otevřít v něm konkrétní projekt (v našem případě app) a vpravo nahoře stisknout zelené „play“ tlačítko.

### Spuštění přes shell script

Obrázek 4 Postup špuštění programu přes IntelliJ

Pokud máte k dispozici program pro spouštění .sh souborů, program lze snadno spustit přes soubor run.sh, který se nachází ve složce spolu s input.txt a app.jar.

### Spuštění přes terminál

Nejjednodušší způsob spuštění přes terminál v OS Windows je vstup do složky s input.txt a app.jar. Klikněte levým tlačítkem myši do pole s cestou k souboru (vedle prohledávání) až celá cesta zmodrá. Nyní napište do tohoto pole *cmd* a stiskněte *ENTER*. Otevře se vám příkazový řádek. Nyní do příkazového řádku vepište *java -Dfile.encoding=UTF-8 -jar app.jar* a stiskněte *ENTER*, tím se program spustí.

Pokud se nacházíte na Linuxu / MacOS, musíte jít do vybrané složky přes příkaz cd. Pro vstup do adresáře za cd napište název adresáře. Pro výstup z adresáře napište cd .. (dvě tečky). Nyní stačí napsat do terminálu pouze *java -Dfile.encoding=UTF-8 -jar app.jar* a stiskněte *ENTER*, tím se program spustí.

# Závěr

Závěrem bych si dovolil zhodnotit průběh realizace této semestrální práce. Nejlehčí byl prvotní návrh programu na základě zadání. Postupem vývoje se však objevovaly různé problémy, které na začátku nebyly zjevné. Mezi nejnáročnější části patřilo vymýšlení fungování programu do detailu a následná implementace. Neméně náročné byla i implementace již existujících algoritmů.

Jednalo se o první projekt takového rozsahu za dobu studia. Osobně jsem na tuto práci pyšný. Podařilo se mi dosáhnout minimálních požadavků i přes fakt, že v rámci studia na FEK je programování spíše minoritní.

## Přínos jednotlivých členů týmu

Níže lze nalézt detailnější přínos jednotlivých členů týmu. Jedná se o co nejvíce objektivní zhodnocení přínosu.

**Radek Nolč**:

* Návrh programu na základě zadání
* Zkonstruování UML diagramu
* Vyhotovení programu na základě UML diagramu
* Implementace potřebných datových struktur
* Implementace Floyd-Warshall algoritmu a dalších algoritmů
* Naprogramování čtení a parsování vstupního souboru
* Naprogramování chodu simulace
* Optimalizace časové a výpočetní náročnosti
* Oprava / zakomponování připomínek při průběžné kontrole práce
* Dokumentační komentáře všech tříd
* Vygenerování Javadoc
* Vygenerování UML diagramů za účelem vyhotovení strukturované dokumentace
* Strukturovaná dokumentace

**Petr Chrz**:

* Minimální úsilí
* Neuspokojivé dokumentační komentáře
* Nepoužitelná strukturovaná dokumentace