TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

OBJEKTOVĚ ORIENTOVANÝ FRAMEWORK PRO PROVOZNÍ DATA PODZEMNÍCH ZÁSOBNÍKŮ PLYNU

Bakalářská práce

Studijní program: B2646 – Informační technologie
Studijní obor: 1802R007 – Informační technologie

Autor práce: **Jaroslav Hrabal**
Vedoucí práce: doc. Ing. Otto Severýn, Ph.D.

Liberec 2015

Akademický rok: **2014/2015**

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies

OBJECT ORIENTED FRAMEWORK FOR UNDERGROUND GAS STORAGE OPERATIONAL DATA

Bachelor thesis

Study programme: B2646 – Information Technology
Study branch: 1802R007 – Information Technology

Author: **Jaroslav Hrabal**
Supervisor: doc. Ing. Otto Severýn, Ph.D.

Liberec 2015

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií
Akademický rok: **2014/2015**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Jaroslav Hrabal**
Osobní číslo:	**M12000134**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Objektově orientovaný framework pro provozní data podzemních zásobníků plynu**
Zadávající katedra:	**Ústav mechatroniky a technické informatiky**

Zásady pro vypracování:

1. Seznamte se s problematikou podzemního skladování plynu obecně a dále s metodikou sběru a archivace provozních dat PZP na pracovišti Geo-services firmy RWE Gas Storage s.r.o.
2. Navrhněte a v jazyce Java implementujte softwarové rozhraní, které umožní přístup k takto uloženým datům. Rozhraní bude respektovat hierarchii zásobník - skupina sond - sonda - veličina. Dále umožní základní manipulace s daty - agregace v čase a ve skupině, základní čištění dat, interpolace, výpočet odvozených veličin (kumulativních objemů, statických tlaků atp.) a export dat do souborů v obecně použitelných formátech.
3. Pomocí rozhraní vytvořeného v bodě 2 navrhněte a implementujte aplikaci pro zobrazování okamžitého stavu PZP.

Rozsah grafických prací: **dle potřeby dokumentace**

Rozsah pracovní zprávy: **30–40 stran**

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

[1] **Eckel, B.: Thinking in Java, 4th edition, Prentice Hall, 2006.**

[2] **Dake, L.: Fundamentals of reservoir engineering. Elsevier, 1978.**

Vedoucí bakalářské práce: **doc. Ing. Otto Severýn, Ph.D.**
Ústav mechatroniky a technické informatiky

Konzultant bakalářské práce: **doc. Ing. Dalibor Frydrych, Ph.D.**
Ústav nových technologií a aplikované informatiky

Datum zadání bakalářské práce: **10. října 2014**

Termín odevzdání bakalářské práce: **15. května 2015**

prof. Ing. Václav Kopecký, CSc.
děkan

L.S.

doc. Ing. Milan Kolář, CSc.
vedoucí ústavu

V Liberci dne 10. října 2014

Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum: 15. 5. 2015

Podpis: Hrabal

Poděkování:

Rád bych na tomto místě chtěl poděkovat všem, s jejichž pomocí tato práce vznikla. Zvláště chci poděkovat svému vedoucímu práce, jímž je doc. Ing. Otto Severýn, Ph.D. za jeho odborné vedení bakalářské práce a pomoc, kterou mi při práci poskytl. Také bych rád poděkoval své rodině (především svému otci) a svým přátelům.

Anotace

Bakalářská práce se zabývá objektově orientovaným programováním. Zaměřuje se na jeho užívání a na návrh frameworku schopného ukládání a manipulace s daty získanými z měření parametrů na podzemních zásobnících plynu. Framework byl realizován použitím jazyku Java. v práci jsou popsány metody pro načítání datových souborů, ukládání dat do paměti, matematické operace s daty, výběr dat na základě parametrů a pro zobrazení dat. Pomocí navrženého frameworku je sestaveno grafické rozhraní umožňující jednoduchou práci s daty.

Klíčová slova

Java, podzemní zásobník plynu, objektově orientované programování, framework, grafické rozhraní.

Annotation

This bachelor’s thesis deals with object-oriented programming. It focuses on using object-oriented programming to develop a framework capable of storing and manipulating data obtained by measuring underground gas storage. The framework was built using Java. The work describes methods to read data files, store data, work data using mathematical functions, select data and to view data. The framework is used to make a graphical user interface for easy data manipulation.

Key Words

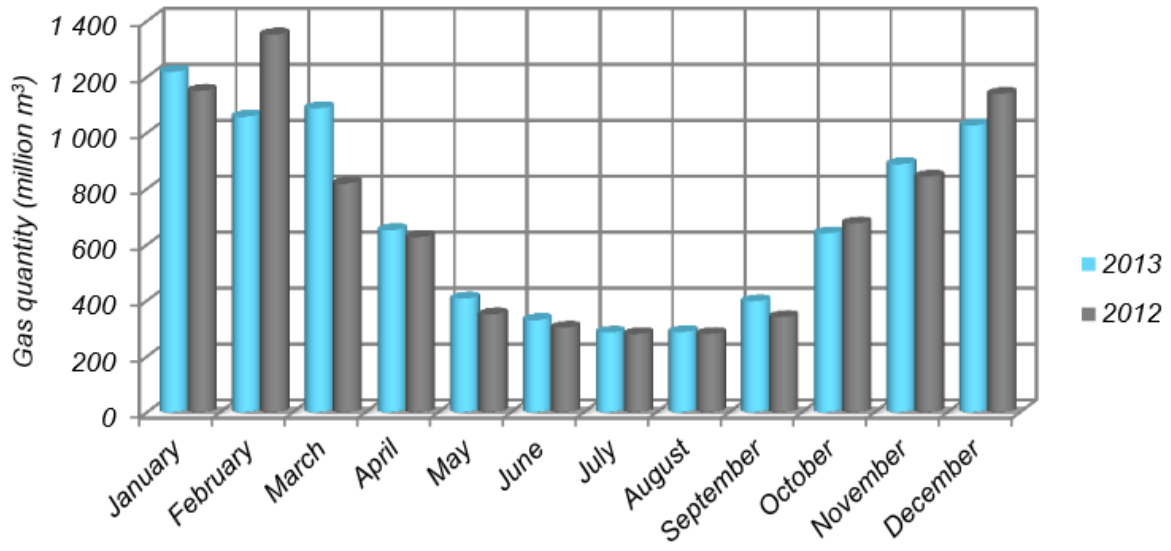
Java, underground gas storage, object-oriented programming, framework, graphical user interface.

Obsah

Seznam obrázků, grafů a tabulek:

1 Úvod

Technologie skladování plynu se rozvíjí v souvislosti s využíváním plynů v průmyslu i komunální sféře a s požadavky na zvětšování skladovací kapacity pro vykrývání sezónních i denních špiček. Zásobování plynem nelze realizovat metodou „just in time“, jak je to běžné u jiných komodit. Rozvoj využití plynných paliv jako ekologicky nezávadného zdroje energie má v posledním desetiletí značně stoupající tendenci ve všech průmyslově vyspělých státech.



Graf 1: Spotřeba plynu v letech 2012-2013 [1]

Distribuce plynných paliv s nepravidelnou roční odběrovou křivkou s výraznou kulminací spotřeby v zimních měsících, graf 1, se stala jedním z nejzávažnějších problémů, na jehož vyřešení závisí provoz podniků, služeb a domácností. Zemní plyn se tak stává sezóní surovinou a tím i možným nástrojem politicko-ekonomického nátlaku. Důležitou skutečností je i lokalizace světových zásob plynu, přičemž nejvýznamnější současné zdroje zemního plynu se nacházejí v politicky nestabilních oblastech. v České republice se nacházejí pouze malá ložiska zemního plynu, která zdaleka nemohou pokrýt požadovanou potřebu (pouze 1%). Nezbytou nutností je tak doprava plynu a jeho skladování, respektive vytváření sezónních zásob, ze kterých je možno krýt špičkové odběr, popřípadě výpadky v dodávkách od producentů, ať již jsou způsobeny jakýmikoliv důvody. k zajištění dostatečného množství plynu v období sezónních nebo denních špičkových odběrů se jeví jako nejvhodnější metoda jeho podzemního skladování.

Provoz podzemních zásobníků předpokládá vysoce sofistikovaný způsob budování technologického celku a manipulace s médiem. Při těchto činnostech je zcela nezbytné využívat metody numerického modelování ať již v rámci budování zásobníku, ale především při jeho provozu.

2 Základní principy ukládání plynů do podzemních zásobníků

2.1 Vlastnosti plynů

Základní vlastností využívanou pro skladování plynu je jejich stlačitelnost. Ideální plyn je dokonale stlačitelný bez vnitřního tření a je využíván pro popis mechanických a termodynamických vlastností plynů. Reálné plyny se svými vlastnostmi mírně liší, přičemž ideálním plynům se přibližují při dostatečně vysoké teplotě a nízkých tlacích. Pro termodynamické děje v plynech platí stavová rovnice ideálního plynu:

$$\boldsymbol{pV = NkT} \qquad (1)$$

kde $\boldsymbol{p}$ je tlak plynu, v je objem, $\boldsymbol{N}$ celkový počet částic plynu, $\boldsymbol{T}$ termodynamická teplota a k Boltzmannova konstanta (vyjadřuje množství energie potřebné k zahřátí jedné částice ideálního plynu o jeden kelvin). Při změnách objemu plynu tak dochází k změnám jeho teploty, které lze charakterizovat jako polytropický děj (krajními případy jsou adiabaticky a izotermický děj), jelikož u reálných podmínek nelze systém izolovat od okolí. Obecně při expanzi plynu teplota klesá a při jeho stlačování stoupá. Tomu musí být přizpůsobena technologie a prováděno chlazení a dohřev plynu.

Zemní plyn těžený na ložiscích je směs plynů s převažujícím metanem (CH_4), příměsí dalších plynných uhlovodíků (ethan – C_2H_6, propan – C_3H_8, butan – C_4H_{10}), oxidu uhličitého, dusíku, sirovodíku a stopami vzácných plynů a kyslíku. Důležitou vlastností zemního plynu vedle jeho hořlavosti je i výbušnost při smísení s kyslíkem. Na některých ložiscích může podíl nehořlavých plynů dosahovat i větších hodnot a plyn je tak nutno před jeho použitím nebo transportem upravovat.

2.2 Zdroje zemního plynu

Metan je v přírodě produkován několika hlavními způsoby. Na povrchu nebo mělce pod povrchem vzniká bakteriálním rozkladem organické hmoty (bioplyn, skládkový plyn a podobně), přičemž obsah metanu závisí na vlhkosti substrátu a přístupu kyslíku. Při vyšší vlhkosti a omezeném kontaktu s atmosférou dochází k metanogenezi (produkci metanu), za opačných podmínek k acidogenezi, kdy je organická hmota rozkládána na oxid uhličitý a dusík. v hlubších zónách zemské kůry dochází k termogenické přeměně pohřbené organické

hmoty. v závislosti na teplotě prostředí vzniká ropa (při nižších teplotách) nebo zemní plyn (při vyšších teplotách), který pak migruje horninovým prostředím. Metan je také poměrně běžnou součásti ložisek uhlí, kde vzniká při karbonizaci rostlinných zbytků a je adsorbován na uhelné sloje. Má velmi nízkou příměs dalších plynu a komplikuje těžbu z důvodu jeho výbušné směsi s kyslíkem. Posledním typem vzniku metanu je jeho anorganická syntéza během tuhnutí magmatu, tento typ metanu nemá průmyslový význam.

2.3 Oxid uhličitý

Ukládání oxidu uhličitého do podzemních zásobníků je jednou z možných cest snižování jeho emisí do ovzduší a tím i předcházení dopadu činností člověka na globální oteplování planety Země. Má však také řadu odpůrců poukazující na agresivitu tohoto plynu a dlouhodobou neudržitelnost těsnosti zásobníků. v současné době již k ukládání oxidu uhličitého do podzemních struktur dochází, avšak nikoliv z důvodu zabránění jeho emisí do ovzduší, ale z důvodů snížení nákladů na dopravu zemního plynu a zlepšení výtěžnosti ložisek. Jedná se o ložiska těžená z ropných plošin v Norsku, s vyšším obsahem oxidu uhličitého. Zemní plyn je dopravován ve stlačeném stavu speciálními tankery. Vyčištěním zemního plynu od nespalitelných příměsí se snižuje objem přepravované suroviny a tak je ekonomicky výhodnější provádět čištění již na místě těžby. Vhodně uložené odpadní plyny pak také umožňují zvýšit efektivitu těžby tím, že vytlačují surovinu z okraje ložiska do centrální těžené části.

