VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

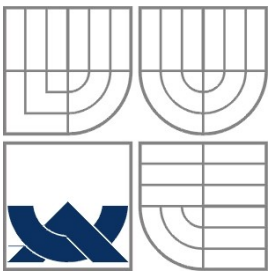
# FRAMEWORK PRO TVORBU GENERÁTORŮ DAT

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

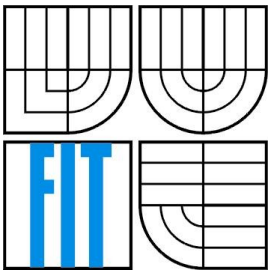
AUTOR PRÁCE
AUTHOR

BC. BLAŽEJ KŘÍŽ

BRNO 2012



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

# FRAMEWORK PRO TVORBU GENERÁTORŮ DAT

FRAMEWORK FOR DATA GENERATORS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — BC. BLAŽEJ KŘÍŽ
AUTHOR

VEDOUCÍ PRÁCE — DOC. ING. JAROSLAV ZENDULKA, CSC.
SUPERVISOR

BRNO 2012

**Vysoké učení technické v Brně - Fakulta informačních technologií**

Ústav informačních systémů Akademický rok 2011/2012

# Zadání diplomové práce

Řešitel: **Kříž Blažej, Bc.**
Obor: Informační systémy
Téma: **Framework pro tvorbu generátorů dat**
**Framework for Data Generators**
Kategorie: Data mining

Pokyny:

1. Seznamte se s problematikou generátorů testovacích dat a seznamte se s dostupnými generátory.
2. Navrhněte koncepci frameworku pro tvorbu generátorů dat. Framework musí poskytovat základní funkcionalitu typickou pro generátory testovacích dat a prostředky pro přizpůsobení specifickým podmínkám generátoru pro konkrétní účely.
3. Framework implementujte ve zvoleném programovacím jazyce.
4. Použitelnost ilustrujte na vhodné případové studii.
5. Zhodnoťte dosažené výsledky.

Literatura:

- Test (Sample) Data Generators. Dostupné na http://www.webresourcesdepot.com/test-sample-data-generators/.
- DTM Data Generator. Dostupné na http://www.sqledit.com/dg/.
- Agrawal, R., Srikant, R.: Mining Sequential Patterns. Research Report. IBM Research division, kap. 4.1. Dostupné na http://www.almaden.ibm.com/cs/projects/iis/hdb/Publications/papers/icde95_rj.pdf

Při obhajobě semestrální části diplomového projektu je požadováno:

- Body 1 a 2.

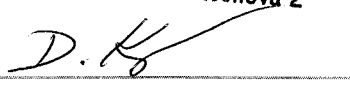
Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci ročníkového a semestrálního projektu (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Zendulka Jaroslav, doc. Ing., CSc.**, UIFS FIT VUT
Datum zadání: 19. září 2011
Datum odevzdání: 23. května 2012

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2



doc. Dr. Ing. Dušan Kolář
*vedoucí ústavu*

# Abstrakt

Tato diplomová práce se zaměřuje na problém generování dat. V úvodní části představuje několik aplikací pro generování dat a popisuje proces generování. Dále se zabývá vývojem frameworku pro tvorbu generátorů dat a demonstrační aplikace pro ověření funkčnosti tohoto frameworku.



# Abstract

This master's thesis is focused on the problem of data generation. At the beginning, it presents several applications for data generation and describes the data generation process. Then it deals with development of framework for data generators and demonstrational application for validating the framework.



# Klíčová slova

Generátor dat, testovací data, syntetická data, framework, testování, generování dat, databáze, proud dat.



# Keywords

Data generator, test data, synthetic data, framework, testing, data generation, database, data stream.



# Citace

Blažej Kříž: Framework pro tvorbu generátorů dat, diplomová práce, Brno, FIT VUT v Brně, 2012

# Framework pro tvorbu generátorů dat

## Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana doc. Ing. Jaroslava Zendulky, CSc.
Další informace mi poskytl Ing. David Petrík.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Blažej Kříž
20.5.2012

## Poděkování

Především bych rád poděkoval panu doc. Ing. Jaroslavu Zendulkovi, CSc., za poskytnuté konzultace a odborné vedení práce a dále panu Ing. Davidu Petríkovi za jeho rady a připomínky.

# Obsah

# 1 Úvod

Říká se, že žijeme v informačním věku. Otázkou, jestli a proč toto pojmenování opravdu vystihuje současnou společnost, se zde zabývat nebudeme a přenecháme ji k diskuzi jiným. Je pochopitelné, že ruku v ruce s pokrokem roste specializace práce a prohlubují se znalosti ve všech odvětvích. Čím více toho lidstvo dokáže a umí, tím více musí znát, vědět, pamatovat si. Kdysi nosili naši předkové všechno vědění pouze ve svých hlavách a znalosti si předávali především ústně. Později začali psát. Knihy byly nejprve vzácné, ale postupem času jich přibývalo a stávaly se běžnou součástí života stále více lidí. Díky rozvoji informačních technologií přecházíme v posledních desetiletích při ukládání informací od papíru na elektronická média.

Informační technologie nám dovolují pohodlně pracovat s mnohonásobně většími objemy dat, než jsme si byli schopni před sto lety vůbec představit. Nic však není zadarmo. Vývoj nástrojů IT nás stojí mnoho času, sil a peněz. Protože tyto nástroje pracují s daty a informacemi, je pro jejich testování, které do procesu vývoje neodmyslitelně patří, nutno mít k dispozici velké množství vhodných testovacích dat.

Testovací data lze získat několika způsoby: z již nasazených a používaných nástrojů, ručním vytvořením, nebo si pro jejich výrobu sestrojit pomůcku – generátor. Rozhodnutí, který ze způsobů zvolit, nepochybně závisí na parametrech dat. Generátor lze s výhodou použít v případech, kdy požadovaná data mají velký objem, relativně jednoduchou strukturu a nevyžadujeme, aby odrážela skutečný stav. Nejběžnějším případem bude zřejmě naplnění tabulky nebo několika propojených tabulek databáze, dalšími pak třeba generování provozu v počítačové síti, výstupu nějakého snímacího zařízení, emailové komunikace, webových stránek atd. Jednoduché a obecně platné pravidlo zní, že generátor (nebo jiný nástroj) nám má práci na problému šetřit, takže se vyplatí v případě, že jeho vývoj a použití je méně nákladný, než kdybychom daný problém řešili bez něj.

Cílem diplomové práce je vytvoření prostředí poskytujícího podporu při vývoji různých druhů generátorů dat. Prostředí, kde jsou vyřešeny typické postupy a obecné problémy generování. Mělo by být jednoduché, srozumitelné a dostatečně obecné, aby dokázalo pokrýt co nejširší spektrum požadavků na typ a formát generovaných dat.

Diplomová práce obsahuje kromě úvodu a závěru osm kapitol, ve kterých dává nahlédnout do současného stavu na poli programů pro generování dat (kapitola 2), popisuje generování jako proces složený z několika dílčích kroků (kapitola 3), definuje požadavky na vytvářený framework (kapitola 4) a podle nich navrhuje framework spolu s demonstrační aplikací (kapitoly 5 a 6). V dalších kapitolách se krátce věnuje implementaci, testování, použití a zhodnocení navrženého řešení.

# 2 Vybrané existující generátory dat

V této kapitole se stručně seznámíme s několika vybranými generátory dat. Díky tomu získáme alespoň hrubý přehled o současné situaci v oblasti a vytvoříme si představu o schopnostech i nedostatcích těchto řešení. Pro srovnání bylo vybráno po dvou zástupcích z kategorií placených a neplacených produktů. Oba zde uvedené a několik dalších volně dostupných nástrojů pro generování dat můžeme nalézt v přehledu [1].

## 2.1 Komerční řešení

Pokud má zákazník možnost výběru mezi placenou a neplacenou variantou a rozhodne se dát přednost té první, pak v očekávání, že se mu investice nějakým způsobem vyplatí. Ať už jde o pokročilejší funkce programu, rychlejší řešení problémů, osobní přístup, pravidelné aktualizace, školení uživatelů nebo cokoliv jiného, placený produkt musí z pohledu zákazníka nabízet něco navíc, aby si svou cenu ospravedlnil. Výrobci jsou si tohoto faktu samozřejmě dobře vědomi a snaží se požadavek splnit.

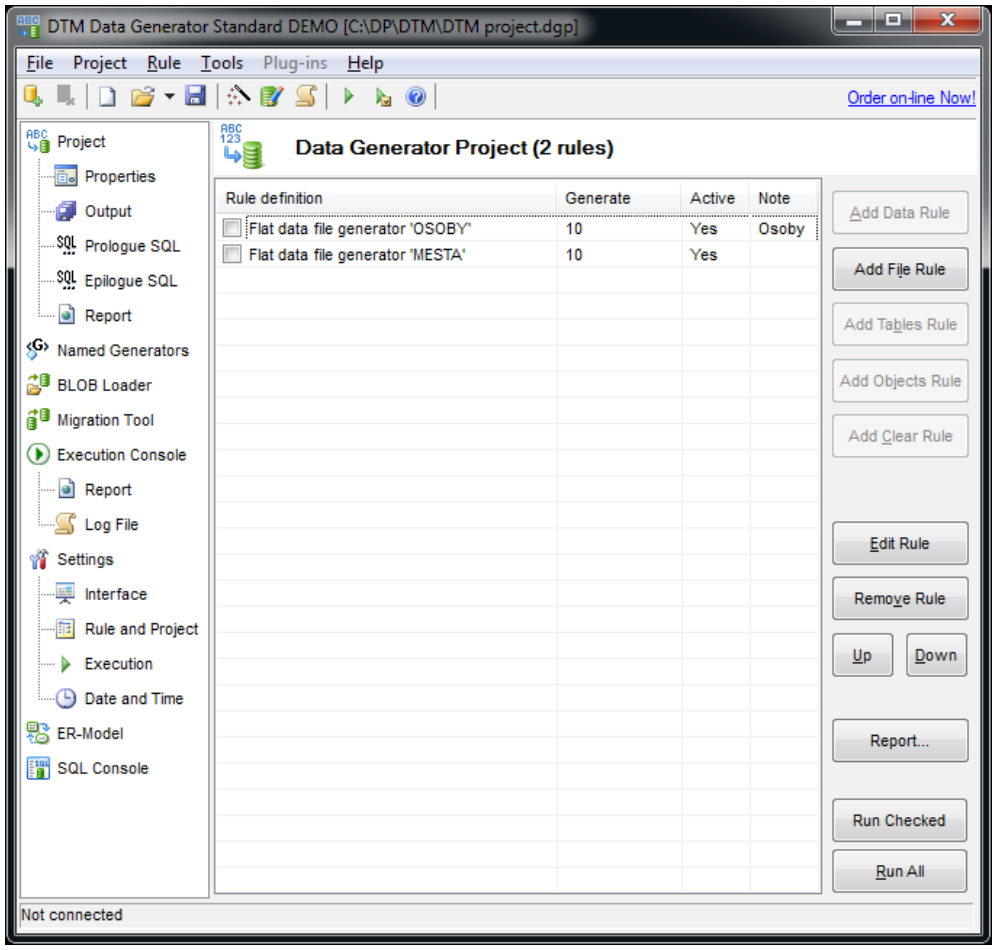
### 2.1.1 DTM Data Generator

Firma *DTM soft* vyvíjí celou řadu nástrojů pro práci s databázemi [2]. Jedním z nich je i *DTM Data Generator (obrázek 2.1)*. Nabízí množství pokročilých funkcí, např.:

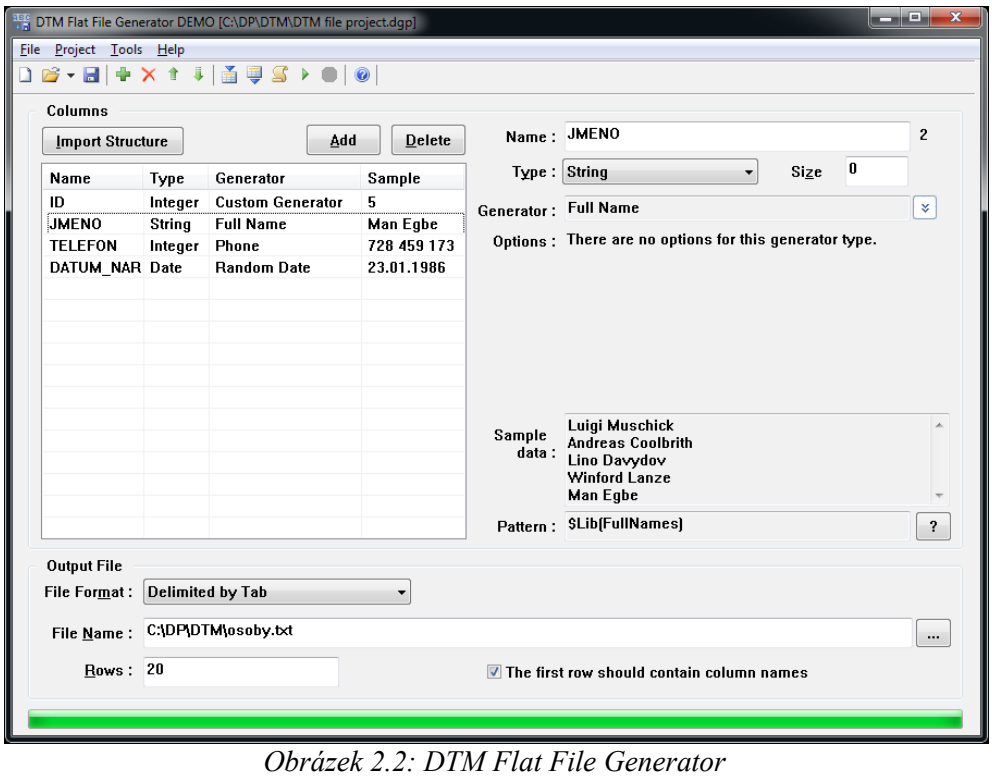
- import struktury databáze s automatickým řešením cizích klíčů
- knihovnu s předdefinovanými seznamy hodnot roztříděnými do několika kategorií
- dvanáct různých metod vyplnění sloupce tabulky včetně SQL příkazu nebo možnosti vytvořit vlastní generátor
- náhled na vzorek dat vygenerovaný vybraným pravidlem
- průvodce pro snadnější vytvoření projektu - souboru generovacích pravidel
- tiskovou sestavu s přehledem informací o provedených generováních
- náhled na ER model připojené databáze
- SQL konzoli, log a další...

