Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Katedra informačních technologií
Akademický rok: **2007/2008**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Tomáš ZATÍRANDA**

Studijní program: **B2646 Informační technologie**

Studijní obor: **Informační technologie**

Název tématu: **Výukové pásmo vývojových prostředků AutoCADu v jazyce C**

Zásady pro vypracování:

Teoretická část:
Technologie .NET pro knihovní prvky ObjectARX
Úvod do technologie
Formáty ukládání souboru
Závislosti a omezení
Vývojové prostředí
Základy programovacího jazyka
Základy práce s technologií .NET
Příklady - vypsání textu do příkazové řádky, vykreslení kružnice, jednoduché dialogové okno, reagující dialog
Testovací příklady - použití příkazů AutoCADu v aplikaci .NET, aplikace s uživatelským rozhraním
Testovací otázky
Implementační část:
Rozšíření webové prezentace http://www.cadforum.cz/cadforum/Vyvojove-prostredky-AutoCADu/System/Hlavni/frmHlavniSet.htm o kapitolu Vývojové prostředky pro AutoCAD technologií .NET
Vytvoření výukového pásma k prezentaci

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

Ellen Finkelstein: Mistrovství v AutoCADu pro verze 2004 až 2006, CP Books, 2005
Nagel Ch., Evjen B., Glynn J., Skinner M.,W.: C 2005 – Programujeme profesionálně, Computer Press, 2005
http://webak.upce.cz/áhajek/iajce/ - Algoritmizace a jazyk C
http://webak.upce.cz/áhajek/irae/ - Programování řídících aplikací
http://www.autodesk.com/objectarx - Autodesk - Developer Center – ObjectARX
http://www.cadforum.cz/cadforum/Vyvojove-prostredky-AutoCADu/System/Hlavni/frmHlavniSet.htm - Vývojová prostředí AutoCADu

Vedoucí bakalářské práce: **Ing. Zbyněk Kopecký**
Ústav elektrotechniky a informatiky

Datum zadání bakalářské práce: **30. listopadu 2007**
Termín odevzdání bakalářské práce: **16. května 2008**

doc. Ing. Simeon Karamazov, Dr.
děkan

V Pardubicích dne 29. dubna 2008

Abstrakt

Tato práce se zabývá problematikou tvorby aplikací pro AutoCAD pomocí technologie .NET pro knihovní prvky ObjectARX. Nastiňuje základy práce ve vývojovém prostředí Visual Studio 2005, programování v jazyku C# a práci s .NET, včetně několika ukázkových příkladů.

Klíčová slova

c#, .NET, AutoCAD, Visual Studio

Title

AutoCAD development tools tutoriál in C# language

Abstrakt

This work conserned with developing applications for AutoCAD using .NET technology for ObjectARX library items. Show basics of work in Visual Studio 2005, C# programming and work with s .NET, including four examples

Klíčová slova

c#, .NET, AutoCAD, Visual Studio

Obsah

1. Úvod 8

2. Úvod do technologie 8

2.1 Co je .NET? 8

2.2 Součásti .NET 9

2.3 .NET Framework 9

2.4 CLR – Common Language Runtime 9

2.5 CTS 10

2.6 Typová bezpečnost 11

2.7 Management paměti 12

2.8 MSIL 12

2.9 Bezpečnost .NET Framework 12

2.10 Shrnutí 13

2.10.1 Inteligentní symbolická interpretace 13

2.10.2 Programovací styl 13

2.10.3 Výhody programování v .NET 14

2.10.4 Automatický management paměti 14

3. Závislosti a omezení 14

3.1 Požadavky na systém 14

3.1.1 Doporučené požadavky na HW 14

3.1.2 Požadavky na SW 15

3.2 Požadavky na uživatele 15

4. Formáty ukládání souborů 15

5. Vývojové prostředí Microsoft Visual Studio 2005 16

5.1 Vytvoření projektu 17

5.2 Solution Explorer a Vlase View 18

5.3 Properties 18

5.4 Toolbox 19

5.5 Error list 20

5.6 Ladění 20

5.7 Zarážky 21

5.8 Krokování programu 21
5.9 Rychlé zobrazení hodnoty proměnné 23
5.10 Nástroj Quick Watch 23
6 Základy jazyka C# 23
6.1 Základy objektově orientovaného programování 24
6.1.1 Charakteristické rysy OOP 24
6.1.2 Abstrakce 24
6.1.3 Členské funkce 25
6.1.4 Dědičnost 25
6.1.6 Objekt 25
6.1.7 Polymorfismus 26
6.1.8 Předefinování 26
6.1.9 Přetěžování funkcí 26
6.1.10 Přetěžování operátorů 26
6.1.11 Rozhraní (interface) 27
6.1.12 Zapouzdření 27
6.2 Direktivy preprocesoru 27
6.3 Datové typy 28
6.3.1. Hodnotové typy 28
6.4 Referenční typy 31
6.5 Proměnné 31
6.5.1 Životnost 31
6.5.2 Konstanty 32
6.6 Operátory 32
6.7 Třídy 34
6.7.1 Metody a parametry 35
6.7.2 Konstruktory a destruktory 36
6.7.3 Dědičnost a polymorfismus 37
6.7.4 Konstruktory v odvozených třídách 39
6.8 Pole 39
6.8.1 Deklarace pole 39
6.8.2 Vícerozměrná pole 40
6.8.3 Vícerozměrná nepravidelná pole 40
6.8.4 Kopírování polí 41

6.9 Podmíněné příkazy 41
6.9.1 Příkaz if 41
6.9.2 Příkaz switch...case 41
6.10 Cykly 42
6.10.1 Cyklus while 42
6.10.2 Cyklus do-while 42
6.10.3 Cyklus for 42
6.10.4 Cyklus foreach 42
6.11 Skokové příkazy 43
6.11.1 Příkaz break 43
6.11.2 Příkaz continue 43
6.11.3 Příkaz goto 43
6.11.4 Příkaz return 43
6.11.5 Příkazy checked a unchecked 44
6.11.6 Příkaz throw 44
7. Základy práce s technologií .NET 44
7.1 Knihovny tříd 44
7.2 Vytvoření nového projektu .NET 45
7.2.1 Ručně vytvořený 45
7.2.3 Pomocí nástroje AutoCAD C# Application Wizard 46
7.3 Tvorba aplikace .NET 46
7.3.1 Definování Příkazů AutoCADu 46
7.3.2 Používání transakcí 47
7.3.3 Přístup k uživatelskému rozhraní 47
7.3.4 ExtensionApplication a CommandClass Atributy 48
8. Příklady 48
8.1 Vypsání textu do příkazové řádky 48
8.2 Výzva k uživatelskému vstupu 51
8.3 Získání geometrické vzdálenosti 52
8.4 Vykreslení kružnice 52
8.5 Přidávání uživatelských dat 54
8.5.1 Uživatelské kontextové menu 54
8.5.2 Ukotvitelné paletové okno a Drag and Drop 56
8.5.3 Entita vybraná z modálního formuláře 61

9. Závěrečné zhodnocení 64
10. Použitá literatura 65

Seznam obrázků a tabulek

Obrázek 1. Common Language Runtime - kompilace a spuštění 10
Obrázek 2. Založení nového projektu 16
Obrázek 3. Nabídka build 17
Obrázek 4. okno Class View 18
Obrázek 5. Okno Properties 19
Obrázek 6: Toolbox 19
Obrázek 7. Error list 20
Obrázek 8: Základní typy 28
Tabulka 1: Primitivní datové typy celočíselné 29
Tabulka 2. Reálné datové typy a typ decima 29
Tabulka 3. tabulka priority operátorů 32
Tabulka 4. mapování prefixů na jmenné prostory 45

1. Úvod

AutoCAD je populární software pro 2D a 3D projektování a konstruování (CAD), vyvinutý firmou Autodesk. Na jádru Autodesk byla Autodeskem vyvinuta sada profesních aplikací určených pro CAD v oblasti strojírenské konstrukce, stavební projekce a architektury, mapování a terénních úprav. AutoCAD poskytuje řadu API rozhraní (AutoLISP/VisualLISP, VBA, ObjectARX, .NET) a je tak i otevřenou platformou pro nadstavbové aplikace třetích firem. Řada firem toho využívá a s výhodami z toho plynoucími vytváří vlastní aplikace, či upravuje stávající tak, aby co nejlépe vyhovovaly jejich specifickým potřebám.

Cílem této práce je přiblížit techniku tvorby těchto aplikací, včetně základů práce ve vývojovém prostředí Visual Studio 2005 a popisu .NET a jazyka C#, které mají v současnosti stále větší zastoupení a usnadnit tak počáteční práci lidem, kteří se chtějí této problematice věnovat.

2. Úvod do technologie .NET

.NET modifikuje a rozšiřuje AutoCAD a na AutoCADu založené produkty přímým přístupem do databázových struktur AutoCADu, vlastní definicí příkazů a dalším, využitím kteréhokoliv jazyku podporovaného .NET.

2.1 Co je .NET?

- Microsoft technologie web-based infrastruktury
- Hladká interakce mezi aplikacemi a Internetem
- Přístupové informace kdykoliv, kdekoliv, z jakéhokoliv zařízení

2.2 Součásti .NET

- .NET Framework využívaný pro tvorbu a běh všech druhů softwaru, včetně Web-based aplikací, smart client aplikací a XML webových služeb
- Vývojové nástroje jako Microsoft Visual Studio .NET
- Sada serverů, které integrují, spouští, obsluhují, a řídí Webové služby a Web-based aplikace
- Klientský software pomáhající vývojářům předat obsáhlé a účinné uživatelské zkušenosti díky souboru prostředků a existujících produktů.

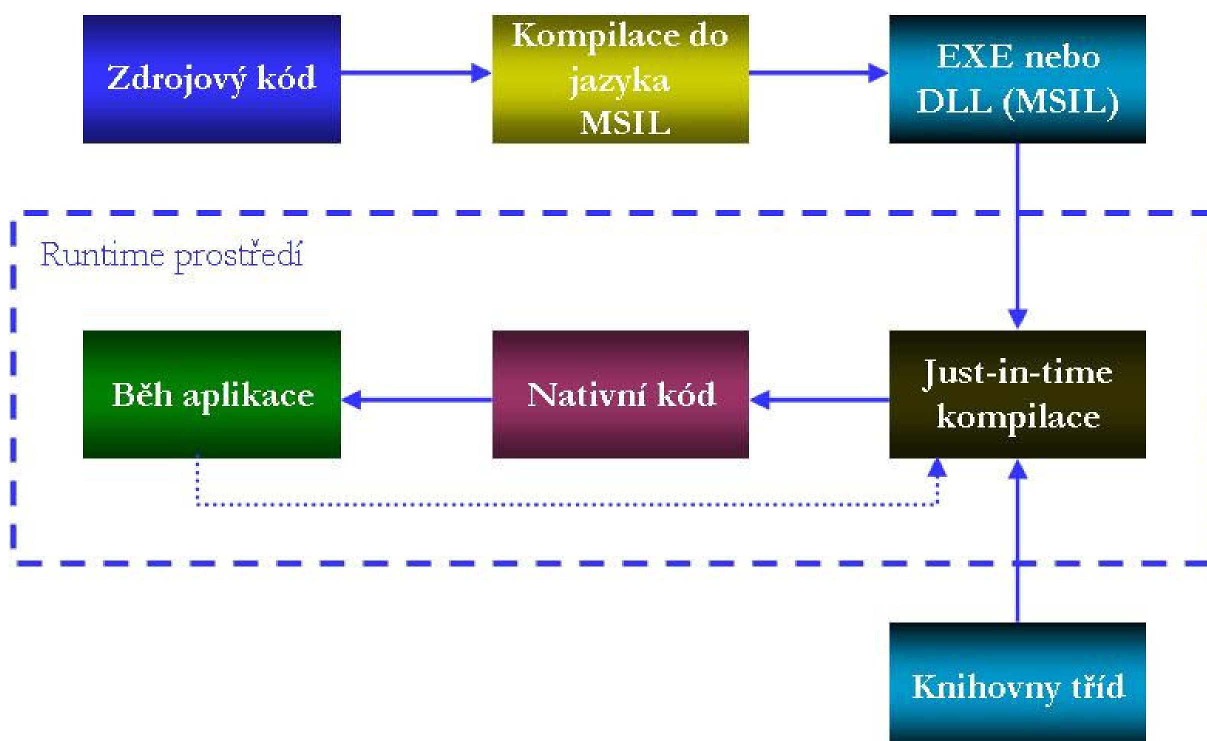
2.3 .NET Framework

Jeho jádro je založené na principech objektově orientovaného programování a všechny základní služby zpřístupňuje široké škále programovacím jazykům. .NET Framework automaticky podporuje třídy, metody, vlastnosti, konstruktory, události, polymorfismus atd. Ve výsledném efektu to znamená, že není podstatné, ve kterém programovacím jazyce komponenty vytváříme případně, jaké komponenty používáme. .NET Framework také řeší některé problémy související s bezpečností. Dalším problémem, který .NET Framework řeší, je nasazování a instalace aplikací (označovaný jako DLL Hell).

