VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

GENERÁTOR KÓDU PRO TESTOVACÍ RÁMEC REDDEER
CODE GENERATOR FOR REDDEER TEST FRAMEWORK

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **Dominik Jelínek**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Petr Číka, Ph.D.**
SUPERVISOR

BRNO 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní obor **Teleinformatika**
Ústav telekomunikací

Student: Dominik Jelínek ***ID:*** 177264
Ročník: 3 ***Akademický rok:*** 2016/17

NÁZEV TÉMATU:

Generátor kódu pro testovací rámec RedDeer

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s vývojovým prostředím Eclipse a jeho architekturou. Zaměřte se zejména na tvorbu zásuvných modulů a vytvořte základní zásuvný modul s jednoduchým uživatelským rozhraním.
Dále se seznamte s testovacím rámcem RedDeer, který se používá na automatizování testů pro zásuvné moduly vývojového prostředí Eclipse. Popište základní principy testovacího rámce RedDeer a navrhněte řešení, jak lze ulehčit práci psaní automatizovaných testů pomocí generovaného kódu. Navrhnuté řešení implementujte jako rozšíření zásuvného modulu vytvořeného v první části práce.

DOPORUČENÁ LITERATURA:

[1] VOGEL, Lars. Contributing to the Eclipse IDE Project: Principles, Plugins and Gerrit Code Review. Lars Vogel: vogella series, 2015, 230 s, ISBN 97839437471571

[2] Příručka testovacího rámce RedDeer [online dokumentace]. ©2016 [cit. 20160923]. Dostupné z: http://jboss-reddeer.github.io/reddeer/

Termín zadání: 1.2.2017 ***Termín odevzdání:*** 8.6.2017

Vedoucí práce: Ing. Petr Číka, Ph.D.
Konzultant: Mgr. Andrej Podhradský

doc. Ing. Jiří Mišurec, CSc.
předseda oborové rady

ABSTRAKT

Bakalářská práce se zabývá návrhem a tvorbou nového zásuvného modulu do vývojového prostředí *Eclipse*. V teoretické části práce je popsána architektura vývojového prostředí *Eclipse*, jsou zde probrány grafické knihovny programovacího jazyka *Java* a vysvětleny možnosti rozšiřování vývojového prostředí *Eclipse* o vlastní zásuvné moduly. Dále jsou v teoretické části práce vysvětleny způsoby testování aplikací a popsán způsob sestavování (kompilování) aplikací. Popis praktické části práce se zabývá vývojem zásuvného modulu, návrhem interní logiky pro generování kódu a posléze implementací navrženého řešení generátoru kódu pro testovací rámec *RedDeer*.

KLÍČOVÁ SLOVA

Eclipse, RedDeer, Plug-in, SWT, JFace, Apache Maven

ABSTRACT

This bachelor's thesis describes the design and creation of a new *Eclipse* plug-in. The theoretical part of the thesis describes the architecture of the *Eclipse* development environment, discusses *Java* graphical libraries and explains options for extending the *Eclipse* development environment through its own new plug-ins. Next part of the thesis describes types of application testing and presents ways of application compiling. Practical part of the thesis explains the development of a plug-in, details of internal logic for code generating and describes the implementation of the designed solution for the code generator for the *RedDeer Test Framework*.

KEYWORDS

Eclipse, RedDeer, Plug-in, SWT, JFace, Apache Maven

JELÍNEK, Dominik. *Generátor kódu pro testovací rámec RedDeer*. Brno, 2017, 55 s. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: Ing. Petr Číka, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Generátor kódu pro testovací rámec RedDeer“ jsem vypracoval(a) samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor(ka) uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil(a) autorská práva třetích osob, zejména jsem nezasáhl(a) nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom(a) následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora(-ky)

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce panu Ing. Petru Číkovi, Ph.D. a externímu odbornému konzultantovi ze společnosti Red Hat panu Mgr. Andreji Podhradskému za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Brno ....

podpis autora(-ky)

OBSAH

Úvod **13**

1 Vývojové prostředí Eclipse **14**
1.1 Architektura vývojového prostředí *Eclipse* . . . 14
1.2 Pracovní prostor a zdroje . . . 17
1.3 Verze *Eclipse* . . . 18

2 Grafické knihovny **20**
2.1 Knihovny *AWT* a *Swing* . . . 20
2.2 Knihovna *SWT* . . . 20
2.3 Knihovna *JFace* . . . 22

3 Rozšiřování vývojového prostředí Eclipse **23**
3.1 Vývoj zásuvných modulů . . . 23
3.1.1 Funkce . . . 25
3.2 Instalace nových zásuvných modulů . . . 26
3.2.1 Instalace prostřednictvím vývojového prostředí *Eclipse* . . . 26
3.2.2 Instalace prostřednictvím exportovaného souboru . . . 26
3.2.3 Instalace prostřednictvím instalačního *Eclipse* manažeru . . . 27

4 Testování aplikací **30**
4.1 Automatizované testování . . . 30
4.2 Testovací rámec *RedDeer* . . . 31

5 Sestavení a kompilace aplikace **32**
5.1 Zásuvný modul *Tycho* . . . 34

6 Vývoj generátoru kódu pro testovací rámec RedDeer **35**
6.1 Příprava pro vývoj zásuvného modulu . . . 35
6.2 Postup implementace generátoru kódu . . . 36
6.3 Grafické uživatelské rozhraní . . . 37
6.4 Interní logika generování kódu . . . 40
6.5 Instalace do vývojového prostředí . . . 42
6.6 Experimentální ověření . . . 42

7 Závěr **44**

Literatura **46**

Seznam symbolů, veličin a zkratek **49**

Seznam příloh **50**

A Návod pro vytvoření jednotlivých částí projektu **51**

B Obsah přiloženého DVD **55**

SEZNAM OBRÁZKŮ

1.1 Blokové schéma architektury projektu *Eclipse*. 15
1.2 Okno aplikace vývojového prostředí (*Workbench window*). 16
1.3 Stromová struktura projektů ve vývojovém prostředí *Eclipse*. 17
1.4 Logická struktura rozhraní pro práci se zdroji v *Eclipse*. 18
2.1 Widgety z knihovny *SWT*. 21
2.2 Stránka možností nastavení (*preference page*). 22
3.1 Dialogové okno exportu. 27
3.2 Export zásuvného modulu pomocí *Eclipse IDE*. 28
3.3 Dialog pro vytvoření *Feature projektu*. 29
6.1 Diagram vývoje zásuvného modulu. 36
6.2 Struktura *RedDeer CodeGen* projektu. 36
6.3 První strana průvodce generátoru kódu *RedDeer*. 37
6.4 Druhá strana průvodce generátoru kódu *RedDeer*. 38
6.5 Třetí strana průvodce generátoru kódu *RedDeer*. 39
6.6 Diagram navržené interní logiky generování kódu. 40
6.7 Instalace zásuvného modulu *RedDeer CodeGen* do *Eclipse IDE*. . . . 42
6.8 Rozšíření struktury *RedDeer CodeGen* projektu. 43
A.1 Nastavení parametrů *Maven* projektu. 51
A.2 Nastavení parametrů *RedDeer CodeGen* Feature projektu. 51
A.3 Nastavení parametrů *RedDeer CodeGen* Plug-in projektu. 52
A.4 Výsledná struktura *RedDeer CodeGen* projektu. 52
A.5 Nastavení pro spuštění *Maven* buildu. 53
A.6 Úspěšně sestavený projekt. 53

SEZNAM TABULEK

2.1 Manažeři rozvržení grafické knihovny *SWT*. 21
5.1 Základní adresářová *Maven* struktura pro *Java* projekt. 32

SEZNAM VÝPISŮ

3.1 Struktura souboru `plugin.xml`. 24
3.2 `MANIFEST.MF` 25
5.1 Struktura souboru `pom.xml`. 33
6.1 Ukázka výstupu generátoru kódu po uspořádání *Java* třídy. 41

SEZNAM TECHNICKÝCH VÝRAZŮ

Bakalářská práce obsahuje technické výrazy, které jsou ponechány v původním anglickém znění. V českém jazyce výrazy nelze přesně specifikovat nebo není zapotřebí je překládat, protože se jedná o normální pojmy v problematice, kterou se bakalářská práce zabývá. Seznam technických výrazů s krátkým popisem:

Build	aktuální verze programu, kterou je možné sestavit bez chyb, např. pro testování nebo distribuci aplikace
Bundle	běžný *JAR* archiv obsahující několik speciálních hlaviček, které definují všechny závislosti, „životní cyklus“ a status přidaného zásuvného modulu
Display	zajišťuje ovládání všech operací, jako jsou např. uživatelské interakce (klikání na tlačítka apod.)
Eclipse Equinox	základní projekt *Eclipse IDE*, jedná se o specifikaci základního *OSGi* rámce
Eclipse IDE	vývojového prostředí *Eclipse*
Eclipse UI	uživatelské rozhraní vývojového prostředí *Eclipse*
Eclipse update manager	průvodce pro instalování nových zásuvných modulů a rozšíření do vývojového prostředí *Eclipse*
Eclipse Platform	základ celého vývojového prostředí, obsahuje pouze základní nástroje a pár zásuvných modulů
Feature	vlastnost, část z celkové skupiny vlastností ve funkci
Features	funkce, seskupuje více zásuvných modulů do jedné skupiny
JUnit	rámec pro vytváření jednotkových testů v jazyce *Java*
Layout Manager	nebo-li „manažer rozvržení“, umožňuje uspořádat jednotlivé prvky *SWT* dle zvolených pravidel v zobrazované oblasti aplikace
Open Source	otevřený software, počítačový software s volně přístupným zdrojovým kódem
OSGi bundle	běžný *JAR* archiv obsahující několik speciálních hlaviček, které definují všechny závislosti, „životní cyklus“ a status přidaného zásuvného modulu
Perspective	definuje počáteční sadu a rozložení komponent v okně aplikace
Plug-in	nebo-li zásuvný modul, nejmenší samostatná jednotka v *Eclipse*
RedDeer	testovací rámec, který se používá na automatizování testů pro zásuvné moduly vývojového prostředí *Eclipse*

Repository	vzdálené nebo lokální úložiště, které obsahuje soubory *Maven* projektu
Resources	zdroje pro uživatelský prostor
Shell	reprezentuje okno aplikace (zásuvného modulu)
Update site	obsahuje repository, umožňuje pomocí odkazu nainstalovat zásuvný modul do vývojového prostředí
View	grafická komponenta z knihovny *JFace*, která slouží zejména k prezentaci dat, ale rovněž umožňuje navigaci v zobrazených informacích
Widget	část grafického rozhraní aplikace (zásuvného modulu), která umožňuje uživateli interakci s aplikací nebo operačním systémem jako takovým
Wizard	grafická komponenta knihovny *JFace*, jedná se o tzv. průvodce nastavením
Workbench	uživatelská pracovní plocha vývojového prostředí, nebo-li *Workbench window*
Workspace	definuje pracovní prostor, kde se nachází většina uživatelských prostředků, jako jsou projekty, složky, nastavení a další
Workspace root	domovský adresář celé stromové struktury pracovního prostoru (workspace)

ÚVOD

Při vývoji nových aplikací se k ověření správné funkčnosti zdrojových kódů vytvářejí testovací scénáře (tzv. testy). Pro urychlení a větší efektivitu při ověřování správnosti aplikace se používá automatizované testování. Jedním z testovacích rámců pro automatizované testování je testovací rámec *RedDeer* [1].

RedDeer je používaný zejména pro automatizované testování grafického rozhraní vývojového prostředí *Eclipse*. Mezi jeho výhody patří snadné rozšíření o nové vlastnosti a zásuvné moduly.

Zadáním bakalářské práce je analyzovat a navrhnout řešení pro ulehčení vytváření automatizovaných testů pomocí generovaného kódu. Za účelem splnění požadavků a potřeb na generování kódu pro testování grafického rozhraní vývojového prostředí *Eclipse* byl navržen generátor kódu pro testovací rámec *RedDeer* – pojmenovaný „***RedDeer CodeGen***“.

První kapitola teoretické části bakalářské práce se věnuje vývojovému prostředí *Eclipse*. Je zde popsána architektura vývojového prostředí, vysvětlen způsob práce vývojového prostředí se souborovým systémem a distribuce, ve kterých je vývojové prostředí *Eclipse* dostupné.

Navazující druhá kapitola se zabývá grafickými rámci, které jsou používány pro tvorbu uživatelského rozhraní v jazyce *Java*. Jsou zde probrány pouze grafické rámce, které mají spojitost s vývojovým prostředím *Eclipse*.

Třetí kapitola teoretické části popisuje možnosti rozšiřování vývojového prostředí *Eclipse* o nové zásuvné moduly. Součástí této kapitoly jsou způsoby instalace nových zásuvných modulů do vývojového prostředí.

Následuje kapitola, která se zaměřuje na principy a varianty testování aplikací. V této kapitole je představen testovací rámec *RedDeer*.

Pátá kapitola se věnuje nástroji *Apache Maven* [2], který je určen pro sestavení a kompilaci aktuální verze aplikace (zásuvného modulu).

V poslední šesté kapitole je popsán vývoj generátoru kódu pro testovací rámec *RedDeer*, který byl navržen jako řešení zadání bakalářské práce. Součástí této kapitoly je postup vytvoření a instalace zásuvného modulu generátoru kódu do vývojového prostředí, dále popis grafického rozhraní zásuvného modulu vytvořeného generátoru kódu, vysvětlení interní logiky principu generování kódu a ukázka experimentálního ověření funkčnosti vygenerovaných zdrojových kódů.

1 VÝVOJOVÉ PROSTŘEDÍ ECLIPSE

Pod pojmem *Eclipse* [3] se skrývá open source komunita, která je lehce rozšiřitelná o mnoho nástrojů pro vývoj různorodých aplikací za pomoci zásuvných modulů a přídavných rozšíření. Tato komunita zahrnuje velké množství projektů. Pro bakalářskou práci je důležitý zejména projekt *Eclipse (Eclipse Project)*, jehož součástí je *Eclipse SDK (Standard Development Toolkit)*, spíše známé jako *Eclipse IDE (Integrated Development Environment)* – vývojové prostředí *Eclipse*, které je k dispozici pod licencí *EPL (Eclipse Public License)* [4].