Kromě plnění a čištění tabulek umí také vytvářet databázové objekty (tabulky, pohledy a procedury) a textové soubory. Pro tvorbu textových souborů se navíc s generátorem dat nainstaluje i další aplikace – *DTM Flat File Generator* (*obrázek 2.2*).

Občasná nepřehlednost je kompenzována slušnou nápovědou [3], dodávanou v instalaci ve formátu *.chm* (Microsoft Compiled HTML Help). Program je silně orientován na přímou práci s databází a pro většinu funkcí vyžaduje její připojení. Chybí například tvorba SQL skriptů jako alternativa k přímé práci. Výrobce uvádí podporu všech běžných databázových rozhraní a moderních operačních systémů Windows (2000, XP, 2003/2008 Server, Vista, Windows 7). Jedna licence základní verze označené *Standard* stojí 150$.




*Obrázek 2.1: DTM Data Generator – hlavní okno aplikace*




*Obrázek 2.2: DTM Flat File Generator*

### 2.1.2 Datamaker

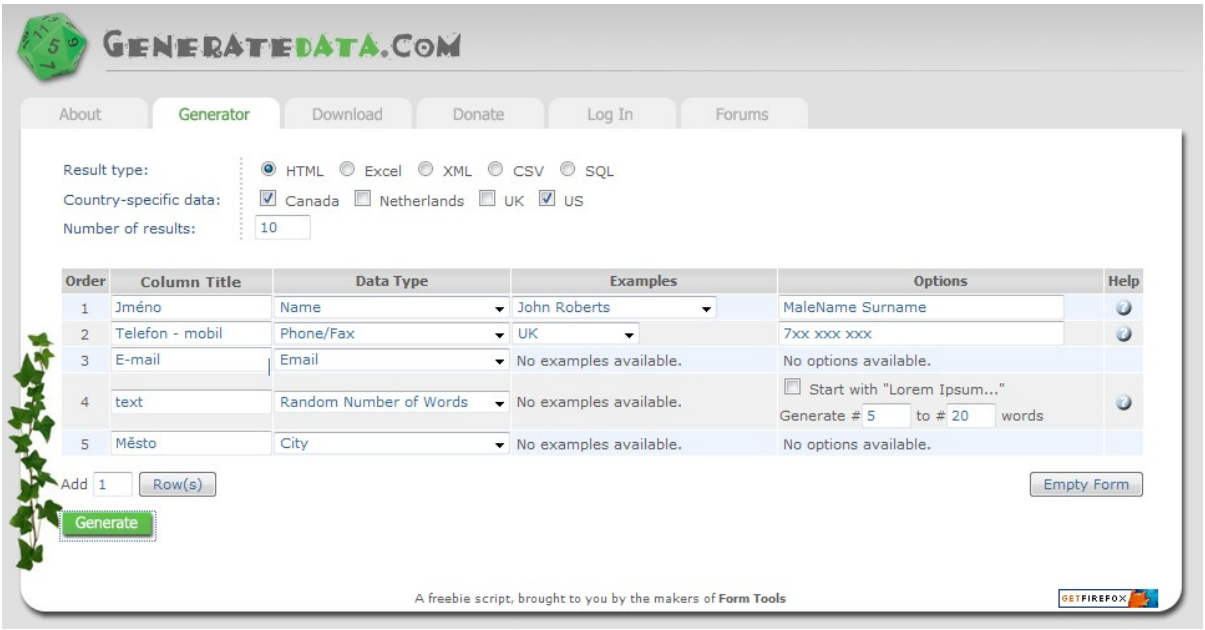
Ještě o úroveň pokročilejší funkce oproti předchozímu generátoru nabízí aplikace Datamaker, produkt britské společnosti Grid-Tools [4]. Nejedná se pouze o generátor dat, ale o celý systém pro správu testovacích dat. Umí nejen vytvářet syntetická data, ale i zatemnit citlivá produkční data, vyrábět podmnožiny a vzorky produkčních databází atd. Je určen pro nasazení ve velkých společnostech, čemuž odpovídá i cena v řádu desítek tisíc dolarů [5]. Svými schopnostmi a zaměřením daleko přesahuje téma této práce. Je zde zmíněn jednak pro představu o současné úrovni velkých SW pro správu testovacích dat a také pro demonstraci rozdílů mezi placenými produkty různých cenových kategorií.

## 2.2 Volně dostupná řešení

Ve srovnání s komerčními řešeními se jedná o podstatně jednodušší aplikace. To ovšem neznamená, že by nenalezly uplatnění. Naopak. Srozumitelnost a přehlednost, které lze díky absenci složitějších funkcí dosáhnout, mohou být při výběru generátoru v mnoha případech rozhodujícím faktorem.

### 2.2.1 GenerateData

První vybraný z kategorie volně dostupných se nachází na stránce www.generatedata.com. Aplikace je napsána pro online použití kombinací JavaScriptu, PHP a MySQL. Poskytuje pouze základní funkce pro naplnění tabulky hodnotami předdefinovaných datových typů a pevně uložených seznamů s omezenými možnostmi nastavení. Funkce však nabízí v přehledném grafickém prostředí, veškeré nastavení probíhá na jednom formuláři (*obrázek 2.3*), ovládání je přirozené a intuitivní. Kladem je také podpora pěti výstupních formátů. Ukázka jednoho z nich je přiložena na *obrázku 2.4*.




*Obrázek 2.3: GenerateData – formulář pro nastavení*

| Jméno | Telefon - mobil | E-mail | text | Město |
|---|---|---|---|---|
| Moses Richardson | 746 935 034 | morbi.tristique.senectus@eratvelpede.ca | scelerisque, lorem ipsum sodales purus, in | Fairmont |
| Porter Saunders | 777 991 155 | facilisis@massalobortis.com | velit. Pellentesque ultricies dignissim lacus. Aliquam rutrum | Evanston |
| Damon Soto | 726 304 174 | taciti.sociosqu.ad@montesnasceturridiculus.edu | libero. Integer in magna. Phasellus | Beloit |
| Harlan Marks | 719 115 689 | Proin.nisl@feugiat.edu | odio vel est tempor | Kalispell |
| Joseph Thomas | 753 355 016 | nulla.Donec@indolorFusce.edu | ac mattis ornare, lectus ante dictum mi, | Fajardo |
| Burke James | 790 352 533 | ligula@Nullamsuscipitest.org | lectus ante dictum mi, | Zanesville |
| Valentine Lane | 701 157 959 | Fusce.mi.lorem@convallis.ca | malesuada fames | Greensburg |
| Leroy Golden | 799 797 554 | amet.risus.Donec@ornaretortor.edu | elementum at, egestas a, | Dana Point |
| Allen Rios | 716 601 800 | lorem.vitae.odio@inlobortis.com | ipsum cursus vestibulum. Mauris | Cudahy |
| Ferdinand Floyd | 779 774 164 | euismod.est@tinciduntorciquis.com | ligula. Aenean euismod mauris eu elit. Nulla | Charlotte |

*Obrázek 2.4: GenerateData – výstup ve formátu HTML*

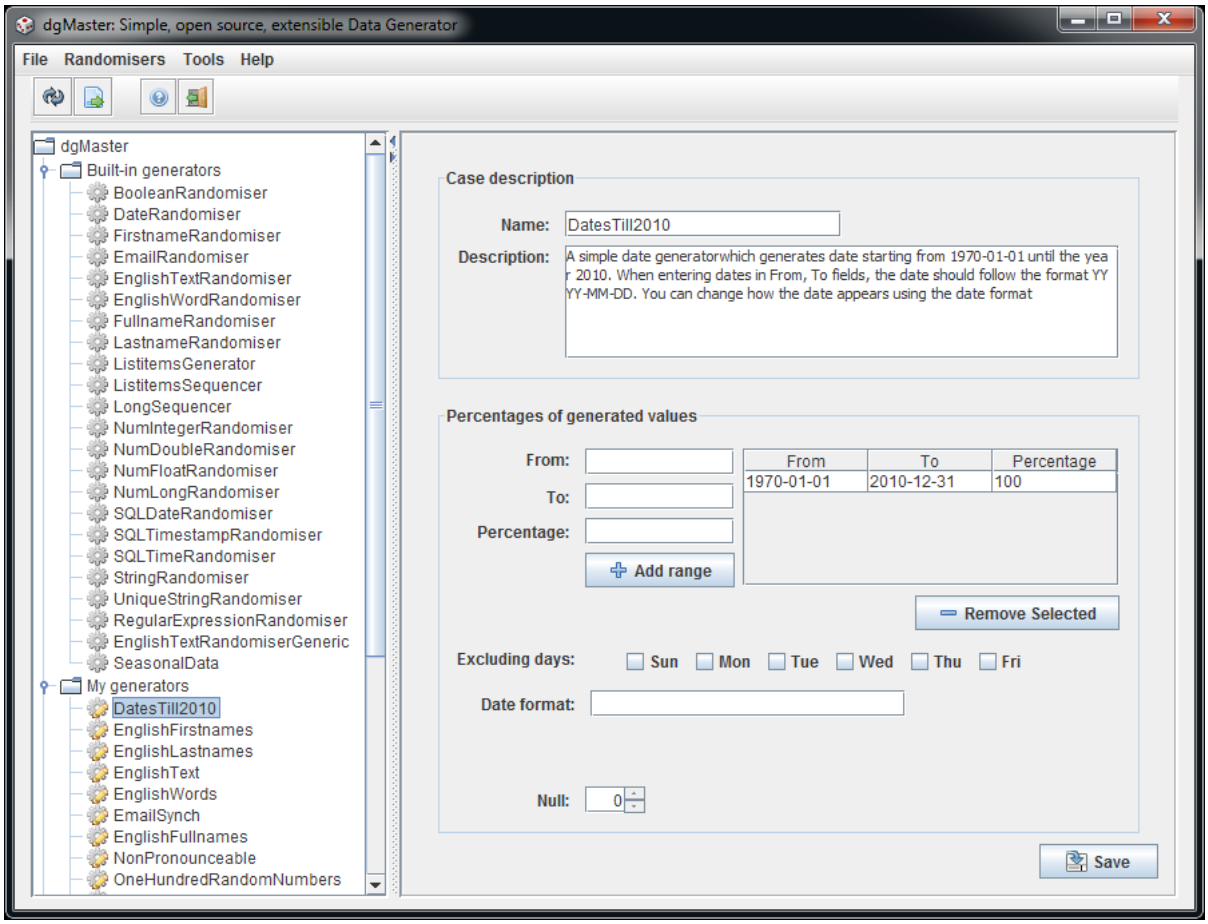
### 2.2.2 dgMaster

Projekt dgMaster [6] je psán v jazyce Java. Ve srovnání s předchozí aplikací nabízí více funkcí. Základem je sada nastavitelných předvytořených generátorů nejpoužívanějších datových typů a struktur (Bool, Date, Numeric, String, SQLTimestamp, English full names, Email atd.) s možností uložit jejich používaná nastavení. Na *obrázku 2.5* vidíme příklad nastavení generátoru pro datum.

Autor chápe, že pouze s předvytvořenými generátory v mnoha případech nelze vystačit, a proto vybízí uživatele-vývojáře k psaní vlastních úprav a vylepšení. Poskytuje konfigurovatelný logovací mechanismus pro snadnější pochopení chodu aplikace a návod, jak vytvořit vlastní generátory. Dále zmiňuje absenci možnosti spuštění z konzole, ale dodává, že díky způsobu návrhu aplikace je možné tuto funkci v případě potřeby dopsat.