2.4 CLR – Common Language Runtime

CLR si lze ztotožnit s pojmem virtuálního stroje při použití programovacího jazyka Java. Podobně jako u programovacího jazyka Java, nejsou zdrojové kódy kompilovány přímo do nativního kódu, který lze provádět, ale do intertmediárního jazyka. U programovacího jazyka Java je výsledkem soubor s příponou CLASS a tento je pak prováděn virtuálním strojem Javy. Podobně v prostředí .NET jsou

zdrojové soubory libovolného programovacího jazyka zkompilovány do intermediárního jazyka (nazvaného MSIL – Microsoft Intermediate Language). V případě, že má být taková aplikace spuštěna, systém detekuje, že jde o aplikaci v MSIL a spustí Just-In-Time kompilátor. Ten vygeneruje skutečné instrukce cílové platformy



Obrázek 1. Common Language Runtime - kompilace a spuštění

Jedním z hlavních cílů při vývoji .NETu je podpora různých programovacích jazyků. Důležitým prvkem CLR je podpora společného typového systému (Common Type Systém – CTS). Vedle CTS definovaném na systémové úrovni, CLR realizuje typovou bezpečnost a obecný objektově orientovaný model.

2.5 CTS

CTS je nezávislý na jazykové implementaci. Objektově orientované jazyky a přístup k tvorbě softwaru jsou již léta součástí vývojových nástrojů. Implementace se

v různých prostředích ale výrazně liší. Některé programovací jazyky jsou čistě objektově orientované (například Smalltalk), jiné implementují vedle objektově orientovaných principů také některé neobjektové vlastnosti. V čistě objektově orientovaných programovacích jazycích jsou všechny datové typy představovány třídami a všechny proměnné jsou ve skutečnosti objekty.

Objektovost dotažena do úplného konce je značně výhodná, ale přináší nemalou daň v podobě poklesu výkonu. Proto přidáváme čistě objektovým jazykům tzv. základní typy a množinu operací, které zlepšují výkonnost daného programovacího jazyka. Typový systém je rozdělen do dvou hlavních kategorií: hodnotové typy a referenční typy. V .NETu je vše (podobně jako v Javě) objektem. Základem každého typu je třída nazvanáSystem.Object. Výjimku tvoří hodnotové typy. Jak již bylo uvedeno, pro zvýšení výkonnosti jsou i do .NETu přidány některé základní typy jako různé druhy celých nebo reálných čísel. Někdy je ovšem nutné, pracovat i s těmito typy jako s objekty. .NET nám umožňuje provádět automatickou konverzi hodnotového typu na referenční.

Operaci, která převod zajistí se říká Boxing. Opačnému procesu - převedení hodnotového typu na referenční - se říká Unboxing. Všechny hodnotové typy mají definovány odpovídající třídu, na kterou jsou konvertovány. Každý základní typ je reprezentován implicitně svou hodnotou, ale v případě potřeby je možné jej převést na odpovídající objekt a pracovat s ním jako s každým jiným objektově orientovaným typem. Díky tomu se daří dosáhnout požadované plné objektovosti typového systému bez zaplacení výkonnostního penále.

2.6 Typová bezpečnost

Jedním z nejdůležitějších požadavků kladený na společný typový systém je typová bezpečnost. Můžeme jí také chápat jako garanci, že nad definovanými typy nemohou být provedeny nepovolené operace. Typová bezpečnost odstraňuje chyby pramenící z nekontrolované manipulace s poměnými nebo pamětí. Každý objekt vytvořený v programu je striktně typový a typová je i reference, která se na něj odkazuje. Každý typ je navíc sám zodpovědný za přístupová práva ke svým členům.

2.7 Management paměti

Obecným cílem managementu paměti je uvolnit systémové zdroje držené objektem a následně i paměti, kterou zabírá ve chvíli, kdy objekt již nikdo nevyužívá. .NET Framework má technologii Garbage Collection. Ta přináší automatickou správu paměti.

2.8 MSIL

Spustíme-li aplikaci o jejíž provádění se stará CLR, hovoříme o řízeném kódu. Pokud je spuštěna aplikace která není napsaná pro prostředí .NET, nebo se výhod řízeného kódu explicitně zřekneme, je kód neřízený. Ne všechny jazyky umí generovat řízený kód. Příkladem jazyka s absolutní volností je jazyk C++.

Výsledkem kompilátoru jazyka schopného generovat řízený kód je MSIL - MicroSoft Intermediate Language. MSIL je procesorově nezávislý jazyk podobný assembleru. Oproti asembleru je však mnohem vyspělejší. Umí pracovat s objekty, volat virtuální metody, pracovat s prvky pole nebo zpracovávat výjimky. Důvodem pro zavedení tohoto jazyka je snaha o jednoduché přenášení existujícího kódu mezi různými platformami. V současné době neexistuje procesor, na kterém by šlo provádět instrukce MSIL, proto CLR před vlastním spuštěním kompiluje (pomocí JIT kompilátoru) MSIL instrukce do nativního kódu na dané, konkrétní platformě. Hlavní výhodou použití intermediárního jazyka je platformní nezávislost.

2.9 Bezpečnost .NET Framework

Pojem bezpečnosti se dotýká několika oblastí:

- typová bezpečnost - typová bezpečnost v tomto kontextu se týká pouze bezpečného přístupu k paměti objektů. Během JIT kompilace probíhá verifikační proces zkoumající obsah metadat v manifestu a obsah MSIL kódu a ověřující zda-li je kód typově bezpečný;
- identita kódu - na základě informací o kódu definuje práva přidělená aplikacím;

- code access security (přístupová bezpečnost kódu) - tento typ bezpečnosti umožňuje nastavit důvěryhodnost kódu na požadovanou úroveň v závislosti na tom, odkud kód pochází a dalších aspektech daných identitou kódu;
- povolení (permissions) - CLR umožňuje aplikacím vykonávat jen ty operace, pro která má jejich kód povolení. Běhové prostředí používá speciální objekty zvané permissons pro implementaci omezení, která jsou kladena na řízený kód;
- bezpečnost založená na rolích - využívá identity asociované s daným exekučním kontextem;

- kryptografické služby - .NET Framework obsahuje celou sadu kryptografických objektů, které implementují známé algoritmy pro hashování, kryptování a digitální podpisy.

2.10 Shrnutí

- Common Language Runtime (CLR)
- Objektově orientované programovací prostředí
- Společné vykonávací prostředí pro .NET aplikace
  - podobné Java VM – ale s mnohem silnější součinností
- Framework Class Library (FCL)
- Objektově orientovaná sbírka znovu použitelných typů

2.10.1 Inteligentní symbolická interpretace

- Vyzrálá stavba jazyků
- Společné programming pitfalls addresování
- Management paměti, důsledná obsluha vyjímek, jednotné řetězce

2.10.2 Programovací styl

- Několik podporovacích jazyků

2.10.3 Výhody programování v .NET

- Pevná objektově orientovaná vývojová platforma
- Automatický management paměti – Garbage collection
- Podpora různých jazyků

2.10.4 Automatický management paměti:

- Starý způsob (C++)

```
char *pName=(char*)malloc(128);
strcpy(pName,"Hello");
//...
free(pName);
```

- Nový způsob - .NET
  - C++ - String *pName=new String("Hello")
  - VB - Dim Name As String = "Hello"
  - C# - String Name="Hello";
  - Garbage collection ovládá dealokaci; žádné 'delete'!

3. Závislosti a omezení

3.1 Požadavky na systém

3.1.1 Doporučené požadavky na HW:

- Pentium IV nebo lepší (většinou se uvádí 2 GHz)
- 512 MB RAM
- 750 MB volného místa na disku pro instalaci AutoCADu + 2 GB pro instalaci Visual Studia
- VGA monitor (doporučeno rozlišení 1024 × 768 true color nebo vyšší).

Dodatečné požadavky pro 3D:

- Procesor 3.0GHz nebo více

- Operační paměť 2GB
- Grafická karta 128MB

3.1.2 Požadavky na SW:

- Microsoft® Windows® XP Professional nebo Home Edition (SP1 or SP2), Windows XP Tablet PC Edition (SP2), or Windows 2000 (SP3 or SP4)
- Windows NT 4.0, Windows 98 SE, Windows 98, Windows ME, a Windows 95 nejsou podporovány

Další software nutný pro tvorbu aplikací .NET

Abyste mohli vytvářet aplikace pomocí .NET je nutné mít nainstalované Microsoft Visual studio 2005 či novější verze (.NET frameworky a všechny potřebné service packy a jsou součástí instalace)

ObjectARX Software Developers Kit - jedná se o soubory knihoven, hlavičkové soubory, příklady a další podpůrný software. ObjectARX SDK je možné zdarma získat na internetových stránkách firmy Autodesk

3.2 Požadavky na uživatele:

- znalost prostředí AutoCADu a výkresové databáze
- znalost objektového programování a jazyka C#
- znalost vývojového prostředí Visual Studio výhodou

4. Formáty ukládání souborů

Při tvorbě aplikace v .NET se používá několik typů souborů (.cs, .resx, atd.). Jedná se o soubory používané vývojovým prostředím Microsoft Visual Studio. Projekt vytváříme v prostředí Visual Studio jako DLL - dynamicky slinkovanou knihovnu.

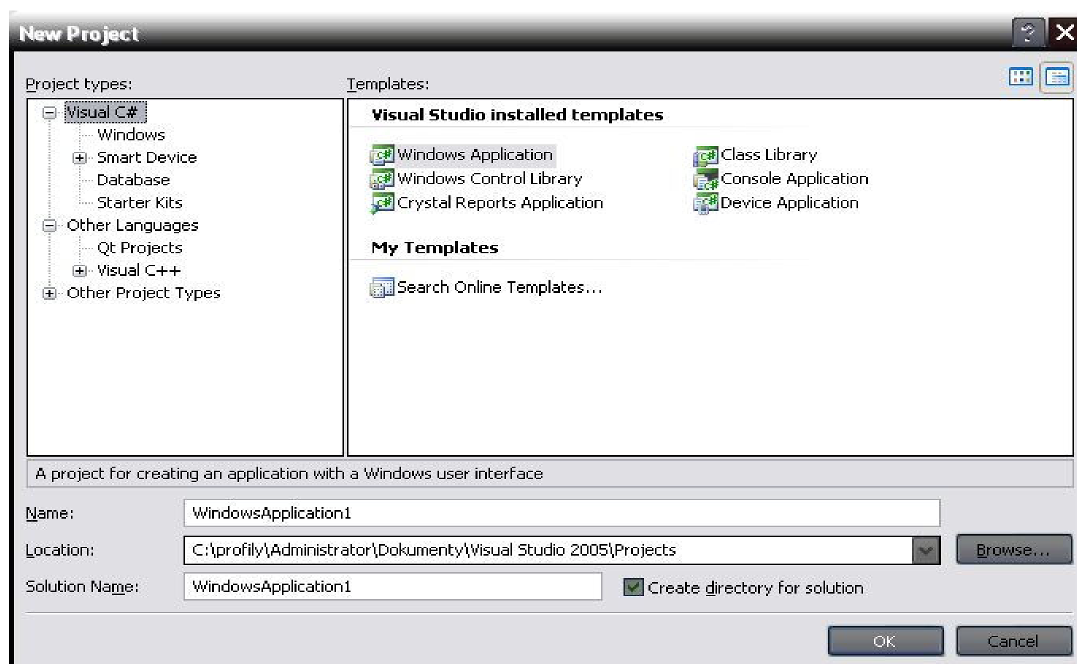
Soubor aplikace .NET je možné do AutoCADu načíst příkazem **NETLOAD** napsaným do příkazového řádku. Ten umožňuje načíst .DLL soubor dané aplikace (načte ji do AutoCADu).

Příkaz lze zahrnout do standardních "autoload" nástrojů AutoCADu nebo lze aplikaci načítat pomocí "demandload" mechanismů(specifikují, zda a kdy AutoCAD požaduje načtení aplikací třetí strany, když kresba obsahuje objekty vytvořené v té aplikaci).

5. Vývojové prostředí Microsoft Visual Studio 2005

Visual Studio 2005 je určeno pro programování klasických desktopových, serverových, webových (ASP.NET) i mobilních aplikací na platformách Windows a .NET 2.0. Z programovacích jazyků jsou k dispozici Visual C++ (nativní i řízené), Visual C#, Visual Basic a Visual J#.

Nový projekt vytvoříte příkazem Project podnabídky New v nabídce File. Po zadání tohoto příkazu se objeví dialog nového projektu New project uvedený na obrázku 2.



Obrázek 2. Založení nového projektu

V levém panelu dialogu New Project se nachází seznam typů projektů pro jednotlivé jazyky Visual Studia 2005. Vyberte C#. Pravý panel dialogu New Project nyní obsahuje seznam šablon pro různé typy aplikací, které můžete vytvářet. Tyto projektové šablony vám pomůžou s vytvořením počátečních souborů, kódů a dalších nastavení zvoleného projektu. Na výběr máte několik šablon. Zvolíte si tu, kterou potřebujete, zvolíte název a umístění projektu. Jakmile budete spokojeni, klepněte na tlačítko OK a Visual Studio 2005 poté vygeneruje soubory a složky projektu. Zároveň se objeví vývojové prostředí, ve kterém můžete začít s prací.

5.1 Spuštění projektu

Aby se dal projekt spustit jako aplikace, musí se sestavit další soubory, což můžete udělat z nabídky Build uvedené na obrázku 2.3 některým z následujících 4 příkazů:

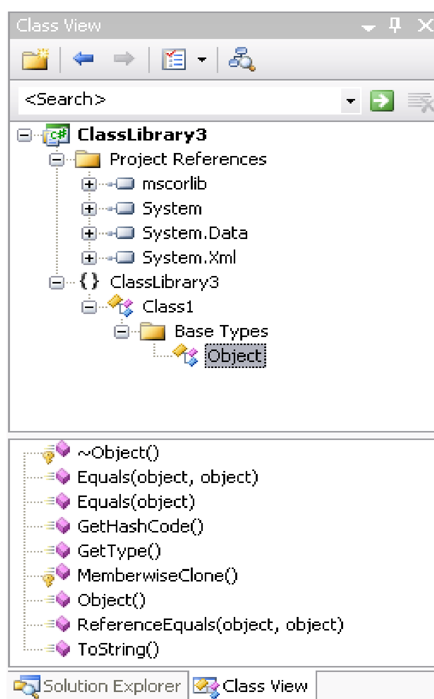
- Build Solution (sestavit)
- Rebuild Solution (znovu sestavit)
- Build NázevProjektu
- Rebuild NázevProjektu

Obrázek 3. Nabídka build

Rozdíl mezi Build Solution a Build NázevProjektu je ten, že první se týká celého řešení (solution) a druhý pouze určitého projektu.