Vývojové prostředí *Eclipse* slouží zejména pro vývoj a tvorbu aplikací v jazyce *Java*, ale je navrženo velmi flexibilně, proto ho díky velkému množství zásuvných modulů a rozšíření lze využít nejen pro programování v jazyce *Java*, ale také pro vývoj v programovacích jazycích, mezi něž patří například *C++*, *PHP* a další.

Eclipse IDE je multiplatformní open source aplikace – podporuje všechny nejpoužívanější distribuce operačních systémů (Windows, Mac OS, Linux), což je jedna z hlavních výhod. Mezi další výhody patří lehká rozšiřitelnost nebo možnost individuálně přizpůsobit rozvržení pracovního prostředí dle potřeb programátora.

1.1 Architektura vývojového prostředí *Eclipse*

Celé vývojové prostředí *Eclipse* je postaveno na konceptu zásuvných modulů (plug-inů), které mohou být vzájemně závislé. Zásuvný modul (plug-in) je nejmenší samostatná jednotka v *Eclipse*. O architektuře prostředí *Eclipse* [5] lze hovořit jako o „plug-in architektuře“. Z obrázku 1.1 je patrné rozdělení vývojového prostředí *Eclipse* na tři základní části:

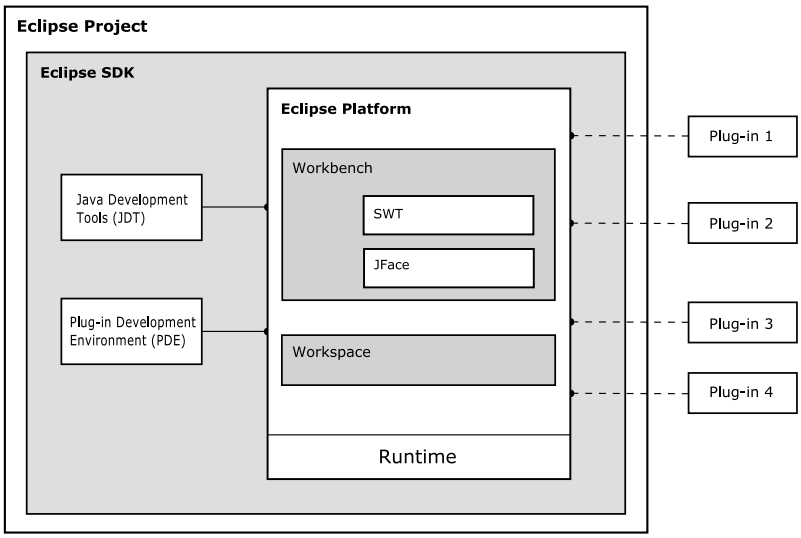
- *Eclipse platforma (Eclipse Platform)*,
- nástroje pro vývoj v jazyce *Java (Java Development Tools, JDT)*,
- prostředí pro vývoj zásuvných modulů (*Plug-in Development Environment, PDE*).

Tyto části tvoří základ projektu *Eclipse*, ve kterém je ***Eclipse Platform*** [6] základním kamenem celého prostředí, obsahuje pouze základní nástroje a pár plug-inů. Platforma *Eclipse* je dále tvořena několika komponentami:

- uživatelská pracovní plocha (*Workbench*),
- pracovní prostor (*Workspace*),
- běhové prostředí *Eclipse* (*Eclipse Runtime*).

Pod pojmem ***Workbench*** je možné si představit uživatelskou pracovní plochu vývojového prostředí nebo-li okno aplikace (*Workbench window*) (obr. 1.2). Na pracovní ploše může být otevřeno několik oken aplikace. Základní části uživatelské pracovní plochy vývojového prostředí tvoří:

- **perspektiva** (perspective) – hlavní perspektivou je *Java Perspective*, definuje počáteční sadu a rozložení komponent v okně aplikace (seskupuje prvky – pohled, editor, menu lišta, nástrojová lišta),
- **pohled** (view) – slouží zejména k prezentaci dat, ale rovněž umožňuje navigaci v zobrazených informacích (např. *Project Explorer*, který zobrazuje projekty a dostupné zdroje v pracovním prostoru),
- **editor** – základní komponenta pro každou perspektivu vývojového prostředí *Eclipse*, je využívána zejména pro práci se soubory (např. *Java editor* nebo obyčejný *Text editor*),
- **menu lišta** (menu bar) – stejná jako u většiny aplikací, obsahuje položky Soubor (File), Upravit (Edit), Nápověda (Help) a další,
- **nástrojová lišta** (toolbar) – zobrazuje ikony pro rychlý přístup k nejvíce používaným funkcím vývojového prostředí, zobrazované ikony lze individuálně přizpůsobit.



Obr. 1.1: Blokové schéma architektury projektu *Eclipse*.

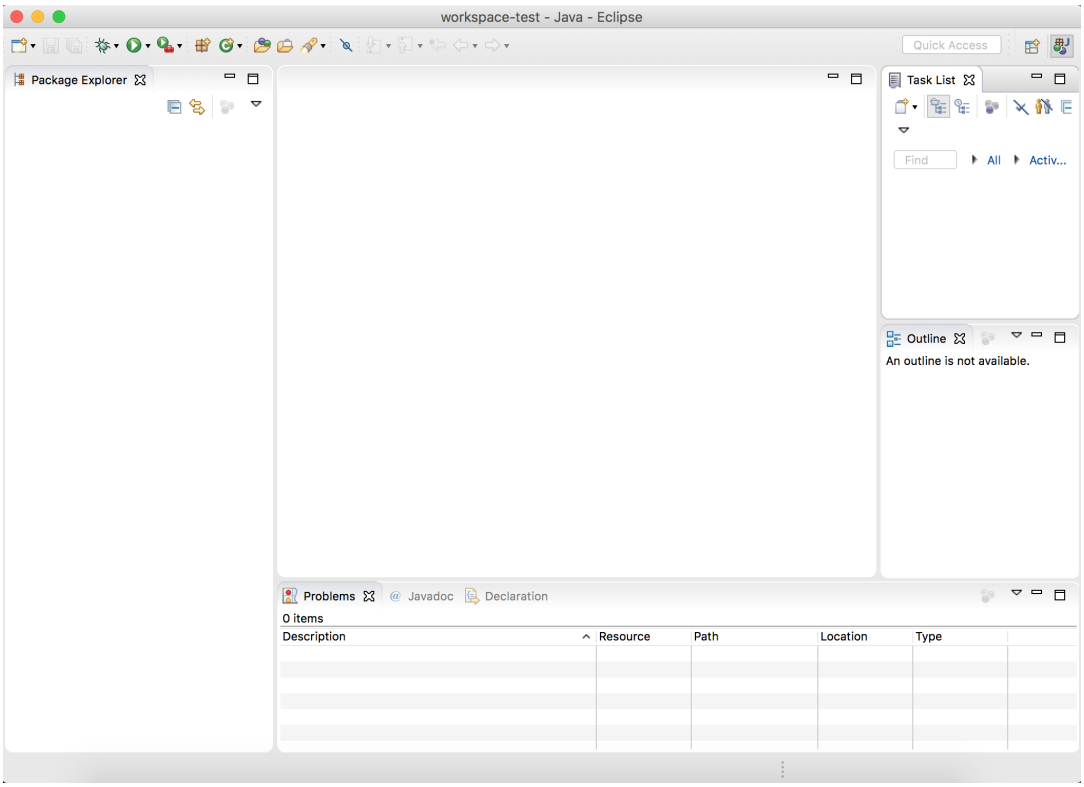
Komponenta ***Workspace*** definuje pracovní prostor, kde se nachází většina uživatelských prostředků, jako jsou projekty, složky, nastavení a další. Tyto položky následně *Eclipse* využívá pro práci.

Komponenta ***Eclipse Runtime*** – „běhové prostředí“ je zodpovědná za strukturu a definici všech dostupných plug-inů, které má prostředí nainstalováno a také má za úkol spouštět samotnou aplikaci *Eclipse*.

Běhové prostředí *Eclipse* je založeno na specifikaci rámce ***OSGi*** [7]. Jedná se o specifikaci modulárního dynamického systému pro programovací jazyk *Java*, který umožňuje v prostředí *Eclipse* přidávání a odebírání zásuvných modulů za běhu aplikace. Základním pojmem v logice *OSGi* je ***OSGi balíček (OSGi bundle)*** [8].

Bundle je běžný *JAR* archiv obsahující několik speciálních hlaviček, které definují všechny závislosti, „životní cyklus“ a status přidaného zásuvného modulu. Každý nově přidaný zásuvný modul je nový *OSGi bundle* v *OSGi* logice.

Pro práci a vývoj aplikací založených na *OSGi* rámci existuje hned několik projektů, ale základem celého *Eclipse IDE* je projekt ***Eclipse Equinox*** [9]. *Eclipse Equinox* je implementací základního *OSGi* rámce a poskytuje pro běh aplikací tzv. *OSGi-based runtime*. Každou část *Eclipse IDE* tedy tvoří *OSGi bundle* (*JDT*, *PDE*, *SWT*, *JFace* a další) a při spouštění vývojového prostředí jsou všechny části pomocí specifikace *Eclipse Equinox* poskládány do výsledného běhového prostředí *Eclipse IDE*.



Obr. 1.2: Okno aplikace vývojového prostředí (*Workbench window*).

Java Development Tools (JDT) je plug-in, který umožňuje používat *Java editor*, wizard (průvodce nastavením), možnost formátování zdrojového kódu, ale zejména dva důležité nástroje – **debugger** a **kompilátor**, které jsou podstatnou částí vývojového prostředí *Eclipse*.

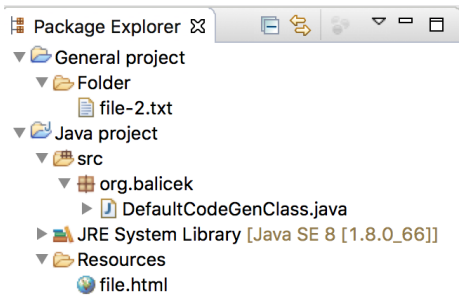
Plug-in Development Environment (PDE) je zásuvný modul, který poskytuje rozšířené nástroje do prostředí *Eclipse* pro tvorbu nových zásuvných modulů.

1.2 Pracovní prostor a zdroje

Důležitá součást vývojového prostředí *Eclipse*, zejména z pohledu uživatele, je jeho pracovní prostor (Workspace) a jeho zdroje (Resources) [3]. Již při zapnutí vývojového prostředí je nutné vybrat adresář na disku, který bude sloužit jako pracovní prostor a bude poskytovat zdroje pro nové projekty. Jedná se o hlavní uzel, kde jsou shromažďována všechna uživatelská data. Pro uživatele se zde nachází tři základní typy zdrojů – projekty (projects), adresáře (folders) a soubory (files).

- **Projekt** (Project) – je součástí pracovního prostoru, obsahuje adresáře (folders) nebo soubory (files), projekt lze popsat jako kontejner, který shromažďuje všechny dostupné zdroje.
- **Adresář** (Folder), **Soubor** (File) – jedná se o obyčejné prvky, jako se nachází v souborovém systému operačního systému, adresář může obsahovat jiné adresáře nebo soubory a soubor již obsahuje posloupnost bajtů (určitý druh dat).

Pracovní prostor nemusí existovat pouze jeden, lze jich definovat hned několik a lze mezi nimi jednoduše přepínat. V jednom okamžiku je *Eclipse IDE* schopno pracovat pouze v jednom pracovním prostoru. Každý pracovní prostor obsahuje specifická uživatelská nastavení vývojového prostředí a projekty, které byly do tohoto pracovního prostoru přidány.



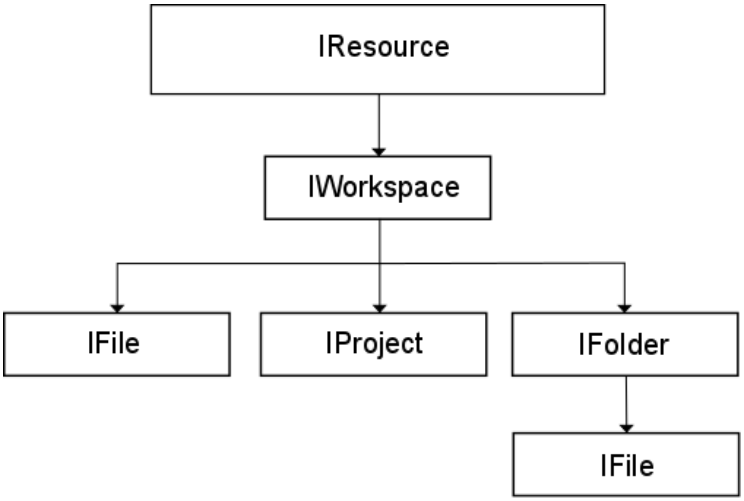
Obr. 1.3: Stromová struktura projektů ve vývojovém prostředí *Eclipse*.

Z pohledu struktury je pracovní prostor uspořádán do stromu (obr. 1.3), kde nejvyšší vrstvu tvoří projekt a další vrstvy tvoří adresáře a soubory. Existuje ještě tzv. kořenový adresář pracovního prostoru (*Workspace root*), který je vytvořen vždy při založení nového pracovního prostoru a je dostupný po celou dobu jeho existence.

- ***Workspace root*** – domovský adresář celé stromové struktury pracovního prostoru.

Vývojové prostředí *Eclipse* využívá pro komunikaci s pracovním prostorem a jeho dostupnými zdroji zásuvný modul `org.eclipse.core.resources`. Pro *Eclipse* není workspace pouze obyčejným adresářem souborového systému, ale je chápán jako logický kontejner, který obsahuje dostupné zdroje (projekty, složky, soubory a další).

Pro práci s logickým kontejnerem používá *Eclipse* aplikační rozhraní – ***API*** (obr. 1.4), které umožňuje jednodušší a efektivnější práci s prostředky pracovního prostoru. Rozhraní definuje pro každý prvek pracovního prostoru další rozhraní, což umožňuje lépe pracovat s každou částí z dostupných zdrojů.



Obr. 1.4: Logická struktura rozhraní pro práci se zdroji v *Eclipse*.