Zajímavým nápadem je umožnění nastavení "*seed value*" generátoru. Jednoduše tak lze synchronizovat generátory, aby při použití stejného seznamu jako zdroje dat vybraly stejné sekvence hodnot. Podrobnější informace lze nalézt v kapitole 3.1.3 uživatelské příručky, dostupné na webu projektu.

Vzhledem k faktu, že na webu je u poslední novinky uvedeno datum 12.7.2009, a nejnovější verze ke stažení je ze stejného data, lze usuzovat, že činnost na projektu byla přerušena. Poslední verze sice slibuje podporu práce s databází, ale pouze nedokončenou a na vlastní riziko. Jediným implementovaným výstupním formátem tak zůstal textový soubor.




*Obrázek 2.5: dgMaster – nastavení generátoru DateRandomiser*

## 2.3 Shrnutí

Práce samozřejmě nemůže ani zdaleka postihnout všechny produkty, které trh nabízí. Nicméně i z takto stručného pohledu si můžeme něco odnést. Vyvolává spoustu otázek a témat k diskuzi, které musí být při tvorbě frameworku buď zodpovězeny nebo alespoň brány na vědomí:

- komplexnost nebo jednoduchost a jednoúčelovost generátorů
- zaměření na databáze nebo obecný výstup
- přenositelnost
- role a význam grafického prostředí
- předpřipravené seznamy hodnot
- každá z vybraných aplikací vhodná pro jiný případ použití

# 3 Proces generování dat

Tato kapitola se zabývá rozborem problému generování dat pomocí počítačového programu. Snaží se nalézt společné prvky a vlastnosti, které jsou nezbytnými součástmi generování a je proto vhodné je zahrnout do vytvářeného frameworku. Nalezené prvky pak jeden po druhém rozebere a analyzuje.

Na problém generování dat lze nahlížet jako na proces složený ze čtyř základních fází:

1. Vytváření datových struktur
2. Plnění vytvořených struktur hodnotami
3. Formátování naplněných struktur do výstupní podoby
4. Odeslání na výstupní rozhraní

Ať už generujeme jakákoliv data (záznamy do databáze, síťový provoz, průběh funkce atd.), vždy potřebujeme vytvořit datové typy a struktury, naplnit je vhodně zvolenými hodnotami a takto vygenerovaná strukturovaná data naformátovat a odeslat na výstup. Každé fázi procesu se podrobněji věnuje jedna z následujících podkapitol.

## 3.1 Vytvoření datových struktur

Pro účely této práce chápeme pojem *data* jako soubor hodnot různých datových typů. *Informace* jsou pak data, která mají sémantiku (význam). Můžeme říct, že data jsou surovinou, z níž získáváme informace. Více o různých významech těchto slov lze nalézt např. v [7] nebo [8].

Abychom mohli pohodlně pracovat s velkými objemy dat, je potřeba data organizovat. Bez dobré organizace je velmi obtížné informace z objemných dat získat a data tak ztrácí svoji cenu.

Vhodným prostředkem pro organizaci dat jsou datové struktury. Strukturováním se data pojmenovávají, třídí a seskupují do logicky souvisejících celků, což ulehčuje orientaci a přispívá k lepšímu pochopení významu a snadnější práci.

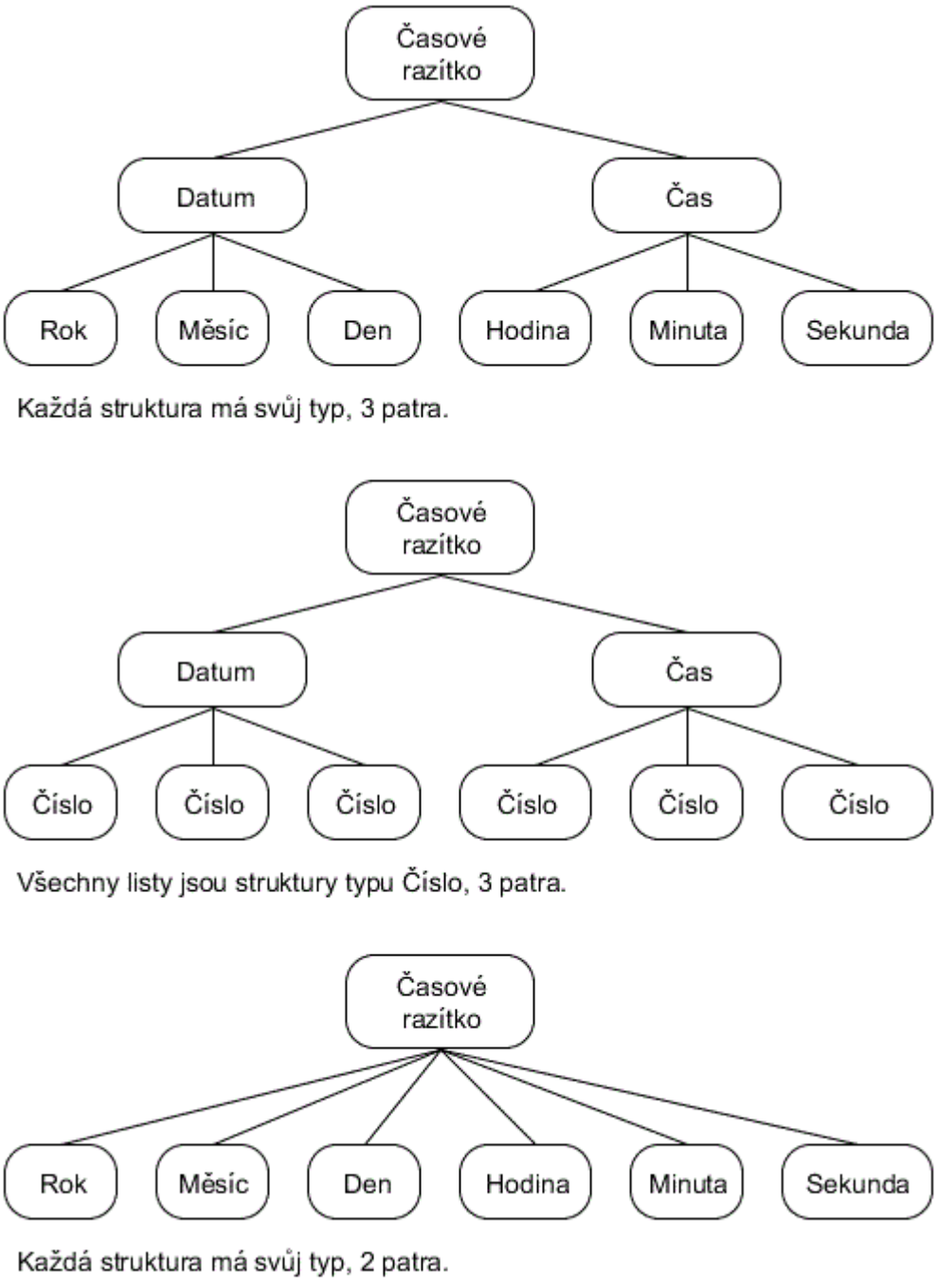
Data jsou souborem hodnot různých datových typů. Nejjednodušší datové struktury tedy budou takové, které obsahují pouze jednu hodnotu jednoho datového typu. Příkladem může být celé číslo, desetinné číslo nebo textový řetězec. Jejich skládáním pak vznikají struktury složitější. Například strukturu pro reprezentaci času s přesností na sekundy dostaneme složením tří celých čísel (hodiny, minuty, sekundy).

Datum lze také, stejně jako čas, složit ze tří celých čísel (rok, měsíc, den). Tvarem si tedy mohou tyto dvě struktury odpovídat. Přesto bychom zřejmě chtěli, aby pro datum a čas existovaly různé typy struktur. Jak již bylo zmíněno výše, struktura data pouze neseskupuje, ale také pojmenovává, popisuje a určuje jejich význam. Kromě dat samotných tedy obsahuje i metadata.

Při použití přístupu z předchozího odstavce lze vytvořit množství typů i těch nejjednodušších struktur. K velmi obecnému celému číslu dostaneme jen pro vyjádření data a času řadu konkrétnějších alternativ: sekunda, minuta, hodina, den, měsíc a rok. Zde bychom se ale měli zamyslet a uvážit, zda se nám jejich vytvořením místo pomoci a ulehčení naopak práce neztíží a neznepřehlední.

Typy datových struktur musí být na jedné straně dostatečně obecné kvůli široké možnosti použití, ale zároveň nesmí směšovat významově nesouvisející data. Při jejich tvorbě je třeba dbát na vyváženost mezi oběma přístupy.

Využívání jednodušších struktur při vytváření složitějších vede na stromovou hierarchii, kde rodič může mít *n* synů (*n* je přirozené číslo). Pro uložení dat se nabízejí dva přístupy. První data ukládá pouze v listech, druhý v libovolném vrcholu stromu. Dáme přednost prvnímu, který je sice přísnější, ale přehledně odděluje jednoduché struktury, zaměřené na samotná data, od složitějších, řešících především popis, význam a vztahy - metadata. Na *obrázku 3.1* jsou uvedeny tři z možných variant konstrukce stromu struktur pro časové razítko.




*Obrázek 3.1: Varianty konstrukce stromu struktur pro časové razítko – data uložena v listech*

## 3.2 Naplnění struktur hodnotami

Použijeme-li stromovou hierarchii struktur s uložením dat v listech, jak bylo naznačeno v předchozí podkapitole, potom uzel naplníme hodnotami naplněním všech jeho synů, list přiřazením vhodné hodnoty (nebo hodnot, pokud list obsahuje více datových položek). Při plnění struktury je třeba zohlednit veškerá omezení, která se na výběr hodnot můžou vztahovat. Tato omezení plynou jednak z typu a významu samotné struktury a zadruhé ze závislostí na ostatních strukturách.

### 3.2.1 Datové závislosti

Generování dat bez datových závislostí je relativně snadné. Omezení pro výběr hodnot se během generování nemění, každou strukturu lze nastavit a naplnit kdykoliv nezávisle na ostatních. Pokud ovšem závislosti v generovaných datech jsou, situace se poněkud komplikuje a v některých případech se dokonce stává prakticky neřešitelnou.

Omezíme-li generování pouze na takový objem dat, který budeme schopni uložit do paměti počítače, lze se při výběru hodnot pro aktuálně plněnou strukturu odkázat na libovolnou dříve vygenerovanou hodnotu. Pokud všechna generovaná data ale v paměti mít uložena nemůžeme nebo nechceme, jako například při generování nekonečného proudu dat, ukládáme pouze informace nutné pro dodržení závislostí. Jednoduchým příkladem je proměnná pro počítadlo hlídající autoinkrementaci čísla. Bohužel i objem takovýchto informací může někdy narůst do rozměrů přesahujících kapacitu paměti.

V následujícím textu se pokusíme datové závislosti rozdělit do kategorií z pohledu umístění závisejících dat. Volbou jejich umístění lze totiž ovlivnit způsob řešení jejich generování. Každou kategorii stručně popíšeme pomocí vybraného příkladu a zároveň na něm ukážeme i obecné řešení situace.

#### Závislost uvnitř struktury

Obě hodnoty, mezi nimiž existuje závislost, se objeví uvnitř jedné struktury. Například ve struktuře nesoucí informace o osobě budou položky věk a datum narození. Věk je atribut odvozený od data narození (a dnešního data, ale dnešní datum můžeme pro jednoduchost považovat za dostupnou konstantu).

Závislost tohoto typu vyřešíme jednoduše vhodným pořadím vyplnění položek struktury. Závislou položku (odvozený atribut věk) můžeme vyplnit až po vyplnění všech položek, na kterých závisí.

#### Závislost na jiné struktuře uvnitř instance stromu

Strom obsahuje alespoň dvě struktury, kdy obsah jedné závisí na obsahu druhé. Např. údaje o téže osobě jsou vyplněny ve více dokumentech a musejí si odpovídat. Sice jde o duplicitní data, ale můžeme se setkat s případem, kdy je uložení duplicitních dat vyžadováno.

Vhodným řešením je nadřazená struktura, která se o závislost stará. Tím se závislost převede na případ předchozí kategorie. Díky stromové hierarchii už navíc nadřazená struktura musí existovat.

### Závislost na předchozích instancích stromu

Strom struktur popisuje záznam databázové tabulky osob. Jedním ze sloupců tabulky je uživatelské jméno, které má mít pro každý záznam jedinečnou hodnotu.

Jestliže nemáme předchozí záznamy uloženy, musíme si vést seznam již vygenerovaných uživatelských jmen a nedovolit, aby nové jméno odpovídalo některému z předchozích.

## 3.3 Formátování do výstupní podoby

Struktury jsou vytvořeny a naplněny daty. Tím však ještě proces generování nekončí. Stejná datová struktura může mít spoustu výstupních podob. Před odesláním na výstupní rozhraní je třeba obsah struktur do jedné z těchto podob převést a struktury tak sjednotit do celku, který je možno odeslat na výstup.