5.2 Solution Explorer a Class View

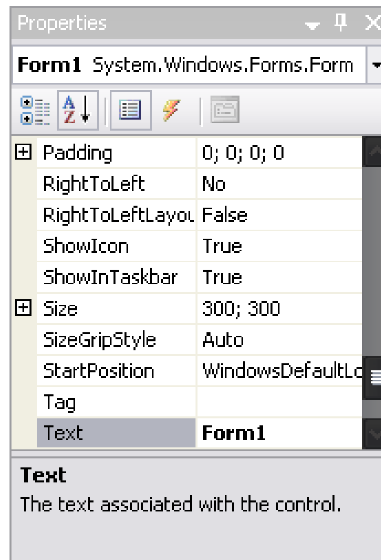
Pomocí Solution Exploreru a Class View je možné jednoduchým způsobem pracovat se jednotlivými projekty, soubory, třídami, metodami a proměnnými. Lze je do projektu přidávat, mazat, editovat a vyhledávat. Pokud se potřebujete rychle přesunout na místo deklarace třídy stačí dvakrát kliknout nad jménem třídy.



Obrázek 4. okno Class View

5.3 Properties

Zobrazí se výběrem položky Properties z nabídky View. V okně vlastností se nachází seznam různých atributů a charakteristik formuláře(v případě, že tvoříte např. windows aplication), jako jsou jeho výška, šířka, barva, pozadí, text záhlaví okna, atd. Vlastnosti objektů formuláře se mohou nejen prohlížet, ale během návrhu aplikace můžete jejich hodnoty také měnit.



Obrázek 5. Okno Properties

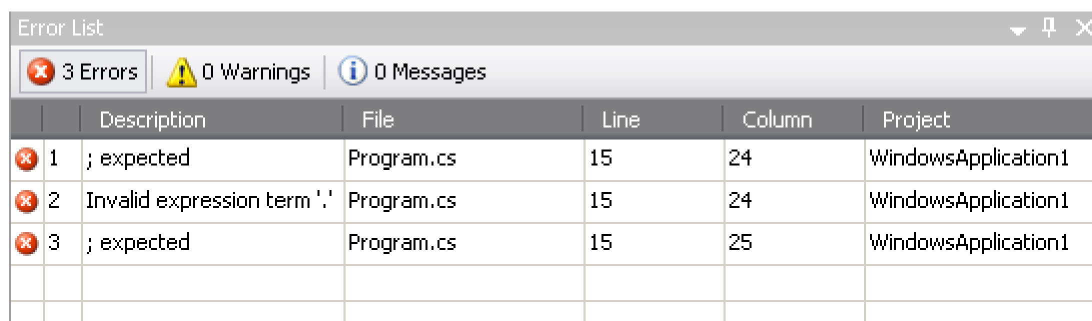
5.4 Toolbox

Visual Studio zobrazuje ovládací prvky, které lze vkládat do formulářů v Toolboxu, viz obrázek. Položky jsou rozděleny do několika kategorií. Ovládací prvek lze vložit několika způsoby. Jednou možností je poklepání v Toolboxu na zvolený ovládací prvek, který se poté objeví někde na formuláři. Dalším způsobem je přetažení zvoleného prvku na formulář.

Obrázek 6. Toolbox

5.5 Error list

Zobrazují se v něm chybová hlášení a varování například při špatně deklarované proměnné, či jakémkoliv jiném prohřešku proti pravidlům. Je zde zobrazen popis chyby, řádek na kterém je, soubor a projekt, ve kterém je chyba obsažena.



		Description	File	Line	Column	Project
	1	; expected	Program.cs	15	24	WindowsApplication1
	2	Invalid expression term '.'	Program.cs	15	24	WindowsApplication1
	3	; expected	Program.cs	15	25	WindowsApplication1

Obrázek 7. Error list

5.6 Ladění

Podporu ladění (debugging) poskytuje třída debut, která se nachází ve jmenném prostoru Systém.Diagnostic. Tento jmenný prostor je možné na začátku kódu importovat následovně:

```
Using System.Diagnostic;
```

Od tohoto okamžiku můžete používat pouze výraz Debut namísto delšího System.Diagnostic.Debug. Třída debut obsahuje metodu WriteLine.

```
Debut.WriteLine(parametr);
```

Metoda WriteLine vypisuje hodnotu předaného parametru do okna výstupu Output a vypisuje do okna výstupu pouze tehdy, pokud spustíte aplikaci nabídky Debug | Start Debugging. Do okna výstupu Output obvykle nezapisuje pouze metoda WriteLine. Většinou jsou zde také informace generované vývojovým prostředím.

5.7 Zarážky

Zarážky označují místa, před nimiž se běh programu při ladění zastaví. Po zastavení běhu programu je možné zkontrolovat hodnoty proměnných, popřípadě je změnit a pokračovat v krokování nebo nechat program běžet dále. Zarážky je možné vkládat během psaní kódu programu (režim editace) nebo v režimu ladění.

Umístění pomocí plovoucí nabídky:

- ukažte kurzorem myši na řádek do něhož má být vložena zarážka,
- stiskněte pravé tlačítko myši - otevře se plovoucí nabídka,
- z nabídky vyberte položku ***Insert / Remove Breakpoint***.

Umístění zarážky klávesovou zkratkou:

- umístěte kurzor na řádek, do kterého má být vložena zarážka a zmáčkněte klávesovou zkratku ***F9***(lze i z nabídky Debug příkazem Toggle Breakpoint)

Umístění zarážky pouhým kliknutím

- stačí pouze kliknout na začátek řádku, kam chcete vložit Breakpoint

Lze též umístit Breakpoint přímo na funkci výběrem z nabídky Debug | New Breakpoint | Break at Function

Umístění zarážky je signalizováno červeným kolečkem zobrazeným u řádku kódu.

Zarážky je možné při ladění programu i dočasně vyřadit. Z plovoucí nabídky (vyvolané pravým tlačítkem myši) vyberte položku ***Disable Breakpoint***. Zarážka změní barvu na bílou s červeným okrajem. Narazí-li ladící program na skrytou zarážku ignoruje ji. Později zarážku můžete opět aktivovat příkazem ***Enable Breakpoint*** (z plovoucí nabídky).

5.8 Krokování programu

Krokování programu se využívá jako základní prostředek nalezení chyb v programu. Během krokování můžeme sledovat hodnoty proměnných v programu,

správnost vyhodnocení proměnných atd. Do režimu krokování můžeme přejít několika způsoby. Nejjednodušší je zmáčknout klávesovou zkratku ***F11***. Jinou možností je postup:

- otevřete nabídku ***Debug***,
- vyberte položku ***Step Into***.

Visual Studio přejde do ladícího režimu. Objeví se nástrojový panel Debug. V panelu nabídek (menu) se přidá položka Debug, která nahradí položku Build. Ve spodní části se skryje okno Output a místo něho se zobrazí okna Watch (sledované výrazy) a Locals.

Visual Studio nabízí několik možností krokování programu:

- krokování typu ***Step Into*** vstupuje i do volaných funkcí. Krok *Step Into* je možné spustit:
  - pomocí nabídky ***Debug***, položka ***Step Into***,
  - klávesovou zkratkou ***F11***,
- krokování typu ***Step Over*** nevstupuje do volaných funkcí (funkci považuje za jedinný příkaz). Krok *Step Over* je možné spustit:
  - pomocí nabídky ***Debug***, položka ***Step Over***,
  - klávesovou zkratkou ***F10***,
- krokování typu ***Step Out*** umožňuje ukončit krokování funkce, v jejímž těle se právě nacházíme a pokračovat za jejím voláním. Krok *Step Out* je možné spustit:
  - pomocí nabídky ***Debug***, položka ***Step Out***,
  - klávesovou zkratkou ***Ctrl + F11***,

- Režim ladění je možné kdykoliv ukončit použitím příkazu *Stop Debugging*. Příkaz je možné vyvolat:

- pomocí nabídky ***Debug***, položka ***Stop Debugging***,
- klávesovou zkratkou ***Shift + F5***,

5.9 Rychlé zobrazení hodnoty proměnné

V režimu ladění ukažte kurzorem myši na proměnnou, její hodnota se okamžitě zobrazí pomocí bubliny:

Touto metodou je možné zjistit i hodnotu podmínky nebo části výrazu. Potřebujeme-li zobrazit hodnotu podmínky (části výrazu) vybereme požadovanou část a umístíme nad ni kurzor. Hodnota je opět zobrazena pomocí bubliny.

5.10 Nástroj Quick Watch

Nástroj *Quick Watch* slouží pro rychlé zobrazení hodnoty výrazu. Postup použití:

- umístíme textový kurzor v okně kódu na proměnnou,
- zmáčkneme klávesovou zkratku ***Ctrl + D*** nebo otevřeme nabídku ***Debug*** a z ní položku ***Quick***
- otevře se dialogové *okno Quick Watch* se zobrazeným identifikátorem proměnné v poli ***Extenssion*** a zobrazenou hodnotou proměnné v poli ***"Current value:"***. Pokud nám výraz nevyhovuje můžeme ho libovolně změnit.
- po stisknutí tlačítka ***Recalculate*** (přepočítej) se výraz vyhodnotí a výsledek se zobrazí.

6. Základy jazyka C#

Jazyk C# vyvinula firma Microsoft. Byl představen spolu s celým vývojovým prostředím .NET. Jak název napovídá, vychází tento jazyk v mnohém z programovacího jazyka C/C++, ale v mnoha ohledech je daleko bližší programovacímu jazyku Java. Základní charakteristiky jazyka jsou:

- Jazyk C# je čistě objektově orientovaný.
- Obsahuje nativní podporu komponentového programování.

- Podobně jako Java obsahuje pouze jednoduchou dědičnost s možností násobné implementace rozhraní.
- Vedle členských dat a metod přidává vlastnosti a události.
- Správa paměti je automatická. O korektní uvolňování zdrojů aplikace se stará garbage collector.
- Podporuje zpracování chyb pomocí vyjímek.
- Zajišťuje typovou bezpečnost a podporuje řízení verzí
- Podporuje atributové programování.
- Zajišťuje zpětnou kompatibilitu se stávajícím kódem jak na binární tak na zdrojové úrovni.

Překladače jazyka C# jsou case sensitive. Rozlišují tedy velká a malá písmena

6.1 Základy objektově orientovaného programování

6.1.1 Charakteristické rysy objektově orientovaného programování

Charakteristickými rysy objektově orientovaného programování (OOP) jsou použití tříd, zapouzdření a polymorfismus. OOP slouží k urychlení tvorby výsledné aplikace. Splnění požadavku urychlení tvorby aplikace pomáhají:

- využití dědičnosti a polymorfismu k rychlejšímu sestavení datových typů,
- lepší čitelnost programu,
- menší pravděpodobnost výskytu chyb (díky zapouzdření),
- opakované použití již jednou napsaného kódu.

V prostředí OOP je možné se setkat s následujícími pojmy:

6.1.2 Abstrakce

Označuje skutečnost, že třídu je možné chápat jako černou skříňku. Nezajímá nás co je uvnitř pouze jak se to ovládá zvenčí.

6.1.3 Členské funkce

Označení funkcí patřící jedné metodě. Členské funkce implementují chování objektů třídy.

6.1.4 Dědičnost

Jazyk C++ umožňuje tvorbu datové struktury, která dědí vlastnosti od jiné datové struktury a poté doplnit do vytvářené struktury jedinečné vlastnosti. Struktury, které dědí data a funkce od jiných struktur jsou uspořádány v hierarchii dědičnosti. Takto napsaný kód není pouze sám opětovně použitelný, ale opětovně použitelné jsou i datové struktury (třídy) obsahující proměnné a funkce.

Odvozené třídy (potomci) mohou převzít vlastnosti (datové složky i metody) rodičovské třídy. Odvozená třída blíže specifikuje rodičovskou třídu. Například rodičovskou třídou může být třída ovoce a odvozenou třídou potom pomeranč.

Jedním ze základních principů OOP je, že potomek (instance odvozené třídy) může vždy zastoupit předka (instanci rodičovské třídy). Pro náš příklad to znamená, že pomeranč je ovoce. (Pozor obrácený vztah neplatí, objekt rodičovské třídy nemůže zastoupit objekt odvozené třídy. Neplatí, že každé ovoce je pomeranč.)

6.1.5 Instance

Jiné označení pro objekt, jedná se o proměnnou nebo konstantu typu třída.

6.1.6 Objekt

Objektem se označují objekty reálného světa, které slouží jako vzor třídám, stejně i jako instance tříd (objektových typů) nebo oblast paměti, s kterou lze manipulovat (konstanty, proměnné, funkce...).

6.1.7 Polymorfismus

Při práci s různými objekty provádíme podobné operace (např. kreslíme objekt). Často se ale stane, že postup vykreslování objektů je zcela odlišný (nejenom počtem a typem parametrů - jako je tomu u přetěžování). Při psaní takového programu se stane, že v době deklarace funkce neznáme typ instance, s kterou bude pracovat (viz dědičnost). C++ tuto deklaraci umožňuje pomocí použití polymorfismu. Funkci deklarujeme v nadřazené třídě objektů jako virtuální a až teprve v jednotlivých objektech (potomcích) ji implemntujeme. Při vykreslování objektu program zajistí, že je vyvolána správná funkce.