Od obyčejného systémového adresáře se workspace odlišuje zejména skrytým adresářem `.metadata`. Jedná se o adresář uvnitř pracovního prostoru, který je vytvořen automaticky a jsou zde uchovány důležité informace o struktuře pracovního prostoru a jeho vlastnostech.

Dalším důležitým prvkem pracovního prostoru je skrytý soubor `.project`, který je automaticky vytvořen zásuvným modulem `org.eclipse.core.resources` uvnitř každého nově vytvořeného projektu. Uvnitř souboru `.project` jsou uložena metadata projektu a slouží pouze pro čtení.

1.3 Verze *Eclipse*

Projekt *Eclipse* obsahuje mnoho verzí *Eclipse IDE* (např. známé verze *Luna*, *Mars*, *Neon*). Pro implementaci praktické části této bakalářské práce bude využita nejnovější vydaná verze vývojového prostředí *Eclipse* – ***Eclipse Neon*** (verze 4.6.3), která bude schopna zásuvný modul plně podporovat. Mezi další podporované verze vývojového prostředí *Eclipse* bude patřit varianta založená na *Eclipse Neon* od společnosti *Red Hat*, nesoucí název ***Devstudio*** (*Red Hat JBoss Developer Studio 10.4.0*).

Každá z verzí *Eclipse IDE* je k dispozici ke stažení v několika distribucích, přičemž každá z distribucí má předinstalované plug-iny na práci, pro kterou je vydána (např. *Eclipse IDE for C/C++ Developers* obsahuje vše nezbytné pro programování v jazyce *C/C++*).

Mezi základní distribuce pro práci v jazyce *Java* patří ***Eclipse IDE for Java Developers*** a speciálně pro vývoj zásuvných modulů v jazyce *Java* je připravena distribuce ***Eclipse IDE for Eclipse Committers*** (obsahuje potřebné *Eclipse PDE*).

2 GRAFICKÉ KNIHOVNY

Vývoj grafického uživatelského rozhraní v programovacím jazyce *Java* podporuje mnoho grafických knihoven. Mezi nejznámější knihovny patří *AWT*, *Swing*, *SWT* a s ním spojená knihovna *JFace*. Práce se věnuje zejména knihovnám *SWT* a *JFace* z důvodu jejich použití při implementaci zásuvných modulů pro *Eclipse IDE*.
Knihovny *SWT* a *JFace* byly pro implementaci vybrány z následujících důvodů:

1. vývojové prostředí *Eclipse* je tvořeno za pomoci těchto knihoven,
2. vlastnosti, které tyto knihovny poskytují pro tvorbu zásuvných modulů odpovídají zadání práce,
3. plná kompatibilita a funkčnost vytvořeného zásuvného modulu.

2.1 Knihovny *AWT* a *Swing*

Grafická knihovna *Abstract Windowing Toolkit (AWT)* [11] je částí *JFC (Java Foundation Classes)* [12]. Tato knihovna je dobrým základem pro tvorbu grafického uživatelského rozhraní, ale není vhodná pro moderní a složitější aplikace, protože ji schází mnoho z moderních (dnes již obvyklých) ovládacích prvků. Pro menší aplikace, které nepotřebují pestré uživatelské rozhraní, je *AWT* dostačující.

Grafická knihovna *Swing* [13], byla vytvořena jako nadstavba knihovny *AWT* a rovněž slouží k práci s grafickým rozhraním na platformě *Java*. Pokud není grafické rozhraní vytvořené prostřednictvím knihovny *Swing* napsáno správně, může docházet k pomalému vykreslování. Výhoda této knihovny je vyšší přenositelnost programů.

2.2 Knihovna *SWT*

Grafická knihovna *Standard Widget Toolkit (SWT)* [14] tvoří základní uživatelské rozhraní, které umožňuje nativní vzhled aplikací na různých platformách. Díky těmto vlastnostem se stává aplikace multiplatformní – aplikaci je možno spustit na různých operačních systémech (*Windows*, *Linux*, *Mac OS*) a cílem je, aby byl vzhled vždy přizpůsoben danému prostředí operačního systému.

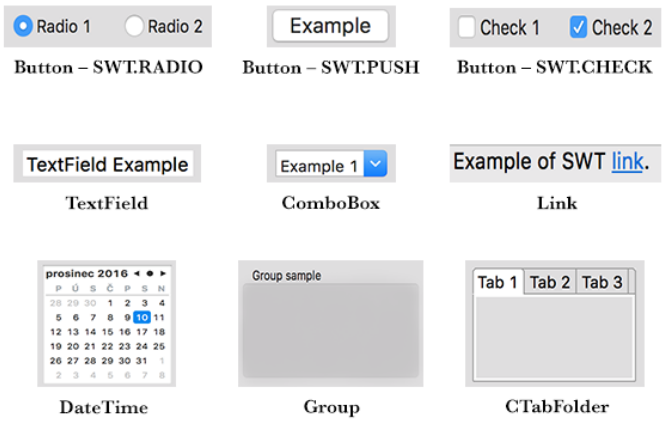
Mezi základní prvky (**widgety, widgets**) knihovny *SWT* patří tlačítka (buttons), textová pole (text fields) a další (obr. 2.1). Nedílnou součástí grafického rámce *SWT* je tzv. **manažer rozvržení** (*Layout Manager*) viz tab. 2.1, který umožňuje uspořádat jednotlivé prvky *SWT* dle zvolených pravidel v zobrazované oblasti aplikace. ***Widget*** je část grafického rozhraní aplikace (zásuvného modulu), která umožňuje uživateli interakci s aplikací nebo operačním systémem jako takovým. Nejjedno-

dušším příkladem definice widgetu je tlačítko, jehož stisknutím má uživatel možnost vyvolat určitou akci v daném prostředí.

Manažer rozvržení	**Popis**
AbsoluteLayout	umožňuje přesně specifikovat pozici programátorem
FillLayout	uspořádá widgety o stejné velikosti do jednoho řádku
RowLayout	vloží widget do řádku či sloupce a umožňuje dále specifikovat rozvržení pomocí „layout“ parametrů
GridLayout	uspořádá widgety do mřížky
FormLayout	určí polohu pro umístění vzhledem k rodičovskému kontejneru

Tab. 2.1: Manažeři rozvržení grafické knihovny *SWT*.

Při tvorbě zásuvného modulu, který bude rozšiřovat samotné *Eclipse IDE*, je doporučeno využívat pouze prvky ze standardní knihovny *SWT*, kterou rovněž pro vykreslování grafiky používá *Eclipse Workbench*. Tím pádem by nemělo docházet k žádným kolizím a chybnému zobrazování.

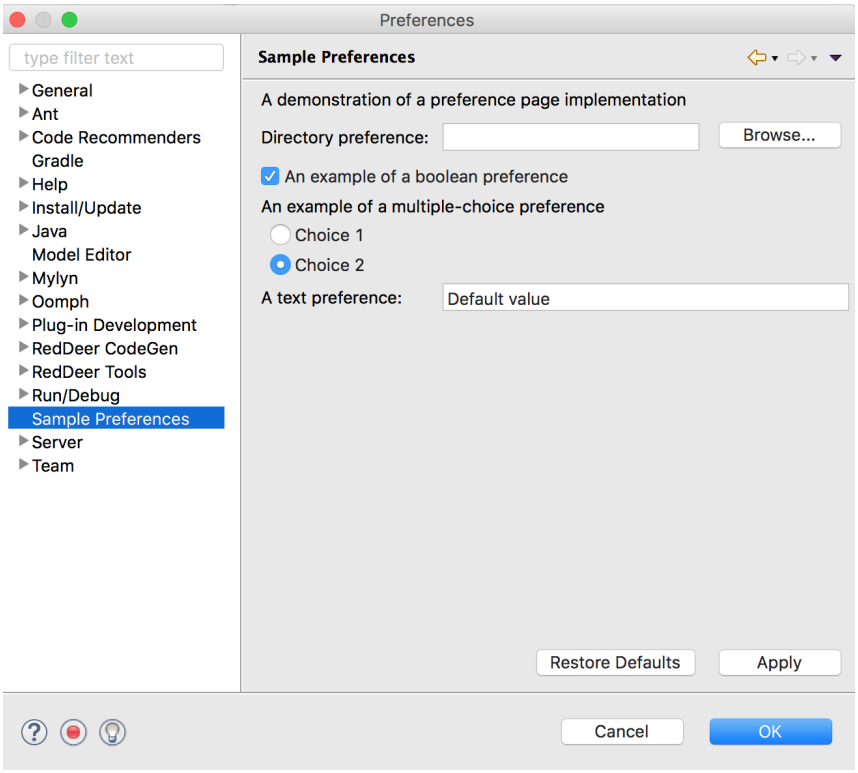


Obr. 2.1: Widgety z knihovny *SWT*.

Důležitými částmi knihovny *SWT* jsou ***shell*** a ***display***. *Shell* reprezentuje okno aplikace a *display* zajišťuje ovládání všech operací, jako jsou např. uživatelské interakce (klikání na tlačítka apod.) a zajišťuje komunikaci mezi aplikačním vláknem uživatelského rozhraní a všemi ostatními vlákny aplikace. Každá aplikace nebo zásuvné moduly, které jsou vytvořeny za pomoci knihovny *SWT* musí obsahovat nejméně jednu komponentu *display* a jednu nebo více komponent *shell*.

2.3 Knihovna *JFace*

Eclipse *JFace* [15] je sada zásuvných modulů, která rozšiřuje knihovnu *SWT*, aniž by ji překryla. Pro práci s komponentami *JFace* je tedy nezbytné využít knihovnu *SWT*. Mezi komponenty knihovny *JFace* patří vyskakovací okna (*dialogs*), náhledy (*views*), průvodci nastavením (*wizards*) nebo stránky možností nastavení (*preference pages*) znázorněné na obrázku 2.2. Tyto možnosti umožňují programátorovi změnit nastavení prezentovaných dat a zejména usnadňují uživateli práci s danou aplikací nebo zásuvným modulem.



Obr. 2.2: Stránka možností nastavení (*preference page*).

3 ROZŠIŘOVÁNÍ VÝVOJOVÉHO PROSTŘEDÍ ECLIPSE

Účelem této kapitoly je popsat způsoby rozšiřování vývojového prostředí *Eclipse* o nové zásuvné moduly [16] a popsat možnosti instalace nově vytvořených zásuvných modulů do vývojového prostředí *Eclipse*.

3.1 Vývoj zásuvných modulů

Eclipse Platform je navrženo jako prostředí, které samo o sobě obsahuje pouze několik základních funkcí. Rozšíření funkcí vývojového prostředí *Eclipse* je zajištěno pomocí zásuvných modulů (plug-inů) [17]. *Eclipse* je vývojové prostředí, které nabízí velké množství plug-inů, a proto patří mezi často využívaná a oblíbená vývojová prostředí. Nejvíce využívané je pro práci s programovacím jazykem *Java*, ale mezi oblíbené patří i další programovací jazyky jako je *C++*, webové *HTML* a mnoho dalších.

Eclipse IDE nabízí pro vytváření vlastních zásuvných modulů jiný zásuvný modul s názvem *PDE (Plug-in Development Environment)*. *PDE* poskytuje pro tvorbu nového plug-inu nabídku několika předem definovaných šablon, které již mají implementovány základní grafické ovládací prvky z knihoven *SWT* a *JFace*. Šablon je hned několik typů a ve vývojovém prostředí *Eclipse* jsou dostupné v záložce `File > New > Project > Plug-in Development > Plug-in Project`. Mezi oblíbené patří např. šablona plug-inu pro tvorbu textového editoru, plug-in s náhledem (*view*) a ukázkový „*Hello, world plug-in*“.

Při vývoji nového zásuvného modulu existuje několik souborů, které definují jeho vlastnosti. Hlavním je *XML* soubor **`plugin.xml`** (výpis 3.1), který se nachází v kořenovém adresáři zásuvného modulu. Uvnitř je definováno, jak zásuvný modul rozšiřuje vývojové prostředí, jaké jsou jeho vlastnosti a jakým způsobem je implementována jeho funkcionalita.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <?eclipse version="3.4"?>
3  <plugin id="org.jboss.reddeer.codegen"
4          name="CodeGen"
5          class="org.jboss.reddeer.codegen.CodeGen"
6          version="0.0.1-qualifier">
7
8    <extension point="org.eclipse.ui.views">
9      <category id="sample"
10                name="Sample␣Category">
11     </category>
12
13     <view category="sample"
14           class="sample.views.SampleView"
15           icon="icons/sample.gif"
16           id="sample.views.SampleView"
17           name="Sample␣View">
18     </view>
19   </extension>
20
21   <requires>
22     <import plugin="org.eclipse.core.runtime"/>
23     <import plugin="org.eclipse.core.resources"/>
24     ...
25   </requires>
26 </plugin>
```

Výpis 3.1: Struktura souboru `plugin.xml`.

Soubor má strukturu *XML* a je tedy rozdělen do sekcí pomocí *XML* značek. Hlavní značka `<plugin>`, obsahuje důležité atributy popisující vlastnosti zásuvného modulu. Některé zásuvné moduly potřebují ke své funkčnosti jiné zásuvné moduly, tyto reference na vyžadované plug-iny jsou uvedeny v sekci značky `<requires>`, kde je vždy minimálně importován zásuvný modul **`org.eclipse.core.runtime`** nezbytný pro spuštění plug-inu. Atributy značky `<plugin>` a reference na vyžadované zásuvné moduly mohou být umístěny zvlášť do souboru **`MANIFEST.MF`** (výpis 3.2), který se nachází v adresáři `META-INF` v kořenovém adresáři zásuvného modulu.