Formátování struktur bude díky stromové hierarchii vypadat podobně jako jejich plnění. Formátování uzlu znamená formátování všech jeho synů a sestavení výsledků do podoby, odpovídající vlastnostem a významu uzlu. U listů pak místo se syny pracujeme přímo s hodnotami různých datových typů.

## 3.4 Odeslání na výstup

Nejvhodnější čas, kdy odeslat připravená data na výstup, nelze obecně určit. Závisí nejen na datech, která generujeme, ale také na výstupním rozhraní a způsobu, jakým s daty během celého procesu pracujeme. Obecně lze říci, že vhodným časem pro odeslání je okamžik naformátování struktury, tvořící na výstupu nějaký rozumný celek. Takový celek může být v procesu generování reprezentován podstromem, celým stromem nebo dokonce několika stromy.

Po odeslání dat na výstup je možné zpracované struktury odstranit. Nejpozději před odstraněním bychom si ale měli uložit informace nutné pro dodržení datových závislostí při generování dalších dat.

# 4 Požadavky na vytvářený framework

Framework má poskytnout podporu při vývoji specializovaných generátorů dat. Tedy prostředí, kde jsou vyřešeny obecné postupy a typické problémy, aby se vývojář mohl soustředit pouze na řešení svého vlastního zadání. Nesmí ovšem být příliš komplikovaný, aby čas potřebný pro jeho nastudování nebyl delší než úspora, kterou jeho použití přinese.

Pro popis požadavků použijeme stručnou, ale výstižnou a přehlednou formu nečíslovaného seznamu. Požadavky rozdělíme nejprve na funkční a nefunkční a dále pak podle jejich důležitosti (priority).

Jednotlivé funkční požadavky vycházejí z rozboru procesu generování v předchozí kapitole nebo z informací a zkušeností sebraných při studiu již existujících generátorů. Nefunkční požadavky se výběrem jazyka spolu s přenositelností a komentováním kódu snaží vývojářům zajistit pohodlné používání frameworku.

Spolu s frameworkem je třeba vyvinout aplikaci pro ověření a demonstraci jeho funkčnosti.

## 4.1 Funkční požadavky

### Vysoká důležitost

- postihnutí jednotlivých fází procesu generování dat, jak jsou popsány v předchozí kapitole, implementací obecných postupů, ze kterých lze vycházet při vývoji specializovaných řešení
  - vzorové datové struktury
  - mechanismus pro plnění struktur
  - mechanismus pro zformátování naplněných struktur do výstupní podoby
- řešení datových závislostí
- neomezená velikost výstupu (možnost generování časově neomezeně dlouhého proudu dat)
- možnost pozastavit/zrušit probíhající generování

### Střední důležitost

- načtení seznamu hodnot ze souboru – souvisí s následujícím bodem
- podpora různých způsobů plnění:
  - šance pro nevyplnění hodnoty (simulace neúplných dat)
  - číselná řada s volitelným počátečním číslem a krokem
  - náhodná hodnota ze seznamu
  - sekvenční výběr hodnot ze seznamu

### Nízká důležitost

- podpora pro uložení/načtení vygenerovaných, ale neformátovaných dat pro jejich pozdější zpracování
  - opakování stejného výstupu
  - stejná data s různým formátováním
- grafický editor datových struktur

- funkce "preview" – vygenerování, naformátování a zobrazení vzorku dat dle aktuálního nastavení

## 4.2 Nefunkční požadavky

### Vysoká důležitost

- implementace v jazyce Java nebo C#
- přenositelnost
- komentování kódu
  - krátké komentáře u méně přehledných nebo náročných bloků kódu
  - komentáře důležitých tříd a metod způsobem dovolujícím automatické generování dokumentace (např. Javadoc) nebo přiložení dokumentace obdobného záběru

### Střední důležitost

- podpora pro přesun implementovaných generátorů mezi instancemi aplikace

## 4.3 Požadavky na demonstrační aplikaci

### Vysoká důležitost

- podpora pro výstup dat do souboru
- spustitelnost generování z konzole

### Střední důležitost

- grafické uživatelské rozhraní pro nastavení a spuštění generování

### Nízká důležitost

- podpora pro další výstupní rozhraní
  - komunikace s DB serverem
  - odesílání dat na síťové rozhraní
  - …

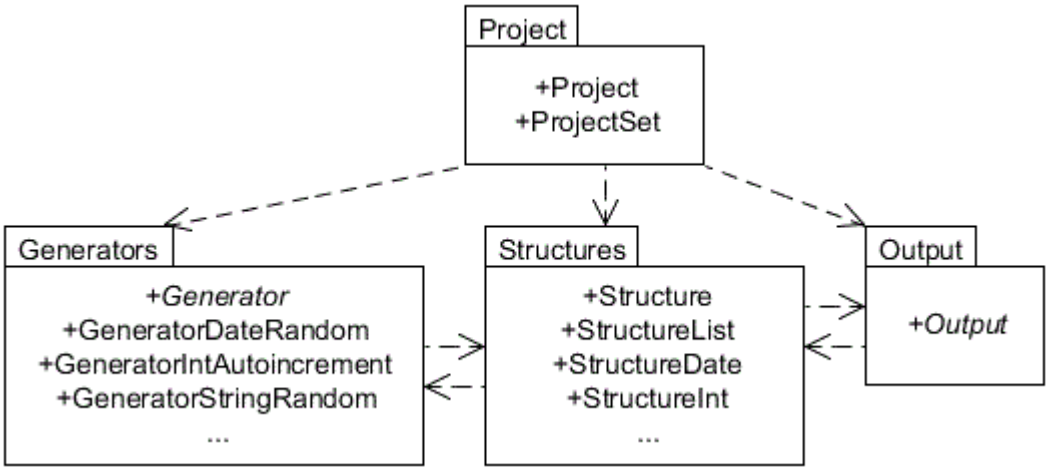
# 5 Analýza a návrh frameworku

Při návrhu frameworku vyjdeme z pohledu předchozích kapitol, kde se na generování díváme jako na proces. Jeho jednotlivé kroky budeme řešit oddělenými mechanismy, které pak zastřešíme a spojíme do jednoho celku. Přitom budeme respektovat zadané požadavky. Protože se jedná o obecný nástroj vyžadující rozšiřování uživatelem, jeho základní konstrukce musí být dostatečně jednoduché, přehledné a srozumitelné, aby neodradily od studia a používání.

Framework můžeme rozdělit na dvě části. V první se nachází sada balíků (*packages*), které samy o sobě obsáhnou proces generování a postačí pro realizaci nejjednodušších případů generování. Druhá část pak obsahuje veškeré balíky rozšiřující funkčnost.

## 5.1 Základní balíky

Do této skupiny zahrneme balíky nezbytné pro každé generování. Jedná se o základní stavební kameny a uživatel je musí vždy použít, protože bez nich by nebyl proces úplný. Jejich výčet a vazby vidíme na *obrázku 5.1*.




*Obrázek 5.1: Schéma vazeb základních balíků*

### 5.1.1 Structures

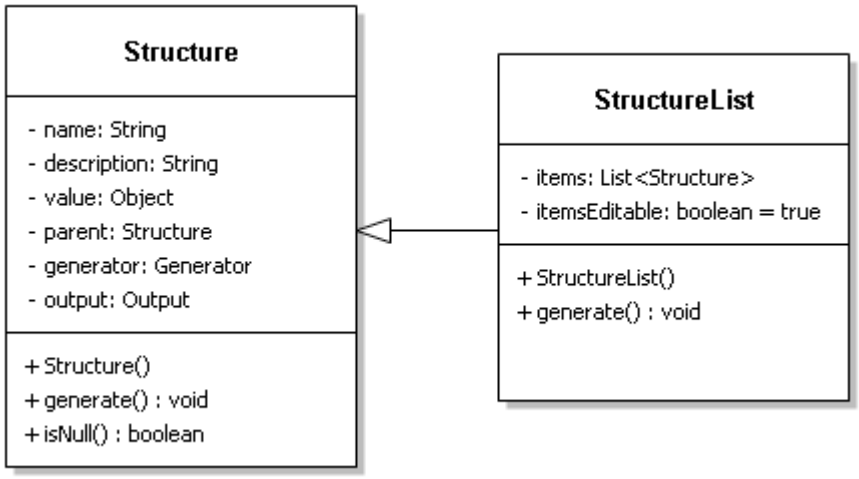
První balík, jehož základem je třída *Structure*, poskytuje prostor pro uložení vygenerovaných hodnot a vymezuje strukturu generovaných dat. Třída *Structure* obsahuje následující atributy:

- *name* – jméno
- *description* – popis
- *value* – atribut pro uložení hodnoty (dat, které struktura obaluje)
- *parent* – odkaz na strukturu rodiče
- *generator* – objekt zajišťující naplnění struktury daty – vyplnění atributu *value*
- *output* – objekt zajišťující formátování struktury do výstupní podoby a odeslání dat na výstup

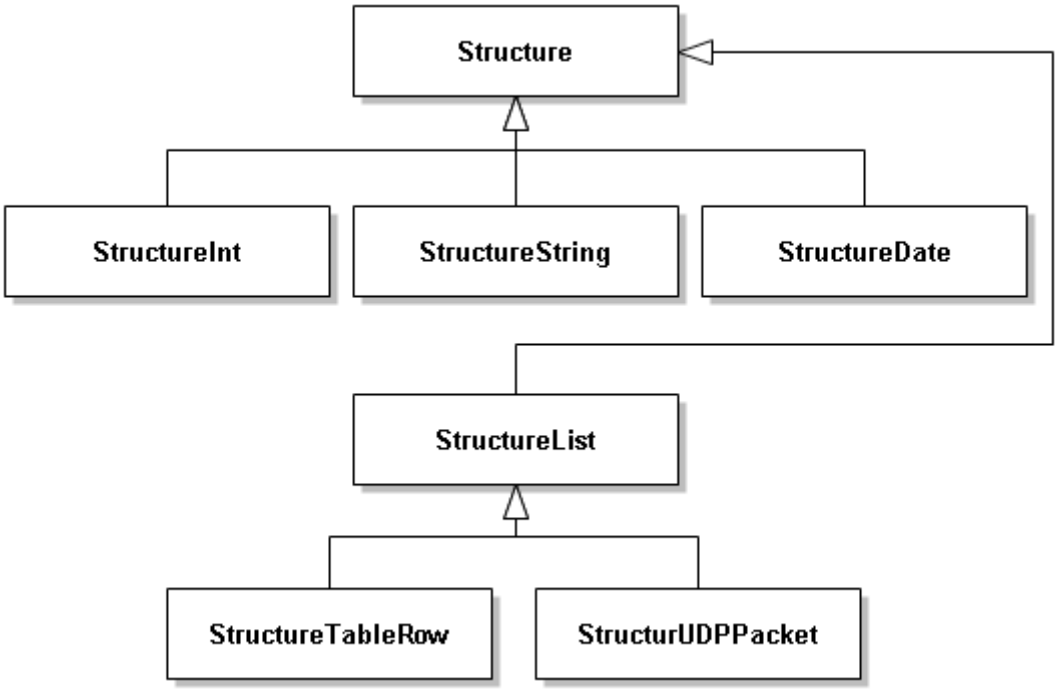
Třída *Structure* je obecným předkem všech tříd struktur. Její potomci jsou specializováni tak, aby mohli plnit úlohy konkrétních součástí stromové hierarchie. Zavoláním metody *generate()* dojde ke spuštění přiřazeného generátoru a naplnění struktury daty.

Další důležitou třídou je *StructureList*, která má oproti třídě *Structure* navíc atribut seznamu struktur *items*. Díky tomu může obsahovat jednu nebo více jiných struktur a zastávat tak ve stromu funkci rodiče.

Ostatní třídy dědí dle potřeby buď z třídy *StructureList*, pokud ve stromu plní funkci rodiče, nebo pouze z třídy *Structure*, pokud slouží pouze pro uložení hodnoty. V kapitole 3 bylo zmíněno, že dáme přednost přístupu, kdy hodnoty jsou uloženy pouze v listech stromu. Vývojář ale k tomuto rozhodnutí není nucen. Díky tomu, že všechny třídy struktur jsou potomky třídy *Structure,* je mu umožněno uložit si hodnotu i do nelistových uzlů stromu. Vztah tříd ukazuje *obrázek 5.2*.




