Západočeská univerzita v Plzni

Fakulta pedagogická

Bakalářská práce

2014 **Vojtěch Marton**

ZÁPADOČESKÁ UNIVERZITA V PLZNI
FAKULTA PEDAGOGICKÁ
KATEDRA VÝPOČETNÍ A DIDAKTICKÉ TECHNIKY

Tvorba sady příkladů pro předmět Programování 2

BAKALÁŘSKÁ PRÁCE

Vojtěch Marton

Informatika se zaměřením na vzdělávání

Vedoucí práce: Mgr. Tomáš Přibáň, Ph.D.

Plzeň, 2014

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně s použitím uvedené literatury a zdrojů informací.

Plzeň, 23.prosince 2013

.....

vlastnoruční podpis

Děkuji vedoucímu bakalářské práce Mgr. Tomáši Přibáňovi za metodickou a odbornou pomoc a za další cenné rady při vytváření mé bakalářské práce.

Originál zadání práce .

Obsah

1 ÚVOD

Vývojové prostředí je program, který usnadňuje programátorům práci. Existuje jich celá řada a každé vývojové prostředí je většinou zaměřeno na jeden programovací jazyk. Tato bakalářská práce je zaměřena na vývojové prostředí Delphi od firmy Embarcadero, ve kterém se pracuje s objektově orientovaným programovacím jazykem Object Pascal.

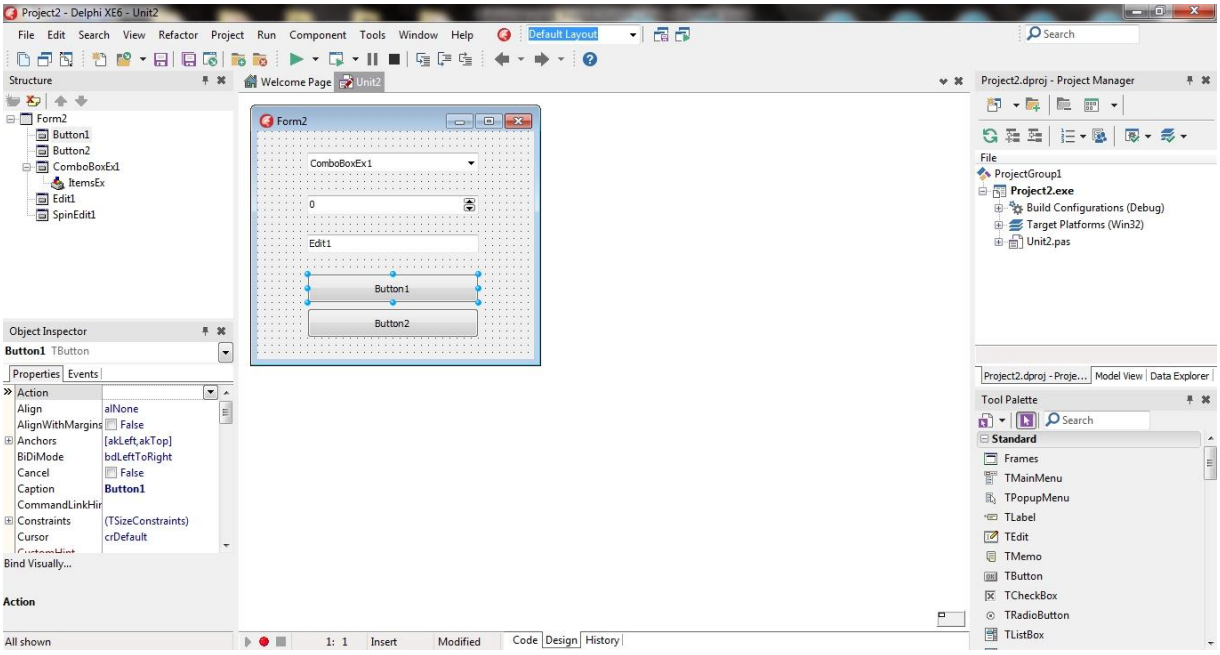
Hlavním cílem této bakalářské práce je seznámit studenty s prací v tomto programovacím prostředí, díky kterému lze vytvářet rychle a efektivně desktopové aplikace. Dále se zaměřuji na práci s komponenty a s principy objektově orientovaného programování s ukázkou vlastních příkladů.

Toto téma jsem si zvolil, protože si myslím, že vytváření aplikací v Embarcadero Delphi společně s programovacím jazykem Object Pascal, je dobrým způsobem, jak studenty seznámit s objektově orientovaným programováním. Dále si myslím, že programování v tomto vývojovém prostředí je velmi zajímavé, zvláště v nenovějších verzích Delphi, ve kterých lze vytvářet aplikace pro mnoho platforem.

Tato práce se nejprve zabývá vývojovým prostředím Embarcadero Delphi XE6, kdy se student seznámí s historií Delphi, se vzhledem prostředí a seznámí se s nástroji, které programátorovi umožňují pracovat například s komponenty, zdrojovým kódem nebo se soubory projektu. V druhé kapitole se student seznámí s principy objektově orientovaného programování a to se zapouzdřením, dědičností a polymorfizmem, kdy jsou představeny i ukázky zápisu kódu v jazyce Object Pascal. Třetí kapitola je pro tuto práci stěžejní. Obsahuje sadu příkladů, které jsou postaveny tak, aby se student seznámil s co nejvíce komponenty a jejich vlastnostmi a zároveň aby demonstrovaly principy objektově orientovaného programování.

2 PROGRAMOVACÍ PROSTŘEDÍ EMBARCADERO DELPHI A JEHO FUNKCE

Embarcadero Delphi je grafické vývojové prostředí, které je určeno pro tvorbu aplikací v programovacím jazyce Object Pascal. Pro tuto práci jsem si zvolil vývojové prostředí Embarcadero Delphi XE6. Pomocí tohoto vývojového prostředí lze vytvářet nativní aplikace pro operační systémy Windows, Mac, iOS a Android. Z těchto operačních systémů je tedy zřejmé, že lze vytvářet jak desktopové aplikace, tak aplikace pro mobilní zařízení jako jsou tablety, chytré telefon a dokonce i wearables. Dále lze vytvářet i mnoho dalších projektů jako například webové nebo konzolové aplikace. V Embarcadero Delphi XE6 je tedy mnoho možností, avšak tato práce se bude především zabývat klasickými desktopovými aplikacemi (VCL Form Aplication).



Obrázek 1 - Vývojové prostředí Embarcadero Delphi XE6

2.1 Historie a vývoj Embarcadero Delphi

Před vývojem programovacího prostředí Delphi, vydala firma Borland v roce 1983 programovací jazyk Turbo Pascal 1. Ten byl kompatibilní se

systémy DOS, CP/M-80 a CP/M-86. O rok později, v roce 1984, firma Borland vydala druhou verzi Turbo Pascal 2. Ta podporovala procesor Intel 8087. O tři roky později byly vydány další verze každý rok a to v roce 1986 Turbo Pascal 3, 1987 Turbo Pascal 4, 1988 Turbo Pascal 5, 1989 Turbo Pascal 5.5 (zde už se objevily objekty), 1990 Turbo Pascal 6, 1991 Turbo Pascal for Windows 1.0 (první verze pro 16 bitový Windows 3.0) a 1992 Turbo Pascal for Windows 1.5. V tomto roce byla vydána také verze pod názvem Turbo Borland Pascal 7, která byla jak 16 bitová tak 32 bitová. **(Delphi.cz, 2014)**.

14. února 1995 byla vydána první verze Delphi 1 od firmy Borland. Jednalo se o 16 bitovou verzi, která byla podporována systémem Windows 3. Již v této první verzi byla součástí VCL knihovna (Visual Component Library), ve které jsou uloženy komponenty. Dále podporovala databáze a to přes BDE (Borland Database Engine) a SQL Links. O rok později, v roce 1996, firma Borland vydala druhou verzi Delphi 2. Ta už byla 32 bitová a byla podporovaná operačním systém Windows 95. Dále podporovala Database Grid, která umožňovala uživatelům práci s velkým množstvím dat. V roce 1997 byla vydána třetí verze Delphi. Ta podporovala rozhraní včetně vícenásobné dědičnosti. Dále podporovala, například Code Insigth (automatické doplňování zdrojového kódu), Component Templates, Active Forms a TClientDataSet. 17. července 1998 byla vydána verze Delphi 4. Ta již byla podporována operačním systémem Windows 98. V této verzi se objevily například dynamická pole a přetěžování metod. V roce 1999 byla vydána pátá verze Delphi, která podporovala přetěžování metod, podporu XML, Desktop layouts, Language Translation a DBGo ADO (používá se k připojení MySQL serveru). O dva roky později, v roce 2001, byla vydána verze Delphi 6. V této verzi se poprvé objevil DbExpress (slouží pro připojení k databázi, jedná se o vylepšenou verzi Borland Database Engine – BDE). Dále se objevily novinky jako Structure Windows a SOAP Web Services. Další verze Delphi 7 se objevila v roce 2002. V této verzi se poprvé objevil vzhled pro Windows XP a Web Application Development.

V roce 2003 vyšla verze Delphi 8, která jako první podporovala technologii .NET. O rok později vyšla další verze pod názvem Delphi 2005, která obsahovala novinky jako Error Insight, Unit Testing, Data Explorer, Histori Tab a Multi-unit namespaces. V dalším roce vyšla verze Delphi XE6 – ta podporovala Operator Overloading, MySQL a Unicode v DbExpress. **(Delphi.cz, 2014; Slideshare.net, 2014)**.

Verze Delphi 2007 již vyšla pod firmou CodeGear, která je jednou z divizí firmy Embarcadero. Vyšla v roce 2006. Ta oficiálně přestala podporovat Windows 98 a začala podporovat Windows Vista. Dále podporovala Unicode databáze a bylo vydáno mnoho dalších rozšíření této verze. V roce 2008 vyšla verze Delphi 2009 (také od firmy CodeGear), která se stala první Unicode verzí. V roce 2009 vyšla verze Delphi 10. Ta již vyšla pod firmou Embarcadero. Tato verze podporovala operační systém Windows 7. Dále obsahovala kompilaci na pozadí, podporovala ovládání programu gesty například myší, Debuger Visualizers, Source Code Formatter, Direct2D canvas a další novinky. V září 2010 vyšla od firmy Embarcadero verze Delphi XE. Tato verze obsahovala vylepšený a zrychlený help, integraci SVN (systém pro správu zdrojových kódů), Regular Expressin Library a Indy WebBroker. V roce 2011 byla vydána verze Delphi XE2, která je 64 bitová a je podporována operačním systémem Mac OSX. Mezi nejzajímavější novinky této verze patří FireMonkey, což je multiplatformní uživatelské rozhraní (Win32, Win64, OSX, iOS, Android a Linux) vyvinutý firmou Embarcadero. Dále obsahuje Styly VCL, nativní zip RTL, vylepšený nástroj pro XML dokumentaci, FastReport atd. O rok později byla vydána další verze a to Delphi XE3. Zde nalezneme vylepšenou knihovnu FireMonkey FM2, vylepšenou podporu pro OSX a Metrostyly pro VCL. Jednou z nejnovějších verzí Delphi je Delphi XE4. Ta byla vydána v dubnu 2013. Podporuje Apple iOS a FireDac (přístup k databázi). Předposlední verzí je Delphi XE5. Firma Embarcadero ji vydala 19. září 2013. Tato verze podporuje Android, obsahuje vylepšení pro iOS, obsahuje kompletní verze FireDac a podporuje REST. Poslední verzí Delphi je Delphi

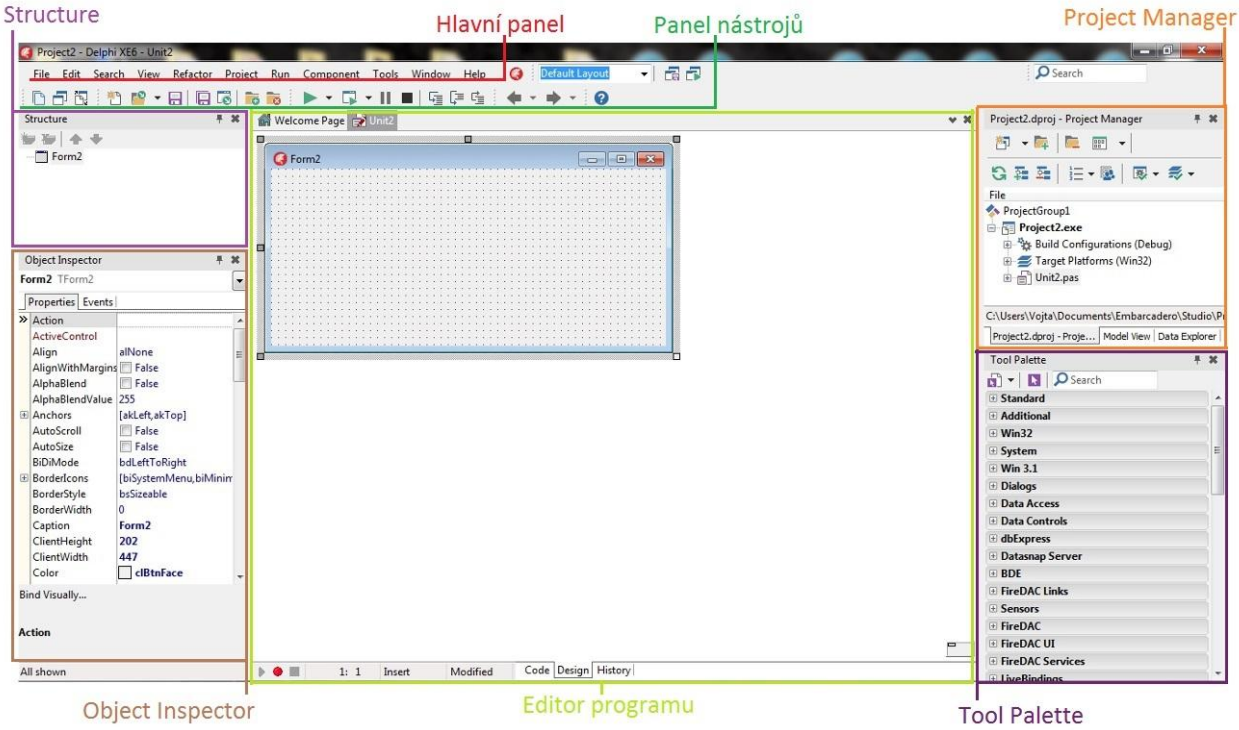
XE6. Ta byla vydána v dubnu 2014 a jedná se o vylepšenou verzi XE5, kdy bylo opraveno přes 2000 ohlášených chyb a zvýšil se celkový výkon aplikací na všech platformách. Mezi novinky této verze patří nové styly VCL, komponenty VCL pro čidla a rozšíření FireDAC, LiveBinding a databázových funkcí včetně FDMemTable. **(Delphi.cz, 2014; Slideshare.net, 2014)**

Firma Embarcadero pro vývoj aplikací nenabízí pouze vývojové prostředí Delphi, ale také C++ Builder. Rozdíl mezi Embarcadero Delphi a Embarcadero C++ Builder je jen v tom, že aplikace jsou psané v jazyce C++ a ne v jazyce Object Pascal. Dále firma Embarcadero nabízí RAD Studio a jedná se o nejúplnější sadu vývojářských nástrojů pro vytváření aplikací pro Windows, Mac, mobilní zařízení, PHP a pro web. Rad Studio tedy obsahuje Delphi, C++ Builder a HTML5 Builder, který je ve verzi Rad Studio XE6. **(embt.cz, 2014)**

2.2 Práce v programovacím prostředí Embarcadero Delphi XE6

Jak už bylo v úvodu této kapitoly zmíněno, pro tuto práci jsem si zvolil Embarcadero Delphi XE6, která je k dispozici v trial verzi na dobu třiceti dnů na webových stránkách firmy Embarcadero. Tuto verzi Delphi jsem si zvolil, protože se jedná o nejnovější verzi Delphi, která vyšla v dubnu 2014. Mezi hlavní přednosti této verze patří multiplatformnost. Lze tedy vytvářet aplikace pro operační systémy Windows, Mac a také pro mobilní zařízení se systémy Android a iOS. Bohužel zatím v Delphi nelze vytvářet aplikace pro operační systémy Linux. Jednou z novinek této verze je funkce LiveBinding, která programátorovi pomáhá rychle a efektivně propojit vlastnosti různých objektů. Dále tato verze získala nové komponenty a styly, takže aplikace mohou získat modernější vzhled.

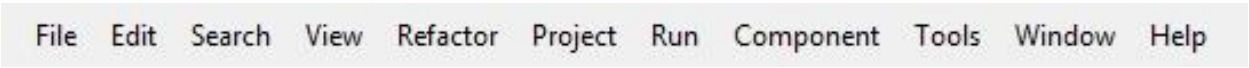
Vývojové prostředí Embarcadero Delphi XE6 se ve výchozím nastavení skládá ze sedmi hlavních částí. V horní části vývojového prostředí nalezneme hlavní panel a hned pod ním se nachází panel nástrojů. V pravé části nalezneme Tool Palette (paleta komponent) a Project Manager. V levé části se nachází Object Inspector (inspektor objektů) a nad ním nástroj Structure. Nakonec v prostřední části vývojového prostředí se nachází editor programu, ve kterém se nám otevírají formuláře a zdrojové kódy.



