ZÁPADOČESKÁ UNIVERZITA V PLZNI
FAKULTA PEDAGOGICKÁ
KATEDRA VÝPOČETNÍ A DIDAKTICKÉ TECHNIKY

PŘÍPRAVA KOMPONENT PRO E-KURZ PROGRAMOVÁNÍ 2

BAKALÁŘSKÁ PRÁCE

Lenka Krblichová
Informatika se zaměřením na vzdělávání

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

Plzeň, 2015

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

V Plzni, 15. duben 2015

.....

vlastnoruční podpis

Poděkování

Ráda bych poděkovala vedoucímu bakalářské práce Mgr. Tomáši Přibáňovi, Ph.D. za odborné vedení, vstřícnost při konzultacích, cenné rady a připomínky při zpracování mé bakalářské práce.

OBSAH

SEZNAM ZKRATEK

PGM1P	Programování 1 - název předmětu
PGM2B	Programování 2 - název předmětu
DLL	Dynamic-link library - dynamicky linkované knihovny
RAD	Rapid Application Development - rychlý vývoj aplikací
BDE	Borland Database Engine - program pro správu databází
XML	Extensible Markup Language - rozšiřitelný značkovací jazyk
CLX	Component Library for Cross-Platform
PNG	Grafický formát souborů
OOP	Object Oriented Programming - objektově orientované programování

ÚVOD

V předmětu Programování 1 (PGM1P) vyučovaného na katedře výpočetní a didaktické techniky Fakulty pedagogické Západočeské univerzity v Plzni se studenti naučili programovat ve vývojovém prostředí Delphi a jeho jazyku Object Pascal. Programovali zde příklady s výstupem do konzole. V předmětu Programování 2 (PGM2B), který na předchozí předmět navazuje, naučené techniky dále rozvíjejí s důrazem na zásady objektově orientovaného programování. Pro studenty je jeden z hlavních rozdílů mezi předměty v tom, že v rámci předmětu PGM2B využívají k tvorbě vlastních miniaplikací tzv. komponenty. Komponenty v prostředí Delphi jsou v podstatě předpřipravené balíčky či jakési malé programy, které vykonávají určitou činnost, jako jsou např. formuláře, tlačítka, plátna pro kreslení, a mnoho dalších. Díky tomu můžeme snadněji připravit uživatelsky přívětivé programy. Základem tvorby programů v Delphi je to, že si vybereme jednotlivé komponenty, které bychom chtěli mít v programu, a vývojové prostředí se za nás postará o základní část kódu spojenou s vybranými komponentami. My jen rozšíříme kód například o vlastnosti komponent, funkce, procedury, obsluhy událostí, aby daný program dělal to, co od něj požadujeme.

Cílem této bakalářské práce je představit pokročilejší techniky programování, které jsou obsahem předmětu Programování 2. Dále budu vytvářet sadu vhodných příkladů v programu Turbo Delphi, které budou reprezentovat tento předmět a budou užitečné pro samostatnou přípravu studentů na hodiny. Vrcholem této sady příkladů bude simulátor mobilního telefonu. Nejprve budu programovat samostatné programy (kalkulačku, hodiny, kreslící plátno, kontakty,…), které výsledně spojím do závěrečného komplexního programu. Volba programu Turbo Delphi je především z toho důvodu, jelikož se s ním studenti setkají ve škole během výuky. Pro animace budu používat nejnovější verzi Delphi, což je Delphi XE7, aby měli studenti možnost seznámit se i s nejnovější verzí vývojového prostředí. Vybrala jsem si tuto bakalářskou práci zaměřenou na vývojové prostředí Delphi s programovacím jazykem Object Pascal, jelikož mi přijde tento jazyk srozumitelnější pro začínající programátory než jiné programovací jazyky, které se vyučují například na středních školách.

1 HISTORIE

Vytvoření integrovaného grafického vývojového prostředí Delphi předcházel dlouhý vývoj jazyka Pascal. Samotný jazyk Pascal byl vytvořen v roce 1969 a původně byl určen hlavně k výuce programování. [1] Programovací jazyk Pascal byl postupně vylepšován, až roku 1983 firma Borland představila jazyk Turbo Pascal. [2] Tento jazyk se stal standardem. [3] „S verzí Turbo Pascal 5.5 přidala firma Borland podporu pro objektové orientované programování. Později se Borland rozhodl, že je potřeba komplikovanějších vlastností a začal pracovat na Delphi, přičemž vycházel z návrhu jazyka Object Pascal společnosti Apple. V raných verzích nazýval tento jazyk rovněž Object Pascal, později jméno změnil na programovací jazyk Delphi.“ [4]

Velkou předností Delphi v té době bylo to, že „umožňovaly vizuálně navrhnout grafické uživatelské rozhraní, na jehož základě je automaticky vytvářena kostra zdrojového kódu, což výrazně urychlovalo vývojový cyklus. Programování v Delphi je z velké části založeno na použití komponent. Komponenta je malý program (balíček funkcí), který vykonává určitou činnost (například zobrazuje text nebo obrázky, přehrává multimédia, komunikuje s databází, zprostředkovává FTP přenos, atd.). Velkou předností Delphi proti některým konkurenčním produktům jsou právě knihovny komponent, které jsou jejich součástí. Dodávané komponenty významně usnadňují tvorbu aplikací. Další komponenty lze stáhnout z internetu nebo přímo vytvářet vlastní komponenty.“ [5]

V době, kdy se na trhu objevil operační systém Windows, přidali vývojáři vizuální prostředí a vznikla první verze Delphi, a to Delphi 1. [6]

1.1 DELPHI 1

Tato 16ti-bitová verze byla poprvé představena v roce 1995. [6; 7] Delphi 1 se vyznačovalo vlastním překladačem, vizuálním prostředím, přístupem k databázím a dynamickými knihovnami DLL. [3; 6] V souvislosti s touto verzí vznikl nový pojem RAD (Rapid Application Development), což ve volném překladu znamená rychlý vývoj aplikací. [3]

1.2 DELPHI 2

S nástupem Windows 95 se musela vytvořit nová verze Delphi, která by umožňovala vytvářet rychlejší aplikace pomocí 32bitového překladače. Tato verze byla primárně určena pro Windows 95 a Windows NT a byla představena v roce 1996. [3] Změny byly také ve funkcích. Mezi ně patřily například vylepšení a rozšíření knihovny objektů, lepší podpora databází, snazší zacházení s řetězci, podpora technologie OLE a nový datový typ Variant. [3; 6] Novinkou byla i vizuální dědičnost formulářů. Důležitou částí Delphi 2 bylo ale také zachovat kompatibilitu se starší verzí. [3]

1.3 DELPHI 3

Delphi 3 se objevilo v roce 1997 a stejně jako v předchozí verzi přibyla vylepšení. Mezi tato vylepšení patřila podpora ladění knihoven DLL, usnadnění práce s technologiemi COM a ActiveX a také vývoj aplikací pro web. [3] Novinkou byla technologie Code Insight a Web Broker. [6] Technologie Code Insight je našeptávač kódu.

1.4 DELPHI 4

Čtvrtá verze, vydaná r. 1998 byla zaměřena na usnadnění tvorby aplikací a zdokonalení ladícího programu. Nově jsou vytvořeny plovoucí panely nástrojů, které lze ukotvit tam, kam uživatel potřebuje. Změnila se i navigace ve zdrojovém kódu a také doplňování tříd. Objevuje se zde i nástroj Module Explorer, metoda přetěžování nebo otevřená pole. [3] Novinkou je podpora aplikací CORBA a nové verze BDE (Borland Database Engine). [6] Architektura CORBA nám zajišťuje mezivrstvu síťové komunikace.

1.5 DELPHI 5

V roce 1999 byla zveřejněna nová verze. Došlo k vylepšení editoru, podpory databází a ladícího programu. Byly vytvořeny nové vlastnosti komponent VCL a také přibyla další skupina komponent označená jako Internet Express, která slouží pro práci s jazykem XML. Navíc se objevil nový software pro týmový vývoj s názvem TeamSource. S rozšířením počtu uživatelů a z důvodu uživatelské přívětivosti byli vývojáři nuceni přidat také nástroj, který zajišťuje překlad aplikací do jiných jazyků. [6; 3; 7]

1.6 DELPHI 6

Delphi 6 vyšlo v roce 2001 a s sebou přineslo spoustu novinek. Mezi ně patřila podpora jazyka XML v databázových aplikacích, možnost vytváření aplikací, které fungují na více platformách a nové navigační okno, jež umožňovalo procházet ve stromové struktuře soubory v projektu. [6] V této verzi se vývojáři zaměřili na kompatibilitu s vývojovým prostředím Kylix. Toto prostředí je velmi podobné Delphi s tím rozdílem, že je vytvořeno na tvorbu aplikací v operačním systému Linux. Aby byla zajištěna kompatibilita, vyvinuli novou knihovnu CLX (Component Library for Cross-Platform) a s tím i skupiny komponent VisualCLX, DataCLX a NetCLX. Kromě vytváření nových knihoven a komponent věnovali vývojáři svůj čas, také vylepšení již zaběhnutých vlastností funkcí. Jednalo se o zlepšení knihovny VCL, integrovaného vývojového prostředí, ladícího programu a knihovny RTL. [3]

1.7 DELPHI 7 A 8

Delphi 7, které bylo spuštěno v roce 2002, bylo zaměřené na vývoj aplikací pro operační systém Linux a také na vývoj webových aplikací. Nově se zaměřilo na schéma Windows XP a Enterprise Class Reporting. Enterprise Class Reporting je sada nástrojů pro vývojáře, která pomáhá vytvářet víceplatformní tiskové sestavy. Delphi 8 bylo představeno v roce 2003 a bylo primárně jen o aplikacích pro technologii .NET. [7; 8]

1.8 DELPHI 2005 - 2010

Tyto verze byly vydávány každý rok od roku 2004 do roku 2009, kromě roku 2007, kdy nebyla vydána žádná verze. [8] Delphi 2005 se spojilo s jazykem C#, dále se vylepšila podpora pro .NET a VCL. Novými nástroji se staly Help Insight, Sync Edit, Error Insight a správa historie. [9] Delphi 2006 bylo zaměřeno na přetěžování operátorů, statické metody a vlastnosti. Nově přibyly komponenty TGridPanel, TFlowPanel, TTrayIcon, vylepšilo se vyhledávání v paletě nástrojů, podpora pro MySQL a podpora Unicode pro dbExpress. Nová verze Delphi 2007 se zaměřila na podporu operačního systému Windows Vista, nové dbExpress 4 a vylepšení nástroje FastCode. V roce 2008 vychází Delphi 2009. V této verzi je vylepšena podpora PNG a Unicode. Také je zde nová sada komponent Ribbon Controls a technologie DataSnap pro síťovou komunikaci. Ve verzi 2010 se vývojáři zaměřili na podporu operačního systému Windows 7 a na podporu dotyků a gest pro dotyková zařízení. Také byl zaveden nový nástroj IDEInsight a vylepšené formátování kódu. [8]

1.9 DELPHI XE - XE7

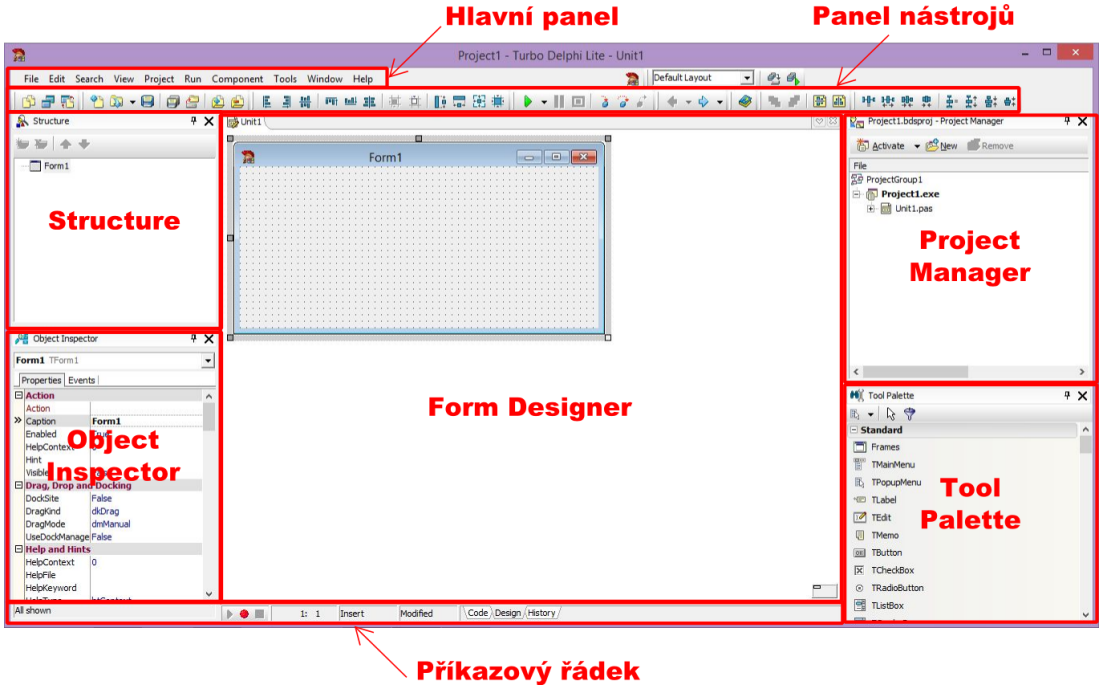
Delphi XE vyšlo v roce 2010 a představilo nástroje CodeSite a AQTime, vylepšení dbExpress a komponenty TTimeZone, TBinaryReader a TBinaryWriter. [8; 2] V roce 2011 byla představena verze Delphi XE2 zaměřená na 64bitové systémy a podporu iOS a Mac OSX. Novými nástroji byly Documentation Insight pro XML dokumentaci, FireMonkey a Live Bindings. Další verze Delphi XE3 (r. 2012) podporuje virtuální klávesnice a obsahuje metrostyly pro VCL a FMX. V roce 2013 vyšly dvě verze Delphi, a to XE4 a XE5. Obě tyto verze se zaměřují na dotyková zařízení. Ve verzi XE4 jsou změny jako návrhář formulářů pro mobilní zařízení, Apple iOS, virtuální klávesnice pro iOS a komponenty TWebBrowser a TListView. Verze XE5 je hlavně o systému Android. Je zde podpora rovnou pro několik verzí. Jsou to JellyBean, IceCream, Sandwitch a Gingerbread. Nově je zde komponenta pro upozornění, podpora pro styly iOS7 a konfigurovatelný návrh formuláře pro mobilní zařízení. Nejaktuálnějšími verzemi jsou XE6 a XE7, které byly vydány v roce 2014. Je zde možno vytvářet aplikace pro platbu pomocí komponenty TInAppPurchase. Další nové komponenty jsou TTaskBar a TMultiView. V Delphi XE7 je

navíc multiplatformní knihovna pro paralelní programování (System.Threading) a podpora nové verze Android - Kit Kat4. [8; 2]

