Třetí část pojednává o pokročilejších programátorských technikách a také o styku Perlu s jeho okolím.

Velmi důležitá je hned první kapitola o modulech, jejich vytváření a používání. Prostřednictvím modulů totiž můžete snadno a rychle rozšiřovat schopnosti jazyka v těch směrech, ve kterých to potřebujete. Perliví programátoři zastávají názor, že byste pokud možno měli psát moduly do svých programů tak, aby byly později znovu použitelné při řešení dalších problémů. Hovoří v této souvislosti o ekologickém programování.

Následují odkazy, jejichž prostřednictvím můžete vytvářet libovolně složité datové struktury. Kniha pokračuje kapitolou o komunikaci vašeho programu s uživatelem a s prostředím, které jej obklopuje.

Objektově orientované programování je další z velkých témat současnosti. Dozvíte se zde, jak tyto techniky používat v Perlu a co od nich můžete očekávat. V kapitole o funkcionálním programování si pak trochu zažonglujeme s funkcemi.

Závěrečná dvojice kapitol je věnována práci s databázemi a vytváření programů, které se používají ke zpracování informací pro web. Právě na tomto poli si Perl vydobyl velmi silnou pozici.

10 Moduly

Nenamoduluje-li mne Julie, namoduluji ji já. Protože moduly jsou moderní. Moduly jsou módní.

Moduly umožňují rozdělit zdrojový text vašeho programu do několika značně nezávislých částí. To se uplatní v řadě případů: U velkých projektů, aby velikost souborů nepřesáhla rozumnou mez. U týmových projektů, kdy různí programátoři pracují na různých částech programu. A především při využívání celé řady již existujících modulů.

Znalci tvrdí, že samotný Perl je jen polovina zábavy. Zbytek tvoří využívání početného množství modulů, které lze získat v archivu CPAN (viz příloha 18 na straně 275), případně i jinde.

V této kapitole vás seznámím se základními mechanismy, které budete při využívání a vytváření modulů potřebovat.

10.1 Balíky

Jakmile na programu pracuje několik lidí nebo chcete použít nějakou existující knihovnu, objeví se problém s identifikátory. Jak se vyhnout jejich kolizím? Aby programátor Vomáčka nepoužil pro svůj pomocný podprogram vyzvedávající data z databáze název *vyzvedni*, protože ho již použila programátorka Svíčková pro svůj podprogram realizující vyzvednutí zboží ze skladu.

Pokud si vzpomínáte, tenhle problém jste už jednou viděli. Tehdy se jednalo o názvy proměnných v podprogramech a jeho řešením bylo zavedení lokálních identifikátorů. Podobné řešení existuje i zde. Nazývá se balík (package).

Pomocí balíků lze identifikátory rozdělit do izolovaných skupin, oficiálně se jim říká *jmenné prostory*. Nerad to přiznávám, ale až dosud jsem vám vlastně lhal. Ve skutečnosti se *každý* identifikátor skládá ze dvou částí: jména balíku a vlastního jména. Navzájem se oddělují dvojicí dvojteček. Například název proměnné má plný tvar:

$»balík«::»proměnná«

Každý balík je světem sám pro sebe a identifikátory v něm nemají nic společného s identifikátory ostatních balíků. Takže pokud bude programátor Vomáčka realizovat balík *databaze* a programátorka Svíčková balík *sklad*, nic jim nebrání, aby si zavedli podprogramy *databaze::vyzvedni* a *sklad::vyzvedni*. Nebudou si nijak překážet.

Abyste se neupsali, zavádí Perl příkaz:

package *»jméno«*;

Od okamžiku jeho použití se *»jméno«* stává implicitním jménem balíku. Pokud u identifikátoru neuvedete balík, automaticky se mu předřadí „*»jméno«::*“. Příkaz **package** platí tak dlouho, dokud nenastane jedna z následujících situací:

- použijete další **package**, kterým nastavíte jiný implicitní balík,
- příkaz **package** byl použit uvnitř bloku a tento blok skončil nebo
- skončil soubor, ve kterém byl **package** použit.

Implicitním balíkem je *main*. Takže všechny identifikátory, o kterých jsem se dosud zmiňoval, měly plný název *main::cosi*. Viděli jste to třeba v ukázce chybového hlášení na straně 34.

Příklad: Jednoduchý program:

```
1  package Data;
2  $cena = 23.50;
3
4  package main;
5  $pocet = 10;
6  $cena = $Data::cena * $pocet;
7
8  print "Celková cena: $cena (s DPH ";
9  print $cena*1.22, ")\n";
```

zavádí dva balíky (*Data* a *main*) a celkem tři proměnné (*$Data::cena*, *$main::pocet* a *$main:cena*). Skutečnost, že proměnná *$cena* existuje v obou balících není nijak na závadu. Jen když chci použít identifikátor z jiného balíku, musím to explicitně vyjádřit, jako na řádku 6. ■

Vřele vám *ne*doporučuji střídat několik balíků v rámci jednoho souboru. Je dobrým zvykem použít **package** jen jednou, a to na začátku souboru. Pokud si zafixujete tuto konvenci, výrazně si zjednodušíte posuzování, do kterého balíku patří identifikátor, který se vyskytuje kdesi daleko ve zdrojovém textu.

10.2 Moduly

Balíky jsou jen prvním krokem na cestě k vícehlavým programátorským týmům. Umožňují, aby se programátoři nepoprali o identifikátory. To pravé ořechové však přinášejí až moduly.