Využití ukládání oxidu uhličitého v podmínkách České republiky je však vysoce nepravděpodobné. z hlediska nakládání s tímto plynem by se však jednalo o obdobný technologický postup jako v případě zemního plynu (bez nutnosti instalovat technologie zpětné těžby), přičemž oxid uhličitý je ve směsi s kyslíkem nevýbušný.

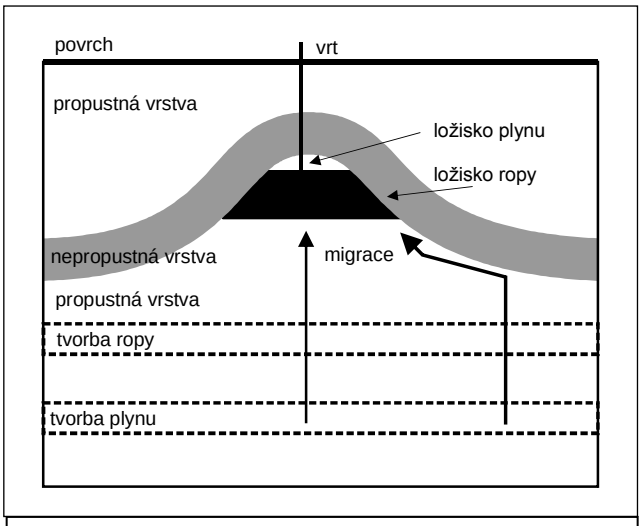
2.4 Základní možnosti ukládání plynů

Plyny mohou být ukládány do vybudovaných technologických zařízení na povrchu - plynojemů. Jedná se většinou o tlakovou nádrž a technologii na stlačování plynu. Toto ukládání má své výhody a nevýhody. Především výstavba těchto úložných kapacit je drahá a neopomenutelné je i bezpečnostní riziko havárií, popřípadě i cíle teroristických útoků. Tyto

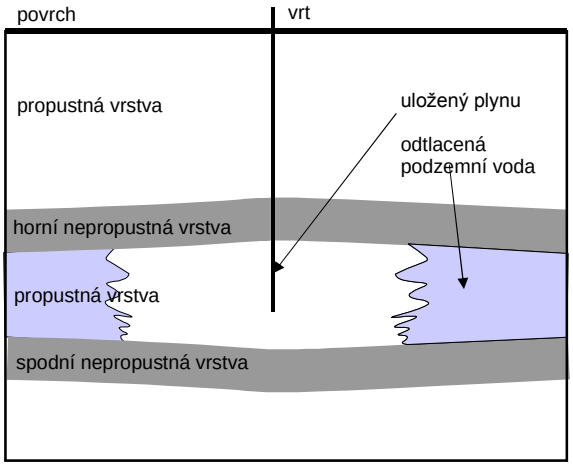
kapacity nejsou reálně využitelné pro ukládání sezónních rezerv zemního plynu, a jsou používány jako pohotovostní zásoba plynu.

Pro uložení velkých objemů zemního plynu, jsou využívány vhodné struktury povrchové části zemské kůry. Jako technicky nejjednodušší lze takto upravit vytěžená ložiska zemního plynu nebo ropy. Tato ložiska vznikla tak, že zemní plyn (ropa) migrovala z místa svého vzniku horninovým prostředím a byla zachycena ve vhodné horninové struktuře. Takováto struktura se nazývá ropná nebo plynová past. Při tomto procesu jsou využity základní fyzikální vlastnosti horninového prostředí i migrujícího média.

Plyn nebo kapalina mohou migrovat pouze propustným horninovým prostředím. Propustnost je určována efektivní porozitou (existencí propojených pórů mezi zrny horninové matrice) nebo puklinatostí (systémem propojených puklin horniny). Vyšší propustnost vykazují nezpevněné sedimenty (písky štěrky) a pískovce. Sedimenty s vysokým obsahem jílu neobsahují spojité póry ani pukliny a jsou pro plyn a kapaliny (voda, ropa) nepropustné. Plyn je litostatickým tlakem (tlakem horninových vrstev) vytlačován k zemskému povrchu. Ropa má nižší hustotu než voda a tak stoupá k zemskému povrchu. Pokud narazí na nepropustnou vrstvu je postup zastaven a dochází k její akumulaci. Vrstvy hornin jsou zvrásněny a vytváří antiklinály (místa vyklenutí) a synklinály (místa poklesnutí). Uhlovodíky tak migruji směrem k povrchu a jsou koncentrovány pod vrcholem antiklinály. Idealizované schéma tohoto procesu je uvedeno na obrázku 1. Vytěžením ložiska, tedy odčerpáním ropy a zemního plynu, se na uvolněné místo dostane podzemní voda. Ta je většinou nevhodná pro využití, protože obsahuje soli a zbytky ropných uhlovodíků. Takové místo pak může být využito pro znovuukládání zemního plynu, pokud není těžbou porušena nepropustná vrstva. Případně netěsnosti způsobené vrty lze technicky zajistit.



Obrázek 1: Schéma ložiska plynu strukturního typu



Obrátek 2: Schéma zásobníku plynu aquiferového typu

Podobné geologické struktury lze využít i v případech, že neobsahovaly ropu nebo zemní plyn. Nezbytná je však přítomnost nepropustné vrstvy a struktury antiklinály nebo propustné vrstvy uzavřené dvěma nepropustným vrstvami, kde se vtláčený plyn zachytí a nemůže unikat mimo toto kontrolované pásmo. Takovéto struktury bývají obvykle vyplněny podzemní vodou. Pokud je struktura hlouběji uložena, pak je podzemní voda stlačena litostatickým a hydrostatickým tlakem, tj. tlakem nadložních hornin a sloupce vody. Po provrtání izolační vrstvy může být voda vytlačena až na zemský povrch. Tento parametr je důležitý z důvodu možného stlačení plynu. Voda ve struktuře způsobuje protitlak. Plyn tak musí svým tlakem vytěsnit vodu z pórů horniny a tím je uvolnit pro ukládání.

Pro ukládání plynu lze také využít přírodní struktury, které neobsahují podzemní vodu. Jedná se o masivní horniny nebo prostory vytěžené z jiného důvodu. Typickým příkladem jsou uhelné, solné nebo rudné doly. Sůl a uhlí tvoří v zemské kůře sloje a sůl také tlakem hornin může být vytlačena a vytvářet deformační pně. Po těžbě soli zůstávají uvolněné rozsáhlé prostory, přitom okolní horniny jsou většinou nepropustné a bez přístupu vody. Pokud se takto vytěžené ložisko utěsní, tedy se odstraní veškeré netěsnosti vytvořené člověkem, vznikne dutina ve skalním masivu (kaverna), kterou lze vyžít pro ukládání plynu. Obdobná situace může nastat u uhelných a u rudných dolů, kde se těžilo uhlí nebo kovy pro průmyslové využití.

Poslední možností je vybudovat podzemní zásobník v kompaktním skalním masívu, třeba v žule, který neobsahuje netěsnosti (pukliny). Je však nutno vytěžit horninu a vytvořit tak umělou dutinu ve skalním masívu. Je zřejmé, že těženou horninu nelze ekonomicky výhodně uplatnit na trhu a náklady na tento typ zásobníku budou vysoké. Výhodou je však zpřístupnění prostoru a důkladné utěsnění zásobníku, což u přírodních struktur je vyloučeno a u prostor po odtěžených surovinách bývá poněkud složitější. Manipulace s plynem v kavernových zásobnících je technicky výhodnější.

2.5 Základní parametry úložišť plynů v přírodních strukturách

U přírodních struktur v sedimentárních horninách se jako úložný prostor využívá pórovitost popřípadě puklinatost hornin, tedy drobné volné prostory mezi zrny sedimentu nebo puklinami pevnějších hornin. Tyto prostory jsou běžně vyplněny vodou. Stanovovány jsou tak základní parametry zásobníku:

Kolektorská hornina – je horninová struktura schopná pojmout vtláčený plyn. Jedná se o propustné horniny, které plyn akumulují ve svých pórech. Její vlastnosti jsou charakterizovány především propustností a pórovitostí.

Izolátor – je horninová struktura tvořená nepropustnou horninou, která tvoří přírodní fyzikální barieru zabraňující úniku vtláčeného plynu. Její vlastnosti jsou charakterizovány propustností.

Porozita – je objem póru v hornině. Udávána je v % a představuje teoretický prostor přístupný pro vtláčení plynu. Porozita se udává jako celková (objem všech póru) nebo efektivní (objem pórů vzájemně propojených). Některé póry totiž mohou být uzavřeny mezi zrny minerálů a nemusí komunikovat s ostatními póry. Velikost této veličiny závisí na typu horniny. Nejvyšší efektivní porozitu vykazují štěrkopísky a nejnižší jíly. u hornin hlouběji uložených pak jistou roli hraje i litostatický tlak, který způsobuje kompakci sedimentů, tedy i snižování porozity.

Propustnost – je schopnost horniny propouštět kapalinu nebo plyn. Je funkcí rychlosti šíření kapaliny vlivem hydraulického gradientu (spádu hladiny) nebo hydrostatickho tlaku a viskozity kapaliny. u plynů je funkcí rychlosti šíření vlivem tlaku. Udává se v jednotkách $m.s^{-1}$.

Retenční schopnost – je schopnost horniny pojmout určité množství kapaliny nebo plynu. Udává je v jednotkách kg/m^3. u kapalin je v podstatě funkcí efektivní pórovitosti, jelikož kapaliny jsou nestlačitelné. u plynů je situace poněkud složitější a retenční schopnost je vedle pórovitosti horniny určována tlakem plynu. v přírodních strukturách nelze tlak plynu zvyšovat nad určitou mez stanovenou podmínkami lokality.

Celková kapacita zásobníku – je množství plynu, jež je možnou do zásobníku uložit. Udává se ve standartních kubických metrech Sm^3 (m^3 za teploty 15 °C, a absolutního tlaku: 1.01325 bar), u přírodních zásobníků proti sobě působí dva základní parametry a to

hydrostatický tlak podzemní vody přítomné v horninové struktuře a tlak vtláčeného plynu. Se zvyšováním hloubky zásobníku roste i hydrostatický tlak vody a tím roste i možný tlak uskladněného plynu. Zároveň je vytlačována podzemní voda a rozšiřován úložný prostor zásobníku. Úložný prostor nelze zvyšovat nad hraniční geometrii struktury, jelikož by docházelo k úniku plynu mimo tuto strukturu.

2.6 Základní parametry úložišť plynů v umělých strukturách

V umělých strukturách je plyn ukládán do kaveren v horninových masívech, ať už byly vyhloubeny účelově pro těžbu surovin nebo jako speciální zásobník plynu. Jediným parametrem vztaženým na přírodní podmínky je celková kapacita zásobníku. Ta je definována dostupným úložným objemem a maximálním pracovním tlakem. v zásobnících se také musí dodržovat rozsah daný úřady, kde každý zásobník má danou maximalní a minimální velikost pracovních tlaků.

2.7 Technické parametry úložišť plynů

Těžební výkon - je nejčastěji vyjádřen jako míra množství plynu, které může být denně vytěženo ze zásobníku. Většinou se měří v Sm^3/den. Těžební výkon podzemního zásobníku je proměnná závislá na mnoha faktorech (množství plynu v zásobníku v určitou dobu, tlak uvnitř zásobníku, schopnost komprese zásobníku, vlastnosti povrchních zařízení spojených se zásobníkem a dalších faktorech). Těžební výkon je závislý na celkovém objemu plynu v zásobníku. Je nejvyšší v okamžiku, kdy je naplněna kapacita zásobníku a postupně s vyčerpáním zásob klesá, jelikož klesá i tlak v zásobníku. Platí tedy přímá úměra mezi těžebním výkonem a zásobou plynu

Vtláčený výkon - je podobně jako těžební výkon míra množství plynu, které může být denně vtlačeno do zásobníku. Obvykle se udává v jednotkách Sm^3/den. Na rozdíl od těžebního výkonu je vtláčený výkon nejnižší, když je zásobník plný. Platí nepřímá úměra mezi vtláčením výkonem a zásobou plynu.

Poduška – je minimální zásoba plynu v zásobníku, která je nezbytně nutná pro provoz technologie. Jedná se o zásobu plynu, kterou nelze využít pro zásobování.