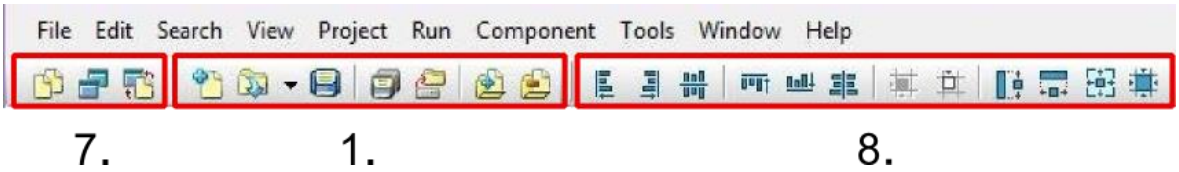
2 POPIS PROSTŘEDÍ DELPHI A JEHO FUNKCE

2.1 POPIS PROSTŘEDÍ

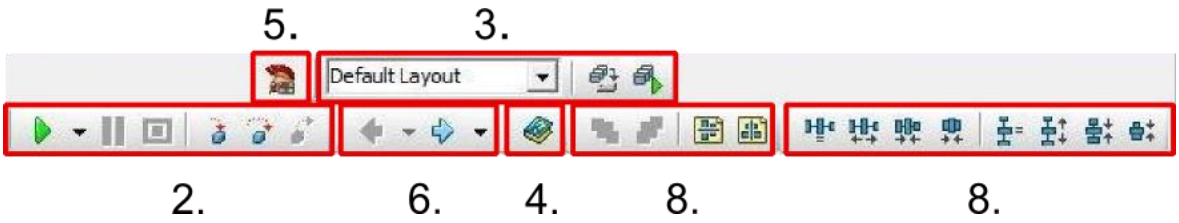
Prostředí Delphi se dá vizuálně rozdělit do několika částí, jak je naznačeno na obrázku č. 1. Největší část okna zabírá *Form Designer* pro návrh a tvorbu formulářů. V návrhu formuláře můžeme libovolně upravit velikost pomocí uchopení a tažení za body na okraji. Na formulář můžeme umisťovat komponenty z části *Tool Palette (Paleta Komponent)*, která se nachází v pravém dolním rohu. Jednotlivé vlastnosti a události komponent použitých na formuláři upravujeme v panelu *Object Inspector* v levém dolním rohu. Přehled již použitých komponent se nám zobrazuje ve stromové struktuře v panelu *Structure*, jinak také nazývaném *Object Treeview*. Tento panel se nachází přímo nad panelem *Object Inspector*. Pokud bychom chtěli v projektu použít více formulářů, zobrazí se nám jejich struktura v levém horním rohu okna v panelu *Project Manager*. V horní části okna se nachází *Hlavní panel*. Tento panel obsahuje klasickou nabídku, jako například *File, Edit, View, Project, Run* a další. Dále obsahuje lištu, kde nalezneme *Toolbars (Panel Nástrojů)* (viz obrázek 2 a 3). Tuto lištu si můžeme upravit podle vlastní potřeby. Upravení lišty nalezneme v záložce *View* → Toolbars → *Customize*. Pokud bychom si nastavení změnili, můžeme ho vrátit zpět do původního nastavení na tom samém místě po kliknutí na tlačítko Reset. Na liště *Toolbars* nalezneme panely *Standard, Debug, Desktop, Custom, Align, Spacing, Position, Personality, Browser a View*. Pokud bychom se chtěli přepnout ze zobrazení formuláře do psaní kódu, provedeme to pomocí tlačítka *Toggle Form/Unit* v panelu *FileListBox*, nebo klávesovou zkratkou F12. [10; 6]



Obrázek 1: Prostředí Delphi



Obrázek 2: Toolbars 1



Obrázek 3: Toolbars 2

Toolbars (Panel nástrojů):

1. Standard - obsahuje klasické položky pro tvorbu souborů (Nové, Uložit, Otevřít, …).
2. Debug - tento panel obsahuje tlačítka pro ladění programu.
3. Desktop - nastavení zobrazení okna programu. Lze zde nastavit, zda mají být jednotlivé panely samostatné nebo tvořit jeden celek.
4. Custom – zde je pouze tlačítko nápovědy, ale je možné si tento panel rozšířit o libovolné ikony.
5. Personality - obsahuje ikonu verze programu.
6. Browser - v tomto panelu jsou tlačítka Vpřed a Vzad, která slouží pro procházení kódu.
7. View - pro zobrazení návrhu formuláře nebo editoru kódu.
8. Align, Spacing, Position - umožňují pozicovat komponenty na formuláři.

2.2 USNADNĚNÍ PŘI PSANÍ KÓDU

Psaní kódu nám může usnadnit několik technologií. Jednou z nejčastěji používaných je technologie *Code Insight*. Tato technologie se automaticky vyvolá ve dvou případech. V prvním případě se vyvolá, pokud napíšeme identifikátor a za něj tečku (například *Label1.*). V druhém případě se vyvolá po napsání funkce a otevírací závorky (například *Length(*). Oba dva případy můžeme vyvolat ručně, a to pomocí klávesové zkratky Ctrl+mezerník/Ctrl+Shift+mezerník. Další technologií, která nám pomůže při programování, je *Code Templates*. Tato technologie se nám bude hodit například při psaní cyklů, ale i složitějších deklarací. Použijeme jí napsáním zkratky pro daný příkaz a potvrzením klávesovou zkratkou Ctrl+J. Pokud při psaní kódu zapomeneme, jak byla definovaná určitá deklarace procedury, stačí myší najet na název volané procedury, stisknout klávesu Ctrl a kliknout. Tím přeskočíme v kódu na deklaraci dané funkce. Tato technologie se nazývá *Code Browser*. Pokud bychom se chtěli vrátit zpátky na místo, odkud jsme sem přeskočili, klikneme na šipku Zpět v panelu *Browser*. [6]

2.3 STRUKTURA PROGRAMOVACÍ JEDNOTKY

Programovací jednotka, jinak také Unit má vlastní strukturu kódu. Vlastní program má na začátku klíčové slovo *unit* a za ním název programu (defaultně Unit1). Dále je kód rozdělen na dvě části *interface* a *implementation*. V části interface se definují veškeré knihovny, které potřebujeme ke správné funkčnosti programu. Tyto knihovny jsou uvedeny za klíčovým slovem *uses*. Dále definujeme jednotlivé třídy, jejich metody a proměnné za slovem *type*. Druhá část kódu se nachází v části *implementation* a v ní zapisujeme zbývající kód, tj. definice jednotlivých metod, ať už vlastních nebo těch, které jsou vytvořeny nějakou událostí.

2.4 ULOŽENÍ PROJEKTU

Při ukládání vlastních programů je výhodné každý program uložit do vlastní složky pojmenované podle toho, co daný program dělá. Je to z toho důvodu, že Delphi neukládají jen jeden jediný soubor, ale několik souborů s různými údaji.

Seznam přípon souborů: [6]

- *.dpr - soubor s vlastním projektem. Je zde nastaveno kolik máme formulářů, jednotlivé unity a počáteční inicializace.
- *.pas - soubor s vlastním kódem programu.
- *.dfm - soubor s vlastnostmi použitých formulářů a komponent.
- *.res - soubor se zdroji Windows pro přiřazení ikony.
- *.exe - soubor s vlastním spustitelným programem. Tento program lze spustit na jakémkoliv počítači bez ohledu na to, zda je tam nainstalováno programovací prostředí Delphi.
- *. dcu - soubor, který obsahuje již zkompilované Unity

2.5 DEBUGGER

Debugger je velmi užitečný pomocník při psaní programů. Pokud zjistíme chybu při běhu programu nebo chceme zjistit, že námi naprogramovaná posloupnost příkazů provádí to, co od ní očekáváme, můžeme si kód tzv. „odkrokovat“. Na řádek, od kterého chceme spustit krokování, umístíme tzv. *Breakpoint*. Umístíme ho tak, že klikneme myší na začátek dané řádky. Před číslem řádky se nám zobrazí červený bod. Jestliže chceme kontrolovat hodnotu určité proměnné, vložíme jí do *Watch Listu*, a to pomocí nabídky *Run → Add Watch*, nebo stisknutím klávesové zkratky Ctrl+F5. Nyní spustíme program pomocí volby v hlavním panelu *Run → Run*, funkční klávesou F9 nebo tlačítkem na panelu *Debug*. Program běží do doby, než narazí na *Breakpoint*. Od této chvíle začínáme program krokovat. Pokud chceme program procházet opravdu řádku po řádce, použijeme funkční klávesu F7, tlačítko *Trace Into* na panelu *Debug* nebo v hlavním panelu *Run → Trace Into*. Pokud bychom se nechtěli při krokování vnořovat do procedur a funkcí, použijeme funkční klávesu F8 nebo tlačítko *Step Over*, které je umístěné na stejných místech jako tlačítko předchozí. Jestliže máme problém v cyklu a nechceme ho procházet od začátku, když víme, že chyba je až v druhé polovině cyklu, můžeme si nastavit *Breakpoint* pomocí logické podmínky. Umístíme si *Breakpoint* na daný řádek a klikneme na něj pravým tlačítkem myši. Vybereme možnost *Breakpoints properties* a do řádku *Condition* zapíšeme logickou podmínku (např. *a = 10*, pokud chceme, aby nám cyklus zastavil, když proměnná *a* je rovna deseti). Debugger nám zastaví program, až když dosáhne dané podmínky. Krokování vypneme tlačítkem *Program Reset* na panelu *Debug*, klávesovou zkratkou Ctrl+F2 nebo možností v hlavním panelu *Run → Program Reset*. Tímto tlačítkem nám neskončí jen krokování, ale také se ukončí spuštěný program. [6]

3 POKROČILÉ TECHNIKY PROGRAMOVÁNÍ

3.1 OBJEKTOVĚ ORIENTOVANÝ PŘÍSTUP K PROGRAMOVÁNÍ

Objektově orientovaný přístup (OOP) můžeme definovat jako „snahu modelovat při řešení úloh principy reálného světa“. [10] Tuto definici si můžeme jednoduše vysvětlit tak, že každá věc v našem životě je určitý objekt, ať už je to dům, auto, zvíře nebo člověk. Každý objekt má své vlastnosti a určité chování. Například auto má vlastnosti jako počet dveří, barvu, velikost nádrže a pod jeho chování můžeme zařadit jízdu dopředu, dozadu, zastavení a další. Pomocí těchto jednoduchých objektů můžeme sestavit složitější celky. Mezi základní principy OOP patří objekty, třídy, zapouzdření, dědičnost a polymorfismus.

3.1.1 OBJEKTY

Jak už bylo řečeno objekt má určité vlastnosti a chování. Vlastnosti objektu můžeme definovat také jako charakteristiku nebo popis. Vlastnosti určujeme pomocí jednotlivých proměnných. O chování objektu můžeme říct, že je to také reakce na nějakou událost. Ty nastavujeme pomocí procedur a funkcí. Procedury a funkce souhrnně nazýváme metody a proměnné uvnitř třídy označujeme jako atributy. [6] Definování jednotlivých objektů se provádí ve třídě.

