

Řešené příklady v C# **aneb C# skutečně prakticky**

Josef Pírk

Obsah

1 Řešené příklady v C# aneb C# skutečně prakticky ..	25
1.1 Rozsah knihy	25
1.2 Verze .NET s příklady	25
1.3 Typografické konvence	26
1.3.1 Překlady termínů	26
1.4 Některá pravidla pro použité zdrojové kódy	27
1.4.1 Pravidla pro užití jmenných prostorů (namespace)	27
1.4.2 Výstupy pro příklady	28
1.4.3 Použití znaků „...“ v kódu	29
1.4.4 Upozornění pro začínající programátory!	29
2 Něco ze základních postupů C#	29
2.1 Přiřazení hodnot proměnným	29
2.1.1 Deklarace proměnné typu int s inicializací	29
2.1.2 Deklarace a přiřazení proměnné typu float s inicializací	29
2.1.3 Deklarace proměnné typu bool s inicializací	30
2.1.4 Přiřazení a změna řetězcové hodnoty	30
2.1.5 Přiřazení hexadecimální hodnoty (šestnáctková soustava)	30
2.1.6 Přiřazení prvku z výčtu	30
2.1.7 Přiřazení prázdné reference	31
2.2 Větvění programu.....	31
2.2.1 if	31
2.2.2 switch	31
2.2.3 goto	32
2.3 Smyčky	32
2.3.1 Příkaz for	32
2.3.2 Příkaz while	32
2.3.3 Příkaz do...while	33
2.3.4 Příkaz foreach	33
2.3.5 continue – přerušení smyčky -> skok na začátek smyčky	34
2.3.6 break – přerušení smyčky -> úplné ukončení smyčky	34
2.4 Volání metod a předávání parametrů	34
2.4.1 Předávání parametrů metodám	35
2.4.2 Vyvolání metody s parametry a vrácení výsledku metody	35
2.4.3 Předání pole jako parametru	35
2.4.4 Vyvolání metody s parametry a vrácení hodnoty přes parametr (out) ..	36
2.4.5 Vyvolání metody s parametry a vrácení hodnoty přes parametr (ref) ..	36
2.4.6 Volání statické metody deklarované v jiném jmenném prostoru	37
3 Přetěžování operátorů	37
3.1 Přetížení operátoru „+“ ve vlastní nové třídě	38
3.1.2 Přetížení operátoru pro implicitní přetypování na typ int	39
3.1.3 Přetížení operátoru pro explicitní přetypování na typ int	39
3.1.4 Přetížení operátorů true a false	40
3.2 Indexery (indexers)	40
3.2.1 Vytvoření jednoduchého indexeru v rámci formuláře	41
4 Práce s instancí objektů	41
4.1 Dynamické vytvoření a zrušení instance objektů.....	41
4.1.1 Vytvoření nové instance objektu Timer	41
4.1.2 Zrušení objektu Timer	42
4.1.3 Použití klíčového slova this	42
4.2 Práce s konstruktory	43
4.2.1 Konstruktor s jedním parametrem	43
4.2.2 Statický konstruktor	43
4.3 Testy na typy objektů.....	44
4.3.1 Test na třídu instance objektu (či na předka objektu)	44
4.3.2 Test na třídu instance objektu a její přetypování na zvolenou třídu	44
4.3.3 Výpis typu instance objektu	45
4.4 Přetypování	45
4.4.1 Explicitní přetypování double na int	45
4.4.2 Přetypování parametru sender na třídu Button	46
4.5 Deklarace vlastností (properties)	46
5 Práce s událostmi	47
5.1.1 Dynamické přidělení události	48
5.1.2 Přidělení více reakcí na jednu událost	48
5.1.3 Odstranění jedné reakce na událost (jsou-li již přiděleny)	49
5.1.4 Ovlivnění výsledku přes událost	49
6 Práce s výjimkami	50
6.1.1 Obsluha obecné (nekonkrétní) výjimky	51

6.1.2	Obsluha konkrétní výjimky	51
6.1.3	Získání bližších informací o výjimce	51
6.1.4	Vyvolání výjimky	52
7	Práce s konzolí – konzolová aplikace.....	53
7.1.1	Výstup textu na konzoli	53
7.1.2	Výstup na konzoli do čtyř řádků přes jeden příkaz	53
7.1.3	Formátovaný výstup na konzoli	54
7.1.4	Určení výstupního proudu (pro XmlTextWriter)	54
7.1.5	Vymazání konzole zvolenou barvou (.NET 2.0)	54
7.1.6	Ukončení přes <Ctrl+C> – odchytení klávesy na konzoli (.NET 2.0) ..	55
8	Práce s komponentami z palety nástrojů	55
8.1	Společné pro všechny vizuální komponenty	56
8.1.1	Změna pozice prvku přes Control.Left a Control.Top	56
8.1.2	Změna pozice prvku přes Control.Location	56
8.1.3	Změna délky prvku přes Control.Width	57
8.1.4	Změna délky prvku přes Control.Size	57
8.1.5	Změna barvy písma a změna barvy prvku pozadí prvku	57
8.1.6	Změna barvy rodičovského objektu	57
8.1.7	Skrytí objektu na formuláři	57
8.1.8	Zákaz práce s objektem – objekt je viditelný, zákaz je graficky rozlišen ..	58
8.1.9	Reakce na vstup do prvku (podporuje-li ji, např. textové pole)	58
8.1.10	Reakce na výstup z prvku (podporuje-li ji, např. textové pole)	58
8.1.11	Reakce na získání zaostření prvku (fokusu)	59
8.1.12	Reakce na ztrátu zaostření prvku (fokusu)	59
8.1.13	Získání velikosti prvku jako typ Rectangle	59
8.1.14	Control.SetStyle a přepsání notificační metody Control.OnNotifyMessage()	59
8.1.15	Změna kurzoru	60
8.1.16	Načtení kurzoru ze souboru	60
8.2	Button	60
8.2.1	Změna textu na tlačítku	60
8.2.2	Změna textu i obrázku na tlačítku s definicí jeho zarovnání	61
8.2.3	Změna barvy tlačítka po najetí myší a změna na Flat styl (.NET 2.0) ..	61
8.2.4	Zkrácení dlouhého textu tlačítka a doplnění znaky “...” (.NET 2.0)	61
8.3	ComboBox	61
8.3.1	Naplnění hodnot do ComboBoxu s aktivací první hodnoty	61
8.3.2	Přidání položek do ComboBoxu a jejich setřídění dle abecedy	62
8.3.3	Vyhledání hodnoty v ComboBoxu	62
8.3.4	Nastavení typu ComboBoxu pouze pro výběr hodnot (zákaz přepisu) ..	62
8.4	DataGrid	63
8.4.1	Dynamické přidání tří položek do DataGridu	63
8.4.2	Dynamické přidání položek do DataGridu, zákaz editace druhého sloupce	63
8.4.3	Přidání řádků tabulky pro DataGrid s podrobnějším nastavením	64
8.4.4	Dynamické vytvoření vlastního sloupce v DataGridu	64
8.4.5	Nastavení celého DataGridu pouze pro čtení	65
8.4.6	Informace o aktivním řádku a sloupci v DataGridu	65
8.5	CheckBox.....	65
8.5.1	Nastavení textu a zaškrtnutí CheckBoxu	66
8.5.2	Povolení a nastavení třetího stavu CheckBoxu	66
8.6	CheckBoxList	66
8.6.1	Přidání tří položek, první dvě jako zaškrtnuté	66
8.6.2	Procházení a výpis zaškrtnutých položek	66
8.6.3	Procházení a výpis indexů zaškrtnutých položek	67
8.7	Label.....	67
8.7.1	Změna textu labelu	67
8.7.2	Zarovnání textu do středu objektu	67
8.8	LinkLabel	67
8.8.1	Zobrazení www stránky po kliknutí na LinkLabel	67
8.9	ListBox	68
8.9.1	Přidání tří položek do ListBoxu	68
8.9.2	Přidání položek do ListBoxu z pole	68
8.9.3	Přidání položek z pole přes vlastnost ListBox.DataSource	68
8.9.4	Přidání položek promítnutím ze sloupce tabulky	69
8.9.5	Smazání dosavadních položek, přidání položek a označení položky	69
8.9.6	Odstranění druhé položky	69
8.9.7	Zaostření položky dle hodnoty	70
8.9.8	Zaostření položky dle hodnoty – v přesném tvaru	70
8.9.9	Zobrazení zvolené položky po poklepání myší	70
8.9.10	Použití znaku tabulátoru pro zarovnání položek	70
8.10	ListView	71

8.10.1 Přidání položek do ListView	71
8.10.2 Přidání položek do ListView a označení druhé z nich	71
8.10.3 Dynamické přidání dvou sloupců a výpis do sloupců	72
8.10.4 Reakce na výběr prvku	72
8.10.5 Označení celého řádku v ListView	73
8.10.6 Výpis zaškrtnutých položek	73
8.10.7 Přepnutí na výpis jako „velké ikony“	73
8.10.8 Přepnutí na výpis jako „malé ikony“	73
8.10.9 Přepnutí na výpis jako „seznam“	73
8.11 MonthCalendar	74
8.11.1 Označení 15.1.2005 (skok do měsíce ledna/2005)	74
8.11.2 Označení 15.1.2005 symbolem dnešního dne	74
8.11.3 Označení více dnů v intervalu 3.1.2005 – 7.1.2005	74
8.11.4 Zobrazení dvou měsíců v řádcích i sloupcích	74
8.11.5 Zvýraznění tří dnů tučným písmem	75
8.11.6 Zjištění zvoleného data	75
8.12 Panel.....	75
8.12.1 Nastavení 3D typu okraje	75
8.12.2 Rolující panel s dynamicky vytvořeným tlačítkem	75
8.13 PictureBox.....	76
8.13.1 Načtení obrázku ze souboru a jeho přizpůsobení na velikost PictureBox	76
8.14 RichEditControl.....	76
8.14.1 Výstup nezformátovaného textu	76
8.14.2 Postupné přidávání různě formátovaných slov do textu	77
8.14.3 Výstup RTF textu – použití bold a italic fontu na část textu	77
8.14.4 Uložení RTF textu do souboru	77
8.14.5 Načtení RTF textu ze souboru	78
8.14.6 Povolení a zpracování poklepu na odkaz	78
8.14.7 Vyhledání hodnoty	78
8.15 RadioButton	78
8.15.1 Označení přepínače programově	78
8.16 StatusBar	79
8.16.1 Úprava textu v prvním oddílu	79
8.16.2 Dynamické přidání tří nových oddílů	79
8.17 TabControl	80
8.17.1 Dynamické přidání tří záložek	80
8.17.2 Nastavení aktivní záložky	80
8.18 TextBox	80
8.18.1 Nastavení inicializačního textu a vstup do prvku	80
8.18.2 Přidání více řádků do víceřádkového (multiline) textboxu	80
8.18.3 Přidání obou rolujících lišt do víceřádkového textboxu	81
8.18.4 Rozdělení řádků TextBox pomocí objektu podpory regulárních výrazů (Regex)	81
8.18.5 Vypsání označeného textu	82
8.18.6 Označení části řetězce v TextBoxu	82
8.18.7 Označení celého textu a jeho vložení do schránky s jeho smazáním ...	82
8.18.8 Vyvolání validace editačního pole	83
8.19 Toolbar	83
8.19.1 Reakce na stisk tlačítka na toolbaru	83
8.20 TrackBar	83
8.20.1 Inicializace prvku	84
8.20.2 Obsluha změny hodnoty TrackBar.Value	84
8.21 TreeView	84
8.21.1 Naplnění stromu	84
8.21.2 Zobrazení názvu aktivní položky stromu po kliknutí	85
8.21.3 Obsluha editace textu položky stromu	85
8.21.4 Při označení/odznačení rodičovského uzlu stromu, označ/odznač přímé potomky	85
8.22 WebBrowser (.NET 2.0)	86
8.22.1 Navigace na specifickou stránku	86
8.22.2 Navigace na domovskou stránku	86
8.22.3 Krok zpět (o stránku dozadu)	86
8.22.4 Krok dopředu (o stránku dopředu)	86
8.22.5 Získání zdrojového kódu webové stránky	87
8.22.6 Uložení klientské části obrazovky webové stránky do souboru .bmp	87
9 Práce s nevizuálními komponentami	87
9.1 ErrorProvider	87
9.1.1 Vizualní zvýraznění vzniklé chyby	87
9.2 NotifyIcon.....	88
9.2.1 Jednoduchý příklad NotifyIcon	88
9.3 PerformanceCounter	89