Provozní kapacita zásobníku – je množství plynu, se kterým lze manipulovat, tedy využít je pro ukládání a těžbu.

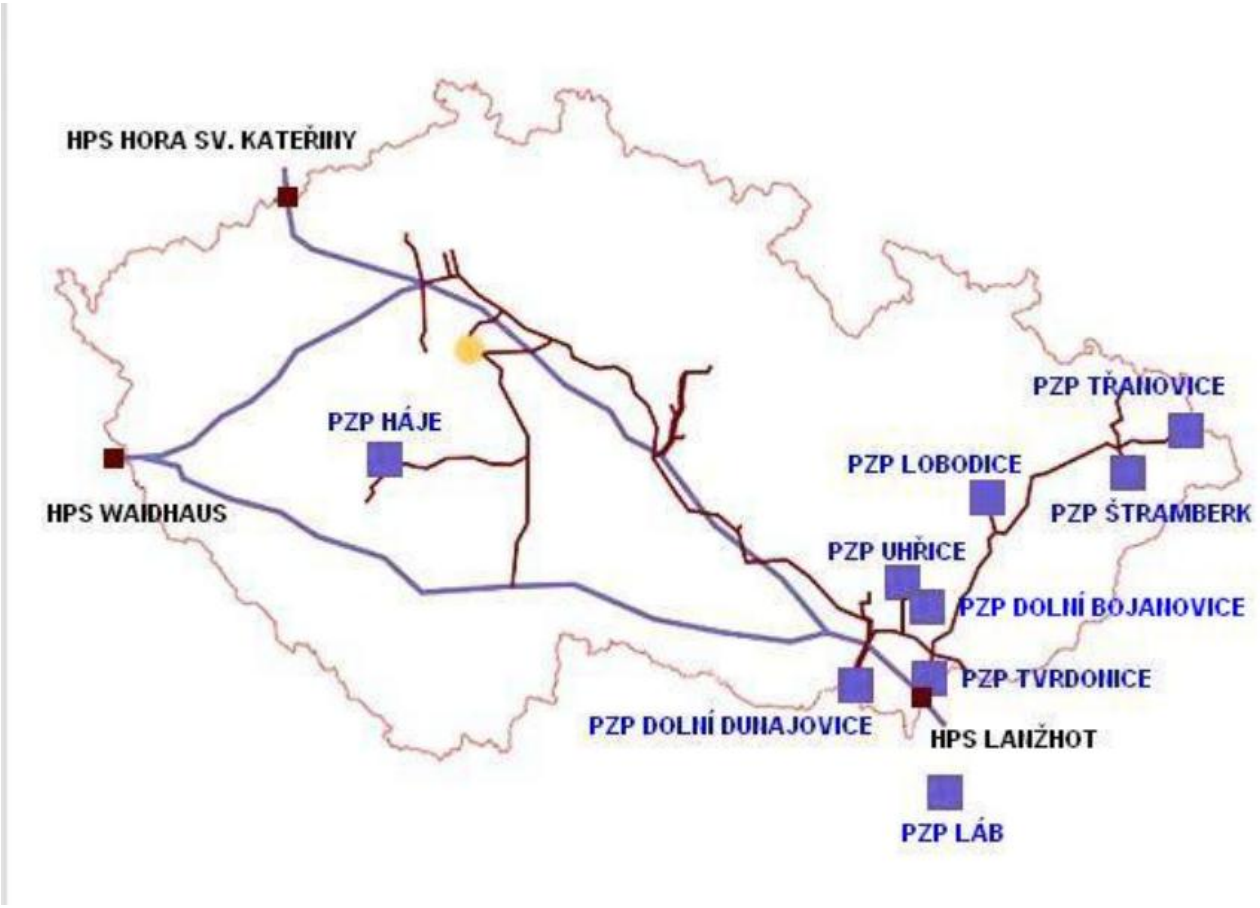
Aktivní skladovací kapacita - charakterizuje objem plynu, který může být odtěžen, aniž by zásobník byl ohrožen z hlediska vtláčení a zabezpečení požadovaných nároků na příští těžební sezónu

Špičkový výkon - je maximálním výkonem podzemního zásobníku, který může být realizován na základě daných provozních podmínek. Vyrovnávání odběrových špiček je technicky výhodnější z kavernových zásobníků.

2.8 Příklady zásobníků provozovaných v České Republice

Podle informací převzatých ze serveru www.mojeenergie.cz byl první tuzemský podzemní zásobník plynu uveden do provozu v Lobodicích v roce 1965. v té době byl využíván pro skladování svítiplynu. Ve zmodernizované podobě funguje dodnes. v současnosti celková tuzemská kapacita podzemních zásobníků plynu dosahuje hodnoty 3,072 mld.m^3. Celkový maximální denní teoretický těžební výkon všech tuzemských zásobníků tedy činí 58,7 mil.Sm3 zemního plynu**.** Uvedené hodnoty zajišťují České republice v poměru k celkové tuzemské spotřebě zemního plynu (uskladňovací kapacita se pohybuje na úrovni 35,43 % celkové roční spotřeby) jedno z čelných míst mezi státy EU.

Nejvíce podzemních zásobníků plynu provozuje společnost RWE Gas Storage. Šest jejích zásobníků disponuje úhrnnou kapacitou 2,321 mld.Sm3 při maximálním denním těžebním výkonu 43,7 mil.Sm3 . Jedná se o podzemní zásobníky plynu Lobodice, Tvrdonice, Štramberk, Dunajovice, Háje a Třanovice. Další podzemní zásobník plynu v Uhřicích vlastní a provozuje společnost Moravské naftové doly a posledním zásobníkem provozovaným v tuzemsku je zásobník v katastru obce Dolní Bojanovice, vlastněný společností SPP Bohemia. Tento zásobník je však zatím na základě smluv využíván pouze pro Slovensko, s jehož plynárenskou soustavou je přímo propojen. Rozmístění podzemních zásobníků plynu v České republice a jejich parametry je uvedeno na obrázku 3 a tabulce 1.



Obrázek 3: Umístění podzemních zásobníků v ČR [2]

	Lobodice	Tvrdonice	Štramberk	Dunajovice	Háje	Třanovice	Uhřice	Dolní Bojanovice
vlastník	RWE Gas Storage	RWE Gas Storage	RWE Gas Storage	RWE Gas Storage	RWE Gas Storage	RWE Gas Storage	MND	SPP Bohemia
typ	aquifer	plynové ložisko	plynové ložisko	plynové ložisko	kaverna	plynové ložisko	plynové ložisko	plynové ložisko
rok uvedení do provozu	1965/91	1975	1983	1989	1998	2001	2001	1999
provozní zásoba určená pro obchod s plynem (mil.m^3)	177	460	480	900	59	240	180	576
maximální denní těžební výkon (mil.m^3)	3,6	7,0	7,0	16,0	6,0	4,1	6,0	9,0
hloubka obzoru (m)	440	1260	575	1048	950	440	1750	1660

Tabulka 1: Základní parametry podzemních zásobníků v ČR [2]

2.8.1 Zásobník Dolní Dunajovice

Jedná se o zásobník, který je vybudován na plynonosné struktuře na Jižní Moravě. Strop korektorských hornin byl naražen v hloubce 1050 m pod povrchem. Po odtěžení části zásob zemního plynu bylo rozhodnuto o využití geologické struktury pro ukládání zásob plynu a byl tak vybudován . největší zásobník plynu v České republice, který byl uveden do provozu již v roce 1989. Maximální denní těžební výkon činí 16 mil m^3, což je cca 1,77% provozní zásoby. Těžební výkon je právě určován hloubkou uložení plynonosné struktury.

Obrázek 4: Letecká fotografie zásobníku Dolní Dunajovice [4]

Zásobník [3] je vybaven technologií na vtláčení plynu, která se skládá ze vstupních filtrů pro čištění plynu, měřením množství plynu před vtláčením do zásobníku, čtyř kompresorů, chladiče plynu, odlučovače oleje pro odloučení oleje z plynu, sběrným střediskem s měřícími a regulačními tratěmi sond, provozními sondami, plynovody s propojovacími kolektory. Technologie pro těžbu se skládá z Provozních sond a zařízení pro nástřik metanolu, sběrného střediska s měřícími a regulačními tratěmi sond, zařízení pro separaci (odvodnění ložiskové vody) a ohřev plynu (zvýšení teploty plynu před snížením jeho tlaku), sušením plynu (odstraněním vodních par z plynu) a regenerace glykolu, výstupních filtrů pro čištění plynu, měřením množství plynu před jeho expedicí do přepravní soustavy, plynovodů s propojovacími kolektory

3 Objektově orientované programování

Objektové programování (OOP) je styl programování. Jsou úlohy, které lze přes objekty řešit velmi efektivně a naopak – jsou věci, na které se hodí například funkcionální přístup. OOP spočívá v rozdělení celého programu na dílčí části a postupném řešení těchto menších části programu. Tím, že problém rozdělíme na menší části, stává se celkový projekt snáze řešitelný, snáze analyzovatelný a snáze udržovatelný. Důležité je, že každá část spravuje pouze to, co jí bylo přiděleno.

Objekt je struktura, která obsahuje určitě proměnné a určité funkce, které se vztahují k myšlence objektu. Například pokud bychom měli objekt Bod, mohl by obsahovat proměnné X a Y, reprezentující x-ovou a y-ovou souřadnici. a teď by to chtělo vymyslet nějakou funkci, která by se k tomuto objektu hodila. Co můžeme dělat s bodem/souřadnicí? Bod můžeme třeba posouvat, takže objekt Bod by mohl mít u sebe funkci Posun(), která by daný bod posunula o nějaké ty pixely.

3.1 Výhody OOP

Objektově orientované jazyky se zaměřují na předměty, které uživatel vnímá a ty pak použijí jako základní jednotky. Objektově orientované technologie má mnoho výhod:

- Snadnost v návrhu softwaru, je možné myslet spíše v problému prostoru, než v bitech a bajtech stroje. Pracuje se s pojmy, na vysoké úrovni a abstrakcí. Snadný design vede ke zvýšení produktivity vývoje software.
- Snadné na údržbu softwaru: objektově orientovaný software je srozumitelnější, a proto lze snadněji testovat, ladit a udržovat.
- Opakovaně použitelný software: nemusíte znovu psát stejné funkce pro různé situace. Nejrychlejší a nejbezpečnější způsob, jak rozvíjet nové aplikace, je použít existující kódy - plně prověřené a osvědčené.

3.2 Dědičnost

Dědičnost nám dovoluje vyvoření hierarchie tříd, které se postupně z generace na generaci rozšiřují. Dědičnost se používá v případech, kdy se chceme vyhnout opakování kódu.

Dědění je významný nástroj pro vytváření opakovaně využitelných programových modulů. Programový modul by měl být zároveň uzavřený (pro jeho použití není potřeba nic přidávat) a otevřený (uživatel by měl mít možnost nevhodné věci modifikovat a nové přidávat). Dědění umožnuje ponechat vše potřebné ze základní třídy, dodat chybějící věci a změnit to, co se nám nelíbilo.

3.3 Polymorfizmus

Polymorfiymus je jiným slovem vícetvarost, mnohotvarost. Jedná se o schopnost využívat v programovém textu stejnou syntaktickou podobu metody s různou vnitřní interpretací (voláme stejnou metodu, která dělá něco jiného). Polymorfizmus se používá, když má nějaká třída více potomků, ke kterým přistupujeme jednotlivým způsobem. Využití polymorfismu je vhodné u abstraktních tříd, nebo rozhraním, kde nadefinujeme abstraktní metody vstupnímy parametry a návratovým typem a při použití je překryjeme (implementujeme). Polymorfismus je také možné uplatnit u neabstraktních tříd.

3.4 Zapouzdření

Třída je datový typ, který v sobě zapouzdřuje data a funkce (*atributy* a *metody* v názvosloví OOP). Atributy můžeme chápat jako proměnné primitivních datových typů, nebo instance jiných tříd. Metody umožňují funkcionalitu třídy a provádí operace nad jejími atributy. Systém zapouzdření je dale vybaven přístupovými modifikátory, podle kterých jsou či nejsou metody a atributy přístupné z jiných tříd. Tento způsob organizace kódu je bezpečnější, než způsob známý z C nebo Pascalu, kde jsou data i funkce obvykle globální a přístupná z jakéhokoliv místa programu.