Obrázek 2 - Vzhled vývojového prostředí

2.2.1 Hlavní panel

Hlavní panel obsahuje základní nabídku, pomocí které je možné ovládat celé vývojové prostředí. (Písek 2002, s. 21).



Obrázek 3 - Hlavní panel

Nabídka **File** je obvyklou nabídkou ve všech programech ve Windows a obsahuje příkazy pro práci se soubory. Například otevírání, zavírání,

ukládání nebo tisk a také zde nalezneme příkaz pro zavření vývojového prostředí. Nabídka **Edit** je další obvyklou nabídkou a zde nalezneme příkazy, které využijeme při editaci zdrojového kódu. Nabídka **Search** obsahuje funkce pro lepší práci s dlouhým zdrojovým kódem. V nabídce **View** nastavujeme, které nástroje vývojového prostředí budou vidět a které nikoliv. **Project** slouží k manipulaci s otevřeným projektem. Nabídku **Run** využijeme ve chvíli, kdy budeme chtít spustit napsaný program. Pro spravování komponent slouží nabídka **Component**. Nabídka **Tools** slouží k individuálnímu přizpůsobení nástrojů a celého vývojového prostředí. Pro nápovědu lze využít nabídku **Help.** (Písek 2002, s. 22).

2.2.2 Panel nástrojů

Jak už bylo popsáno, panel nástrojů se nachází pod hlavním panelem. Panel nástrojů umožňuje přístup k často využívaným příkazům pomocí tlačítek. Ve vývojovém prostředí Embarcadero Delphi XE6 je k dispozici pět základních panelů, a to **Standard** (příkazy pro práci se soubory), **View** (pro zobrazení formuláře se kterým chceme pracovat), **Debug** (používáme pro ladění programu), **Desktop** (ukládá nastavení rozmístění oken na obrazovce) a **Custom** (panel pro vlastní modifikace). (Písek 2002, s. 22). Obrázek č. 4 ukazuje, jak vypadá panel nástrojů v Delphi XE6.

Kromě těchto pěti základních panelů, jsou v Delphi XE6 k dispozici i další panely nástrojů, jako například Align, Spacing, Position, Personality nebo Browse.

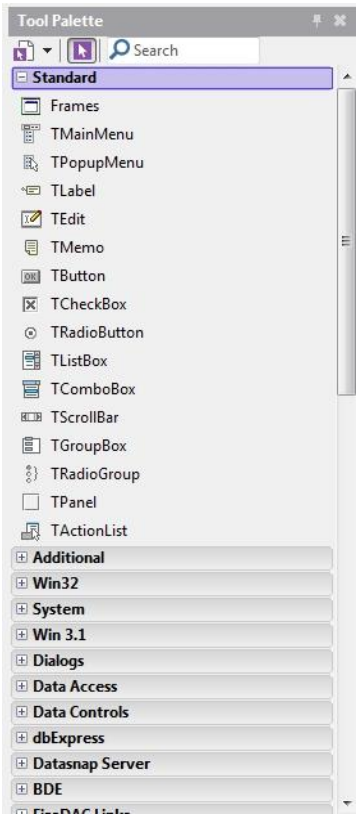
Obrázek 4 - Panel nástrojů

2.2.3 Tool Palette (paleta komponent)

Jedná se o zvláštní druh panelu nástrojů, kde nalezneme komponenty, které můžeme vkládat do svých programů. Komponenty jsou

na paletě komponent rozděleny do tematických skupin podle jejich účelu využívání. Počet komponent a tematických skupin se odvíjí i od verze Delphi a s každou novou verzí je paleta komponent rozšířena. (Písek 2002, s. 23).

V Delphi XE6 paleta komponent obsahuje čtyřicet sedm tematických skupin s komponenty. Mezi nejvýznamnější komponenty patří komponenty **Button** (tlačítko), **Edit** (textové pole), **Label** (popisek), **Memo** (textové okno), **CheckBox** (zaškrtávací okénko), **RadioButton** (přepínací tlačítko), **MainMenu** (hlavní panel programu), **PopupMenu** (vyskakovací menu) a **ComboBox** (rozbalovací seznam). Tyto komponenty nalezneme v kartě Standard. Významné komponenty nalezneme také v kartě Samples, a to **Gauge** (pro reprezentaci hodnot v procentech), **SpinButton** (tlačítko se šipkami) a **SpinEdit** (číselný vstup pomocí šipek). Další významné komponenty jsou **Timer** (časovač) z karty System, **ColorDialog** (dialog pro výběr barvy) z karty Dialogs a **TrackBar** (posuvník s měřítkem) z karty Win32.

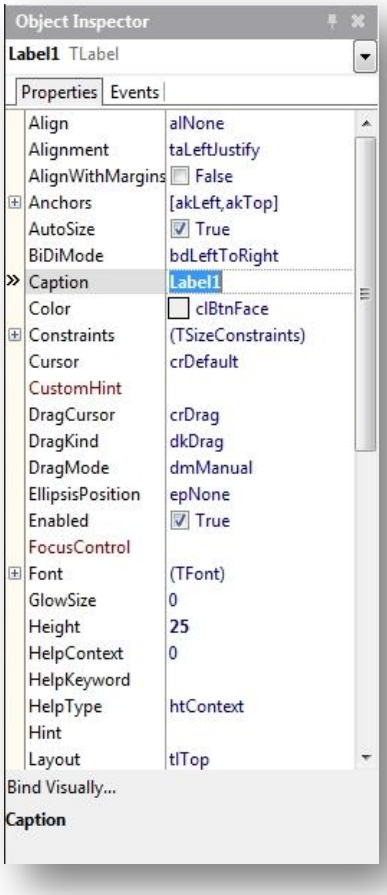


Obrázek 5 - Tool Palette

2.2.4 Object inspector

Object inspector, neboli inspektor objektů, je podle autora publikace Delphi začínáme programovat Slavoje Píska, snad ten nejpoužívanější nástroj ve vývojovém prostředí. Jak je na obrázku č. 6 dobře vidět, Inspektor objektů je rozdělen do tří částí, a to na rozbalovací seznam komponentů, které jsou umístěny v programu, kartu Properties (vlastnosti) a na kartu Events (události). Při navrhování programu pracujeme s Inspektorem objektů tak, že vybereme příslušnou komponentu a nastavíme jí vlastnosti (properties) a události (events). (Písek 2002, s. 29).

V tabulce č. 1 jsou představeny nejvýznamnější vlastnosti komponent a formulářů a v tabulce č. 2 jsou představeny nejvýznamnější události komponent a formulářů.



Obrázek 6 - Object inspector

Vlastnost	Význam	Vlastnost	Význam
Name	Jedinečný název	Top	Počet pixelů ze shora
Caption	Popisek komponenty	Left	Počet pixelů zleva
Text	Text v komponentě	Width	Šířka komponenty
Autosize	Komponenta se přizpůsobí textu	Height	Výška komponenty
WordWrap	Zalamování textu v komponentě	MinValue	Minimální číselná hodnota
PasswordChar	Maskování textu	MaxValue	Maximální číselná hodnota
Visible	Viditelnost komponenty	ReadOnly	Jen pro čtení
Enabled	Přístupnost komponenty	Font	Písmo
Color	Barva komponenty	Align	Zarovnání komponenty
Value	Číselná hodnota v komponentě	Transparen	Průhlednost

Tabulka 1 - Přehled nejvýznamnějších vlastností (Zdroj: vlastní zpracování dle Embarcadero Delphi XE6)

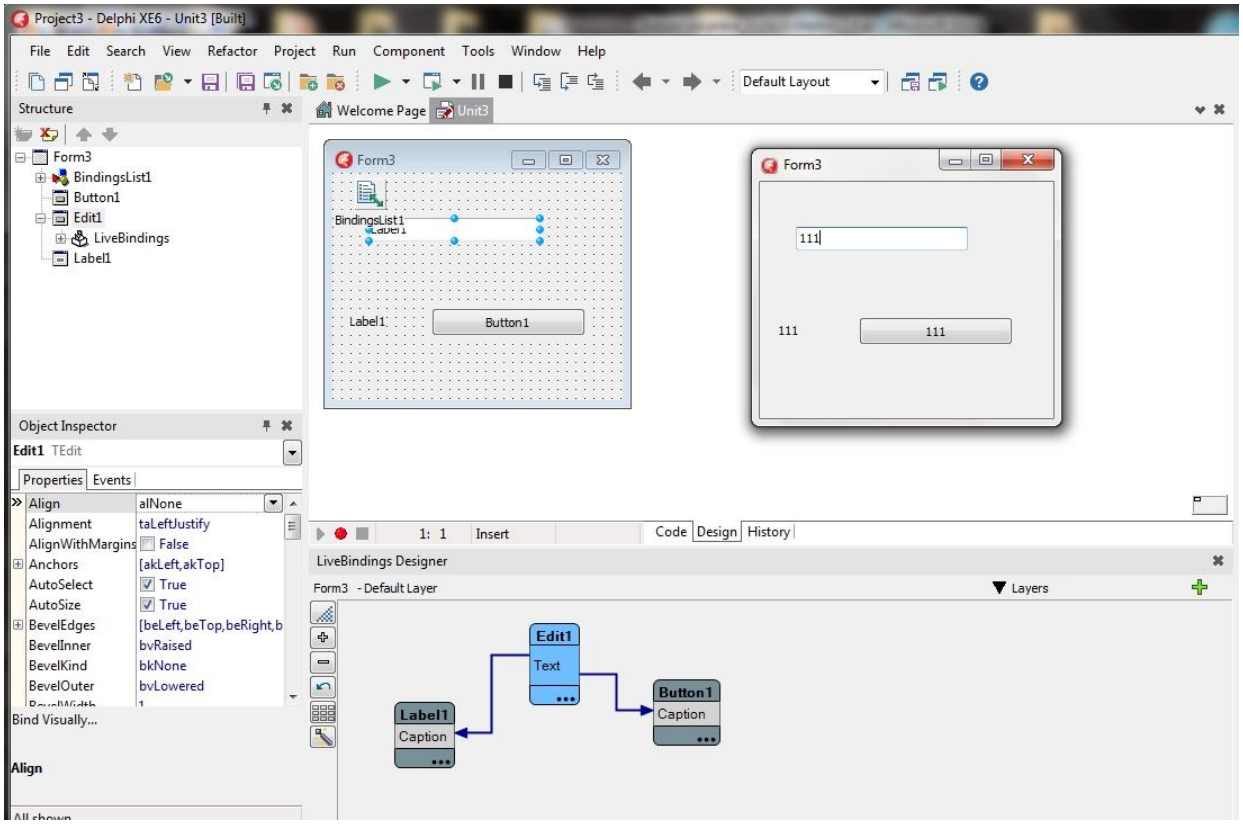
Událost form.	Význam	Událost komp.	Význam
OnCreate	Nastává při vytvoření formuláře	OnClick	Nastává při kliku levým tlačítkem myši.
OnClose	Nastává při zavření formuláře	OnDblClick	Nastává při dvojitém kliku levým tlačítkem myši
OnCloseQuery	Generuje se ještě před OnClose	OnMouseMove	Nastává při pohybu myši nad komp.
OnDestroy	Nastává při zrušení formuláře	OnMouseDown	Nastává při stisku tlačítka myší
OnActive	Nastává při aktivaci okna	OnMouseUp	Nastává při stisku tlačítka myši
OnDeactive	Nastává při deaktivaci okna	OnKeyDown	Nastává při stisku klávesy
OnHide	Nastává při skrytí formuláře	OnKeyPress	Nastává při stisku klávesy (jen ASCII)
OnShow	Nastává při zobrazení formuláře	OnKeyUp	Nastává při uvolnění klávesy
		OnEnter	Nastává při aktivaci komponenty
		OnExit	Nastává při deaktivování komponenty

Tabulka 2 - Nejvýznamnější události formulářů a komponent (Zdroj: vlastní zpracování dle Embarcadero Delphi XE6)

2.2.5 LiveBindings

Jak už bylo výše zmíněno, LiveBindings, neboli živé vazby, je nová funce Delphi XE6 a v předešlých verzích ji tedy nenajdeme. Jak je zřejmé z obrázku č. 6, tuto funkci nalezneme ve spodní části nástroje Object Inspector jako Bind Visually. Tato funkce umožňuje propojit vlastnosti

objektu s vlastnostmi jiných objektů. To nám ulehčí a hlavně urychlí práci. Na obrázku č. 7 je představen jednoduchý příklad, jak LiveBindings funguje.



Obrázek 7 - LiveBindings

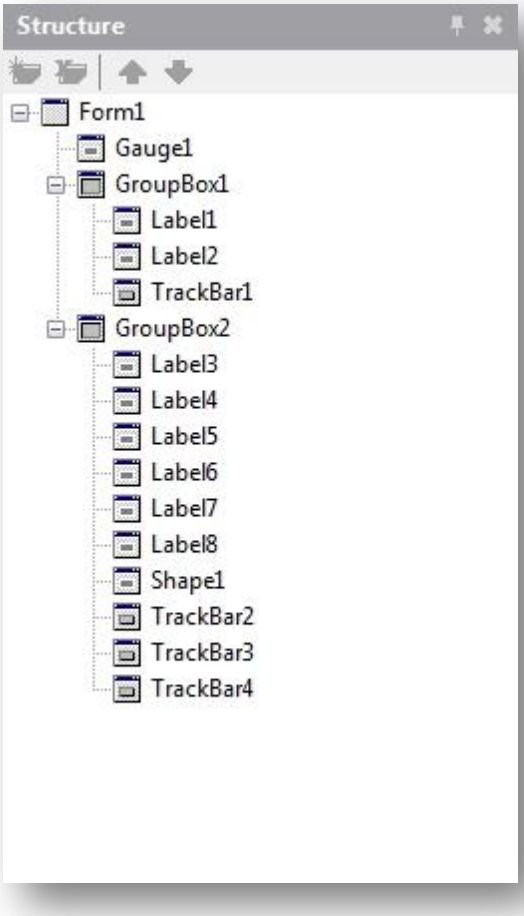
Na obrázku č. 7 je vidět, že na formuláři máme tři komponenty, a to Edit, Label a Button. Ve spodní části obrázku se nachází LiveBinding Designer, kde vidíme jednotlivé objekty a jejich vlastnosti. Na tomto příkladu byla vlastnost Text z objektu Edit1 propojena s vlastností Caption objektu Label1 a s vlastností Caption objektu Button1. To způsobí, že když za běhu programu budeme měnit vlastnost text objektu Edit1, změní se následně vlastnosti Caption objektů Label1 a Button1.

2.2.6 Structure (struktura)

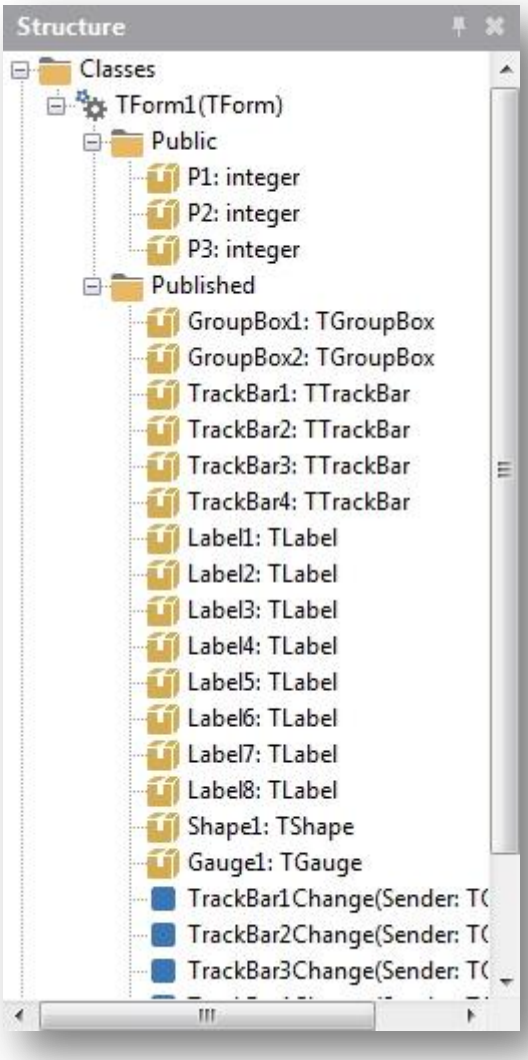
Pokud tvoříme formulář, okno Structure nám slouží pro zobrazení hierarchické struktury všech objektů, které jsou umístěny na aktuálním

formuláři. Díky tomu lze snadno a pohodlně vybírat libovolný objekt, se kterým chceme pracovat. (Písek 2002, s. 29).