Poznámka - někdy bývá označováno i přetěžování funkcí za druh polymorfismu.

6.1.8 Předefinování

Pojem předefinování úzce souvisí s polymorfismem. Předefinování funkce znamená, že je funkce znovu definovaná v odvozené třídě (aby správně reagovala na své volání). Předefinovaná funkce musí být v rodičovské třídě deklarována jako virtuální.

6.1.9 Přetěžování (overloading) funkcí

Přetěžování = definice více funkcí se stejným identifikátorem (jménem). Funkce se od sebe rozlišují pomocí počtu a typu parametrů.

6.1.10 Přetěžování (overloading) operátorů

Obdoba přetěžování funkcí. Přetížení operátorů znamená rozšíření operátoru na uživatelem definované typy (výčtové a objektové). Rozlišení operace, kterou má překladač uskutečnit je opět pomocí počtu a typu parametrů. Typickým příkladem přetíženého operátoru je operátor +. Jsou li použity jako operandy čísla, jedná se o matematickou operaci sčítání. Jsou-li ale použity jako operandy řetězce, jedná se o operaci složení.

6.1.11 Rozhraní (interface)

Prostředek pro výměnu informací mezi třídami, funkcemi, moduly ... .

6.1.12 Zapouzdřenost

Reálné objekty jsou v programu složeny ze svých vlastností (data) a chování (metody). Vlastnosti a metody zastřešuje (zabaluje) třída, která je reprezentuje konkrétními hodnotami.

- izolace nebezpečných operací
- vazba mezi daty a metodami, které s nimi manipulují
- public metody a data –vnější ovladače objektu
- private, protected metody a data –vnitřní operace

poznámka:

Jeden C# soubor může obsahovat i **více deklarací tříd**, na rozdíl od Java programu**.** Deklarace jedné třídy nebo struktury může být rozložena **do více souborů.,** tzv. "partial classes"

6.2 Direktivy preprocesoru

Podobně jako v jazyce C/C++ i jazyk C# umožňuje využít několik direktiv preprocesoru a tak řídit předzpracování zdrojového kódu. Direktivy preprocesoru začínají znakem: #. Existuje celá řada různých direktiv **#define**, **#undef**, **#if**, **#endif**, **#elif, #else**. Pomocí nich lze ovlivnit, které části kódu budou zpracovávány.

Další zajímavou direktivou preprocesoru je **#pragma**. Ta umožňuje zakázat či povolit vypisování některých varovných hlášení.

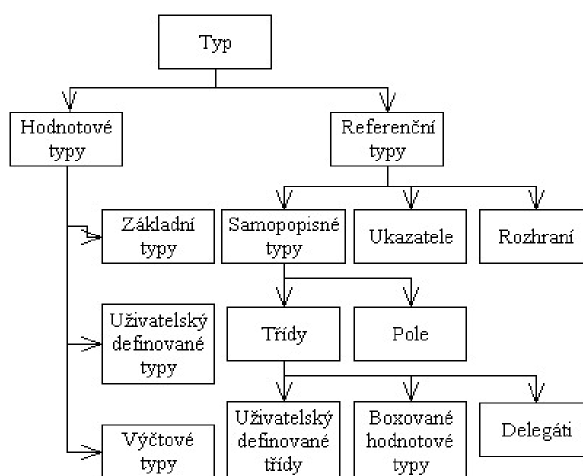
Velice užitečná je direktiva **#region** a **#endregion**. Ty jsou preprocesorem ignorovány, ale využívá jich například Visual Studio 2005. Umožňují definovat

oblasti v kódu. Tyto pak umí VS formátovat (zabalit, rozbalit). Takto zpřehledňují vytvořený zdrojový program.

6.3 Datové typy

Jazyk C# (podobně jako jiné programovací jazyky v prostředí .NET Framework) využívá společný typový systém CTS. Jazyk sám žádné speciální, s ostatními jazyky nekompatibilní, typy nepřináší. Typový systém je rozdělen do dvou hlavních kategorií:

- hodnotové typy
- referenční typy.



Obrázek 8: Základní typy

6.3.1. Hodnotové typy

Instance hodnotových typů (proměnné) jsou ukládány na zásobník programu a program s nimi pracuje přímo. Hodnotové proměnné, jak napovídá jejich název, obsahují pouze hodnotu proměnné daného typu bez jakýchkoliv doplňujících informací. Hodnotové typy v jazyce C# lze dále rozdělit do tří kategorií: základní typy, struktury a výčtové typy.

Primitivní datové typy:

Základní typy se příliš neliší od jiných programovacích jazyků. Primitivní datové typy se dělí na celočíselné a reálné. Jednotlivé typy jsou popsány v tabulkách 1 a 2.

Typ	**Rozsah**	**Velikost (byty)**	**Typ v .NET Framework**
byte	bez znaménka (hodnoty 0 až 255)	1	Byte
bool	Hodnoty true nebo false	1	Boolean
sbyte	se znaménkem (hodnoty -128 až 127)	1	SByte
char	znaky unicode	2	Char
short	se znaménkem (hodnoty -32 768 až 32 767)	2	Int16
ushort	bez znaménka (hodnoty 0 až 65 535)	2	UInt16
int	se znaménkem (hodnoty -2 147 483 648 až 2 147 483 647)	4	Int32
uint	bez znaménka (0 až 4 294 967 295)	4	UInt32
long	se znaménkem -9 223 372 036 854 775 808 až 9 223 372 036 854 775 807	8	Int64
ulong	bez znaménka 0 až 0xffffffffffffffff	8	UInt64

Tabulka 1: Primitivní datové typy celočíselné

Při přiřazování hodnot proměnným nesmíme zapomenout na to, jak se v C# s čísly zachází. Pokud pracujeme s typem byte, nelze například jednotlivé proměnné typu byte sčítat bez explicitní konverze do byte.

```
byte x = 1, y = 2, sum = 0;
sum = (byte)(x + y); // explicitni konverze
```

Typ	**Rozsah**	**Velikost (byty)**	**Typ v .NET Framework**
float	pohyblivá řádová čárka, jednoduchá přesnost, 7 platných cifer	4	Single
double	pohyblivá řádová čárka , jednoduchá přesnost, 16 platných cifer.	8	Double
decimal	pevná řádová čárka, 28 platných cifer, vhodný pro finanční operace	16	Decimal

Tabulka 2. Reálné datové typy a typ decimal

Struktury:

Struktury jsou vlastně uživatelsky definovaným hodnotovým typem. Je to jakási odlehčená třída. Od tříd se ale liší. Proměnné typu struktura jsou umístěny na zásobník, struktury nepodporují dědičnost a struktura sama se nemůže stát základem pro vytvoření nového typu. Struktury jsou logickým sdružením atributů, metod, vlastností, indexerů, operátorů, a případně dalších vnořených typů. Struktury mohou mít definovány dokonce i konstruktory.

Když máme takto vytvořenou strukturu a chceme ji někde použít, je třeba vytvořit její instanci pomocí operátoru **new** .

```
struct Zvirata_A_Nohy {
   int pocetNohou;
   string nazevZvirete;
   public Zvirata_A_Nohy(int novyPocetNohou, string novy-
NazevZvirete) {
      pocetNohou = novyPocetNohou;
      nazevZvirete = novyNazevZvirete;
   }
   public int vratPocetNohou() {
      return pocetNohou;
   }
   public string vratNazevZvirete() {
      return nazevZvirete;
   }
}
```

Výčtové typy:

Výčtové typy se hodí pro celočíselné konstanty. Právě díky tomu, že každé položce ve výčtovém typu můžeme přiřadit i jejich hodnoty. Dokonce lze dvěma položkám přiřadit stejnou hodnotu a C# si potom vybere jednu hodnotu jako primární (z důvodů reflexe a řetězcových konverzí). Výčtové typy mají následující omezení:

- nemohou definovat své vlastní metody
- nemohou implementovat rozhraní
- nemohou definovat své indexery nebo vlastnosti

```
enum DnyVTydnu {
   neděle, pondělí, úterý, středa, čtvrtek, pátek, sobota};
```

6.4 Referenční typy

Na rozdíl od hodnotových typů - referenční neuchovávají přímo hodnotu samotnou, nýbrž odkaz na místo v paměti, konkrétně na hromadě, kde je skutečná instance uložena. Tuto instanci nazveme objektem. V jazyce C# existují tyto referenční typy:

typ object – jde o alias třídy System.Object. Všechny ostatní třídy rozšiřují tuto třídu. Object je tedy společným základem pro všechny ostatní typy. Do takové proměnné můžeme přiřadit jakýkoliv jiný typ;

typ string – slouží k uložení textových řetězců. Jde opět o alias k třídě System.String. Práce s proměnnou typu String je podobná jako v Javě. Takovouto proměnnou lze přímo naplnit řetězcovou konstantou (bez nutnosti použití operátoru new). String se od ostatních tříd liší při porovnávání, jsou porovnávány hodnoty objektů (textové řetězce), ne pouze hodnoty odkazů;

typ třída (class), typ rozhraní (interface), typ pole, typ delegát (delegate).

6.5 Proměnné

6.5.1 Deklarace:

datový_typ identifikátor;

příklad:

```
int i;
```

Kompilátor nás nenechá použít tuto proměnnou bez inicializace. Pro proměnnou alokuje 4 byty. Inicializujeme pomocí operátoru přiřazení =.

```
i = 10;
```

Nebo naráz:

```
int i = 10;
```

Několik proměnných stejného datového typu:

```
int x = 10, y =20; // x a y jsou typu int
```

C# považuje neinicializovanou proměnnou za chybu. Členské proměnné tříd, pokud nejsou inicializovány explicitně, jsou nastaveny na nulovou (default) hodnotu. Lokální proměné metod musí být inicializovány explicitně.

6.5.2 Životnost

- Členská proměnná třídy existuje po dobu existence objektu.
- Lokální proměnné existují po dobu vykonávání bloku kódu.
- Proměnné deklarované ve smyčkách existují po dobu vykonávání bloku smyčky.

6.5.3 Konstanty

```
const int a = 100; // hodnotu nelze měnit
```

- Musí být deklaraci inicializovány.
- Musí být vyčíslitelné v čase kompilace. Nelze inicializovat výrazem, použijte read-only field. a jsou implicitně statické.

6.6 Operátory

Množina operátorů v jazyce C# je až na pár výjimek identická s operátory v jiných programovacích jazycích.

<table>
<tr><th>Kategorie</th><th>Operátory</th></tr>
<tr><td>Primární</td><td>(x) x.y f(x) a[x] x++ x-- new typeof sizeof checked unchecked</td></tr>
<tr><td>Unární</td><td>+ - ! ~ ++x --x (T)x</td></tr>
<tr><td>Multiplikativní</td><td>* / %</td></tr>
<tr><td>Aditivní</td><td>+ -</td></tr>
<tr><td>Bitové posuny</td><td><< >></td></tr>
<tr><td>Relační</td><td>< > <= >= is as</td></tr>
<tr><td>Rovnost</td><td>== !=</td></tr>
<tr><td>Logické AND (bitové)</td><td>&</td></tr>
<tr><td>Logické XOR (bitové)</td><td>^</td></tr>
<tr><td>Logické OR (bitové)</td><td>|</td></tr>
<tr><td>AND</td><td>&&</td></tr>
<tr><td>OR</td><td>||</td></tr>
<tr><td>Podmíněný výraz</td><td>? :</td></tr>
<tr><td>Přiřazení</td><td>= *= /= %= += -= <<= >>= &= ^= |=</td></tr>
</table>

Tabulka 3. tabulka priority operátorů

Příklady:

- **typeof** - slouží k reflexi. Vrátí instanci třídy System.Type.

```
Type t = typeof(Console);
foreach(MemberInfo info in t.GetMembers()) Con-
sole.WriteLine(info);
```

- **sizeof** - slouží k zjištění velikosti hodnotových typů. Tato operace je považována za nebezpečnou a proto musí být tento operátor umístěn v bloku unsafe.

Definování bloku unsafe se explicitně zříkáme bezpečnostních kontrol prostředí .NET. Blok je definován pomocí klíčového slova unsafe a složenými závorkami. Je nutné překladači povolit unsafe bloky.

```
// compile with: /unsafe
class Program
{
 unsafe
 static void Main(string[] args)
 {
  int size = sizeof(int);
  Console.WriteLine(size);
  Console.ReadLine();}}
```

- Při porovnávání stringů pomocí operátoru == je porovnávána hodnota. Pokud chceme porovnat reference, musíme alespoň jeden řetězec přetypovat.

```
string a="hello";
string b="hello";
string c=String.Copy(a);
Console.WriteLine(a==c); //True
Console.WriteLine((object)a==(object)b); //True,same con-
stant.
Console.WriteLine((object)a==(object)c); //False
```

Operátory is a as

- Pomocí operátoru is jsme schopni zjistit, zda je daný prvek instancí dané třídy, případně implementuje-li dané rozhraní.

```
if (someObject is SomeInterface) //vrátí true v případě, že
someObject implementuje dané rozhraní
{
  return someObject;
}
else
{
  return null;
}
```

- Operátor as provádí přetypování na daný typ. Pokud přetypování není možné, vrací null.

```
return someObject as SomeInterface;
```

6.7 Třídy

Třídou se v C# rozumí programová datová struktura. Třída může obsahovat tyto členy:

1. **položky** (field) – členské proměnné, udržují stav objektu

2. **metody** – jde o funkce, které implementují služby objektem poskytované. Každá metoda má návratovou hodnotu, pokud nic nevrací, je označena klíčovým slovem void. V jazyce C# lze přetěžovat metody. Přetížené metody se musí lišit v počtu parametrů, v typu parametrů nebo v obojím. Výjimečná metoda je statická metoda s názvem Main. Právě pomocí této metody je projekt spouštěn. Je-li v projektu definováno více metod Main, je nutné při kompilaci zadat jako parametr jméno jedné třídy z těchto tříd. Metoda Main této třídy je pak spuštěna. Každá třída může obsahovat maximálně jednu metodu Main;

3. **vlastnost** (property) – je také označována za chytrou položku. Navenek vypadají jako položky, ale umí kontrolovat přístup k jednotlivým datům;

4. **indexer** – u některých tříd je výhodné definovat operátor []. Indexer je speciální metoda, která umožňuje aby se daný objekt choval jako pole;

5. **operátory** – v jazyce C# máme možnost definovat množinu operátorů sloužících pro manipulaci s jejími objekty;

6. **událos**t (event) – jejím účelem je upozorňovat na změny, které nastaly např. v položkách tříd.