9.3.1 Dotaz na volnou paměť v MB	89
9.4 Process	89
9.4.1 Pokus o spuštění procesu notepad.exe	89
9.4.2 Spuštění notepad.exe a čekání na vstup	90
9.4.3 Ukončení spuštěné aplikace pomocí existujícího procesu	90
9.5 ServiceController	90
9.5.1 Získání seznamu existujících služeb	90
9.5.2 Zjištění stavu služby, její zastavení a spuštění	91
9.5.3 Výpis závislých služeb	91
9.6 Timer	92
9.6.1 Zobrazení času na formuláři	92
9.6.2 Dynamické vytvoření Timeru na formuláři	92
9.7 ToolTip	92
9.7.1 Nastavení víceřádkového tooltipu	92
10 Práce s formuláři	93
10.1.1 Metoda volaná bezprostředně před zobrazením formuláře	93
10.1.2 Nastavení tzv. Cancel tlačítka pro formulář	93
10.1.3 Nastavení tzv. Accept tlačítka pro formulář	93
10.1.4 Získání identifikačního čísla okna	94
10.1.5 Výpis jména aktivního formuláře	94
10.1.6 Zjištění aktivního prvku formuláře	94
10.1.7 Nastavení formuláře tak, aby byl vždy „navrchu“	95
10.1.8 Maximalizace aktivního formuláře	95
10.1.9 Minimalizace aktivního formuláře do lišty	95
10.1.10 Skrytí formuláře po minimalizaci	95
10.1.11 Zakázání uzavření formuláře	95
10.1.12 Procházení všech prvků na daném rodiči (formuláři)	96
10.1.13 Procházení všech prvků typu Button na daném rodiči (formuláři)	96
10.1.14 Dynamické vytvoření prvku na formuláři	97
10.1.15 Postupné zobrazení formuláře pomocí změny jeho průhlednosti	97
10.1.16 Definování nového tvaru formuláře přes Form.OnPaint	98
10.2 Vytváření modálních a nemedálních formulářů	99
10.2.1 Vytváření modálního formuláře	99
10.2.2 Výpis hodnoty textového pole z vyvolaného modálního formuláře po jeho ukončení	99
10.2.3 Zobrazení dialogu jako nemedálního okna	99
10.3 MDI aplikace/MDI dětská okna	100
10.3.1 Vytvoření MDI dětského okna	100
10.3.2 Seznam MDI dětských oken do položky menu	101
10.3.3 Přístup k aktivnímu dětskému oknu	101
10.3.4 Zjištění MDI dětských oken formuláře	102
10.3.5 Zaostření druhého dětského okna v seznamu	102
10.3.6 Seřazení dětských oken do kaskády	102
10.3.7 Seřazení dětských oken do horizontálních dlaždic	102
10.3.8 Seřazení dětských oken do vertikálních dlaždic	103
10.3.9 Seřazení minimalizovaných dětských oken	103
11 Předdefinované dialogy	104
11.1 Standardní dialogy	104
11.1.1 Výpis textu do dialogu	104
11.1.2 Výpis textu (v dialogu) na více řádek	104
11.1.3 Dotazový dialog Ano/Ne	104
11.2 Speciální dialogy	105
11.2.1 Dialog pro výběr barvy	105
11.2.2 Dialog pro výběr písma	105
11.2.3 Dialog pro výběr souboru	106
11.2.4 Nastavení vícenásobného filtru pro výběr souborů	107
11.2.5 Dialog pro výběr adresáře	107
12 Drag & Drop (“táhni a pusť”)	108
12.1.1 Tažení textu z prvku TextBoxu do ListBoxu	108
12.1.2 Tažení souborů/zástupců do formuláře	109
13 Práce s klávesnicí	109
13.1.1 Reakce na stisknutou klávesu	110
13.1.2 Testování stisku klávesy <Shift> + znaku	110
13.1.3 Testování stisku klávesy <Shift> + <Alt> + znaku za pomoci KeyEventArgs	110
13.1.4 Testování stisku klávesy <Shift> + <Alt> + znaku	110
13.1.5 Práce s klávesami v konzolové aplikaci	111
14 Práce s menu.....	112
14.1.1 Dynamické vytvoření hlavního menu formuláře a jeho položek	112
14.1.2 Programové vyvolání kontextového menu	113
14.1.3 Změna údajů u první položky kontextového menu	113

14.1.4	Reakce položky menu na stisk – prohození příznaku zaškrnutí	113
14.1.5	Zvýraznění určité položky menu	113
14.1.6	Vyvolání akce po „najetí“ na položku menu myší nebo klávesou	114
14.1.7	Připojení kontextového menu k prvku ikony v liště	114
14.2	ContextMenuStrip (.NET 2.0)	114
14.2.1	Vytvoření menu, přidání obsluhy a zobrazení nad volajícím tlačítkem	114
15	Práce se schránkou (clipboard)	115
15.1.1	Kopírování textu z TextBoxu do schránky	115
15.1.2	Kopírování textu ze schránky do TextBoxu	115
15.1.3	Test na přítomnost textového formátu ve schránce	115
15.1.4	Kopie instance objektu do schránky/ze schránky	116
16	Pole a kolekce objektů	117
16.1	Pole (array)	117
16.1.1	Vytvoření a inicializace pole o třech prvcích	117
16.1.2	Vytvoření a inicializace pole Color[]	117
16.1.3	Vytvoření a inicializace pole typu Point[]	117
16.1.4	Dynamické vytvoření pole (přes Array.CreateInstance())	118
16.1.5	Vícerozměrné pole a vrácení počtu rozměrů pole	118
16.1.6	Inicializace prvku pole na základě volané metody	118
16.1.7	Vymazání pole	119
16.1.8	Kopie prvků z pole do pole	119
16.1.9	Klonování pole – kopie pole do nově vytvořeného pole	119
16.1.10	Vzestupné a sestupné seřídění pole	120
16.1.11	Prohledávání pole	120
16.2	Kolekce prvků (ArrayList)	121
16.2.1	Vytvoření kolekce prvků	121
16.2.2	Hledání v kolekci prvků	121
16.2.3	Binární vyhledávání	122
16.3	Fronta (queue)	123
16.3.1	Přidání prvků do fronty a výběr z fronty	123
16.4	Zásobník (stack)	124
16.4.1	Založení zásobníku, výběr hodnot a procházení prvků	124
16.5	Hašovací tabulka	124
16.5.1	Ukázka práce s hašovací tabulkou	125
16.5.2	Hašovací kód (hashCode)	125
16.5.3	Přepsání GetHashCode()	125
16.6	SortedList (.NET 2.0)	126
16.6.1	Založení seznamu, vložení hodnot a vyhledání hodnoty dle klíče	126
17	Práce s datovým typem řetězec, datum, výčet a struktura	127
17.1	Práce s řetězci	127
17.1.1	Přiřazení prázdného řetězce	127
17.1.2	Inicializace řetězce z pole znaků char[]	127
17.1.3	Převod řetězce do pole char[]	128
17.1.4	Spojení prvků pole do řetězce a jejich oddělení specifickým řetězcem	128
17.1.5	Vytvoření daného počtu stejných znaků	128
17.1.6	Zjištění délky řetězce	128
17.1.7	Procházení řetězce po jednotlivých znacích	128
17.1.8	Vrácení části řetězce	129
17.1.9	Vyhledání pozice prvního výskytu podřetězce v řetězci	129
17.1.10	Vyhledání pozice posledního výskytu podřetězce v řetězci	129
17.1.11	Vyhledání slova od počátku do výskytu jednoho ze seznamu znaků	129
17.1.12	Použití String.Format()	130
17.1.13	Další formátování výstupu přes String.Format()	130
17.1.14	Rozdělení řetězce dle oddělovačů	131
17.1.15	Převod logické hodnoty na řetězec	131
17.1.16	Test na shodnost řetězců – velikost písmen rozhoduje	132
17.1.17	Test na shodnost řetězců – velikost písmen nerozhoduje	132
17.1.18	Převod řetězce na malá písmena	132
17.1.19	Převod řetězce na velká písmena	132
17.1.20	Náhrada části řetězce za jiný	133
17.1.21	Odstranění mezer z počátku a konce řetězce	133
17.1.22	Odstranění specifických znaků z počátku a konce řetězce	133
17.1.23	Test na znaky na konci řetězce	133
17.1.24	Odstranění posledního znaku z řetězce	134
17.1.25	Dotaz, zda je první znak řetězce písmeno	134
17.1.26	Dotaz, zda je první znak řetězce číslo	134
17.1.27	Dotaz, zda je první znak řetězce číslo nebo písmeno	134
17.1.28	Dotaz, zda je první znak řetězce malé písmeno	134
17.1.29	Dotaz, zda je první znak řetězce velké písmeno	134
17.1.30	Výpis uvozovek v řetězci	135

17.2 StringBuilder	135
17.2.1 Vytvoření a přidání textu do StringBuilder	135
17.3 Práce s datem a časem	135
17.3.1 Zjištění aktuálního datumu a času	135
17.3.2 Vytvoření datumu z jednotlivých prvků data	136
17.3.3 Test, zda současný rok je přestupný	136
17.3.4 Vrácení jména aktuální časové zóny (pro běžný čas)	136
17.3.5 Vrácení jména aktuální časové zóny (pro letní čas)	136
17.3.6 Zjištění, zda je v předaném datu aktuální zóny užíván letní čas	136
17.3.7 Rozklad aktuálního času na hodinu, minutu a vteřinu	137
17.3.8 Přidání daného počtu dní k datumu	137
17.3.9 Získání datumu z řetězce	137
17.3.10 Zformátování datumu dle masky	137
17.3.11 Připravené metody pro formátování data a času	138
17.3.12 Pokus o převod řetězce do datumu v očekávaných formátech	139
17.4 TimeSpan	139
17.4.1 Práce s rozdílem dvou datumu	139
17.4.2 Získání intervalu od začátku do konce volání	140
17.5 Práce s výčtovým typem	140
17.5.1 Deklarace výčtu a proměnné typu výčet	140
17.5.2 Použití OR operátoru pro nastavení více hodnot výčtu	140
17.5.3 Nastavení bitových hodnot do proměnné a jejich testování	141
17.5.4 Odnastavení bitové hodnoty	141
17.5.5 Procházení prvků výčtu	141
17.6 Práce se strukturami	142
17.6.1 Deklarace struktury a proměnné struktury	142
18 Převody a zaokrouhlování	143
18.1 Převody	143
18.1.1 Převod řetězce na číslo	143
18.1.2 Převod z řetězce na boolean	143
18.1.3 Převod řetězce na double	143
18.1.4 Převod hodnoty do dvojkové, osmičkové a šestnáctkové soustavy ..	143
18.1.5 Ošetření chybného převodu	144
18.1.6 Binární posun – posunutí hodnoty o dva bity doleva	144
18.2 Převody souřadnic	145
18.2.1 Převod souřadnice v rámci prvku na souřadnice obrazovky	145
18.2.2 Převod souřadnic obrazovky na souřadnice v rámci prvku	145
18.3 Zaokrouhlování	145
18.3.1 Zaokrouhlení na daný počet desetinných míst	145
18.3.2 Zaokrouhlení na nejbližší nižší celé číslo	146
18.3.3 Zaokrouhlení na nejbližší vyšší celé číslo	146
18.3.4 Nastavení hodnoty „reálné kladné nekonečno“ do typu double	146
19 Kódování	146
19.1.1 Převod znaku na ASCII číselno hodnotu	146
19.1.2 Převod číselné ASCII hodnoty na znak	147
19.1.3 Převod UNICODE na kódování ASCII (a převod řetězce na byte[]) ..	147
19.1.4 Vytvoření řetězce z pole byte[] pomocí UNICODE kódování	147
20 Práce s databází	147
20.1 Datové spojení (OleDb)	148
20.1.1 Otevření spojení	148
20.1.2 Reakce na změnu stavu spojení – událost	149
OleDbConnection.StateChange	149
20.1.3 Test na stav spojení	149
20.1.4 Výpis použitého poskytovatele z OleDbConnection.ConnectionString ...	149
20.1.5 Seznam všech tabulek pod spojením	150
20.1.6 Seznam uživatelských tabulek pod spojením	150
20.1.7 Seznam všech tabulek pod spojením s upřesněním typu objektu	150
20.1.8 Výpis seznamu všech sloupců nad všemi tabulkami pod spojením ..	150
20.1.9 Naplnění seznamu existujících tabulek pod spojením	151
20.2 Datový adaptér	151
20.2.1 Otevření adaptéru a naplnění do datové množiny	151
20.3 Datová množina (DataSet)	151
20.3.1 Naplnění datové množiny z adaptéru a napojení do DataGridu	151
20.3.2 Naplnění datové množiny z adaptéru a napojení do DataGridu	
– varianta s pojmenováním tabulky	152
20.3.3 Vytvoření XML souboru z dat datové množiny	152
20.3.4 Načtení prvních tří řádek do datové množiny a vrácení jako XML ..	152
20.4 Datová tabulka (DataTable)	153
20.4.1 Odkaz na tabulku z datové množiny	153
20.4.2 Výpis počtu řádků tabulky	153
20.4.3 Procházení řádků tabulky	153