Pokud ale tvoříme vlastní zdrojový kód, tak okno Structure nám slouží jako průzkumník kódu, který usnadňuje orientaci ve zdrojovém kódu. (Písek 2002, s. 28)



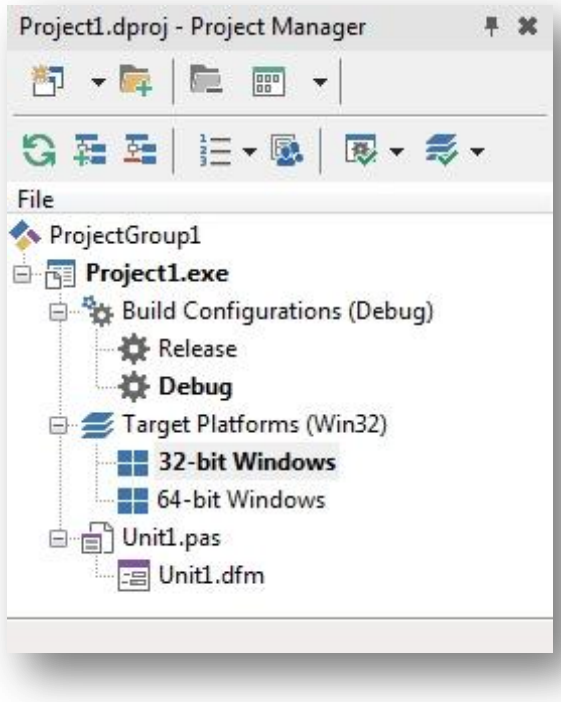
Obrázek 8 - Struktura při návrhu vzhledu



Obrázek 9 - Struktura při editaci zdrojového kódu

2.2.7 Project Manager

Project Manager je nástroj, který zobrazuje obsah aktuálního projektu. Jak je vidět na obrázku č. 10, tento obsah je reprezentován stromovou strukturou, kde jsou jednotlivé uzly a položky označeny ikonami a popiskem. Po kliknutí pravým tlačítkem myši na položku nebo uzel se nám zobrazí příkazy, které položka nebo uzel nabízí. Prvním uzlem ve stromové struktuře je Project Group, pod kterým jsou všechny projekty. Umožňuje například přidávat a vytvářet nové projekty nebo ukládat projektové skupiny. Dále následují uzly projektů, typicky to jsou soubory s příponou .exe. V těchto uzlech lze projekty kompilovat, spouštět, spouštět bez debuggru, ukládat, přejmenovávat, odstraňovat soubory generované z projektu (strojové kódy), odstraňovat soubory z projektu atd. V uzlech projektů

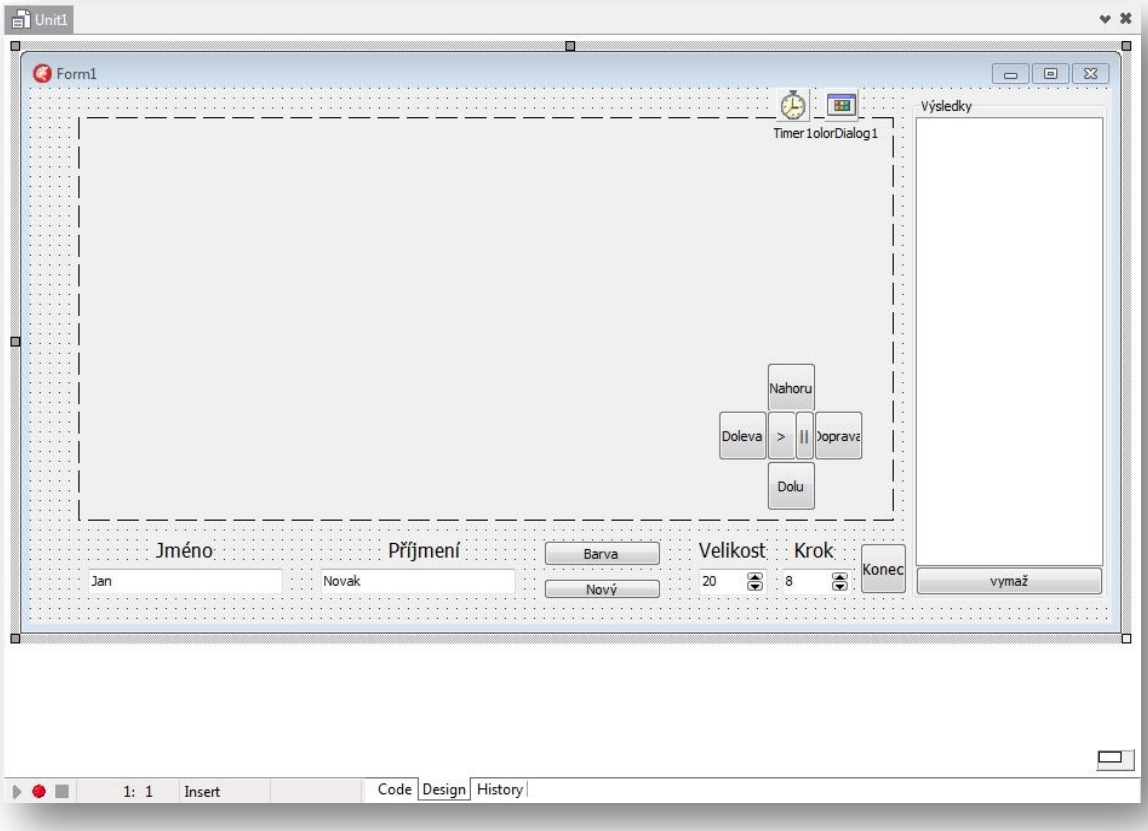


Obrázek 10 - Project Manager

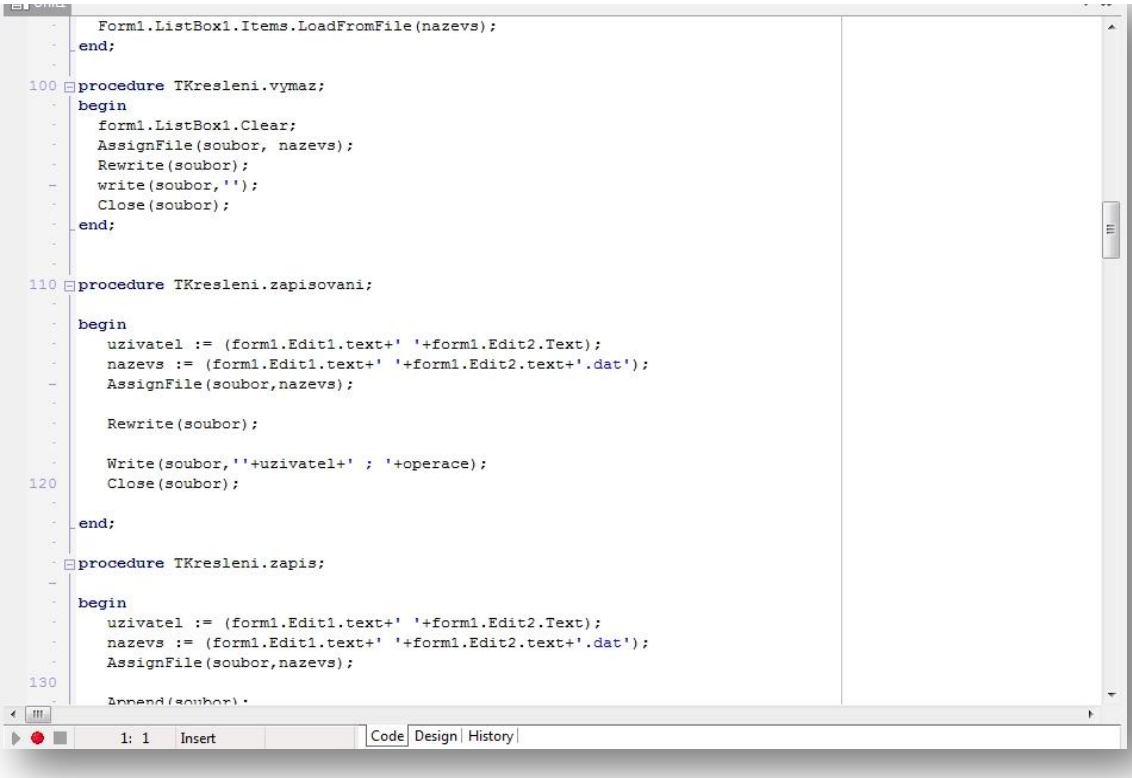
nalezneme další uzly a to Build Configurations (vytváří a zobrazuje konfigurace), Target Platforms (volíme cílovou platformu) a uzly Unit (soubory zdrojových kódů s příponou .pas).

2.2.8 Editor programu

Editor programu nalezneme uprostřed vývojového prostředí, a to mezi paletou komponent a inspektorem objektů. V této části vytváříme samotný program. Editor programu je rozdělen na tři části a to na Code, Design a History. V části Design vytváříme vzhled, a to pomocí formulářů a komponent. V části Code najdeme samotný zdrojový kód programu. V poslední části, v části History nalezneme provedené změny.



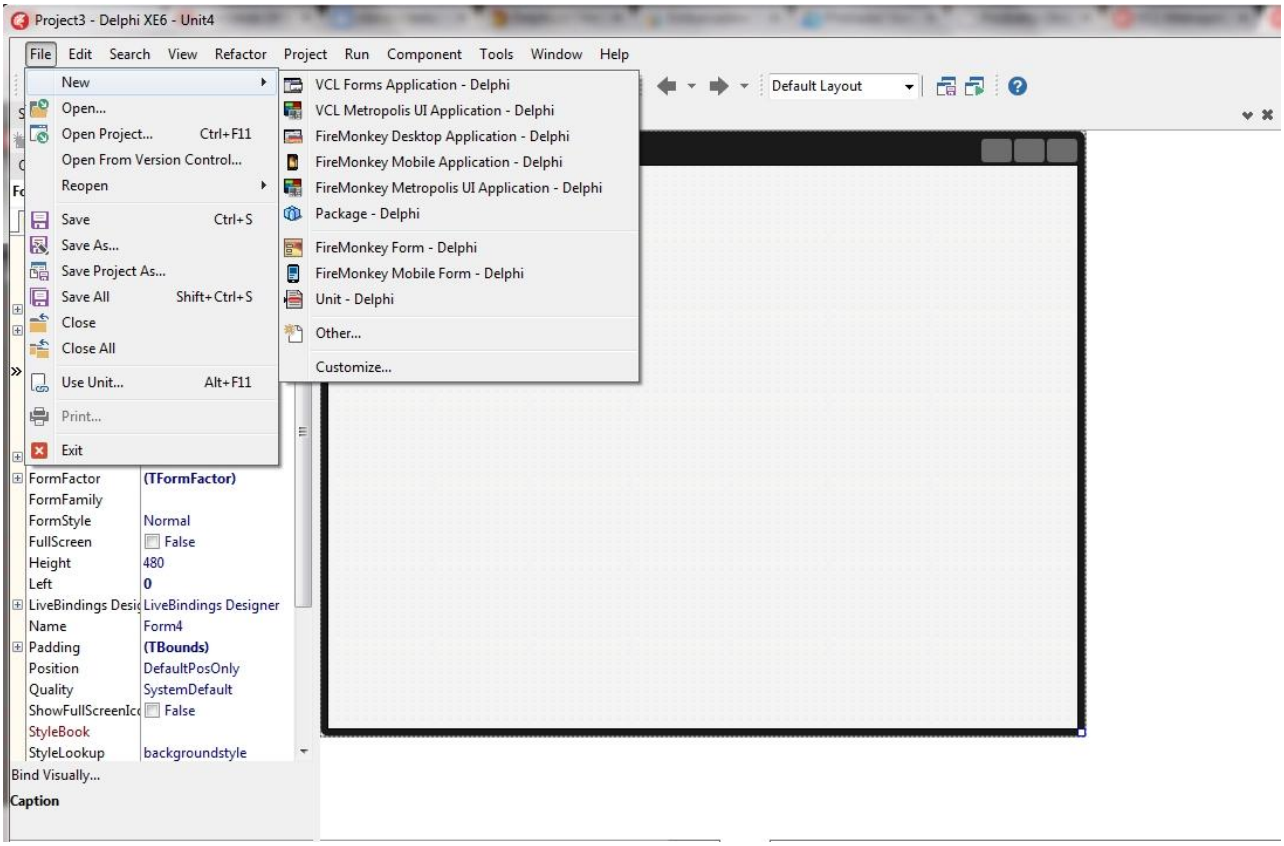
Obrázek 9 - Editor programu při návrhu vzhledu



Obrázek 11 - Editor programu při editaci zdrojového kódu

2.2.9 Vytvoření projektu

V Delphi XE6 lze vytvořit hned několik druhů projektů. Jak je zřejmé z obrázku č. 11, všechny projekty vytváříme příkazem **File → New → druh projektu**.



Obrázek 12 - Editor programu při editaci zdrojového kódu

První z projektů je VCL Forms Application. Tento projekt je pro tuto práci stěžejní, protože s tímto druhem projektu budeme dále pracovat. Jedná se o projekt, který se skládá alespoň z jednoho formuláře a jedné programové jednotky neboli Unity. (Písek 2002, s. 32). Tento druh projektu nalezneme i v předešlých verzích Delphi, na rozdíl od moderně stylizovaných projektů jako jsou VCL Metropolis UI Application nebo FireMonkey Desktop Application. Projekt VCL Forms Application lze vytvářet jak pro 32 bitový Windows, tak pro 64 bitový Windows.

Jak je na obrázku č. 11 vidět, další z projektů je VCL Metropolis UI Application. Tento projekt vytváří rámec, který obsahuje stylizované prvky, které jsou kompatibilní s uživatelským rozhraním Windows 8. V tomto projektu máme na výběr ze tří šablon, a to Blank Metropolis UI Application (prázdná aplikace), Grid Metropolis UI Application (obsahuje několik položek, z nichž každá otevře stránku DetailView), Split Pane Metropilis UI Application (obsahuje menší počet položek jako u předchozí šablony a každá položka se otevírá ve SplitPaneView). U tohoto projektu si lze zvolit platformu, a to buď 32bitový Windows, nebo 64bitový Windows.

Dále můžeme vytvořit projekt FireMonkey Desktop Application, který vytváří formulář podobný formuláři ve VCL Forms Application. Před vytvořením projektu vybíráme ze dvou možností, a to HD FireMonkey Application a 3D FireMokey Application. U těchto aplikací můžeme nastavit tři cílové platformy, a to 32bitový Windows, 64bitový Windows a OS X (MAC).

V posledních verzích Delphi se objevil projekt FireMonkey Mobile Application. Jedná se o aplikace pro mobilní zařízení se systémy Android a iOS. V tomto druhu projektu se nám neotevírá klasický formulář, ale vzhled mobilního zařízení, pro které chceme aplikaci tvořit.

Pokud vyžadujeme modernější vzhled naší aplikace, je tu k dispozici projekt FireMonkey Metropolis UI Application. Ten vyváří rámec pro aplikace FireMonkey ve stylu Metropolis UI a obsahuje stylizované prvky, které jsou kompatibilní s Metropolis UI.

Dále lze vytvořit projekt Package (slouží pro vytváření balíčků), VCL Form (klasický formulář), FireMonkey Form (FireMonkey formulář), FireMonkey Mobile Form (formulář pro mobilní zařízení) a Unit (jen samotná programová jednotka).

Příkazem **File → New → Other** otevřeme okno New Items, kde najdeme další možné aplikace, které lze v Delphi XE6 tvořit, jako například

konzolové aplikace (Console Application), servisní aplikace (Service Application), anebo MDI aplikace (MDI Application).

Pro otevření existujícího projektu, použijeme příkaz **File → Open Project** nebo zkratku Ctrl + F11 a pro rychlé otevření projektu, který byl otevřený nedávno, použijeme příkaz **File → Reopen**. (Písek 2002, s. 32).

2.2.10 Soubory projektu

Typická aplikace vytvořená v Delphi, se skládá z mnoha různých typů souborů. Prvním z nich je soubor s příponou **.dpr** (Delphi project). Tento soubor je hlavním souborem projektu (jeden soubor na jeden projekt). Soubor s příponu dpr slouží jako vstupní bod pro spustitelný soubor, protože obsahuje odkazy na jiné soubory v projektu a spojuje formuláře s příslušnými unitami. Tento soubor bychom neměli editovat ani odstraňovat. Může se objevit i soubor s příponou **.dproj** (Delphi project), což je také soubor projektu.

Dalším důležitým souborem je soubor s příponou **.pas** (Delphi source file). Ten obsahuje zdrojový kód aplikace, a právě tento soubor se nám otevírá v editoru programu v kartě Code.Dále se vytváří soubory s příponou **.dfm** (Delphi form), které obsahují informace (vlastnosti) objektů, které jsou obsaženy ve formuláři. Tyto soubory jsou vždy spárovány se soubory .pas.

Ve složce se soubory nalezneme i soubor s příponou **.res** (Windows resource file). Tento soubor je potřebný pro kompilaci a Delphi ho generuje automaticky. Obsahuje informace v binární podobě.Dále se vytvoří složka podle názvu platformy, pro kterou jsme aplikace vytvořili (např. Win32 nebo Win64). V této složce nalezneme další složku pod názvem Debug a v ní nalezneme soubory s příponou **.exe** (Application Executable) a **.dcu** (Delphi compiled unit). Soubory s příponou **.exe** jsou spustitelné soubory a vytváří se hned po kompilaci a spuštění projektu. Soubory s příponou **.dcu** jsou binární soubory, které vznikají po kompilaci souboru .pas.

