Cumulant – program usnadňující tvorbu série výukových projektů postupně kumulujících funkcionalitu vyvíjeného programu

Rudolf Pecinovský

ICZ a.s., Na Hřebenech II 1817, 140 00 Praha 4
VŠE Praha, Nám. W. Churchilla 4, 130 67 Praha 3
rudolf@pecinovsky.cz

Abstrakt. Článek seznamuje s programem, který usnadní přípravu sérií výukových projektů, v nichž se vyvíjený program postupně zdokonaluje, přičemž každá z etap je prezentována jako samostatný projekt přiřazený k příslušné lekci učebnice nebo výukového kurzu. Vysvětluje, jak lze specifikovat sestavení jednotlivých nejlépe projektů ze tříd definovaných v souhrnném, učitelském projektu a ukazuje, jaké prostředky program nabízí k tomu, aby mohl učitel definovat několik verzí postupně vyvíjené třídy v jednom zdrojovém souboru a jak specifikovat, která část zdrojového kódu se zanese do výsledného programu jednotlivých lekcí.

1 Úvod

Připravujeme-li učebnice či kurzy programování, tak je doplňujeme sadou programů, na nichž buď probíranou látku vysvětlujeme, anebo nám slouží jako vzorová řešení úkolů, které řeší studenti, aby si ověřili, že vykládanou látku pochopili. Jak je naznačeno v [3], doprovodné programy můžeme navrhovat podle několika koncepcí. Nejvhodnější, ale bohužel také nejpracnější, je koncepce, při níž látku demonstrujeme na několika málo (nejlépe na jednom) postupně vyvíjených a vylepšovaných projektech, které studenti občas doplňují o vlastní výtvory.

Pracnost uvedené koncepce spočívá v tom, že musíme neustále dbát na vzájemnou konzistenci jednotlivých verzí vyvíjených programů. Přitom se může stát, že při vývoji verzí pro pozdější lekce zjistíme, že musíme upravit i zdrojové kódy verzí předchozích. Takováto oprava pak zákonitě ovlivní všechny verze počínaje tou nejstarší upravovanou.

Důsledkem těchto nepříjemných vlastností je to, že pokud se již nějaký vyučující rozhodne doprovodit svůj kurz či učebnici takovýmito doprovodnými programy, snaží se je měnit pouze přidáváním celých zdrojových souborů. Potřebují-li výjimečně udělat v některém z dříve definovaných zdrojových souborů úpravy, snaží se omezit na pouhé přidání dalších metod.

S většinou nepříjemných průvodních jevů vývoje takto koncipovaných doprovodných programů může pomoci aplikace *Cumulant*, s jejímiž základními vlastnostmi seznamuje tento článek.

2 Celková koncepce

Aplikace *Cumulant* je koncipována jako samorozbalovací archiv, který obsahuje zdrojové kódy zastřešujícího projektu a případné další potřebné soubory. Z nich pak na požádání vygeneruje zdrojové kódy požadovaných verzí doprovodných programů daného kurzu či knihy.

Protože zastřešující projekt má v sobě začleněny všechny soubory všech generovaných projektů, je v dokumentaci programu označován jako *monolit*. Takto bude označován i v dalším textu tohoto článku.

Generace požadovaných verzí doprovodných programů probíhá interně ve dvou etapách:

- V první etapě si generátor zjistí, z jakých souborů monolitu mají být jednotlivé požadované projekty sestaveny a jak mají být tyto soubory v generovaných projektech uspořádány do balíčků.
- V druhé etapě pak převádí jednotlivé soubory do vytvářených projektů, přičemž každý ze souborů prochází preprocesorem, který upraví výslednou podobu souboru do tvaru odpovídajícího danému cílovému projektu.

Celý generátor i se zdrojovými soubory vytvářených projektů je umístěn v archivu JAR, v němž bývají typicky vytvořeny dvě virtuální složky:

- Složka `AUX` obsahující soubor `Projects.txt` s informacemi pro řízení požadované generace projektů a případné další pomocné soubory. Jedním z nich je i soubor `Basic.properties`, v němž jsou umístěny některé klíčové informace týkající se plánovaného převodu. Mezi nimi hraje důležitou roli vlastnost `rrf` (relative root folder), která zadává balíček, jenž je (pra)rodičovským balíčkem všech balíčků vytvářeného projektu a označujeme jej proto jako relativní kořenem souborů celého monolitu. Vůči němu se pak mohou zadávat relativních adresy všech převáděných souborů.
- Složka `SRC` obsahující zdrojové soubory monolitu. Tato složka je sice nepovinná, protože veškerá data mohou být ve složce `AUX`, ale z praktických důvodů bývá vhodné umístit zdrojové soubory ve zvláštní složce, kam je zkopírujeme ze svého pevného disku.