20.4.4 Změna nadpisu pro DataGrid	154
20.5 Datový příkaz – dotaz (OleDbCommand)	154
20.5.1 Vrácení jedné hodnoty z dotazu	154
20.5.2 Procházení výsledku dotazu (procházení vrácených řádků dotazu) ..	154
20.5.3 Dotaz s parametry	155
20.5.4 Změna v datech přes datový příkaz	155
20.6 Datové relace (DataRelation)	155
20.6.1 Relace mezi tabulkami a výstup do DataGridu	156
20.7 Datové sloupce	157
20.7.1 Vytvoření kalkulačního sloupce s pevně nastavenou hodnotou	157
20.7.2 Vytvoření kalkulačního sloupce na základě funkce	157
20.7.3 Vytvoření kalkulačního pole s řetězcovou funkcí	157
20.8 Editace dat (update, insert, delete)	158
20.8.1 Procházení řádků tabulky, které byly změněny (zde přes DataGrid) ..	158
20.8.2 Dogenerování UPDATE, INSERT, DELETE příkazu	158
20.8.3 Uložení dat	159
20.8.4 Přidání nového záznamu do tabulky ZAMESTNANCI	159
20.8.5 Smazání řádků tabulky ZAMESTNANCI s ID_ZAMESTNANEC větším než 3	160
20.8.6 Procházení řádků a změna hodnot kalkulačního sloupce	160
20.9 Navázání dat (databinding)	161
20.9.1 Vytvoření nové databinding vazby – navázání hodnoty pole do TextBox.Text	161
20.9.2 Získání managera pro obsluhu vazeb	161
20.9.3 Posun na předchozí větu	162
20.9.4 Posun na následující větu	162
20.9.5 Reakce na změnu pozice Binding managera – výpis aktuálního řádku do Label	162
20.9.6 Vytvoření vazby s možností formátování výstupu	163
20.9.7 Vytvoření vazby na pole	163
20.10 Práce s daty v .NET 2.0	164
20.10.1 Napojení dat na BindingSource	164
20.10.2 Zákaz přidávání řádku – nastavení přes BindingSource	165
20.10.3 Použití BindingNavigator	165
20.10.4 Zaostrění specifické buňky tabulky DataGridView	166
20.10.5 Označení a průchod řádků DataGridView (označení celých řádků) .	166
20.10.6 Označení a průchod buněk DataGridView	166
20.10.7 Přidání tlačítka na místo druhého sloupce v DataGridView	167
20.10.8 Reakce na stisk tlačítka v DataGridView.....	167
20.10.9 Přidání fixovaného ComboBoxu mezi sloupce DataGridView	167
21 Regulární výrazy	168
21.1 Důležité zástupné znaky pro vyhodnocování regulárních výrazů	168
21.2 Testy na platnost výrazů (odpovídá masce?)	169
21.2.1 Hledání náhradou za znak	169
21.2.2 Test dle prvního znaku	170
21.2.3 Hledání tečky	170
21.2.4 Povolení více přípustných znaků	170
21.3 Rozklady regulárních výrazů do výskytů (Match)	170
21.3.1 Rozklad na jednotlivé znaky, které nejsou mezerou	170
21.3.2 Rozklad na jednotlivé znaky/slova, která nejsou mezerou	171
21.3.3 Vrácení znaků řetězce, za kterými je mezera	171
21.3.4 Rozklad na jednotlivé znaky/slova, za kterými je mezera	171
21.3.5 Separace hodin z řetězce	172
21.3.6 Separace datumu z řetězce	172
21.4 Rozklad regulárních výrazů s využitím skupin (Group)	172
21.4.1 Rozklad datumu a času s využitím skupin	173
22 Funkce	173
22.1 Matematické funkce	173
22.1.1 Vrácení absolutní hodnoty	173
22.1.2 Použití konstanty PI	173
22.1.3 Vrácení minimální hodnoty ze dvou čísel	174
22.1.4 Vrácení minimální hodnoty ze tří čísel	174
22.1.5 Vrácení odmocniny z hodnoty	174
22.1.6 Vrácení hodnoty sinu úhlu	174
22.1.7 Celočíselné dělení a získání zbytku po celočíselném dělení	174
22.2 Náhodná čísla	175
22.2.1 Generování náhodného čísla v daném rozsahu	175
22.2.2 Generování náhodného čísla mezi 0.0 – 1.0	175
23 Práce s grafikou	175
23.1 Fonty	176

23.1.1	Tvorba instance nového fontu Tahoma	176
23.1.2	Tvorba instance nového implicitního fontu	176
23.1.3	Změna fontu objektu	176
23.1.4	Vrácení výšky fontu	176
23.1.5	Procházení jmen instalovaných fontů (.NET 2.0)	177
23.2	Barvy	177
23.2.1	Vyhledání barvy dle jména	177
23.2.2	Přístup k systémovým barvám	177
23.2.3	Vytvoření nové barvy z palety a obarvení formuláře	178
23.3	Pera (pens)	178
23.3.1	Vytvoření červeného pera	178
23.3.2	Využití předdefinovaného pera	178
23.3.3	Vytvoření červeného přerušovaného pera o šířce 5 bodů	178
23.3.4	Vytvoření červeného pera o šířce 5 bodů s definovaným tvarem zakončení	179
23.4	Štětce (brush)	179
23.4.1	Vytvoření štětce plného vzoru	179
23.4.2	Využití předdefinovaného štětce	179
23.4.3	Vytvoření štětce se vzorkem	179
23.4.4	Vytvoření a použití gradientního štětce	180
23.4.5	Vytvoření štětce dle dat z bitmapy	180
23.5	Kresba čar	180
23.5.1	Vykreslení čáry	180
23.5.2	Vykreslení více čar jedním příkazem (dle pole Point[])	181
23.5.3	Vykreslení čáry o šířce 5 bodů se šipkou na konci	181
23.6	Kresby základních geometrických útvarů	182
23.6.1	Kresba rámečku	182
23.6.2	Kresba rámečku vyplněného červenou barvou	182
23.6.3	Kresba kružnice	182
23.6.4	Kresba elipsy	182
23.6.5	Kresba vyplněné elipsy	182
23.6.6	Kresba kruhové výseče	182
23.6.7	Kresba kruhové výseče vyplněné červenou barvou	183
23.7	Kresba „grafického“ textu	183
23.7.1	Jednorázové vykreslení grafického textu	183
23.7.2	Jednorázové vykreslení grafického „pootočeného“ textu	183
23.7.3	Formátování výstupu grafického textu – centrování do středu oblasti ..	184
23.7.4	Vykreslení vertikálního grafického textu	184
23.7.5	Vykreslení textu do rámečku, změnění velikosti grafického řetězce ..	184
23.8	Obrázky	185
23.8.1	Načtení bitmapy ze souboru	185
23.8.2	Vykreslení bitmapy na formulář	185
23.8.3	Dynamické kreslení do bitmapy	185
23.8.4	Dynamické kreslení do bitmapy – další varianta	186
23.8.5	Jednorázové vykreslení obrázku ve skutečné velikosti	186
23.8.6	Jednorázové vykreslení obrázku 10x zmenšeného	186
23.8.7	Otočení obrázku	187
23.8.8	Klonování obrázku – vytvoření nového obrázku z části jiného	187
23.8.9	Šikmé vykreslení obrázku	187
23.8.10	Načtení obrázku přes pole byte[], vytvoření datového proudu MemoryStream (.NET 2.0)	188
23.9	Ostatní kolem grafiky... ..	188
23.9.1	Přepsání metody Form.OnPaint()	188
23.9.2	Překreslení barvy celého objektu	189
23.9.3	Vykreslení obrazce s gradientním vzorem	189
23.9.4	Zadání souřadnic pro grafiku absolutní hodnotou	190
23.9.5	Sjednocení a průnik dvou oblastí	190
24	Práce s adresáři	190
24.1.1	Založení adresáře	190
24.1.2	Kontrola existence adresáře a případné založení nového	190
24.1.3	Naplnění seznamu logicky připojených diskových jednotek	191
24.1.4	Informace o adresáři	191
24.1.5	Přesun adresáře	191
24.1.6	Smazání adresáře i s podadresáři	191
24.1.7	Vrácení aktivního adresáře	191
24.1.8	Změna aktivního adresáře	192
24.1.9	Informace o změně v adresáři	192
24.1.10	Informace o umístění speciálních adresářů	192
24.1.11	Vypsání cesty k systémovému (system32) adresáři	193
24.1.12	Vypsání cesty pro ukládání aplikačních dat – pro všechny uživatele	193
24.1.13	Vypsání cesty pro ukládání aplikačních dat – pro konkrétního uživatele ..	193
25	Práce se soubory.....	193

25.1 Obecná práce se soubory	193
25.1.1 Informace o souboru	193
25.1.2 Testování atributů souboru	193
25.1.3 Kontrola existence souboru	194
25.1.4 Kopírování souboru	194
25.1.5 Kopírování souboru – vždy přepiš existující	194
25.1.6 Přesun souboru	195
25.1.7 Smazání souboru	195
25.1.8 Výpis data posledního zápisu do souboru	195
25.1.9 Načtení a zobrazení obsahu adresáře (disku) do stromu TreeView ..	196
25.1.10 Procházení určitých typů souborů (zde dle koncovky)	196
25.1.11 Vrácení jména konfiguračního souboru	197
25.2 Práce s cestami	197
25.2.1 Výpis celé cesty ke spuštěnému souboru	197
25.2.2 Extrakce a výpis cesty bez jména souboru	197
25.2.3 Extrakce a výpis jména souboru	197
25.2.4 Extrakce a výpis koncovky souboru	197
25.2.5 Extrakce a výpis root adresáře	198
25.2.6 Záměna koncovky u jména souboru	198
26 Práce s daty v souborech	198
26.1 Čtení/zápis textových souborů	198
26.1.1 Vytvoření StreamReader z FileInfo	198
26.1.2 Vytvoření StreamWriter z FileInfo	198
26.1.3 Načtení textu ze souboru přes StreamReader	199
26.1.4 Zápis textu do souboru přes StreamWriter	199
26.1.5 Kopie textového souboru přes FileStream	199
26.1.6 Kopie textového souboru přes FileStream s využitím bufferu	200
26.1.7 Kopie textového souboru přes StreamReader/StreamWriter	200
26.1.8 Zápis a čtení do souboru s kódováním ASCII	201
26.1.9 Načtení celého souboru do řetězce (.NET 2.0)	201
26.2 Čtení/zápis binárních souborů	201
26.2.1 Zápis pevné délky pole bytů do souboru přes FileStream	201
26.2.2 Zápis proměnné délky pole bytů do souboru přes FileStream	202
26.2.3 Binární zápis řetězce do souboru přes BinaryWriter	202
26.2.4 Zápis a čtení jednotlivých typů do/z binárního souboru	203
26.2.5 Binární kopie dvou souborů přes BinaryReader/BinaryWriter	204
26.3 Asynchronní práce se soubory	204
26.3.1 Asynchronní zápis textu do souboru	204
26.3.2 Asynchronní zápis textu do souboru s čekáním na ukončení zápisu ..	205
27 Práce s XML	206
27.1.1 Čtení prvků ze XML souboru	206
27.1.2 Naplnění ComboBoxu z dat XML souboru	207
27.2 Práce s XML přes XmlTextWriter/XmlTextReader	207
27.2.1 Vytvoření XML souboru přes XmlTextWriter	208
27.2.2 Načtení XML dokumentu a kopie do nového XML souboru	209
27.2.3 Načtení a procházení XML souboru přes XmlTextReader	209
28 Serializace dat	210
28.1.1 Serializace celé instance objektu do souboru	210
28.1.2 Další způsob serializace – tentokrát přes XmlSerializer	211
29 Práce s registry	213
29.1.1 Načtení záznamu z prvku registru	213
29.1.2 Načtení hodnoty z prvku registru – neexistuje-li větev, založí novou	213
29.1.3 Založení nové větve a zápis dvou záznamů do registru	214
29.1.4 Načtení názvů nižších podklíčů registru	214
30 Tisk	214
30.1 PrintDocument	214
30.1.1 Tisk tří řádků na tiskárnu	214
30.1.2 Tisk prvku z formuláře na tiskárnu	215
30.2 PrintPreviewDialog	215
30.2.1 Zobrazení tiskového náhledu.	215
30.3 PrintPreviewControl	216
30.3.1 Dynamické vytvoření náhledu	216
30.4 PrintDialog	217
30.4.1 Zobrazení tiskového dialogu	217
30.5 PageSetupDialog	217
30.5.1 Změna nastavení tiskárny	217
31 Práce se sítí a v síti	218
31.1.1 Informace o síti	218
31.1.2 Informace o síti přes objekt Environment	219
31.1.3 Získání jména stanice a čísla IP adresy stanice	219