3 ZÁKLADNÍ VLASTNOSTI A PRINCIPY OBJEKTOVĚ ORIENTOVANÉHO PROGRAMOVÁNÍ V JAZYCE OBJECT PASCAL

Jak už bylo v předešlé kapitole zmíněno, v Delphi se pracuje v programovacím jazyce Object Pascal. Jak je z názvu patrné, jedná se o objektově orientovaný programovací jazyk.

3.1 Základní filozofie objektově orientovaného programování (OPP)

Program, který je napsán podle pravidel OOP, je složen ze skupin objektů. Tyto objekty obsahují určitá data a definovaná rozhraní, díky kterým spolu vzájemně objekty komunikují a manipulují s daty. Pro práci v OOP je třeba znát tři hlavní rysy OOP, a to zapouzdření, dědičnost a polymorfizmus. (Písek 2002, s. 89-90). Tyto tři rysy budou dále podrobněji popsány.

3.2 Objekty

Pro práci v objektově orientovaném programování je třeba správně pochopit, co je to objekt. Objekty si lze představit jako věci, co jsou okolo nás, jako například dům, auto, letadlo apod. Všechny tyto věci, neboli objekty, mají nějaké atributy (vlastnosti) a chování. Například u objektu auto bychom mohli říct, že jeho atributy jsou barva, výška, šířka, váha, motor, počet dveří a spousta dalších atributů. Mezi jeho chování patří například zastavení, změna směru, změna rychlosti a mnoho dalších činností. Atributy a chování skutečného objektu následně programátor převede pomocí objektově orientovaného jazyka do třídy, která se skládá z atributů (vlastnosti objektu) a metod (procedury a funkce, které popisují chování objektu). (Keogh 2006, s. 13-21).

3.3 Třídy

Třída je v podstatě šablona, ve které jsou deklarované atributy a metody skutečných objektů. Nejedná se tedy o objekt, třídy pouze popisují atributy a chování objektů. Z tříd následně vytváříme objekty, které jsou datového typu třída. Takovémuto objektu se říká instance třídy. (Keogh 2006, s. 22-28).

V Object pascalu se třída deklaruje klíčovým slovem class. Je nepsaným pravidlem, že název třídy začíná velkým písmenem T, je tedy dobré toto pravidlo dodržovat. (Písek 2002, s. 90). Následující ukázka kódu znázorňuje, jak lze v Delphi nadefinovat třídu Kalendář, která má vlastnosti den, měsíc a rok (které jsou typu integer) a dále obsahuje procedury pro zobrazení data a pro nastavení data.

```
type
TKalendar = class
        den:integer;
        mesic:integer;
        rok:integer;
        procedure zobrazDatum;
        procedure nastavDatum(dd, mm, rrrr :integer);

end;
```

Jak už bylo výše zmíněno, aby bylo možno pracovat se třídou, musíme vytvořit instanci třídy. Příklad vytvoření instance třídy znázorňuje následují úryvek kódu.

```
var
nastenyKalendar : TKalendar;
stolniKalendar : TKalendar;
```

K atributům a metodám jednotlivých instancí tříd přistupujeme přes tečkovou notaci.

```
…..
nastenyKalendar.den:= 21;
stolniKalendar.nastavDatum(21,11,2015);
…...
```

Mezi proměnou datového typu třída a proměnnou datových typů jako integer, byte, záznam apod. je zásadní rozdíl. Zatímco u datových typů jako například integer je možné proměnnou hned po deklaraci používat, v případě instance třídy se vytvoří jen odkaz do paměti, kde je objekt uložen. Je tedy nutné před prvním použitím objektů instance třídy objekty vytvořit. K tomu nám slouží konstruktory. Konstruktor je speciální metoda, která se deklaruje stejně jako běžná metoda, jen s použitím klíčového slova constructor. Konstruktor vytvoří v paměti objekty dané třídy, a poté provede všechny naprogramované příkazy. (Písek 2002, s. 91). Následující úryvek kódu ukazuje předešlou deklaraci třídy TKalendar doplněnou o konstruktoru Konstrukt.

```
type
TKalendar = class
        den:integer;
        mesic:integer;
        rok:integer;
        procedure zobrazDatum;
        procedure nastavDatum(dd, mm, rrrr :integer);
        constructor Konstrukt(dd, mm, rrrr:integer);

end;
```

V případě, kdy není nutná inicializace objektů, nemusíme vytvářet vlastní konstruktor, ale můžeme využít konstruktoru **Create**, který se pro každou třídu vytváří automaticky. (Písek 2002, s. 92).

„Pokud tedy chceme začít pracovat s nějakou třídou, tak prvním příkazem přiřadíme referenci konkrétní instanci třídy vytvořenou konstruktorem.“ (Písek 2002, s. 92). To uděláme například takto:

```
nastenyKalendar := TKalendar.create;
stolniKalendar := TKalendar.Konstrukt(21,11,2015);
```

Na konci programu je nutné každý vytvořený objekt zrušit. K tomuto účelu slouží destruktory, které se vytvářejí pomocí klíčového slova destructor. Každá vytvořená třída automaticky obsahuje destruktor s názvem Destroy. Ten nevoláme přímo, ale prostřednictvím metody Free.

3.4 Zapouzdření

Zapouzdření je jednou z důležitých vlastností tříd. Zapouzdření znamená, že všechny objekty třídy jsou uzavřeny uvnitř třídy a mezi jednotlivými instancemi jsou na sobě nezávislé. (Písek 2002, s. 92). V předchozí podkapitole byly deklarovány instance třídy nastenyKalendar a stolniKalendar. Tyto instance mají stejné vlastnosti - den, mesic, rok, které jsou typu integer, procedury zobrazDatum a nastavDatum, které popisují jejich chování a konstruktor Konstrukt. Zapouzdření zajišťuje to, že vlastnost den z instance nastenyKalendar, je naprosto nezávislá od vlastnosti den z instance stolniKalendar.

Zapouzdření je tedy dobrý způsob, jak dostat atributy a metody pod jednu „střechu“, ale to není jediná funkce zapouzdření. Hlavním důvodem pro používání zapouzdření je prevence vzniku chyb, které vznikaly při nesprávném používání atributů a metod. V podstatě zapouzdření umožňuje programátorovi umístit do tříd atributy a metody a stanovit pravidla, která slouží ke kontrole přístupu. (Keogh 2006, s. 35-37).

Z těchto bezpečnostních důvodů mohou být vlastnosti a metody tříd zařazeny do jedné ze čtyř skupin, a to do Public, Private, Published a Protected. Objekty ve skupině Public budou veřejné a zajišťují komunikaci

třídy s jejím okolím. Skupina Private obsahuje objekty třídy, které jsou soukromé a mohou se používat v rámci jedné jednotky. Pro ostatní jednotky jsou nepřístupné. Objekty třídy ve skupině Published se chovají obdobně jako objekty třídy ve skupině Public, jen s tím rozdílem, že navíc obsahují informace o běhu programu. Poslední skupinou je skupina Protected. Objekty v této skupině jsou chráněné a chovají se podobně jako objekty soukromé, jen s rozdílem, že nejsou omezeny jen na třídu, ve které byly vytvořeny. Objekty třídy můžou používat i třídy zděděné bez ohledu na to, v jaké jednotce jsou uloženy. (Písek 2002, s. 97-99).

3.5 Dědičnost

Dědičnost je další důležitou vlastností objektově orientovaného programování, protože představuje způsob, kterým objekt získá atributy a chování jiného objektu díky vztahu, které se říká relace. Dědičnost se dá ukázat například na objektech Člověk a Student, protože když Člověk bude mít atributy Jméno, Příjmení a Adresa a chování Sezení, Stání a Chození, tak tyto atributy a chování využijeme i u objektu Student (protože Student je taky člověkem). Navíc ale Student bude potřebovat další atributy a chování jako Studijní číslo, Obor, Docházení do školy nebo Psaní testu. Student tedy zdědí všechny atributy a chování od objektu Člověk, které může libovolně používat a navíc bude mít své atributy. (Keogh 2006, s. 15-16).

V praxi dědičnost tedy využijeme ve chvíli, kdy máme již definovanou třídu a potřebujeme nadefinovat novou, která je až na několik rozšíření stejná jako ta první. Je tedy zbytečné definovat celou novou třídu od znova. Při definování nové třídy napíšeme za klíčové slovo class, do závorky název třídy, od které chceme dědit, a tato třída zdědí všechny objekty z rodičovské třídy. V těle potomka jen stačí definovat rozšiřující objekty. Na následující ukázce bude představena dědičnost, a to rodičem TKalendar a potomkem TPripominka, která bude dědit atributy a chování ze třídy TKalendar a bude obsahovat vlastní atributy a chování pro připomínání událostí.

Zde je definovaná rodičovská třída TKalendar.

```
type
TKalendar = class
        den:integer;
        mesic:integer;
        rok:integer;
        procedure zobrazDatum;
        procedure nastavDatum(dd, mm, rrrr :integer);
end;
```

Zde je definovaný potomek třídy TKalendar třída TPripominka.

```
type
TPripominka = class (TKalendar)
        pripomenout : boolean;
        procedure nastavPripomenuti(pripomenout : boolean);
end;
```

Takto definovaná třída zdědila proměnné den, mesic, rok a metody zobrazDatum a nastavDatum. Dále tato třída obsahuje novou proměnnou pripomenout (která je typu boolean) a proceduru nastavPripomenuti.

3.6 Polymorfismus

Posledním hlavním rysem objektově orientovaného programování je Polymorfismus neboli mnohotvarost. Jde o nejvyšší stupeň OOP. Polymorfismus představuje vlastnost metod, kdy se při stejném volání provádí různé kódy. (Písek 2002, s. 90-106).

3.6.1 Překrývání metod

Překrývání metod je úzce spojeno s dědičností. V kapitole dědičnost bylo psáno, že potomek zdědí od rodičovské třídy všechny objekty. Pokud potomek zdědí proceduru zobrazDatum a tuto metodu přepíšeme, původní

metoda se odstraní a počítá se jen s nově nadefinovanou metodou zobrazDatum v potomkovi. To bylo dáno statickými metodami a statickou vazbou. Je tedy předem jasné, že pokud voláme proceduru zobrazDatum v potomkovi, myslíme právě tu nově nadefinovanou proceduru zobrazDatum. (Písek 2002, s. 103-104).

Pokud bychom v potomkovi chtěli využít zděděnou metodu zobrazDatum a také nově nadefinovanou metodu zobrazDatum, je potřeba použít virtuální metody s pozdní vazbou. Pak je tedy potřeba při deklaraci procedury zobrazDatum v rodičovské třídě označit tuto metodu jako virtuální klíčovým slovem **virtual** a novou proceduru zobrazDatum v potomkovi označit klíčovým slovem **override**. To bude mít za následek, že zděděná procedura bude pouze potlačena (nikoliv odstraněna jako to bylo v případě použití statické metody se statickou vazbou). Použitím virtuální metody programu říkáme, že není dopředu jasné, zda se bude volat metoda z rodiče nebo z potomka. To se vyhodnocuje až za běhu programu podle objektu, který „volá“ proceduru zobrazDatum, a právě tento přístup představuje polymorfismus. (Písek 2002, s. 103-104). V následující ukázce kódu je definice rodičovské třídy TKalendar a potomka TPripominka s upravenou metodou zobrazDatum na virtuální metodu. Budou představeny jen změny.

```
 type
TKalendar = class
        …….
        procedure zobrazDatum;virtual;
        …….
end;

type
TPripominka = class (TKalendar)
…….
procedure zobrazDatum;override;
```

```
…….
end;
```

Virtuální metody mají i své nevýhody a to především rychlost zpracování. Je to dáno tím, že se zjišťují odkazy na třídy jejich zpracování, a to výrazně snižuje rychlost programu. (Písek 2002, s. 104).

Kromě virtuálních metod můžeme využít i dynamické metody. Ty využívají také pozdní vazbu a pracují stejně jako virtuální metody. Definice dynamických metod je také stejná jen místo klíčového slova virtual píšeme dynamic. Rozdíl mezi virtuálními a dynamickými metodami je v jejich vnitřní prezentaci. U dynamických metod je kladen důraz na velikost kódu, a proto jsou pomalejší než virtuální metody. Výhoda těchto metod je taková, že mohou ušetřit značné množství paměti při složitých objektových hierarchiích. (Písek 2002, s. 104-105).

V Delphi se nachází ještě jeden typ metod, a to abstraktní metody. Pokud nějaká třída obsahuje abstraktní metodu, tak jí sama nevyužívá a předpokládá se, že tuto metodu budou využívat až potomci této třídy. Abstraktní metody deklarujeme klíčovým slovem **abstract**. (Písek 2002, s. 105-106). Následující ukázka kódu představuje deklaraci abstraktní metody nastavPripomenuti v rodiči a následnou deklaraci metody nastavPripomenuti v potomkovi.

```
type
TKalendar = class
        …….
        procedure nastavPripomenuti;virtual;abstract;
        …….
end;
```

```
type
TPripominka = class (TKalendar)
    …….
    procedure nastavPripomenuti;override;

    …….
    end;
```

Takto zděděná metoda se používá stejně jako všechny ostatní metody, ale je důležité vědět, že se v žádném případě nesmí volat abstraktní metoda přímo v rodiči. To způsobí chybu a program bez varování ukončí svou činnost. (Písek 2002, s. 105-106).

3.6.2 Přetěžování metod

Delphi podporuje i přetěžování metod. To umožňuje, aby v jedné třídě bylo definováno více metod se stejným názvem, avšak tyto metody musí mít odlišné parametry. Programátor si tedy nemusí pamatovat více názvů metod, ale jen jeden. Metody přetěžujeme klíčovým slovem **overload**. Na následující ukázce kódu je uveden příklad přetěžování metod pretizenaMetoda.

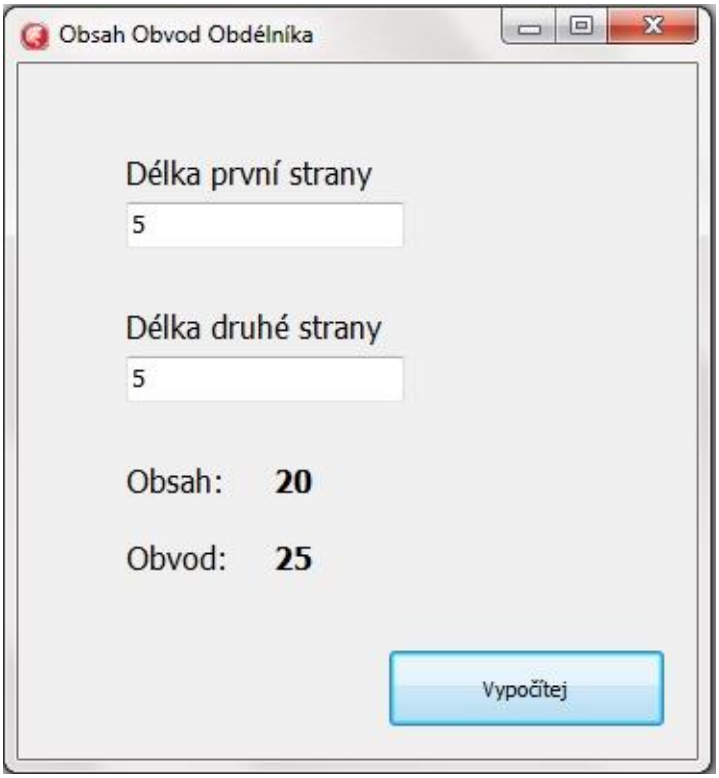
```
type
TUkazkaPretizeni = class

        procedure pretizenaMetoda; overload;
        procedure pretizenaMetoda (a, b:integer); overload;
        procedure pretizenaMetoda (a, b :double); overload;
        procedure pretizenaMetoda (a, b :String); overload;


        end;
```

Když voláme přetíženou metodu, program se rozhoduje, kterou z metod provede. To posuzuje podle parametrů, které jsme do volané metody napsali.

4 VLASTNÍ SADA PRAKTICKÝCH PŘÍKLADŮ DLE ZÁSAD OPP, VHODNÝCH PRO VÝUKU PŘEDMĚTU

4.1 Příklad 1

Vytvořte program, který vypočítá obvod a obsah obdélníka.



Obrázek 13 - Obvod obsah obdélníka

Tento jednoduchý příklad je zaměřený na seznámení se s vývojovým prostředím Delphi a na práci v něm. Tento příklad je především zaměřený na komponenty a jejich vlastnosti a události. Student se v tomto příkladu naučí vytvářet vlastní projekt, vkládat komponenty do formuláře a pracovat se základními vlastnostmi a událostmi komponent Edit, Label a Button.