U jednotlivých členů třídy můžeme použít modifikátory přístupu. Modifikátor je nutné aplikovat na každého člena zvlášť. Jeho uvedení není povinné, implicitní hodnota je private. Možné modifikátory jsou:

1. public – člen označený tímto modifikátorem je dostupné bez omezení;
2. private – člen je přístupny pouze členům stejné třídy;
3. protected – přistupovat k takovému členu můžeme uvnitř vlastní třídy a ve všech třídách, pro které je třída základem;
4. internal – člen je přístupný všem v rámci jedné assembly.

Dalším možným modifikátorem je modifikátor static, pomocí něj lze deklarovat, že je daný člen třídy statický. Uvnitř nestatických metod třídy můžeme použít klíčové slovo this, to reprezentuje referenci objektu na sebe sama.

6.7.1 Metody a parametry

Parametry jsou obvykle předávány do metody hodnotou. Funkce získá kopii skutečných parametrů a jakékoliv modifikace uvnitř těla metody se nepromítnou zpět. V jazyce C# máme dvě řešení.

1. Předání parametru odkazem. Metoda si nevytváří vlastní kopii, nýbrž přímo modifikuje proměnnou, která ji byla předána. Takovéto parametry označíme klíčovým slovem ref. Předávané parametry musí být inicializovány.

2. Definovat parametry jako výstupní. Takové parametry označíme klíčovým slovem out. Parametry pak přenášejí své hodnoty směrem ven z metody. Hlavním rozdílem oproti předávání parametrů odkazem je, že předané proměnné nemusí být inicializovány před voláním metody. Uvnitř těla funkce je brán parametr jako neinicalizovaný.

Ukázka kódu:

```
class Test
{
  public void Swap(ref int a, ref int b){
    int tmp;
    tmp=a;
    a=b;
    b=tmp;
  }
  public void GetXY(Point somePoint,out int x, out int y){
    x=somePoint.GetX();
    y=somePoint.GetY();
  }
  public static void Main()
  {
    int a=5,b=3;
    Swap(ref a,ref b);
    GetXY(new Point(1,2),out a,out b);
  }
}
```

V jazyce C# musí být každá funkce metodou třídy. Pokud potřebujeme samostatné funkce, můžeme vytvořit statické metody třídy. Takové jsou např. elementání funkce třídy System.Math nebo metoda WriteLine třídy System.Console:

```
Console.WriteLine("sin({0})={1}",x,Math.Sin(x));
```

Metoda s proměnným počtem parametrů:

Chceme-li definovat metodu s proměnným počtem parametrů, využijeme klíčové slovo params. To musíme uvést před posledním parametrem. Typ tohoto parametru musí být pole.

```
public void someMethod(params object[] p); //deklarace me-
tody s proměnným počtem parametrů
```

6.7.2 Konstruktory a destruktory

Konstruktor nevrací žádnou hodnotu (ani void). Každá třída má definovaný implicitně konstruktor bez parametrů a s prázdným tělem. Každý objekt by pak měl být vytvořen pomocí operátoru new. Jiným typem konstruktoru je tzv. statický konstruktor. Pomocí něj lze inicializovat statické položky. Jiné položky pomocí tohoto typu konstruktoru inicializovat nelze. Tento konstruktor nesmí mít žádné parametry a je vyvolán ještě před vytvořením prvního objektu. Pro statické

konstruktory dále platí, že jsou spouštěny v náhodném pořadí. Nelze se tedy v jednom spoléhat na druhý.

```
class Point
{
  static short dimension;
  short x=0
  private short y=0;

  public Point()
  {
  }
  public Point(short nx, short ny)
  {
    x=nx;
    y=ny;
  }
  static Point()
  { //statický konstruktor
    dimension=2;
  }
  static void Main()
  {
    Point a = new Point();
    Point b = new Point(1,2);
  }
}
```

Chceme-li zavolat konstruktor stejné třídy, můžeme to udělat pomocí klíčového slova this. Syntaxi demonstruje následující příklad.

```
public Point(Point p):this (p.GetX(), p.GetY())
{
}
```

Destruktor má název začínající tildou (~) následovaný jménem třídy. Od konstruktoru se liší tím, že nesmí mít žádné formální parametry a nemůže být přetížen. Pokud to není nutné, nemusí být definován.

6.7.3 Dědičnost a polymorfismus

Představuje jeden ze základních rysů objektově orientovaného programování. Dědění tříd umožňuje definovat nové třídy na základě tříd, které již existují. Třídy, od kterých odvozujeme nové třídy jsou nazývány rodičovské třídy. Odvozené třídy jsou označovány jako potomci nebo dceřiné třídy. Skupiny provázaných tříd (vztahem předek-potomek) jsou nazývány dědické hierarchie.

Odvozená třída "zdědí" vlastnosti svých předků - bude obsahovat všechny nestatické datové složky svých předků a bude moci používat jejich metody, typy a statické složky (pokud jí to dovolí přístupová práva).

Dědění se využívá v OOP pro vyjádření specifikace: Předek popisuje abstraktnější pojem, zatím co potomek popisuje konkrétnější, přesněji vymezený pojem

Jazyk C# definuje pouze jednoduchou dědičnost. Dědičnost v definici třídy vyjádříme dvojtečkou uvedenou za jménem třídy a názvem základní třídy. Pro metody odvozené třídy pak platí:

chceme-li předefinovat veřejnou metodu třídy kterou dědíme, musíme použít klíčové slovo new. V tomto případě záleží na typu reference, jaká metoda se zavolá;

chceme-li realizovat polymorfismus, použijeme virtuální metody. Postup je následující: metodu základní třídy označíme jako virtuální (klíčové slovo virtual) a metodu v odvozené třídě označíme klíčovým slovem override (má se chovat polymorfně).

```
class A
{
  public void SomeMethod()
  {
  }
  public virtual void AnotherMethod()
  {
  }
}
class B : A
{
  public new void SomeMethod()
  {
    //původní metoda je překryta
  }
  public override void AnotherMethod()
  {
  }
}
class Run
{
  static void Main()
  {
    A a=new B();
    a.SomeMethod(); //spustí původní metodu třídy A
    a.AnotherMethod(); //spustí metodu třídy B
  }
}
```

6.7.4 Konstruktory v odvozených třídách

Chceme-li volat konstruktor základní třídy v odvozené třídě, můžeme k tomu využít klíčové slovo base. Syntaxi demonstruje následující příklad:

Ukázka kódu :

```
public SomeName(…):base(…){ … }
```

6.8 Pole

Pole chápeme jako množinu proměnných stejného datového typu. S takovým polem pak zacházíme jako s celkem. Na jednotlivé prvky pole (proměnné) se můžeme odkazovat pomocí jejich indexů pole a poté s nimi pracovat jako s obyčejnými proměnnými. Každý počáteční prvek pole v C# má index 0, pokud tedy máme pole délky N, index posledního prvku bude N-1.

6.8.1 Deklarace pole

V C# existují dva způsoby, jak deklarovat pole. První ze způsobů více naznačuje objektovost polí. Ten si ukážeme jen "ve zkratce", více se budeme zabývat druhým, klasickým a určitě i praktičtějším způsobem.

Jazyk C# umožňuje vytvořit instanci pole pomocí statické metody **System.Array.CreateInstance()**. Tento způsob více odpovídá objektové charakteristice tvorby objektů.

Ukázka kódu:

```
using System;
...
Array pole = Array.CreateInstance(typeof(int), 4);
```

Vytvořili jsme instanci třídy **Array** typu **int** se čtyřmi prvky. Pokud budeme toto pole chtít naplnit, nemůžeme již k jednotlivým prvkům přistupovat pomocí

indexů, jako jsme byli zvyklí u běžných polí. Je třeba k těmto prvkům přistupovat pomocí metod **SetValue()** a **GetValue()**.

Ukázka kódu :

```
...
pole.SetValue(3, 0); // na nultou pozici pole vložíme číslo
3 Console.WriteLine(pole.GetValue(0)); // vytiskneme 0.
prvek pole ...
```

Stejným způsobem, ale i s možností přistupovat k jednotlivým prvkům pole pomocí indexů, můžeme tehdy, pokud vytvoříme pole druhou (následující) konstrukcí. První způsob tedy opustíme. Pro jednorozměrná pole v jazyce C# platí deklarace:

```
typ[] nazev_pole;
```

6.8.2 Vícerozměrná pole

Pokud chceme deklarovat vícerozměrné pravidelné pole, píšeme nejprve typ pole, následují hranaté závorky, do kterých napíšeme pouze čárky podle toho, kolikarozměrné pole chceme používat.

Příklad: Deklarace 2-rozměrného pole s názvem **pole** typu **int**

```
int[,] pole = new int[2,2] { {2, 1}, {4, 3} };
```

6.8.3 Vícerozměrná nepravidelná pole

V jazyce C# lze používat i nepravidelná pole. Nepravidelná pole se vyznačují různou délkou jednotlivých "řádků" pole. Pokud například chceme vytvořit pole, které má v prvním řádku 3 prvky a v druhém 2 prvky, provedeme to následovně:

Ukázka kódu :

```
int[][] pole;
pole = new int[2];
pole[0] = new int[3];
pole[1] = new int[2];
```

Přístup k jednotlivým prvkům pole se pak provádí následovně: **pole[0][0]** nám vrátí hodnotu prvku v první řadě a prvním sloupci.

6.8.4 Kopírování polí

Místo obvykle používaného cyklu pro kopírování pole **x** do pole **y,** lze použít metodu **CopyTo()** dostupnou všem polím.

Ukázka kódu:

```
for(int i=0; i<=x.Length; i++) y[i] = x[i];
CopyTo() x.CopyTo(y,0);
```

První parametr udává cílové pole, do kterého se má kopírovat druhý parametr udává počáteční index, na který se bude ukládat. Pokud cílové pole od daného počátečního indexu nebude dostatečně dlouhé, vyvolá se výjimka.

6.9 Podmíněné příkazy

6.9.1 Příkaz if

Má stejnou syntaxi jako v ostatních programovacích jazycích. Více příkazů se musí uzavřít do bloku {}. Pro jeden příkaz nemusíme požívat závorky.

```
if (podmínka)
   statement(s)
else
   statement(s)
```

6.9.2 Příkaz switch...case

Příkaz if je přehledný, pokud testujeme dvě podmínky. Pak se stává nepřehledným. Příkaz switch používáme pro testování více vstupních podmínek. Jeho zápis je ve tvaru:

```
switch (vyraz) {
   case hodnota1 :
      prikazy;
   break;
   case hodnota2 :
      prikazy;
   break;
   ...
   default : prikazy;
}
```

6.10 Cykly

6.10.1 Cyklus while

```
while (podminka) { ... }
```

Cyklus while se provádí tak dlouho, dokud podmínka podminka nabývá hodnoty true. Pokud podmínka nabude hodnoty false, cyklus skončí (tělo cyklu se přeskočí). U toho cyklu se může stát, že se neprovede ani jednou, neboť pomínka se vyhodnocuje před tělem cyklu.

6.10.2 Cyklus do-while

Cyklus do-while je podobný jako cyklus while s tím rozdílem, že podmínka se vyhodnocuje až po průchodu těla cyklu. Tedy tělo cyklu se provede pokaždé alespoň jednou.

6.10.3 Cyklus for

```
for(int i = 0; i < 10; i++) System.Console.WriteLine(i);
```

6.10.4 Cyklus foreach

Ideální k procházení všech prvků kontejneru.

```
using System;
using System.Collections;
...
ArrayList a = new ArrayList();
a.Add(1);
a.Add("string");
a.Add(new object);
foreach(object obj in pole)
   Console.WriteLine(obj.GetType());
...
```

6.11 Skokové příkazy

Příkazy, které způsobí, že program přeruší přirozenou posloupnost plnění příkazů (tedy za sebou, jak jsou napsány), se nazývají skokové příkazy. Tehdy program přejde na jiné místo v programu rozdílné od této přirozené posloupnosti.

6.11.1 Příkaz break

Příkaz break způsobí, že program vyskočí z nejbližšího cyklu, ve kterém se právě nachází. Kromě cyklů se break smí použít i v příkazu switch.

6.11.2 Příkaz continue

Lze použít pouze v cyklech. Pokud použijeme tento příkaz, způsobí to, že přeskočíme zbytek těla cyklu a začneme novou iteraci (další opakování cyklu).

6.11.3 Příkaz goto

Tento příkaz se většinou v novějších programovacích jazycích nesetkával s oblibou. Platilo, že čím více příkazů goto, tím horší je kvalita programu. Jazyk C# tento příkaz obsahuje, ale snaží se zabránit jeho zneužití, např. nelze skočit dovnitř blokového příkazu. Tento příkaz je doporučován pro použití v příkaze switch nebo pro předání řízení do vnějšího bloku, ale ostatní použití není nepovolené.