```
1  Manifest-Version: 1.0
2  Bundle-ManifestVersion: 2
3  Bundle-Name: Sample
4  Bundle-SymbolicName: org.sample
5  Bundle-Version: 0.0.1-qualifier
6  Bundle-Activator: org.sample.Activator
7  Require-Bundle: org.eclipse.ui,
8    org.eclipse.core.runtime,
9    ...
```

Výpis 3.2: `MANIFEST.MF`

V průběhu vývoje nového zásuvného modulu je zapotřebí velmi často testovat jeho funkčnost, proto je nutné pokaždé plug-in nainstalovat do vývojového prostředí a poté vyzkoušet, jestli všechny implementované vlastnosti fungují správně. Postup opětovné instalace při každé kontrole funkčnosti plug-inu je velmi neefektivní, proto *Eclipse* umožňuje za běhu spustit nové okno vývojového prostředí (nový *workbench*) nad právě běžící instancí *Eclipse IDE*. Nově vytvořená instance vývojového prostředí obsahuje vyvíjený plug-in a je možné ihned otestovat jeho funkce.

Ke spuštění nového běhového prostředí slouží položka **`Run > Run As > Eclipse Application`** nebo **`Run > Run Configurations...`**, kde lze více specifikovat běhové prostředí, které bude pro testování vytvořeno.

3.1.1 Funkce

Eclipse IDE je strukturované jako množina zásuvných modulů, kde každý zásuvný modul rozšiřuje vývojové prostředí o nové funkce. Zásuvné moduly jsou nainstalovány do lokálního úložiště počítače a jsou aktivovány podle požadavků uživatele na práci.

Zásuvné moduly mohou být seskupeny do „**funkcí**“ (*features*). Funkce mohou obsahovat více plug-inů, které mají většinou určitý význam pro práci, na kterou je „**vlastnost**“ (*feature*) určena. *Feature* je část z celkové skupiny vlastností ve funkci, která může být samostatně stažena a nainstalována do vývojového prostředí jako nová funkcionalita.

Výhodou funkcí je zejména možnost nainstalovat do vývojového prostředí v jednom kroku více zásuvných modulů namísto instalování několika plug-inů samostatně a následného zjišťování, zda-li jsou již nainstalovány všechny potřebné zásuvné moduly pro práci na novém projektu. Přehled nainstalovaných vlastností a plug-inů ve vývojovém prostředí lze zobrazit v **`Help > Installation Details > Features`** nebo **`Help > Installation Details > Plug-ins`**.

Pro vytvoření a implementování nového zásuvného modulu jako vlastnosti (*feature*) existuje ve vývojovém prostředí *Eclipse* tzv. ***Feature Project*** (`File > New > Other... > Plug-in Development > Feature Project`).

3.2 Instalace nových zásuvných modulů

Pro testování zásuvného modulu při jeho vývoji je možné spustit nové běhové prostředí, ve kterém je nový zásuvný modul již nainstalován. Instalace zásuvného modulu do prostředí *Eclipse* ale znamená, že je nainstalovaný zásuvný modul ve vývojovém prostředí k dispozici ihned po spuštění. Existují tři způsoby instalace:

- exportovat zásuvný modul přímo do aktuálně spuštěného *Eclipse IDE*,
- exportovat zásuvný modul jako *JAR* soubor (.jar) a nakopírovat jej manuálně do adresáře **`plugins`**, který se nachází v adresářové struktuře instalace *Eclipse*,
- vytvořit **instalační odkaz** (*Update site*) a prostřednictvím ***instalačního Eclipse manažeru*** (*Eclipse update manager*) plug-in nainstalovat.

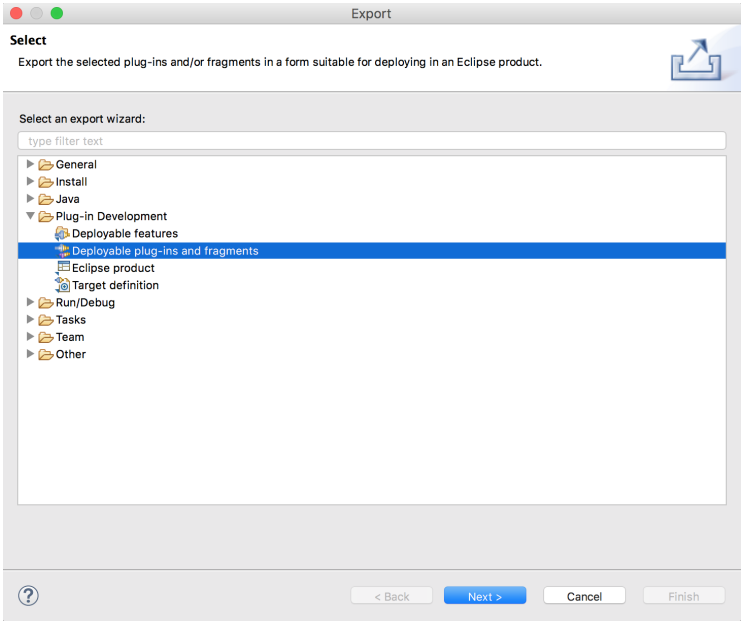
3.2.1 Instalace prostřednictvím vývojového prostředí *Eclipse*

První možnost instalace nově vytvořeného zásuvného modulu je přímo prostřednictvím *Eclipse IDE*. Instalace spočívá v exportování zásuvného modulu do aktuálně spuštěného vývojového prostředí. V *Eclipse IDE* instalaci usnadňuje dialog pro exportování (obr. 3.1). V menu **`File > Export`**. Po otevření dialogu průvodce dále vybrat položku **`Plug-in Development > Deployable plug-ins and fragments`**.

Poté je zapotřebí v seznamu zásuvných modulů vybrat plug-in, který bude exportován a na záložce **`Destination`** vybrat položku `Install into host. Repository`. (znázorněno na obr. 3.2). Stiskem tlačítka **`Finish`** dojde k exportování (instalaci) zásuvného modulu do vývojového prostředí *Eclipse*. Jakmile je export dokončen, vývojové prostředí si vyžádá restart pro zavedení změn. Po opětovném spuštění je nový zásuvný modul již plně k dispozici.

3.2.2 Instalace prostřednictvím exportovaného souboru

Druhou možností instalace nově vytvořeného zásuvného modulu do vývojového prostředí *Eclipse* je exportovat zásuvný modul jako *JAR* soubor a manuálně jej přidat do instalačního adresáře *Eclipse*. Pro exportování zásuvného modulu do *JAR* souboru lze využít průvodce pro exportování (obr. 3.2). V menu volba **`File > Export`**. Po otevření dialogu průvodce dále vybrat položku **`Plug-in Development > Deployable plug-ins and fragments`**. Poté je nutné v seznamu zásuvných modulů vybrat



Obr. 3.1: Dialogové okno exportu.

plug-in, který bude exportován a zvolit umístění (záložka `Destination`, položka `Directory`), kam bude *JAR* soubor zásuvného modulu vygenerován.

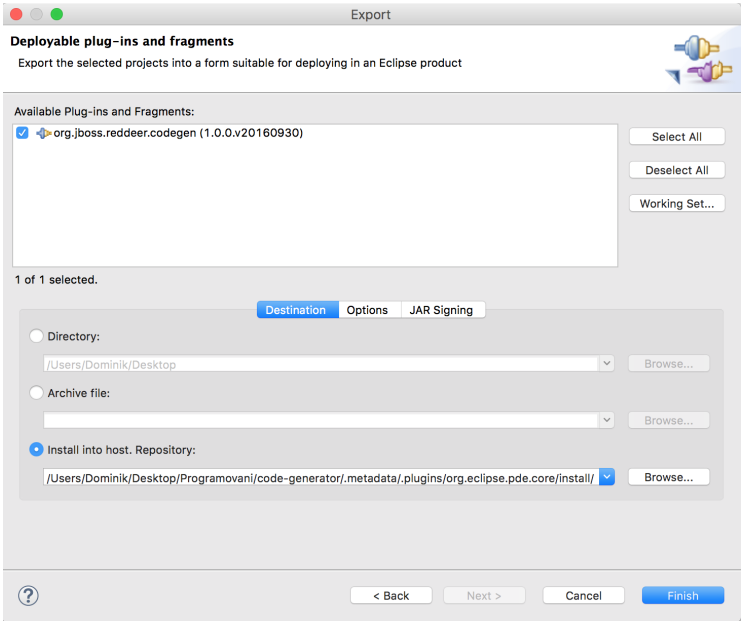
Jakmile je generování u konce, stačí *JAR* soubor zkopírovat do adresáře **`plugins`**, který se nachází v adresářové struktuře instalace *Eclipse* – `<path>/Eclipse/plugins` (např. `C:/Program Files/Eclipse/plugins` pro Windows). Cesta k umístění instalace *Eclipse* (`<path>`) se liší v závislosti na operačním systému. Poslední nutný krok pro dokončení instalace zásuvného modulu je restartování *Eclipse IDE*.

Instalace pomocí manuálního kopírování *JAR* souboru se nedoporučuje, protože si každý uživatel musí sám ohlídat všechny závislosti (reference) zásuvného modulu na jiných modulech nebo knihovnách vývojového prostředí a to je při větším počtu závislostí velmi problematické.

3.2.3 Instalace prostřednictvím instalačního *Eclipse* manažeru

Poslední variantou, jak nainstalovat nově vytvořený zásuvný modul do *Eclipse IDE*, je prostřednictvím tzv. **instalačního odkazu** (*update site*). Instalační odkaz je tvořen soubory, které jsou umístěny na vzdáleném nebo lokálním úložišti (serveru).

Umístění souborů na vzdáleném úložišti poskytuje velkou výhodu, jak při distribuci nového zásuvného modulu, tak při vydávání nových aktualizací. Stačí uživatelům poskytnout odkaz na vytvořené úložiště, kde je zásuvný modul zpřístupněn



Obr. 3.2: Export zásuvného modulu pomocí *Eclipse IDE*.

a uživatelé již jednoduše nový zásuvný modul nainstalují do svého vývojového prostředí pomocí instalačního *Eclipse* manažeru.

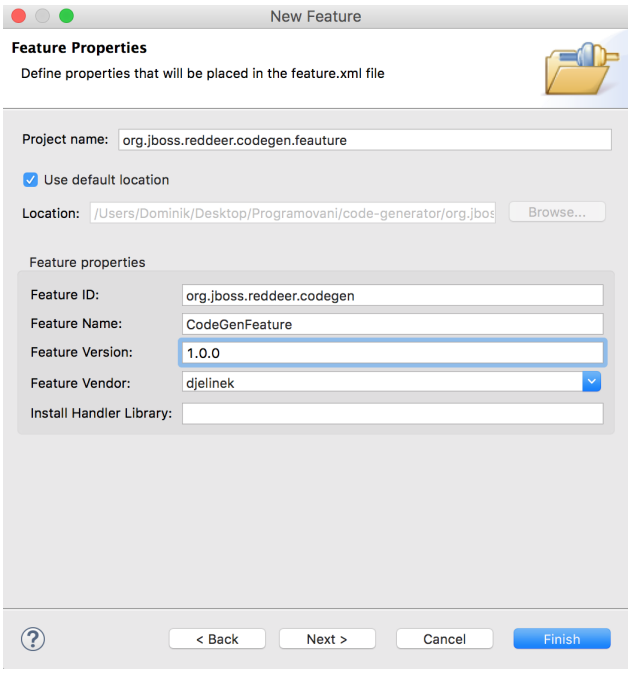
K vytvoření instalačního odkazu jsou zapotřebí tři kroky. V prvním kroku je nutné vytvořit *Feature Project*.
V menu **`File > New > Other... > Plug-in Development > Feature Project`**. V otevřeném dialogu viz obr. 3.3 je zapotřebí vyplnit všechny nezbytné informace a tlačítkem **`Next`** se posunout dále. V dalším okně stačí vybrat z nabídky dostupných zásuvných modulů ten, pro který bude instalační odkaz vytvořen a tlačítkem **`Finish`** vytvoření dokončit.

Druhý krok je vytvoření *Category Definition* do *Feature projektu*, který byl vytvořen o krok dříve. Toho lze dosáhnout pomocí **`File > New > Other... > Plug-in Development > Feature Definition`**. Tento krok je důležitý, protože bez vytvoření *Definition Category* by nebyl *Eclipse update manager* schopný zobrazit zásuvný modul, který se na *update site* nachází.

Třetí krok je vytvoření již samotné *update site*, v menu zvolením **`File > Export > Plug-in Development > Deployable Features`**. V zobrazeném dialogu je zapotřebí vybrat *Feature projekt*, který byl vytvořen v prvním kroku a na záložce nastavení **`Destination > Directory`** zvolit umístění, odkud bude zásuvný modul dostupný pro pozdější instalování.

Pro přiřazení zásuvného modulu do *Definition Category*, která byla vytvořena v druhém kroku instalace slouží záložka **`Options > Categorize repository`**.

Posledním krokem je nainstalování nového zásuvného modulu do *Eclipse IDE*



Obr. 3.3: Dialog pro vytvoření *Feature projektu*.

prostřednictvím nově vytvořené *update site*. K tomu slouží již zmíněný *Eclipse update manager* (**`Help > Install New Software...`**). V zobrazeném dialogovém okně, lze učinit stiskem tlačítka **`Add...`**, kde přidáním cesty k umístění lokálního nebo vzdáleného úložiště a následným vybráním zásuvného modulu z nabídky *Eclipse update manager* bude nový zásuvný modul nainstalován. Pro úspěšné dokončení instalace je nutné vývojové prostředí restartovat.

4 TESTOVÁNÍ APLIKACÍ

Při každém vývoji nové aplikace je důležitou součástí vývoje průběžné testování aplikace [18]. Testovat funkčnost aplikace lze dvěma způsoby:

1. **manuálně** (osoba nebo-li „tester“ manuálně testuje aplikaci),
2. **automaticky** (tzv. automatizované testování).

Testování aplikace spočívá ve vytvoření zdrojového kódu (testu), který automaticky ověřuje správnost určité dílčí části celku, např. test pro součet dvou čísel v aplikaci kalkulačka. Vytváření testů se dělí na několik úrovní podle úhlu pohledu z hlediska „životního cyklu“ vyvíjené aplikace:

1. **jednotkové** – testuje se část programu (např. jedna *Java* třída projektu),
2. **integrační** – testuje se integrování nové funkcionality do projektu,
3. **akceptační** – ověřují, zda jsou splněny podmínky pro používání programu uživatelem.

Mezi nejznámější a nejvíce používané testovací rámce, zejména pro programovací jazyk *Java*, patří testovací rámec ***JUnit*** [19], který umožňuje v objektově orientovaném programování vytvářet **Jednotkové testy** (*Unit testy*) [20]. V dnešní době je testovací rámec *JUnit* integrován do většiny vývojových prostředí.

Jednotkový test slouží pro otestování specifické části zdrojového kódu programu. Pro každou funkci programu by měl existovat jednotkový test, který ověří, zda se funkce chová dle požadavků a očekávání. Jednotkové testy fungují převážně na principu tzv. **black-box** (test je psán bez znalosti implementované logiky uvnitř testované funkce, pouze prostřednictvím vstupních dat a očekávaného výstupu funkce).

4.1 Automatizované testování

Pro efektivní testování při vývoji větších aplikací se využívá automatizované testovaní, které umožňuje opakované spouštění testů nebo testovacích scénářů.

Hlavní výhoda vytváření automatizovaných testů spočívá zejména v možnosti spustit test několikrát za sebou, tzn. pokud dojde ke změně funkcionality aplikace, pro kterou je test přizpůsoben, test změnu detekuje a informuje o změně chování testované aplikace.

Mezi další výhody patří například možnost spouštění velkého množství testů najednou, generování velkých vstupních dat nebo zátěžové testování.

4.2 Testovací rámec *RedDeer*

Testovací rámec *RedDeer* [1] byl vytvořen jako alternativa testovacího rámce *SWTBot* [21]. Je dostupný jako zásuvný modul do vývojového prostředí *Eclipse*. Slouží pro testování uživatelského rozhraní aplikací založených na *Eclipse*, *SWT* a *GEF* [22]. *RedDeer* obsahuje aplikační rozhraní, které usnadňuje práci s *SWT* a uživatelským rozhraním *Eclipse (Eclipse UI)*, a proto je vhodný pro jednoduché použití při vytváření testů pracujících s uživatelským rozhraním.

Rámec *RedDeer* obsahuje několik nástrojů pro lepší práci s uživatelským rozhraním při vytváření testů, například nástroje:

- ***RedDeer Project Wizard*** – speciální průvodce (*wizard*) pro tvorbu *RedDeer* projektu v *Eclipse IDE*,
- ***RedDeer Spy View*** – po spuštění zobrazuje informace o widgetu, na který uživatel zamíří kurzorem myši,
- ***RedDeer Log Parser*** – slouží pro formátovaný zisk informací ze souboru pro uchování záznamů o běhu aplikace (`.log`).

Dále využívá a rozšiřuje vlastnosti rámce *JUnit* o nové funkcionality [23], například o nové testovací požadavky, které slouží pro přípravu testovacího prostředí. Mezi základní rozšiřující testovací požadavky patří např.:

- ***Clean workspace*** – tato anotace zařídí čistý workspace před spuštěním testů,
- ***Open perspective*** – testy jsou spuštěny s předem definovanou perspektivou vývojového prostředí *Eclipse*,
- ***JRE*** – definuje verzi *JRE (Java Runtime Environment)*, se kterou budou testy spuštěny,
- ***Server*** – specifikuje server, který je nutný pro běh testů,
- ***Database*** – definuje název vyžadované databáze, která bude použita pro všechny testy.

5 SESTAVENÍ A KOMPILACE APLIKACE

Apache Maven [2] je nástroj pro správu, řízení a kompilování (buildování, sestavování) aplikací. Výsledkem kompilace je aktuální verze programu tzv. **build**, kterou je možné sestavit bez chyb, např. pro testování nebo distribuci aplikace. *Maven* je navržený zejména pro podporu programovacího jazyka *Java*, ale lze jej využít i pro projekty psané v jiném programovacím jazyce založeném na *JVM* [24].

Adresář	**Popis**
Kořenový adresář projektu	obsahuje všechny adresáře projektu a `pom.xml`
`src/main/java`	obsahuje zdrojové soubory (`.java`)
`src/main/resources`	obsahuje zbylé soubory aplikace
`src/test/java`	obsahuje soubory pro spouštění testů
`src/test/resources`	obsahuje soubory využívané v testech

Tab. 5.1: Základní adresářová *Maven* struktura pro *Java* projekt.

Projekt *Apache Maven* je založen na principu modulární architektury (*Plug-in-Based Architecture*) jako *Eclipse IDE* a funguje tedy na principu volání jednotlivých plug-inů. *Maven* obstarává pouze poskytování a spouštění předem definovaných plug-inů.

Pro základní instalaci *Maven* není vytvořeno grafické rozhraní a ovládá se prostřednictvím příkazové řádky, je tedy nutné znát ovládací příkazy. Pro usnadnění a urychlení práce lze integrovat *Maven* do vývojového prostředí (*Eclipse IDE*, *NetBeans* a další), které poskytuje uživatelsky více přívětivé ovládání všech základních funkčností.

Důležitý pojem v *Maven* architektuře je tzv. **Artefakt**. Jedná se o výstupní soubor, který vznikne po dokončení sestavení aplikace (*.jar*, *.war*, *.ear*, ...). Každý *Maven* projekt má vždy pouze jeden artefakt, který je vytvořen na základě definovaných závislostí v kořenovém souboru `pom.xml`.

Artefakty jsou shromažďovány v úložištích (***repositories***). Každý nový *Maven* projekt, který bude následně sestavován lze považovat za jedno úložiště (*repository*). Existují dva typy úložišť:

- **lokální** (*local*) – odkazuje se na kopie souborů instalace v místním úložišti,
- **vzdálené** (*remote*) – převážně se jedná o servery (lokální nebo *HTTP*), k úložišti se poté přistupuje pomocí protokolu `file://` nebo `http://`.

Maven využívá pro popis a definici projektu koncept popisu jako objektu – ***Project Object Model***. Koncept popisu projektu jako objektu je realizován za pomoci

jednoduché *XML* struktury (výpis 5.1), která definuje jednotlivé části projektu, závislosti na ostatních projektech, externích knihovnách a nástrojích. Soubor, který definuje strukturu objektu je nazván **pom.xml** a je umístěn v kořenovém adresáři projektu viz tab. 5.1.