*Obrázek 5.2: Vztah tříd Structure a StructureList*

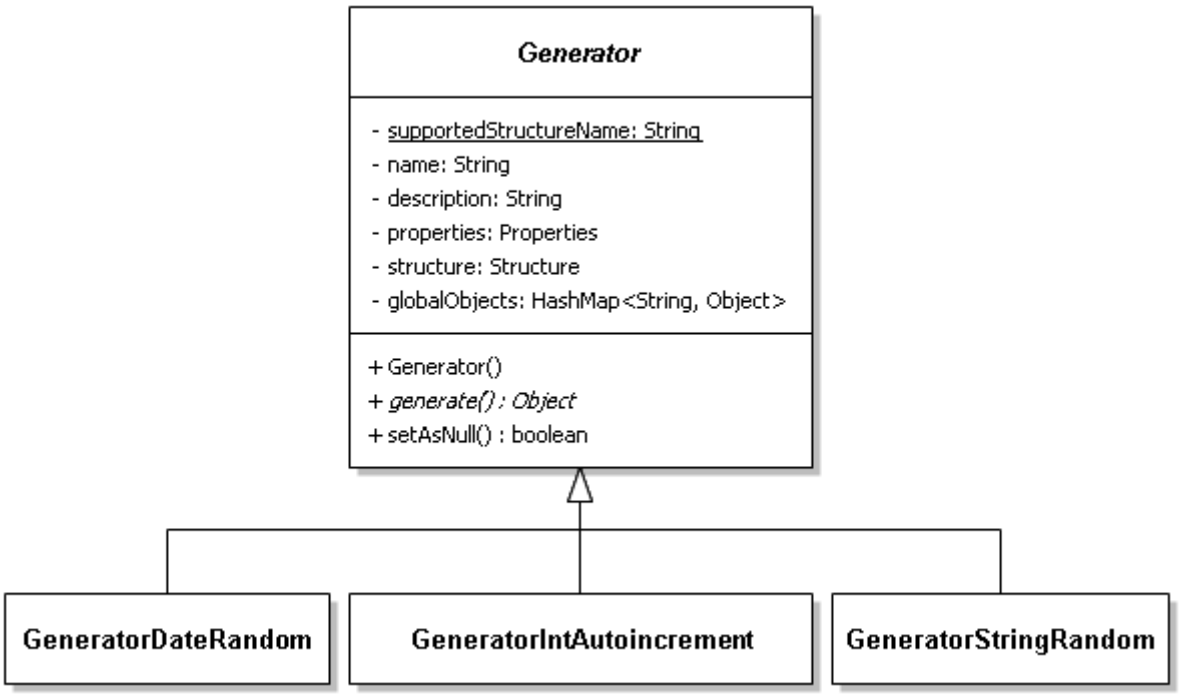



*Obrázek 5.3: Příklady potomků tříd Structure a StructureList*

Potomky tříd *Structure* a *StructureList* jsou již struktury konkrétních datových typů nebo struktur (*obrázek 5.3*). Několik obecně použitelných potomků je přímo součástí frameworku, další, více specializované, jsou pak v balících demonstračních příkladů.

Pokud rodičovský uzel (instance třídy *StructureList* nebo některého z jeho potomků) reprezentuje strukturu pevně daného tvaru, může být vhodné zároveň s ním vytvořit i jeho syny a nedovolit pozdější úpravy jejich počtu nebo pořadí. Pro tuto možnost existuje ve třídě *StructureList* příznak *itemsEditable*, jehož nastavením vývojář dodatečné úpravy povoluje/zakazuje. Vytvoření a nastavení stromu tříd z balíku *Structure* odpovídá první fázi procesu generování dat.

### 5.1.2 Generators




*Obrázek 5.4: Třída Generator a vybrané příklady potomků*

Balík *Generators* má na starosti druhou fázi procesu – naplnění struktur hodnotami. Základní třídou tohoto balíku a předkem všech tříd generátorů je abstraktní *Generator (obrázek 5.4)* s atributy:

- *supportedStructureName* – název struktury, kterou umí generátor naplnit; třídní atribut
- *name* – jméno
- *description* – popis
- *properties* – seznam parametrů generátoru, položkami seznamu jsou dvojice (klíč, hodnota)
- *structure* – odkaz na přiřazenou strukturu
- *globalObjects* – odkaz na globální objekty

Generátor je vždy svázán s jedním typem struktury. Strukturu je možno naplnit mnoha způsoby, proto lze generátorům nastavit parametry. Pro jeden typ struktury navíc můžeme vytvořit množství různých generátorů.

Sada podporovaných parametrů je individuální záležitostí každého generátoru. Jediným parametrem, který má podporu již v třídě *Generator*, je parametr s klíčem "*null*". Jeho hodnota (číslo

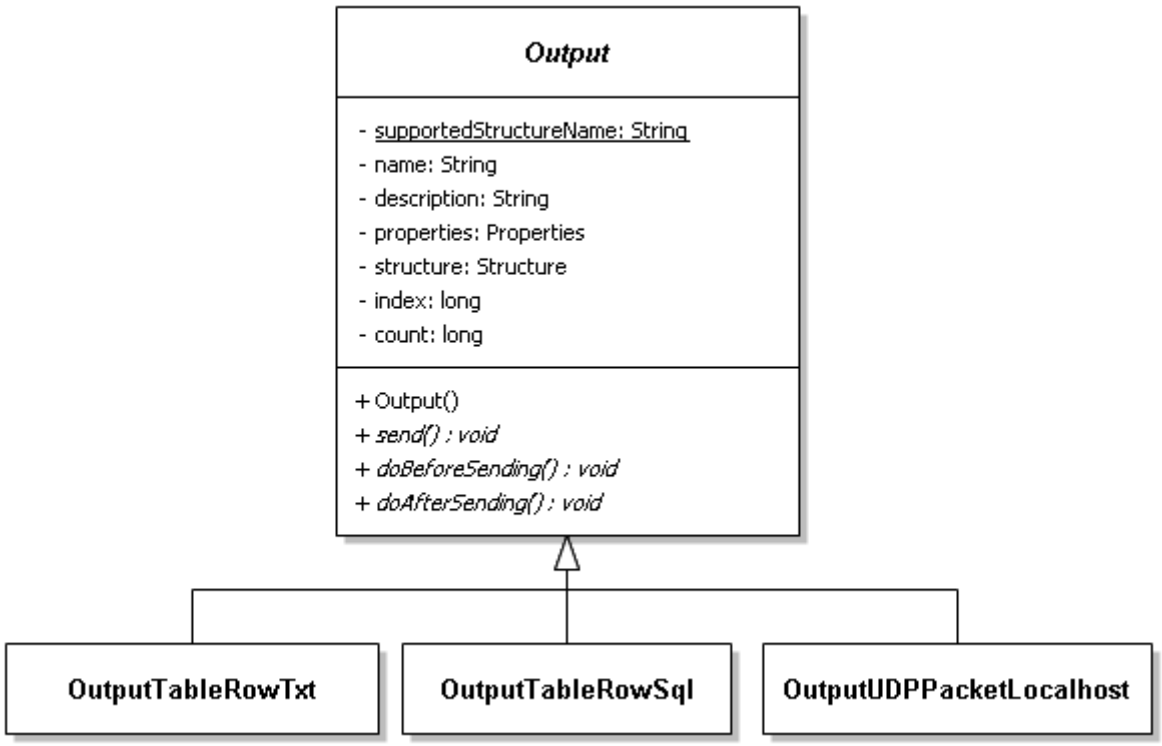
z intervalu <0;100>) udává procento struktur, které budou vyplněny hodnotou *null*. Tímto způsobem vyplnění struktury simulujeme neúplnost dat.

Naplnění rodičovského uzlu znamená naplnění všech jeho synů. Generátor pro uzel má možnost syna naplnit buď zavoláním generátoru pro strukturu synova typu nebo bez použití takového generátoru. Pro ilustraci generátor času *GeneratorTimeRandomRoundToMin* má za úkol vyplnit strukturu *StructureTime*, která má syny *StructureHour*, *StructureMin* a *StructureSec*. Čas má být zaokrouhlen na minuty. Použije generátory *GeneratorHourRandom* a *GeneratorMinRandom* a strukturu *StructureSec* vyplní nastavením jejího atributu *value* na 0. Uvedený příklad samozřejmě není jedinou správnou možností, ale pouze jedním z mnoha řešení, které navíc předpokládá existenci konkrétních struktur a generátorů. Je zde zmíněn pouze pro objasnění faktu, že pro plnění struktury není nezbytně nutné využít generátor.

Jestliže rodičovský uzel nemá žádný generátor přiřazen, jeho metoda *generate()* postupně volá generování všech jeho synů. Pokud uvažujeme dříve zmiňovaný přístup, kdy data jsou uložena jen v listech stromu, pak pro jejich vygenerování stačí přiřadit generátory pouze všem listům stromu.

Stejně jako balík *Structures*, i balík *Generators* obsahuje kromě základní třídy *Generator* i několik jejích obecně použitelných potomků. Specializovanější generátory opět nalezneme v balících demonstračních příkladů.

### 5.1.3 Output




*Obrázek 5.5: Třída Output a vybrané příklady potomků*

Zatímco generátor hodnoty do struktur zapisuje, úkolem tříd z balíku *Output* je jejich čtení, skládání, převod do výstupní podoby (třetí fáze procesu) a odeslání na výstup (čtvrtá fáze procesu). Stejně jako

*Generators*, i *Output* má svou základní abstraktní třídu. Jmenuje se *Output (obrázek 5.5)* a jejími atributy jsou:

- *supportedStructureName* – název struktury, kterou umí *Output* zpracovat; třídní atribut
- *name* – jméno
- *description* – popis
- *properties* – seznam parametrů výstupu, položkami seznamu jsou dvojice (klíč, hodnota)
- *structure* – odkaz na přiřazenou strukturu
- *index* – pořadové číslo právě zpracovávané struktury v rámci spuštěného projektu
- *count* – celkový počet struktur generovaných v rámci spuštěného projektu

Stejně jako generátor, i výstup je svázán s jedním typem struktury, pro jeden typ struktury můžeme vytvořit více různých výstupních tříd s parametry a zpracování uzlu znamená zpracování všech jeho synů. Způsob zpracování stromu struktur je však odlišný.

Generátory jsou obvykle přiřazovány nejnižším patrům stromu a uzly vyšších pater pouze dávají pokyn synovským uzlům, aby provedly generování. Při sběru, formátování a odesílání vygenerovaných dat je tomu právě naopak. Data je třeba nejprve sestavit do větších celků, a teprve ty pak odesílat na výstupní rozhraní. O celé sestavení a odeslání stromu struktur se tak může starat i jediný potomek třídy *Ouput*, který zvládne zpracovat strukturu kořene stromu. Pokud by implementace takové třídy nebyla z nějakého důvodu vhodná, pak je samozřejmě možné vytvořit potomky třídy *Ouput* pro nižší patro, které si třída zpracovávající kořenový uzel může zavolat.

Před začátkem generování dat projekt nastaví parametr *count* a zavolá metodu *doBeforeSending()*, která provede další operace potřebné pro inicializaci použité výstupní třídy. V průběhu generování je pro každou hlavní strukturu volána metoda *send()* zajišťující její zpracování včetně odeslání na výstupní rozhraní. Na konci generování, po zpracování poslední generované struktury, pak metoda *doAfterSending()* ukončí práci s výstupním rozhraním.

Na rozdíl od *Structures* a *Generators*, balík *Output* obsahuje pouze abstraktní třídu *Output*. Ta v současné době nemá žádného potomka, který by byl dostatečně obecný, aby mohl být součástí frameworku. Všichni potomci řeší konkrétní typy výstupu demonstračních příkladů, a proto patří do jejich balíků, které leží mimo balík frameworku.

### 5.1.4 Project

Poslední ze základních balíků, *Project*, obsahuje pouze dvě třídy. Třída *Project* zastřešuje celý proces generování jednoho stromu struktur. Zde jsou její atributy:

- *name* – jméno
- *description* – popis
- *mainStr* – hlavní struktura projektu (kořen stromu struktur)
- *runAfter* – seznam názvů projektů, na kterých je daný projekt závislý
- *count* – počet generovaných instancí hlavní struktury
- *index* – pořadové číslo právě zpracovávané hlavní struktury
- *sleepTime* – čas prodlevy (čekání) mezi generováním dvou instancí hlavní struktury
- *stopped* – příznak běhu/zastavení generování

V následujícím textu může být jako synonymum ke slovním spojením *strom struktur* nebo *instance hlavní struktury projektu* pro přehlednost a lepší čitelnost použit výraz *záznam*. I když každý

z uvedených pojmů má mírně odlišný význam, v mnoha případech nepotřebujeme tyto drobné rozdíly zdůrazňovat a pojmy se stávají zaměnitelnými.

Projekt uchovává v atributu *mainStr* strom struktur a přidává k němu informace a funkce potřebné pro řízení procesu generování. Jeho metoda *run()* rozhoduje, kdy bude hlavní struktura naplněna, zformátována, odeslána na výstup. Nastavením příznaku *stopped* je možno běh této metody ručně zastavit a tím vynutit ukončení procesu před jeho plánovaným doběhnutím. Dále lze projektu nastavit počet generovaných záznamů (atribut *count*) a čas prodlevy mezi generováním dvou záznamů (atribut *sleepTime*).

Pro potřeby archivace a znovupoužití podporuje třída *Project* uložení vytvořeného projektu do souboru a samozřejmě také načtení takto uloženého projektu. K tomuto účelu slouží třídní metody *save()* a *load()*. Metoda *printInfo()* pak dovoluje vypsat základní informace o projektu.