Jak už bylo v první kapitole popsáno, nový projekt vytvoříme příkazem **File → New → VCL Forms Application – Delphi**. Tím jsme si vytvořili nový projekt, jehož základem je formulář. Z Tool Palete si do

formuláře přetáhneme dvě komponenty Edit pro vstupy do programu, komponentu Button a pro popisky šest komponent Label. Obsah komponent Edit změníme ve vlastnosti Text, popisky komponent Label a Button změníme ve vlastnosti Caption. Jak je na obrázku 13 zřejmé, popisky mají větší nebo tučnější písmo než je ve výchozím nastavení. Kterékoliv úpravy písma se řeší ve vlastnosti Font. Každá komponenta má vlastnost, která danou komponentu jednoznačně identifikuje. Tato vlastnost se jmenuje Name, která se automaticky nastaví podle typu komponenty. Například vložíme-li do formuláře komponentu Button, vlastnost Name této komponenty se nastaví na Button1. Po přidání dalšího tlačítka do formuláře se vlastnost Name automaticky nastaví na Button2, Button3 atd. Pokud ve formuláři budeme mít velké množství komponent, tak může nastat problém v jejich názvech. Ten problém je takový, že při velkém množství komponent se nám můžou vyskytovat komponenty s názvem Label23, Button9, RadioButton10 apod. a ztrácíme orientaci v kódu. Z tohoto důvodu je dobré vlastnost Name měnit podle svého. Po označení komponenty Button klikneme v Object Inspectoru na kartě Events na událost OnClick (nebo dvakrát klikneme na komponentu Button). Tímto se nám vytvoří prázdná procedura, jejíž obsah bude obsluhovat událost OnClick. V tomto jednoduchém příkladě nejsou řešeny vlastní třídy metody apod., takže to co se má provést při kliknutí myší na komponentu Button napíšeme rovnou do procedury TForm4.Button1Click, což je patrné na obrázku 14.

```
procedure TForm4.Button1Click(Sender: TObject);
var stranaA, stranaB, obsah, obvod :integer; //deklarace lokálních proměnných
begin

stranaA:=StrToInt(Form4.Edit1.Text);     //do proměnné stranaA přiřadíme obsah z vlastnosti
stranaB:=StrToInt(Form4.Edit2.Text);     //do proměnné stranaB přiřadíme obsah z vlastnosti

obsah:=stranaA*stranaB;     //do lokální proměnné obsah přiřadíme výsledek obsahu
obvod:=2*stranaA + 2*stranaB;  //do lokální proměnné obvod přiřadíme výsledek obvodu

Form4.Label5.Caption:=IntToStr(obvod);     //do Labelu pro výsledky obvodu vložíme výsledek
Form4.Label6.Caption:=IntToStr(obsah);     //do Labelu pro výsledky obsahu vložíme výsledek
```

Obrázek 14 - Ukázka zdrojového kódu pro výpočet obsahu a obvodu

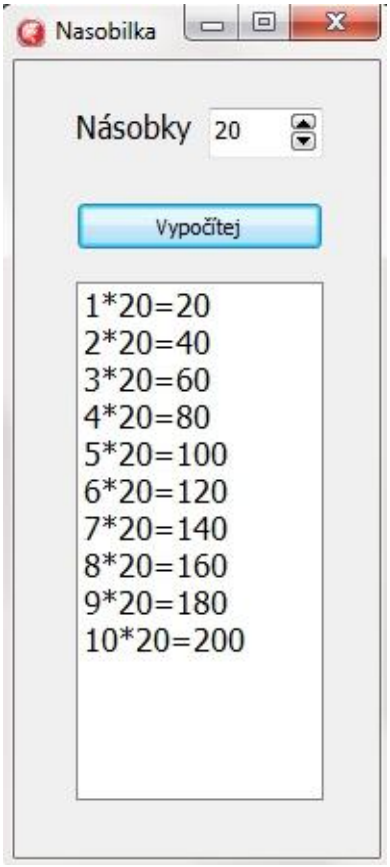
Při psaní zdrojového kódu bychom neměli zapomínat na psaní komentářů, protože pokud se ke kódu vrátíme po delším čase, tak si již nebudeme pamatovat, jak byl program řešen. Komentáře také slouží jiným programátorům, aby se rychle zorientovali v našem programu. Jednořádkové komentáře píšeme za dvě lomítka (*//*) a víceřádkové komentáře vkládáme do symbolů **{** a **}**.

V tomto programu jsme pro přehlednost deklarovali lokální proměnné stranaA, stranaB, obvod a obsah, které jsou typu integer. Do proměnné stranaA jsme vložili obsah z komponenty Edit1 a do proměnné stranaB jsme vložili obsah z komponenty Edit2. Jelikož obsah komponenty Edit je typu String, musíme převést tuto hodnotu na datový typ integer funkcí StrToInt. Do proměnné Obsah jsme vložili výpočet obsahu a do proměnné obvod výpočet obvodu. Nakonec výsledky přiřadíme do vlastností Caption komponent Label5 a Label6. Opět musíme výsledky přetypovat, protože vlastnost Caption je typu String.

Deklarace lokálních proměnných byla jen z důvodu přehlednosti. Tento příklad bychom mohli řešit mnohem rychleji, například obsah obdélníka bychom mohli řešit i takto: *Form4.Label5.Caption:=IntToStr(StrToInt(form4.Edit1.Text)*StrToInt(form4.Edit2.Text));*

4.2 Příklad 2

Vytvořte program Násobilka, který bude vypisovat násobilku čísel od jedné do sta. V programu vytvořte vlastní třídu a metodu.



Obrázek 15 - Násobilka

Tento příklad slouží k procvičení vytváření vlastních tříd, metod a k procvičení cyklu s pevným počtem opakování. Dále se student seznámí s komponenty SpinEdit a Listbox.

Po vytvoření vzhledu aplikace je třeba se seznámit s některými vlastnostmi komponenty SpinEdit. Vlastností MinValue nastavíme minimální hodnotu (v našem případě1) a vlastností MaxVaklue natavíme maximální hodnotu (v našem případě 100). Tím si zajistíme, že číselný vstup od uživatele nebude nižší než 1 a vyšší než 100. Vlastnost Value udává aktuální číselnou hodnotu v komponentě. U komponenty ListBox je důležitá vlastnost Items, ve které se ukrývá obsah ListBoxu. K této vlastnosti existují příkazy, jako například add (přidání obsahu) nebo clear (vymazání obsahu).

Ve zdrojovém kódu nejprve vytvoříme vlastní třídu TVypocet, která bude obsahovat proměnnou **hodnota** (která je typu integer) a proceduru **vypisNasobky**, která obsahuje vstupní parametr typu integer. Jak lze vidět na obrázku 16, následně je třeba deklarovat instanci třídy (***Vypocet : TVypocet;***). Dále je nutné vytvořit objekt. V tomto případě bude nejvhodnější tento objekt vytvořit v nějaké inicializační události formuláře například v události formuláře OnCreate. Napsáním ***Vypocet:= TVypocet.Create****;* vytvoříme objekt Vypocet a alokovali jsme si místo v paměti pro tento objekt.

Dále je třeba v programu definovat proceduru **vypisNasobky**, která bude vypisovat násobky do komponenty ListBox1. Je zřejmé, že pro tuto činnost se bude hodit cyklus s pevným počtem opakování. Jak je vidět na obrázku č. 16, v cyklu vypisujeme hodnoty do komponenty ListBox1 příkazem **Form4.ListBox1.Items.add**. Obsah komponenty ListBox je typu string, tudíž musíme vkládaný obsah (který je typu integer) přetypovat na string funkcí IntToStr. Dále definujeme událost komponenty Button1 OnClick, která nám smaže obsah komponenty ListBox1 příkazem **Form4.ListBox1.Items.clear** a voláme proceduru vypisNasobky, která nám ListBox1 opět naplní. Nakonec musíme vytvořený objekt Vypocet zrušit

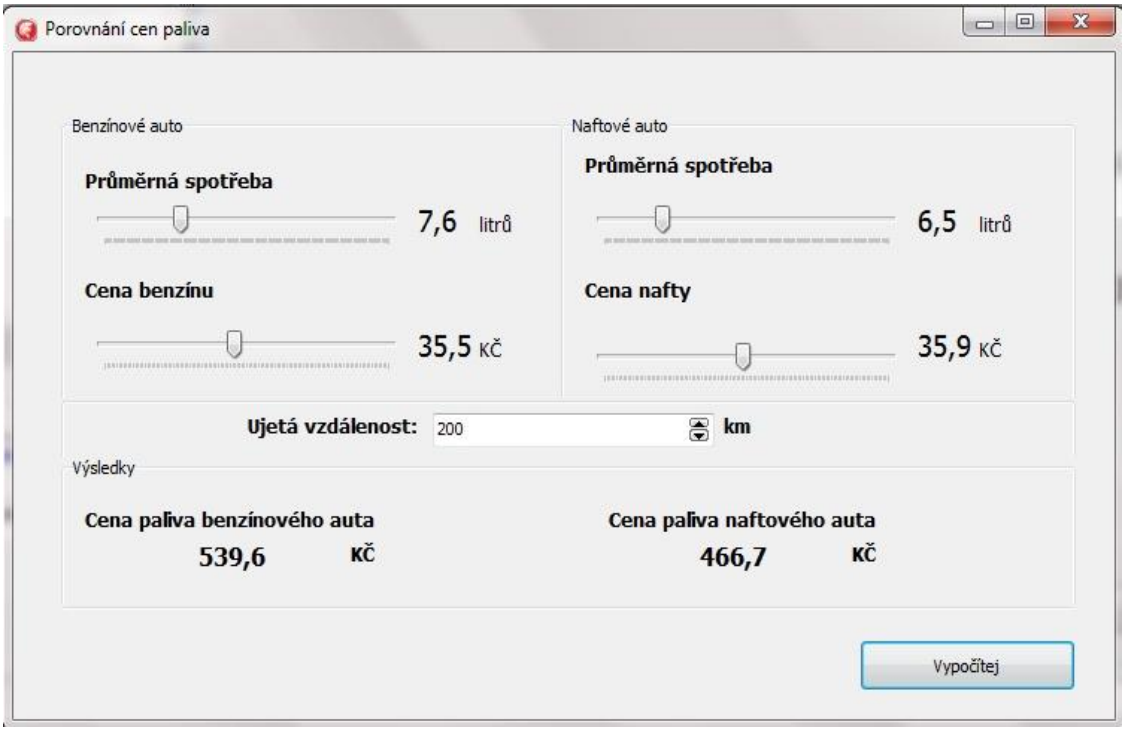
(uvolnit vypůjčené místo v paměti) destruktorem. To se provádí v události formuláře OnClose (nastává při zavření formuláře) a napíšeme to takto: **Vypocet.Free;**. Pro zrušení objektu volaláme metodu Free, která volá destruktor.

```
var
  Form4: TForm4;
  Vypocet: TVypocet;

implementation

{$R *.dfm}

procedure TVypocet.vypisNasobky(a:integer);
var I,vysl : integer;
begin
  vysl:=0;

    for I := 1 to 10 do
      begin
        vysl:=I*a;
        Form4.ListBox1.Items.Add(IntToStr(I)+'*'+IntToStr(a)+'='+IntToStr(vysl));
      end;

end;

procedure TForm4.Button1Click(Sender: TObject);
var cislo : integer;
begin
Form4.ListBox1.Items.Clear;
cislo:=Form4.SpinEdit1.Value;

Vypocet.vypisNasobky(cislo);

end;
```

Obrázek 16 - Ukázka zdrojového kódu programu Násobilka

4.3 Příklad 3

Vytvořte program, který bude porovnávat ceny paliva na cestu u benzínového a naftového automobilu podle průměrné spotřeby, ceny paliva a ujeté vzdálenosti. Při tvorbě využijte vlastní třídu TAuto ze které budou vytvořeny objekty BenzinoveA a NaftoveA. Hodnoty spotřeby a ceny paliva vkládejte jako desetinné číslo s využitím komponenty TrackBar. Pro výpočet ceny paliva použijte funkci, která bude definovaná v samostatné programové jednotce.



Obrázek 17 - Porovnání cen paliva

Hlavním cílem tohoto programu je na jednoduchém příkladu představit studentovi zapouzdření. V programu si student vytvoří dvě instance z jedné třídy. Tyto instance mají stejné atributy a chování, přičemž atributy a metody z jedné instance jsou nezávislé na atributech a metodách instance druhé. Student se dále naučí připojit do programu druhou programovou jednotku a z ní používat třídu a její atributy a chování. Dále se student seznámí s komponentou TrackBar a s jejími vlastnostmi.

Při vytváření vzhledu programu jsme použili komponentu TrackBar. Nejvýznamnější vlastnosti TrackBaru jsou vlastnosti Min (minimální číselná hodnota), Max (maximální číselná hodnota) a Position (číselná hodnota na kterou ukazuje ručička TrackBaru). V zadání bylo napsáno, že hodnoty z TrackBaru mají být desetinná čísla. Pokud chceme nastavit, aby minimální hodnota byla 3 a maximální 20, je tedy třeba tyto hodnoty zadat do vlastnosti Min a Max desetkrát vyšší (pro desetiny). V programu pak tyto hodnoty dělíme deseti a dostáváme desetinné číslo. Na obrázku 17 lze vidět, že vedle TrackBarů jsou vypsány jejich hodnoty. Tyto popisky nejsou

součástí komponenty, ale jedná se o komponentu Label. Je tedy třeba kliknout na událost komponenty TrackBar OnChange (při změně) a v této události nastavit vypisování hodnot z vlastnosti Position do vlastnosti Caption komponenty Label. To uděláme například takto: *Form4.LabelPSB.Caption := FloatToStr(Form4.TrackBarPSB.Position/10);.* Práce se zbylými komponentami v tomto programu by měla být zřejmá již z předchozích příkladů.

Ve zdrojovém kódu jako první vytvoříme vlastní třídu **TAuto**. Tato třída bude obsahovat atributy **prumSpotreba** (pro průměrnou spotřebu) a **cenaPaliva** (pro cenu paliva). Tyto atributy budou typu real, protože budeme pracovat s desetinnými čísly. Z této třídy si vytvoříme dvě instance třídy, a to **BenzinoveA** a **NaftoveA**.

Dále si vytvoříme samostatnou programovou jednotku Vzorec.pas příkazem **File → New → Unit-Delphi**. V této jednotce vytvoříme vlastní třídu TVypocetCeny. V této třídě bude pod skupinou **Private** proměnná **a** (typ integer) a pod skupinou **Public** funkce **cenaPaliva** s parametry **prumSpotreba**, **cenaPaliva** a **vzdálenost**. Návratová hodnota bude typu real. Dále v této programové jednotce tuto funkci definujeme. Funkce bude pracovat s tímto vzorcem: ((průměrná spotřeba v litrech na sto kilometrů * cena paliva za litr) / 100) * vzdálenost v kilometrech). Všechny tyto hodnoty se nacházejí v parametrech této funkce. Výsledek z tohoto vzorce a také výstup z této funkce přiřadíme do proměnné **Result**. Tuto programovou jednotku do našeho programu připojíme tak, že v části programu **uses** napíšeme **Vzorec**.

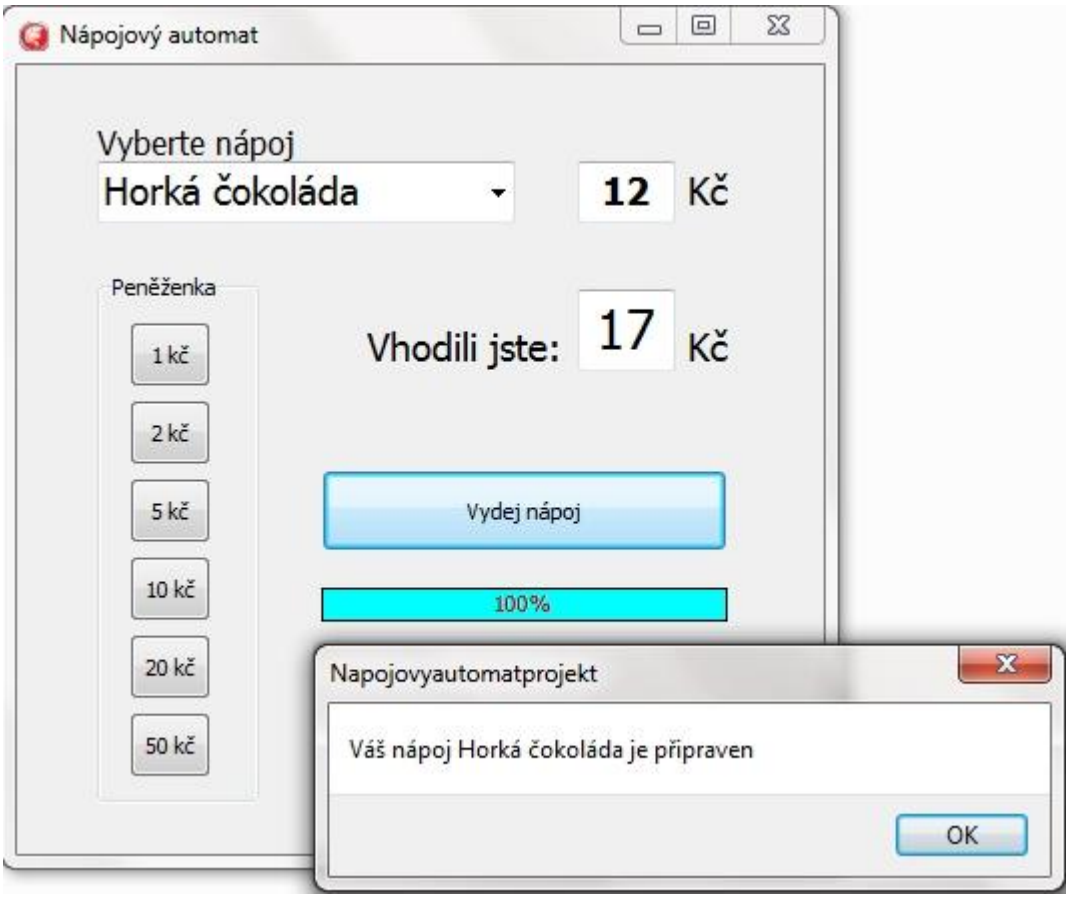
Poté deklarujeme instance třídy BenzinoveA (TAuto), NaftoveA (TAuto) a Vypocet (TVypocetCeny). Následně vytvoříme tyto objekty konstruktérem Create v události formuláře **OnCreate.** Na konci programu je potřeba tyto objekty odstranit z paměti. To uděláme například v události formuláře **OnClose** metodou **free**.

