Automatická digitalizace plošného grafu křivky z obrázku

2D curve automatic digitizing from figure

Bc. Michal Heczko

Diplomová práce
2008

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav aplikované informatiky
akademický rok: 2007/2008

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Michal HECZKO**
Studijní program:	**N 3902 Inženýrská informatika**
Studijní obor:	**Informační technologie**
Téma práce:	**Automatická digitalizace plošného grafu křivky z obrázku**

Zásady pro vypracování:

1. **Proveďte literární průzkum o technologii MS .NET Framework a o vývoji aplikací ve vývojovém prostředí MS Visual Studio.**
2. **Shrňte teoretické poznatky o programovacím jazyku C/sharp.**
3. **Seznamte se s danou problematikou digitalizace, proveďte literární průzkum, a dále průzkum zda existuje nějaký software, který se danou problematikou zabývá a jakým způsobem je realizován.**
4. **Za pomocí vývojového prostředí MS Visual Studio vytvořte program pro automatickou digitalizaci plošného grafu jediné křivky z obrázku, tj. převod obrázku ve formě rastru na proměnné ve formě vektorů hodnot x-ové a y-ové osy s možností jejich vykreslení. V práci popište řešení.**
5. **Tento vámi vytvořený program umístěte na elektronické médium, např. CD-ROM, jako přílohu vaší práce.**

Rozsah práce:

Rozsah příloh:

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

Seznam odborné literatury:

1. **KAČMÁR Dalibor. Programujeme .NET aplikace ve Visual Studiu .NET. Brno: Computer Press, a.s. 2001. 344 s. ISBN 80-722-6569-5.**
2. **ELLER Frank. C/sharp – začínáme programovat. Praha: Grada Publishing, a.s. 2002. 240 s. ISBN 80-247-0324-6.**
3. **NAGEL Christian, EVJEN Bill, GLYNN Jay, SKINNER Morgan, WATSON Karli, JONES Allen. C/sharp 2005 – programujeme profesionálně. Brno: Computer Press, a.s. 2006. 1400 s. ISBN 80-251-1181-4.**
4. **SELLS Chris. C/sharp a WinForms. Brno: Zoner Press 2005. 645 s. ISBN 80-86815-25-0.**
5. **BAYER Jürgen. C/sharp 2005 : velká kniha řešení. Brno: Computer Press, a.s. 2007. 816 s. ISBN 978-80-251-1620-3.**

Vedoucí diplomové práce: **Ing. Karel Perůtka, Ph.D.**
Ústav řízení procesů

Datum zadání diplomové práce: **20. února 2008**

Termín odevzdání diplomové práce: **19. května 2008**

Ve Zlíně dne 20. února 2008

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
L.S.

prof. Ing. Vladimír Vašek, CSc.
děkan

doc. Ing. Ivan Zelinka, Ph.D.
ředitel ústavu

ABSTRAKT

Cílem této diplomové práce je vytvoření aplikace, která bude sloužit pro automatickou digitalizaci plošného grafu křivky z obrázků.

První část je věnována popisu vývoje aplikací pro operační systémy MS Windows, zejména popisu technologie MS .NET Framework a programovacího jazyka C#.

Ve druhé části jsou shrnuty poznatky o existujících řešeních v dané oblasti a je zde popsán vytvořený program včetně popisu použitých algoritmů.

Klíčová slova: C#, digitalizace, GDI+, MS .NET Framework, MS Visual Studio, plošné grafy

ABSTRACT

The goal of this master thesis is to make an application, which will be serving to 2D curve automatic digitizing from figure.

In the first part, there is described application development for MS Windows operation systems, especially there is described MS .NET Framework technology and C# programming language.

In the second part there are summarized basic knowledge of the existing solutions in this area and there is described the application, which was made within this master thesis, including description of applied algorithms.

Keywords: C#, digitizing, GDI+, MS .NET Framework, MS Visual Studio, 2D curve

Děkuji vedoucímu diplomové práce panu Ing. Karlu Perůtkovi Ph.D. za odborné vedení, rady a připomínky, které mi poskytoval při řešení diplomové práce.

Prohlašuji, že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků, je-li to uvolněno na základě licenční smlouvy, budu uveden jako spoluautor.

Ve Zlíně dne 19. května 2008

…….
Bc. Michal Heczko

OBSAH

ÚVOD

Úkolem této diplomové práce je vytvoření programu pro automatickou digitalizaci plošného grafu křivky z obrázku. Program by měl umožnit načtení obrazových dat ze souboru nebo ze scanneru a následně je digitalizovat. Na výstupu by měl být k dispozici vektor souřadnic s možností vykreslení grafu v počítači.