3.1.2 TŘÍDY

Třídu definujeme pomocí klíčového slova *class*. Je zvykem, že pro lepší přehlednost v kódu začínají jména tříd velkým písmenem „T“ (např. *type TAuto = class*) a končí klíčovým slovem *end*. Každá třída může obsahovat jednotlivé atributy a metody. Pokud bychom chtěli vytvořit instanci třídy, což je konkrétní objekt dané třídy, musíme zavolat speciální metodu - konstruktor. Využívá se zejména při vytvoření instance třídy s počátečními parametry. Můžeme si také vytvořit vlastní konstruktor, nebo zavolat vestavěný konstruktor *Create*. Pro správné ukončení instance třídy je nutné zavolat destruktor, což je speciální metoda třídy. Destruktor nikdy nevoláme přímo, ale pomocí metody *Free*. [6] Jako příklad instance třídy může být určité reálné auto, ze třídy TAuto, které má nastavené atributy (například pět dveří, červenou barvu, 45l velikost nádrže). Pro

vytvoření instance zavoláme konstruktor Auto1 := *TAuto.Create(5, clRed, 45);*. Pro zrušení instance třídy stačí zavolat *Auto1.Free;*. Praktická ukázka použití je v příkladu Posun obrazců.

3.1.3 ZAPOUZDŘENÍ

Zapouzdření je určitý princip, který nám pomáhá chránit data před nechtěnou změnou z vnějšku. Data ochráníme tím, že zajistíme, aby se k atributům objektu nepřistupovalo přímo, ale například zavoláním metody. Díky tomu získáme přehlednější a snadno udržitelný kód. Dalším způsobem ochrany je použití tzv. specifikátorů přístupů. Označují se klíčovými slovy *public*, *private* a u dědičnosti *protected*. [6]

- Private - atributy a metody definované v dané třídě jsou vidět jen v té třídě, ve které jsou nadefinované.
- Public - atributy a metody definované v dané třídě jsou vidět i v ostatních třídách.
- Protected – k atributům a metodám dané třídy může přistupovat nejen ta daná třída, ale i třídy, které jsou z ní odděděné.

3.1.4 DĚDIČNOST

Dědičnost je další princip objektově orientovaného přístupu k programování, kdy dědíme atributy a metody z jedné třídy do druhé. Odděděnou třídu pak můžeme rozšířit, upravit nebo specifikovat. Jedním z učebnicových příkladů je třída auto, kdy auta dělíme na osobní a nákladní. Osobní auto má atributy počet dveří, počet osob a nákladní auto má navíc atribut nosnost. Vytvoření odděděné třídy se provede zápisem *type TNakladniAuto = class(TOsobniAuto)*.

Při dědění se vytváří určitá vazba mezi původní třídou (rodičem) a odděděnou třídou (potomkem). Vždy se dědí pouze jedním směrem a to od rodiče k potomkovi.

Dělení dědičnosti podle vazby: [11]

- Jednoduchá dědičnost - vazba jeden rodič - jeden potomek. Potomek má vždy jen jednoho rodiče, ale rodič může mít více potomků.
- Vícenásobná dědičnost - vazba více rodičů - jeden potomek. Potomek může mít více rodičů a tím může získat i více atributů a metod, jelikož je dědí od více nadřazených tříd. Tato vazba nelze použít v jazyce Object Pascal.

3.1.5 POLYMORFISMUS

Polymorfismus nám umožňuje, aby se v programu vyskytovalo více tříd, které budou mít metodu se stejným názvem, ale odlišným kódem uvnitř této metody. U polymorfismu se používá také označení přetížení metod, které nám umožňuje uvnitř jedné třídy použít více metod se stejným názvem, ale jinými parametry. Změna parametrů se ale netýká výstupní hodnoty metody. Metodu, kterou chceme přetížit, označíme klíčovým slovem *overload*.

Vazby: [6]

- Statické (časné) - tyto vazby vznikají při překladu programu. V tomto okamžiku se určuje, která metoda respektive z jaké třídy se metoda použije a během programu to již nelze změnit.
- Dynamické (pozdní) - vazby vznikají až za běhu programu. Používají se za pomoci virtuálních metod, které určují, že daná metoda existuje, ale to, ze které třídy bude metoda volána, se určí až později. Metodu, kterou chceme překrýt, označíme klíčovým slovem *virtual* a metodu, která ji bude překrývat, označíme klíčovým slovem *override*. Pokud bychom chtěli překrýt konstruktor z nadřazené třídy, použijeme klíčové slovo *inherited*. Tyto dynamické vazby jsou vhodné, pokud za běhu programu budeme měnit třídu, kterou chceme právě používat. Názorná ukázka je v příkladu Posun obrazců.

3.2 DYNAMICKÉ DATOVÉ TYPY A STRUKTURY

Dynamické proměnné jsou takové proměnné, u kterých se jejich velikost a umístění v paměti vytváří až za běhu programu. Běžně v programech používáme statické proměnné, které se vytváří při spuštění programu a které můžeme snadno definovat. Při využití dynamických proměnných nám tyto proměnné nezabírají místo v paměti. Navíc ne vždy při spuštění programu víme, jakou velikost bude mít daná proměnná.

3.2.1 POLE

Dynamické pole je datový typ, který se používá v případě, že velikost pole nastavujeme až při běhu programu. Velikost pole se nastavuje pomocí příkazu *SetLength()* a při práci s ním můžeme použít funkce *SizeOf* pro zjištění velikosti pole a funkce *Low* a *High* pro zjištění nejnižšího a nejvyššího indexu pole.[6] Příkladem může být pole, do kterého chceme uložit záznamy ze souboru. Velikost pole si určíme podle počtu záznamů v souboru a poté budeme soubor procházet a jednotlivé záznamy ukládat do pole. Tento příklad je použit i v programu Kontakty. Jiným příkladem může být dynamicky vytvářené pole tlačítek. Toto pole se vytvoří například při spuštění programu a po ukončení programu se pole zruší. Tento příklad je v programu Kalkulačka. Zdrojové kódy obou příkladů jsou v příloze A.

3.2.2 ZÁZNAM (RECORD)

Záznam nám umožňuje sloučit spolu související proměnné různých datových typů do jedné proměnné typu záznam. To se dá využít, pokud potřebujeme ukládat data o určitém objektu. Například použijeme záznam „Osoba“, který bude mít nastaveny položky záznamu „jméno“, „příjmení“ a „věk“. Jméno a příjmení bude *String* a věk bude *Integer*.

Deklarace toho záznamu:

```
type Osoba = record
   jmeno : String;
   prijmeni : String;
   vek : Integer;
end;
```

Pro přiřazování hodnot do těchto proměnných se používá zápis:

```
Osoba.jmeno := 'Jan';
Osoba.prijmeni := 'Novák';
Osoba.vek := 30;
```

Pokud bychom chtěli pracovat s více položkami záznamu, je výhodné používat příkaz *with*, abychom nemuseli neustále psát, o jaký záznam se jedná.

```
with Osoba do
begin
   jmeno := 'Jan';
   prijmeni := 'Novák';
   vek := 30;
end;
```

Použití záznamů je ukázáno v příkladu Kontakty, jehož zdrojový kód je v příloze A.

3.2.3 UKAZATEL (POINTER)

Ukazatel je proměnná, která neobsahuje data, ale pouze adresu umístění v paměti. [6] Může ukazovat na proměnné typu *String*, *Integer*, *Byte*, *Pole*, *Záznam* a další. Je zvykem, že se veškeré názvy proměnných typu ukazatel píší s „P“ na začátku pro odlišení od ostatních proměnných. Deklarace ukazatele je například: *type UkazatelNaInt = ^Integer;* poté deklarace proměnné *PCislo : UkazatelNaInt;*. Pokud bychom chtěli zjistit adresu, na které je proměnná uložena použijeme příkaz *UkazatelNaInt := Addr(název_proměnné);* nebo *UkazatelNaInt := @název_proměnné;*. Jestliže chceme zapisovat hodnotu do proměnné, máme dvě možnosti. Můžeme zapisovat pomocí proměnné ukazatele *PCislo^ := 2;* nebo pomocí nové proměnné, která bude datového typu *Integer* a bude propojena s proměnnou ukazatele. Pro druhý případ bude zápis *cislo : Integer; PCislo := Addr(cislo);* a *cislo := 2;*. Pro práci s ukazateli můžeme využít dva páry metod. Jsou to metody *New* a *Dispose*, které použijeme, pokud nevíme jakou velikost určit proměnné. Tyto metody to obstarají za nás. Velikost, která se nastaví pro danou proměnnou, se určuje podle datového typu této proměnné. Zápis může být *New(PCislo);* a *Dispose(PCislo);*. Druhý pár metod *GetMem* a *FreeMem* velikost proměnné určit neumí a proto ji musíme nadefinovat zápisem *GetMem(PCislo,SizeOf(Integer));* *FreeMem(PCislo,SizeOf(Integer));*. Po zrušení proměnné metodou *Dispose* nebo *FreeMem* je zapotřebí ještě změnit adresu v ukazateli, aby se stále neukazovalo na místo, kde již

není žádná proměnná. To se provede příkazem *NIL* (Not In List). Tento příkaz se přiřazuje do proměnné vždy po použití metody pro zrušení. Zápis je *PCislo := NIL;*. [6; 12]

Ukazatele můžeme rozdělit do dvou skupin: [12]

- Typizovaný - ukazatel, který je proměnnou určitého datového typu (*String*, *Integer*, *Char*, …).
- Netypizovaný - ukazatel, který je proměnnou datového typu *Pointer*. Tento ukazatel může být jakéhokoliv datového typu, ale před použitím ho musíme přetypovat.

3.2.4 SEZNAM

Seznam můžeme definovat jako „řadu záznamů, které jsou v paměti dynamicky alokovány, a každý z nich obsahuje ukazatel na následující položku v seznamu“. [12]

Rozdělení seznamů: [12]

- Jednoduchý spojový seznam - seznam, ve kterém můžeme procházet pouze jedním směrem, většinou dopředu. V seznamu je tedy uložen ukazatel na další prvek.
- Dvousměrný spojový seznam - seznam, ve kterém můžeme procházet záznamy dopředu i zpět. Obsahuje ukazatele na předchozí i na další prvek.

4 PRAKTICKÁ ČÁST

Jedním z cílů této bakalářské práce je vytvořit sadu příkladů, vhodnou k procvičení a přípravě na hodiny předmětu PGM2B. Jelikož výsledným komplexním programem bude simulátor mobilního telefonu, je zapotřebí nejprve vytvořit jednotlivé aplikace, které poté umístíme do simulátoru. Níže je popsáno zadání a postup programování jednotlivých aplikací. Kódy všech aplikací naleznete v příloze A.

4.1 HODINY (BUDÍK, STOPKY, ČASOVAČ)

4.1.1 ZADÁNÍ

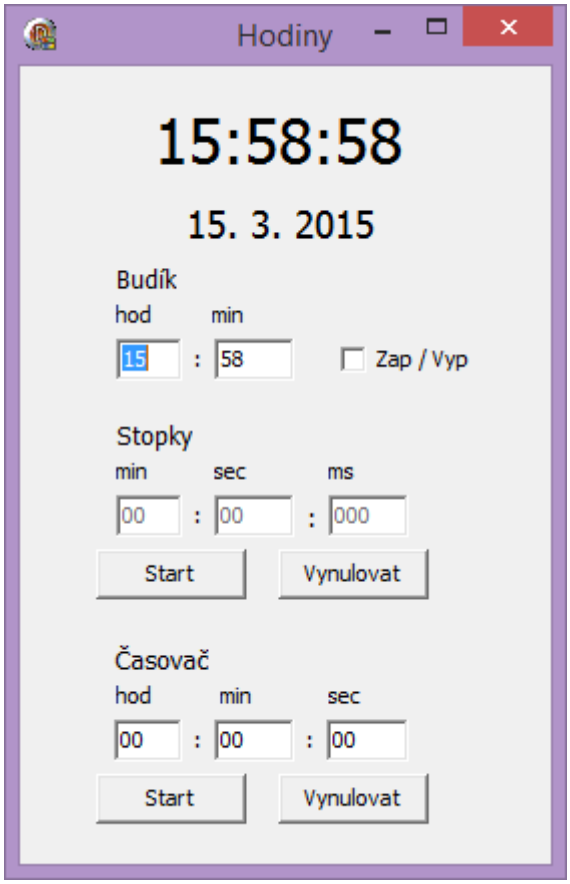
Vytvořte program, ve kterém bude aktuální čas ve formátu hh:min:sec a aktuální datum. U času zajistěte zobrazení vteřin v reálném čase. Dále naprogramujte budík, který lze zapnout či vypnout podle potřeby a nastavit na určitou hodinu a minutu. Ve zvolený čas budík zazvoní. Stopky se budou zobrazovat ve formátu min:sec:ms. Během spuštění je možné je zastavit a poté vynulovat. Jako poslední naprogramujte časovač neboli minutku. Bude zde možné zvolit počet hodin, minut a vteřin, po kterých časovač zazvoní. Zajistěte, aby se dal časovač spustit, zastavit a také vynulovat.

4.1.2 CÍLE

Studenti si na tomto příkladu procvičí práci s komponentou Timer. Také si zde vyzkouší práci s datovými a časovými funkcemi.