3 Řídící soubor `Projects.txt`

Pro specifikaci obsahu jednotlivých cílových projektů byl vytvořen velice jednoduchý DSL (Domain Specific Language). Teoreticky by sice bylo možno využít formátu XML nebo JSON, jenomže oba formáty jsou zbytečně „upovídané“, protože v zájmu obecnosti vyžadují konkrétní specifikaci každého zadaného údaje. Tu lze ale jednoduše odvodit z jeho umístění údaje v textu. Syntaxe jazyka řídícího souboru je jednoduchá – její zápis v EBNF je ve výpisu 1.

Jazyk je řádkově orientovaný, takže každý příkaz je na jednom řádku. Program čte řídící soubor, interpretuje v něm nalezené příkazy a předzpracované informace pak předává objektům, které mají na starosti přípravu podkladů pro následující tvorbu projektů. Pravidla pro jednotlivé řádky řídícího souboru jsou následující:

- Je-li řádek prázdný (= není-li na něm žádný nebílý znak), je ignorován.
- Začíná-li řádek znakem měny (¤), je považován na komentář a je ignorován; v jiných pozicích je znak měny používán jako separátor.
- V řádcích začínajících `PROJEKT` ¤ se text za separátorem interpretuje jako název projektu a současně i jeho kořenové složky, tj. složky, do níž bude projekt vygenerován.
- V řádcích začínajících `PACKAGE` ¤ se text za separátorem interpretuje jako název balíčku, takže specifikuje název cílové složky převáděných souborů v rámci vytvářeného projektu.
- V řádcích začínajících `FOLDER` ¤ se text za separátorem interpretuje jako cesta ke složce, z níž se budou číst všechny následující soubory (tj. název zdrojového souboru se složí s cestou) dokud nebude nastaveno jinak.
- V řádcích začínajících `BLOCK` ¤ se text za separátorem interpretuje jako název bloku, což je skupina tříd, kterou se má opakovaně vkládat do více projektů, a proto je vhodné ji definovat separátně.
- V řádcích začínajících `INCLUDE` ¤ se text za separátorem interpretuje jako název bloku, který se má na daném místě vložit do aktuálně vytvářeného projektu.
- Řádek začínající textem `END` označuje poslední řádek vstupu, za nímž jsou už jen odložené informace, které aktuální generaci nijak neovlivní.
- Neplatí-li žádné z předchozích pravidel, je text vlevo od prvního separátoru považován za název cílového souboru a text vpravo od separátoru za název zdrojového souboru, který může být navíc zadán včetně relativní cesty. Není-li uveden název zdrojového souboru, předpokládá se, že je stejný jako název souboru cílového.

  Obsahuje-li text alespoň dva separátory, je text za druhým z nich interpretován jako ID projektu, do nějž bude soubor v následujícím kroku převáděn.

Předchozí popis není podrobnou uživatelskou dokumentací aplikace. Pouze se snaží ukázat, že aplikace myslí i na mnohé nestandardní potřeby, které mohou při tvorbě takto koncipovaných doprovodných programů nastat.

Výpis 1: Syntaxe jazyka specifikujícího obsah jednotlivých projektů

```
Program = { Block | Project }
BPBody  = { Package }
Block   = "BLOCK"  "¤"  BlockName  [ "¤" Comment ]
          "\n"  BPBody
Project = "PROJECT"  "¤"  ProjectName  [ "¤" Comment ]
          "\n"  BPBody
Package = "PACKAGE"  "¤"  [ PackageName ]  "\n"
          { Folder  |  Include }
Folder  = "FOLDER"  "¤"  [ ">P"  |  SourcePath ]
          "\n"  { File }
Include = "INCLUDE"  "¤"  BlockName  "\n"
File    = { DestFileName  "¤"
                          ( "¤"  |  SourceFileName )
```

```
[ "¤"  ProjectID ]  "\n"
```

4. Organizace zdrojových souborů

Předchozí kapitola popisovala, jak lze specifikovat, které soubory budou v jednotlivých verzích vytvářených projektů. Vedle toho je ale potřeba specifikovat i to, jak se budou zdrojové soubory z monolitu převádět na soubory cílových projektů.

Základní problém převodu spočívá v tom, že většina zdrojových souborů vystupuje v průběhu vývoje v několika postupně vylepšovaných podobách. Není vhodné mít pro každou z těchto podob vlastní verzi zdrojového souboru, protože tím vyvstávají velké problémy spojené s udržováním jejich vzájemné konzistence. Na druhou stranu ani není vhodné specifikovat všechny úpravy do jediného souboru, protože při větším počtu úprav se tento soubor stane naprosto nepřehledným a náchylným k zanesení nejrůznějších chyb.

Rozdělíme-li však posloupnost definic do více souborů, budeme muset vyřešit problém s jejich pojmenováním. Veřejné třídy se v Javě musejí jmenovat stejně jako jejich zdrojový soubor. Mohli bychom dát jednotlivé verze do různých balíčků, ale pak by nám mohl počet balíčků v monolitu nepříjemně narůst.