6.11.4 Příkaz return

Tímto příkazem se vrací řízení volající funkci. Příkaz return může být spojen i s hodnotou, pokud metoda, která return použije, má návratovou hodnotu jinou než void.

6.11.5 Příkazy checked a unchecked

Pokud používáme konverzi typů, někdy se stane, že při konverzi z jednoho typu na druhý si chceme být jisti, že převod proběhl úspěšně. C# pro ten případ používá příkaz checked. Když potom chceme ověřit správnost konverze, uvedeme naši konverzi v kontrolovaném bloku příkazu checked.

6.11.6 Příkaz throw

Příkaz throw slouží k vyvolání výjimky. Uplatní se tehdy, pokud chceme ošetřit nějakou část kódu. V případě, kdy se vyskytnou jiné podmínky v programu, než jsme si představovali, můžeme pomocí throw vyvolat nějakou výjimku.

7. Základy práce s technologií .NET

7.1 Knihovny tříd

Ve vytvářených aplikacích budeme muset používat prostředky, které nám umožní pracovat s AutoCADem. V prostředí .NET se těmto prostředkům říká třídy a funkce. Třídy a funkce jsou uloženy v knihovnách tříd. ObjectARX obsahuje několik knihoven tříd pro různé použití. Ve výkladu se dozvíte jaké třídy a funkce knihovny obsahují a které knihovny musíte přidat do projektu, aby pracoval správně.

Managed třídy implementují funkčnost databáze a umožňují psát aplikace, které čtou a píšou do souborů ve formátu drawing(DWG). Také poskytují pžístup k UI elementům AutoCADu včetně příkazového řádku, dialogových oken a dalších.Pro úplný přehled manager wrapper tříd můžete nahlédnout do AutoCAD Managed Class Reference.

ObjectARX managed wrappery jsou umístěny ve dvou souborech:

- acdbmgd.dll obsahuje wrappery pro ObjectDBX™ API. Tato DLL je součástí Autodesk® RealDWG SDK a je obsažena v AutoCADu.
- acmgd.dll obsahuje wrappery pro většinu API AutoCADu. Tato DLL je obsažena v AutoCADu.

Primitivní datové typy C++ a ObjectARX jsou are mapovány na korespondující ekvivalenty v .NET. Většina tříd ObjectARX mapuje jednu managed wrapper třídu. Ačkoli jsou zde vyjímky, první čtyři písmena názvu třídy ObjectARX často poskytují klíč ke korespondující managed jmenný prostor(namespace).

Následující tabulka ukazuje nejobvyklejší mapování prefixů tříd ObjectARX na .NET jmenné prostory.

ObjectARX class prefixes and .NET namespaces Unmanaged Prefix	**Managed Namespace**
AcDb	Autodesk.AutoCAD.DatabaseServices
AcRx	Autodesk.AutoCAD.Runtime
AcEd	Autodesk.AutoCAD.ApplicationServices
AcUt	Autodesk.AutoCAD.DatabaseServices, Autodesk.AutoCAD.ApplicationServices
AcCm	Autodesk.AutoCAD.Colors
AcGe	Autodesk.AutoCAD.Geometry
AcGi	Autodesk.AutoCAD.GraphicsInterface
AcLy	Autodesk.AutoCAD.LayerManager
AcPl	Autodesk.AutoCAD.PlottingServices

Tabulka 4. mapování prefixů na jmenné prostory

7.2 Vytvoření nového projektu .NET

Základní kroky k vytvoření .NET solution použitím Visual Studia a manager wrapperů ObjectARX jsou stejné ať použijete Microsoft Visual C# .NET nebo Visual Basic .NET.

7.2.1 Ručně vytvořený:

1. Ve Visual Studiu .NET, vytvořte class library solution a projekt.
2. Vyberte Add Reference z Project menu nebo Solution Exploreru.
3. Zadejte cestu k umístění AutoCADu a vyberte acdbmgd.dll and acmgd.dll.
4. V hlavním souboru třídy přidejte prostory, které budete používat.

Například:

```
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.DatabaseServices;
using Autodesk.AutoCAD.Runtime;
```

7.2.3 Pomocí nástroje AutoCAD Managed C# Application Wizard:

- Nejdříve je nutné ho nainstalovat a to předtím, než spustíme Visual Studio. Pomocník se nachází v balíku ObjectARX SDK v adresáři *\utils\ObjARXWiz*. Pomocníka musíme nejprve rozbalit a poté můžeme spustit instalaci. Přes celou instalaci budete naváděni.,
- Po skončení instalace můžeme spustit Visual Studio
- V dialogovémokně New Project vybereme Visual C# projekty a šablonu AutoCAD Managed C# Project. Pojmenujeme a nastavíme adresáře, kam se soubory uloží. Povrdíme.
- Objeví se AutoCAD Manager CSharp Application Wizard. Necháme Enable Unmanaged Debugging neoznačené. Registered Developer Symbol bude mít hodnotu, kterou jsme vložili, když jsme pomocníka instalovali. Klikneme na Finis pro vytvoření projektu
- Když se podíváme na projekt, který pomocník vytvořil, v Solution Explorer vidíme, že na acdbmgd a acmgd už byly vytvořeny reference. V hlavním souboru .cs byly rovněž přidány jmenné prostory a atribut CommandMethod

7.3 Tvorba aplikace .NET

7.3.1 Definování Příkazů AutoCADu:

Poté, co máme vytvořen .NET projekt, jsme připraveni definovat třídy, které budou implementovat a vykazovat funkcionalitu naší aplikace. Musíme dodržovat specifický protokol AutoCADu k registraci příkazů.

Pro každý příkaz AutoCADu, který je definován, musí aplikace obsahovat metodu s atributem CommandMethod. Tento atribut může mít jeden nebo více atributů. V jeho nejjednodušší formě předáváme pouze jeho název.

Příkazové metody by měly být deklarovány buď jako instance nebo statické metody. Statické příkazové metody jsou v C# deklarovány jako statická klíčová slova.

7.3.2 Používání transakcí

1. Začněte transakci před try blokem zavoláním metody Transaction.StartTransaction()
2. Uvnitř try bloku použijte objekt transaction pro operace s databází
3. Předejte transakci na konci try bloku zavoláním metody Transaction.Commit()
4. Uvolněte transakci zavoláním metody Transaction.Dispose()

Tato sekvence zaručuje, že objekt transaction je zrušen nehledě na výjimky, které se mohou vyskytnout uvnitř try bloku.

7.3.3 Přístup k uživatelskému rozhraní

AutoCAD managed třídy dovolují aplikacím používat mnoho AutoCAD UI zdrojů a připojit efektivně nové formuláře do rozhraní AutoCADu. Některé třídy ve jmenném prostoru Autodesk.AutoCAD.Windows poskytují pístup do dialogových oken, jako je Linetype and Color. Tyto třídy poskytují metodu ShowDialog, která zobrazí formulář. Při používání těchto tříd aplikace automaticky získává trvalou velikost a pozici dialogového okna.

Třídy Autodesk.AutoCAD.Windows také předkládají rozhraní některých rozšiřitelných UI elementů včetně palet, tray položek, status baru. Palety nástrojů obdrží rozsáhlé pokrytí v třídách a rozhraní ve jmenném prostoru Autodesk.AutoCAD.Windows.ToolPalette. Ikona AutoCADu zrovna jako pick point pick set bitmapy, mohou být zpřístupněny přes třídu Autodesk.AutoCAD.Windows.Visuals.

K implementaci uživatelských formulářů do AutoCAD managed API, aplikace rozšiřují třídy .NET System.Windows.Forms přímo. Nicméně by by takové aplikace neměly volat metodu Form.ShowDialog. Místo toho by měly používat metody Autodesk.AutoCAD.ApplicationServices.Application.ShowModalDialog a ShowModelessDialog pro zobrazení jejich uživatelských formulářů. Použití

Form.ShowDialog v rozšiřující aplikaci AutoCADu může vyústit v neočekávané chování.

7.3.4 Používání ExtensionApplication a CommandClass Attributů

Když AutoCAD načte managed aplikaci, dotáže se aplikace na uživatelský atribut ExtensionApplication. Pokud je atribut nalezen, AutoCAD nastaví přidružený typ atributu jako vstupní bod aplikace. Pokud nalezen není, AutoCAD prohledává všechny exportované typy implementace IExtensionApplication. Pokud není nalezen vůbebec, AutoCAD jednoduše přeskočí inicializační krok aplikace. Atribut ExtensionApplication může být připojen pouze k jednomu typu.

Atribut CommandClass může být deklarován pro kterýkoliv typ, který definuje příkazové handleny AutoCADu. Pokud aplikace užívá atribut CommandClass, musí deklarovat instanci tohoto atributu pro každý typ, který obsahuje metada příkazového handlenu AutoCADu.

8. Příklady

8.1 Vypsání textu do příkazové řádky

Zde vytvoříme knihovnu .NET dll, která může být načtena do AutoCADu. Přidá nový příkaz do AutoCADu pojmenovaný „AhojSvete”. Když uživatel spustí příkaz, text “Ahoj Svete” bude vytisknut do příkazového řádku.

1) Spusťte Visual Studio a založte nový projekt pomocí File> New> Project. V dialogovém okně New Project vyberte Visual C# Projects . Vyberte šablonu “Class Library”. Pojmenujte ji například „Projekt1“ a nastavte lokaci, kam mají být vytvořené soubory uoženy. Potvrďte OK pro vytvoření projektu.

2) Pokud není solution explorer zobrazený, můžete tak učinit z nabídky „View” výběrem „Solution Explorer”. To nám umožní procházet soubory v projektu a přidávat reference na obsluhované nebo COM Interop assemblies.

3) V Class1.cs si povšimněte, že byla již vytvořena veřejná třída. Přidáme náš příkaz do této třídy. Abychom to mohli udělat, potřebujeme třídy AutoCAD .NET managed wrappers. Tyto wrappery jsou obsaženy ve dvou obsluhovaných modulech. K přidání referencí na tyto moduly klikněte pravým tlačítkem na „References” v Solution Exploreru and zvolte „Add Reference”. V „Add Reference” dialogu vyberte „Browse”. Zvolte cestu do adtesáře AutoCadu. (C:\Program Files\AutoCAD 2007\) Najděte „acdbmgd.dll” a potvrďte OK. Klikněte na „Browse” znovu a vybrete „acmgd.dll” a opět potvrďte. V Solution Explorer klikněte pravým tlačítkem na obě knihovny a zvolte „Properties“. V obou případech změňte hodnotu Copy Local na false.

Připojení do AutoCAD Managed API – AcMgd.dll a AcDbMgd.dll

4) Použijte Object Browser k prozkoumání tříd dostupných v těchto obslužných modulech. (View > Object Browser. Rozbalte „AutoCAD .NET Managed Wrapper” (acmgd) objekt. V tomto příkladu bude použita instance „Autodesk.AutoCAD.EditorInput.Editor ” k zobrazení textu v příkazovém řádku AutoCADu Rozkřikněte objekt „ObjectDBX .NET Managed Wrapper” (acdbmgd). Třídy v tomto objektu budou použity k přístupu a editaci entit ve výkresu AutoCADu.

5) Když máme reference na třídy, můžeme je importovat. V Class1.cs nad deklarací Class1 importujte následující jmenné prostory (namespace): ApplicationServices, EditorInput a Runtime.

```
using Autodesk.AutoCAD.ApplicationServices;
using Autodesk.AutoCAD.EditorInput;
using Autodesk.AutoCAD.Runtime;
```

6) Nyní přidáme náš příkaz do třídy Class1. K přidání příkazu, který může být zavolán z AutoCADu použijeme atribut „CommandMethod”. Tento atribut je poskytován jmenným prostorem Runtime. Přidejte následující atribut a funkci do Class1.

```
[CommandMethod("AhojSvete")]
public void AhojSvete()
{
}
```

7) Když je příkaz „AhojSvete” spuštěn v AutoCADu, bude zavolána funkce AhojSvete. V této funkci přidáme instanci třídy editor, která obsahuje metody pro přístup příkazového řádku AutoCADu. Editor pro aktivní dokument v AutoCADu může být vrácen použitím třídy Application. Poté, co je editor vytvořen, použijeme metodu WriteMessage pro zobrazení „Ahoj Svete“ na příkazovém řádku. Přidejte následující do funkce AhojSvete:

```
Editor ed =
Application.DocumentManager.MdiActiveDocument.Editor;
ed.WriteMessage("Ahoj Svete");
```

8) K otestování v AutoCADu musíme nechat Visual Studio spustit relaci AutoCADu. Pravým tlačítkem klikněte na „Projekt1” v Solution Exploreru a vyberte „Properties”. V dialogu vlastností Projekt1 zvolte ‘Debug’, zaškrtněte ‘Start External Program’ a zvolte cestu k souboru acad.exe. Po této změně nastavení stiskněte klávesu F5 ke spuštění relace AutoCADu.

Použitím příkazu „NETLOAD” načteme obslužnou aplikaci. Napište NETLOAD do příkazového řádku AutoCADu pro otevření dialogu „Choose .NET Assembly”. Zvolte lokaci se souborem „Projekt1.dll” (..\Projekt1\bin\debug), vyberte jej a otevřete.

Napište “AhojSvete” do příkazového řádku. Pokud vše šlo dobře, měl by se objevit text „Ahoj Svete”. Přepněte do Visual studia a vložte breakpoint na řádek: ed.WriteMessage(“Ahoj Svete”). Spusťte příkaz AhojSvete v autocadu znovu a povšimněte si, že můžete krokovat zdrojovým kódem.