Druhá třída balíku *Project* má název *ProjectSet*. Umožňuje spojovat související projekty do skupin a s těmito skupinami pak pracovat jako s celky. Má následující atributy:

- *name* – jméno
- *description* – popis
- *projects* – seznam projektů ke spuštění
- *globalObjects* – skladiště informací pro komunikaci mezi projekty

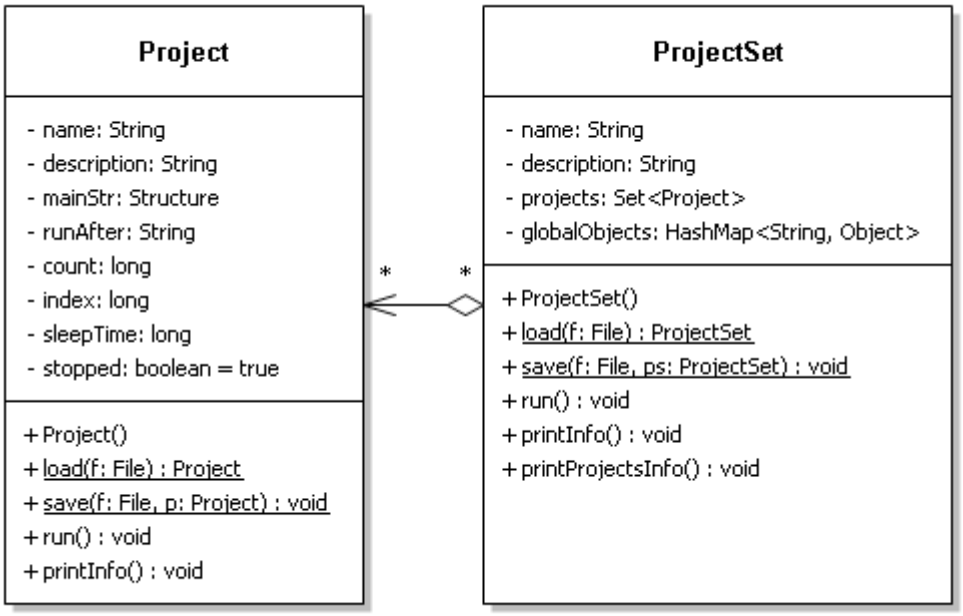
Třída *ProjectSet* leží na rozmezí mezi základními a rozšiřujícími prvky. Pokud bychom spouštěli pouze jeden projekt, proces generování by se bez ní obešel. Často ale bude spouštěno více vazbami propojených projektů a pro tento případ už je její použití nutné.

*ProjectSet* řídí pořadí spouštění projektů kontrolou a aktualizací jejich atributu *runAfter*. Projekt lze spustit až po spuštění všech projektů, jejichž názvy jsou v jeho *runAfter* seznamu. Po úspěšném doběhnutí projektu je jeho jméno odstraněno ze všech *runAfter* seznamů ostatních projektů.

V rámci *ProjectSetu* je také k dispozici objekt *globalObjects* pro možnost sdílení informací mezi jednotlivými projekty. Poskytuje prostředek pro řešení datových závislostí přesahujících rozměr jednoho projektu.

Spojení několika projektů do *ProjectSetu* je částečně nahraditelné spojením několika stromů struktur do jednoho stromu přidáním společného kořenového uzlu. Tím ovšem přicházíme o všechny výhody, které *ProjectSet* nabízí, jako například možnost individuálního nastavení jednotlivých projektů. Navíc bychom stejně museli mít přichystány vhodné třídy z balíků *Structures* a *Output*, které by kořenový uzel vyžadoval.

Stejně jako *Project*, i *ProjectSet* podporuje uložení a načtení do souboru třídními metodami *save()* a *load()* a výpis základních informací metodou *printInfo()*. K tomu přidává možnost výpisu informací o všech projektech, ze kterých je složen – metoda *printProjectsInfo()*. Obě třídy jsou k vidění na společném *obrázku 5.6*.




*Obrázek 5.6: Třídy Project a ProjectSet z balíku Project*

## 5.2 Rozšiřující prvky

Všechny nástroje, které je možno, ale není nutno při generování využít, zařadíme mezi rozšiřující prvky do balíku *Tools*. Budou sem patřit například předdefinované seznamy hodnot, pomocné třídy generující různá rozložení hodnot, třídy pro specializovanou práci se soubory nebo jinými výstupními rozhraními atd. V průběhu vývoje se počítá s jejich postupným přidáváním a začleňováním dle potřeby.

### 5.2.1 Randoms

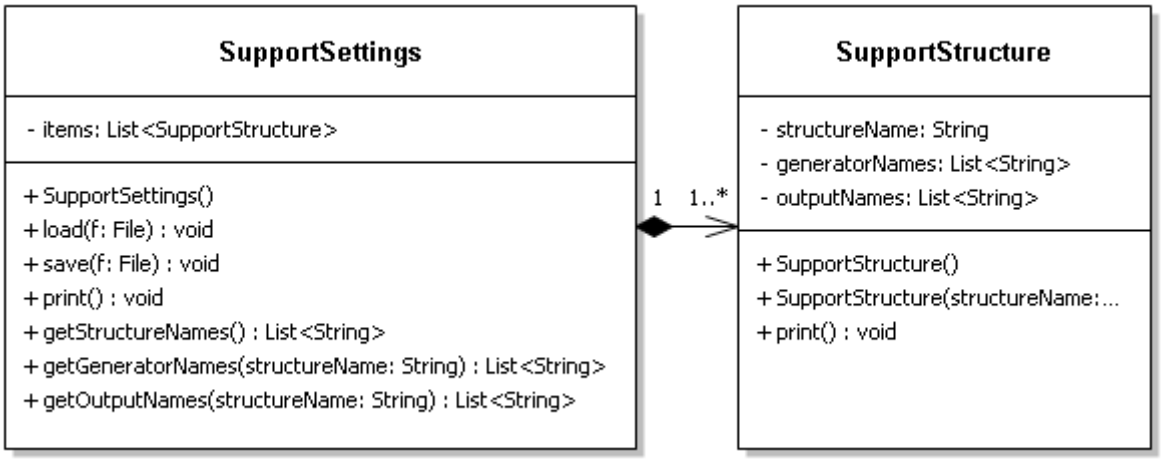
Jedná se o třídu složenou z několika statických metod pro ulehčení generování pseudonáhodných hodnot. Prozatím umí generovat čísla a kalendářní data ze zadaného rozsahu nebo vybrat řetězec ze zadaného seznamu. Postupně může být vylepšována a rozšiřována o další metody podobných funkcí.

### 5.2.2 SupportSettings

Jak bylo řečeno u popisu tříd *Generator* a *Output*, tyto třídy jsou vždy svázány s jedním typem struktury. Název třídy struktury, se kterou daný *Generator*, resp. *Output* umí spolupracovat, lze uložit do třídního atributu *supportedStructureName*. Tento decentralizovaný způsob uložení má však své nevýhody. Například pokud by chtěl vývojář v aplikaci nabídnout uživatelovi seznam všech generátorů, které umí naplnit určitou strukturu, takový seznam by se mu obtížně sestavoval. Přitom se jedná o funkci, u které lze předpokládat, že se bude v aplikacích pro generování dat vyskytovat. Proto framework nabízí ještě alternativu v podobě *SupportSettings*.

Třída *SupportSettings* umožňuje spravovat informace o kompatibilitě tříd z balíků *Structures*, *Generators* a *Output* centrálně pomocí práce nad seznamem objektů vnitřní třídy *SupportStructure*. Na kompatibilitu se dívá z pohledu struktury. Instance *SupportStructure* obsahuje název třídy

struktury (*structureName*) a seznamy názvů tříd generátorů a výstupů (*generatorNames*, *outputNames*), které je možno této struktuře přiřadit. Atributy a operace obou tříd si můžeme prohlédnout na *obrázku 5.7*.




*Obrázek 5.7: Třída SupportSettings se svou vnitřní třídou SupportStructure*

Pomocí metod *load()* a *save()* můžeme obsah *SupportSettings* uložit do nebo načíst ze souboru ve formátu XML. Díky tomu lze poměrně pohodlně nastavení prohlížet a upravovat i v libovolném textovém editoru. Metoda *getStructuresNames()* vrací seznam jmen všech tříd struktur, metody *getGeneratorNames()* a *getOutputNames()* pak seznam jmen všech tříd generátorů/výstupů pro zadanou třídu struktury. Poslední metoda *print()* vypíše obsah *SupportSettings* na standardní výstup.

*SupportSettings* lze využít nejen jako zdroj informací o kompatibilitě tříd, ale také k registraci tříd do aplikace. Pouhou změnou obsahu XML souboru tak pro jednu a tu samou aplikaci dosáhneme zpřístupnění různých tříd a tím i různých možností generování. Můžeme mít například aplikaci podporující generování dat z několika tematických oblastí a pro každou oblast speciální XML soubor, který nám znepřístupní třídy nesouvisející se zvolenou oblastí.

Třídu lze samozřejmě dle potřeby rozšířit o ukládání dalších konfiguračních dat, nebo pro tato data vytvořit vlastní nástroj.

# 6 Analýza a návrh demonstrační aplikace

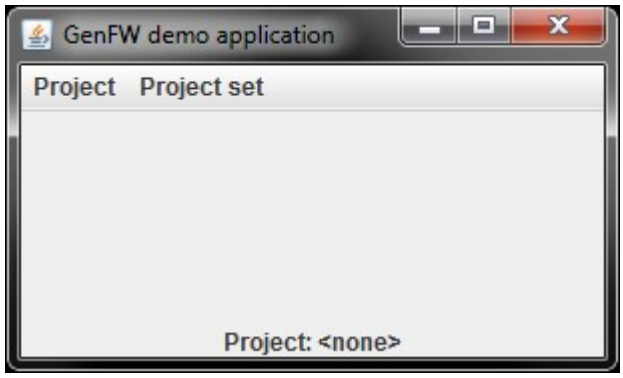
Kromě samotného frameworku vytvoříme ještě jednoduchou aplikaci, která má ověřit a demonstrovat funkčnost navrženého frameworku. Její vývoj proběhne zároveň s vývojem frameworku. Aplikace tak zároveň poslouží jako nástroj pro průběžné testování implementovaných funkcí. Důraz by měl být kladen na srozumitelnost, přehlednost a rychlou dostupnost demonstrovaných funkcí.

## 6.1 Uživatelské rozhraní

První částí demonstrační aplikace je grafické uživatelské rozhraní (GUI). Skládá se pouze ze dvou tříd umístěných v balíku s názvem GUI. Jedná se o standardní hlavní okno s funkcemi přístupnými přes rozbalovací menu. Do tohoto okna je při načtení projektu přidán panel zobrazující jeho obsah.

### 6.1.1 Hlavní okno

Hlavní okno (třída *MainWin*) má dvě základní části. Panel menu, zabírající tenký pruh horní části okna, a kontejner pro zobrazení obsahu projektu, pokrývající zbytek plochy okna. Dokud není načten žádný projekt, tento kontejner je prázdný s výjimkou v dolní části umístěného stavového řádku, zobrazujícího název načteného projektu – *obrázek 6.1*.




*Obrázek 6.1: Hlavní okno před načtením projektu*

Menu je rozděleno na dvě části. První, s názvem *Project*, se věnuje práci se samostatným projektem, bez jeho začlenění do skupiny projektů. Pracuje se zde tedy přímo s instancí třídy *Project*. Obsahuje tlačítka pro:

- uložení projektu do souboru – *Save to file...*
- načtení uloženého projektu ze souboru – *Load from file...*
- načtení demonstračního projektu s názvem *UDP packets – Load UDP demo*
- spuštění projektu – *Run*
- ukončení aplikace – *Exit*

Dokud není načten žádný projekt, tlačítka pro uložení a spuštění projektu nejsou stisknutelná, protože jejich akce nelze provést. Akce uložení a načtení ze souboru spouští standardní dialog pro

práci se soubory. Při načtení demonstračního projektu se objeví dvě dialogová okna pro nastavení parametrů projektu. Demonstračnímu projektu bude věnováno více prostoru v kapitole 6.2.1. Spuštění projektu volá metodu *run()* třídy *Project* a provede tedy druhou, třetí a čtvrtou fázi procesu generování, jak byly popsány v kapitole 3. Akce ukončení aplikace přímo nesouvisí s třídou *Project*, ale objevuje se zde, protože je zvykem, že aplikace tuto akci na posledním místě nejlevějšího rozbalovacího menu obsahuje.