4.1.3 POPIS ŘEŠENÍ

Tento příklad může mít dva způsoby řešení. Jelikož má tento program provádět několik funkcí, budu jednotlivé funkce nazývat miniaplikacemi (hodiny, budík, časovač a stopky). Tyto miniaplikace můžeme vytvořit na jednom formuláři, nebo pro každou miniaplikaci můžeme vytvořit vlastní formulář. Oba tyto příklady naleznete v příloze A.

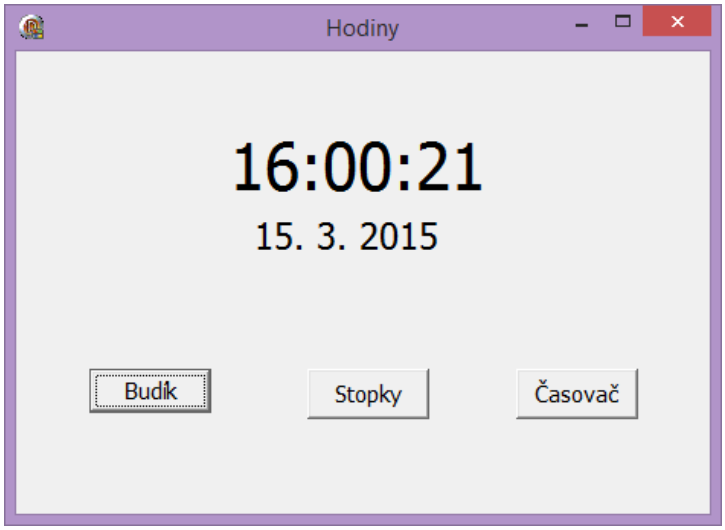


Obrázek 4: Hodiny - jeden formulář

Hodiny

Použité komponenty: *2x TLabel, TTimer.*

Do formuláře si přetáhneme tyto komponenty z *Tool Palette* a nastavíme si událost formuláře *OnShow*, kterou nalezneme v sekci *Visual*. V této události nastavíme do obou komponent typu *Label* aktuální čas a datum pomocí funkcí *Time* a *Date*. Abychom měli každou vteřinu aktuální čas, nastavíme si komponentu *Timer* na hodnotu 1000 v sekci *Miscellaneous* pod názvem *Interval*. Interval komponenty je v milisekundách. U této komponenty musíme také nastavit její jedinou událost *OnTimer*. Do této vlastnosti se přidává metoda, která se má opakovat za daný časový okamžik. V našem případě to bude procedura *FormShow*. Komponenta *Timer* funguje jen tehdy, pokud má nastavenou vlastnost *Enabled* na hodnotu *True*.



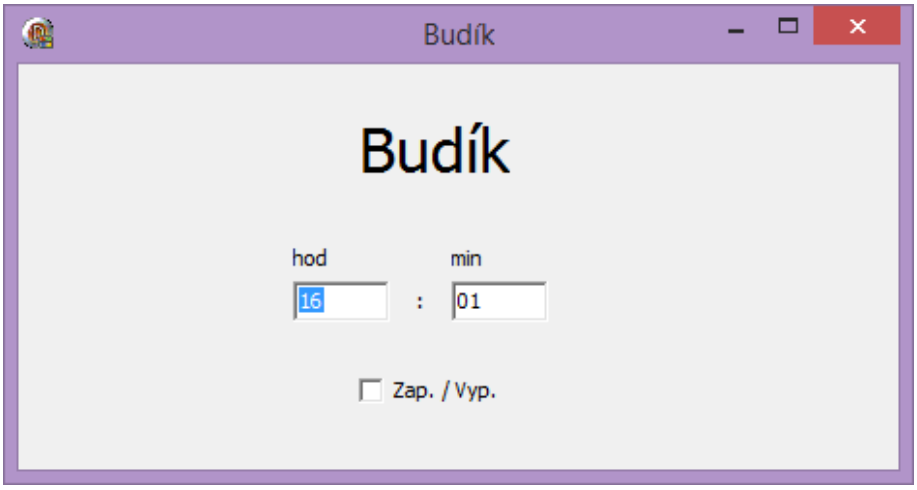
Obrázek 5: Hodiny

Budík

Použité komponenty: *3x TLabel, 2x TEdit, TTimer, TCheckBox.*

V této miniaplikaci nám komponenty typu *Label* budou sloužit pouze jako popisky. Do komponent typu *Edit* bude uživatel zapisovat hodiny a minuty, kdy má budík zazvonit. Pro zapnutí a vypnutí budíku použijeme komponentu *CheckBox*. Komponentu *Timer* nastavíme stejně jako v předchozím příkladu, tedy hodnotu vlastnosti *Interval* nastavíme na 1000 milisekund. Do události této komponenty přidáme proceduru komponenty *CheckBox* s názvem *OnClick*.

Jelikož chceme, aby se nám po spuštění formuláře zobrazil v komponentách *Edit* aktuální čas (hodiny a minuty), vytvoříme si událost formuláře *OnShow*. V této proceduře přiřadíme do komponent aktuální čas pomocí příkazu *FormatDateTime('hh', Now);* pro hodiny a pro minuty nám bude stačit změnit *'hh'* na *'nn'*. Nyní nastavíme proceduru *CheckBox1Click*. Jako první v této proceduře budeme testovat, zda byla komponenta *CheckBox* zaškrtnuta nebo ne. Nás zajímá, jen pokud byla zaškrtnuta. Poté si do proměnných uložíme uživatelem zadaný čas z komponent *Edit* a do jiných proměnných si uložíme aktuální čas. Tyto proměnné pak porovnáme v podmínce, zda jsou stejné. Pokud jsou stejné, využijeme funkci *Windows.Beep (700,900);* ve které nastavujeme frekvenci tónu a délku trvání v milisekundách.



Obrázek 6: Budík

Stopky

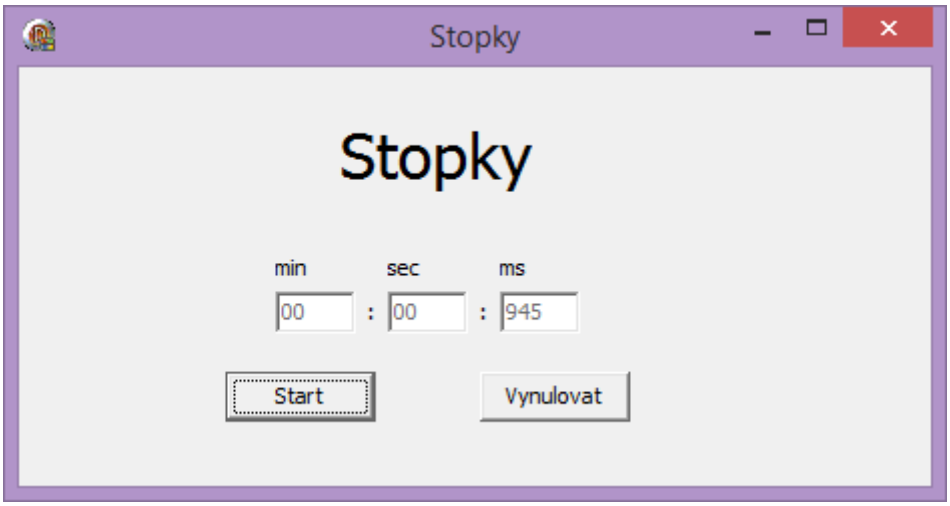
Použité komponenty: *4x TLabel, 3x TEdit, TTimer, 2x TButton.*

Komponenty typu *Label* využijeme opět pro popsání jednotlivých komponent typu *Edit*. Jelikož nám stopky počítají čas od spuštění k zastavení, do jednotlivých komponent typu *Edit* budeme ukládat milisekundy, sekundy a minuty. Do těchto komponent nechceme, aby nám uživatel jakýmkoliv způsobem zasahoval, proto nastavíme jejich vlastnost *Enabled* na *False*. Dále zde budeme mít dvě tlačítka. První tlačítko bude sloužit ke spuštění stopek, bude mít proto popisek „Start“. Stopky můžeme ovšem při běhu pozastavit, proto po spuštění stopek změní popisek na „Stop“. Druhé tlačítko nám bude sloužit k vynulování stopek po jejich zastavení. Proto při běhu stopek bude mít toto tlačítko nastavenou vlastnost *Enabled* na *False*. Komponenta *Timer* bude mít nastavenou vlastnost *Interval* na 10 ms a v události bude přiřazena procedura *Timer1Timer*, kterou získáme tím, že dvakrát „poklikáme“ do políčka této události.

Nejprve si naprogramujeme událost *OnClick* druhého tlačítka pro nulování. Zde budeme nulovat komponenty *Edit* a to tak, že jim nastavíme hodnoty na nulu. Toto nulování se provede pouze v případě, kdy hodnota *Enabled* komponenty *Timer* je nastavena na *False*. Tedy pouze tehdy, když jsou stopky zastaveny. V proceduře události *OnClick* prvního tlačítka se testuje, zda komponenta *Timer* běží a podle toho se nastavují vlastnosti tlačítek, *Timeru* a také se zde přičítá čas do celkového času, v případě, že stopky byly

pozastaveny. V proceduře *Timer* budeme zapisovat čas stopek do komponent *Edit*. Tedy čas, který uběhl od spuštění stopek do teď. Jelikož potřebujeme čas rozdělit do jednotlivých *Editů* podle toho, zda jsou to minuty, vteřiny nebo milisekundy, musíme si vytvořit proměnné a do těch si vypočítat správný čas, podle toho o jakou jednotku času se jedná.

```
procedure TForm3.Timer1Timer(Sender: TObject);
begin
if Timer1.enabled=truethen
begin
rozdilMiliSec := MilliSecondsBetween(StartStopky, Time);
rozdilMiliSec := rozdilMiliSec + CelkemStopky;
rozdilSec := rozdilMiliSec div 1000;   //celočíselné dělení
rozdilMiliSec := rozdilMiliSecmod 1000;   //zbytek
ifrozdilSec>=60 then
begin
rozdilMin := rozdilSec div 60;
rozdilSec := rozdilSecmod 60;
        end;
      Edit1.Text := Format('%.2d',[rozdilMin]);
      Edit2.Text := Format('%.2d',[rozdilSec]);
      Edit3.Text := Format('%.3d',[rozdilMiliSec]);
   end;
end;
```



Obrázek 7: Stopky

Časovač

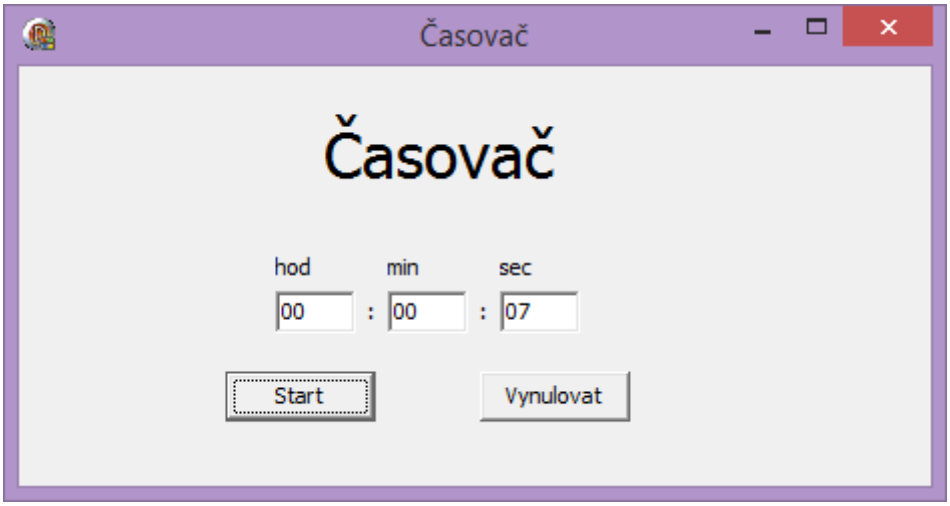
Použité komponenty: *4x TLabel, 3x TEdit, TTimer, 2x TButton.*

Časovač si lze představit jako kuchyňskou minutku. Nastavíme si určitý čas a spustíme. Čas se bude odpočítávat a až bude na nule, tak pípne.