Při vývoji aplikace bylo rozhodnuto, že třídy, které budou nakonec vloženy do projektů, budou definovány jako vnořené, tj. statické interní třídy vnějších tříd, jejichž název se bude skládat z názvu dané (vnořené) třídy následovaným informací o rozsahu lekcí, pro něž je určen příslušný zdrojový kód. Dokumentační komentář těchto vnějších tříd by mohl obsahovat informace o tom, čím se liší jednotlivé podoby jejich vnitřních tříd, abychom si tak usnadnili rozhodování a rozsahu případných pozdějších úprav.

5. Konverze zdrojových souborů do cílového tvaru

Bylo nutno také rozhodnout jak zanést do zdrojového kódu třídy informace o tom, jak budou vypadat její jednotlivé verze dané třídy v různých lekcích. K tomu byly nakonec použity preprocesorové příkazy, které měly tvar řádkových komentářů. Byly použity následující preprocesorové komentáře:

`//%%+`

Ukončuje úvodní přeskakovanou pasáž, která obsahující řádky, které se nebudou kopírovat do cílového souboru.

`//%A+`

Uvozuje skupinu řádků, které budou přidány do cílového souboru pouze při splnění podmínek daných parametry na zbytku řádku (budou rozebrány dále). Skupina musí být ukončena preprocesorovým komentářem `//%A-`.

`//%I+`
Uvozuje skupinu řádků, které jsou v daném souboru zakomentovány, aby soubor vyhovoval syntaktickým pravidlům. Při splnění podmínek daných parametry na zbytku řádku budou tyto řádky odkomentovány a zařazeny do cílového souboru, při jejich nesplnění se celá skupina při převodu přeskočí. Skupina musí být ukončena preprocesorovým komentářem `//%I-`.

`//%X+`
Uvozuje skupinu příkazů, které budou při splnění podmínek daných parametry na zbytku řádku při zařazování do cílového souboru přeskočeny. Nebudou-li podmínky splněny, stanou se součástí výstupního souboru. Skupina musí být ukončena preprocesorovým komentářem `//%X-`.

`//%%-`
Uvozuje konec souboru, která se do cílového souboru nikdy nepřevádí.

6 Parametry preprocesorových komentářů `//%X+`, `//%I+` a `//%X+`

Parametry preprocesorových komentářů určují, pro které projekty se má daný preprocesorový komentář aktivovat. Zapisují se ve tvaru (symbol `4P` označuje první 4 znaky názvu projektu, které specifikují jeho `ID`):

`<4P` mezi vybrané se přidají projekty s ID menším než zadané

`<=4P` mezi vybrané se přidají projekty s ID menším nebo rovným zadanému

`=4P` mezi vybrané se přidá projekt se zadaným ID

`>4P` mezi vybrané se přidají projekty s ID větším než zadané

`>=4P` mezi vybrané se přidají projekty s ID větším nebo rovným zadanému

`-4P` z množiny doposud vybraných projektů se odeberou projekty s ID rovným či větším než zadané,

`-=4P` z množiny doposud vybraných projektů se odeberou projekty s ID větším než zadané

Parametry přitom musejí být zadávány v pořadí odpovídajícímu rostoucímu ID. Preprocesorový komentář

```
//%A+ >105 -110  >=115
```

uvozuje skupinu, která bude zařazena do výstupního souboru pro projekty s ID od 105 do 110 a pak pro projekty s ID větším nebo rovným 115.

7. Závěr

Příspěvek seznámil s programem, který může výrazně usnadnit přípravu doprovodných programů k učebnicím a kurzům, při nichž se vyvíjený program v průběhu výuky postupně zdokonaluje a každé jeho postupná verze odpovídající stavu v příslušné lekci nebo dokonce jen její částí je zveřejněna jako samostatný projekt, aby studenti mohli co nejsnadněji sledovat jednotlivé kroky a při problémech se opět mohli snadno vrátit k předchozímu kroku. Postupně ukázal jednotlivé řešené problémy a způsob jejich řešení.

Poděkování

Tento příspěvek vznikl za podpory grantu *Výzkum metodik výuky programování a možností jejich zlepšení* vypsaného nadací RPF.

Literatura

[1] BERGIN, Joseph: Fourteen Pedagogical Patterns. *Proceedings of Fifth European Conference on Pattern Languages of Programs.* (EuroPLoP™ 2000) Irsee 2000.

[2] POST Ed: *Real Programmers Don't Use Pascal*, Datamation 1983, For downoad at http://www.ee.ryerson.ca/~elf/hack/realmen.html.

[3] PECINOVSKÝ Rudolf: *Programming Textbooks and Courses.* Proceeding of Objects 2011. ISBN 978-80-554-0432-5.