```
1  <project xmlns="http://maven.apache.org/POM/4.0.0" ...>
2      <modelVersion>4.0.0</modelVersion>
3      <groupId>org.sample</groupId>
4      <artifactId>parent</artifactId>
5      <version>0.0.1-SNAPSHOT</version>
6      <packaging>pom</packaging>
7      <name>Sample Parent POM file</name>
8      <dependencies>
9        <dependency>
10         <groupId>junit</groupId>
11         <artifactId>junit</artifactId>
12         <version>4.4</version>
13       </dependency>
14     </dependencies>
15 </project>
```

Výpis 5.1: Struktura souboru `pom.xml`.

Důležitá část struktury `pom.xml` je oblast závislostí (`<dependencies>`). U každého projektu mohou být definovány jiné závislosti. Jedná se o závislosti na externích knihovnách nebo plug-inech. Jednotlivé `<dependency>`, jsou jednoznačně identifikovány pomocí atributů `<groupId>` a `<artifactId>`. Při spuštění buildu projektu *Maven* automaticky vyhledá a nainstaluje všechny předem definované závislosti. Vyhledávání probíhá v předem definovaných úložištích (*repositories*). Úložiště se definují v souboru `pom.xml` a mezi známé patří např. *Maven Repository* nebo *Maven 2 Central Repository*. Lze založit i vlastní lokální nebo serverová úložiště, které lze taktéž využít jako závislost v `pom.xml`.

Pokud je projekt rozsáhlý a je tvořen více podprojekty nebo je rozdělen na několik plug-inů, pak každý z podprojektů (plug-inů) obsahuje vlastní `pom.xml` soubor, který dědí všechny vlastnosti a nastavení `pom.xml` souboru z kořenového adresáře. Všechny další soubory `pom.xml` mohou přidávat nové vlastnosti výsledného buildu projektu. Dědění všech `pom.xml` souborů projektu pouze z jednoho souboru v kořenovém adresáři umožňuje sestavit kompletní build provedením pouze jednoho příkazu, což je velká výhoda celého konceptu *Project Object Model*.

Koncept sestavování aplikací je definován jako „životní cyklus buildu“ (**Build Lifecycle** [25]). Pojem *Build Lifecycle* vznikl, protože každý build prochází několika fázemi, kde každá fáze vytvoří mezivýsledek z celého buildu – tzn. build nemusí projít celým životním cyklem pro dosažení požadovaného výsledku. Tři základní životní cykly buildu jsou:

- ***default (or build)*** – výsledkem je odeslání na server,
- ***clean*** – výsledkem je vyčištění (odstranění) předchozího buildu projektu,
- ***site*** – výsledkem je vytvoření projektové dokumentace.

Každý z životních cyklů buildu prochází odlišnými fázemi. Zjednodušená ukázka fází životního cyklu *default (or build)*:

- ***validate*** – kontrola správnosti projektu a dostupnosti všech nezbytných informací,
- ***compile*** – kompilace zdrojových souborů,
- ***test*** – ověření zdrojového kódu jednotkovými testy,
- ***package*** – vytvoření balíčku určeného pro distribuci, např. ve formátu *JAR*,
- ***verify*** – kontrola balíčku pomocí integračních testů,
- ***install*** – instalace balíčku do lokálního úložiště (repository),
- ***deploy*** – odeslání balíčku do vzdáleného úložiště na server (sdílené úložiště).

5.1 Zásuvný modul *Tycho*

Tycho [26] je množina *Maven* plug-inů, která slouží k sestavování *Eclipse* komponent, jako jsou zásuvné moduly, *OSGi bundles*, *Eclipse Features*, *Update sites* a další. Buildování *Eclipse* komponent zajišťuje `tycho-maven-plugin`. *Tycho* využívá co nejvíce informací o komponentě z dostupných metadat, např. pro získání závislostí zásuvného modulu využívá informace v souboru `MANIFEST.MF`.

Po přidání pluginu `tycho-maven-plugin` do `pom.xml` v nastavení *Maven* buildu je *Tycho* při buildu staženo a lze kompilovat *Eclipse* komponenty. Základní balíčky *Tycho* plug-inu pro *Eclipse* komponenty:

- `eclipse-plugin`,
- `eclipse-feature`,
- `eclipse-repository`.

6 VÝVOJ GENERÁTORU KÓDU PRO TESTOVACÍ RÁMEC REDDEER

Hlavním cílem praktické části bakalářské práce je vytvoření zásuvného modulu do vývojového prostředí *Eclipse* a návrh interní logiky pro generování kódu, který má za účel zjednodušení práce při psaní automatizovaných testů využívajících grafické rozhraní *Eclipse*. Po analýze možností generování kódu byl vytvořen zásuvný modul generátoru kódu, který byl pojmenován „***RedDeer CodeGen***“.

V této kapitole je vysvětlen postup vývoje generátoru kódu, detailně popsáno grafické rozhraní vytvořeného zásuvného modulu, následuje vysvětlení interní logiky generování kódu a popis experimentálního ověření funkčnosti vygenerovaného zdrojového kódu.

6.1 Příprava pro vývoj zásuvného modulu

Pro vytvoření zásuvného modulu do vývojového prostředí *Eclipse*, který bude možné sestavit (kompilovat, buildovat) nástrojem *Apache Maven* je nejprve zapotřebí stáhnout správnou verzi *Eclipse IDE* a doinstalovat všechny nezbytné zásuvné moduly.

RedDeer CodeGen je vytvořen prostřednictvím vývojového prostředí ***Eclipse Neon – Eclipse IDE for Java Committers***[1]. V nově nainstalovaném *Eclipse IDE* je nyní k dispozici zásuvný modul sloužící pro vývoj nových zásuvných modulů (*PDE*). Dále je zapotřebí nainstalovat do vývojového prostředí ***Maven plug-in***, který umožňuje vytvořit a kompilovat *Maven* projekty. *Maven* je dostupný pro instalaci z *Maven Update site*[2], proto lze pro instalaci do vývojového prostředí využít instalační *Eclipse* manažer.

Pro funkčnost kompilace *Maven* projektů, které obsahují zásuvný modul je dále zapotřebí mít v *Eclipse IDE* nainstalovaný ***Tycho plug-in***. *Tycho* je při první kompilaci projektu automaticky stažen a nainstalován prostřednictvím *Maven* pluginu.

Souhrn nezbytných kroků pro vytvoření *Maven* projektu, který obsahuje zásuvný modul s jednoduchým uživatelským rozhraním:

1. stáhnout správnou verzi *Eclipse IDE (Eclipse Neon – Eclipse IDE for Java Committers)*,
2. nainstalovat zásuvný modul *Apache Maven*,
3. při prvním pokusu o kompilaci (sestavení buildu) projektu potvrdit doinstalování zásuvného modulu *Tycho*.

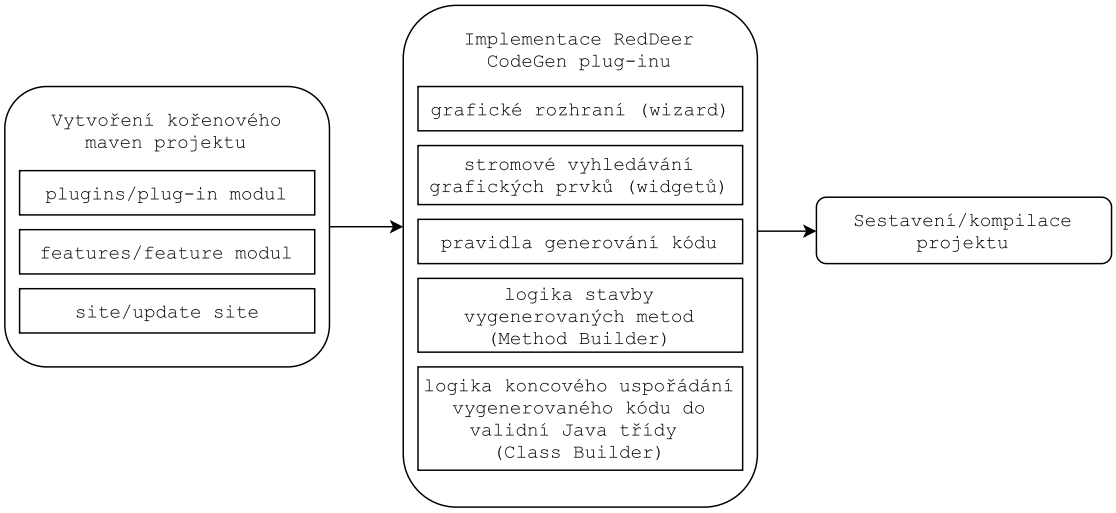
[1] `http://www.eclipse.org/downloads/packages/`
[2] `http://download.eclipse.org/technology/m2e/releases/`

6.2 Postup implementace generátoru kódu

Postup vytvoření generátoru kódu pro testovací rámec *RedDeer* lze rozdělit do tří částí diagramu, který je znázorněn na obrázku 6.1:

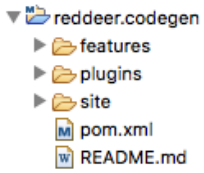
- založení nového *Maven* projektu,
- vytvoření zásuvného modulu,
- sestavení/kompilace projektu.

Prvním krokem vytvoření generátoru kódu pro testovací rámec *RedDeer* je založení nového projektu, ve kterém je zapotřebí manuálně vytvořit strukturu, která odpovídá struktuře *Maven* projektu viz obr. 6.2.



Obr. 6.1: Diagram vývoje zásuvného modulu.

Připravený projekt slouží jako základ pro implementaci dalších částí generátoru kódu, znázorněných v digramu na obrázku 6.1. Následující krok vývoje již popisuje implementaci samotného zásuvného modulu generátoru kódu a jeho vlastností. Vlastnosti jednotlivých části tohoto kroku jsou více popsány v kapitolách „*6.3 Grafické uživatelské rozhraní*“ a „*6.4 Interní logika generování kódu*“.



Obr. 6.2: Struktura *RedDeer CodeGen* projektu.

Posledním krokem vývoje generátoru kódu je výsledné sestavení (kompilace) projektu, které zajistí vytvoření instalačního odkazu, ze kterého lze následně vytvořený zásuvný modul nainstalovat do vývojového prostředí *Eclipse*.

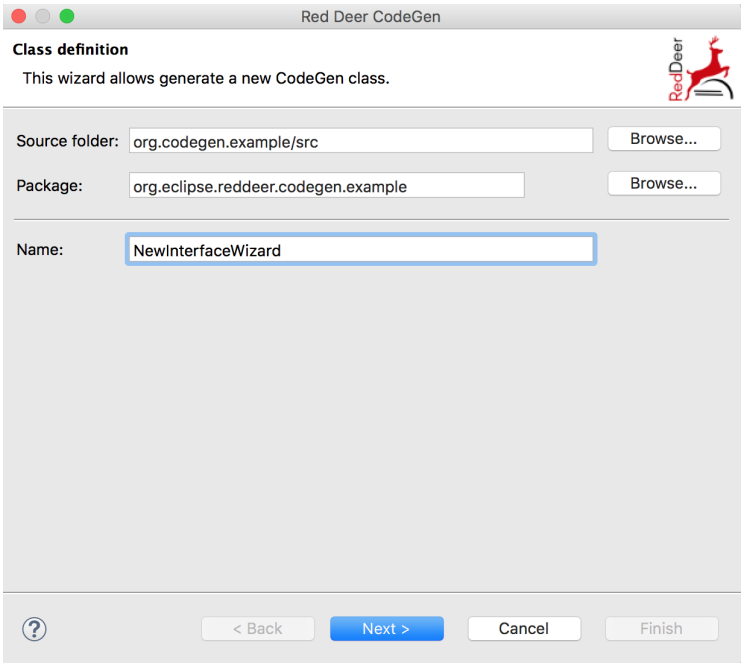
Podrobný postup vytváření jednotlivých částí projektu je uveden v příloze bakalářské práce.