Jednotky našeho časovače budou hodiny, minuty a sekundy. To budou i popisky jednotlivých komponent *Edit*. Tlačítka budou fungovat stejně jako v předchozím příkladu. Komponentu *Timer* nastavíme na 100 milisekund a přiřadíme jí opět její vlastní proceduru *Timer1Timer.*

Druhé tlačítko bude mít událost *OnClick* stejnou jako v minulém příkladu. Toto tlačítko tedy bude provádět nulování komponent *Edit*. Procedura události *OnClick* prvního tlačítka testuje, zda běží *Timer* a podle toho popisuje tlačítka a nastavuje vlastnost *Enabled* komponenty *Timer*. Také se zde ukládá čas, který si nadefinoval uživatel, do proměnných. V proceduře *Timeru* se testuje, zda čas už vypršel. Pokud čas nevypršel, vypočte se zbývající čas na hodiny, minuty a vteřiny a zapíše se do komponent typu *Edit*.

```
procedure TForm4.Timer1Timer(Sender: TObject);
var  pom, rozdil_odpocet_min, rozdil_odpocet_sec, rozdil_odpocet_hod : integer;
begin
CelkemOdpocet := SecondsBetween(StartOdpocet,0) + cas_odpoctu;
ifCelkemOdpocet = SecondsBetween(Time, 0) then
begin
    Edit3.Text := '00';
    Edit2.Text := '00';
    Edit1.Text := '00';
    Timer1.Enabled := false;
    Button1.Caption := 'Start';
    Button2.enabled := true;
    Edit3.Repaint;
Windows.Beep(700,900);
  end
else
begin
    pom := SecondsBetween(Time,StartOdpocet);
rozdil_odpocet_hod := (cas_odpoctu - pom) div 3600;
    pom := (cas_odpoctu - pom) mod 3600;
rozdil_odpocet_min := pom div 60;
rozdil_odpocet_sec := pom mod 60;
    Edit1.Text := Format('%.2d', [rozdil_odpocet_hod]);
    Edit2.Text := Format('%.2d', [rozdil_odpocet_min]);
    Edit3.Text := Format('%.2d', [rozdil_odpocet_sec]);
  end;
end;
```



Obrázek 8: Časovač

4.1.4 MODIFIKACE

U tohoto příkladu můžeme modifikovat miniaplikaci budík o přidání dalších upozornění a také možnost odložení upozornění.

4.2 KALKULAČKA

4.2.1 ZADÁNÍ

Vytvořte kalkulačku, která bude umět počítat základní matematické operace. Kalkulačka bude obsahovat dynamicky vytvářená tlačítka. Výstup z kalkulačky bude rozdělen do dvou řádků, kde první řádek bude klasický displej, na kterém bude zobrazen aktuální výsledek. Druhý řádek bude zobrazovat lištu s operandy a operátory, které byly použity ve výpočtu.

4.2.2 CÍLE

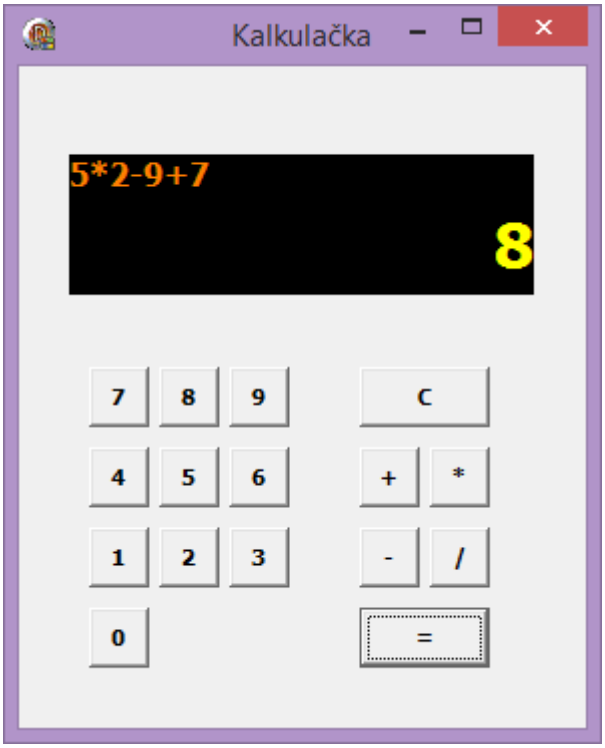
Na tomto příkladu si studenti vyzkouší dynamické vytváření objektů a jejich následné odstranění z paměti v případě, že se s objekty dále nebude pracovat. Také si zde procvičí práci s poli.

4.2.3 POPIS ŘEŠENÍ

Použité komponenty: *2x TLabel, TButton*

V tomto programu budeme používat dynamicky vytvářená tlačítka. Jelikož budeme potřebovat pro otevření formuláře tyto tlačítka vytvořit, budeme pracovat v proceduře události *FormShow*. Pro vytvoření tlačítek si zavedeme proměnnou typu pole, které nastavíme velikost pomocí procedury *SetLength.* Velikost tohoto pole je podle počtu tlačítek, tj. 10 (číslice 0-9). Budeme procházet cyklus *for,* ve kterém bude probíhat vytvoření tlačítek, nastavení jejich popisku, velikosti a pozice ve formuláři. Také jim nesmíme zapomenout nastavit událost *OnClick,* po stisku tlačítka. Takto máme vytvořená tlačítka. Pro použití kalkulačky potřebujeme také vytvořit tlačítka operací. Ta vytvoříme stejně jako tlačítka operandů, jen pro ně deklarujeme nové a menší pole. Pro pojmenování tlačítek si vytvoříme funkci, ve které si pomocí podmíněného příkazu *case* definujeme, jaký název tlačítka se vytvoří v daném kroku cyklu.

V proceduře *OnClick* po stisku operandu budeme pouze zapisovat danou hodnotu tlačítka do obou komponent typu *Label*. Procedura po stisku operace bude trochu složitější. Podmíněným příkazem *case* rozlišíme, jaké znaménko operace bylo stisknuto a dále se provede výpočet. Následně zde budeme porovnávat, co bylo stisknuto za znaménko operace. Pokud byly operace sčítání, odčítání, násobení nebo dělení, tak se zapíše postup do lišty s použitými operandy a operátory. Pokud bylo stisknuto tlačítko pro výsledek, výsledek se zobrazí na displeji. Pokud bychom chtěli počítání vynulovat a stiskli tlačítko „C“, nastaví se nám do komponent a pomocné proměnné nula. Jelikož jsme dynamicky vytvářeli tlačítka, potřebujeme je při zavření programu také uvolnit z paměti. To provedeme tak, že na každé tlačítko zavoláme metodu *Free.*



Obrázek 9: Kalkulačka

4.2.4 MODIFIKACE

U tohoto příkladu je možné doplnit omezení řádku s hodnotami a operátory podle šířky dané komponenty a po zaplnění data v řádku vymazat a pokračovat od začátku. Také je možné dodělat další funkce kalkulačky jako například mocnina, odmocnina, procenta.

4.3 KRESLENÍ MYŠÍ

4.3.1 ZADÁNÍ

Vytvořte plátno, na kterém bude možné kreslit myší čáry, kruhy a čtverce. Bude zde možnost změny barvy a tloušťky pro všechny objekty a velikosti pro kruhy a čtverce. Vytvořený obrázek bude možné uložit do souboru. Před uložením si uživatel zvolí název souboru. Pokud se obrázek nepovede, bude zde možnost smazání obrázku z plátna.

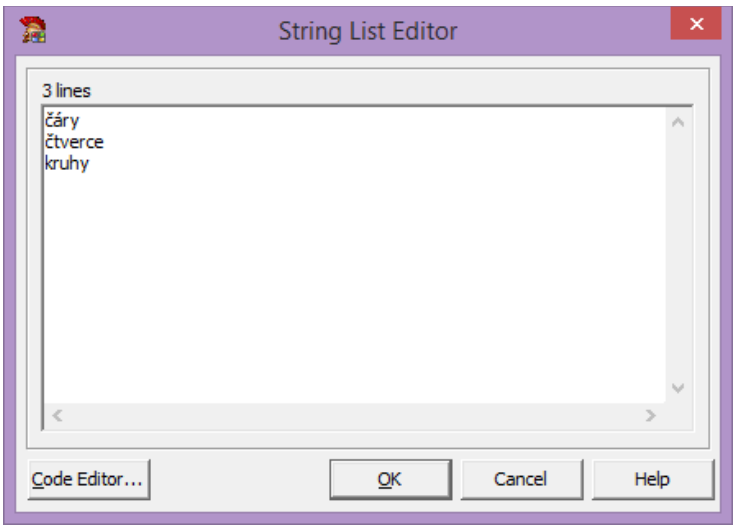
4.3.2 CÍLE

V tomto příkladu se studenti naučí vytvářet vlastní třídy a konstruktory. Také si vyzkouší volání destruktorů.

4.3.3 POPIS ŘEŠENÍ

Použité komponenty: *TImage, 2x TLabel, 3x TButton, TEdit, TRadioGroup, 2x TSpinEdit, TColorDialog.*

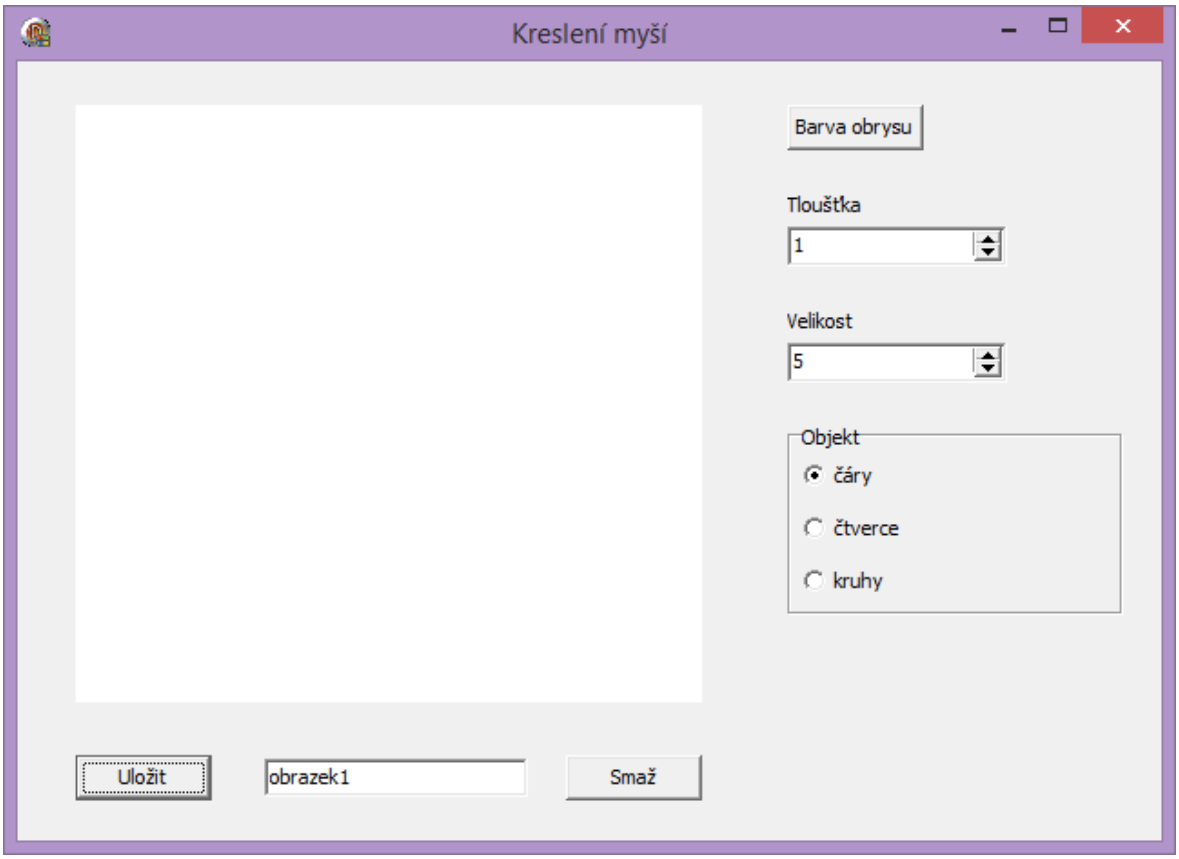
Do formuláře si umístíme všechny komponenty a upravíme u nich popisky *Caption*. U komponenty *Edit,* zobrazený text upravíme ve vlastnosti *Text*. Do této komponenty budeme zapisovat název obrázku, který budeme chtít uložit. V tomto příkladu budeme mít tlačítka „Uložit“, „Smaž“ a „Barva obrysu“. V komponentě *RadioGroup* si budeme vybírat tvar, který budeme kreslit (tj. čára, kruh, čtverec). Jednotlivé volby tvarů nastavíme v komponentě v sekci *Localizable* pod názvem *Items*.



Obrázek 10: Nastavení položek v komponentě RadioGroup

Pomocí komponent *SpinEdit* budeme nastavovat tloušťku čáry a velikost obrazců. Nastavíme si počáteční hodnotu *Value* u komponent *SpinEdit* a také jejich minimální a maximální hodnotu (*MinValue* a *MaxValue*).

Nyní můžeme začít vytvářet vlastní kód. Začneme nastavením jednotlivých tlačítek. Pro provázání komponenty *ColorDialog* s tlačítkem „Barva obrysu“ vytvoříme událost OnClick na tomto tlačítku. Do procedury této události umístíme podmínku, zda bylo zobrazeno dialogové okno barvy. To zajistíme příkazem *if ColorDialog1.Execute() then…* Tlačítko pro uložení obrázku bude obsahovat jednoduchý příkaz *Image1.Picture.SaveToFile(Edit1.Text+'.bmp');*. V tomto příkazu nesmíme zapomenout na přidání přípony k názvu obrázku, jinak bychom si obrázek nemohli zobrazit. Poslední tlačítko nám bude mazat obrázek z plátna. To zařídíme tím, že si plátno vyplníme obdélníkem, který bude stejně velký jako komponenta *Image*. U komponent *SpinEdit* potřebujeme nastavit události *OnChange*, které se nám budou volat po každé změně těchto komponent. V procedurách si tedy do proměnných nastavíme aktuální hodnotu *SpinEditu*.