31.1.4 Získání jména stanice z IP adresy	219
31.1.5 Získání informací o adrese ze jména domény	219
31.2 Internet	220
31.2.1 Zobrazení www stránky v implicitním prohlížeči	220
31.2.2 Stažení kódu stránky z internetu	220
31.2.3 Stažení obrázku z internetu a jeho vykreslení	220
31.2.4 Stažení obrázku z internetu přes PictureBox (.NET 2.0)	221
31.3 Pošta	221
31.3.1 Jednoduché odeslání poštovní zprávy	222
31.3.2 Odeslání poštovní zprávy s podrobnějším nastavením	222
31.3.3 Odeslání poštovní zprávy s přílohou	222
31.4 Bezpečnost (Security)	223
31.4.1 Vracení jména současného přihlášeného uživatele	223
31.4.2 Informace o přítomnosti rolí v účtu uživatele	223
32 Práce se zdroji	223
32.1.1 Naplnění seznamu připojených zdrojů	224
32.1.2 Načtení řetězce ze zdroje	224
32.1.3 Načtení ikony ze zdroje a její vykreslení na formuláři	224
32.2 Konfigurační soubor	224
32.2.1 Čtení prvku z konfiguračního souboru	225
32.2.2 Výpis všech <AppSettings> klíčů a hodnot z konfiguračního souboru ..	225
33 Práce s procesy	225
33.1.1 Výpis probíhajících procesů do ListBoxu	225
33.1.2 Vyhledání konkrétního procesu	226
33.2 Volání externích procesů	226
33.2.1 Zobrazení www stránky	226
33.2.2 Zobrazení souborů v adresáři (vyvolání „průzkumníka“)	226
33.2.3 Zobrazení textového souboru v notepadu	226
33.2.4 Nastavení podrobnějších informací k vytvářenému procesu	227
34 Práce s vlákny	227
34.1.1 Vytvoření a spuštění nového vlákna	227
34.1.2 Vytvoření a spuštění nového vlákna s čekáním na jeho dokončení ..	228
34.1.3 Pojmenování současného vlákna	228
34.1.4 Nastavení priority současného vlákna na maximum	228
34.1.5 Přerušení činnosti vlákna na zvolenou dobu	228
34.1.6 Provedení operace asynchronně v jiném vláknu (.NET 2.0)	229
35 Regionální nastavení systému	229
35.1.1 Vracení jména nastavené kultury	230
35.1.2 Řazení dle abecedy s ohledem na nastavenou kulturu	230
35.1.3 Vracení symbolu pro měnu pro nastavenou kulturu	231
35.1.4 Procházení existujících kultur	231
35.1.5 Nastavení specifické kultury pro vlákno a zobrazení datumu v této kultuře	231
35.1.6 Výpis dnů v týdnu ve specifické kultuře	231
35.1.7 Změna systémového desetinného oddělovače	232
35.1.8 Přepnutí do němčiny – datum (vypisuje názvy dnů v němčině)	232
36 Práce s médii	232
36.1 Zvuky	232
36.1.1 Pípnutí	232
36.1.2 Přehrání WAV souboru	233
36.1.3 Přehrání zvuku přes System.Media.SoundPlayer (.NET 2.0)	233
36.1.4 Přehrání systémových zvuků – jsou-li přiřazeny (.NET 2.0)	233
36.2 Použití MCI příkazů	233
36.2.1 Přehrání MP3 souboru	234
36.2.2 Programové otevření dvířek CD mechaniky	234
36.2.3 Programové uzavření dvířek CD mechaniky	234
36.2.4 Přehrání AVI video souboru	234
36.2.5 Přehrání AVI video souboru s umístěním na pozici obrazovky	235
36.2.6 Přehrání MPEG video souboru se specifikací od-do snímku	236
36.2.7 Přehrání MPEG video souboru s uzavřením po přehrání	236
37 Práce s Windows Management Instrumentation (WMI) 237	
37.1.1 Dotaz na souborový systém používaný logickým diskem „c:“	238
37.1.2 Dotaz na informace o logickém disku „c:“	238
37.1.3 WMI dotaz na data o procesoru	239
37.1.4 WMI dotaz na data o biosu	240
37.1.5 Změna v systému aplikovaná pomocí WMI	240
38 Náповěda	241
38.1.1 Obsluha události HelpRequested	241
39 Implementace připravených rozhraní (interfaces) .. 241	

39.1.1 Implementace rozhraní IDisposable	241
39.1.2 Implementace rozhraní IEnumerable	242
39.1.3 Implementace rozhraní IComparable	244
39.1.4 Implementace rozhraní IFormattable	246
39.1.5 Implementace rozhraní ICloneable	247
40 Komplety a reflexe	248
40.1 Práce s komplety (assemblies)	250
40.1.1 Přístup k informacím o vstupním kompletu	251
40.1.2 Nalezení vstupního bodu kompletu	251
40.1.3 Výpis referencovaných kompletů ze vstupního kompletu	252
40.2 Reflexe	252
40.2.1 Dotazy ohledně typu proměnné	252
40.2.2 Převod aktuální hodnoty výčtu na řetězec	252
40.2.3 Informace o typech v prováděném kompletu	253
40.2.4 Zjištění informací o třídě (konstruktory třídy)	254
40.2.5 Procházení metod uvnitř daného typu	254
40.3 Dynamická kompilace kódu	254
40.3.1 Kompilace dynamicky vytvořeného kódu z programu -> vytvoření test.exe	255
41 Nezabezpečený kód a Garbage Collection	256
41.1 Nezabezpečený kód	256
41.1.1 Zjištění velikosti typu – přes sizeof	256
41.1.2 Přímá alokace dané velikosti ze zásobníku	256
41.1.3 Vytvoření reference na místo v paměti	257
41.2 Garbage Collection	257
41.2.1 Vynucení uvolnění nepotřebné paměti	257
41.2.2 Vynucení uvolnění prostředků pomocí bloku using	258
42 Přístup k Windows API	258
42.1.1 Volání funkce Windows API	258
42.1.2 Odeslání a přijetí uživatelské Windows zprávy	259
42.1.3 Získání HDC	259
43 Různé testování čehokoli...	260
43.1.1 Získání šířky pracovní plochy z proměnné Screen	260
43.1.2 Informace o názvu fontu pro menu (.NET 2.0)	260
43.2 SystemInformation	260
43.2.1 Jakým způsobem byl zaveden počítač?	260
43.2.2 Test na přítomnost myši	261
43.2.3 Další způsob získání šířky obrazovky	261
43.3 SystemEvents	261
43.3.1 Jednoduchý časovač (timer) přes SystemEvents	261
43.3.2 Obsluha ukončení sezení (session)	262
43.3.3 Testování nízké paměti	262
43.3.4 Reakce na změnu v nastavení displeje	262
43.3.5 Reakce na přidání/odebrání fontu do/ze systému	263
43.3.6 Reakce na změnu času stanice	263
44 Tipy a triky v práci s jazykem a prostředím	264
44.1.1 Přepsání chování existujících vlastností	264
44.1.2 Překrytí override metody	265
44.1.3 Přepsání metody ToString()	265
44.1.4 Využití klávesy <TAB> pro přidání kódu do třídy	266
44.1.5 Ochrana před celočíselným přetečením	266
44.1.6 Použití rezervovaného slova pro proměnnou, metodu, konstantu	267
44.1.7 Nová třída s využitím šablony třídy (.NET 2.0)	268
45 Něco z ostatních oblastí...	269
45.1 Aplikace	269
45.1.1 Filtrování specifického typu zprávy přes Application	269
45.1.2 Zpracování všech čekajících zpráv	270
45.1.3 Ukončení celé aplikace	270
45.1.4 Reakce na ukončení aplikace	270
45.2 Příkazový řádek	270
45.2.1 Získání celého příkazového řádku i s parametry	270
45.2.2 Procházení předaných parametrů příkazového řádku	271
45.3 Proměnné prostředí	271
45.3.1 Procházení a výpis hodnot existujících proměnných prostředí	271
45.3.2 Načtení hodnoty proměnné prostředí	271
46 Programové ladění programu	272
46.1.1 Výstup informace do ladicího okna	272
46.1.2 Výstup ladicí informace do kategorie	272
46.1.3 Podmíněný výstup ladicí informace	272
46.1.4 Stromový výpis ladicí informace	273

46.1.5 Výstup Assert() ladicí informace	273
46.1.6 Vyvolání chyby s možností ukončit program	273
46.1.7 Směrování ladicích hlášení do souboru a do proudu konzole	274
46.2 Záznam informací do žurnálu	274
46.2.1 Test existence daného typu žurnálu	274
46.2.2 Generování zápisu do žurnálu	274
46.2.3 Výpis hlášení ze žurnálu	274
47 C# a ASP .NET	275
47.1 Postup pro vytvoření ASP .NET aplikace v C#:	275
47.2 ASP .NET – příklady bez práce s daty	276
47.2.1 Prvotní inicializace formuláře	276
47.2.2 Dynamický výpis textu přes Response	276
47.2.3 Přepsání metody Page.Render()	277
47.2.4 Zobrazení dialogového okna s hlášením	277
47.2.5 Vyvolání stránky do dialogu	277
47.2.6 Odeslání e-mailu z ASP .NET	278
47.2.7 Vytvoření cookie a odeslání na stanici	278
47.2.8 Načtení hodnoty cookie	278
47.3 ASP .NET – práce s daty.....	279
47.3.1 Otevření dat a napojení na DataGrid	279
47.3.2 Jednoduchý jednorázový výpis dat do DataGridu	279
47.3.3 Změna barvy pozadí v ASP .NET DataGridu aplikované	
na třetím řádku	279
47.3.4 Dynamické formátování sloupce DataGridu na datum	280
47.3.5 Navrácení textu z odkazu LinkButton v DataGridu	280
48 Atributy	281
48.1.1 Tvorba nového atributu	281
48.1.2 Použití atributu – praktická ukázka na Obsolete atributu	282
49 Direktivy předprocesoru	282
49.1.1 Definování symbolu pro předprocesor	282
49.1.2 Podmíněná kontrola direktivy „SYMBOL“	282
49.1.3 Podmíněná kontrola „SYMBOL/SYMBOL1“ – použití operátoru OR ..	283
49.1.4 Využití direktivy #DEBUG	283
49.1.5 Využití direktiv #region a #endregion	283
50 Jak do programu přidat...	284
50.1 ...nový formulář?	284
50.2 ...nový jmenný prostor s novou třídou?	284
50.3 ...dll knihovnu?	284
50.3.1 Ukázková implementace nové .dll knihovny	284
50.4 ...uživatelský prvek (User Control)?	285
50.4.1 Umístění uživatelského prvku do projektu	285
50.4.2 Umístění uživatelského prvku do vlastní knihovny	285
50.5 ...konfigurační soubor?	286
50.6 ...novou vlastnost konfigurace? (Properties) (.NET 2.0)	286
51 Změny jazyka C# specifikace 2.0	287
51.1 Šablony typů (generics)	287
51.1.1 Omezení pro šablony typů (constraint)	288
51.2 Anonymní metody	289
51.3 Neúplné třídy (partial class)	289
51.4 Null–podporující typy (nullable types)	290
51.4.1 Zjištění, zda null–podporující typ má přiřazenou hodnotu	290
51.4.2 Vracení jiné hodnoty, je-li null–podporující proměnná null	290
Rejstřík	291

1 Řešené příklady v C# aneb C# skutečně prakticky

Dostává se Vám do rukou publikace, při jejíž tvorbě byl brán zřetel na maximální jednoduchost a praktičnost při každodenním použití.

Prvotním impulsem, který vedl k napsání této knihy, bylo poskytnout jakýsi "lexikon", který programátor vezme do ruky, když mu bude něco nejasné, když nebude zrovna vědět, jak to či ono implementovat, když bude pouze tušit, že snad takto nějak by to přece mohlo být, že snad takto nějak to již někdy dělal, jakpak to jen ale bylo...?

Podařilo-li se mi tento úkol splnit, musí čtenář posoudit sám. Doufám, že mu bude tato publikace sloužit k jeho plné spokojenosti.

1.1 Rozsah knihy

Je zřejmé, že žádná kniha tohoto „pragmatického rozsahu“ není schopná obsáhnout základy jazyka C#, popsat .NET model i s jeho metodami, vývojové prostředí *Visual Studio* či samotného C# a k tomu ještě přinést již zmíněný lexikon příkladů a postupů.