Druhá část menu, *Project set*, pracuje se skupinou projektů, tedy s instancí třídy *ProjectSet*. Skrývá pouze dvě tlačítka, věnovaná druhému demonstračnímu příkladu *DBTables*:

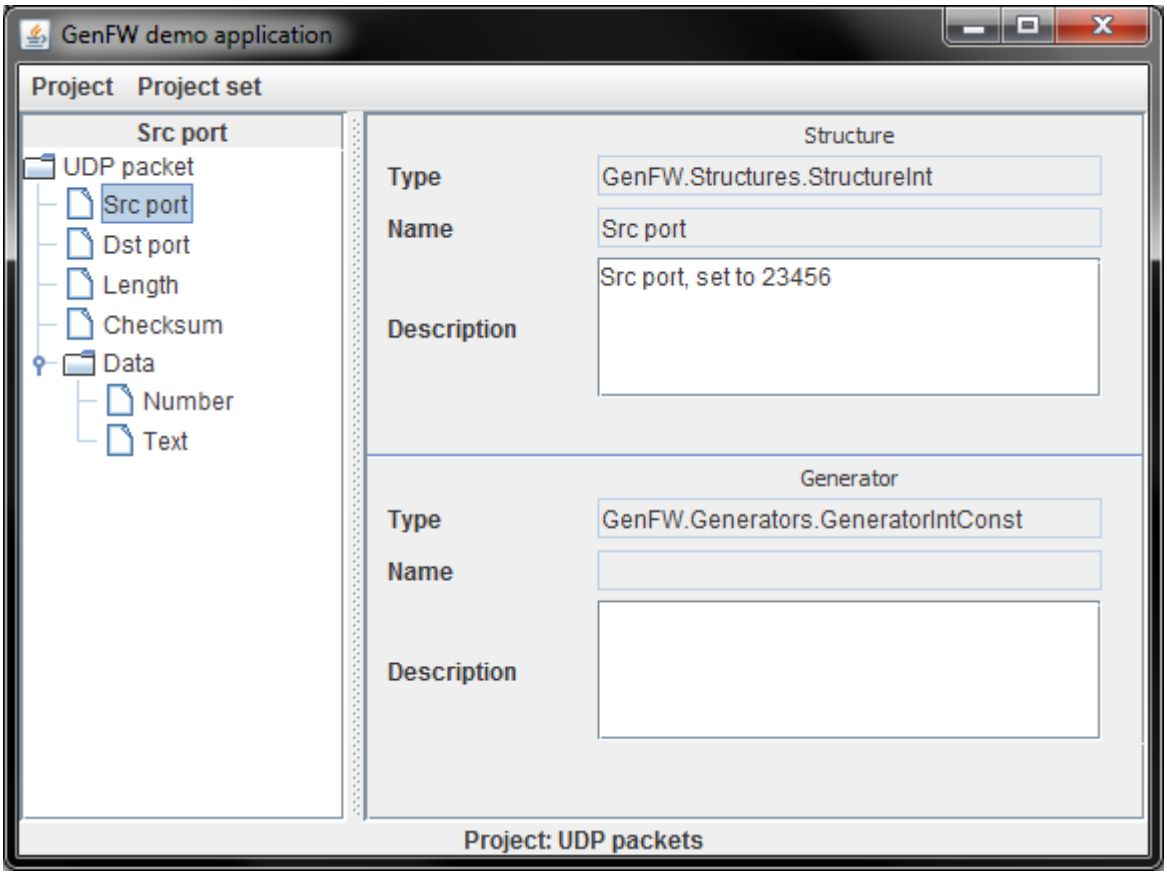
- načtení *DBTables* a jejich spuštění s výstupem do .txt souboru - *DBTables to txt*
- načtení *DBTables* a jejich spuštění s výstupem do .sql souboru - *DBTables to sql*

Tlačítka jsou dostupná po celou dobu běhu aplikace. Obě provedou kompletní proces generování od vytvoření struktur až po odeslání dat na výstup. První dvě fáze procesu mají společné, liší se ale ve třetí a čtvrté fází. Zatímco *DBTables to txt* připravuje vygenerovaná data pro přehledné zobrazení ve formátu .txt, *DBTables to sql* je obalí do SQL skriptu, který pak může být použit pro vložení vygenerovaných dat do databáze. Jejich výstupem jsou soubory *car.txt* a *owner.txt*, resp. *car.sql* a *owner.sql.*

### 6.1.2 Panel projektu

Druhou částí GUI je panel (třída *PanelProject*), který po načtení projektu vyplní dosud nevyužitou část hlavního okna aplikace. Na levé straně panelu se nachází strom struktur načteného projektu, vpravo pak formulář se základními informacemi o aktuálně označené struktuře. Posuvník mezi stromem a formulářem dovoluje upravit poměr šířek těchto dvou komponent. Na *obrázku 6.2* vidíme hlavní okno aplikace s načteným projektem a zobrazeným panelem projektu.

Úspěšným načtením projektu dojde nejen k vytvoření a zobrazení panelu, ale jsou také zpřístupněny tlačítka menu pro uložení a spuštění projektu. V jeden okamžik je možno mít načtený maximálně jeden projekt. Načtením druhého projektu dojde automaticky k uzavření prvního projektu a aktualizaci panelu projektu.




*Obrázek 6.2: Hlavní okno po načtení projektu UDP packets*

### 6.1.3 Spuštění v konzoli

Součástí uživatelského rozhraní je i možnost spuštění demonstrační aplikace pouze v konzoli bez grafických prvků. Toho lze dosáhnout použitím argumentů příkazové řádky. Následuje výčet podporovaných argumentů a jejich krátký popis:

| | |
|---|---|
| `-p project.p` | načte a spustí projekt uložený v souboru `project.p` |
| `-ps projset.ps` | načte a spustí skupinu projektů uložených v souboru `projset.ps` |
| `jiné parametry` | vypíše nápovědu |
| `bez parametrů` | otevře hlavní okno aplikace |

Kód obsluhující konzolový provoz je součástí třídy *MainWin*. Po úspěšném načtení projektu (nebo skupiny projektů) dojde ještě před jeho (jejím) spuštěním k výpisu základních informací na standardní výstup zavoláním metody *printInfo()*. Po dokončení procesu generování se aplikace sama ukončí.

## 6.2 Demonstrační příklady

Jak je již patrné ze struktury menu, demonstrační aplikace obsahuje dva příklady. Jeden příklad samostatného projektu s názvem *UDP packets* a jeden příklad skupiny projektů pojmenovaný

*DBTables*. Oba dohromady tvoří balík *Examples*, ve kterém má každý z nich svůj vlastní balík obsahující veškeré potřebné specializované třídy struktur, generátorů a výstupů. Navíc je v každém z těchto balíků třída zajišťující správné vytvoření a nastavení příkladu.

Cílem příkladů není ohromit složitými konstrukcemi ani kvanty vygenerovaných dat s množstvím složitých vazeb, ale naopak co nejjednodušeji a nejsrozumitelněji demonstrovat funkce navrženého frameworku a možnosti, které nabízí.

### 6.2.1 UDP pakety

Prvním z demonstračních příkladů je generování paketů protokolu UDP – balík *UDPPackets*. UDP paket má pevně stanovený formát, popsaný v dokumentu RFC 768 [9]. Skládá se z hlavičky a datové části. Hlavičku tvoří čtyři 16bitová čísla, ale vyplnění je povinné pouze u dvou z nich (cílového portu a délky paketu). Datová část pak může obsahovat libovolná data do velikosti necelých 64kB. Schéma formátu UDP paketu vidíme na *obrázku 6.3*.

| **Offset (bitů)** | **0 - 15** | **16 - 31** |
|---|---|---|
| **0** | Zdrojový port | Cílový port |
| **32** | Délka | Kontrolní součet |
| **64+** | Data | |

*Obrázek 6.3: Schéma formátu UDP paketu*

Generování UDP paketů bylo jako příklad vybráno hned z několika důvodů. Díky jednoduchému a pevně danému formátu hlavičky se snadno zorientujeme ve stromu struktur, který UDP paketu odpovídá. Všechny struktury mohou být naplněny konstantními hodnotami, a přesto se ve výsledku jedná o validní paket, který může být odeslán na síť. Síťové rozhraní je typickým příkladem rozhraní, u kterého si lze snadno představit potřebu předem neomezeného počtu generovaných záznamů. Možnost časově neomezeného generování je přitom jedním z požadavků na vyvíjený framework. Datovou část lze naplnit téměř čímkoliv, jediným omezením je její maximální velikost. UDP paket lze snadno reprezentovat pouze instancí třídy *Project*, není třeba využívat třídu *ProjectSet*.

Tento příklad demonstruje způsob použití základních tříd frameworku, tvorbu jejich potomků, nastavení atributů a propojení vytvořených instancí do fungujícího celku. Z vlastností frameworku pak ukazuje především:

- oddělení prvních tří fází procesu generování dat – pro strukturu lze použít různé generátory a pro vygenerované hodnoty různé formy výstupu
- podporu vícepatrového stromu struktur – modelem UDP paketu je strom o třech patrech
- práci se souborem – uložení a načtení projektu
- nekonečné generování – projekt je možno nastavit, aby generoval pakety, dokud neobdrží příkaz k zastavení
- výstup na síťové rozhraní – jednou z podporovaných forem výstupu je odeslání dat na síť

Při načtení příkladu přes tlačítko *Load UDP demo* se objeví dvě dialogová okna. První, pro nastavení výstupního formátu, dá na výběr ze tří forem výstupu – textový soubor, standardní výstup a odeslání na síť. Po zvolení možnosti *TXT File* bude v aktuálním adresáři vytvořen soubor *UDP.txt*

a obsah struktur se bude v čitelné podobě generovat do něj. Volba *Stdout* odesílá stejný text na standardní výstup. Tlačítko *Localhost* pakety pošle na přednastavený port číslo 34567 adresy lokálního síťového zařízení. Druhým dialogovým oknem se aplikace zeptá, jestli má nastavit nekonečné generování. Odpovědí *No* zvolíme generování vzorku pěti paketů, odpovědí *Yes* generování až do okamžiku zastavení zásahem uživatele. Zvolená kombinace výstupu a počtu generovaných paketů je uložena k projektu. Pokud pak projekt uložíme do souboru volbou *Save to file...*, bude v souboru zaznamenáno i toto nastavení a při načtení souboru (*Load from file...*) bude automaticky aplikováno.

### 6.2.2 Jednoduché tabulky

Druhý demonstrační příklad (balík *DBTables*) generuje data pro dvě tabulky. V tabulce *Owner* se nachází jména, příjmení a telefonní čísla osob. Tyto osoby mohou vlastnit automobily. Údaje o automobilech obsahuje druhá tabulka s názvem *Car*. Pro každé auto ukládá jeho výrobce, model, datum prodeje a vlastníka – osobu z tabulky *Owner*. Obě tabulky mají primární klíč v podobě sloupce *ID*, vlastník je v tabulce *Car* reprezentován sloupcem *Owner_ID* s cizím klíčem odkazujícím na sloupec *ID* tabulky *Owner*. Pro každého výrobce existuje seznam modelů aut, které vyrábí. Pro vývojáře bude možná srozumitelnější formou popisu níže přiložený SQL kód, kterým lze tabulky vytvořit:

```
CREATE TABLE `OWNER` (
  `ID` bigint(20) NOT NULL AUTO_INCREMENT,
  `NAME` varchar(20) NOT NULL,
  `SURNAME` varchar(20) NOT NULL,
  `PHONE` bigint(20) NOT NULL,
  PRIMARY KEY (`ID`)
);

CREATE TABLE `CAR` (
  `ID` bigint(20) NOT NULL AUTO_INCREMENT,
  `MANUFACTURER` varchar(20) NOT NULL,
  `MODEL` varchar(20) NOT NULL,
  `PURCHASE_DATE` date NOT NULL,
  `OWNER_ID` bigint(20) NOT NULL,
  PRIMARY KEY (`ID`),
  KEY `OWNER_ID` (`OWNER_ID`)
);

ALTER TABLE `CAR`
  ADD CONSTRAINT `CAR_FK_1` FOREIGN KEY (`OWNER_ID`)
  REFERENCES `OWNER` (`ID`);
```

I pro výběr druhého příkladu existuje hned několik důvodů. Výše uvedené tabulky se věnují stručnému popisu obecně známých vztahů dvou typů objektů reálného světa – osoby a auta. Tabulky

jsou základem relačních databází, často používaného nástroje pro uložení dat. Druhý příklad má vlastnosti, kterými se liší od prvního a vhodně jej tak doplňuje:

- data obsahují datové závislosti
- díky dvěma tabulkám se nabízí použití třídy *ProjectSet*
- formát tabulky není omezen tak přísně jako formát UDP paketu

Protože funkce nekonečného výstupu a uložení do souboru již byly demonstrovány u prvního příkladu, druhý příklad tyto možnosti nenabízí. Veškeré parametry jsou přednastaveny, celý proces generování proběhne stiskem jediného tlačítka. Vygeneruje se 10 záznamů tabulky *Owner* a 15 záznamů tabulky *Car*. Zvolit lze pouze formu výstupu – textové soubory *owner.txt* a *car.txt* s řádkem hlavičky tabulky a daty zarovnanými do sloupců nebo soubory SQL skriptů *owner.sql* a *car.sql* pro vložení vygenerovaných dat do databáze.

# 7 Implementace a testování

Implementace frameworku a demonstrační aplikace je založena na analýze a návrhu popsaných v předchozích kapitolách. Použité prostředky i zvolené postupy byly vybírány tak, aby respektovaly požadavky na vstřícnost k vývojáři, ať už z pohledu jednoduchosti a srozumitelnosti, dostupnosti nebo přenositelnosti.