Můžete prozkoumat atribut CommandMethod. Povšimněte si, že má sedm různých variant. My jsme použili pouze tu nejjednodušší s pouze jedním parametrem(názvem příkazu).

8.2 Výzva k uživatelskému vstupu

1) V minulém projektu jsme vytvořili instanci třídy „Autodesk.AutoCAD .EditorInput.Editor” pro napsání zprávy do příkazového řádku AutoCADu. V tomto příkladu využijeme tuto třídu k vyzvání uživatele, aby zvolil bod ve výkresu a potom zobrazíme hodnoty souřadnic, které uživatel zvolil. Jako v minulém případu importujeme Autodesk.AutoCAD.ApplicationServices a Autodesk.AutoCAD.EditorInput.

2) Přejmenujeme řetězec v CommandMethod na něco více smysluplného, jako například „ZvolBod“. (Jméno funkce může zůstat stejné). Třída PromptPointOptions předaná metodě editor.GetPoint method. Na začátku funkce vytvoříme instanci objektu používajícího tuto třídu a vložíme řetězec „Vyber bod“. Instance třídy PromptPointResult je vrácena z editor.GetPoint, takže musíme vytvořit instanci také.

```
PromptPointOptions prPointOptions
  = new PromptPointOptions("Vyber bod");
PromptPointResult prPointRes;
```

3) Dále vezmeme objekt editor a použijeme metodu GetPoint předávanou PromptPointOptions objektu. Položíme objekt PromptPointResult rovný návratové hodnotě metody GetPoint. Můžeme otestovat status PromptPointResult a vrátit, pokud není v pořádku.

```
prPointRes = ed.GetPoint(prPointOptions);
if (prPointRes.Status != PromptStatus.OK)
{
  ed.WriteMessage("Error");
}
```

4) Nyní má PromptPointResult validní bod, a tak můžeme vypsat hodnoty do příkazového řádku. Použijeme metodu WriteMessage. Metoda ToString z PromptPointResult.Value to zvládne snadno.

```
ed.WriteMessage("Zvolil jsi bod " +
prPointRes.Value.ToString());
```

5) Stiskněte F5 pro spuštění ladici relace AutoCADu. Napište NETLOAD do příkazového řádku a otevřete požadovaný dll soubor. Do příkazového řádku vložte název příkazu, který jsme vytvořili (Vlozbod). Po dotázání na bod klikněte do výkresu. Pokud je vše v pořádku, vidíte hodnoty souřadnic vámi zvoleného bodu

v příkazovém řádku. V třídě Class.cs vložte breakpoint na řádek „ed.WriteMessage("Error");" Poté spusťte příkaz Vlozbod znovu. Tentokrát stiskněte escape místo zadávání bodu. Status PromptPointResult teď nebude v pořádku, takže je zavoláno "ed.WriteMessage("Error")". Rovněž si můžete všimnou, že hodnota bodu v prPointRes je nastavena na (0,0,0).

8.3 Získání geometrické vzdálenosti

1) Nyní přidáme další příkaz, který získá vzdálenost mezi dvěma body. Vytvoříme nový příkaz v Class.cs pojmenovaný například getdistance pod funkcí k získání bodu. Použijeme atribut CommandMethod a napíšeme string pro příkaz „getdistance". Ve funkci pro příkaz použijeme PromptDistanceOptions namísto of PromptPointOptions. Také výsledek GetDistance je PromptDoubleResult, takže to použijeme místo PromptPointResult:

```
PromptDistanceOptions prDistOptions = new
PromptDistanceOptions("Vyberte prvni bod:");

PromptDoubleResult prDistRes;
prDistRes = ed.GetDistance(prDistOptions);
```

2) Jako v minulém případe otestujeme PromptDoubleResult. Poté použijeme metodu WriteMessage pro zobrazení hodnot v příkazovém řádku

```
if (prDistRes.Status != PromptStatus.OK)
{
  ed.WriteMessage("Error");
}
else
{
  ed.WriteMessage("Vzdalenost je: "
    + prDistRes.Value.ToString());
}
```

8.4 Vykreslení kružnice

V tomto příkladu vytvoříme 'Employee object' (kružnici) uvnitř blokové definice jménem 'EmployeeBlock', která sídlí v 'EmployeeLayer' která má blokovou referenci asociovanou s nim, vloženou v Model Space.

1) Vytvoříme příkaz s názvem ‘CREATE’, která volá funkci CreateEmployee(). Uvnitř této funkce vložíme jednu kružnici do MODELSPACE na pozici 100,100,0 s poloměrem 20.

```
[CommandMethod("CREATE")]
public void CreateCircle()
```

2) Nadeklarujeme objekty, které použijeme. Circle je kružnice, kterou přidáme do ModelSpace. Abychom mohli vložit kružnici, musíme otevřít ModelSpace pomocí BlockTableRecord. Veškerou interakci s databází zapouzdříme v této funkci objektem Transaction.

```
Circle circle;
BlockTableRecord btr;
BlockTable bt;
Transaction trans;
```

3) Vymezíme hranice této interakce členem StartTransaction() z Transaction Manageru a vytvoříme kružnici.

```
trans =
HostApplicationServices.WorkingDatabase.TransactionManager.
StartTransaction();

circle = new Circle(new Point3d(100, 100, 0),
Vector3d.ZAxis, 20);
bt =
(BlockTable)trans.GetObject(HostApplicationServices.Working
Database.BlockTableId, OpenMode.ForRead);

btr =
(BlockTableRecord)trans.GetObject(HostApplicationServices.W
orkingDatabase.CurrentSpaceId, OpenMode.ForWrite);
```

4) Použijeme btr referenci pro přidání naší kružnice a ujistíme se, že transakce o tom ví. Poté transakci spustíme a když je vše hotovo, zrušíme ji.

```
btr.AppendEntity(circle);
trans.AddNewlyCreatedDBObject(circle,true);//
trans.Commit();
trans.Dispose();
```

5) Spustíme funkci, abychom viděli, jak funguje. Měla by vytvořit bílou kružnici s poloměrem 20 na pozici 100, 100, 0.

8.5 Přidávání uživatelských dat

V tomto příkladu se podíváme, čeho je schopna část uživatelské rozhraní .NET API. Začneme uživatelským kontextovým menu. Poté implementujeme modeless, ukotvitelnou paletu podporující Drag and Drop. Pak demonstrujeme vybírání entit z modálního formuláře.

8.5.1 Uživatelské kontextové menu

Doposud všechny naše kódy pouze reagovaly na příkazy definované atributem CommandMethod. Pro vykonání load-time initializace aplikace AutoCAD .NET umí implementovat specifickou třídu, která to dovolí. Abychom to mohli udělat, třída potřebuje implementaci rozhraní IExtensionApplication .NET a předložit atribut assembly-levelu, který specifikuje tuto třídu jako ExtensionApplication. Třída potom může reagovat na one-time načítací a uvolňovací události.

```
[assembly: ExtensionApplica-
tion(typeof(Lab6_CS.AsdkClass1))]
class AsdkClass1 : IExtensionApplication
{
```

Modifikujeme třídu pro implementaci tohoto rozhraní. Modré čáry, které se objevá indikují, že zde jsou nějaké požadované metody k implementaci. Jmenovitě Initialize() a Terminate(). Když implementujeme rozhraní, tato bázová třída je čistě virtuální.

```
public void Initialize()
{
       AddContextMenu();
}
public void Terminate()
{}
```

Pro přidání našeho menu musíme implementovat ‘ContextMenuExtension’ pro použití. Tato třída je členem jmenného prostoru Autodesk.AutoCAD.Windows namespace. K použití ContextMenuExtension potřebujeme inicializovat one with

new, vyplnit nezbytné vlastnosti a konečně zavolat Application.AddDefaultContextMenuExtension(). Způsob, jakým kontextové menu funguje je, že pro každý údaj určujeme specifickou členskou funkci, která bude zavolána při kliknutí. Toto děláme pomocí .NET ‘Delegates’. Používáme C# keywords += and -=, abychom určili, že chceme aby byla událost řízena jednou z našich funkcí.

Přidáme ‘ContextMenuExtension’ členskou proměnnou, a následující dvě funkce pro přidání a odstranění našeho uživatelského kontextového menu.

```
void AddContextMenu()
{
    try
    {
        m_ContextMenu = new ContextMenuExtension();
        m_ContextMenu.Title = "Acme Employee Menu";
        Autodesk.AutoCAD.Windows.MenuItem mi;
        mi = new Autodesk.AutoCAD.Windows.MenuItem("Create
Employee");
        mi.Click +=  new EventHandler(CallbackOnClick);
        m_ContextMenu.MenuItems.Add(mi);
        Auto-
desk.AutoCAD.ApplicationServices.Application.AddDefaultCont
extMenuExtension(m_ContextMenu);
    }
    catch
    {
    }
}

void RemoveContextMenu()
{
    try
    {
       if( m_ContextMenu != null )
       {
          Auto-
desk.AutoCAD.ApplicationServices.Application.RemoveDefaultC
ontextMenuExtension(m_ContextMenu);
       m_ContextMenu = null;
       }
    }
    catch
    {
    }
}
```

Povšimněme si funkce ‘CallbackOnClick’. Tato funkce bude zavolána jako odpověď na výběr položky menu. V našem příkladu je vše, co chceme, aby byla zavolána naše členská funkce ‘Create()’.

```
void CallbackOnClick(object Sender, EventArgs e)
{
    Create();
}
```

Nyní zavoláme funkci AddContextMenu() z Initialize(), a zároveň RemoveContextMenu() z Terminate().

Data AutoCADu (včetlě výkresových databází) jsou uložena v dokumentech, kde příkazy, které přistupují k entitám uvnitř mají práva provádět modifikace. Když spustíme náš kód v odpovědi na kliknutí kontextového menu, přistupujeme k dokumentu zvenčí struktury příkazu. Když kód, který zavoláme zkusí změnit dokument přidáním Employee, spadne to. Abychom to udělali správně, musíme zamknout dokument pro přístup a pro toto používáme metodu Document.LockDocument().

```
void CallbackOnClick(object Sender, EventArgs e)
{
    DocumentLock docLock =    Auto-
desk.AutoCAD.ApplicationServices.Application.DocumentManage
r.MdiActiveDocument.LockDocument();
    Create();
    docLock.Dispose();
}
```

Když teď spustíme kód, máme pracující kontextové menu.

8.5.2 Modeless ukotvitelné paletové okno s Drag and Drop

S .NET Spomůžeme vytvořit jednoduchý formulář a začlenit ho v našich paletách. Můžeme vytvořit instanci uživatelského objektu 'PaletteSet' pro obsažení našeho formuláře a upravení sady palet styly, které preferujeme.

Přidáme nový UserControl do projektu kliknutím pravého tlačítka na projekt v Solution Esploreru a vybereme 'User Control'. Pojmenujeme ho jako 'ModelessForm'. Použijeme 'ToolBox' k přidání 'Edit Boxů' a 'Labelů'.

Použijeme okno ‘Properties’ k nastavení tří edit boxů. Nastavíme parametry::

```
<First, top edit box>
(Name) = tb_Name
Text = <Chose a name>

<Second edit box>
(Name) = tb_Division
Text = Sales

<Third edit box>
(Name) = tb_Salary
Text = <Chose a salary>

<'Drag to Create Employee' Label> (Step 7, Below)
(Name) = DragLabel
Text = 'Drag to Create Employee'
```

Za účelem vytvoření instance objektu palety s .NET API jsou vytvořeny instance uživatelsky ovládaného objektu (náš ModelessForm), a objektu ‘. Člen The PaletteSetu Add je volán předávaním uživatelsky ovládaného objektu a po zavolání.

Dále potřebujeme přidat příkaz pro vytvoření palety. Přidáme funkci do třídy zvanou CreatePalette a CommandMethod(), terý definuje příkaz nazvaný “PALETTE”. Toto je kód, který vytváří instanci palety:

```
ps = new Autodesk.AutoCAD.Windows.PaletteSet("Test Palette
Set");
ps.MinimumSize = new System.Drawing.Size(300, 300);
System.Windows.Forms.UserControl myCtrl = new Modeless-
Form();
//ctrl.Dock = System.Windows.Forms.DockStyle.Fill;
ps.Add("test", myCtrl);
ps.Visible = true;
```

Přidáme kód do metody CreatePalette(). ‘ps’ musí být deklarováno vně definice funkce:

```
private Autodesk.AutoCAD.Windows.PaletteSet ps;
```

Přidáme kód do metody ke zkontrolování, zda je ps null před vytvářením instance palety. Můžeme experimentovat s objektem PaletteSetStyles. Také můžeme experimentovat s nastavením jako je neprůhlednost.

```
ps.Style = PaletteSetStyles.ShowTabForSingle;
ps.Opacity = 90;
```

Poznámka: Budete potřebovat přidat dva jmenné prostory pro objekty PaletteSet and PaletteSetStyles.

Než budeme pokračovat, provedeme rychlou úpravu. Přidáme následující členy do třídy AsdkClass1.

```
public static string sDivisionDefault = "Sales";
public static string sDivisionManager = "Fiona Q. Farnsby";
```

Tyto hodnoty budou použity jako defaultní pro Division a Division Manager. Když jsou deklarovány jako statické, jsou vytvářeny jejich instance jednou za instanci aplikace v assembly-load time.

Podpora Drag and Drop Support v Modeless formuláři:

Když uživatel přetáhne z palety do AutoCAD editoru, je získána pozice a je vytvořena nová instance Employee používající tyto hodnoty.