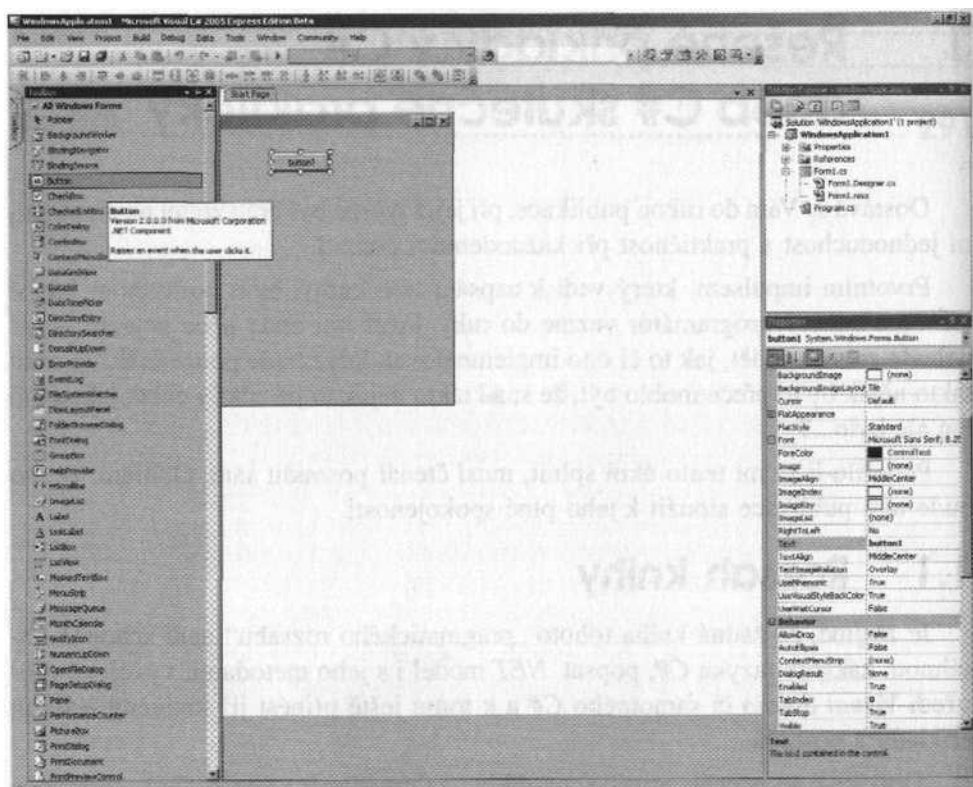
Dotknu-li se něčeho nového, například zabrousím-li ve výkladu k *překrytým (override)* metodám, pokusím se na dvou třech řádcích pojem vysvětlit. Zmiňuji se o tom proto, že někomu by se mohl zdát výklad některých termínů příliš stručný - je to však proto, že z hlediska účelu této knihy jsou tyto výrazy uvedeny spíše navíc a předpokládá se, že čtenář má již určitou zkušenost s programováním i se znalostí, prací a použitím objektového programovacího jazyka.

1.2 Verze .NET s příklady

Většina příkladů byla testována v prostředí *Microsoft Visual Studio (C#)* s knihovnou *.NET Framework 1.1*. Lze předpokládat, že velká část z nich bude funkční i v dalších verzích .NET knihovny.

Ukázky, které mají za svým titulkem poznámku „(NET2.0)“ byly testovány v beta verzi *Microsoft C# Express 2005 Beta*, která již nese novější verzi knihovny. *.NET 2.0. Microsoft C# Express* si lze pak volně stáhnout na adrese:

<http://lab.msdn.microsoft.com/express/vcsharp/default.aspx> Příklady .NET 2.0 nebudou na starší verzi NET 1.1 pracovat.



1.3 Typografické konvence

Rezervovaná slova jazyka *C#* a názvy objektů, struktur, rozhraní, výčetů apod. náležející přímo knihovně *.NET* budou uvedeny **tučným písmem**.

Odkazy na názvy vlastních proměnných, čísel, řetězců a názvů technologií budou zvýrazněny *kurzívou*.

Ukázkové kódy pak budou odlišeny tímto podbarvením.

1.3.1 Překlady termínů

Kniha se snaží užívat české termíny, doprovázené alespoň v prvním případě anglickým překladem. Překlad všech důležitějších termínů lze nalézt zde:

bezpečnost	- security
datový proud (tok)	- stream
fronta	- queue
hašovací kód	- hash code
jmenný prostor	- namespace
komplet	- assembly
ladění	- debugging
výčet	- enumeration
omezení	- constraint
poklepání myši	- doubleclick
poskytovatel	- provider
průhlednost	- opacity
přetypování	- typecasting
sezení	- session
schránka	- clipboard
událost	- event
uživatelský prvek	- usercontrol
vázání dat (vazby)	- databinding
vlastnost	- property
(neúplné třídy v <i>C#</i>)	- partial types
(šablony typů v <i>C#</i>)	- generics
zásobník	- stack
zdroj	- resource

1.4 Některá pravidla pro použité zdrojové kódy

Hned na začátku bude definováno, jakým způsobem se bude se zdrojovým kódem pracovat. To se týká zejména použití jmenných prostorů a určení výstupních „nástrojů“ pro vrácené výsledky z metod.

1.4.1 Pravidla pro užití jmenných prostorů (namespace)

Pro ilustraci předpokládejme tuto deklaraci, která připojuje objekt **XmlDocument**, jenž představuje dokument ve formátu *XML*:

```
XmlDocument doc = new XmlDocument();
```

V případě, že nebude v hlavičce připojena deklarace **using** pomocí:

```
using System.Xml;
```

oznámí kompilátor při kontrole chybu.

Druhou možností, jak jmenný prostor určit, a to již bez použití příkazu **using**, je jmenným prostorem přímo specifikovat objekt. Pak by deklarace vypadala takto:

Tento způsob sice ušetří jedno použití **using**, nicméně znamená nutnost pokaždé prefixovat objekt příslušným jmenným prostorem, což se vztahuje i na každý jiný prvek, např. výčet, který se ve jmenném prostoru nachází.

Proto tato publikace používá první způsob s tím, že bude pokaždé zdůrazněno, je-li nutné připojit některý další jmenný prostor. Výjimku tvoří příklady pro *.NET 2.0* - zde bude použita celá specifikace jmenného prostoru, nebude-li řečeno jinak.

1.4.2 Výstupy pro příklady

Pro ilustraci výstupů mohou být v kódech použity tyto elementy:

- 1) `MessageBox.Show()` - zobrazuje údaj do dialogového boxu
- 2) `listBox1` - prvek **ListBox** pro výpisy rozsáhlejších informací
- 3) `textBox1` - případně i prvek **TextBox**

Dále je nutné poznamenat, že některé příklady se mohou zdát na první pohled nesmyslné a že by bylo lepší řešit je zcela jinak. Příkladem může být např. kód uvedený u příkazu **continue**:

```
for ( int i = C; i < 3; i++ )
{
    if { i == 1 ) continue; MessageBox.Show( "Krok č. " + i.ToString());
}
```

U tohoto kódu se vždy zobrazí dialog pouze jednou - potom se smyčka opustí. Pak by bylo samozřejmě jednodušší vypsát dialog přímo, bez použití smyčky. Pro ilustraci příkazu **continue** však tento kód vyhovuje - i když testování v reálném životě vypadá často složitěji. To by však navíc vyžadovalo i popis proměnných, které do testování logické podmínky **continue** vstupují, což by čtenáře zbytečně obtěžovalo - a to není cílem této knihy.

1.4.3 Použití znaků.... "v kódu

Tři znaky teček („...“), které budou použity uvnitř kódu, znamenají „libovolný další jiný kód“.

1.4.4 Upozornění pro začínající programátory!

Jazyk C# stejně jako jazyky C/C++ rozlišuje velikost písmen! Lze proto použít kód:

```
int Int = 10;
```

a jeho výsledkem je deklarace a inicializace proměnné *Int* datového typu **int**.

2 Něco ze základních postupů C#

V této kapitole si ukážeme některé základní pravidla práce s jazykem C#.

2.1 Přiřazení hodnot proměnným

Začneme deklaracemi a přiřazením hodnot proměnným.

2.1.1 Deklarace proměnné typu int s inicializací

int představuje velmi často používaný celočíselný typ.

```
int i = 5;
```

Následně je demonstrativně provedeno přiřazení nové hodnoty do této proměnné.

2.1.2 Deklarace a přiřazení proměnné typu float s inicializací

Typ **float** obecně vyhovuje pro požadavky na uložení čísla s desetinnou čárkou. Aby kompilátor věděl, že desetinné číslo je přiřazováno jako **float**, musí mu to být řečeno znakem „f“ za přiřazovanou hodnotou. Jinak je číslo chápáno jako **double**.

```
float fValue = 56.75f;
```

2.1.3 Deklarace proměnné typu bool s inicializací

bool je logická proměnná, která může nabývat stavů *Ano/Ne* - (**true/false**).

```
bool bValue = false; bValue = true;
```

Následně je demonstrativně provedeno přiřazení nové hodnoty do této proměnné.

2.1.4 Přiřazení a změna řetězcové hodnoty

Řetězcová hodnota představuje text. Tento text je uvozen znakem uvozovek („“).

```
sValue = "Změna hodnoty";
```

Následně je demonstrativně provedeno přiřazení nové hodnoty do této proměnné.

2.1.5 Přiřazení hexadecimální hodnoty (šestnáctková soustava)

```
int i = '\x0201';
```

Přiřazovaná hexadecimální hodnota představuje desítkově číslo 513.

2.1.6 Přiřazení prvku z výčtu

```
enum Vyska {
    mala, stredni, vysoka }
private void button1.Click(object sender, System.EventArgs e) { Vyska vyska = Vyska.stredni;
}
```

Bázový typ výčtu (jeho prvků) jsou implicitně **int** hodnoty. Hodnoty prvků jsou číslovány vzestupně, přičemž pomocí

znaménka „=" je možné za prvkem přiřadit explicitně konkrétní **int** hodnotu. Pro možnost přiřazení jiného typu nežli **int** lze výčet deklarovat takto:

```
enum Vyska: byte
{...}
```

2.1.7 Přiřazení prázdné reference

Nadsazeně si lze toto přiřazení představit jako nastavení nuly do číselného typu (**int**, **double**). Říká, že reference *ib* nemá přiřazenou hodnotu - nikam neukazuje.

2.2 Větvení programu

2.2.1 if

Základní větvicí příkaz. Logický výraz musí být v závorkách. Spojování jednotlivých podmínek výrazu se provádí pomocí logických operátorů.

```
if ( ( i > 3 ) && ( i < 10) )
    MessageSex.Show( "Splněn" );
else
    MessageSex.Show( "Nesplněno");
```

Testuje, zda je *i* větší nežli 3 a menší nežli 10. Nejpoužívanější logické operátory pro spojování výrazů:

&& - logické A
|| - logické NEBO
! - negace výrazu (uvádí se před výraz)

2.2.2 switch

Umožňuje přehledně rozvětvit kód dle více testovaných hodnot jednoho výrazu.

Výraz za **switch** musí být v závorce. Každá alternativa čase musí být ukončena explicitním přenosem řízení mimo příkaz **switch** nebo na některé jiné návěští **case** v témže příkazu **switch**. To může obstatat příkaz **break**, **goto**, **continue**, **return**, **throw**.

Blok **default** je proveden, když hodnota nebyla mezi čase hodnotami naleze- na. Tento **default** blok je nepovinný.

```
int i = 2;
switch (i)
{
    case1: MessageBox.Show( "Jedna" ); break; case2 :
    MessageBox.Show ( "Dvě" ); break;
    case3 : MessageBox.Show( "Tři" ); break;
    default : MessageBox.Show( "Jiná" ); break;
}
```

2.2.3 goto

Přímý odkok na jiné místo v rámci metody. Místo, kam se tímto příkazem přenese řízení, je označeno *návěstím*, za kterým je dvojtečka.

```
int i = 1;

zvysit:
    MessageBox.Show( ( i++ ).ToString() );
```

2.2 Smyčky

2.3.2 Příkaz while

Cyklus s testem podmínky na počátku. Vykonává daný počet opakování po dobu, po kterou je splněna úvodní podmínka. V případě nesplnění této podmínky se smyčka opouští.

```
int i = 1;
bool bWhile = true;
while ( bWhile )
{
    MessageBox.Show( i.ToString());
    i++;
    bWhile = ( i < 4 );
}
```

Tento kód zobrazí 3x příslušný dialog.

2.3.3 Příkaz do...while

Velmi podobné smyčce **while**, avšak test se provádí až na konci - to znamená, že smyčka se v každém případě (není-li jinak násilně přerušena) provede alespoň jednou.

```
int i = 1;
bool bWhile;
MessageBox.Show( i.ToString () );
bWhile=(i < 4 );
```

Tento kód zobrazí 3x příslušný dialog.

2.3.4 Příkaz foreach

Jazyky C ani C++ tento druh smyčky nativně neobsahovaly - zavádí ji až C#. Umožňuje procházet prvky kolekcí nebo polí.

```
string[] aArray = new string[] {"Jedna", "Dva", "Tři"};
foreach ( string sValue in aArray ) MessageBox.Show( sValue );
```

Vypíše postupně všechny tři prvky pole.