Pro implementaci projektu byl zvolen programovací jazyk Java. Podle indexu *Tiobe* [10] jde o nejpopulárnější jazyk posledních deseti let. Mezi jeho výhodami bývá zmiňována přenositelnost, jednoduchost, množství dostupných knihoven komponent, knih a dalších studijních materiálů [11], [12]. Jako vývojové prostředí sloužilo známé a používané *Eclipse IDE* s rozšířením pro tvorbu grafických aplikací *WindowBuilder* [13]. Zdrojem grafických komponent byla především knihovna *Swing*, která je již od verze 1.2 součástí platformy *Java Standard Edition* [14], na formulář panelu projektu byla využita ještě knihovna *Forms* společnosti *JGoodies* [15]. Komentáře jsou psány ve formátu zpracovatelném technologií *Javadoc*, což umožňuje automatické generování dokumentace. Veškerý kód včetně komentářů je psaný v anglickém jazyce.

Z důvodu snadného uložení a načtení celého projektu nebo skupiny projektů pomocí tříd *ObjectInputStream* a *ObjectOutputStream* [16] všechny třídy frameworku implementují rozhraní *Serializable*. Pro potřeby ladění a testování je k dispozici logovací mechanismus, který zaznamenává průchod vybranými metodami a lze jej aktivovat zvlášť pro každou fázi procesu generování.

Testování probíhalo převážně souběžně s implementací spouštěním implementovaných funkcí v demonstrační aplikaci a kontrolou správného běhu pomocí pohledu *Debug perspective* prostředí *Eclipse IDE* a logovacího mechanismu. Funkčnost výstupu na síťové rozhraní u příkladu *UDPPackets* byla ověřena analyzátorem *Wireshark* [17]. Aplikace *phpMyAdmin* [18] pak posloužila při kontrole výstupu ve formátu SQL skriptu z příkladu *DBTables*. Demonstrační aplikace byla úspěšně otestována na operačních systémech *Windows 7 Professional, Windows XP SP3* a *Ubuntu 10.10* s nainstalovaným JRE ve verzi 1.7.0.

# 8 Použití frameworku

V předchozích kapitolách byl framework popsán z pohledu vývoje software, přičemž nejvíce prostoru bylo věnováno jeho návrhu. Zde se na něj pokusíme nahlédnout očima uživatele-vývojáře a řekneme si, kdy a jakým způsobem jej můžeme využít.

Jak již bylo zmíněno dříve, framework má poskytnout podporu při vývoji specializovaných generátorů dat. Základním předpokladem pro jeho použití je tedy potřeba tvorby syntetických dat a rozhodnutí vyrobit si k tomuto účelu vlastní nástroj. Dalším předpokladem je pak alespoň zběžná orientace v objektovém programování a jazyce Java.

## 8.1 Zdroje informací

Ještě před začátkem vývoje vlastního generátoru je vhodné seznámit se s myšlenkami, na kterých je framework postaven. K tomu můžou posloužit úvodní kapitoly této práce. Při vlastním vývoji lze jako zdroje informací využít opět textu této práce, zejména kapitol věnujících se návrhu frameworku a demonstrační aplikace. Dalšími zdroji mohou být vygenerovaná *Javadoc* dokumentace nebo komentovaný zdrojový kód demonstrační aplikace, kde jsou funkce frameworku použity v jednoduchých příkladech.

## 8.2 Vývoj vlastního generátoru

Vývoj generátoru lze rozdělit na tři části:

- vytvoření potřebných tříd struktur, generátorů a výstupů orientovaných na oblast, pro kterou chceme data generovat
- vytvoření projektů a skupin projektů
- vytvoření aplikace pro pohodlnou správu projektů

Jednotlivé části vývoje přitom nemusí, a s největší pravděpodobností nebudou, probíhat v uvedeném pořadí. Výběr modelu životního cyklu vývoje generátoru i jeho částí je ponechán čistě na rozhodnutí vývojáře.

První část framework podporuje balíky *Structures*, *Generators* a *Output*, kde jsou implementovány obecně použitelné třídy, ze kterých lze při tvorbě vlastních tříd dědit. V demonstrační aplikaci se může vývojář inspirovat několika ukázkami specializovaných tříd a využitím atributů *properties* nebo *globalObjects* pro předávání informací a řešení datových závislostí.

Pro druhou část vývoje je určen balík *Project*. Jednoduché generátory si mohou vystačit pouze s jeho třídami *Project* a *ProjectSet*, jako se to podařilo příkladům *UDPPackets* a *DBTables*. Jestliže poskytovaná funkčnost nebude dostačující, nabízí se opět možnost dědění a rozšíření těchto tříd.

Třetí část lze pojmout mnoha různými způsoby a její podoba bude silně ovlivněna přístupem vývojáře k částem předchozím. Pokud jsou projekty předem připraveny a nastaveny, stačí velmi prostá konzolová aplikace pro jejich spouštění. Takové projekty jsou však použitelné velice úzce. Na druhou stranu čím větší volnost a konfigurovatelnost připustíme, tím rozsáhlejší a složitější bude aplikace, která má všechny tyto možnosti nabídnout.

U demonstračních příkladů šlo především o jednoduchost a srozumitelnost. Díky tomu mohla být i aplikace poměrně stručná a jednoduchá. Zobrazením stromu struktur projektu a formuláře pro nastavení struktury (kapitola 6, strana 25, *obrázek 6.2*) ale naznačuje cestu složitějším aplikacím. Dynamické sestavování obsahu formuláře na základě vybrané struktury a možnost editace stromu struktur mohou být prvními kroky při tvorbě větších aplikací. S tím jdou ovšem ruku v ruce i rostoucí nároky na vývoj ostatních částí generátoru.

# 9 Zhodnocení výsledků

Kapitola nejprve popíše a zhodnotí výsledky, kterých bylo do této chvíle dosaženo. Poté se zamyslí nad případným pokračováním v práci. Nastíní možnosti rozšíření a směr, kterým se může další vývoj ubírat.

## 9.1 Dosažené výsledky

Framework poskytuje podporu při vývoji různých druhů generátorů dat, čímž splňuje základní cíl, který byl uveden v úvodu práce. Je navržen dostatečně obecně, aby vývojáře zbytečně neomezoval a nebyl vázán na konkrétní typ nebo formát generovaných dat. Obsahuje kód obsluhující všechny fáze procesu generování, jak byly definovány v kapitole 3. Kód je psán s ohledem na rozšiřitelnost srozumitelnou formou s komentáři, třídy jsou rozděleny do tematických balíčků. Použité technologie zaručují přenositelnost, která byla během práce vyzkoušena testováním na třech různých operačních systémech.

Spolu s frameworkem byla vyvinuta ještě demonstrační aplikace, která ověřila jeho funkčnost a posloužila pro průběžné testování implementovaných funkcí. Součástí této aplikace jsou dva příklady z různých oborů. Jejich úkolem je ukázka využití frameworku při řešení konkrétních případů generování dat. Vývojářům by aplikace měla pomoci zejména jako materiál pro rychlejší orientaci a zkrátit tak čas potřebný pro studium frameworku.

## 9.2 Možnosti dalšího vývoje

Možná se podařilo sepsat několik nejdůležitějších myšlenek a položit základ prostředí pro podporu vývoje generátorů. Téma je ale natolik obsáhlé, že započatá práce může pokračovat ještě velmi dlouho.

Zaměříme-li se na úpravy a vylepšení menšího rozsahu, nabízí se například funkce uložení vygenerovaných dat pro pozdější formátování, vylepšení systému logování a výjimek, přidání mechanismů pro práci se seznamy hodnot (předpřipravené seznamy, načítání seznamů ze souboru, různé způsoby výběru hodnoty atd.) nebo s rozdělením pravděpodobnosti.

V balících *Structures* a *Generators* najdeme kromě základních tříd i několik jejich potomků. Jedná se sice o třídy pro již konkrétní datové typy nebo struktury, ale díky jejich obecné použitelnosti byly zařazeny přímo do frameworku. Doplnění dalších obecných tříd by také určitě bylo krokem kupředu.

Náročnější rozšíření by mohla zahrnovat pokročilé nástroje pro řešení datových závislostí nebo podporu pro tvorbu rozsáhlých aplikací pro správu projektů. Samostatnou kapitolou jsou pak tematicky zaměřené balíky vně frameworku. U nich jsou možnosti vývoje a rozšiřování téměř nekonečné.

# 10 Závěr

Cílem diplomové práce bylo prostřednictvím frameworku poskytnout podporu při vývoji specializovaných generátorů dat. Řešení začalo semestrálním projektem, v rámci kterého bylo nastudováno dané téma, stručně jsme se seznámili s několika existujícími generátory a rozebrali problém generování dat. Ten byl definován jako proces složený z vytvoření datových struktur následovaného naplněním těchto struktur vhodnými hodnotami, zformátováním do požadované podoby a odesláním na výstup. Dále byly určeny požadavky na vyvíjený framework a popsán návrh jeho řešení s ohledem na obecnost postupu generování, jednoduchost a přehlednost.

Diplomová práce vychází ze semestrálního projektu. Rozšiřuje návrh frameworku, rozvádí ho do větších podrobností. Přidává návrh demonstrační aplikace, která ověřuje a předvádí funkčnost frameworku na jednoduchých příkladech a zároveň sloužila jako nástroj pro průběžné testování. Po kapitole popisující zejména nástroje použité při implementaci a testování obou návrhů následuje text věnovaný způsobu použití frameworku, zhodnocení dosažených výsledků a možnostem budoucích rozšíření a vylepšení.

Jako celek se diplomová práce věnuje postupně všem bodům zadání od nastudování problematiky, přes návrh a implementaci frameworku až po ilustraci použitelnosti a zhodnocení výsledků. Vyvinutý framework postihuje celý proces generování, ukazuje obecný přístup k problému a systémem balíků pro jednotlivé fáze procesu poskytuje přehledné a snadno rozšiřitelné řešení.

# Literatura


[1] *Test (Sample) Data Generators*. [online]. 2008.
Dostupné z: http://www.webresourcesdepot.com/test-sample-data-generators/

[2] *DTM Database Tools*. [online]. 1998 – 2012. Dostupné z: http://www.sqledit.com

[3] *DTM Data Generator Online Documentation*. [online]. 2004 – 2011.
Dostupné z: http://www.sqledit.com/dg/help/index.html

[4] *Datamaker*. [online]. 2010. Dostupné z: http://www.grid-tools.com/products/datamaker.php

[5] *Latest version of GT Datamaker slashes inheritance tax for database developers*. [online]. 2006. Dostupné z: http://www.sourcewire.com/releases/rel_display.php?relid=24892

[6] MICHALAKOPOULOS, M. *DgMaster - data generator*: *simple, free, extensible.* [online]. 2007 - 2009. Dostupné z: http://dgmaster.sourceforge.net/

[7] HRUŠKA, T., KŘIVKA, Z. *Informační systémy (IIS,PIS)*: *Pojem informačního systému, data, procesy, transakce*. Listopad 2008. Studijní opora. FIT VUT Brno.

[8] ŠLAPÁK, O. *Data, informace, znalosti. E-logos / Katedra filosofie*. 2003. ISSN 1211-0442.
Katedra informačních technologií VŠE.
Dostupné z: nb.vse.cz/kfil/elogos/miscellany/slapa103.pdf

[9] POSTEL, J. *RFC 768 - User Datagram Protocol*. [online]. Srpen 1980.
Dosupné z: http://tools.ietf.org/html/rfc768

[10] *TIOBE Software: Tiobe Index*. [online]. 2012.
Dostupné z: http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

[11] *The Top 10 Programming Languages - IEEE Spectrum*. [online]. Říjen 2011.
Dostupné z: http://spectrum.ieee.org/at-work/tech-careers/the-top-10-programming-languages

[12] *Top 10 Most Popular Programming Languages - English 4 IT Reading Activity*. [online]. 2012.
Dostupné z: http://www.english4it.com/reading/40

[13] *Window Builder*. [online]. 2012.
Dostupné z: http://www.eclipse.org/windowbuilder/

[14] *Java SE Technologies at a Glance*. [online]. 2012.
Dostupné z: http://www.oracle.com/technetwork/java/javase/tech/index.html

[15] *JGoodies FormLayout | JGoodies*. [online]. 2012.
Dostupné z: http://www.jgoodies.com/freeware/libraries/forms/

[16] *Object Streams (The Java™ Tutorials > Essential Classes > Basic I/O)*. [online]. 2012.
Dostupné z: http://docs.oracle.com/javase/tutorial/essential/io/objectstreams.html

[17] *Wireshark · Go deep*. [online]. 2012.
Dostupné z: http://www.wireshark.org/

[18] *phpMyAdmin*. [online]. 2012.
Dostupné z: http://www.phpmyadmin.net/home_page/

# Seznam příloh

Příloha 1. CD s elektronickou verzí technické zprávy, zdrojovými kódy a spustitelným JAR archivem