6.3 Grafické uživatelské rozhraní

Zásuvný modul generátoru kódu pro testovací rámec *RedDeer* (*RedDeer CodeGen*) disponuje jednoduchým uživatelským rozhraním. Pro grafickou implementaci vlastností zásuvného modulu byla využita grafická komponenta z knihovny *JFace – Wizard* (průvodce nastavením).

Vytvořený průvodce nastavením je základní ovládací prvek celého zásuvného modulu generátoru kódu. Nastavení je z důvodu přehlednosti rozděleno na tři části (strany). Wizard lze otevřít/aktivovat prostřednictvím klávesové zkratky:

- **`ALT + G`** – otevře wizard generátoru kódu nad jiným aktivním oknem (např. otevře průvodce generátoru kódu nad aktivním průvodcem pro tvorbu nového *Java* projektu).



Obr. 6.3: První strana průvodce generátoru kódu *RedDeer*.

Na první straně průvodce nastavením (obr. 6.3) uživatel definuje základní vlastnosti generované *Java* třídy:

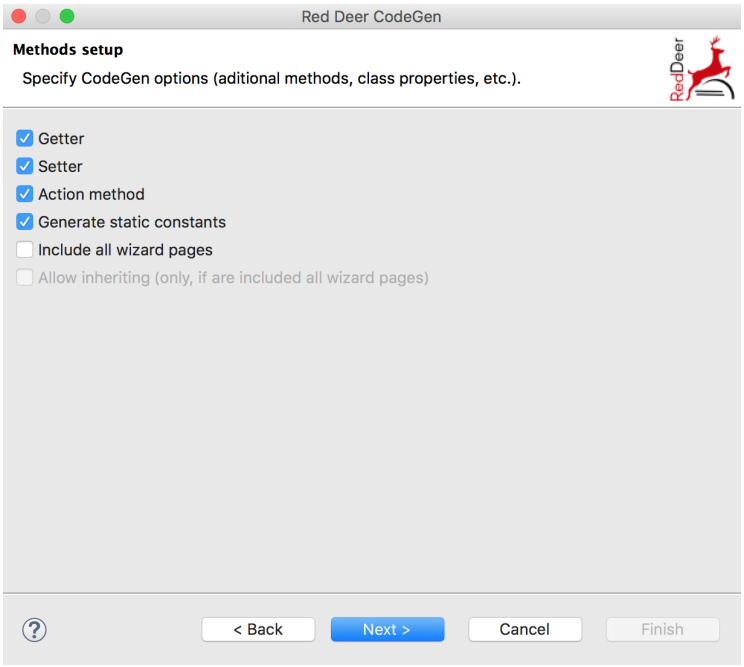
- **adresář zdrojových souborů** (*Source Folder*),
- **balíček** (*Package*), do kterého bude *Java* třída vygenerována,
- **název třídy** (*Name*).

Implementace pro nastavování těchto tří polí je děděna z rodičovské třídy pro tvorbu nových stran průvodce nastavením – **`NewTypeWizardPage.java`**, která je součástí *Eclipse* balíčku `org.eclipse.jdt.ui.wizards`. Zděděné vlastnosti byly následně přizpůsobeny pro zásuvný modul generátoru kódu *RedDeer CodeGen*.

Pro umožnění pokračování v průvodci na další stranu nastavení je uživatel povinen vyplnit všechna pole.

První strana disponuje funkcí předvyplnění polí *Source Folder* a *Package* za účelem usnadnění a urychlení práce. Pole jsou vyplněna na základě označení projektu před samotným spuštěním zasuvného modulu *RedDeer CodeGen*.

Druhá strana průvodce generátoru kódu (obr. 6.4) umožňuje uživateli přesněji specifikovat parametry a metody, které bude výsledná *Java* třída obsahovat. Volitelné vlastnosti generátoru se vybírají pomocí zaškrtávacích polí (tzv. checkboxů). Použité grafické prvky patřící do knihovny *SWT* – **`SWT.CHECK`**.



Obr. 6.4: Druhá strana průvodce generátoru kódu *RedDeer*.

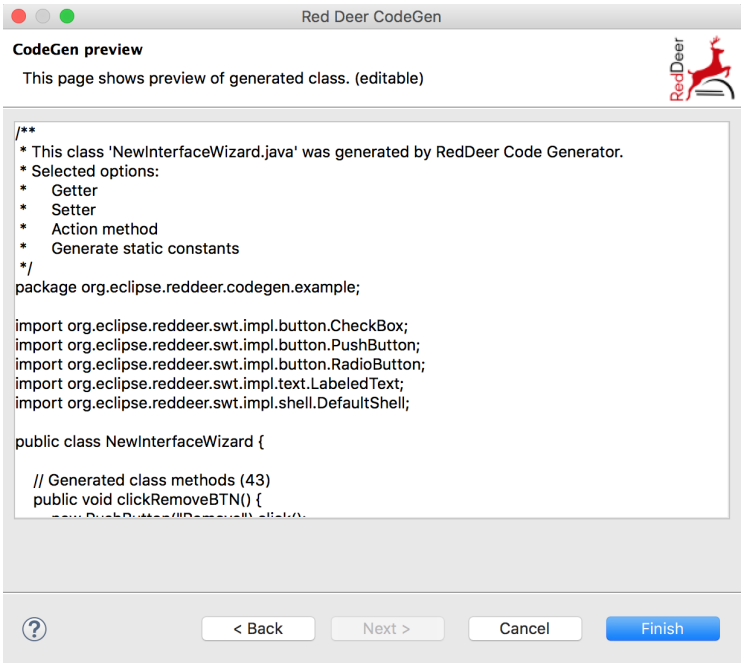
Základní předvybrané vlastnosti jsou:

- **get metoda** (*Getter*) – metoda vracející textový parametr widgetu,
- **set metoda** (*Setter*) – metoda nastavující novou hodnotu vlastnosti widgetu,
- **interakce** (*Action method*) – např. pro tlačítko se vygeneruje metoda kliknutí (interakce),
- **konstanty** (*Generate static constants*) – platí zejména pro grafický prvek, který slouží jako list možností (*Combo box*), pro každou z položek listu je vygenerována statická textová konstanta,

- **zahrnout všechny stránky průvodce** (*Include all wizard pages*) – po vybrání tohoto pole vyhledává generátor kódu všechny widgety ze všech stran právě aktivního grafického prvku (např. wizardu).

Poslední pole, které může uživatel vybrat je přímo zásivislé na předchazející možnosti (zahrnout všechny stránky průvodce). Za podmínky, že je toto pole zaškrtnuto, může uživatel vybrat následující vlastnost „dědění vlastností“, která je popsána níže:

- **dědění vlastností** (*Allow inheriting*) – zajišťuje dědění z již existujících *Java* tříd testovacího rámce *RedDeer*, obsahujících metody pro práci s grafickými prvky knihovny *JFace* (`WizardDialog.java` a `PreferenceDialog.java`), dědění je využito z důvodu možnosti použití již stávajících funkčností pro widgety, které jsou pro každý wizard společné, zejména tlačítka: *Back*, *Next*, *Cancel*, *Finish* (pro `WizardDialog`) a *OK*, *Cancel* (pro `PreferenceDialog`).



Obr. 6.5: Třetí strana průvodce generátoru kódu *RedDeer*.

Třetí strana průvodce generováním kódu (obr. 6.5) zobrazuje náhled vygenerovaného kódu ještě před tím, než bude vytvořen soubor do projektu. Náhled zdrojového kódu lze před stiskem tlačítka *Dokončit* (*Finish*) upravovat. Na této straně průvodce nastavením je z knihovny *SWT* použit grafický prvek textového pole s vlastností podpory víceřádkového textu – **`SWT.MULTI`**.

6.4 Interní logika generování kódu

Navržená interní logika generování kódu odpovídá diagramu znázorněnému na obrázku 6.6. Princip spočívá v zaznamenání klávesové zkratky (`ALT + G`), která následně otevře průvodce nastavením zásuvného modulu. Navazující logika generátoru kódu je popsána v následujících třech podkapitolách:

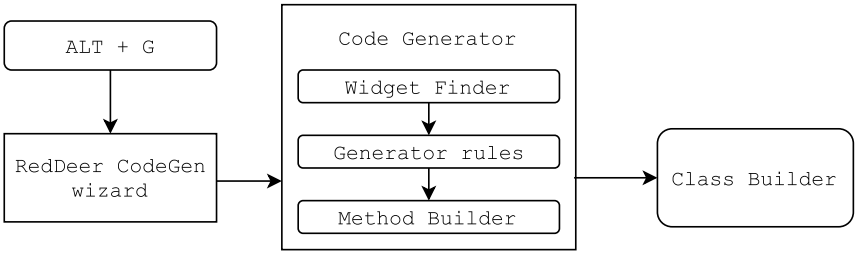
1. vyhledávání grafických prvků (widgetů) – *Widget Finder*,
2. generování kódu – *Generator rules*,
3. uspořádání *Java* třídy – *Class Builder*.

Vyhledávání grafických prvků

Důležitá část interní logiky generování kódu spočívá ve správném získání všech widgetů (grafických prvků) z okna (např. wizardu), nad kterým byl generátor kódu spuštěn. Generátor kódu je v základním nastavení implementován tak, že vyhledává pouze viditelné widgety na právě aktivní straně wizardu.

V zásuvném modulu generátoru kódu je vyhledávání widgetů implementováno prostřednictvím balíčku **`org.eclipse.reddeer.codegen.finder`**, který obsahuje třídy `Finder.java` a `ControlFinder.java`. Prostřednictvím těchto *Java* tříd je aktivní strana wizardu prohledána a výsledkem je seznam všech nalezených widgetů.

Grafické komponenty *Eclipse IDE* jsou uspořádány do stromové struktury, to znamená, že pro nalezení všech widgetů je nutno prohledávat každý rodičovský prvek (parent, rodič) do hloubky, tzn. otestovat zda některý ze zanořených widgetů (children, potomek) neodpovídá parametrům pro generování kódu.



Obr. 6.6: Diagram navržené interní logiky generování kódu.

Generování kódu

Generování kódu v zásuvném modulu řídí balíček **`org.eclipse.reddeer.codegen`**, ve kterém je implementována *Java* třída `CodeGenerator.java`. Úkolem této třídy je projít seznam vyhledaných widgetů z předchozího kroku a podle určených pravidel pro generování kódu získat metody a vlastnosti, které jsou poté dále zpracovávány.

Pravidla generování kódu jsou v projektu implementována v balíčcích:

- **`org.eclipse.reddeer.codegen.rules`**,
- **`org.eclipse.reddeer.codegen.rules.simple`**.

Pro každý widget je definováno speciální pravidlo, které udává počet, typ a tvar funkcí, které mohou být následně vygenerovány. Mezi momentálně podporované widgety s definovanými pravidly patří:

- tlačítka (*ButtonCodeGenRule*),
- seznamy hodnot (*ComboCodeGenRule*),
- textová pole (*TextCodeGenRule*),
- okna aplikace (*ShellCodeGenRule*).

Uspořádání *Java* třídy

Výstupem generátoru kódu je kompletní a kompilovatelná *Java* třída. Uspořádání probíhá na základě vygenerovaných funkcí a parametrů z předchozího kroku, které odpovídají vybraným vlastnostem v průvodci nastavení generátoru kódu. Výsledné uspořádání *Java* třídy je v zásuvném modulu generátoru kódu řešeno prostřednictvím balíčku **`org.eclipse.reddeer.codegen.builder`**, který obsahuje *Java* třídy:

- **`MethodBuilder.java`**,
- **`ClassBuilder.java`**.

MethodBuilder slouží pro tvorbu jednotlivých funkcí, které generátor kódu vytváří pro každý z nalezených widgetů.

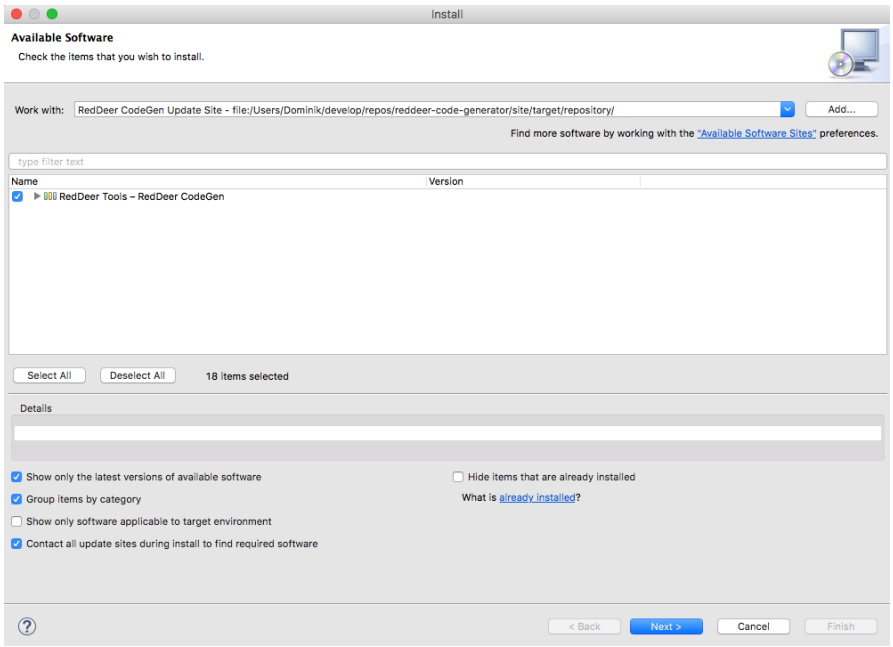
ClassBuilder představuje instanci kompletní *Java* třídy, která obsahuje závislosti (tzv. importy) použitých balíčků, tělo třídy, konstanty a definice funkcí, které generátor vygeneroval. Hlavní funkcí třídy *ClassBuilder* je správně uspořádat a naformátovat všechen vygenerovaný kód do výsledného tvaru validní *Java* třídy.