3.5 OOP v Javě

3.5.1 Třídy a instance

V Javě je třída definování objektů stejného druhu. Jinými slovy, třída je plán, šablona, nebo prototyp, který definuje a popisuje vlastnosti a chování společné pro všechny objekty stejného druhu.

Instance je realizace konkrétní položky třídy. Všechny instance třídy mají podobné vlastnosti. Třída se skádá z následujících položek:

- Jméno (nebo identity): identifikuje třídu.
- Proměnné (nebo atribut, stav, pole): obsahuje atributy třídy.
- Metody (nebo chování, funkce, operace): obsahuje chování třídy.

Jinými slovy, třída zapouzdří atributy (data) a chování (operace, které fungují na základě údajů).

3.5.2 Definice tříd

V Javě, používáme klíčové slovo class, které nám definuje třídu. Podle konvence pojmenování by měl název třídy být podstatné jméno nebo jmenná fráze skládající se z několika slov. Všechna slova by také měli začínat velkým písmenem.

3.5.3 Vytváření instancí tříd

Chceme-li *vytvořit instanci třídy,* musíme deklarovat identifikátor instance (jméno instance) a konstrukt instance (tj alokace paměti pro instanci a inicializovat instanci) pomocí „new“ operátoru.

3.6 Kolekce v Javě

Třída Collections se skládá pouze ze statických *metod*, které pracují na kolekcích nebo je vracejí. Přestože můžou být kolekce použity přímo, obvykle pracujeme s jejich pod-rozhranímy. k realizaci frameworku byli použity kolekce implementující rozhraní List, Map a Iterator.

3.6.1 Rozhraní List

Rozhraní List je seřazená kolekce. Uživatel tohoto rozhraní má přesnou kontrolu nad umístěním každého vloženého prvku. Uživatel může přistoupit k prvku na základě jeho indexu a může vyhledávat prvky v listu.

Narozdíl od množin, seznamy umožňují duplicitní prvky. To platí i pro prvky null.

Rozhraní List poskytuje čtyři metody pro indexový přístup k prvkům. Seznamy začínají indexem nula. Tyto metody však zabírají čas v závislosti na velikosti indexu. Procházení senamu je proto výhodnější provádět cyklem, na rozdíl od indexově založených metod. Dále rozhraní poskytuje dvě metody pro hledání, vkládání a mazání prvků.

ArrayList je implementace rozhraní List, která umožňuje pole s dynamickou velikostí.

3.6.2 Rozhraní Map

Asociativní pole je v informatice abstraktní datový typ složený z párů klíčů a hodnot, kde se každý klíč může objevit pouze jednou v kolekci. Operace spojené s tímto typem umožnují:

- Přidání nových párů do kolekce
- Odebrání párů z kolekce
- Změnu hodnot pro existujících párů
- Vyhledání hodnoty určitého klíče

V javě lze tento typ vytvořit za použití rozhraní Map, které obsahuje tři implementace: HashMap, TreeMap a LinkedHashMap. Ve vývoji frameworku byla použita pouze třída TreeMap.

Třída TreeMap implementuje rozhraní map. Jedná se o seřazenou mapu podle klíčů nebo rozhraní Comparator přiloženého při vytváření mapy. Treemapy zaručují časovou náročnost log(n) pro operace containsKey, get, put a remove. Využívají algoritmů Cormen, Leiserson a Rivest.

3.6.3 Rozhraní Iterator

Rozhraní Iterator umožnuje průchod nad kolekcí. Iterator zastupuje abstraktní třídu Enumeration ve frameworku Java Collections. Iterator se liší od enumerací ve dvou bodech:

- Iterator umožňuje odběr prvků z kolekce při iteraci.
- Vylepšuje názvy metod.

4 Specifikace požadavků

Hlavním úkolem práce je vytvoření frameworku. Pro jeho realizaci byl vybrán programovací jazyk java ve stylu objektově orientovaného programování. Na framework jsou kladeny následující požadavky.

4.1 Uložení dat

Framework musí být schopen přečíst data ze skupinových, hlavičkových a datových souborů. Tyto data následně uloží do uspořádané struktury objektů. Data reprezentují hodnoty měření a je k nim také potřeba uchovávat jejich čas, kdy byli naměřeny, název a jednotky měřené veličiny a název sondy u které měření proběhlo. Pro dodržení tohoto požadavku se měření ukládají do objektu, který obsahuje název veličiny, název jednotek a asociativní pole se záznamem měření.

4.2 Umožnění přístupu k uloženým datům

Výběr dat probíhá na základě uživatelem zadaných parametrů. Konkrétně musí framework umožnit přístup podle časů měření, které určí hledaný časový úsek. Za tímto účelem bylo navrženo uchovávání dat v datové struktuře *TreeMap*, kde jsou data automaticky uspořádany podle časů měření. Data je také potřeba výbírat podle určitých sond, nebo skupin sond, do kterých sama náleží.

4.3 Dodržení hierarchie zásobník – skupina sond – sonda – veličina

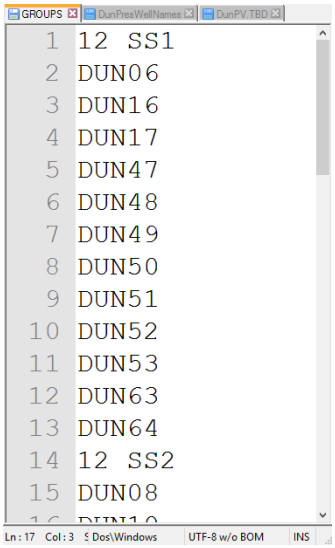
Objektově orientované programování bylo zvoleno, protože umožňuje snadnější porozumění uživatelem. Pro jeho nejlepší využití musí framework použít objekty, které představují praktické věci. k vyjádření veličin slouží objekty typu *Quantity*. Sondy jsou představeny jako objekty *Well*, kde jejich skupiny jsou zastoupeny pomocí stringových proměnných. Objekty *Storage* pak vyjadřují zásobníky.

4.4 Umožnění základních manipulací s daty

Framework by také měl obsahovat metody pro agregace v čase a ve skupině, základní čištění dat, interpolace, výpočet odvozených veličin (kumulativních objemů, statických tlaků atp.).

5 Uložení a popis provozních dat

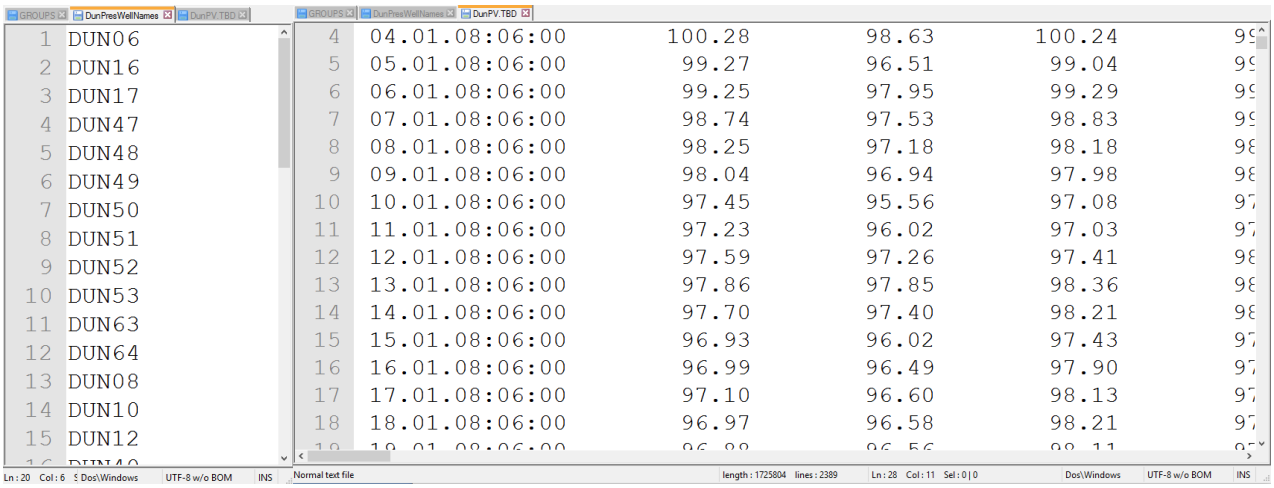
Pro potřeby bakalářské práce byla předána provozní data. Jedná se o data popisující vývoj teploty, tlaku a provozních objemů plynu v období od 1.8.2018 do 15.7.2014. Data jsou zaznamenávána automaticky, čímž jsou eliminovány chyby při odečtech a archivaci dat. Data jsou předávána ve formátu TBD.



Obrázek 5: Skupinový TBD soubor

Rozmístění sond na zásobníku je definováno ve skupinovém souboru, kde je na každém řádku název nové sondy, nebo název nové skupiny sond, do které následující sondy patří. Nová skupina se pozná tak, že začíná číslicí, narozdíl od sondy, která začíná písmenem. Příklad skupinového souboru je zobrazen na obrázku 5.

Vlastní data měření jsou zaznamenány v datovém souboru. v prvním sloupci jsou časy měření a každý další sloupec obsahuje hodnoty měření pro jednotlivé sondy. Jejich pořadí je uvedeno v hlavičkovém souboru. Ukázka datového a hlavičkového souboru se nachází na obrázku 6.



Obrázek 6: Hlavičkový a datový TBD soubor

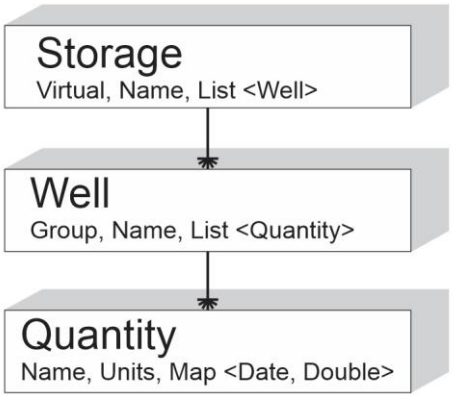
6 Design Frameworku

Následující text se bude věnovat rozvržení frameworku. Jsou zde vypsány třídy a jejich veřejné metody. Framework se skládá z pěti balíčků: *definition*, *functions*, *objects*, *tools* a *main*.

6.1 Objects

Balíček *objects* vlastní třídy na vytváření objektů reprezentujících reálné zásobníky, sondy a veličiny. Jsou zde také metody sloužící pro ukládání, načítání nebo hledání datových zánamů. Strukturu objektů je možné videt na obrázku 7. Konkrétně se jedná o:

- *Storage* – Představuje zásobník. Obsahuje název zásobníku, počet skupin sond, název virtuálu, do kterého patří a list objektů *Well*
- *Well* – Představuje sondu. Obsahuje název sondy, název skupiny a list objektů *Quantity*
- *Quantity* – Představuje veličinu. Obsahuje název veličiny, název jednotek a hashmapu časů a dat měření.



Obrázek 7: Struktura objektů

6.2 Definition

Tento balíček obsahuje třídy, které se starají o čtení a ukládání dat. Třída *LoadTBD* čte data ze souborů typu TBD.

Její statická metoda *createStorage* umožnuje vytvořit objekt *Storage* pomocí skupinového souboru. Na vstup je potřeba zadat název zásobníku a adresu skupinového souboru. Metoda pak na výstupu vrátí vytvořený objekt *Storage*.

Dále třída *LoadTBD* obsahuje statickou metodu *addToStorage* pro přečtení datových souborů a uložení dat do předem vytvořeného objektu *Storage* reprezentujícího zásobník. Pro její zavolání je potřeba mít vytvořené objekty *Storage, Quantity* a adresy datového a hlavičkového souboru. Metoda je typu void a v průběhu uloží data ze souborů do objektu *Storage*.

6.3 Tools

V balíčku *tools* je třída *View*. Statické metody třídy *View* umožňují výběr a zobrazení dat na základě vstupních parametrů. Na jejich vstupu je potřeba objekt *Storage*, reprezentující zásobník ze kterého chceme data vybírat, objekt *Quantity* pro jednotky vybranného měření a parametry definující náš výběr, kterými mohou být názvy sond, názvy skupin sond a časové omezení měření.

6.4 Functions

Zde se nalézají třídy s metodamy pro práci s daty. Metody jsou statické a předpokládají na vstupu výběr dat, kterým je výstup z některé z metod třídy *View*. Konkrétně se jedná o třídu *Aggregate*, která obsahuje metody pro počítání součtů hodnot. Třída *Aggregate* vlastní statické metody *sum*, *cuSum*, *production*, *injection* a *yield*. Výstupem těchto metod je mapa, kde klíče jsou časy měření a hodnoty jsou proměnné typu *Double*.