Pozn.: Aby kolekce mohla být ve **foreach** použita, musí implementovat rozhraní **IEnumerable** - to s sebou nese povinnost implementace metod tohoto rozhraní.

2.3.5 continue prušení smyčky -> skok na začátek smyčky

```
for ( int i = 0; i < 3; i++ )
{
    if ( i == 1 ) continue;
    MessageBox.Show( "Krok č." + i.ToString() );
}
```

Vypíše dialog s hodnotou „0“ a pak s hodnotou „2“. Vypis „1“ je vynechán.

2.3.6 break - přerušování smyčky -> úplné ukončení smyčky

```
for ( int i = 0; i < 3; i++ )
{
    if ( i == 1 ) break;
    MessageBox.Show( "Krok č." + i.ToString() );
}
```

Vypíše pouze dialog s hodnotou „0“. Pak je smyčka přerušena a pokračuje se až za ní.

Pozn.: Příkaz **break** se často používá k ukončení alternativ čase v příkazu **switch**.

2.3.7 Implementace nekonečné smyčky

```
while ( true )
{
....
break;
}
```

Zobrazuje dva asi nejpoužívanější způsoby pro vytvoření nekonečné smyčky. Smyčka se opouští pomocí rezervovaného slova break.

2.4 Volání metod a předávání parametrů

Metody a jejich volání tvoří jednu z nejdůležitějších oblastí, které je nutné zvládnout.

pozn.: Metoda je název pro funkci, která je deklarována v rámci objektu.

2.4.1 Předávání parametrů metodám

Správná deklarace pro funkci s parametry je tato:

```
public TestHash( string item1, string item2 ) Pozor! Nelze použít tvar: public TestHash (string item1, item2 )
```

2.4.2 Vyvolání metody s parametry a vrácení výsledku metody

```
int GetSum( int _iParam1, int _iParam2 )
{ re turn _iParam1 + _iParam2);
private void button1_Click(object sender, System.EventArgs e)
}
int iResult = GecSum( 10, 20 );
MessageBox.Show( iResult.ToString() );
.....
```

Vypíše hodnotu „30“.

2.4.3 Předání pole jako parametru

```
void write_array( string [ ] array )
{
string sText = "";
for ( int i = 0 ; i < array.Length; i++) sText = sText + " " + array[i].ToString();
MessageBox.Show( sText );
}
private void button1_Click(object sender, System.EventArgs e)
{
string[] sArray = { "Ahoj", "člověče" }; write_array ( sArray );
}
```

Předání parametrů přes pole a jejich výpis uvnitř metody. Lze tak poměrně jednoduše předat více parametrů, aniž je nutné každý zvlášť deklarovat v hlavičce metody.

2.4.4 Vyvolání metody s parametry a vrácení hodnoty přes parametr (out)

```
void GetSum ( int _iParam1, int _iParam2, out int _iResult )
_iResult = _iParam1 + _iParam2;
private void button1_Click( object sender, System.EventArgs e)
{
    int iResult;
    GetSum( 10, 20, out iResult );
    MessageBox.Show(iResult.ToString() );
}
```

Vyvolá metodu *GetSum()* s parametry *10* a *20*, provede jejich součet a ten vrátí přes parametr typu **out**.

Pozor! Nese-li metoda parametr **out**, musí být s indikací tohoto parametru i vyvolána - čili, i před příslušným parametrem ve volání metody musí být rezervované slovo **out** použito.

Výsledek volání je uložen do proměnné *iResult* (zde hodnota *30*).

2.4.5 Vyvolání metody s parametry a vrácení hodnoty přes parametr (ref)

```
void GetSum{ int _iParam1, int _iParam2, ref int _iResult )
{
    _iResult = _iParam1 + _iParam2;
    private void button1_Click(object sender, System.EventArgs e)
    {
        int iResult = -1;

        Get.Sum( 10, 20, ref iResult );
        MessageBox.Show ( iResult.ToString());
    }
}
```

Vyvolá metodu *GetSum()* s parametry *10* a *20*, provede jejich součet a ten vrátí přes parametr typu **ref**.

Pozor! Nese-li metoda parametr **ref**, musí být s indikací tohoto parametru i vyvolána - čili i před příslušným parametrem ve volání metody musí být rezervované slovo **ref** použito.

Rozdíl oproti volání **out** je v tom, že hodnota **ref** parametru musí být navíc inicializována před prvním použitím v **ref** volání metody.

Výsledek volání je uložen do proměnné *iResult* (zde hodnota *30*).

2.4.6 Volání statické metody deklarované v jiném jmenném prostoru

Máme-li nový jmenný prostor s objektem *MyMath*, jenž nese jednu statickou metodu *MyMath.Sum()*:

```
namespace MyMathNamespace
public class MyMath
{
    static public int Sum( int iValue1, int iValue2 )
        return ( iValue1 + iValue2 );
}
```

pak pro volání metody ze základní jednotky použijeme: `using MyMathNamespace;`

3 Přetěžování operátorů

V jazyce C# je možné, podobně jako v C++, přetížit určité operátory dané třídy, což vlastně znamená, že si definujeme jejich nový význam - a to opět v rámci té a té třídy, v nichž je přetěžujeme.

Názorně: řekněme například, že máme jednoduchou třídu *MyInt*, která eviduje jednu **int** hodnotu. Pokud použijeme operátor „+“ mezi dvěma instancemi tříd *MyInt*, kompilátor oznámí chybu, protože neví, jak tyto instance sečíst. Abychom si pomohli, můžeme v takovém případě „přetížit“ operátor „+“ ve třídě *MyInt* a určit, co bude vráceno při použití konstrukce „*MyInt* + *MyInt*“.

Přetěžovat v rámci třídy lze unární operátory (+, -, ! atd.), binární operátory (% , &, <<, >> atd.) a srovnávací operátory (=, >, < atd.).

Podobným způsobem lze také přetížit implicitní a explicitní konverzi (třídy) na daný typ třídy.

3.1.1 Přetížení operátoru „+“ ve vlastní nové třídě

```
class MyInt
{
    int value; public MyInt( int value)
    {
        this.value = value;
    }
    public static MyInt operator+ (MyInt left, MyInt right )
    return( new MyInt ( (left.value + right.value) * 2 ) );
}
public override string ToString()
return Convert.ToString( value );
}
}
```

Volání:

```
MyInt i1 = new MyInt ( 10 );
MyInt i2 = new MyInt ( 20 );

MyInt result = i1 + i2;
MessageBox.Show (result.ToString() );
```

zobrazí text *60* - dvojnásobek součtu *10+20*.

pozn.: Ve třídě je vidět způsob přepsání metody *MyInt.ToString()* pro snadný textový výstup.

3.1.2 Přetížení operátoru pro implicitní přetypování na typ int

Vyjdeme-li z předchozí třídy *MyInt*, pak, aby mohlo být použito:

```
int i = result
```

musí být operátor (**int**) rovněž ve třídě přetížen v této metodě:

```
public static implicit operátor int( MyInt obj )
{
    return( obj.value );
}
```

To způsobí, že do proměnné *i* (typu **int**) bude uložena hodnota *60*.

Pozn.: Implicitní konverzi používáme, když se nová hodnota „vejde“ do cílové a nehrozí ztráta dat. Typickým příkladem je převod hodnoty typu **int** do **long -long** má dost prostoru, aby **int** hodnotu pojalo.

3.1.3 Přetížení operátoru pro explicitní přetypování na typ int

Vyjdeme-li opět z předchozí třídy *MyInt*, pak aby mohlo být použito:

```
int i = (int)result;
```

musí být operátor (**int**) rovněž ve třídě přetížen v této metodě:

```
public static explicit operátor int i (MyInt obj )
{
    return i(obj, value );
}
```

To způsobí, že do proměnné *i* (typu **int**) bude uložena hodnota *60*.

Pozn.: Explicitní konverzi používáme, když se nová hodnota „nevejde“ do cílové a hrozí ztráta dat (např. velký **long** do menšího **int**). Také se používá pro konverze mezi instancemi tříd. Je-li definováno implicitní i explicitní přetížení operátoru na stejný typ, kompilátor vyvolá chybu.

3.1.4 Přetížení operátorů true a false

Po přetížení operátora **true** a **false** lze instance třídy přímo podrobovat logickému testování.

```
class MyInt
{
    public int value;
    public MyInt( int value )
    {
        this.value = value;
    }
}

public static bool operátor true ( MyInt obj )
{
    return obj.value > 0;
}

public static bool operátor false ( MyInt obj )
{
    return obj.value <= 0;
}
}
```

```
MyInt i = new MyInt ( -100 );
MessageBox.Shcw ( "true" );
MessageBox.Show( "false" );
```

Zde vytvořená instance třídy *MyInt* vrací **true**, je-li hodnota proměnné *instance třídy MyInt.value* větší než nula. V opačném případě je **false**.

Pozn.: Při přetěžování operátoru **true** je nutné přetížít i **false** - a naopak.

3.2 Indexery (indexers)

Indexer umožňuje odkazovat podobným způsobem jako pole přes index na hodnotu, kterou vrací. Lze tak přistupovat k instanci třídy, která ho implementuje.

3.2.1 Vytvoření jednoduchého indexem v rámci formuláře

```
public string this[ int iValue ];
{
return ( iValue.ToString());
}
private void button1_Click(object sender, System.EventArgs e!)
{ MessageBox.Show( this[50] );
}
```

Tento *indexer* nemá sice žádný praktický účel, ukazuje však, jak *indexer* (zde v rámci formuláře) vytvořit a volat. V tomto příkladě převádí předanou hodnotu 50 na řetězec, který vrací, a jenž se následně zobrazuje. Obecně se pak *indexer* používá k přístupu ke členským polím či kolekcím třídy.

Pozn.: Na rozdíl od C++ se v C# za přetěžování operátoru [] nepovažuje. Přesto je tato část *zařazena* do této kapitoly.

4 Práce s instancí objektů

Třída v jazyce C# může mimo jiné poskytovat tzv. statické proměnné a metody, ke kterým je možné přistupovat přímo, aniž budeme cokoli deklarovat či vytvářet.

Chceme-li však pracovat s proměnnou, která je *instancí* datového typu **object**, musíme tuto proměnnou nejprve vytvořit - říkáme, že vytváříme instanci objektu.

Zde je nutné upřesnění. V případě, že je např. prvek **TextBox** vložen na formulář umístěním ve vývojovém prostředí (v případě C# obvykle *Visual Studio*), není potřeba vytvářet instanci prvku programově - integrované prostředí generuje automaticky odpovídající kód, který se volá při startu aplikace.

4.1 Dynamické vytvoření a zrušení instance objektů

4-1 • 1 Vytvoření nové instance objektu Timer

```
System.Timers.Timer timer = new System.Timers.Timer();
timer.Interval = 500;
```

Instance objektů se vytvářejí přes operátor **new**. Pokud to pro vás bude přijatelnější, instance je vlastně totéž, co proměnná. Tak se pouze nazývají dynamicky vytvářené proměnné.

Pozn.: To může být v C# tak trochu zavádějící, neboť objekt je tu každý typ (i ten, který nese hodnotu - např. **int**).

4.1.2 Zrušení objektu Timer

```
System.Timers.Timer timer = new System.Timer();
timer.Interval = 500;
.....
timer.Dispose;
```

Instance objektů se v C# ruší automaticky. Není proto nutné rušit je manuálně a zatěžovat se navíc pamatováním na to, kdy a kde instanci uvolním. V určitých situacích to však může být výhodnější. Explicitní rušení prostředků se provádí přes přepsanou metodu **Dispose()** z rozhraní **IDisposable**. Důležitou podmínkou však je, aby daná třída toto rozhraní implementovala.

4.1.3 Použití klíčového slova this

U práce s instancemi je potřeba vysvětlit význam klíčového slova **this**. To odkazuje na vytvořenou instanci třídy, ve které je **this** v kódu použito.

Nejlépe to lze pochopit na příkladu. Předpokládejme, že máme jednoduchou třídu pro evidenci jedné číselné **int** hodnoty. Třída obsahuje konstruktor, který **int** předanou hodnotu ukládá do členské proměnné třídy:

```
class MyInt
{
    int value;
    public MyInt( int value )
    this.value = value;
}
```

K vytvoření instance této třídy použijí následující volání:

```
MyInt il = newMyInt( 10 );
```

Abych byl v rámci konstruktem **MyInt()** schopen rozlišit, zda chci pracovat se členskou proměnnou třídy či se stejně pojmenovaným parametrem konstruktoru, použijí klíčové slovo **this**. V takovém případě pak přistupuji k vytvořené instanci, kterou slovo **this** uvnitř třídy zastupuje - čili k proměnné instance této třídy. Volání **this.value** použité v konstruktoru třídy je vlastně stejné, jako volání **il.value**, volané však mimo třídu, přímo na vytvořené instanci.