```
1  package org.balicek;
2  import org.eclipse.reddeer.swt.impl.shell.DefaultShell;
3  public class DefaultCodeGen {
4    // Generated methods (2)
5    public DefaultShell getShellNew() {
6      return new DefaultShell("New");
7    }
8    public String getTextNew() {
9      return new DefaultShell("New").getText();
10   }
11 }
```

Výpis 6.1: Ukázka výstupu generátoru kódu po uspořádání *Java* třídy.

6.5 Instalace do vývojového prostředí

Úspěšným dokončením kompilace projektu je vytvořen lokální instalační odkaz (local update site), ze kterého lze pomocí instalačního *Eclipse* manažeru (**`Help > Install New Software...`**) nainstalovat zásuvný modul do vývojového prostředí. V zobrazeném dialogovém okně je zapotřebí zadat cestu k instalačnímu odkazu (`<local-path>/reddeer-code-generator/site/target/repository`), znázorněno na obrázku 6.7. Po dokončení instalace a následném restartování *Eclipse IDE* je nový zásuvný modul již plně k dispozici.

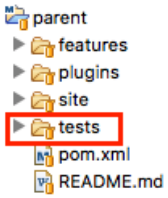


Obr. 6.7: Instalace zásuvného modulu *RedDeer CodeGen* do *Eclipse IDE*.

6.6 Experimentální ověření

Pro možnost ověření správné funkčnosti vytvořeného generátoru kódu pro testovací rámec *RedDeer* byla struktura projektu rozšířena o modul **`tests`** viz obr. 6.8, který je zapotřebí pro automatizované integrační testování.

Tento modul obsahuje vytvořený testovací scénář, který využívá pro práci s grafickým rozhraním metody vygenerované prostřednictvím zásuvného modulu ***RedDeer CodeGen***. Projekt testovacího scénáře byl vytvořen prostřednictvím šablony testovacího rámce *RedDeer* – **`jboss-reddeer-archetype`**. K otestování byl vybrán průvodce nastavením pro vytvoření nového *RedDeer* projektu (*New RedDeer Test Plug-in*).



Obr. 6.8: Rozšíření struktury *RedDeer CodeGen* projektu.

Postup vytvoření testovacího scénáře:

1. prostřednictvím výše zmíněné šablony vytvořit projekt,
2. otevřít průvodce (wizardu) pro vytvoření nového *RedDeer* projektu,
3. aktivovat *RedDeer CodeGen* a vygenerovat všechny potřebné metody (minimálně *Setter*, *Include all pages* a *Allow inheriting*),
4. posledním krokem je implementace testovacího scénáře, která je popsána níže.

Testovací scénář pro experimentální ověření lze spustit prostřednictvím vývojového prostředí (**`Run > Run Configurations... > Maven Build)`**, kde je nutné vytvořit novou spouštěcí konfiguraci, zadat cíle kompilace (*Goals*) pro automatizované integrační testování – **`clean verify`** a kompilaci spustit. Druhá možnost spuštění testovacího scénáře je prostřednictvím příkazové řádky, zadáním příkazu – **`mvn clean verify`**.

Popis vytvořeného testovacího scénáře:

1. otevření průvodce nastavenim pro vytvoření nového *RedDeer* projektu,
2. vytvoření instance třídy vygenerované generátorem kódu *RedDeer CodeGen*,
3. vyplnění všech povinných polí průvodce nastavením a dokončení vytvoření,
4. ověření, zda-li byl projekt správně vytvořen,
5. ověření, zda-li po vytvoření projektu nejsou ve vývojovém prostředí detekovány chyby.

7 ZÁVĚR

Cílem bakalářské práce byl návrh řešení pro ulehčení vytváření automatizovaných testů grafického rozhraní *Eclipse* prostřednictvím generovaného kódu a následná implementace navrženého řešení v podobě zásuvného modulu do vývojového prostředí *Eclipse*.

V rámci analýzy potřeb k řešení zadání bakalářské práce bylo nejprve nutné se blíže seznámit s vývojovým prostředím *Eclipse* a jeho architekturou. Dále bylo zapotřebí nastudovat grafické knihovny, které lze využít pro tvorbu uživatelského rozhraní v jazyce *Java* a lze je použít i pro vývoj grafického rozhraní v *Eclipse IDE*. V poslední části seznámení se s vývojovým prostředím *Eclipse* byla nastudována problematika rozšiřování vývojového prostředí *Eclipse* o nové zásuvné moduly a následné možnosti instalace těchto nových zásuvných modulů do *Eclipse IDE*.

Přesné znění zadání bakalářské práce je „návrh řešení pro ulehčení vytváření automatizovaných testů grafického rozhraní *Eclipse*“, proto bylo tedy dalším nezbytným postupem bližší nastudování možností a způsobů testování aplikací, konkrétně seznámení se s testovacím rámcem *RedDeer*, který slouží právě pro automatizované testování grafického rozhraní *Eclipse*.

Dalším požadavkem zadání bakalářské práce bylo připravit navržené řešení tak, aby mohlo být později integrováno do již zmíněného testovacího rámce *RedDeer*. Z toho důvodu byla dostudována a popsána oblast, která se věnuje sestavení a kompilaci aplikací prostřednictvím nástroje *Apache Maven*.

Po úspěšné analýze potřeb k řešení zadání bakalářské práce a získání všech potřebných znalostí byl navržen zásuvný modul generátoru kódu pro testovací rámec *RedDeer* – pojmenovaný „*RedDeer CodeGen*“.

První část vývoje generátoru kódu byla věnována vytvoření grafického rozhraní zásuvného modulu. Bylo vytvořeno jednoduché uživatelské rozhraní využívající grafické knihovny *SWT* a *JFace*. Pro aktivaci/zobrazení zásuvného modulu nad aktivním oknem vývojového prostředí byla v *Eclipse IDE* vyčleněna klávesová zkratka `ALT + G`.

Následoval návrh interní logiky pro generování kódu. V tomto kroku bylo nutné nejprve přijít na způsob vyhledávání všech grafických prvků aktivního okna vývojového prostředí *Eclipse*. Dále bylo zapotřebí přesně určit, které grafické prvky (widgety) z knihovny *SWT* bude generátor kódu podporovat, protože pro každý z podporovaných widgetů bylo vytvořeno speciální pravidlo generování kódu. Tato pravidla zvlášť pro každý grafický prvek specifikují metody, které mohou být vygenerovány. Posledním krokem návrhu interní logiky generování kódu bylo výsledné uspořádání generovaných metod do validní *Java* třídy.

Poslední kapitola vývoje generátoru kódu pro testovací rámec *RedDeer* byla vě-

nována experimentálnímu ověření funkčnosti implementovaného řešení. Prostřednictvím vytvořeného testovacího scénáře, který využívá metody pro práci s grafickým rozhraním vygenerované generátorem kódu *RedDeer CodeGen*, bylo ověřeno splnění všech požadavků automatizovaného testování grafického rozhraní a otestováno generování validních zdrojových kódů v programovacím jazyce *Java*.

Všechny zadané cíle této práce byly úspěšně splněny, ověřeny a výsledný projekt byl připraven tak, aby mohl být součástí testovacího rámce *RedDeer* a bylo možné nadále pracovat na jeho vývoji a rozšiřování o nové vlastnosti generování kódu pro větší množství *SWT* komponent vývojového prostředí *Eclipse*.

LITERATURA

[1] Eclipse RedDeer. *Eclipse.org: The Eclipse Foundation open source community website.* [online]. 2001 [cit. 2017-05-21]. Dostupné z: <https://projects.eclipse.org/projects/technology.reddeer>

[2] What is Maven? *The Apache Software Foundation* [online]. 1999 [cit. 2016-12-08]. Dostupné z: <http://maven.apache.org/what-is-maven.html>

[3] MOIR, Kim. Eclipse. *The Architecture of Open Source Applications* [online]. [cit. 2016-12-08]. Dostupné z: <http://www.aosabook.org/en/eclipse.html>

[4] Eclipse Public License. *Eclipse.org: The Eclipse Foundation open source community website.* [online]. Verze 1.0. 2001 [cit. 2017-05-21]. Dostupné z: <https://eclipse.org/org/documents/epl-v10.php>

[5] Eclipse Platform: Platform architecture. *Eclipse.org: The Eclipse Foundation open source community website.* [online]. 2001 [cit. 2017-05-21]. Dostupné z: <http://help.eclipse.org/neon/index.jsp?topic=%2Forg.eclipse.platform.doc.isv%2Fguide%2Farch.htm&cp=2_0_1>)

[6] Eclipse Platform: The Workbench. *Eclipse.org: The Eclipse Foundation open source community website.* [online]. 2001 [cit. 2017-05-21]. Dostupné z: <http://help.eclipse.org/neon/index.jsp?topic=%2Forg.eclipse.platform.doc.user%2FgettingStarted%2Fqs-02a.htm>

[7] OSGi Architecture. *OSGi Alliance: The Dynamic Module System for Java* [online]. 1999 [cit. 2017-05-21]. Dostupné z: <https://www.osgi.org/developer/architecture/>

[8] Eclipse Platform: Plug-ins and bundles. *Eclipse.org: The Eclipse Foundation open source community website.* [online]. 2001 [cit. 2017-05-21]. Dostupné z: <http://help.eclipse.org/neon/index.jsp?topic=%2Forg.eclipse.platform.doc.isv%2Fguide%2Fruntime_model_bundles.htm&cp=2_0_3_0_0>

[9] Equinox. *Eclipse.org: The Eclipse Foundation open source community website.* [online]. 2001 [cit. 2016-12-08]. Dostupné z: <http://www.eclipse.org/equinox/>

[10] Eclipse Platform: Resources and the workspace. *Eclipse.org: The Eclipse Foundation open source community website.* [online]. 2001 [cit. 2017-05-21].

Dostupné z: <http://help.eclipse.org/neon/index.jsp?topic=%2Forg.eclipse.platform.doc.isv%2Fguide%2FresInt_workspace.htm>

[11] Abstract Window Toolkit. *Oracle* [online]. 1995 [cit. 2017-05-21]. Dostupné z: <http://docs.oracle.com/javase/8/docs/technotes/guides/awt/>

[12] Java Foundation Classes: A Foundation for Accessibility. *IBM: International Business Machines Corp.* [online]. 1911 [cit. 2017-05-21]. Dostupné z: <https://www-03.ibm.com/able/guidelines/java/snsjavagjfc.html>

[13] Painting in AWT and Swing. *Oracle* [online]. 1995 [cit. 2017-05-21]. <Dostupnéz:http://www.oracle.com/technetwork/java/painting-140037.html>

[14] Eclipse Platform: The Standard Widget Toolkit. *Eclipse.org: The Eclipse Foundation open source community website.* [online]. 2001 [cit. 2017-05-21]. Dostupné z: <http://help.eclipse.org/neon/index.jsp?topic=%2Forg.eclipse.platform.doc.isv%2Fguide%2Fswt.htm&cp=2_0_7>

[15] Eclipse Platform: The JFace UI framework. *Eclipse.org: The Eclipse Foundation open source community website.* [online]. 2001 [cit. 2017-05-21]. Dostupné z: <http://help.eclipse.org/neon/index.jsp?topic=%2Forg.eclipse.platform.doc.isv%2Fguide%2Fjface.htm&cp=2_0_6>

[16] Eclipse IDE Plug-in Development: Plug-ins, Features, Update Sites and IDE Extensions. *Eclipse, Android and Java training and support* [online]. [cit. 2017-05-21]. Dostupné z: <http://www.vogella.com/tutorials/EclipsePlugin/article.html>

[17] Developing Eclipse plug-ins: How to create, debug, and install your plug-in. *IBM: International Business Machines Corp.* [online]. 1911 [cit. 2017-05-21]. Dostupné z: <https://www.ibm.com/developerworks/library/os-ecplug/>

[18] Unit Testing with JUnit. *Eclipse, Android and Java training and support* [online]. Verze 4.3. 21.06.2016 [cit. 2017-05-21]. Dostupné z: <http://www.vogella.com/tutorials/JUnit/article.html>

[19] *JUnit* [online]. Verze 4.12. 2002, 2017-01-11 [cit. 2017-05-21]. Dostupné z: <http://junit.org/junit4/>

[20] KRIPNER, Matěj. Unit testy v Javě a JUnit. *ITnetwork* [online]. Praha [cit. 2017-05-21]. Dostupné z: <https://www.itnetwork.cz/java/pokrocile/java-unit-testy-v-junit>

[21] SWTBot: What is SWTBot? *Eclipse.org: The Eclipse Foundation open source community website.* [online]. 2001 [cit. 2017-05-21]. Dostupné z: <`http://www.eclipse.org/swtbot/`>

[22] GEF. *Eclipse.org: The Eclipse Foundation open source community website* [online]. 2001 [cit. 2017-05-21]. Dostupné z: <`https://eclipse.org/gef/`>

[23] Eclipse RedDeer: User Guide. [online dokumentace]. 2017 [cit. 2017-05-21]. Dostupné z: <`https://github.com/eclipse/reddeer/wiki/User-Guide`>

[24] JVM (Java Virtual Machine). *Javatpoint* [online]. 2011 [cit. 2017-05-21]. Dostupné z: <`https://www.javatpoint.com/internal-details-of-jvm`>

[25] Introduction of the build lifecycle. *The Apache Software Foundation* [online]. 1999 [cit. 2017-05-21]. Dostupné z: <`https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html`>

[26] Tycho. *Eclipse.org: The Eclipse Foundation open source community website.* [online]. 2001 [cit. 2017-05-21]. Dostupné z: <`https://eclipse.org/tycho/`>

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

API	*Application Programming Interface*
AWT	*Abstract Windowing Toolkit*
EPL	*Eclipse Public License*
HTML	*HyperText Markup Language*
IDE	*Integrated Development Environment*
JAR	*Java ARchive*
JDT	*Java Development Tools*
JRE	*Java Runtime Environment*
JVM	*Java Virtual Machine*
PDE	*Plug-in Development Environment*
SDK	*Standard Development Toolkit*
SWT	*Standard Widget Toolkit*
XML	*eXtensible Markup Language*

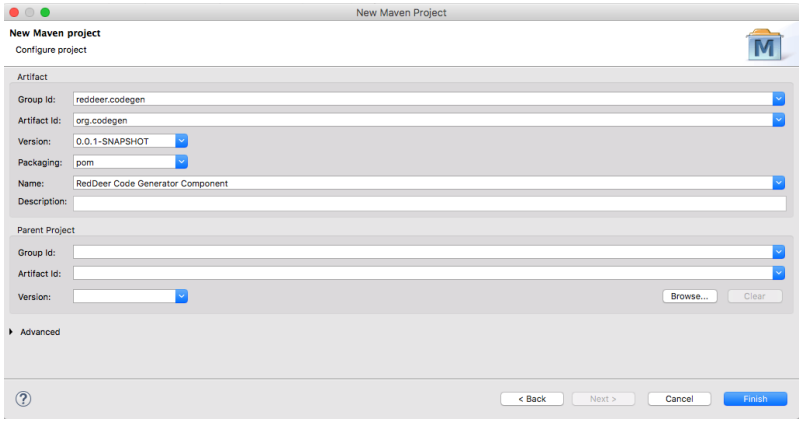
SEZNAM PŘÍLOH

A Návod pro vytvoření jednotlivých částí projektu **51**

B Obsah přiloženého DVD **55**

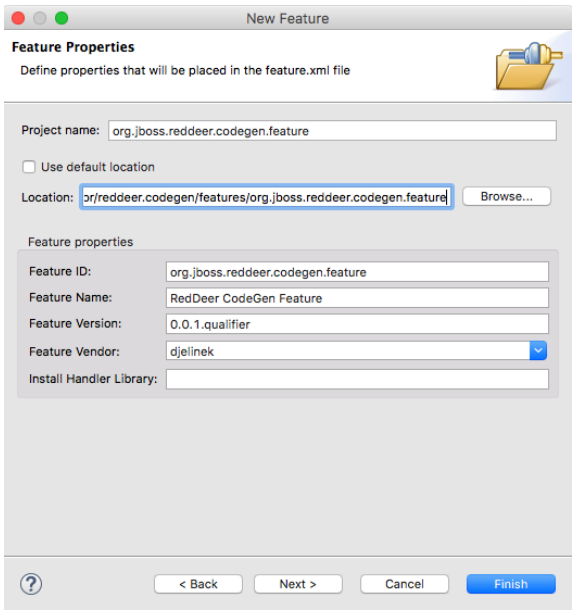
A NÁVOD PRO VYTVOŘENÍ JEDNOTLIVÝCH ČÁSTÍ PROJEKTU

Pro implementaci řešení *RedDeer CodeGen* plug-inu je zapotřebí vytvořit ***Maven project*** (`File > New > Other... > Maven > Maven Project`).



Obr. A.1: Nastavení parametrů *Maven* projektu.

Poté v otevřeném dialogovém okně zaškrtnout možnost *Create a simple project (skip archetype selection)* a po stisku tlačítka **`Next >`** vyplnit údaje dle obrázku A.1. Nakonec potvrdit vytvoření projektu tlačítkem **`Finish`**.

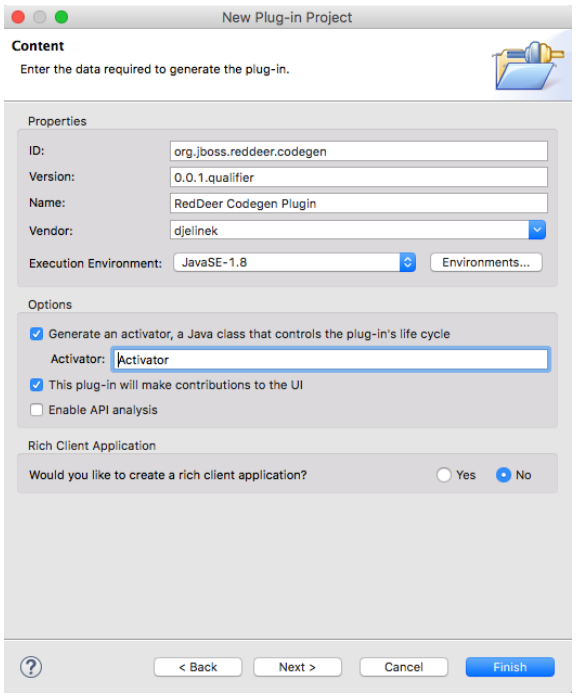


Obr. A.2: Nastavení parametrů *RedDeer CodeGen* Feature projektu.

Uvnitř nově vytvořeného projektu je nutné připravit adresářovou strukturu projektu, která odpovídá struktuře na obrázku 6.2.

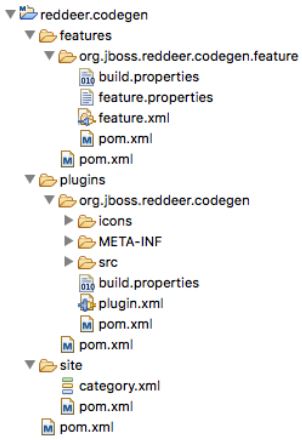
Jakmile je struktura projektu připravena, lze začít postupně vytvářet jednotlivé moduly projektu – **feature**, **plugin**.

Každý z modulů projektu je zapotřebí vytvářet do správného adresáře v připravené adresářové struktuře projektu (`feature > features, plugin > plugins`).



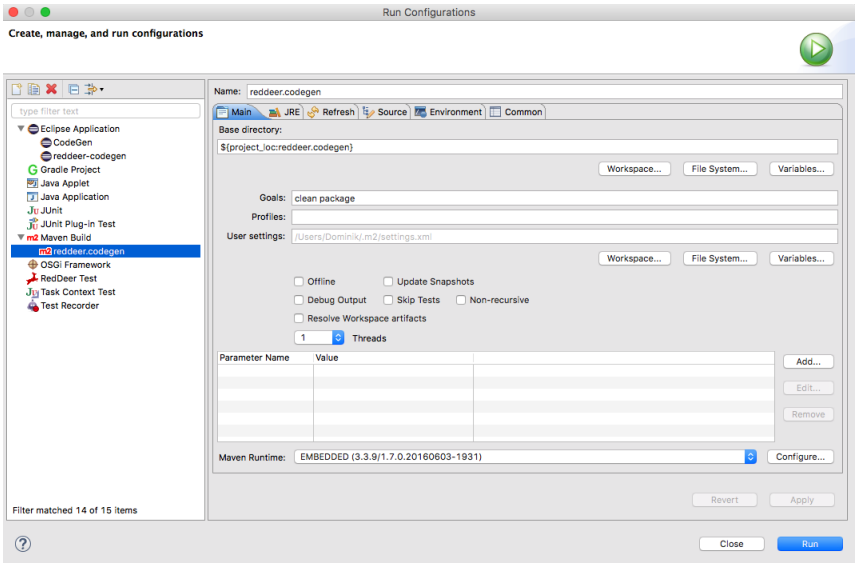
Obr. A.3: Nastavení parametrů *RedDeer CodeGen* Plug-in projektu.

Modul *Features*, nebo-li *Feature Project* (lze nálezt v menu vývojového prostředí `File > New > Other... > Plug-in Development > Feature Project`). Nastavení parametrů *Feature projektu* je stejné jako na obrázku A.2 a vytvoření projektu potvrdit tlačítkem **`Finish`**.



Obr. A.4: Výsledná struktura *RedDeer CodeGen* projektu.

Modul *Plugins*, nebo-li *Plug-in Project* (lze nálezt v menu vývojového prostředí `File > New > Other... > Plug-in Development > Plug-in Project`). Cestu pro vytvoření plug-in projektu zvolit (.../`plugins/org.jboss.reddeer.codegen`).



Obr. A.5: Nastavení pro spuštění *Maven* buildu.

Další nastavení parametrů *Plug-in projektu* bude stejné jako na obrázku A.3. Vytvoření projektu potvrdit stiskem tlačítka **`Finish`**. Do vytvořeného *Plug-in projektu* je zapotřebí implementovat všechny vlastnosti generátoru kódu *RedDeer CodeGen*. Jakmile je zásuvný modul dokončen, je posledním krokem vytvoření a implementace souborů `pom.xml` (viz zdrojové soubory v příloze) na příslušná místa dle výsledné adresářové struktury projektu na obrázku A.4.

Ověření funkčnosti vytvořeného *Maven* projektu lze provézt spuštěním buildu. V *Eclipse IDE* lze nastavení *Maven* buildu upravit prostřednictvím *Run Configurations* (`Run > Run Configurations...`), pro nový čistý build projektu se používá parametr – **`clean package`** viz obrázek A.5.