6.5 Main

K testování a spouštění aplikace byl vyhrazen balíček *main*. Jeho třída *Main* sloužila v průběhu vývoje k hledání chyb a spouštění metod frameworku. Pro demonstraci frameworku bylo navrženo grafické rozhraní s třídami *SelectWindow*, *StorageWindow* a hlavní spouštěcí třídou *StartingApplication*.

7 Použití frameworku

Kapitola použití frameworku se zabývá zacházením s objekty a metodami balíčků *definition*, *functions*, *objects* a *tools*. Dále je zde uvedeno ošetření chyb, které mohou nastat v průběhu provozu programu a datovými typy jednotlivých hodnot. Předpokládané využití frameworku může být rozděleno do pěti kroků. Nejprve je nutné vytvořit strukturu objektů, se kterými budeme pracovat. Dalším krokem je načtení dat z datových souborů do předem vytvořené struktury objektů. Následuje výběr dat na základě názvů skupin, názvů jednotlivých sond nebo časů měření. Takto vybraná data lze pak zobrazit například výpisem na konzoli Nad vybranými daty je také možné provádět matematické operace a vypisovat jejich výsledky.

7.1 Vytvoření objektů podle hlavičkového souboru

Pro vytvoření struktury objektů reprezentujících zásobníky, sondy a skupiny sond je potřeba mít hlavičkový soubor ve formátu TBD. Adresu tohoto souboru společně s názvem zásobníku, který chceme vytvořit je potřeba použít jako vstupní parametry metody *createStorage* příslušné třídy. Pro TBD použijeme metodu *LoadTBD.createStorage*("název zásobníku", "adresa skupinového souboru"). Metoda *createStorage* zkontroluje platnost adresy zadaného souboru. Pokud je adresa neplatná, metoda vypíše na konzoli chybovou hlášku. Metoda předpokládá skupinový soubor ve formátu, kde je na každém řádku název sondy začínající písmenem, nebo název skupiny začínající číslem. Výsledkem metody *createStorage* je objekt typu *Storage* který představuje zásobník popsaný ve skupinovém souboru.

7.1.1 Načtení dat z datových souborů

Pokud máme již vytvořený objekt *Storage*, soubor s naměřenými daty a datový hlavičkový soubor k datovému souboru. Je možné použít metodu *addToStorage* příslušných tříd balíčku *definition* k přečtení souboru a uložení dat do objektu *Storage*. Nejdříve musíme vytvořit objekt *Quantity*, který představuje fyzikální veličinu měření v datovém souboru. Nový objekt je možné vytvořit následujícím způsobem:

```
Quantity quant = new Quantity(''název veličiny'', "jednotky veličiny'');
```

Tento objekt použijeme jako další vstupní parametr metody *addToStorage* společně s adresou datového a hlavičkového souboru a objektem *Storage* do kterého budeme data ukládat. Metodu v tomto případě zavoláme tímto kódem:

```
LoadTBD.addToStorage(storage, quant, adresa hlavičkového souboru, adresa
```

```
datového souboru);
```

Metoda projde všechny záznamy v souborech a doplní data do příslušných objektů. Pokud budou soubory obsahovat sondu, která zatím nemá svou reprezentaci objektem *Well* vytvořenou, metoda vytvoří nový objekt, do kterého data zapíše a uvědomí o tom uživatele výpisem na konzoly. Takto vytvořený objekt ale nebude patřít do žadné skupiny sond.

7.1.2 Výběr a zobrazení dat

Pokud potřebujeme zobrazit nebo vybrat data pro další operace na základě parametrů jako je název skupiny, název sond nebo časové omezení, je možné použít třídu *View* z balíčku *tools*. Nejjednoduší k použití je metoda *all*, která umožňuje zobrazit všechny sondy daného zásobníku. Nejdříve si musíme zvolit veličinou, kterou chceme znázornit. Její název a jednotky musejí odpovídat údajům použitým při vkládání dat. Data zásobníku je možné vybrat tímto kódem:

```
View.all(storage, quant);
```

Metodu je také možné volat s časovým omezením. v tomto případě je nutné si vytvořit dva javovské datumy. Můžeme k tomu například použít vestavenou třídu *SimpleDateFormat*. Příklad volání časově omezené metody může tedy vypadat následovně:

```
DateFormat format = new SimpleDateFormat("dd.MM.yy:HH:mm");
Date date1 = format.parse("02.01.08:06:00");
Date date2 = format.parse("10.01.08:06:00");
View.all(storage, quant, date1, date2);
```

Také je možnost vybírat data podle názvu skupiny sond, o kterou máme zájem pomocí metody *group*. Zde je potřeba přidat do vstupních parametrů stringový řetězec představující název hledané skupiny:

```
View.group(storage, quant, "název skupiny");
```

Třída *View* také obsahuje metodu *wellNames*, která slouží k výběru dat podle konkrétních názvů sond. Jména sond je potřeba vkládat v textovém řetězci, kde jsou jednotlivé názvy oddělené mezerou, lomítkem nebo jiným oddělujícím symbolem. Použití metody *wellNames* má následující podobu:

```
View.wellNames(storage, quant, "název1 název2");
```

Metody *group* i *wellNames* lze opět použít s časovým omezením obdobně jako u metody *all*. Výsledkem metod *all*, *group* a *wellNames* je treemapa s klíčem javovského času představující čas měření a hodnoutou v podobě listu double hodnot. Takto vytvořenou mapu lze zobrazit na konzoli pomocí metody *viewList*, nebo ji lze použít k datovým operacím z balíčku functions.

7.1.3 Práce s daty

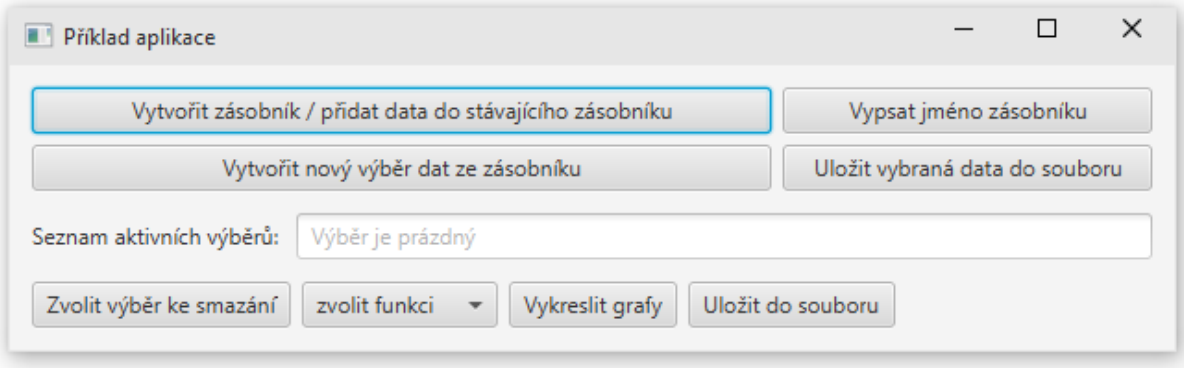
Balíček *functions* slouží k operacím nad daty. Nachází se zde třída *Aggregate*, která obsahuje metodu *sum*. Tato metoda vyžaduje na vstupu treemapu s klíčem datumu javovského formátu a hodnoty ve tvaru listů doubelu. Metoda *sum* vypočítá součet všech hodnot z listu záznamu. Na výstupu je číslo typu double.

Dále třída *Aggregate* obsahuje metodu *cuSum*, která pracuje podobně jako předchozí metoda, ale jednotlivé hodnoty záznamů mezi sebou nesčitá. Provede pouze sumu záznamů a na výstupu se oběví jeden záznam symbolizující hodnoty sum pro každou sondu zvlášť.

Třída *Aggregate* také obsahuje metody *production* a *injection*, které jsou obdobou metody *cuSum*, ale provádějí součty na základě znaménka hodnot měření. *Production* počítá pouze se zápornými hodnotami a *injection* používá pouze hodnoty kladné.

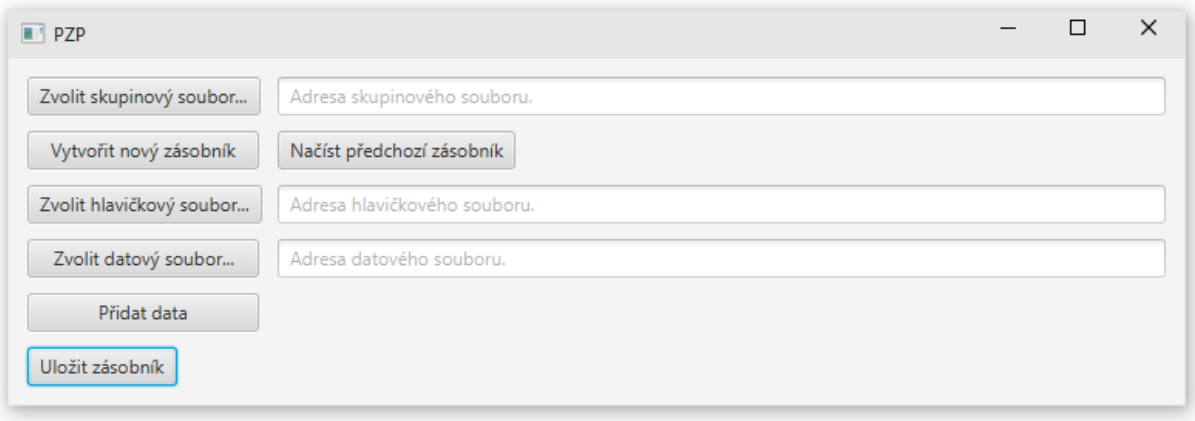
7.1.4 Grafické rozhraní

Pro předvedení frameworku bylo vytvořeno jednoduché grafické rozhraní. s jeho použitím lze jednoduše provést všechny předchozí metody. Po spuštění se oběví okno s možnostmi pro vytvoření zásobníku, výběr dat, uložení výběru dat do souboru a pro provedení matematických operací. Výsledek lze zobrazit pomocí grafu nebo uložit do souboru. Hlavní okno rozhraní je na obrázku 8.



Obrázek 8: Hlavní okno rozhraní

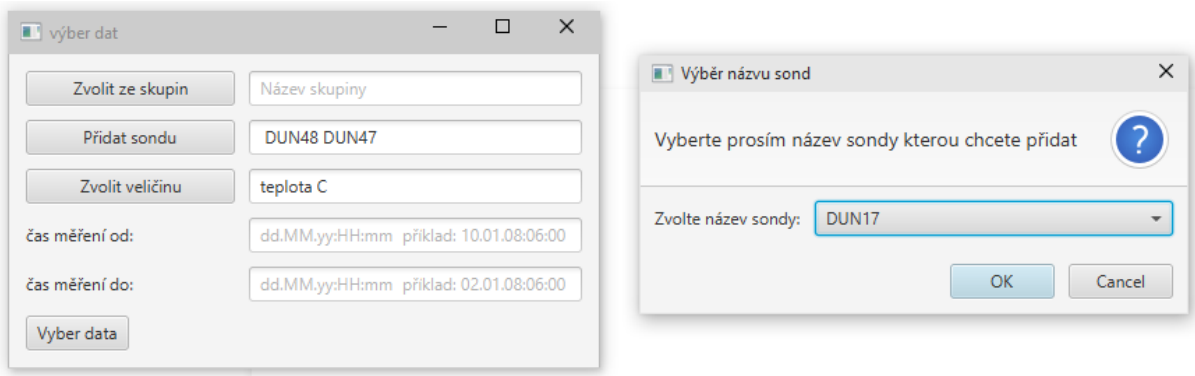
Jako první krok je potřeba vytvořit zásobník ze skupinového souboru. Toho je možné dosáhnout v okně, které se otevře po stisku tlačítka „vytvořit zásobník“. Na obrázku 9 je okno pro vytváření zásobníku ze skupinového souboru a přidávání dat z datového a hlavičkového souboru.