Nejdřív potřebujeme objekt na přetažení. Přidáme dodatečná 'Label' náležící textovým boxům pojmenovaný **DragLabel** a nastavíme text na něco jako 'Drag to Create Employee'. Z tohoto labelu budeme schopni ovládat drag and drop do editoru AutoCADu. K detekování, kdy událost přetažení proběhne, potřebujeme vědět, kdy probíhají určité operace s myší. Nejdříve potřebujeme zaregistrovat událost následujícím kódem v konstruktoru třídy:

```
DragLabel.MouseMove += new Sys-
tem.Windows.Forms.MouseEventHandler(DragLabel_MouseMove);
```

Přidáme tuto funkci do deklarace třídy ModelessForm:

```
private void DragLabel_MouseMove( object sender, Sys-
tem.Windows.Forms.MouseEventArgs e)
{
    if (System.Windows.Forms.Control.MouseButtons == Sys-
tem.Windows.Forms.MouseButtons.Left)
    {
    }
}
```

Běžně ovladače událostí vezmou dva argumenty. Odesilatele jako objekt a 'event arguments'. Pro pohyb myši musíme udělat to samé.

Nyní víme, kdy probíhá operace pohybu myši. Můžeme jít dál a říct, že levé tlačítko mysi je právě stisknuté.

```
if (System.Windows.Forms.Control.MouseButtons ==
System.Windows.Forms.MouseButtons.Left)
{
}
```

Potřebujeme znát způsob, jak detekovat, kdy je objekt puštěn. K tomu použijeme bázovou třídu .NET s názvem DropTarget. Jednoduše vytvoříme třídu, která dědí od této bázové a implementuje metodu, kterou potřebujeme. V našem případě jde o OnDrop().

Přidáme třídu nazvanou 'MyDropTarget' do projektu, která dědí od 'Autodesk.AutoCAD.Windows.DropTarget'. Když přidáme tuto třídu do souboru ModelessForm.cs file, musíme se ujistit, že jsme přidali třídu za třídu ModelessForm. Uvnitř této nové třídy vložíme handler pro OnDrop ebeny.

```
override public void
OnDrop(System.Windows.Forms.DragEventArgs e)
{}
```

Uvnitř této funkce budeme chtít volat členy AsdkClass1 CreateDivision() a CreateEmployee(), předáním hodnot z tb_xxx edit boxů v ModelessForm class. Abychom to udělali, potřebujeme způsob, jak spojit instanci ModelessForm s touto třídou. Njelepší cesta je přes DragEventArgs. Nejdříve musíme spojit událost myši k třídě MyDropTarget.

Přidáme následující řádek dovnitř MouseButtons.Left klauzule klauzule zpět v handlenu mouse-move. MyDropTarget bude zavolán, když kurzor vstoupí do prohlížecí oblasti AutoCADu.

```
Autodesk.AutoCAD.ApplicationServices.Application.DoDragDrop
(this,  this,System.Windows.Forms.DragDropEffects.All, new
MyDropTarget());
```

Všimněte si, že jsme předávali 'this' dvakrát. Poprvé je to pro argument 'Control' argument, a podruhé pro uživatelsky definovaná data, která jsou předávána. Poté, co předíme instanci třídy ModelessForm, můžeme ji použít k získání hodnot edit boxů v čase pustění(drop-time).

Všimněte si, že jsme vytvořili instanci třídy DropTarget jako poslední argument.To je jak náš MyDropTarget override je začleněn do mechanismu.

Zpět v handleru OnDrop použijeme argument DragEventArgs k získání pozice kurzoru v moment puštění. Pak můžeme toto konvertovat do souřadnic předtím než zavoláme CreateDivision a CreateEmployee.

```
Editor ed =
Autodesk.AutoCAD.ApplicationServices.Application.DocumentMa
nager.MdiActiveDocument.Editor;
try
{
  Point3d pt = ed.PointToWorld(new Point(e.X, e.Y));
  //…
```

Vyprostíme objekt ModelessForm předaný uvnitř argumentu DragEventArgs.

```
ModelessForm ctrl =
(ModelessForm)e.Data.GetData(typeof(ModelessForm));
```

Zavoláme členy AsdkClass1.

```
AsdkClass1.CreateDivision(ctrl.tb_Division.Text,
AsdkClass1.sDivisionManager);
AsdkClass1.CreateEmployee(ctrl.tb_Name.Text,
ctrl.tb_Division.Text, Con-
vert.ToDouble(ctrl.tb_Salary.Text), pt);
```

Volání metody AsdkClass1 bez instance AsdkClass1 vyžaduje, aby funkce byly deklarovány jako 'public static'. Poté co mohou veřejné statické metody volat jiné veřejné statické metody, budeme potřebovat změnit několik deklarací funkcí v AsdkClass1 pro použití 'public static'. Provedeme tyto změny.

Konečně, poté co jsme znovu používali události z vnějšku kontextu příkazů AutoCADu, musíme znovu provést zmaknutí dokumentu kolem kódu, který bude modifikovat databázi. Uzamkneme stejně, jako jsme to udělali pro kontextové menu.

8.5.3 Entita vybraná z modálního formuláře

Zde se zaměříme na tvorbu modálního formuláře a jeho skrývání. K získání detailů employee použijemee pomocnou funkci ListEmployee.Nejdříve potřebujeme vytvořit novou třídu formuláře.

Vytvoříme novou třídu Windows Form v projektu. Zavoláme třídu 'ModalForm'. Přidáme tři Edit boxes s labely a dvě tlačítka. Použijeme okno vlastností k nastavení tří edit boxů. Nastavíme vlastnosti na:

```
<First, top edit box>
(Name) = tb_Name
Text = <Blank Text>

<Second edit box>
(Name) = tb_Division
Text = <Blank Text>

<Third edit box>
(Name) = tb_Salary
Text = <Blank Text>

<First, top button>
(Name)  = SelectEmployeeButton
Text = Select Employee

<Second, bottom button>
(Name)  = Close
Text = Close
```

Dále vytvoříme handleny pro tlačítka. Tlačítko 'Close' jednoduše zavolá:

```
this.Close();
```

K zobrazení dialogu vytvoříme příkazovou metodu v této třídě, která vytvoří instantce formuláře jako modální dialog.

```
[CommandMethod("MODALFORM")]
public void ShowModalForm()
{
 ModalForm modalForm = new ModalForm();
```

```
  Auto-
desk.AutoCAD.ApplicationServices.Application.ShowModalDialo
g(modalForm);
 }
```

Tlačítko ‘Select Employee’ nejdříve vybere jednoduchý výběr entity. Pro toto použijeme metodu Editor.GetEntity().

```
PromptEntityOptions prEnt = new PromptEntityOptions("Select
an Employee");
PromptEntityResult prEntRes = ed.GetEntity(prEnt);
```

Přidáme kód do těla handlenu SelectEmployeeButton_Click s nezbytnými nastaveními proměnných databází, editoru, transakce, Try Catch bloku. Nesmíme zapomenout na Dispose.

Otestujeme návratovou hodnotu GetEntity proti PromptStatus.OK. Pokud se nerovnají, zavoláme toto.Show a opustíme handler.

Jakmile máme výsledek a je OK, můžeme zkusit použít metodu PromptEntityResult.ObjectId() pro získání Id objektu pro vybranou entitu. Toto ID může být předáno do funkce AsdkClass1.ListEmployee, dále s fixovaným polem stringů můžeme získat detaily

```
CPH
string[] saEmployeeList = new string[4];

AsdkClass1.ListEmployee(prEntRes.ObjectId, ref saEmploy-
eeList);
if (saEmployeeList.Count == 4)
{  tb_Name.Text = saEmployeeList[0].ToString();
   tb_Salary.Text = saEmployeeList[1].ToString();
   tb_Division.Text = saEmployeeList[2].ToString();
}
```

Přidáme kód, který vyplňuje edit boxy detaily employee. Předtím než otestujeme kód, musíme si zapamatovat, že běží z modálního dialogu, což znamená, že uživatelská interaktivita je je zablokována, když je dialog viditelný. Než vybereme Employee dol seznamu, potřebujeme skrýt formulář, abychom umožnili výběř. Když je hotovo, můžeme znovu zobrazit formulář.

Přidáme kód pro skrytí před vybíráním (před try block) ‘this.Hide’ a kód pro zobrazení formuláře, když je hotovo(ve Finally bloku) , ‘this.Show’.

Přidání stránky do dialogu vlastností AutoCADu

Přidáme další uživatelské ovládáné nazvané ‘EmployeeOptions’ do projektu. Přidáme dva editovací boxy se jmenovkou. Použijeme okno ‘Properties’ k nastavení tří ‘Edit’ boxů ukázaných. Nastavíme vlastnosti na:

```
<First, top edit box>
(Name)= tb_EmployeeDivision
Text = <Blank Text>

<Second edit box>
(Name) = tb_DivisionManager
Text = <Blank Text>
```

K zobrazení uživatelského tab dialoguv .NET API musíme udělat dva kroky. Prvním je zapsání do oznámení pro kdy je dialog vlastností spustěn předáním adresy členské funkce k zavolání. Druhým krokem je implementace odvolávací funkce. Druhý argument předaný callbacku je objekt ‘TabbedDialogEventArgs’, který musíme použít pro zavolání jeho členu ‘AddTab’. AddTab vezme takes a title string, a instanci objektu ‘TabbedDialogExtension’, který zabalí náš formulář. Uvnitř konstruktoru TabbedDialogExtension předáváme novou instanci našeho formuláře a Voláme zpět adresy, které můžeme předat k řízení jako OnOK, OnCancel or OnHelp.

Uvnitř třídy EmployeeOptions přidáme public static funkci nazvanou AddTabDialog která přidává handler pro sysém k volání:

```
public static void AddTabDialog()
{
  Auto-
desk.AutoCAD.ApplicationServices.Application.DisplayingOpti
onDialog += new TabbedDialogEventHandler(TabHandler);
}
```

Pokračujeme a naimplementujeme obdobnou třídu, která odstraní handler použitím klíčového slova -= .

Přidáme následující kód, abychom implementovali handler:

```
private static void TabHandler(object sender,
Autodesk.AutoCAD.ApplicationServices.TabbedDialogEventArgs
e)
{
  EmployeeOptions EmployeeOptionsPage = new EmployeeOp-
tions();
  e.AddTab("Acme Employee Options",
  new TabbedDialogExtension(
  EmployeeOptionsPage,
  new TabbedDialogAction(EmployeeOptionsPage.OnOk)));
}
```

Zde jsme poprvé vytvořili instanci objektu EmployeeOptions. Pak voláme e.AddTab(), předáváme novou instanci objektu TabbedDialogExtension a TabbedDialogAction specifikující, kde zavolat pro tři kacem které jsem zapsali: Ok, Cancel a Help. V tomto příkladi vybereme pouze OK. Jsou tu dvě další override vorte konstruktoru TabbedDialogAction, které řídí ostatní.

Nyní vše, co zbývá, je specifikovat, co se stane v našem volání, které by mělo být 'OnOK'. Hodláme pouze vyplnit sdílené členy třídy hodnotami přidanými do tb_DivisionManager a tb_EmployeeDivision Edit boxů.

```
public void OnOk()
{
    AsdkClass1.sDivisionDefault = tb_EmployeeDivision.Text;
    AsdkClass1.sDivisionManager = tb_DivisionManager.Text;
}
```

9. Závěrečné zhodnocení

Myslím si, že technologie .NET je velmi perspektovní a má před sebou značnou budoucnost. Její využití se uplatňuje v mnoha oblastech stále více. Pro psaní nástavbových aplikací pro AutoCAD je tomu stejně. Také tam její využití narůstá. Tato práce měla za cíl nastínit princip tvorby právě těchto těchto aplikací. Vzhledem k rozsahu práce bylo možné popsat jen základy, ale i to by mělo být dostatečné, pro značné usnadnění počáteční orientace v dané problematice a její pochopení.

10. Použitá literatura

[1] KENT, Jeff *Visual C# 2005 bez předchozích znalostí*. Brno: Computer Press, 2007

[2] *.Net Framework Training Modules [online]*. Dostupné na WWW: <http://www.devhood.com/training_modules/>

[3] *AutoCAD Commands in C# [online]*.Dostupné na WWW: <http://aec.cadalyst.com/aec/article/articleDetail.jsp?id=170463>

[4] *CADing && Cosiny AutoCAD .NET [online]*.Dostupné na WWW: <http://cadingandcoding.blogspot.com/search/label/AutoCAD.Net>

[5] *New AutoCAD Managed C# Project Application Wizard [online]*.Dostupné na WWW: < http://www.codeproject.com/KB/dotnet/newcswizard.aspx>

[6] *Základy jazyka C# [online]*.Dostupné na WWW: <http://mathonline.fme.vutbr.cz/Zaklady-jazyka-C/sc-100-sr-1-a-109/default.aspx http://www.codeproject.com/KB/dotnet/newcswizard.aspx>

[7] *Základy práce s jazykem C# [online]*.Dostupné na WWW: <http://www.cs.vsb.cz/behalek/vyuka/pcsharp >

[8] *Getting started with AutoCAD and .NET [online]*.Dostupné na WWW: <http://through-the-interface.typepad.com/through_the_interface/2006/07/getting_started.html http://www.cs.vsb.cz/behalek/vyuka/pcsharp>

[9] *DevTV: Introduction to AutoCAD .NET Programming and AutoCAD .NET Labs [online]*.Dostupné na WWW: <http://usa.autodesk.com/adsk/servlet/index?siteID=123112&id=1911627>

[10] *Vývojové prostředky AutoCADu [online]*. Dostupné na WWW: <AutoCADu/System/Hlavni/frmHlavniSet.htm >