```
[INFO] Reactor Summary:
[INFO]
[INFO] RedDeer CodeGen Parent POM file ..................... SUCCESS [  0.129 s]
[INFO] RedDeer CodeGen Plugins ............................. SUCCESS [  0.010 s]
[INFO] RedDeer CodeGen Plugin .............................. SUCCESS [  2.381 s]
[INFO] RedDeer CodeGen Features ............................ SUCCESS [  0.005 s]
[INFO] RedDeer CodeGen Feature ............................. SUCCESS [  0.333 s]
[INFO] RedDeer CodeGen Update Site ......................... SUCCESS [ 10.364 s]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 34.690 s
[INFO] Finished at: 2016-12-02T04:31:35+01:00
[INFO] Final Memory: 78M/619M
[INFO] ------------------------------------------------------------------------
```

Obr. A.6: Úspěšně sestavený projekt.

Za předpokladu, že build projektu proběhl bez chyb, bude výstup v konzoli *Eclipse IDE* přibližně podobný výstupu zobrazeném na obrázku A.6.

Úspěšným dokončením buildu projektu je vytvořena *Update site*, ze které lze pomocí *Eclipse update manager* (`Help > Install New Software...`) nainstalovat zásuvný modul do vývojového prostředí. V zobrazeném dialogovém okně je nutné zadat cestu k lokálnímu úložišti (local repository), znázorněno na obrázku 6.7. Po dokončení instalace a následném restartování *Eclipse IDE* je nový zásuvný modul již plně k dispozici.

B OBSAH PŘILOŽENÉHO DVD

Přiložené DVD obsahuje elektronickou verzi bakalářské práce, zdrojové soubory a soubory nezbytné k instalaci. Hlavní dokument elektronické verze bakalářské práce „*xjelin43-bp*“ je umístěn v adresáři `dokumentace`. Zdrojové soubory jsou zabaleny do archivu a umístěny v adresáři `projekt`. Zdrojové soubory lze po rozbalení importovat do vývojového prostředí *Eclipse* jako existující *Maven* projekt. Poslední adresář `repository` je určen k instalaci projektu do vývojového prostředí prostřednictvím instalačního *Eclipse* manažeru (*Eclipse update manager*). Zbylé soubory, které jsou umístěny v kořenovém adresáři obsahují dodatečné informace k projektu a nebo k instalaci zásuvného modulu (`navod-instalace.pdf` a `README.txt`).

- `/` ..... kořenový adresář přiloženého DVD
  - `dokumentace`
    - `xjelin43-bp.pdf` ..... elektronická verze bakalářské práce
  - `projekt`
    - `reddeer-code-generator.zip` ..... zdrojové soubory generátoru kódu
    - `README.txt` ..... popis obsahu balíčku zdrojových souborů
  - `repository` ..... adresář pro instalaci do *Eclipse IDE*
  - `navod-instalace.pdf` ..... návod na instalaci do *Eclipse IDE*
  - `README.txt` ..... soubor s popisem obsahu DVD