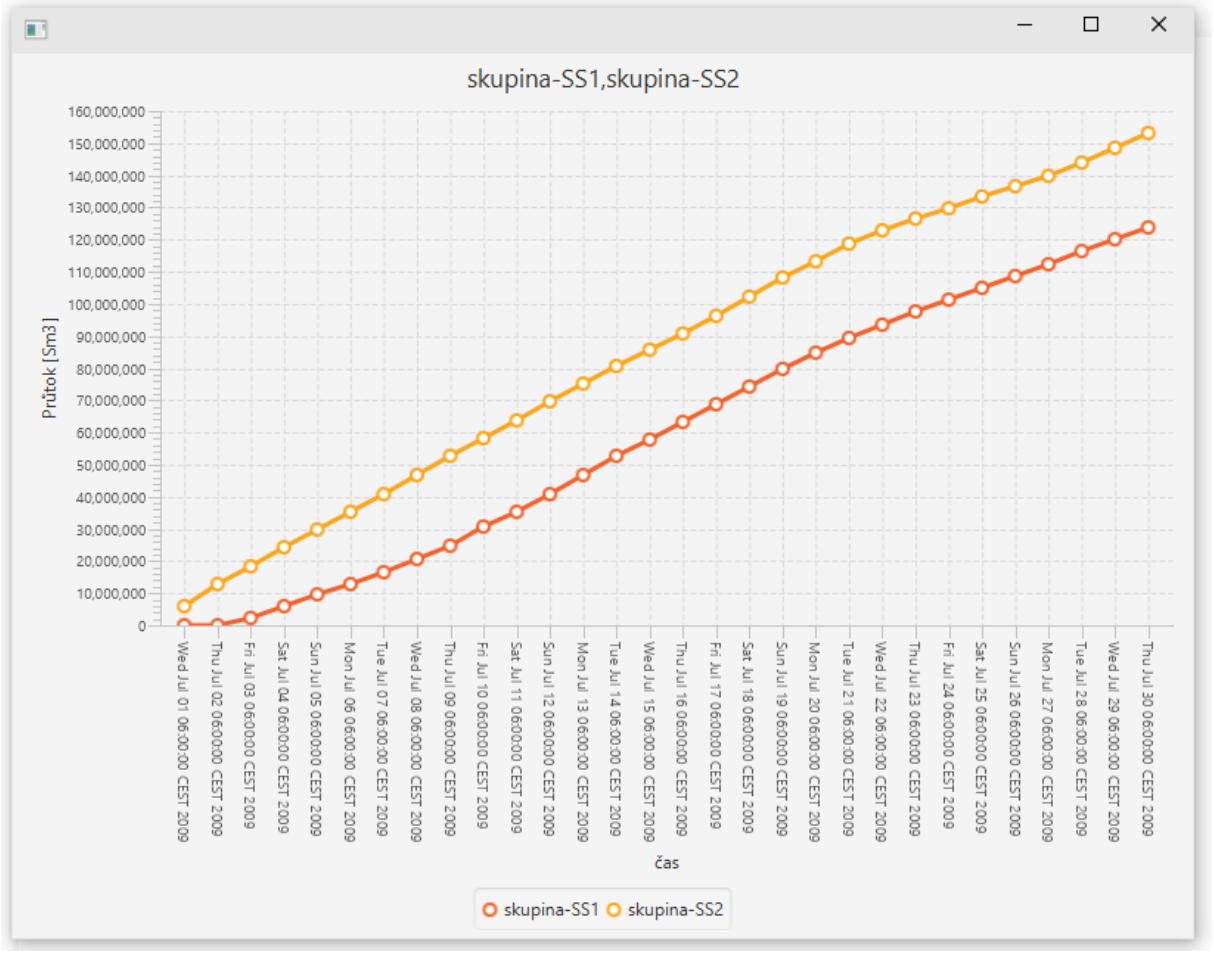
Obrázek 9: Okno pro čtení souborů

Cesty k souborům je možné vybírat pomocí volících tlačítek, nebo je možné je psát přímo. Po zvolení skupinového souboru je potřeba stisknout tlačítko „vytvořit nový zásobník“. Rozhraní se následně zeptá na název zásobníku, po jeho zadaní a stisknutí „ok“ je zásobník vytvořen. Stejným způsobem se dají přidat data z hlavičkového a datového souboru. Po stisknutí tlačítka „přidat data“ se rozhraní zeptá na název veličiny a její jednotky. Následně lze zásobník uložit tlačítkem „uložit zásobník“.

Nyní je potřeba vybrat data. Po stisku tlačítka „vytvořit nový výběr dat ze zásobníku“ se objeví nové okno (obrázek 10), kde je možné zvolit výběrové parametry. Zde je potřeba vyplnit nebo vybrat pouze veličinu a jednotky, ostatní parametry jsou volitelné. Po úspěšném výběru se výsledek znázorní v textovém poli s popiskem „seznam aktivních výběrů“.



Obrázek 10: Okno pro výběr dat



Obrázek 11: Okno grafu výsledku funkce produced

Jednotlivé výběry lze smazat po jednom. Také nad nimi lze provést matematické operace a jejich výsledky zobrazit na grafu, obrázek 11.

8 Programátorský popis frameworku

Tato kapitola obsahuje podrobný popis algoritmů, vnitřních tříd a datových typů.

8.1 Třídy objektů a jejich vnitřní metody

Veškeré instance tříd, které je potřeba inicializovat pomocí operátoru „new“, mohou být nalezeny v balíčku *objects*. Všechny třídy balíčku *objects* obsahují alespoň jeden konstruktor pomocí kterého lze nový objekt vytvořit.

8.1.1 Třída Storage

Třída *Storage* obsahuje stringovou proměnnou *name*, představující název zásobníku. Dále vlastní integerovou proměnnou *groupCount*, která vyjadřuje počet skupin sond v zásobníku. Také má dynamické pole *wellList* z javovské kolekce *List*, který slouží pro objekty typu *Well*. Tyto proměnné je potřeba předávat a měnit mimo třídu, proto třída také obsahuje jejich gettery a settery.

Třída vlastní pouze jeden konstruktor pro vytvoření její instance, který vyžaduje a nastaví proměnnou *name*.

Metoda *getWell* je metodou třídy, která dokáže vrátit objekt *Well* pomocí vstupního textového řetězce představujícího název požadované sondy. Po zavolání metoda vytvoří nový objekt *Well*, který je roven objektu *null*. Následně projde dynamickým polem *wellList* instance která metodu zavolala. k průchodu používá metoda javovského rozhraní *Iterator*. Při každé iteraci pole metoda porovnává proměnnou *name* objektu *Well* s textovým řetězcem ze vstupního parametru metody. Při nalezené shodě se uloží objekt *Well* do předem vytvořené proměnné. Po ukončení vrací metoda proměnnou, ve které se nachází hledaný objekt *Well*. v případě kdy shoda nenastala je na výstupu nulový objekt.

Metoda *getWellNames* slouží k výpisu všech názvů sond v zásobníku. Na vstupu nepotřebuje mít žádný údaj. Prvním krokem inicializuje nové dynamické pole stringových hodnot pomocí kolekce *List*. Následně metoda projde for cyklem pole *wellList* a postupně přidá názvy jednotlivých objektů Well do předem vytvořeného pole stringů, které následně pošle na výstup.

Přidávání objektů *Well* do pole *wellList* je možné pomocí metody *addWell*, která

vyžaduje na vstupu objekt typu *Well*. Metoda má návratový typ void.

8.1.2 Třída Well

Stringové proměnné name a group třídy *Well* představují název sondy a název skupiny sond. Dále má třída dynamické pole objektů *Quantity.* Ve třídě bylo také potřeba zavést gettery a settery těchto proměnných.

Instance třídy lze vytvářet pomocí dvou konstruktorů. První vyžaduje na vstupu stringové proměnné *name* a *group,* které následně uloží do příslušných proměnných instance. Druhý konstruktor vyžaduje pouze stringovou proměnnou *name,* ale na rozdíl od předchozího konstruktoru inicializuje pole objektů *Quantity* novým prázdným polem.

Metodu *getQuantity* je možné použít za účelem vybrání objektu z pole objektů *Quantity* na základě vstupního parametru v podobě textového řetězce, který udává název hledané veličiny. Po zavolání metody proběhne zkouška velikosti pole. Pokud je pole prázdné, znamená to, že sonda nemá žádné záznamy a metoda vrátí objekt *null*. v opačném případě metoda projde pole s využitím javovského rozhraní *Iterator.* Při každé iteraci pole metoda porovnává proměnnou name objektu *Quantity* s textovým řetězcem ze vstupního parametru metody. Při nalezené shodě vrátí nalezený objekt *Quantity*.

Do pole objektů *Quantity* lze přidávat nová data díky metodě *addQuant*. Metoda je typu void. Mezi vstupní parametry patří objekt *Quantity* a treemapa, kde za klíč slouží javovský datum a hodnota je proměnná typu double. Na začátku metody se otestuje, pokud je pole objektů *Quantity* dané instance prázdné. v kladném případě je inicializován nový objekt *Quantity* kde jsou použity stringové proměnné name a units z objektu ve vstupních parametrech. Následně jsou do takto vytvořeného objektu přidány data za použití metody *addData* a treemapy ze vstupních parametrů metody *addQuant*. Pokud pole objektů *Quantity* není prázdné, metoda projde pole for cyklem, kde porovná proměnné všech prvků pole s objektem ve vstupních parametrech. Porovnávání probíhá na základě proměnných *name* daných objektů. v případě, kdy je nalezen hledaný objekt, proběhne přidání dat za použití treemapy ze vstupních parametrů metody *addQuant* a metody *addData*. Následně se změněný objekt uloží do listu a metoda se ukončí. Pokud se shoda nenalezne, metoda se zachová stejně, jako když zjistila, že je pole objektů *Quantity* prázdné.

8.1.3 Třída Quantity

Třída *Quantity* využívá stringových proměnných *name* a *units* reprezentující název a jednotky fyzikální veličiny. Dále vlastní treemapu *data*, kde hodnota je proměnná typu double *a* jako klíč slouží javovský datum. Třída také obsahuje gettery a settery pro tyto dvě proměnné a treemapu.

Jedinný konstruktor třídy vyžaduje všechny tři její složky a při použití je také nastaví.

Pro ukládání dat slouží metoda *addData*. Metoda nevrací žádný výstup a na vstupu vyžaduje treemapu ve stejném formátu, jako je treemapa *data* dané instance. Jediným krokem metody je sloučení těchto dvou treemap pomocí funkce *putAll* z javovské kolekce *TreeMap*.

8.2 Metody pro načítání dat

Metody pro načítání dat jsou obsaženy v balíčku *definitions*. Jedná se o statické metody. Čtení TBD souborů je možné pomocí třídy *LoadTBD*.

8.2.1 Vytvoření objektové struktury podle skupinového souboru

První metodou těchto tříd obsažených v balíčku *definition* je metoda *createStorage*. Na vstupu metody *createStorage* je zapotřebí textového řetězce, který reprezentuje název vytvářeného zásobníku a textový řetězec adresy skupinového souboru.

Po spuštění metody proběhne vytvoření prázdného objektu *Storage* za pomoci proměnné *name*, která odpovídá prvnímu vstupnímu parametru. Vytvoří se stringová proměnná *groupName*, do které se uloží hodnota *null*. Následuje přečtení skupinového souboru, jehož adresa je druhý vstupní parametr. Ke čtení jsou použity třídy *BufferedReader* a *Filereader*. *v* této části programu také může nastat chyba, v případě špatně zadané adresy. Čtení souboru proto probíhá v uzavřeném bloku *try*. *v* případě nalezené chyby se spustí blok *catch*, který nás dokáže na chybnou adresu upozornit. Metoda čte soubor po řádcích. Je zde využita stringová proměnná *line*, do které se čtený řádek uloží. Po přečtení prvního řádku se spustí cyklus while, který skončí pouze v případě, kdy bude nový řádek prázdný.

Prvním krokem cyklu while je kontrola prvního znaku přečteného řádku, kterou umožňuje statická metoda *isNumeric*. Pokud první znak odpovídá číslu, jedná se o novou skupinu. do proměnné *groupName* je uložen textový řetězec ze stringové proměnné *line* a dojde k inkrementaci počtu skupin daného zásobníku. k tomu je použit getter a setter

proměnné *groupCount* objektu Storage, který byl vytvořen po zavolání metody.