Právě v této fázi lze zpozorovat zapouzdření. Pokud budeme chtít přiřadit nějakou hodnotu typu integer do proměnné **a** objektu Vypocet, program při kompilaci zahlásí chybu, protože tato proměnná je deklarována ve třídě TVypocetCeny pod skupinou **Private**. Je tedy soukromá a nemůžeme k ní přistupovat mimo programovou jednotku Vzorec. Funkce **cenaPaliva** je deklarována také v této třídě, avšak pod skupinou **Public**, takže to znamená, že tuto funkci můžeme volat.

Nakonec zbývá událost na tlačítko „Vypočítej“. Tak jak je vidět v ukázce zdrojového kódu na obrázku 18, v této události se budou přiřazovat hodnoty z Trackbarů do proměnných **cenaPaliva** a **prumSpotreba** objektů **BenzinoveA** a **NaftoveA**. Dále je zde deklarovaná proměnná **vzdálenost** (která je typu integer) a do této proměnné se budou vkládat hodnoty z komponenty SpinEdit1. Tato proměnná však není nutná, je tu jen pro přehlednost. Na konec v této události vypisujeme vypočítané výsledky do komponent LabelCenaBenzinu a LabelCenaNafty. Výpis hodnot do těchto komponent můžeme například vypadat takto: *Form4.Label*CenaBenzinu*.Caption:=FloatToStr(BenzinoveA.vypocetCeny(BenzinoveA.PrumSpotreba,BenzinoveA.CenaPaliva,vzdalenost));*. Jak je z této ukázky zřejmé, výpočet se provádí ve funkci, která je přímo v přiřazení do komponenty LabelCenaBenzinu. Jelikož výsledná hodnota z funkce **vypocetCeny** je datového typu real, musíme výsledek přetypovat funkcí **FloatToStr.**

4.4 Příklad 4

Vytvořte program, který bude představovat nápojový automat. Pro výběr nápojů využijte komponentu ComboBox. Jednotlivé nápoje budou instance třídy TNapoj, která bude obsahovat vlastnosti Nazev (datový typ string) a Cena (datový typ integer). Dále bude obsahovat vlastní konstruktor, kterým vytvoříte objekty představující nápoje a zároveň nastavíte jejich vlastnosti. Simulujte i vhazování mincí a dobu potřebnou k přípravě nápoje.



Obrázek 18 - Nápojový automat

Tento příklad studentovy poslouží k dalšímu procvičení zapouzdření a práce s komponenty. Dále se v tomto příkladu naučí pracovat s vlastním konstruktorem, s vyskakovacím oknem Showmessage a s komponentou Gauge.

Jak je z obrázku 19 zřejmé, pro výběr nápojů byla použita komponenta ComboBox. Pro práci s touto komponentou je třeba znát dvě její vlastnosti. První je vlastnost Items, ve které jsou uloženy položky seznamu a druhou vlastností je vlastnost ItemIndex, která nám říká, jaká položka seznamu je vybrána (ItemIndex 0 je první položka, 1 je druhá položka, -1 žádná položka není vybrána). Dále je na obrázku 19 vidět, že pro zobrazení cen jsou zvoleny komponenty Edit. V tomto případě nechceme, aby uživatel do těchto komponent psal hodnoty, ale aby sloužily jen pro čtení. Je tedy třeba u těchto komponent označit vlastnost ReadOnly jako True. Dále tlačítko „Vydej nápoj“ je znepřístupněno, dokud uživatel

„nevhodí“ dostatek peněz. Proto vlastnost Enabled této komponenty označíme jako False. Pro simulaci doby, potřebné k přípravě nápoje jsem si zvolil komponentu Gauge, která nám v procentech znázorňuje nějaký progres a komponentu Timer, která nám v intervalech spouští libovolný kód.

Jak bylo v zadání psáno, každá položka z ComboBoxu bude instancí třídy TNapoj. Proto si vytvoříme vlastní třídu TNapoj, která bude mít dvě vlastnosti, a to cena (typ integer) a nazev (typ string). Tyto vlastnosti budou ve skupině **Private**, takže budou soukromé a budou přístupné jen v rámci programové jednotky. Mimo jednotku budou nepřístupné. V této třídě dále do skupiny Public deklarujeme konstruktor VytvorObjekt, který bude mít dva parametry a to N (pro vlastnost objektu nazev) a C (pro vlastnost objektu cena). Jak je vidět na obrázku č. 20, deklarujeme instance třídy podle nápojů v comboBoxu. V našem případě to budou objekty Caj, HorkaCokolada a Kava. Dále je třeba si vytvořit další vlastní třídu TAutomat. Tato třída bude obsluhovat celý chod nápojového automatu, proto bude obsahovat atributy **cenaNapoje**, **vhozenaCastka** a **nazevNapoje** a dále bude obsahovat procedury **zkontrolujPenize** (kontroluje, zda je vhozen dostatečný obnos), **vhozeniMince** (kontroluje vhozenou částku) a funkci **vratPenize** (zajišťuje vrácení peněz).

```
  TNapoj = class           //třída TNapoj _ každá instance této třídy bude představovat
   private                 //jeden nápoj s vlastnostmi cena a název
    nazev:String;
    cana : integer;
50  public
51  Constructor VytvorObjekt(N:string; C:integer); //deklarace konstruktoru
  end;

  TAutomat = class     //třída TAutomat _ v této třídě budou proměnné a metody sloužící k
  private              //obsluze napojoveho automatu
    cenaNapoje : integer;
    vhozenaCastka: integer;
    nazevNapoje: String;
  public
60  procedure zkontolujPenize;
    procedure vhozeniMince(a:integer);
    function vratPenize(a,b : integer):integer;

  end;

var
  Form4: TForm4;            //deklarace instance třídy TForm4
  Caj: TNapoj;              //3 deklarace instance tříd TNapoj
  HorkaCokolada: TNapoj;
70  Kava: TNapoj;
  Automat:TAutomat;         //deklarace instance třídy TAutomat
```

Obrázek 19 - Zdrojový kód programu Nápojový automat

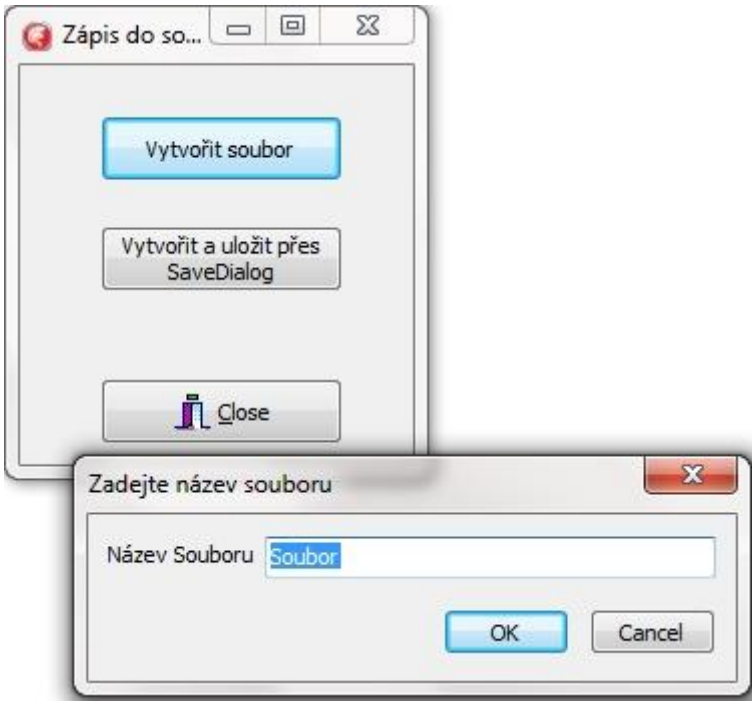
Následně je třeba objekty vytvořit. V zadání je psáno, abychom vytvořili vlastní konstruktor ve třídě TNapoj. Jelikož v této třídy budeme vytvářet více objektů, tak vlastní konstruktor nám ulehčí práci při vytváření objektů. Na obrázku 20 je vidět deklarace konstruktoru VytvorObjekt. Objekt Automat z třídy TAutomat byl vytvořen defaultním konstruktorem Create. Opět jako v předchozích příkladech je vhodné tyto objekty vytvořit v nějaké inicializační události formuláře, například v události formuláře OnCreate. Na konci programu je třeba alokované místo v paměti uvolnit. Objekty zrušíme v události formuláře OnClose procedurou Free.

Pro činnost programu při vybrání nápojů zvolíme událost komponenty ComboBox OnChange, která se provede při změně vybrané položky. V této události přiřazujeme do vlastností **cenaNapoje** a **nazevNapoje** z objektu Automat vlastnosti **cena** a **nazev** z objektů Caj, HorkaCokolada nebo Kava, a to podle toho, jaká položka v ComboBoxu je zvolena (vlastnost ItemIndex).

Na obrázku 20 lze zpozorovat, že pro simulaci vhození mincí bylo zvoleno šest komponent Button, které představují jednotlivé mince. V událostech OnClick těchto komponent se přičítá hodnota „mince“ do vhozené částky (procedura vhozeniMince(castka)) a kontroluje, zda-li vhozená částka dosáhla požadované hodnoty (procedura zkontolujPenize). Pokud „vhozená“ částka dosáhla nebo překročila požadovanou částku, zpřístupní se tlačítko „Vydej nápoj“. V události tohoto tlačítka se zviditelní komponenta Gauge1 a spouští se časovač, který bude po vteřinách (vlastnost interva:=1000) přičítat do vlastnosti Progress komponenty Gauge1 hodnotu 20. Tímto simulujeme dobu potřebnou k přípravě nápoje. Jakmile hodnota ve vlastnosti Progress dosahne sta (100%), zastaví se časovač a vyskočí nám okénko se zprávou (showmessage), kde se píše, že náš nápoj je připraven. Po zavření tohoto okna vyskočí druhé okno, které nás informuje o tom, kolik peněz bylo vráceno. Po zavření tohoto okna se vynuluje hodnota Progress z objektu Gauge1 a vhozenaCastka z objektu automat.

4.5 Příklad 5

Vytvořte program, který bude obsahovat třídu TStudent, která bude potomkem třídy TClovek. TClovek bude obsahovat atributy jméno, příjmení a věk a TStudent bude obsahovat vlastní atributy škola, ročník a rok ukončení. Všechny atributy z třídy TStudent vypisujte do souboru s využitím InputBoxu a SaveDialogu.



Obrázek 20 - Zapisování do souboru

Tento úkol je zaměřený na dědičnost a práci se soubory. Student se naučí využívat dědičnost a zapisovat do souboru. Dále se v tomto příkladu seznámí s InputBoxem, SaveDialogem, a jak je na obrázku 21 vidět, seznámí se i s komponentou BitBtn (BitButton).

Vzhled této aplikace je velmi jednoduchý, obsahuje již známé komponenty z předešlých příkladů. Jedinou výjimkou je komponenta SaveDialog a BitBtn. V tomto příkladu komponenta BitBtn slouží pro zavírání okna. V její vlastnosti Kind se dá zvolit druh. V našem případě byl zvolen druh bkClose, který sám zavírá aplikaci bez jakéhokoliv programování.

Jak už bylo psáno, tento příklad je zaměřen na dědičnost. Proto je třeba začít vytvořením vlastní třídy TClovek, která podle zadání obsahuje vlastnosti Jmeno, Prijmeni a Vek. Dále je třeba vytvořit druhou vlastní třídu TStudent, která bude dědit všechny vlastnosti i chování třídy TClovek a dále bude obsahovat vlastní atributy Rocnik, RokUkonceni a Skola a vlastní proceduru vypisDoSouboru. Ukázka vytvoření těchto tříd je na obrázku 22.

```
TClovek = class            //Rodičovská třída TClovek
  Jmeno, Prijmeni: String;
  Vek: integer;
 end;

TStudent = class (TClovek)          //třída TStudent, která je potomkem třídy TClovek.
  Rocnik, RokUkonceni: integer;
  Skola: String;
  Procedure vypisDoSouboru(nazevSouboru: String);
end;
```

Obrázek 21 - Rodič a potomek

Další postup je již známý z předchozích příkladů, a to deklarace objektů (instancí tříd), vytvoření objektů kontraktory Create v události formuláře OnCreate a zrušení objektů metodou Free v události formuláře OnClose. V tomto příkladě stačí vytvářet jen objekt z třídy **TStudent**, protože objekt z třídy TClovek nepotřebujeme. Instance třídy Student (objekt z třídy TStudent) nyní obsahuje všechny atributy a chování z třídy **TClovek**

a z třídy **TStudent**. Dále je třeba definovat proceduru **VypisDoSouboru**. Tato procedura obsahuje parametr **nazevSouboru** (který je typu string), který využijeme pro vkládání názvu souboru. Na obrázku 23 vidíme definici této procedury. Při práci se soubory musíme nejprve vytvořit spojení se souborem, a to uděláme příkazem **AssignFile**, který spojuje souborovou proměnnou s názvem souboru. Budeme zapisovat do textového souboru, tak si pod deklarace instancí tříd deklarujeme souborovou proměnou **soubor** typu **TextFile**. Dále volíme, jak daný souborem otevřeme, a to tak, že vytvoříme nový, nebo přepíšeme stávající soubor příkazem **Rewrite**. Příkazem **Reset** otevřeme soubor, do kterého chceme zapisovat od začátku, anebo otevřeme soubor příkazem **Append**, kterým zapisujeme do souboru od konce. Z obrázku 23 je zřejmé, že atributy objektu Student zapisujeme pomocí příkazu **Writeln(souborová proměnná, 'text')**. Nakonec musíme daný soubor zavřít, a to příkazem **CloseFile**.

```
procedure TStudent.vypisDoSouboru(nazevSouboru: String);          //procedura která pracuje s textovým
begin

    AssignFile(soubor,nazevSouboru);    //spojení souborové proměnné s názvem souboru
    rewrite(soubor);                   //vytvoříme nový soubor nebo přepíšeme stávající soubor
    Writeln(soubor,'Jmeno: '+Student.Jmeno+' '+Student.Prijmeni); //zapisujeme do souboru jméno a příjm
    Writeln(soubor,'Vek: '+IntToStr(Student.Vek)+' let');         //zapisujeme do souboru věk
    Writeln(soubor,'Skola: '+Student.Skola);                      //zapisujeme do souboru školu
    Writeln(soubor,'Rocnik: '+IntToStr(Student.Rocnik));          //zapisujeme do souboru ročník
    Writeln(soubor,'Rok ukonceni: '+IntToStr(Student.RokUkonceni));//zapisujeme do souboru rok ukončení
    closefile(soubor);    //zavíráme soubor


end;

procedure TForm4.Button1Click(Sender: TObject);   //událost na kliknutí na tlačítko "Vytvoř soubor"
var nazevS:String;
begin
   nazevS:='';

   repeat
        nazevS:=InputBox('Zadejte název souboru','Název Souboru','Soubor');   //input box, kterým se pt
   until nazevS <> '';
        //cyklem repeat-until zajišťujeme to, aby uzivatel vždy zadal název souboru
    Student.vypisDoSouboru(nazevS); //provedeme zápis do souboru
end;

procedure TForm4.Button2Click(Sender: TObject);  //událost, při které spustíme dialog pro uložení a vyt
var nazevS:String;
begin
Form4.SaveDialog1.Execute();     //Execute-této se spouští všechny dialogy - otevýrá se okno dialogu
nazevS:=Form4.SaveDialog1.FileName;   //do proměnné nazevS se uloží název souboru ze SaveDialog1.FileNa

if Form4.SaveDialog1.FileName <> '' then   // pokud nedostaneme název souboru tak nenastane vytvoření s
```

Obrázek 22 - Zdrojový kód pro zapsání do souboru

Nyní potřebujeme vytvořit událost na tlačítko „Vytvoř soubor“, ve které se nás program zeptá na název souboru pomocí InputBoxu. InputBox je vyskakovací okénko, do kterého zadáváme vstup do programu. Jak je na obrázku 23 vidět, příkazem *nazevS:= InputBox('Zadejte název souboru','Název Souboru','Soubor')* spustíme InputBox a do proměnné **nazevS** vkládáme hodnotu z něj. Tento příkaz je v programu ošetřen cyklem repeat until pro případ, kdyby uživatel místo názvu souboru poslal prázdný řetězec. Následně hodnotu z InputBoxu vložíme jako parametr do procedury **VypisDoSouboru**.