Obrázek 11: Příklad - Kreslení myší

Pro samotné kreslení na plátno využijeme události komponenty *Image*, které se jmenují *OnMouseDown*, *OnMouseMove* a *OnMouseUp*. Nejprve si nastavíme akce, které se mají provést po stisku tlačítka myši, tedy událost *OnMouseDown*. Pro všechny tvary chceme, nastavit tloušťku a barvu pera. Pro kreslení čar musíme také nastavit počáteční souřadnice, odkud se začne kreslit. Nyní si nastavíme událost *OnMouseMove* pro kreslení čáry, kde bude jeden jediný příkaz a to vykreslení aktuální pozice čáry. Jelikož chceme, aby se čára přestala kreslit poté, co pustíme levé tlačítko myši, vytvoříme si jednoduchou proměnnou typu *Boolean* například s názvem „kresli“. V proceduře *OnMouseDown* tuto proměnnou nastavíme na hodnotu *true*, v proceduře *OnMouseMove* jí otestujeme, zda je *true* a pokud ano, tak se kreslí čára. V proceduře *OnMouseUp* nastavíme proměnnou na *false* a tím zajistíme, aby se čára přestala vykreslovat. Jelikož chceme vytvořit program pomocí objektově orientovaného programování, vytvoříme si pro kreslení kruhu a čtverce nové třídy, které budou obsahovat konstruktor *Create*, pro vytvoření jejich objektů. V tomto konstruktoru budeme nastavovat pouze souřadnice x, y a velikost objektu. V těle těchto konstruktorů budeme vytvářet daný tvar. V proceduře *OnMouseDown* si podmínkou otestujeme, zda je v komponentě *RadioGroup* zaškrtnutý kruh nebo čtverec a podle toho voláme nejprve destruktor dané třídy a poté si zavoláme konstruktor *Create*, který nám vytvoří nový objekt. Jelikož nechceme, aby se nám ihned po nakreslení kruhu nebo čtverce kreslila čára, je nutné použít naši proměnnou „kresli“.

4.3.4 MODIFIKACE

Jako modifikace u tohoto příkladu může být přidání volby velikosti druhé strany pro vytvoření obdélníku a také zašednutí políček podle námi vybraného objektu, tj. pokud si vybereme kreslení kruhu, nebudeme potřebovat vyplnit stranu b. V jiném případě, pokud si vybereme kreslení čáry, není potřeba nastavovat žádnou velikost, jen tloušťku čáry. Další modifikací může být možnost vybrat si, do jaké složky chce uživatel obrázek uložit. Studenti si také mohou vyzkoušet načíst již vytvořený obrázek a ten poté upravovat.

4.4 POSUN OBRAZCŮ

4.4.1 ZADÁNÍ

Vytvořte program, ve kterém si budete moci zvolit obrazec z více možností a nastavit mu barvu. Obrazec bude možné posouvat po plátně podle zadání osy x a y.

4.4.2 CÍLE

Tento příklad je zaměřený na principy objektově orientovaného programování. Studenti si tedy vyzkouší používání objektů, tříd a konstruktorů. Také se zde setkají s dědičností, polymorfismem a zapouzdřením.

4.4.3 POPIS ŘEŠENÍ

Použité komponenty: *TImage, 2x TSpinEdit, 3x TRadioButton, TButton, TColorDialog, 2x TLabel.*

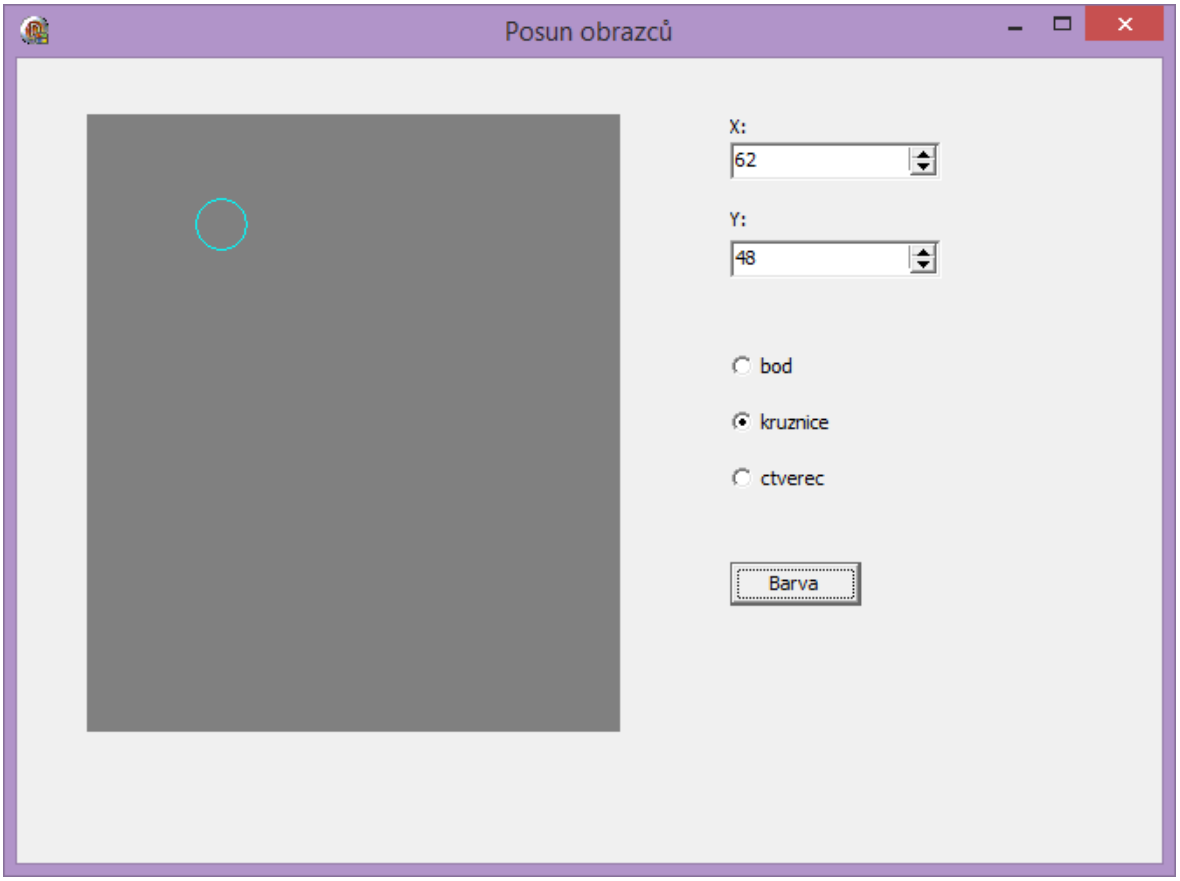
Na plátno si připravíme výše napsané komponenty. Komponenty *SpinEdit* nám budou sloužit ke změně osy x a y. Pro jejich popsání použijeme popisky *Label*. Pomocí komponent *RadioButton* budeme vybírat daný obrazec, který se zobrazí na plátně. Tlačítko *Button* propojíme s komponentou *ColorDialog* pro výběr obrysu obrazce.

V tomto příkladu použijeme dvě programovací jednotky. V jedné bude vizuální zobrazení formuláře a metody, které v sobě budou používat třídy z druhé unity. Nejprve se tedy budeme zabývat druhou unitou. Zde si vytvoříme třídy podle obrazců, které budeme používat v programu. Jako hlavní třídou použijeme „TBod“. V této třídě si nastavíme proměnné pro osu x a y, dvě proměnné pro barvu a také proměnnou plátno, která bude typu *TRect.* Dále si vytvoříme konstruktor *Create* s atributy „new_x“, „new_y“ a „new_barva“. Jelikož si v dalších krocích vytvoříme odděděné třídy od této a budeme chtít konstruktor v nadřazené třídě překrýt, doplníme za zápis konstruktoru klíčové slovo *virtual*. Další metody budou sloužit pro nastavení souřadnice x, souřadnice y, barvy, zobrazení a skrytí objektu. Metody pro zobrazení a skrytí objektu budeme volat při přesunu objektů po osách. Nyní si definujeme konstruktor *Create*. V něm budeme nastavovat privátní proměnné instance třídy, které jsou zadány jako parametry konstruktoru. V procedurách pro nastavení souřadnice x, souřadnice y a barvy přiřadíme

parametry metod do privátních proměnných instance třídy. Procedura „Skryj“ nám nastaví barvu objektu stejnou jako je nastavena barva pozadí a v proceduře „Zobraz“ se bodu nastaví barva vybraná uživatelem. Nyní vytvoříme nové třídy, které budou odděděné od třídy „TBod“. Tyto třídy budou kruh a čtverec. Kruhu přidáme proměnnou pro průměr a čtverec bude mít přidanou proměnnou pro velikost strany. Obě tyto třídy budou mít metody *Create*, „Zobraz“ a „Skryj“, které budou překrývat původní metody. Proto za ně doplníme klíčové slovo *override*. V konstruktorech *Create* budeme překrývat původní konstruktor pomocí klíčového slova *inherited*. Metody „Zobraz“ a „Skryj“ upravíme podle dané třídy.

V tomto okamžiku bychom měli mít vytvořené třídy potřebné pro vytvoření programu. V první unitě si nejdřív ošetříme událost po vytvoření formuláře. Nastavíme si v ní počáteční barvu, vytvoříme instanci třídy „TBod“ a nastavíme maximální hodnotu komponent *SpinEdit*. Dále si ošetříme události po zvolení komponenty *RadioButton*. V této proceduře nejprve zavoláme metodu *Free* pro zrušení původní instance objektu, vytvoříme novou instanci a zavoláme proceduru „Zobraz“ pro daný objekt. Jako poslední si vytvoříme procedury, které budou reagovat na změnu komponent *SpinEdit*. V těchto procedurách vždy skryjeme objekt, nastavíme novou souřadnici podle aktuální hodnoty a zobrazíme objekt.

Zapouzdření jsme v tomto příkladu dosáhli tím, že jsme nastavili proměnné jako privátní a přistupujeme k nim pomocí metod instance třídy. Polymorfismus je v tomto příkladu ukázán na stejném pojmenování několika metod, kdy voláme správnou metodu podle zvoleného objektu.



Obrázek 12: Posun obrazců

4.4.4 MODIFIKACE

Do programu je možné přidat pole pro zvolení průměru kruhu a velikosti čtverce. Také je možné modifikovat čtverec na obdélník a tedy i přidat pole pro zadání druhé strany.

4.5 PRŮZKUMNÍK SOUBORŮ

4.5.1 ZADÁNÍ

Vytvořte program na procházení souborů a adresářů v počítači. V programu bude možnost zvolit diskový oddíl, zobrazit stromovou strukturu adresářů, vypsat soubory ve vybraném adresáři a možnost filtrovat soubory dle jejich typu (textové soubory, obrázky, spustitelné soubory). Po kliknutí na zvolený soubor se zobrazí obsah souboru podle jeho typu (např. text - zobrazení komponenty *Memo*, obrázek - zobrazení obrázku v komponentě *Image*).

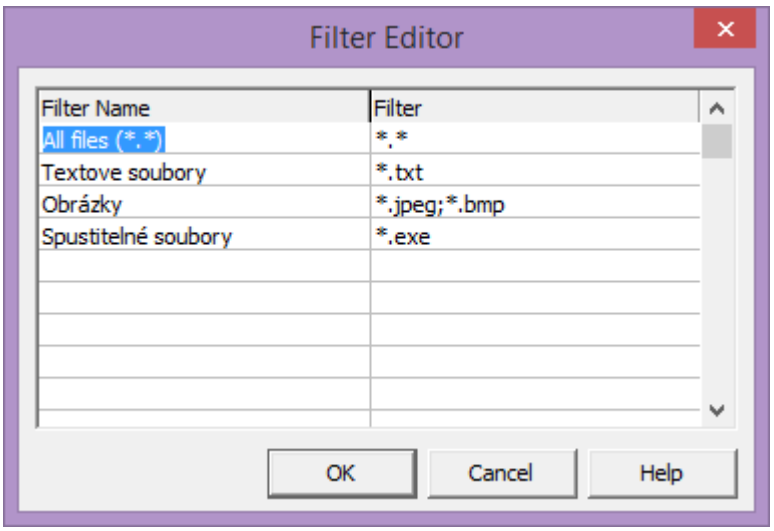
4.5.2 CÍLE

Studenti si na tomto příkladu vyzkouší propojování více komponent mezi sebou a nastavování vlastnosti *Visible* u různých komponent podle toho, kterou komponentu je potřeba zobrazit.

4.5.3 POPIS ŘEŠENÍ

Použité komponenty: *TDriveComboBox, TDirectoryListBox, TFileListBox, TFilterComboBox, TMemo, TImage*.