V opačném případě, kdy je první znak stringové proměnné *line* písmeno, je vytvořen nový objekt *Well*. Textový řetězec *line* slouží jako parametr konstruktoru pro vytvoření tohoto objektu. Následně se použije setter objektu *Well* a proměnné *groupName*. Jako parametr setteru slouží předem vytvořená stringová proměnná *groupName.* Objekt *Well* je pak přidán do objektu *Storage* pomocí metody *addWell.*

Po provedení podmínky dojde k přečtení nového řádku souboru a textový řetězec se uloží do stringové proměnné *line*. u posledního řádku se cyklus while ukončí a metoda vrátí objekt *Storage*, který byl upraven podle souboru.

8.2.2 Ukládání dat z datového souboru

Ukládání dat probíhá statickou metodou *addToStorage*, znázorněnou na obrázku 6. Pro zavolání je potřeba mít hlavičkový datový TBD soubor a předem připravené objekty *Storage a Quantity*. Nejdříve dojde k inicializaci dynamického pole stringových hodnot *wellNames*. Dále metoda přečte hlavičký soubor datového souboru, podle kterého se upraví pole *wellNames*. Toho je dosaženo pomocí třídy *BufferedReader* a *Filereader.* Je také potřeba uvažovat o špatně zadané adrese, proto čtení souboru probíhá uvnitř bloku *try*. v případě nalezení chybi proběhne blok *catch*, který je schopen vypsat chybovou hlášku na konzoli. Vlastní čtení souboru probíhá po řádcích. Po přečtení prvního řádku a uložení textového řetězce do stringové proměnné *line* dojde k *while* cyklu. v každém průběhu cyklu se přidá hodnota textového řetězce do pole *wellNames*. Cyklus se ukončí, pokud není žádný nový řádek a soubor se pak uzavře.

Následně je vytvořeno dočasné dynamické pole *data* hodnot ve formě treemap s prázdným záznamem pro každou položku listu *wellNames*. Dynamické pole *data* se vytváří pomocí javovské třídy *ArrayList* a samotné treemapy jsou ve tvaru, kde klíčem je javovské datum a za hodnotu slouží proměnné typu double. Samotná inicializace treemap probíhá za pomoci stringového rozhraní *Iterator.* Pro každý záznam pole *wellNames* se vytvoří nová treemapa, která je uložena do pole *data*.

Následuje čtení datového souboru. Opět jsou použity třídy *BufferedReader* a *Filereader a* čtení je uzavřeno v bloku *try*, kdy blok *catch* vypisuje případnou chybu špatně zadané adresy datového souboru. Uvnitř bloku *try* je cyklus while, ve kterém se každý řádek

rozdělí podle mezer do nově vytvořeného dynamického pole stringových hodnot *values*. První hodnota pole *values* je uložena ve formátu javovského datumu jako klíč treemap pro všechny následující hodnoty. k rozeznání datumu je použit javovský *parser.* Data z měření jsou pak přidána pro každou sondu do příslušné treemapy jako hodnoty typu double. Toho je dosaženo cyklem for, který projde pole *data.* Pro každou treemapu pole *data* je přidán nový záznam s předpřipraveným klíčem a hodnotou typu double, kde index treemapy odpovídá indexu stringového pole *values.* Po přečtení posledního řádku souboru se ukončí cyklus while *a* dojde k uzavření datového souboru.

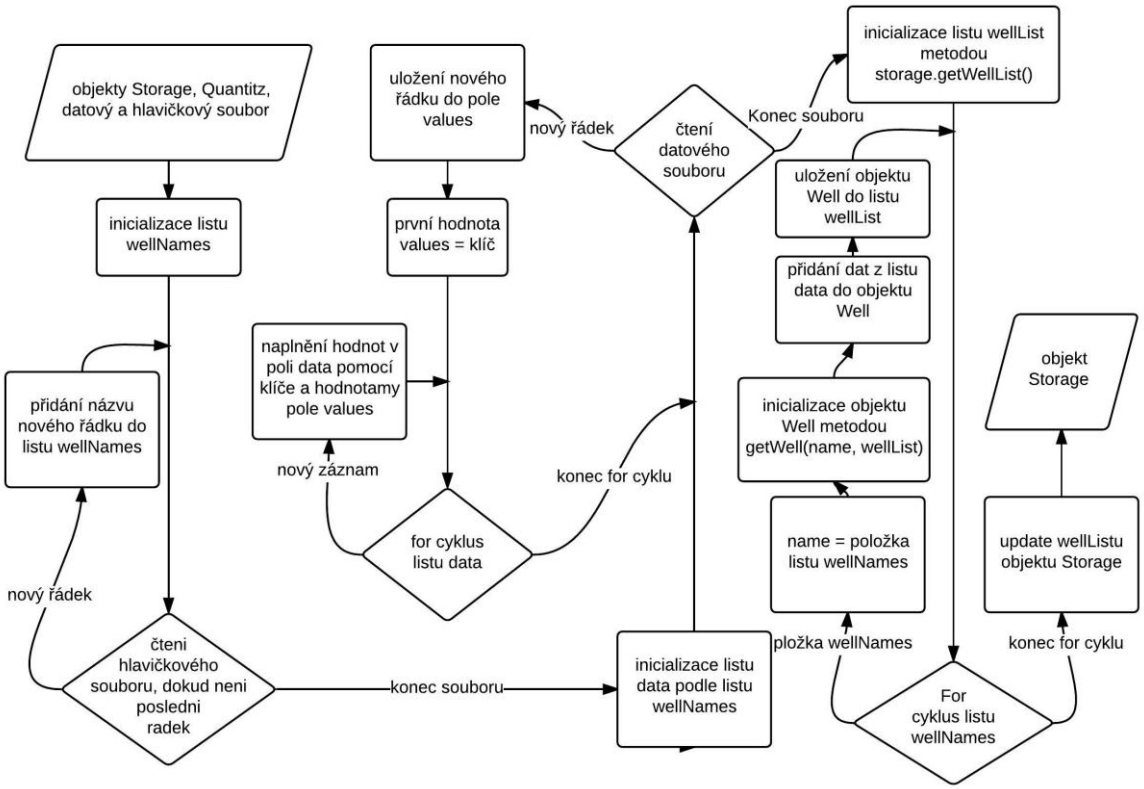
Nyní se zavolá getter objektu *Storage getWellList.* Výsledek getteru se uloží do nového pole objektů *Well* s názvem *wellList*. Následuje for cyklus, který projde polem názvů sond *wellNames. v* každém zavolání cyklu se hodnota pole uloží do nové stringové proměnné *name.* Následně se vyzkouší, pokud je *wellList* získaný z objektu *Storage* prázdný.

V případě prázdného pole *wellList* se zavolá konstruktor objektu Well s parametrem stringové proměnné *name. do* takto vytvořeného objektu jsou přidány data pomocí metody *addQuant* ze třídy *Well*. Prvním vstupním parametrem metody *addQuant* je druhý vstupní parametr metody *addToStorage,* tedy objekt typu *Quantity* představující fyzikální veličinu, jejíž data doplňujeme. Dalším parametrem metody *addQuant* je treemapa z pole *data,* jejíž index odpovídá indexu for cyklu nad polem *wellList,* pro který cyklus for právě probíhá. Objekt *Well* se pak přidá do pole *wellList.*

Za předpokladu, že *wellList* prázdný nebyl, je zavolána statická metoda *getWell* třídy, ve které se metoda *addToStorage* nachází. Vstupnímy parametry metody *getWell* je stringová proměnná *name* a pole *wellList.* Výstup metody *getWell* se uloží do nového objektu *well* typu *Well.* do objektu *well* jsou přidány data metodou *addQuant* ze třídy *Well.* Vstupními parametry metody *addQuant* jsou druhý vstupní parametr metody *addToStorage a* treemapa z pole *data* se stejným indexem jako je index právě probíhajícího for cyklu. Nyní proběhne statická metoda *contains*, která zjistí, jestli se daná sonda nachází ve wellListu. Pokud *wellList* již obsahuje objekt se stejnou stringovou proměnnou *name* jako objekt *well,* dojde k přepsání objektu v poli *wellList* objektem *well.* v opačném případě se objekt *well* přidá do *wellListu* na nové místo. Klíč pro přidání nebo změnu na správné místo se najde pomocí statické metody *getKey*, kde vstupnímy parametry jsou stringová proměnná *name* a pole *wellList*.

V posledním kroku použije metoda *addToStorage* metodu objektu *Storage setWellList,*

která přepíše pole *wellList* objektu *Storage* polem *wellList* s již novými hodnotami. Následně pošle metoda na výstup výsledek ve formě objektu *Storage*.



Obrázek 12: Metoda addToStorage

8.2.3 Pomocné metody v balíčku definition

Metody, které načítají data do struktury objektů z datových a hlavičkových souborů, využívají statických metod *getWell*, *getKey*, *contains* a *isInteger*.

Metoda *getWell* slouží k nalezení určitého objektu ze vstupního parametru pole *wellList* podle vstupního parametru textového řetězce *name*. Po zavolání metoda vytvoří nový objekt *well* typu *Well*, do kterého je uložena hodnota *null*. Následně metoda vytvoří rozhraní *Iterator* objektů *Well*, podle kterého se spustí while cyklus, který porovnává textový řetězec ze vstupu s proměnnou *name* iterovaného objektu. Po nalezení shody metoda vrátí na výstup nalezený objekt Well. Pokud hledaný objekt není nalezen, metoda vrátí objekt *well* s nulovou hodnotou.

GetKey je metoda, která má vstupní parametry textový řetězec *name* a pole *wellList*

objektů *Well*. Účelem metody je zjistit klíč, který odpovídá objektu z pole, jehož proměnná se rovná textovému řetězci *name*. Toho je docíleno pomocí integerové proměnné *key* a for cyklu přes pole *wellList*. Při každém kroku cyklu se inkrementuje proměnná *key* a dojde k porovnání hodnoty získané getterem *getName* z objektu *Well* s hodnotou textového řetězce *name*. v případě rovnosti metoda *getKey* skončí s výstupem proměnné *key*. Pokud metoda hledaný objekt nenalezla, vrátí zápornou hodnotu jedné.

Metoda *contains* zjistí, zda pole *wellList* obsahuje objekt *Well*, který má jméno rovné textovému řetězci *name*. Pole *wellList* i stringová proměnná *name* jsou vstupní parametry metody. Metoda je typu boolean a pracuje podobně jako metoda *getWell*. Při nalezení shody však metoda vrací na výstup hodnotu *true*, v opačném případě hodnotu *false*.

Metoda *isNumeric* je typu boolean. Metoda je využita pro zjištění, zda je vstupní textový řetězec složen z čísel nebo písmen. Vlastní zkouška probíhá v bloku try, kde proběhne parser javovské třídy *Integer*. Pokud parser narazí na chybu, zachytí ji blok catch, který ukončí metodu hodnotou *false*. Za předpokladu, kdy parser proběhl bez problémů, je metoda ukončena hodnotou *true*.

8.3 Metody umožňující výběr dat a jejich zobrazení

O zobrazení dat se starají třídy balíčku *tools*. Je zde třída *View*, která obsahuje metody pro výpis a výběr dat. Mezi tyto metody patří metoda *select*, znázorněná na obrázku 7, která vytvoří treemapu času měření a naměřených hodnot podle vstupních parametrů, kterými jsou objekt *Storage*, objekt *Quantity*, datum prvního a posledního měření a textový řetězec *groupName* a pole stringových hodnot *wellNames*.