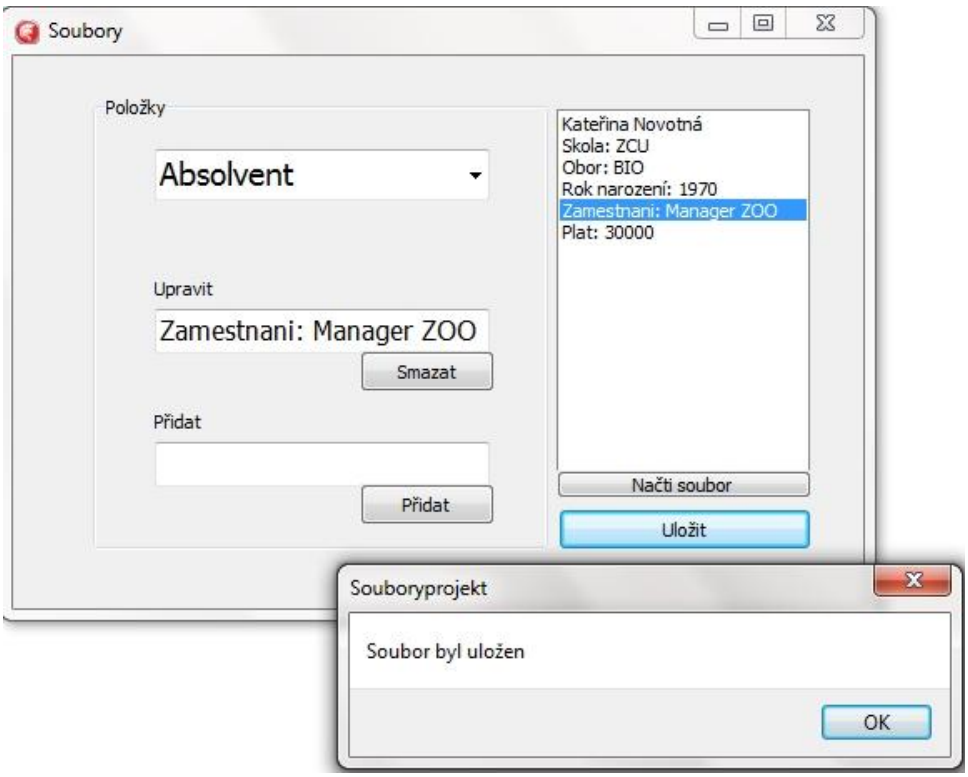
V zadání tohoto příkladu bylo, aby se využilo při ukládání souboru komponenty SaveDialog. To je řešeno pod událostí tlačítka „Vytvořit a uložit přes SaveDialog“. V této události se otevírá dialogové okno příkazem **Execute()** (tímto příkazem se otevírají všechna dialogová okna), jak lze viděť na obrázku 23. SaveDialog nám vrací název souboru ve vlastnosti FileName a tuto hodnotu vkládáme jako parametr do procedury **VypisDoSouboru**. Kdybychom dialogové okno zavřeli, program by volal chybu, protože by pod vlastností FileName byl jen prázdný řetězec. Tato situace je v programu ošetřena podmínkou.

4.6 Příklad 6

Vytvořte program, který bude obsahovat třídu TAbsolvent, která je potomkem třídy TStudent a třída TStudent je potomkem třídy TClovek. Z třídy TStudent a TAbsolvent vytvořte objekty, které budou vypisovat své vlastnosti do souboru. Program tyto soubory bude otevírat (pokud neexistují tak i vytvářet) v komponentě ListBox, kde bude možnost jednotlivé položky upravovat, mazat nebo přidávat. Po uložení by soubory měli zůstat zachované do příštího spuštění aplikace.

Tento program je rozšířením předešlého programu. Tento příklad je opět zaměřený na dědičnost a práci se soubory. Student se dále v tomto

příkladě naučí vypisovat obsah textového souboru do komponenty ListBox a tento obsah v ListBoxu upravovat, mazat či přidávat nový obsah.



Obrázek 23 - Program Soubory

Jak je zřejmé z obrázku 24, vzhled aplikace se skládá z již známých komponent, a to z komponent GroupBox, ComboBox, Edit, Label, Button a ListBox. Na obrázku č. 25 je ukázka vytváření a dědění tříd. Jak je vidět, Rodičovská třída TClovek obsahuje atributy Jmeno, Prijmeni, RokNarozeni a NazevSouboru. Tyto atributy bychom potřebovali využít i ve třídě TStudent a navíc ji rozšířit o atributy Škola, Obor, Rocnik a proceduru vypisSkolu. Je tedy zbytečné deklarovat novou třídu a znova deklarovat ty samé atributy jako jsou ve třídě TClovek. Je tu proto dědičnost, která nám umožní dědit z třídy TClovek veškeré atributy a metody. V tomto programu bychom potřebovali pracovat dále s objektem Absolvent, který by měl mít stejné vlastnosti jako objekt Student (ze třídy TStudent) a dále bychom potřebovali tento objekt rozšířit o vlastnost Zamestnani, Plat a proceduru vypisZamestnani. Proto vytvoříme třetí třídu TAbsolvent, která bude

potomkem třídy TStudent. Tímto třída TAbsolvent bude obsahovat zděděné vlastnosti a metody ze tříd TClovek a TStudent.

Při vytváření programu, tedy začneme s vytvořením vlastních tříd TClovek, TStudent a TAbsolvent. Na obrázku 25 je představena ukázka vytvoření vlastních tříd. Každá třída, která dědí má deklarované jen své vlastní atributy a chování, zbylé atributy a chování dědí z nadřazené třídy.

```
  TClovek = class
    private
    Jmeno, Prijmeni: String;
40  RokNarozeni: integer;
    nazevSouboru: String;

  end;

  TStudent = class (TClovek)  //třída TStudent, která dědí atributy z třídy TClovek
    private
    Skola, obor: String;
    Rocnik: integer;
    public
50  Procedure VypisSkolu;
    constructor VytvorObjektStudent(NS,Jm,Pr,Sk,Ob:String;RN,Ro:integer);
  end;

  TAbsolvent = class (TStudent) //třída TPracující, která zdědila atributy
    private
    Zamestnani: String;
    Plat: integer;
    public
    procedure VypisZamestnani;
60  constructor VytvorObjektAbsolvent(NS,Jm,Pr,Sk,Ob,Zam:String;RN,P:integer);
  end;

var
  Form4: TForm4;             //deklarace instancí tříd
  Student:TStudent;
  Absolvent:TAbsolvent;
  soubor: textfile;         //deklarace proměnné typu TextFile
```

Obrázek 24 - Vytvoření vlastních tříd v šestém příkladu

Následuje již známé deklarování objektů (instancí tříd). V tomto příkladě jsme využili vlastních konstruktorů v třídě TStudent a TAbsolvent, kterými inicializujeme i vlastnosti instancí těchto tříd. Objekty Student (z třídy TStudent) a Absolvent (z třídy TAbsolvent) tedy vytvoříme například v události formuláře OnCreate vlastními konstruktory. Nesmíme zapomenout i na zrušení objektů metodou Free v události formuláře OnClose.

V tomto příkladě budeme pracovat se dvěma soubory, protože každý objekt bude vypisovat své atributy do svého souboru. To budou zařizovat dvě procedury **vypisSkolu** ze třídy TStudent a **vypisPovolani** ze třídy TAbsolvent. Tyto procedury budou kontrolovat, zda existuje soubor, ze kterého se budou načítat atributy daného objektu do ListBoxu. Pokud daný soubor neexistuje, vytvoří ho a vypíše do něj atributy daného objektu. Pokud daný soubor existuje, tak nový vytvářet nebude. Zda-li daný soubor existuje nebo ne, zjistíme funkcí **FileExists(‘názevSouboru‘)**. Pokud hodnota této funkce bude **False**, znamená to, že soubor neexistuje. Na obrázku 26 je ukázka definice těchto procedur. Jelikož těmito procedurami ověřujeme existenci souborů již při spuštění programu, je nutné tyto funkce volat při vzniku formuláře, tedy v události formuláře OnCreate.

Na obrázku 24 je vidět, že pod komponentou ListBox je tlačítko „Načti soubor“. Tímto tlačítkem budeme načítat daný soubor do ListBoxu. Nejprve je ale třeba zjistit, kterou položku máme v komponentě ComboBox vybranou, protože podle toho volíme soubor, který se nám načte. Toto může být řešeno například podmíněnými if. Samotné načítání souboru do komponenty ListBox vytvoříme tímto jednoduchým příkazem: *Form4.ListBox1.Items.LoadFromFile(‘nazevSouboru‘);*.

Jako další je třeba zařídit, aby komponenta Edit1 sloužila pro upravování jednotlivých položek v ListBoxu. To se dá zařídit dvěma příkazy. První příkaz bude v události ListBoxu1 OnClick a to tento příkaz: *Edit1.Text := ListBox1.Items[listbox1.itemindex*];. Ten vypíše vybraný řádek

ListBoxu1 do editační řádky komponenty Edit1. Druhý příkaz bude v události komponenty Edit1 OnChange. Ten bude vypadat takto: *ListBox1.Items[listbox1.itemindex] := Edit1.Text;.* Pokud v této události necháme jen tento příkaz, může nastat chyba, a to právě když budeme psát do komponenty Edit1 a přitom nebudeme mít označený řádek v ListBoxu. Toto lze opatřit podmíněným příkazem, kdy se budeme ptát, pokud je řádka v ListBox označena (*if ListBox1.Selected[listbox1.itemindex] = true*), zpřístupní se editační řádka komponenty Edit1 (vlastnost Enabled).

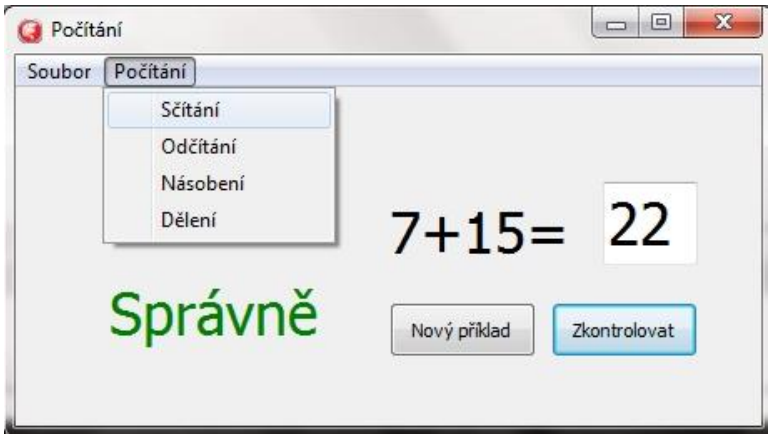
Pro smazání položky v ListBoxu je v příkladě určeno tlačítko „Smazat“. V události OnClick tohoto tlačítka je jen jeden příkaz, a to tento: *(ListBox1.Items.Delete(ListBox1.ItemIndex);*. Pro přidávání položek do ListBoxu je v programu určeno druhé editační okénko komponenty Edit2. Obsah této komponenty vložíme do řádku ListBoxu příkazem *Listbox1.Items.Add(Edit2.Text);*. Ten se nachází v události komponenty Edit2 OnClick. Nakonec obsah ListBoxu uložíme zpět do souboru. To provedeme v události OnClick tlačítka „Uložit“, a to příkazem *ListBox1.Items.SaveToFile(‘nazevSouboru‘);*. Jelikož pracujeme se dvěma soubory (podle objektů), které si přepínáme v komponentě ComboBox, je třeba v této události rozlišit, s jakým souborem pracujeme, a to například podmíněnými příkazy if, kde ověřujeme, jaká položka ComboBoxu je označena.

```
procedure TStudent.VypisSkolu;    //funkce pro vytvoření souboru "souborStudent" pokud neexi
begin

   if FileExists(Student.nazevSouboru)=false then begin //podminka-pokud neexistuje soubor "
    AssignFile(soubor,Student.nazevSouboru);
    rewrite(soubor);
    Writeln(soubor,Student.Jmeno+' '+Student.Prijmeni);
    Writeln(soubor,'Rok narozeni: '+IntToStr(Student.RokNarozeni));
    Writeln(soubor,'Skola: '+Student.Skola);
    Writeln(soubor,'Obor: '+Student.Obor);
    Writeln(soubor,'Rocnik: '+IntToStr(Student.Rocnik));
    closefile(soubor);
  end  else begin
      AssignFile(soubor,Student.nazevSouboru);  //jinak se jen otevře a zavře
      Append(soubor);
      closefile(soubor);
  end;
end;

procedure TAbsolvent.VypisZamestnani; //funkce pro vytvoření souboru "souborAbsolvent" pokud
begin

if FileExists(Absolvent.nazevSouboru)=false then begin //podminka-pokud neexistuje soubor "s


    AssignFile(soubor,Absolvent.nazevSouboru);
    rewrite(soubor);
    Writeln(soubor,Absolvent.Jmeno+' '+Absolvent.Prijmeni);
    Writeln(soubor,'Skola: '+Absolvent.Skola);
    Writeln(soubor,'Obor: '+Absolvent.Obor);
    Writeln(soubor,'Rok narozeni: '+IntToStr(Absolvent.RokNarozeni));
    Writeln(soubor,'Zamestnani: '+Absolvent.Zamestnani);
```

Obrázek 25 - Ukázka zdrojového kódu programu č.6

4.7 Příklad 7.

Vytvořte program Počítání, který bude generovat příklady pro sčítání, odčítání, násobení a celočíselné dělení. Následně bude ověřovat správnost výsledku, který uživatel bude zadávat. Pro každý druh příkladu vytvořte třídu, která bude dědit veškeré atributy a funkci pocitej z třídy TPocitani. Ve zděděných třídách překryjte tuto zděděnou funkci. V programu dále vytvořte menu, ve kterém budete volit druh příkladu.



Obrázek 26 - Program Počítání

V tomto příkladě se student seznámí opět s dědičností a seznámí se s polymorfizmem, konkrétněji s překrýváním metod. Dále se student seznámí s novou komponentou MainMenu, která vytváří menu pod horním okrajem okna aplikace.

Jak je ze zadání příkladu zřejmé, budeme využívat polymorfizmus. Přesněji budeme pracovat s překrýváním metod. Překrývání metod ulehčuje a zpřehledňuje práci programátorovi. Využijeme ji ve chvíli, kdy dědíme atributy a metody z rodičovské třídy a potřebujeme předefinovat zděděnou metodu. Tato metoda se bude jmenovat stejně a bude mít i stejné vstupní parametry, takže programátor si nebude muset pamatovat více názvů metod, ale jen jeden název metody, kterou bude používat. Která z metod se provede, se určí objektem, který s touto metodou pracuje.

Nejprve si vytvoříme třídu TPocitani, ze které budou ostatní třídy dědit veškeré atributy a metody. Za funkci **pocitej**, kterou budeme překrývat, napíšeme klíčové slovíčko **virtual**. Dále vytvoříme třídy, které budou z třídy TPocitani dědit a budou představovat druh příkladu. Vytvoříme tedy třídy TScitani, TDeleni, TNasobeni a TDeleni. Každá tato třída bude mít deklarovanou funkci **pocitej** a za ní bude napsáno klíčové slovo **override** tak, jak je tomu na obrázku 28.

```
  TPocitani = class                          //Rodičovská třída TPocitani
     a, b, vysledek : integer;
40   function pocitej:integer;virtual;
  end;

  TScitani = class (TPocitani)              //Třída TScitani která dědí z třídy TPocitani
44   function pocitej:integer;override;      //zděděnou funkci pocitej překryjeme vlastní
  end;

  TOdcitani =  class (TPocitani)
    function pocitej:integer;override;
  end;
50
  TNasobeni =  class (TPocitani)
    function pocitej:integer;override;
  end;

  TDeleni =  class (TPocitani)
    function pocitej:integer;override;
  end;
```

Obrázek 27 - Třídy programu Počítání

Překryté funkce **pocitej** budou generovat příklady tak, že budeme přiřazovat náhodná čísla do proměnné **a** a proměnné **b** funkcí **Random(a:integer):integer;**. Tyto proměnné pak budeme sčítat, odčítat, násobit nebo dělit podle toho v jaké třídě tato funkce bude nacházet. Dále ve funkcích **pocitej** budeme do pomocné proměnné **Druh** vkládat druh příkladu a to například takto: *Druh := 'Scitani';*. Tato pomocná nám říká, jaký druh příkladu právě počítáme.

Samozřejmě nesmíme zapomenout na deklaraci instancí tříd, vytvoření objektů a zrušení objektů. Pod deklaracemi instancí tříd deklarujeme pomocnou proměnnou typu string (proměnnou **Druh)**, do které budeme vkládat druh příkladu. To nám pomůže při kontrole zadaných výsledku.