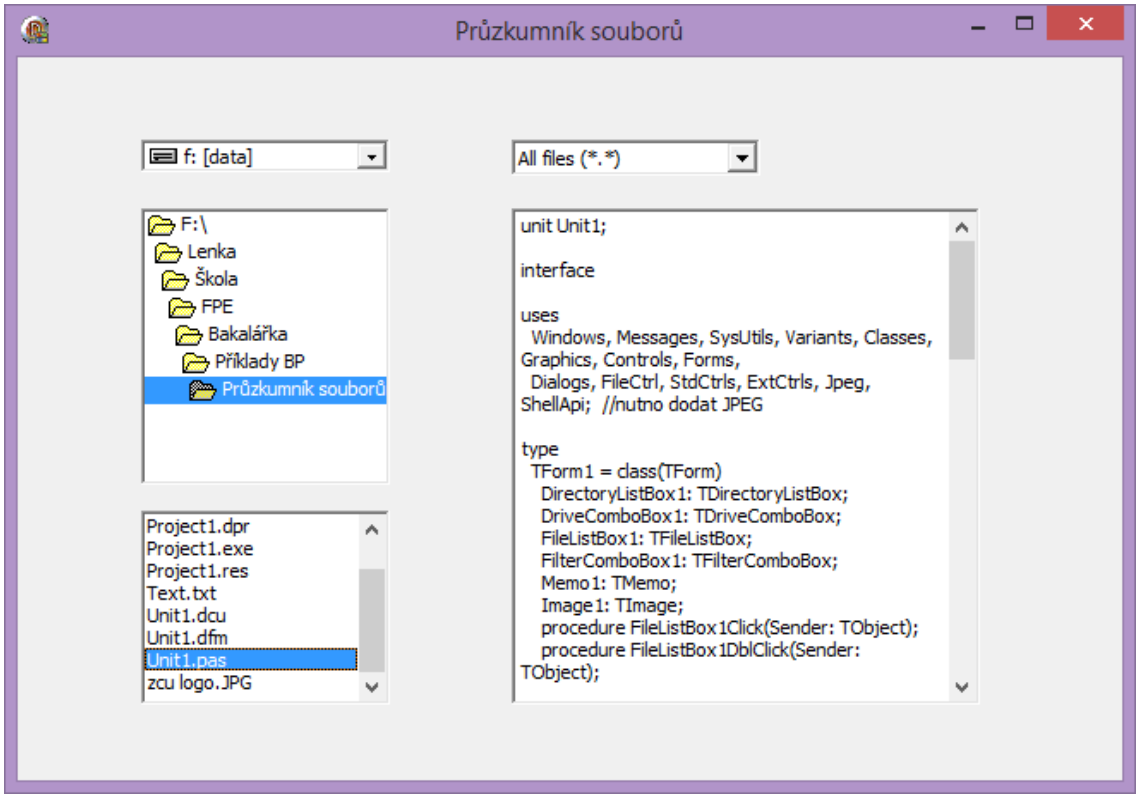
Do formuláře si nejdříve umístíme výše popsané komponenty. Komponenty *DriveComboBox*, *DirectoryListBox*, *FileListBox* a *FilterComboBox* si propojíme pomocí událostí, kde každá z těchto komponent má v části *Linkage* možnost propojení na určitou komponentu. V případě komponenty *DriveComboBox* je to políčko *DirList*, kam připojíme komponentu *DirectoryListBox*. Tímto propojením jsme si zajistili vnitřní komunikaci mezi jednotlivými komponentami. To znamená, že pokud si vybereme disk C, zobrazí se nám pouze adresáře, které jsou na tomto disku. Pokud si dále zvolíme adresář, zobrazí se nám pouze soubory uložené ve zvoleném adresáři. V komponentě *FilterComboBox* si vytvoříme vlastní filtry podle přípon souborů. To se provádí v sekci *Localizable - Filter* (obrázek 13). Do sloupce *Filter Name* si napíšeme název filtru a do sloupce *Filter* danou příponu, před kterou napíšeme hvězdičku pro vyhledávání souborů s jakýmkoliv názvem.



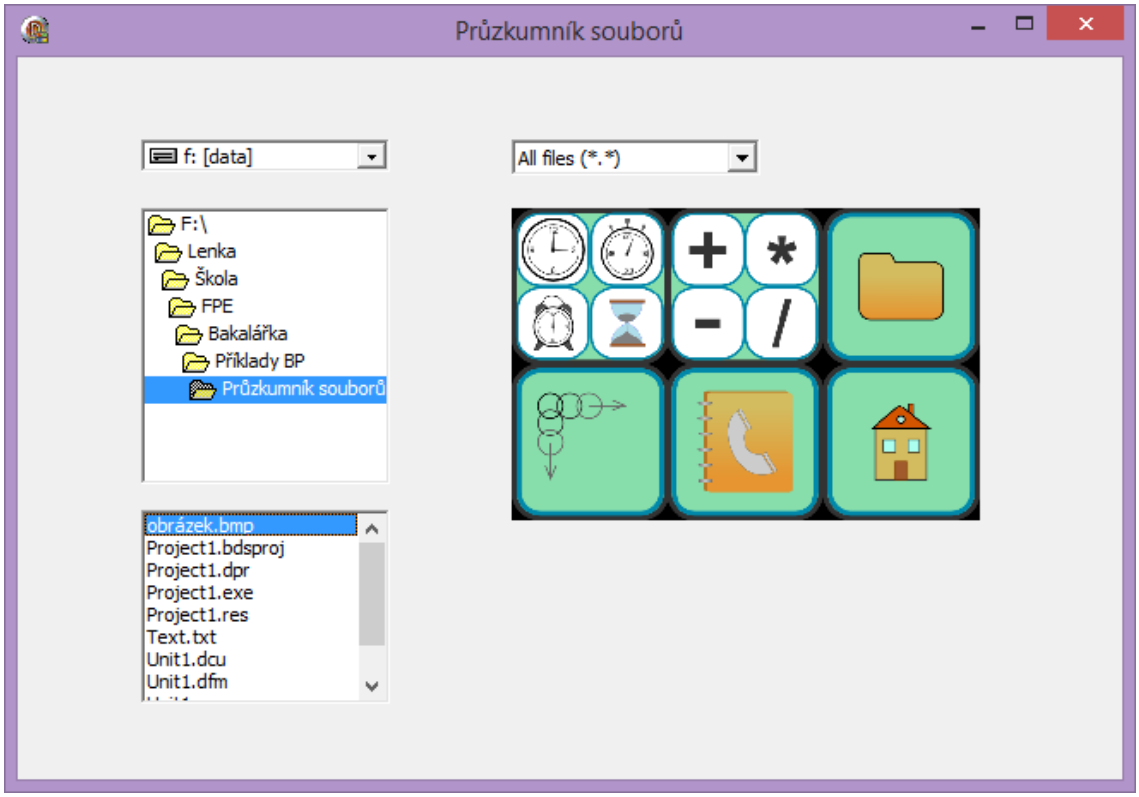
Obrázek 13: Filter Editor

Jelikož víme, že v komponentě Memo budeme chtít zobrazovat i texty, které jsou větší než velikost této komponenty, nastavíme si ve vlastnosti komponenty zobrazení svislého posuvníku. Toto nastavení nalezneme v sekci *Miscellaneous - ScrollBars* pod názvem *ssVertical*.

Teď již máme připravené všechny komponenty a můžeme začít psát kód. V komponentě *FileListBox* si nastavíme událost „po kliknutí“. Nastavení nalezneme v sekci *Input* pod názvem *OnClick*. V této proceduře chceme, aby se po kliknutí zobrazil text v komponentě *Memo*, obrázek v komponentě Image a pokud bychom zvolili spustitelný soubor s příponou .exe, tak by se do komponenty *Memo* vypsal pouze text o tom, že má uživatel na soubor kliknout dvakrát. To vše zajistíme podmínkou podle toho, jaká je přípona vybraného souboru. Komponenty *Memo* a *Image* můžeme mít umístěny vedle sebe, nebo pro lepší přehlednost je můžeme umístit na sebe. Podle toho, kterou komponentu zrovna potřebujeme zobrazit, jim nastavíme vlastnost *Visible*. Pro spuštění souboru s příponou .exe si vytvoříme novou proceduru komponenty *FileListBox*, které nastavíme událost *OnDblClick*. V této proceduře použijeme funkci *ShellExecute*, která zajišťuje spuštění programu. Jelikož tato funkce potřebuje název programu v datovém typu *PAnsiChar*, je nutné stávající název uložený v datovém typu *String* přetypovat na *PAnsiChar*. Celý zdrojový kód tohoto příkladu je umístěn v příloze A pod názvem Průzkumník souborů.



Obrázek 14: Průzkumník souborů s ukázkou textu



Obrázek 15: Průzkumník souborů s ukázkou obrázku

4.5.4 MODIFIKACE

Jako rozšíření může být doplnění filtru o další typy souborů.

4.6 KONTAKTY

4.6.1 ZADÁNÍ

Vytvořte program, pro správu kontaktů. Kontakty bude možné vytvořit, upravit nebo smazat. Kontakt bude obsahovat údaje: jméno, příjmení, telefon a email. Kontakty se budou ukládat do souboru a zároveň budou zobrazené na obrazovce například v komponentě *Listbox*. Seznam kontaktů bude možné řadit vzestupně i sestupně podle jména a bude v něm možné vyhledávat podle příjmení.

4.6.2 CÍLE

Studenti si v tomto příkladu vyzkouší práci s poli, s datovým typem záznam, se souborem a s komponentou *ListBox*. Dále v příkladu budou muset používat řazení a vyhledávání textu.

4.6.3 POPIS ŘEŠENÍ

Použité komponenty: *5xTEdit, 5x TLabel, 4x TButton, TListBox.*

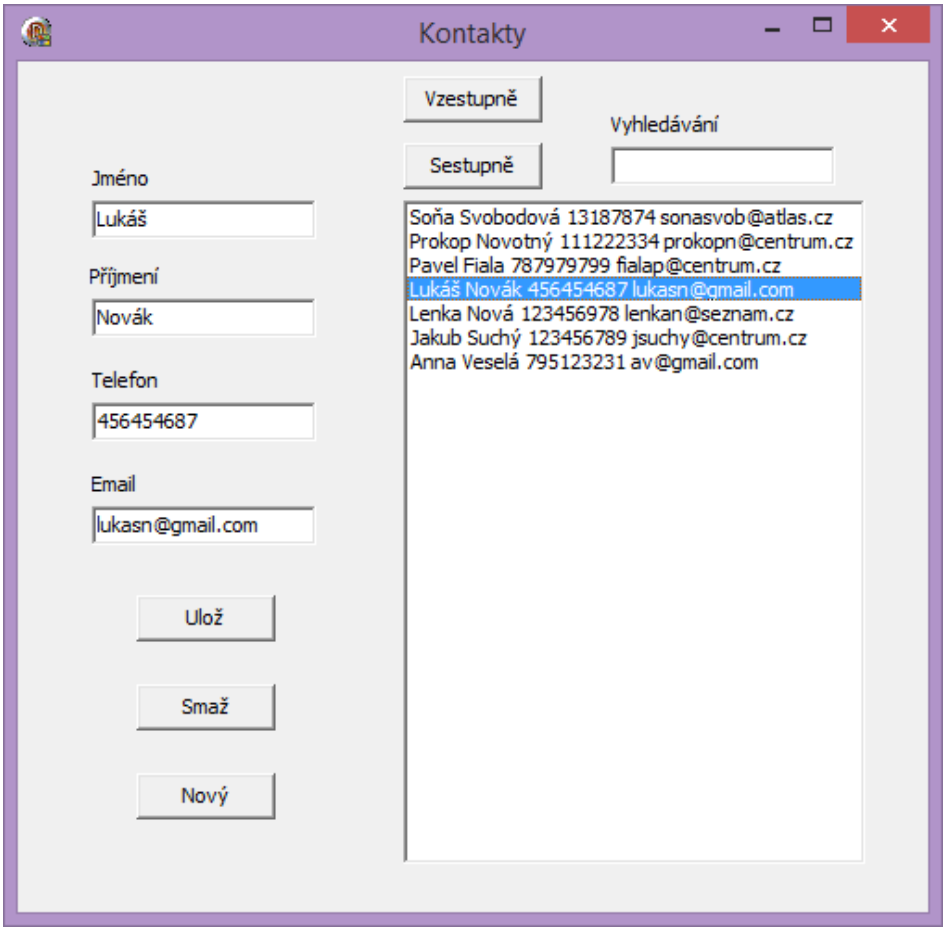
Nejprve si na formulář vložíme komponenty *Edit*, pro zadávání kontaktů a komponenty *Label* pro jejich popsání. Dále zde budeme mít tlačítka „Ulož“, „Smaž“ a „Nový“. Všechny kontakty budeme zobrazovat v komponentě *ListBox*. Pro řazení v *ListBoxu* budeme potřebovat dvě tlačítka „Vzestupně“ a „Sestupně“ a pro vyhledávání použijeme zbývající *Edit* a *Label*.

V tomto programu budeme pracovat s datovým typem *Record (Záznam)*. Do záznamu budeme ukládat každou jednotlivou osobu s uvedenými údaji. Jelikož chceme mít kontakty uložené v externím souboru, budeme muset nadefinovat s jakým souborem má program pracovat a co se do něj bude ukládat. Dále budeme potřebovat pole, abychom věděli kolik osob máme uložených v souboru.