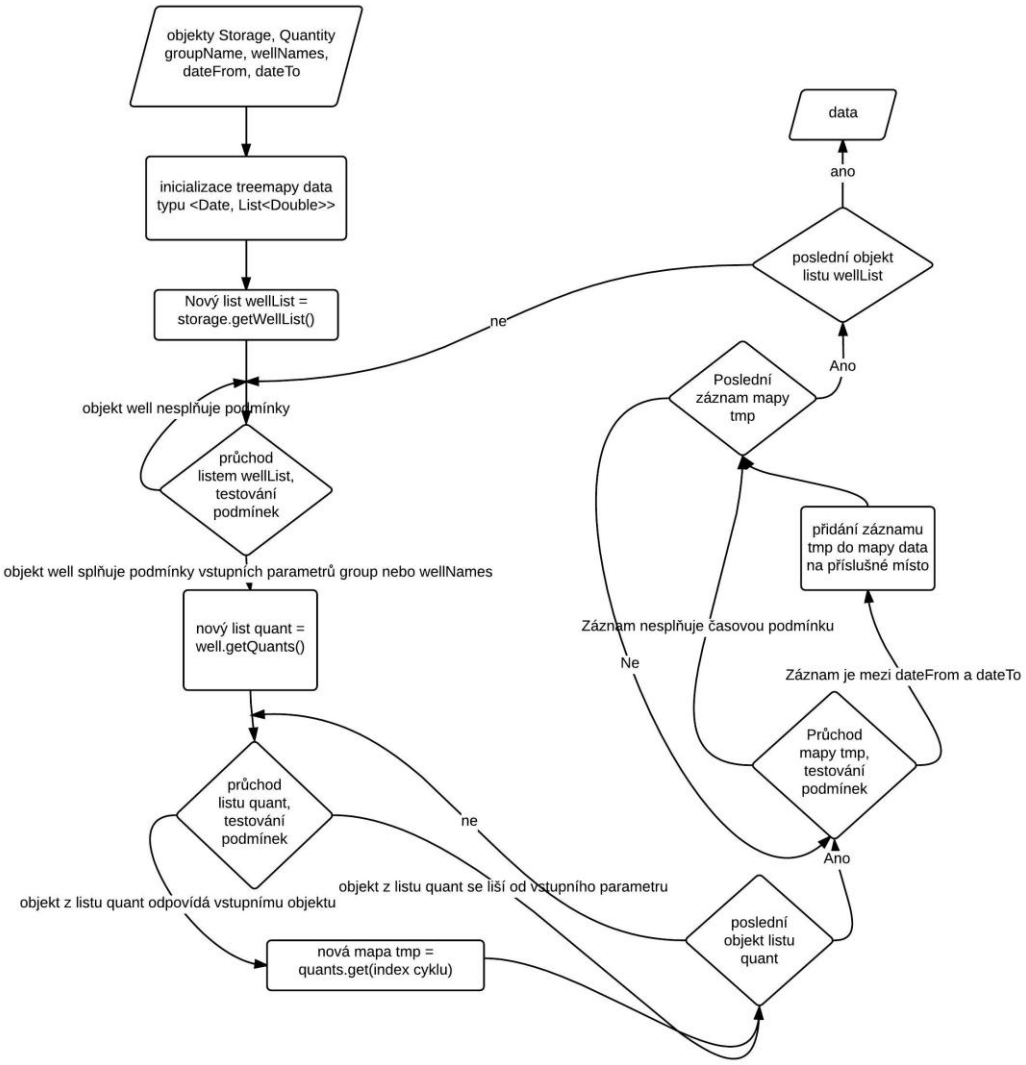
V prvním kroku metody *select* probehne inicializace dynamického pole *names* stringových hodnot a prázdné treemapy *data*, která má klíč v podobě javovského datumu. Hodnoty treemapy *data* jsou uspořádány v dynamickém poli hodnot doubelů. Dále proběhne for cyklus pole *wellList*, který byl získán zavoláním metody *getWellList* objektu *Storage* ze vstupních parametrů.

Pro každý takto nalezený objekt *well* typu *Well* proběhnou podmínky, které ověří zda objekt patří do hledané skupiny sond nebo pokud je uveden v seznamu hledaných sond. První podmínka proběhne na základě porovnání metody *getGroup* objektu *well* s textovým řetězcem *groupName*. Druhá podmínka porovnává výstup metody *contains* pole stringových hodnot

wellNames, kde jako vstupní parametr slouží metoda *getName* objektu *well*.

V případě kdy jsou splněny obě podmínky je do pole *names* přidán název sondy metodou *getName* objektu *well*. Nyní proběhne vytvoření nového dynamického pole *quants* objektů *Quantity* za použití metody *getQuants* z objektu *well*. Následuje další vnořený for cyklus pro pole *quants*. v každém kroku for cyklu je porovnáván objekt *Quantity* z procházeného pole *quants* s objektem *Quantity* ze vstupních parametrů. Porovnávání objektů probíhá na bázi metody *getName*. Pokud nalezená quantita odpovídá zadání, tak proběhne čtení treemapy odpovídajícího objektu *Quantity*. z metody *getData* objektu *Quantity* se uloží výsledek do dočasné treemapy *tmp* ve formátu kde klíč je javovský datum a hodnota je proměnná typu double.

Následně se projdou záznamy treemapy *tmp*. Toho je docíleno použitím rozhraní *Iterator* objektu *entrySet* a cyklu *while*. Pro každý záznam z mapy proběhne zkouška, zda se klíč nachází v rozmezí mezi prvním a druhým datem ze vstupních parametrů. k porovnávání datumů je použit komparátor javovských datumů *compareTo*. Pokud klíč záznamu treemapy *tmp* vyhovuje podmínce, měření je přidáno do treemapy výsledků *data*, kde hodnota je dynamicky přidána k již existujícím hodnotám konkrétního měření. Výstupem metody je treemapa *data* obsahující pouze požadované hodnoty.



Obrázek 13: Metoda select

Třída *View* také obsahuje metody, které využívají metody *select*, ale mají menší počet zadávajících parametrů pro snadnější použití. Jedná se o metody:

- *all* – vypíše všechny sondy ze zásobníku.
- *wellNames* – zvolí pouze ty sondy, jejichž jména byly napsány v textovém řetězci.
- *group* – pracuje pouze se zadanou skupinou sond.

Každá s těchto metod může také obsahovat časové omezení. Tohoto výsledku bylo dosaženo vytvořením šesti metod, kde každá volá metodu *select*, ale parametry, které ji nezajímají, vyplňuje jako *null*.

Výstupy metod *select*, *all*, *wellNames* a *group* je možné vypsat na konzoli pomocí funkce *viewList*. Ta prochází treemapu datumů a dynamických polí typu double, kterou metoda obdrží jako vstupní parametr. Průchodu treemapy je dosaženo použitím rozhraní *Iterator* objektu *entrySet* a cyklu *while*.

8.4 Metody určené pro práci s daty

K operacím s daty slouží balíček *functions*. Je v něm obsažena třída *Aggregate*. Její metody řeší součty mezi daty jednotlivých měření. Metody balíčku *functions* předpokládají na vstupu treemapu s klíčem datumu javovského formátu a hodnoty ve tvaru dynamického pole hodnot typu double.

8.4.1 Základní součtové metody

Metoda *sum* vypočítá sumu všech měření pro každý záznam treemapy zvlášť. Po zavolání metoda inicializuje novou treemapu *sums*, kde klíče jsou javovské datumy a hodnoty mají formát proměnných typu double. Metoda následně spustí for cyklus pro treemapu ze vstupního parametru. Pro každý záznam treemapy se hodnoty sečtou dohromady za použití statické metody *sumArray*. Aktuální klíč mapy a sečtené hodnoty se uloží do předem vytvořené treemapy *sums*. Po provedení cyklu se výsledná treemapa *sums* zobrazí pomocí statické metody *view*. Na závěr metoda vrátí treemapu *sums* jako výsledek.

Metoda *cuSum* řeší kumulativní součty všech hodnot poskytnutých ve vstupním parametru. Prvním krokem je inicializace treemapy *sums*, kde klíče mají formát javovských datumů a hodnoty jsou proměnných typu double. Dále je vytvořena proměnná *total* typu double s nulovou hodnotou. v dalším kroku dojde ke spuštění for cyklu pro treemapu ze vstupního parametru. v každém kroku for cyklu se sečtou dohromady hodnoty záznamu treemapy pomocí statické metody *sumArray*. Tato hodnota se přičte k hodnotě proměnné *total*. Aktuální klíč mapy a proměnná *total* se uloží do výsledné treemapy *sums*, kterou metoda použije pro výstup. Před ukončení metoda *cuSum* použije statickou metodu *view* pro zobrazení treemapy *sums*.

Metody *produced*, *injected* a *yield* pracují na stejném principu, jako metoda *cuSum*, ale ve for cyklu každá používá svou vlastní metodu místo metody *sumArray*. Metoda *produced* je použita k součtu všech záporných hodnot a využívá metody *production*. Metoda

injected umožňuje součet všech kladných hodnot a používá metodu i*njection.* Metoda *yield* spočítá součet všech hodnot v absolutní hodnotě pomocí metody *yieldSum.*

8.4.2 Pomocné metody součtových metod

Metoda *sumArray* vyžaduje na vstupu pole proměnných typu double. Po zavolání vytvoří novou proměnnou total typu double a položí její hodnotu rovnou nule. Následně projde pole ze vstupních parametrů s využitím for cyklu. v Každém kroku přičte hodnotu nalezenou v poli do proměnné *total*. Po přičtení všech hodnot pole se metoda ukončí a vrátí na výstupu proměnnou *total*.

Metody *production*, *injection* a *yieldSum* pracují na podobně jako metoda *sumArray.* Metody *production a injection* mají ve for cyklu podmínku, kterou musí hodnota před přičtením do proměnné *total* splňit. Metoda *production* přičítá pouze záporné hodnoty, zatímco metoda *injection* přičítá pouze hodnoty kladné. Metoda *yieldSum* žádnou podmínku nevlastní, ale do proměnné *total* přičítá absolutní hodnotu přírůstku.

8.5 Grafické rozhraní

Grafické rozhraní, které využívá předchozích metod a umožňuje jejich jednoduché použití, je rozvrženo do třech tříd. Ty jsou umístěny v balíčku main. k vytvoření byla použita platforma *JavaFX*.

Třída *StartingApplication* obsahuje spouštěcí metodu *main*, která po zavolání spustí metodu *start*. v metodě *start* se nachází proměnné a metody nutné k zobrazení hlavního okna rozhraní. Jednotlivé prvky okna jsou umístěny pomocí třídy *GridPane*. z hlavního okna je možné zavolat metody, které spustí metody *start* dalších dvou tříd. Konkrétně se jedná o tlačítko ,,vytvořit zásobník / přidat data do stávajícího zásobníku", které volá metodu *start* třídy StorageWindow a tlačítko ,,vytvořit nový výběr dat ze zásobníku", které spouští metodu start třídy *SelectWindow*. Vykreslování grafů bylo provedeno pomocí třídy *LineChart*.

Třídy *StorageWindow* a *SelectWindow* pracují na stejném principu jako třída *StartingApplication*, ale k výběru adres souborů byla implementována třída *FileChooser*. Pro jednoduché zadávání bylo také použito třídy *TextInputDialog* a *ChoiceDialog* z platformy JavaFX.

9 Závěr

V průběhu práce byl vytvořen funkční framework, který respektuje hierarchii zásobník, skupina sond, sonda a veličina. Framework je schopen vytvořit objektovou strukturu na základě skupinového souboru. Dále dokáže načíst data z datového souboru a podle hlavičkového souboru je uložit do příslušných objektů. Framework umožňuje vybírat data podle názvu sondy, názvu skupiny sond, názvu a jednotek veličiny a času měření. Framework také obsahuje metody pro matematické operace nad vybranými daty a také dokáže zobrazit výsledky těchto operací a výběrů.

Framework je libovolně rozšiřitelný. v případě nutnosti lze snadno přidat metody pro čtení nových typů souborů, další matematické operace a výběr dat podle nových parametrů nebo jiného výstupního formátu.

Pomocí frameworku bylo sestaveno grafické rozhraní, které může přečíst a uložit data z TBD souborů. Následně lze tyto data vybírat podle názvu sondy, názvu skupiny sond, názvu a jednotek veličiny a času měření a provádět s nimy matematické operace. Výsledky výběrů a operací lze zobrazit grafem nebo uložit do souboru.

Seznam použité literatury

[1] Yearly Report on Natural Gas Supply and Consumption in the Czech Gas System 2013, kapitola 3, s.5, www.eru.cz [online] © 2014 Energy Regulatory Office, Dostupné z: http://www.eru.cz/documents/10540/462888/ Annual_report_gas_2013.pdf/08d4cbdf-e991-40db-8888-91bf0f11461c

[2] Rozmístění a parametry podzemních zásobníků v České republice,

mojeenergie.cz [online]. © 2009-2015 [vid. 2010]. Dostupné z: www.mojeenergie.cz

[3] rwe-gasstorage.cz Dolní Dunajovice [online] © 2015 Dostupné z: http://www.rwe-gasstorage.cz/cs/dolni-dunajovice/

[4] Rosťa Jančar, Podívejte se, jak se v Česku skladuje plyn, který se nám teď hodí, technet.idnes.cz [online] © Copyright 1999–2015 [vid. 12. 1. 2009]. Dostupné z: http://technet.idnes.cz/podivejte-se-jak-se-v-cesku-skladuje-plyn-ktery-se-nam-ted-hodi-p6c-/tec_technika.aspx?c=A090108_200359_tec_technika_rja