Následně je čas přejít ke komponentě MainMenu. Nejdůležitější vlastnost této komponenty je vlastnost Items. Po kliknutí na tuto vlastnost v Object Inspectoru se nám otevře MainMenu Designer, kde si vytvoříme všechny položky menu. V našem příkladě je vytvořená položka Soubor, ve které je položka Konec (ukončuje aplikace příkazem **close**). Vedle položky Soubor je položka Počítání, ve které jsou další položky, a to Sčítání, Odčítání, Násobení a Dělení. Po dvojkliku pravým tlačítkem myši na položku, se nám vytvoří ve zdrojovém kódu událost OnClick na danou položku. V událostech OnClick položek Sčítání, Odčítání, Násobení a Dělení budeme volat překryté funkce **pocitej** podle druhu příkladu a do komponenty Label1 vypisovat zadání příkladu.

```
procedure TForm4.Button1Click(Sender: TObject); //Událost při kliku na tlačítko "Nový příklad"
begin
      Label2.Caption:='';
      if druh = 'Scitani' then  begin
           Scitani.vysledek:= Scitani.pocitej;
      Label1.Caption:=IntToStr(Scitani.a)+'+'+IntToStr(Scitani.b)+'=';
        end;

   if druh = 'Odcitani' then  begin
         Odcitani.vysledek:= Odcitani.pocitej;
      Label1.Caption:=IntToStr(Odcitani.a)+'-'+IntToStr(Odcitani.b)+'=';
        end;

   if druh = 'Nasobeni' then  begin
      Nasobeni.vysledek:= Nasobeni.pocitej;
      Label1.Caption:=IntToStr(Nasobeni.a)+'×'+IntToStr(Nasobeni.b)+'=';
        end;

    if druh = 'Deleni' then  begin
      Deleni.vysledek:= Deleni.pocitej;
      Label1.Caption:=IntToStr(Deleni.a)+'÷'+IntToStr(Deleni.b)+'=';
        end;
end;
```

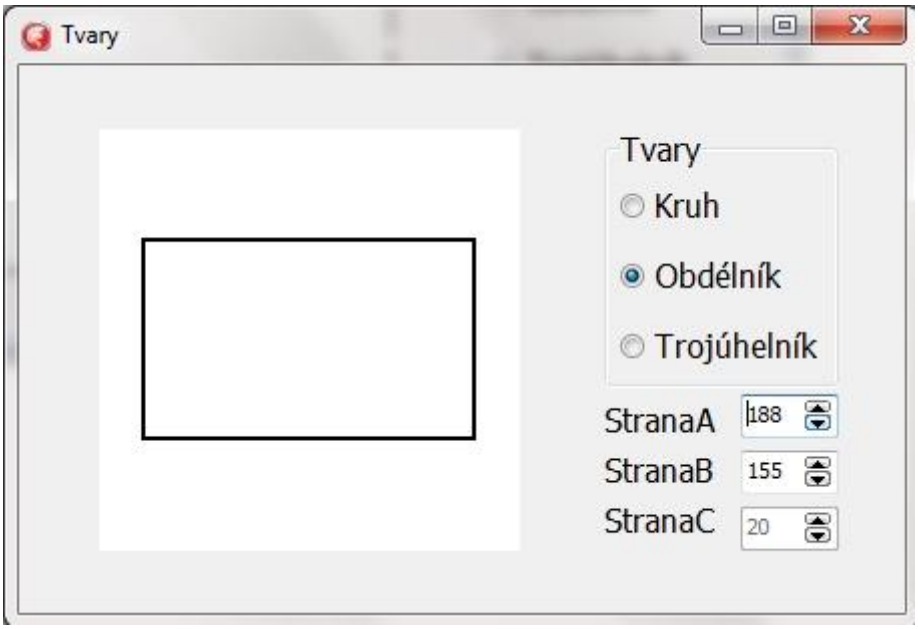
Obrázek 28 - Zdrojový kód programu Počítání

V události OnClick tlačítka „Zkontroluj“ budeme ověřovat správnost zadaného výsledku uživatelem. Zde využijeme pomocnou proměnnou **Druh**. Budeme se podmíněnými příkazy ptát, o jaký druh příkladu se jedná. Podle toho bude program například vědět, že má porovnávat výsledek od uživatele s výsledkem funkce **Scitani.pocitej** a ne s výsledkem funkce **Nasobeni.pocitej**. Dále se v této události bude vypisovat do komponenty Label2, zda byla odpověď od uživatele správná či špatná. Případně se může i měnit barvy písma.

Nakonec v události OnClick tlačítka „Nový příklad“ budeme generovat nový příklad. Opět využijeme pomocnou proměnnou **Druh**, kterou využijeme v podmíněných příkazech. Tím zjistíme, jaký druh příkladu právě počítáme. Následně voláme příslušnou funkci **pocitej** a do komponenty Label1 vypíšeme příklad.

4.8 Příklad 8

Vytvořte program, který ve třídě TMalovani bude obsahovat tři metody s názvem **kresli**. Tyto metody budou přetížené a každá z nich bude provádět jinou činnost. První metoda kresli(a:integer) bude na plátno kreslit kruh, druhá metoda kresli(a,b:integer) bude na plátno kreslit obdélník a třetí metoda kresli(a,b,c:integer) bude na plátno kreslit trojúhelník. Zařiďte i číselný vstup pro změnu velkosti kresleného tvaru.



Obrázek 29 - Program Tvary

Student se v tomto příkladě naučí přetěžovat metody, seznámí se s kreslením v Delphi a naučí se pracovat s komponentami RadioGroup a Imege.

Na obrázku 29 je ukázka vzhledu této aplikace. V tomto příkladu se objevily dvě nové komponenty, a to RadioGroup a Image. RadioGroup je skupina RadioButtonů. S touto komponentou se pracuje obdobně jako s komponentou ComboBox, protože také obsahuje vlastnosti Items a ItemIndex. Komponenta Image je komponenta, jejíž součástí je Canvas, neboli plátno, na které budeme kreslit. Samozřejmě bychom mohli kreslit čistě na formulář, ale při pohybu formulářem by se nakreslený objekt porušil.

Proto je tu komponenta Image, která nakreslený objekt zachová při jakékoliv manipulaci s formulářem.

Jak už bylo psáno, v tomto příkladě budeme využívat přetěžování metod. Když přetěžujeme metody, tak v jedné třídě máme deklarováno více metod se stejným názvem, lišící se ve vstupních parametrech. Na konci každé přetížené metody je klíčové slovo **overload**, který dáváme jasně najevo, že se jedná o přetíženou metodu. Přetížením metod jsme si zpřehlednili zdrojový kód a nemusíme vymýšlet velké množství názvů metod, stačí jen jeden název. Při volání přetížené metody program pozná podle vstupních parametrů, jakou metodu konkrétně voláme.

Na obrázku 30 je ukázka vytvoření třídy TMalovani, ve které jsou deklarované přetížené metody kresli. Jak je z tohoto obrázku zřejmé, každá z metod kresli má jiné vstupní parametry a za každou z přetížených metod je klíčové slovo overload. Po vytvoření třídy deklarujeme objekt (instanci třídy) a následně ji konstruktorem vytvoříme v události formuláře onCreate. Následně objekty zrušíme metodou Free v události formuláře onClose.

```
  TMalovani = class

   public
   Procedure kresli(a: integer);overload; //tato procedura kresli kruh
   procedure kresli(a,b:integer);overload; //tato procedura kresli obdelnik
   procedure kresli(a,b,c:integer);overload; //tato procedura kreslí trojuhelnik
  end;

var
  Form4: TForm4;
  Kresli: TMalovani;

implementation

{$R *.dfm}

//.........................DEFINICE PROCEDUR KRESLI.........................

procedure TMalovani.kresli(a: Integer);   //procedura, která kreslí kruh
begin

   Form4.Image1.Canvas.Ellipse(Form4.Image1.ClientWidth-a,Form4.Image1.ClientWidth-a,a,a);    //kreslíme k
end;

procedure TMalovani.kresli(a: Integer; b: Integer); //procedura, která kreslí čtverec
begin
    form4.Image1.Canvas.Rectangle(Form4.Image1.ClientWidth-a,Form4.Image1.ClientWidth-b,a,b); //kreslíme ob
end;

procedure TMalovani.kresli(a: Integer; b: Integer; c:integer);//procedura, která kreslí trojuhelnik
begin
   form4.Image1.Canvas.Polygon( [Point(Form4.Image1.ClientWidth-a,a), Point(b,b), Point(105,c)]);//kreslím
end;
```

Obrázek 30 - Program Tvary zdrojový kód

Na obrázku 30 jsou zřejmé i definice přetížených metod. Jak už bylo psáno, tyto metody budou kreslit různé tvary na kreslící plátno (Canvas). V Delphi máme možnosti kreslit několik tvarů. Prvním z nich je tvar Ellipse. Ten kreslí elipsy nebo kruh. Druhým tvarem je Rectangle, který kreslí obdélník nebo čtverec. Dále můžeme kreslit Polygon, kdy neurčujeme tvar, ale souřadnice vrcholů kresleného tvaru. Naše první přetížená metoda kresli má jen jeden vstupní parametr, takže tuto metodu můžeme využít pro kreslení kruhu. Druhá přetížená metoda kresli má již dva vstupní parametry, které využijeme pro kreslení obdélníka a třetí přetížená metoda kresli má tři vstupní parametry, které využijeme při kreslení trojúhelníka. Kruh kreslíme příkazem *Image1.Canvas.Ellipse(X1,Y1,X2,Y2);*. Kdybychom si představili kruh ve čtverci, tak souřadnice X1 a Y2 jsou souřadnice levého horního rohu čtverce a souřadnice X2 a Y2 jsou souřadnice pravého dolního rohu čtverce. Obdélník kreslíme příkazem *Image1.Canvas.Rectangle (X1,Y1,X2,Y2)* kdy souřadnice X1,Y1,X2 a Y2 znamenají to samé jako u kreslení kruhu či elipsy. Delphi bohužel neobsahuje žádnou funkci pro kreslení trojúhelníku, proto musíme využít

funkci Polygon ([Point (x,y), Point (x,y), Point (x,y)]), kdy do každého bodu píšeme souřadnice pro každý vrchol tvaru.

Jak už bylo psáno, v programu se nachází komponenta RadioGroup, která nám bude sloužit k přepínání tvarů. V události OnClick této komponenty budeme podmíněnými příkazy zjišťovat jaký radioButton je označený, a to přes ověřování vlastnosti ItemIndex. Podle označeného RadioButtonu budeme volat příslušné metody kresli. Například, bude-li označen RadioButton „Kruh“ (ItemIndex = 0), voláme proceduru Kresli s jedním parametrem. Jako parametry metod kresli, budeme vkládat hodnoty ze SpinEditů, aby uživatel mohl měnit velikost příslušných tvarů. Pokud budeme jen volat příslušné metody pro kreslení tvarů, tak se nám budou kreslit přes sebe. Proto musíme před novým kreslením plátno smazat. To uděláme tak, že budeme kreslit obdélník nebo čtverec ve velikosti plátna ještě před zavoláním příslušné metody kresli.

Dále je v tomto příkladě požadován číselný vstup, kterým bude uživatel nastavovat velikost příslušného tvaru. To je řešeno třemi SpinEdity. V události OnChange každého SpinEditu opět rozlišujeme, který z tvarů je vybrán v komponentě RadioGroup, a podle toho opět mažeme plátno a voláme příslušnou metodu kresli.

5 ZÁVĚR

Embarcadero Delphi je grafické vývojové prostředí, které je určeno pro tvorbu aplikací v programovacím jazyce Object Pascal. V roce 1995 vydala firma Borlad svou první verzi vývojového prostředí Delphi a to pod názvem Delphi1. Od roku 2009 vychází všechny verze vývojového prostředí Delphi pod firmou Embarcadero, která svou poslední verzi vydala v dubnu v roce 2014 pod názvem Delphi XE6. Toto vývojové prostředí se vyznačuje multiplatformností. Lze tedy vytvářet aplikace pro operační systémy Windows, MAC, iOS a Android. Delphi se ve výchozím nastavení skládá z hlavního panelu a panelu nástrojů. Dále se skládá z nástrojů Tool Palette,

Project Manager, Object Inspector a z nástroje Structure. Každý z těchto nástrojů programátorovi pomáhá při vytváření programu.

Vývojové prostředí Embarcadero Delphi pracuje s objektově orientovaným programovacím jazykem Object Pascal. Program se tedy skládá z objektů (instancí tříd), které mají své vlastnosti (atributy) a chování (metody). Objekty vytváříme ze tříd, které můžeme chápat jako šablonu pro objekty, které se z nich vytváří. Lze využít i vlastností OOP, mezi které patří zapouzdření, dědičnost a polymorfizmus. Zapouzdření využijeme ve chvíli, kdy budeme chtít atributy a metody objektu ochránit před neoprávněným přístupem. Dědičnost využíváme, když potřebujeme vytvořit novou třídu, která je rozšířením jiné třídy. Nemusíme tedy deklarovat všechny atributy a metody od znova, pouze je zdědíme od rodičovské třídy. Polymorfizmus je nejvyšší stupeň OOP. Pod polymorfizmus spadá překrývání a přetěžování metod. Překrýváním metod předefinujeme zděděnou metodu a přetěžování metod znamená, že v jedné třídě máme více metod se stejným názvem, lišící se ve vstupních parametrech. Používáme tedy jeden název pro více metod.

Se sadou praktických příkladů se student naučí pracovat ve vývojovém prostředí Embarcadero Delphi. Konkrétně se student naučí pracovat s projektem (vytváření, ukládání, přejmenovávání apod.), naučí se využívat nástroje jako Tool Pallete, Object Inspector, Structure a Project Manager. Z těchto příkladů se student naučí vytvářet vzhled pomocí formulářů a komponent. Dále se seznámí s různými druhy komponent, s jejich vlastnostmi a naučí se tyto komponenty využívat. Student pozná využití objektově orientovaného programování na praktických příkladech, kdy každý z příkladu demonstruje jednu z vlastností OOP.

6 RESUME

Tato bakalářská práce by měla sloužit jako učební pomůcka pro studenty, kteří se zajímají o vývojové prostředím Embarcadero Delphi. Student se naučí pracovat s projektem, využívat nástroje Delphi a pracovat s kompomponenty a s jejich vlastnostmi. Dále tato práce popisuje základní principy objektově orientovaného programování (OOP) v jazyce Object Pascal. Součástí bakalářské práce je sada vlastních praktických příkladů dle zásad OOP, která studenta seznámí s vývojovým prostředím a s objektově orientovaným programováním v praxi.

This bachelor‘s thesis should serve as a learing material for a students, who are interested in Integrated Development Environment Embarcadero Delphi. Student learns how to work with the projekt, use the tools of Delphi and work with components and with their attributes. Furthermore this thesis describes basic principle of object oriented programming (OOP) in language Object Pascal. The part of this bachelor‘s thesis is the set of my own practical examples according to rules of OOP which student inform about IDE and about object oriented programming (OOP) in practice.

7 SEZNAM POUŽITÉ LITERATURY A PRAMENŮ

PÍSEK, Slavoj. *Delphi-začínáme programovat: podrobný průvodce začínajícího uživatele*. 2., upr. A ktualit. Vyd. Praha: Grada, 2002, 325 s. ;. ISBN 80-247-0547-8.

KEOGH, James Edward a Mario GIANNINI. *OOP bez předchozích znalostí: průvodce pro samouky*. Brno: Computer Press, 2006, 222 s. ISBN 80-251-0973-9.

8 SEZNAM POUŽITÝCH INTERNETOVÝCH ZDROJŮ

Verze Delphi. *Delphi.cz: Komunitní portál Delphi* [online]. [cit. 2014-01-10]. Dostupné z: http://delphi.cz/page/Verze-Delphi.aspx

Embarcadero Technologies. *Slideshare* [online]. San Francisco: Embarcadero [cit. 2014-01-10]. Dostupné z: http://www.slideshare.net/embarcaderotechnet?utm_campaign=profiletracking&utm_medium=sssite&utm_source=ssslideview

Rad Studio. *Embarcadero reseller partner* [online]. 2014 [cit. 2014-06-14]. Dostupné z: http://www.embt.cz/cs/produkty/16-rad-studio

Delphi XE6. *Embarcadero* [online]. 2014 [cit. 2014-06-14]. Dostupné z: http://www.embarcadero.com/products/delphi

Filename Extension in Delphi. *About.com: Delphi* [online]. 2014 [cit. 2014-06-19]. Dostupné z:http://delphi.about.com/od/beginners/a/aa032800a.htm

9 SEZNAM OBRÁZKŮ A TABULEK

9.1 Seznam obrázků

9.2 Seznam tabulek

10 PŘÍLOHY

Přílohy se nacházejí na přiloženém CD. Toto CD obsahuje bakalářskou práci ve formátu PDF a docx. Dále obsahuje sadu vlastních příkladů, ze kterých se skládá praktická část bakalářské práce.

1 Bakalářská práce_Vojtěch Marton.docx
2 Bakalářská práce_Vojtěch Marton.pdf
3 Příklad 1
4 Příklad 2
5 Příklad 3
6 Příklad 4
7 Příklad 5
8 Příklad 6
9 Příklad 7
10 Příklad 8