Nejprve si ošetříme událost *FormCreate* při vytvoření formuláře. Do procedury této události nadefinujeme připojení textového souboru k naší proměnné „soubor“. To se provádí příkazem *AssignFile (soubor, 'kontakty.txt');.* Pokud by tento soubor neexistoval, tak ho vytvoříme pomocí příkazu *Rewrite.* Nyní můžeme do souboru ukládat po stisku tlačítka „Ulož“. Otevřeme si tedy soubor příkazem *Reset* a do datového typu Z*áznam* si přiřadíme komponenty *Edit* s vyplněnými údaji. Pozici, na kterou chceme záznam zapsat, si nastavíme příkazem *Seek* a zápis provedeme pomocí příkazu *Write.* Pokud ukládáme nový záznam, budeme ho ukládat vždy na konec souboru. Pokaždé, když přestaneme se souborem pracovat je potřeba tento soubor zavřít příkazem *CloseFile*. Pro zobrazení dat ze souboru si vytvoříme novou metodu s názvem „vypis“. V této metodě si načteme soubor a jednotlivé záznamy uložíme do řádek *ListBoxu*. Pokud bychom chtěli vybraný kontakt z *ListBoxu* zobrazit v jednotlivých políčkách pro úpravu dat, ošetříme si událost *ListBoxu OnClick*. Otevřeme si soubor, nastavíme si pozici v souboru podle pozice položky v *ListBoxu* pomocí vlastnosti *ItemIndex* a přečteme daný záznam pomocí příkazu *Read*. Do jednotlivých komponent *Edit* zapíšeme záznam a nezapomeneme zavřít soubor. Jelikož bychom chtěli uložit modifikovaný kontakt, musíme si v proceduře, která záznam ukládá ošetřit, zda se jedná o nový nebo stávající kontakt a podle toho ho zapsat na danou pozici. Smazání kontaktu budeme provádět tak, že si nastavíme pozici před tímto kontaktem a každý další záznam přepíšeme následujícím záznamem. V případě uložení nového záznamu bychom nemuseli definovat žádné tlačítko, ale může se stát, že si uživatel bude prohlížet uložené záznamy, některý si zobrazí a pak bude chtít vytvořit nový záznam. V tuto chvíli by měl v komponentech *Edit* zobrazené údaje prohlížené osoby. Proto mu nastavíme tlačítko „Nový“ na vymazání textu z těchto komponent.

Nyní máme ošetřené základní vlastnosti programu pro ukládání kontaktů. Tento program si vylepšíme o řazení kontaktů a jejich prohledávání. Vytvoříme si proceduru pro vzestupné řazení kontaktů, tedy od A do Z podle jména. Otevřeme si soubor a zjistíme si jeho délku. Poté si nadefinujeme velikost pole „kontakty“ a uložíme kontakty ze souboru do tohoto pole. Řazení budeme provádět pomocí zaměňování (tzv. *BubbleSort*). Budeme porovnávat dva sousední prvky, a pokud je druhý prvek větší než první, provede se jejich záměna. Porovnání textů provedeme pomocí funkce *CompareText*. Na toto procházení pole musíme použít dva do sebe vnořené cykly *for*, abychom zajistili porovnání i krajních

prvků pole. Následně seřazené pole zapíšeme do souboru. Pro řazení sestupně použijeme stejný kód jen u funkce *CompareText* obrátíme znaménko. Na prohledávání kontaktů podle příjmení si vytvoříme novou proceduru. V této proceduře budeme ukládat hledané slovo z komponenty *Edit* do proměnné a v souboru budeme hledat pomocí funkce *Pos*, která hledá daný řetězec v jiném řetězci. Tato funkce vrací hodnotu „0“, pokud slovo nenajde. Pokud funkce nevrátí tuto hodnotu, označíme v *ListBoxu* danou položku podle pozice.



Obrázek 16: Kontakty

4.6.4 MODIFIKACE

Tento příklad je možné rozšířit o řazení podle příjmení, přidání dalších položek (datum narození, druhé telefonní číslo nebo email). Dále je možné nastudovat další typy řazení a použít je.

4.7 SIMULÁTOR

4.7.1 ZADÁNÍ

Vytvořte komplexní příklad, ve kterém spojíte jednotlivé programy, které jsme si již naprogramovali.

4.7.2 CÍLE

Studenti si vyzkouší použití více jednotek a formulářů v jednom komplexním příkladu.

4.7.3 POPIS ŘEŠENÍ

Použité komponenty: *6x TBitBtn.*

Připravíme si obrázky, které nám budou definovat jednotlivé programy. Pro obsluhu na tomto formuláři budeme používat tlačítka *BitBtn*, na kterých si můžeme zobrazit námi vybraný obrázek (bitmapu). Do složky tohoto programu si zkopírujeme soubory jednotlivých příkladů s příponami .pas a .dfm. Pokud jsme při programování nechávali jednotlivým programům defaultní název Unit je potřeba tento název teď upravit, abychom neměli v jednom programu stejný název pro více unit. To platí i pro názvy formulářů. V okně *Project Manager* po kliknutí pravým tlačítkem na název projektu zvolíme *Add* a přidáme všechny soubory se zdrojovým kódem (*.pas). Všechny unity propojíme pomocí klíčového slova *uses*. Nyní vytvoříme události po kliknutí na jednotlivá tlačítka. V těchto procedurách nastavíme skrytí prvního formuláře a zobrazení formuláře podle daného tlačítka. V jednotlivých podprogramech si musíme ošetřit událost po zavření formuláře, tedy aby se nám zpět zobrazil hlavní formulář simulátoru.

Obrázek 17: Simulátor mobilního telefonu

4.7.4 MODIFIKACE

Tento program je možné modifikovat přidáním vlastní miniaplikace.

4.8 ANIMACE 1 - POPIS PROSTŘEDÍ DELPHI XE7

V této animaci jsem popsala vývojové prostředí Delphi XE7 pomocí učebnicového prvního programu „Ahoj světe“. Jsou v ní popsány jednotlivé panely a ukázán způsob použití jednotlivých komponent. Tato animace je přiložena v příloze A.

4.9 ANIMACE 2 - FUNKCE DEBUGGERU

Druhá animace ukazuje funkčnost *debuggeru* na příkladu řazení pole pomocí zaměňování. Vytvořila jsem pole o třech prvcích a přiřadila do něj hodnoty. Poté jsem naprogramovala řazení, kde jsem použila dva cykly *for* vnořené do sebe. Na tento příklad jsem nahrála animaci, ve které je ukázáno přidání proměnných do *Watch Listu* a následné krokování pro zaměnění prvků pole. V druhé části animace jsem použila stejný příklad, ale u *Breakpointu* jsem nastavila podmínku pro zastavení. Program tedy běží až do chvíle, kdy je splněna nastavená podmínka. Animace i s programem je přiložena v příloze A.

ZÁVĚR

Cílem této bakalářské práce bylo představit pokročilejší techniky v programování a vytvořit vhodnou sadu příkladů sloužící k procvičování programování studentům, kteří studují předmět Programování 2.

V bakalářské práci jsem popsala historii vývojového prostředí Delphi od verze Delphi 1 až do aktuální verze Delphi XE7. V druhé kapitole jsem toto prostředí podrobně popsala a uvedla základní informace pro psaní a ukládání jednotlivých projektů. Také jsem popsala debugger, se kterým by se měli studenti seznámit.

Jedním z cílů bylo představit pokročilé techniky programování a dynamické datové typy. Ty jsem uvedla ve třetí kapitole, kde jsme popsala jednotlivé principy OOP a dynamické datové typy.

Posledním úkolem bylo vytvoření sady příkladů pro samostatnou přípravu studentů na semináře předmětu Programování 2. Pro tuto sadu jsem si vybrala téma simulátoru mobilního telefonu, jelikož je to téma, které je studentům blízké a dokážou si představit funkčnost jednotlivých programů. Celkem jsem vytvořila šest typových příkladů - Hodiny (budík, stopky, časovač), Kalkulačka, Kreslení myší, Posun obrazců, Průzkumník souborů a Kontakty. Všechny tyto příklady jsem spojila do závěrečného komplexního příkladu. Praktickou část bakalářské práce jsem doplnila o dvě animace ukazující popis programovacího prostředí Delphi XE7 a funkčnost debuggeru.

RESUMÉ

The aim of this thesis was to introduce advanced techniques in programming and create an appropriate set of examples, which can be used to practice programming for students who are studying the subject Programming 2 at University of West Bohemia.

In the bachelor thesis is described the history of the integrated development environment Delphi. Described history starts with Delphi version 1 and ends with the current version of Delphi XE7. In the second chapter I described the environment and introduced the basic information for writing and saving individual projects. I also described the debugger, that should be used by students for debugging their applications.

One of the objectives was to introduce advanced techniques in programming and dynamic data types. In the third chapter are detailed described the OOP principles like polymorphism, inheritance and encapsulation. Also there are described dynamic data types like array, record, pointer and list.

The last task was creation a set of examples, which will be used for individual students preparation for a practise. For these exercises I chose topic mobile phone simulator. Because this topic is very close to students. Students can easily imagine the functionality of individual programs.

I created six typical examples - Clock (alarm clock, stopwatch, timer), Calculator, Mouse drawing, Shift patterns, File Explorer and Contacts. Finally I merged all these small examples into the final complex example. The practical part of this bachelor thesis was completed by two animations. First animation shows the integrated development environment Delphi XE7. Second animation shows debugger tool functionality.

SEZNAM LITERATURY

[1] SKŘIVÁNEK, Martin. Delphi - historie a vývoj: historie. *Fakulta informatiky Masarykovy univerzity.* [Online] 2003. [Citace: 5. 12 2014.] http://www.fi.muni.cz/usr/jkucera/pv109/2003p/xskriva1.historie.htm.

[2] Verze Delphi (TP1 → XE7). *Delphi.cz.* [Online] 2014. [Citace: 5. 12. 2014.] http://delphi.cz/page/Verze-Delphi.aspx.

[3] TEIXEIRA, Steve. *Mistrovství v Delphi 6: Průvodce programováním databázových, webových i kompletních podnikových aplikací. Základní i pokročilé programovací techniky. Vývoj aplikací pro Windows i L.* 1. vyd. Praha : Computer Press, 2002. str. 820. ISBN 80-722-6627-6.

[4] Wikipedie: Otevřená encyklopedie: Pascal (programovací jazyk). [Online] c2014. [Citace: 5. 12. 2014.] http://cs.wikipedia.org/w/index.php?title=Pascal_(programovac%C3%AD_jazyk)&oldid=12093151.

[5] Wikipedie: Otevřená encyklopedie: Delphi. [Online] c2014. [Citace: 5. 12. 2014.] http://cs.wikipedia.org/w/index.php?title=Delphi&oldid=11702731.

[6] PÍSEK, Slavoj. *Delphi - začínáme programovat: podrobný průvodce začínajícího uživatele.* 2., upr. a aktualiz. vyd. Praha : Grada, 2002. str. 325. ISBN 80-247-0547-8.

[7] SKŘIVÁNEK, Martin. Delphi - historie a vývoj: vývoj. *Fakulta informatiky Masarykovy univerzity.* [Online] 2003. [Citace: 5. 12 2014.] http://www.fi.muni.cz/usr/jkucera/pv109/2003p/xskriva1.vyvoj.htm.

[8] ČERVINKA, Radek. Historie změn v Delphi od 1 do XE5. *Delphi.cz.* [Online] 24. 10. 2013. [Citace: 5. 12. 2014.] http://delphi.cz/post/Historie-zmen-v-Delphi-od-1-do-XE5.aspx.

[9] JURÁSEK, Vít. Nové Borland Delphi 2005 je na světě. *zive.cz.* [Online] 16. 10. 2004. [Citace: 5. 12. 2014.] http://www.zive.cz/bleskovky/nove-borland-delphi-2005-je-na-svete/sc-4-a-120294/default.aspx.

[10] KADLEC, Václav. *Delphi: hotová řešení : řešené problémy, tipy a zajímavé programy v Delphi, více než 170 stavebních prvků pro vaše aplikace, stručná teorie, praktická vysvětlení.* 1. vyd. Praha : Computer Press, 2003. str. 312. K okamžitému použití. ISBN 80-251-0017-0.

[11] Wikipedie: Otevřená encyklopedie: Objektově orientované programování. [Online] c2015. [Citace: 14. 03. 2015.] http://cs.wikipedia.org/wiki/Objektov%C4%9B_orientovan%C3%A9_programov%C3%A1n%C3%AD.

[12] Objektově orientované programování. *soom.cz.* [Online] 31. 3. 2013. [Citace: 14. 3. 2015.] http://www.soom.cz/clanky/639--Objektove-orientovane-programovani.

[13] BINZINGER, Thomas. *Naučte se programovat v Delphi: podrobný průvodce začínajícího uživatele.* Praha : Grada Publishing, 1998. str. 344. ISBN 80-7169-685-4.

SEZNAM OBRÁZKŮ, TABULEK, GRAFŮ A DIAGRAMŮ

PŘÍLOHY

A) Přílohy na přiloženém CD

- Příklady - zdrojové kódy i zkompilované programy
- Animace
- Vlastní text bakalářské práce