4.2 Práce s konstruktory

Konstruktor je volán těsně před vytvořením vlastní instance datového typu. Třída může deklarovat konstruktor, ale nemusí - v takovém případě je použit konstruktor implicitní. Konstruktor má stejné jméno jako třída, může jich být i více, existuje-li požadavek na inicializaci instance s rozdílnými parametry.

4.2.1 Konstruktor s jedním parametrem

```
public class MyClass
{
    public MyClass ( int iValuel )
    {
        .....
        MessageBox.Show( iValuel.ToString() );
    }
}

MyClass m1 = new MyClass( 10 );
```

Při vytvoření každé nové instance dochází k předání celočíselné hodnoty „10“. Ta je v tomto případě zobrazena.

4.2.2 Statický konstruktor

Statický konstruktor je vyvolán pouze pro první vytvářenou instanci daného datového typu

```
public class myClass
{
    public MyClass (int iValue1)
    {
        .....
        MessageBox.Show(iValue1.ToString())
    }
    Static MyClass()
    {
        .....

        MessageBox.Show( "První" );
    }
}
MyClass m1 = new MyClass( 10 ); MyClass m2 = new MyClass( 20 );
```

V tomto případě dochází k zobrazení textu „*První*“ a "10" pro první vytvářenou instanci datového typu *MyClass*; a textu "20" pro druhou vytvářenou instanci.

4.3 Testy na typy objektů

Instance objektů určitého datového typu lze přetypovat na jiný datový typ. To lze tehdy, existuje-li mezi třídami instance přímá závislost: např. lze přetypovat **TextBox** na základního předka **object**, protože ho již „v sobě obsahuje“; lze přetypovat předanou instanci datového typu **object** na **TextBox**, vím-li, že instance byla vytvořena jako **TextBox**. Nelze však přetypovat **TextBox** na **DataGrid**, protože se jedná o zcela jiný nezávislý objekt.

K tomu, aby se mohlo správně a bez chyb přetypovávat, pomohou následující příklady.

4.3.1 Test na třídu instance objektu (či na předka objektu)

K testování, zda daná instance je dané třídy, se používá operátor **is**.

```
private void listBox1.DoubleClick(object sender, System.EventArgs e)
{
    bool b = ( sender is ListBox); MessageBox.Show( b.ToString () );
}
```

Tento příklad vrací **true**, je-li předaná instance v parametru *sender* třídy **List Box** (je-li odvozená z této třídy).

4.3.2 Test na třídu instance objektu a její přetypování na zvolenou třídu

```
private void listBox1.DoubleClick(object sender, System.EventArgs e)
{
    if ( sender is ListBox)
        ListBox li = (ListBox)sender;
}
```

Testuje, zda je předaná instance objektu v parametru *sender* typu **ListBox**; pokud ano, provádí přetypování instance na typ **ListBox**.

4.3.3 Výpis typu instance objektu

```
private void button1_Click(object sender, System.EventArgs e)
{
    /* pokud není Button, tak... */
    if ( !( sender is Button ) )
        MessageBox.Show( "Prvek není typu Button, ale " + sender.GetType() .ToString());
    return;
}
/* je-li Button, pokračuj zde */
Button button = (Button)sender;
.....
}
```

Testuje, zda parametr *sender* není typu **Button** nebo potomek této třídy. Pokud není, vypíše hlášení, do něhož specifikuje, o jakou třídu objektu se tedy jedná.

4.4 Přetypování

Přetypování umožňuje, aby se s instancí jednoho typu objektu zacházelo jako s instancí jiného typu objektu. To znamená, že objekt je deklarován jako jeden typ, pracuje se s ním jako s jiným typem. Toto však nelze provést u každé z kombinací typů objektů. Lehce lze přetypovat potomka na předka, dobře akceptované jsou také přetypování mezi numerickými typy (v jazyce C# je však stejně každý typ reprezentován jako objekt - což je určitou odlišností od jiných konvenčních objektových jazyků).

4.4.1 Explicitní přetypování double na int

```
double d = 35.6; int i = (int)d;
```

Uloží do celočíselné proměnné *i* **double** hodnotu *d* a to s přetypováním na typ **int**. Uloženo bude *35* - **int** samozřejmě nepracuje s desetinnými místy. Toto přety-pování na (**int**) se nazývá explicitní a mohli jsme si ho dobře všimnout při tvorbě explicitního přetypování u operátorů.

Bez tohoto přetypování by kompilátor ohlásil chybu - jde o konverzi, při níž *1* se ztrácí informace, a proto je vyžadováno, aby ji programátor vyjádřil explicitně. Všechny „zužující“ konverze musí být v C# explicitně předeepsány.

4.4.2 Přetypování parametru sender na třídu Button

```
private void button1_Click(object sender, System.EvemArgs e)
{
    Button b = (Button)sender;
    MessageBox.Shcw( b.Name );
}
```

Parametr *sender* nese odkaz na objekt, který událost vyvolal - v případě tohoto kódu se jedná o událost, která obsluhuje stisk tlačítka. Víme-li tedy, že se jedná např. o tlačítko, mohu přímo přetypovat předaný typ **object** na **Button** a dostat se k jeho vlastnosti **Button.Name**, kterou samotný typ **object** neposkytuje. Tento příklad vypíše název stisknutého tlačítka „*button1*“.

4.5 Deklarace vlastností (properties)

Vlastnosti (properties) zastupují přístup k proměnným třídy. Programátor tedy přes vlastnosti k členským proměnným třídy přistupuje; čili čte, popřípadě upravuje jejich hodnotu. Vlastnost je tedy jakýsi „prostředník“ mezi programátorem a proměnnou uvnitř třídy.

Po pravdě řečeno, programátor je schopen vyjít i bez použití vlastnosti a to tak, že deklaruje proměnné jako veřejně přístupné a pak přistupuje přímo k nim. Tato technika se však nedoporučuje.

Velkou výhodou použití vlastností vidím osobně v tom, že po změně proměnné mohu jako programátor říci, že má dojít i k následné úpravě další proměnné či skupiny proměnných, že mohu okamžitě překreslit upravené grafické vyjádření instance objektu, jedná-li se o grafický objekt, apod. Pokud budu přistupovat přímo k proměnným instance objektu, musím na vše toto myslet a vystavuji se riziku opomenutí a vzniku chyby. Rovněž i pozdější úprava kódu je pro mě jako pro programátora náročnější, musím sledovat všechny tyto závislosti, které jsem mohl včlenit do obsluhy pomocí vlastností.

```
public int Value1 get { return this.iValue1; }
if ( value > 10 ) value - 10;
this. iValue1 = value;
doAction ();
private void doAction() { /* volitelné následné akce */ }
```

V rámci třídy MyClass je vidět nová vlastnost Value1, která podporuje čtení hodnoty (get), tak i její úpravu (set). Vlastní hodnota je potom uložena v privátní(=z venku třídy neviditelné) proměnné iValue1.

V bloku set je před změnou testováno, zda nová hodnota, reprezentovaná pomocí value, není větší než 10. Pokud ano, sníží se nová hodnota na 10. Volání doAction() pak ilustruje i možnost zpracování následných akcí, které se provedou bezprostředně po změně proměnné.

```
MyClass myclass = new MyClass();
myclass.Value1 = 20;
MessageBox.Show(myclass.Value1.ToString());
```

Tento příklad zobrazí hodnotu 10 (v set části dochází ke snížení dle zadané Podmínky).

5 Práce s událostmi

Událost (event) představuje místo, které je vyvoláno pro obsluhu určité akce - typickým příkladem je reakce na stisk tlačítka. Obsluhu události musí mít třída implementována (lze samozřejmě vytvořit událost vlastní).

Pro události se v jazyce C# vytváří objekt typu **delegate**, který událost zastupuje. To mimo jiné umožňuje přiřadit pomocí operátoru += na danou událost více reakcí. Operátor -= ji může zase odebrat.

5.1.1 Dynamické přidělení události

Událost lze přidělit přímo v návrhu formuláře v editoru vývojového prostředí (pro C# obecně Visual Studio) - pak je událost s prvkem svázána staticky (což už(můžeme programově změnit); alternativou je dynamické přidělení události přímo za běhu programu - např. až po startu formuláře, po stisku tlačítka, apod.

Přidělí-li se na obsluhu události např. dvě reakce, jsou také dvě (po sobě) vyvolány

```
private void resizeProc( object sender, System.EventArgs e )
MessageBox.Show("Resize");
}
this.Resize += new EventHandler (resizeProc);
```

Pokud se provede:

```
this.Resize -= new EventHandlen (resizeProc);
```

a událost není zatím přiřazena, nic se neděje.

5.1.2 Přidělení více reakcí na jednu událost

Dynamické přiřazení dvou metod pro obsluhu stisku tlačítka button1:

```
private void Click1 ( object sender, EventArgs e )
MessageBox.Show ( "1" );

private void Click2 (object sender, EventArgs e)
{
    MessageBox.Show( "2" );
}

button1.Click += new EventHandler( Click1 );
button1.Click += new EventHandler( Click2 );
```

Po stisku tlačítka button1 se zobrazí dialog s „1“ a pak s „2“.

5.1.3 Odstranění jedné reakce na událost (jsou-li již přiděleny)

Dynamické přiřazení dvou metod pro obsluhu stisku tlačítka *button1* s následnou ukázkou odebrání metody *Click1*:

```
private void Click1( object sender, EventArgs e )
KessageBox.Show( "1" );

private void Click2 (object sender, EventArgs e)
MessageBox.Show ( "2" );

burtton1.Click += new EventHandler ( Click1 ); button1.Click += new EventHandler ( Click2 );
button1.Click. -= Click1;
```

Metoda je z obsluhy události odstraněna pomocí operátoru -=. Zobrazí se pouze dialog s „2“.

5.1.4 Ovlivnění výsledku přes událost

Často se události používají i ve smyslu notifikačním - v prototypu metody události je tento parametr předán referencí (odkazem).

Obsluha vlastní metody události pak tuto proměnnou buď ponechá nebo upraví. Podle toho, co je v tomto parametru vráceno, se pak chová program, který metodu události vyvolal.

```
Public delegate void CalcHandler( double d1, double d2, ref double result ); public class Suma public Suma( double d1, double d2 )
{
    this.d1 = d1;
    this.d2 = d2;
}

Public double doCalc()
{
```

6.1.1 Obsluha obecné (nekonkrétní) výjimky

```
int i = 10;
int i = 0;
try
{
    i = i / i1;
}
catch
{
    MessageBox.Show( "Nastala vzjímka chyba");
}
```

Předpokládám-li, že výjimka může nastat a neznám přesně její typ (=třidu), výjimka pro mě nemá konkrétní podobu, nevím co očekávám, jen tuším místo, kde by mohly nastat problémy, definuji blok try...catch. Ten libovolnou výjimku zachytí a umožní obsloužit.

6.1.2 Obsluha konkrétní výjimky

```
int i = 10;
int i = 0;
try
{
    i = i / i1
}
catch(System.DivideByZeroExceptions)
{
    MessagBox..Show( "Nastalo dělení nulou." );
}
catch
{
    MessagBox..Show( "Nastala vyjimka chyba." );
}
```

Vim-li předem, který typ výjimky by mohl nastat, obsloužím její typ přímo, úvěrem je možné obsloužit všechny ostatní výjimky. Zobrazí hlášení „*Nastalo dělení nulou.*“.

6.1.3 Získání bližších informací o výjimce

```
int i = 10;
int i = 0;
double result= ( d1 + d2 );
AfLerCalc( d1, d2, ref result );
return (result);
double dl;
double d2;

public event CalcHandler AfterCalc;
private void AfterCalcProc( double dl, double d2, ref double result )
{
    MessageBox.Show( Scring.Format( "Událost proběhla - {0}, {1}", dl, d2 ) );
    result *= 2; }
private void button1_Click(object sender, System.EventArgs e) { Suma s = new Suma( 10, 50 );
s.AfterCalc += new CalcHandler( AfterCalcProc);
double d = s.doCalc();
MessageBox.Show ( String.Format("Výsledek = {0}", d ) );
```

V tomto příkladě je při *Suma.doCalc()* vyvolána událost *AfterCalcProc*. Ta má parametr *result*, který je typu *ref*, což umožňuje přes změnu tohoto parametru v metodě události ovlivnit jeho hodnotu zpět do *Suma.doCalc()*.

6 Práce s výjimkami

Výjimka představuje objekt pocházející z předka *Exception*. Výjimka "obaluje" chybu. Je vytvořena při vzniku chyby (je-li výjimka vyvolána) a jako taková předána do místa obsluhy, kde je zachycena v části **catch**. Objekt výjimky pak popisuje chybu, která vznikla, její typ, případně i další informace důležité pro její obsluhu.

Pro názornost: v případě, že např. programátor provede dělení nulou, je systémem interně vyvolána předdefinovaná systémová výjimka *DivideByZeroException* která by se měla obsloužit. Obsluhou výjimky v jakékoli formě se zabrání pádu programu a zobrazení implicitního, často konečnému uživateli nesrozumitelného chybového hlášení.

```
try
{
    i = i / i1; }
catch ( Exception el )
{
    MessageBox.Show( "Nastala výjimka - chyba * + el.StackTrace );
```

Proměnné *el* budou předány bližší informace o vzniklé výjimce - např. originální chybový text, místo jejího vzniku. Tento příklad řetězcově vypisuje náhled na zásobník v okamžiku vzniku výjimky.

Použití výjimky typu **Exception** je možné, měla by však existovat snaha použít co nejspecializovanější typ, abychom už samotným typem výjimky co nejpřesněji popsali vzniklý problém (např. **ArithmeticException**).

Pozn.: Tato obsluha přes **catch** (typ_vyjimky) je univerzální, zachytí všechny výjimky vzniklé v *.NET* prostředí. Blok **catch** bez specifikace typu výjimky slouží k zachycení výjimek, které vzniknou i mimo prostředí *.NET*- např. v nezabezpečení kódů v C++.

6.1.4 Vyvolání výjimky

```
int i = 10; int i1 = 5;
try { if (i1==5) throw new Exception( "Pětkou nelze dělit." );
catch ( Exception e1 ) {
    MessageBox.Show ( "Chyba - " - e1.Message );
}
```

V případě, že dělitel výrazu je roven pěti, vyvolá výjimku s příslušným hlášením.
Pozn.: Příklad je nesmyslný, lze jej vyřešit i podmínkou, způsob vyvolání výjimky však ukazuje názorně.

7 Práce s konzolí - konzolová aplikace

používá se pro jednodušší aplikace s textovým výstupem v „dosovém“ okně.

Při vytváření projektu konzolové aplikace se postupuje přes "New Project..." -> „ Console Application”.

7.1.1 Výstup textu na konzoli

```
static void Main(string[] args)
{
    Console.WriteLine ( "Výstup na konzoli ...");
    Console. Read ();
}
```

Vypíše na konzoli jeden řádek a čeká na stisk <Enter>.

7.1.2 Výstup na konzoli do čtyř řádků přes jeden příkaz

```
{
ConsoleWrite ("Vystup\nna\nkonzoli\ndo vice radku")
ConsoleRead();
}
```

Vypíše na konzoli pod sebe čtyři řádky a čeká na stisk <Enter>.



7.1.3 Formátovaný výstup na konzoli

```
double d = 55.555;  
Cconsole.WriteLine( "Hodnota je {0:N2} procenc.", d );
```

Lze používat stejné masky, které jsou přístupné u metody **String.Format()**.

Tento příklad vypíše „*Hodnota je 55.56 procent*“.

Pozn.: Příklady na hlubší použití formátovacích masek lze nalézt v kapitole „*Práce s datovými typy*“ - „*Práce s řetězci*“.

7.1.4 Určení výstupního proudu (pro XmlTextWriter)

```
System.Xml.XmlTextWriter writer = new System.Xml.XmlTextWriter( Console.Out );  
writer. WriteStart.Document ( );  
writer. WriteStartElement ( "Text" );  
.....  
writer. WriteEndDocument ( ) ;  
writer. Closes();  
Console. Read();
```

Tímto způsobem lze zajistit, že jednoduchý výstup z instance objektu **XmlTextWriter** bude automaticky směřován na konzoli (**Console.Out**).

7.1.5 Vymazání konzole zvolenou barvou (.NET 2.0)

```
Console.BackgroundColor = ConsoleColor.Gray;  
Console.Clear();  
Console.ForegroundColor = ConsoleColor.Yellow;  
Console.WriteLine( "Vystup na šedé barvě Zlutým písmem" );
```

.NET 1.1 nepodporoval vymazání konzole - za tímto účelem musely být celkem složitě volány funkce jádra *Windows API*. Nyní je tato možnost přímo podporována objektem **Console**.

7.1.6 Ukončení přes <Ctrl+C> - odchytení klávesy na konzoli (.NET 2.0)

```
static void Main (string[] args)  
{  
    Console.CancelKeyPress += new  
        ConsoleCancelEventHandler (Console.CancelKeyPress);  
    Console.WriteLine( "Čekání na ukončení přes CTRL-C );  
    while (true) { };  
}  
  
static void Console.CancelKeyPress(object sender, ConsoleCancelEventArgs e)  
    Console.WriteLine( "Stisk klávesy/CTRL+C" );
```

Konzolová aplikace čeká tak dlouho, dokud není vyvolána událost reakce na **Console.CancelKeyPress** - ta totiž představuje stisk kombinace kláves <Ctrl+C>.

8 Práce s komponentami z palety nástrojů

Paleta nástrojů (tzv. *toolbox*) se implicitně (v závislosti na zvoleném typu vytvářené aplikace) objevuje na levé straně vývojářského prostředí. Objekty, které nabízí, lze myší aktivovat a umístit přímo na formulář - programátor není tedy nucen k tomu, aby každý prvek vytvářel dynamicky (programově).

Komponenty lze rozdělit na:

- vizuální - po spuštění aplikace jsou vidět přímo na formuláři (**TextBox, ListBox, TreeView**)
- nevizuální - mají určitou skrytou činnost, jejich umístění na formuláři je viditelné pouze ve vývojovém stádiu, v běžícím programu však vidět nejsou (**Timer, EventLog**)



8.1 Společné pro všechny vizuální komponenty

Třída Control je společným předkem pro všechny vizuální komponenty. Proto je možné jim poskytované společné rysy využít u všech těchto zděděných prvků (tříd).

Pro demonstraci bude použit objekt TextBox, pamatujte si však, že stejně tak je možné aplikovat tyto příkazy např. nad objektem ListView. Panel atd.

8.1.1 Změna pozice prvku přes Control.Left a Control.Top

```
textBox1.Left = 10; textBox1.Top = 10;
```

Prvek bude přesunut na pozici [10,10].

8.1.2 Změna pozice prvku přes Control.Location

```
textBox1.Location = new Point( 10, 10 );
```

Další varianta pro změnu pozice. Prvek bude přesunut na pozici [10, 10].

8.1.3 Změna délky prvku přes Control.Width

```
textBox1.Width = 100;
```

Délka prvku bude nastavena na 100 pixelů.

8.1.4 Změna délky prvku přes Control.Size

```
textBox1.Size = new Size( 1000, textBox1.Height );
```

Alternativa pro změnu rozměrů - zde nastavuje délku na 100, výšku ponechává původní.

8.1.5 Změna barvy písma a změna barvy prvku pozadí prvku

```
textBox1.ForeColor = Color.Red;
```

```
textBox1.BackColor = Color.Yellow;
```

Nastavuje barvu popředí (barvu textu) na červenou a barvu pozadí na žlutou.

8.1.6 Změna barvy rodičovského objektu

```
if ( textBox1.Parent != null ) textBox1.Parent.BackColor = Color.Red;
```

Zde je vidět přístup k rodičovskému objektu, na kterém je prvek *textBox1* umístěn a změna barvy pozadí tohoto rodiče.

8.1.7 Skrytí objektu na formuláři

Po změně vlastnosti `Control.Visible` je vyvolána událost `Control.VisibleChanged`. Tu lze přepsat, např. následujícím způsobem:

```
Private void textBox1VisibleChanged(object sender, System.EventArgs e)
```

```
MessageBox.Show ("Visible=" + c.Visible.ToString());
```

Jako praktický význam lze uvést např. automatické zobrazení jiného pole po změně viditelnosti, apod.

8.1.8 Zákaz práce s objektem - objekt je viditelný, zákaz je graficky rozlišen

```
textBox1.Enabled = false;
```

Po změně vlastnosti `Control.Enabled` je vyvolána událost `Control.EnabledChanged`. Tu lze přepsat, např. následujícím způsobem:

```
private void textBox1_EnabledChanged(object sender, System.EventArgs e)
```

```
{
```

```
Control c = (Control)sender;
```

```
MessageBox.Show( "Enabled=" + c.Enabled.ToString() );
```

```
}
```

8.1.9 Reakce na vstup do prvku (podporuje-li ji, např. 1 textové pole)

```
private void textBox1_Enter(object sender, System.EventArgs e)
```

```
{
```

```
label1.Text = 'Vstup do prvku';
```

```
}
```

Obsluha této události umožňuje při vstupu do prvku kurzorem provést inicializační akce.

Pozn.: Vstup do vizuálního prvku na formuláři (klávesou, myší či programově) označujeme jako získání fokusu či zaostření. Obvykle je prvek i určitým způsobem graficky odlišen - např. širším rámečkem nebo zvýrazněným textem. Naopak, pokud se prvek stane neaktivním, ztrácí fokus, zaostření.

8.1.10 Reakce na výstup z prvku (podporuje-li ji, např. textové pole)

```
private void textBox1_Leave(object sender, System.EventArgs e)
{
    label1.Text = "výstup z prvku";
}
```

Událost je volána při opuštění prvku.

8.1.11 Reakce na získání zaostření prvku (fokusu)

```
private void textBox1_GotFocus(object sender, EventArgs e)
{
    label1.Text = "Focus prvku";
}
```

Reakce je volána po zaostření prvku. Tato událost je vyvolána až po události Control.Enter.

8.1.12 Reakce na ztrátu zaostření prvku (fokusu)

```
private void textBox1_LostFocus(object sender, EventArgs e)
{
    label1.Text = "Bez focusu";
}
```

Reakce je volána po ztrátě zaostření. Tato událost je vyvolána ještě před události Control.Leave.

8.1.13 Získání velikosti prvku jako typ Rectangle

```
Rectangle r = button1.ClientRectangle;
```

Tato metoda je velmi využívána zejména ve vykreslování grafiky.

8.1.14 Control.SetStyle a přepsání notifikační metody Control.OnNotifyMessage()

Je-li nad prvkem formuláře nastaven atribut ControlStyles.EnableNotifyMessage, je pro každou zprávu, která je obsluhována v proceduře okna (WndProc), volána notifikační metoda OnNotifyMessage. Tuto metodu lze překrýt a upravit chování pro specifickou zprávu.

```
private void button1_Click(object sender, EventArgs e)
{
    this.SetStyle( ControlStyles.EnableNotifyMessage, true );
}
protected override void OnNotifyMessage(Message m)
{
    base.OnNotifyMessage( m);
    if ( m.Msg == '\x201' )
        listBox1.Items.Add( m.ToString()); }
}
```

Tento kód nejprve po stisku tlačítka aktivuje notifikační metodu (ta je implicitně neaktivní). Metoda je přepsána a mimo původní význam testuje, zda se vyskytla zpráva WM_LBUTTONDOWN (stisk levého tlačítka myši) a pokud ano vypíše informaci o této zprávě do ListBoxu.

Pozn.: Control.SetStyle() se používá zejména k úpravě volání způsobu vykreslování přes metodu Paint() + použití vykreslovacích bufferů.

8.1.15 Změna kurzoru

```
this.Cursor = Cursors.WaitCursor; button1.Cursor = Cursors.Default;
```

Nastaví kurzor hodin nad celým formulářem (mimo tlačítko *button1*, kde ponechá kurzor implicitní).

8.1.16 Načtení kurzoru ze souboru

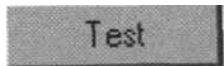
```
Cursor myCursor = new Cursor(@"c:\lano.cur");  
if ( myCursor != null ) this.Cursor = myCursor;
```

Z externího souboru *c:\lano.cur* načítá kurzor a aktivuje ho.

8.2 Button

Klasické tlačítko.

8.2.1 Změna textu na tlačítku



8.2.2 Změna textu i obrázku na tlačítku s definicí jeho zarovnání

```
button1.Text = "Text";  
button1.TextAlign = ContentAlignment.TopCenter;  
button1.Image = Image.FromFile( @"c:\calc.bmp" ); button1.ImageAlign = ContentAlignment.BottomCenter;
```

Určuje se zarovnání textu i zarovnání obrázku (který byl načten ze souboru).

8.2.3 Změna barvy tlačítka po najetí myši a změna na Flat styl (.NET 2.0)

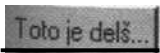
```
button1.FlatAppearance.MouseOverBackColor; button1.FlatStyle = FlatStyle.Flat;
```

Přes *Button.FlatAppearance* lze nastavit způsob vykreslení tlačítka, zde je jako styl použit Flat (ploché).

8.2.4 Zkrácení dlouhého textu tlačítka a doplnění znaky "...".(.NET 2.0)

```
button1.AutoElipsis = true;  
button1.Text = "Toto je delší tex";
```

Delší text, který by jinak přesahoval, je zkrácen a ukončen znaky „...“.



8.3 ComboBox

Seznam hodnot, v jeden okamžik může být viditelná jedna hodnota.

8.3.1 Naplnění hodnot do ComboBoxu s aktivací první hodnoty

```
comboBox1.Items.Clear();  
comboBox1.Items.Add("Hodnota 1");
```