Pro vytvoření programu byla zvolena technologie .NET Framework a programovací jazyk C#. Z této volby vyplynul i výběr vývojového prostředí. Zvoleným vývojovým prostředím je MS Visual Studio v poslední verzi, tj. ve verzi 2008.

Vzhledem k výběru použité technologie je první část této práce věnována popisu technologie .NET Framework a základům jazyka C#. Tato část je rozdělena do tří kapitol:

- Technologie .NET Framework
- Programovací jazyk C#
- Vývojová prostředí

Kapitola o programovacím jazyku C# čtenáři poskytne základní informace o syntaxi tohoto jazyka včetně krátkých ukázek. Tato kapitola se dělí dle jednotlivých oblastí na:

- Struktura programu
- Datové typy
- Modifikátory
- Třídy a objekty
- Pole, struktury a výčtové typy
- Operátory
- Řízení toku programu
- Výjimky a zpracování chyb
- Vybrané metody MS .NET Framework
- Windows Forms
- Práce s grafikou

A co lze očekávat od praktické části? Vzhledem k tomu, že už existují některá řešení pro digitalizaci grafů, je také těmto programům věnována samostatná kapitola. Bohužel však nebylo nalezeno existující řešení, které by provádělo digitalizaci zcela automaticky.

Následně již je popisováno vlastní řešení, které bylo vytvářeno v rámci této diplomové práce. Jedna kapitola je věnována popisu jednotlivých aplikací a knihoven, které jsou

výsledkem této práce, a na závěr jsou uvedeny některé pokročilejší techniky a algoritmy, které byly použity při vytváření programu.

I. TEORETICKÁ ČÁST

1 VÝVOJ APLIKACÍ PRO OPERAČNÍ SYSTÉM MS WINDOWS

Od vzniku prvních počítačů prošlo programování dlouhým vývojem. Rozsáhlý popis historie vývoje aplikaci však není záměrem této diplomové práce, proto tato kapitola pouze stručně shrnuje historii vývoje aplikací pro 32bitové operační systémy MS Windows.

Prvním rozhraním pro vývoj aplikací pro 32bitové operační systémy MS Windows bylo **Win32API**. Toto rozhraní se začalo hojně využívat v roce 1993 s příchodem operačního systému MS Windows NT. Win32API ještě nemělo žádný objektový základ. Jednalo se pouze o soubor funkcí a datových struktur jazyka C. Výhodou je přímý přístup k funkcím operačního systému, avšak je zde i velká nevýhoda, a to náročnost vývoje aplikací. Win32API je možné využívat i v dnešních verzích operačního systému Windows jak samostatně, tak v kombinaci s jinými technologiemi.

Krátce poté se objevila další technologie, která měla za úkol Win32API doplnit o objektové prvky. Tato technologie je označena jako **MFC** (Microsoft Foundation Classes). Jednalo se o knihovnu založenou na objektové formě jazyka C++. Třídy obsažené v této knihovně obsahovaly nejen základní objekty uživatelského rozhraní, ale i řadu tříd zajišťujících funkcionalitu programu. Stejně jako Win32API i MFC prochází neustále vývojem a lze ji využívat v dnešní době. Aktuální verze vývojového prostředí MS Visual Studio[1] přináší již devátou verzi MFC (knihovna *mfc90.dll*).

Určitou revoluci ve vývoji aplikací pro operační systémy Windows přineslo v roce 2002 rozhraní **.NET Framework**. Jedná se již o plně objektový model a přináší velké zjednodušení ve vývoji aplikací pro tento operační systém. Nevýhodou však může být menší rychlost běhu programů, protože se jedná o samostatné softwarové rozhraní, které běží nad operačním systémem. Této technologii jsou věnovány následující kapitoly.

Pro vývoj aplikací pro Windows se samozřejmě nemusí používat pouze technologie společnosti Microsoft. Existuje celá řada dalších technologií, z nichž některé jsou i multiplatformní (aplikace lze přeložit i pro jiné operační systémy, např. GNU/LINUX).

Za zmínku stojí například softwarová knihovna **wxWidgets**[2], která programátorovi nabízí vývoj platformě nezávislých aplikací. Knihovna je šířena zdarma stejně jako i některé

[1] MS Visual Studio 2008, které bylo vydáno 19. 11. 2007
[2] Ke stažení na< http://www.wxwidgets.org/ >

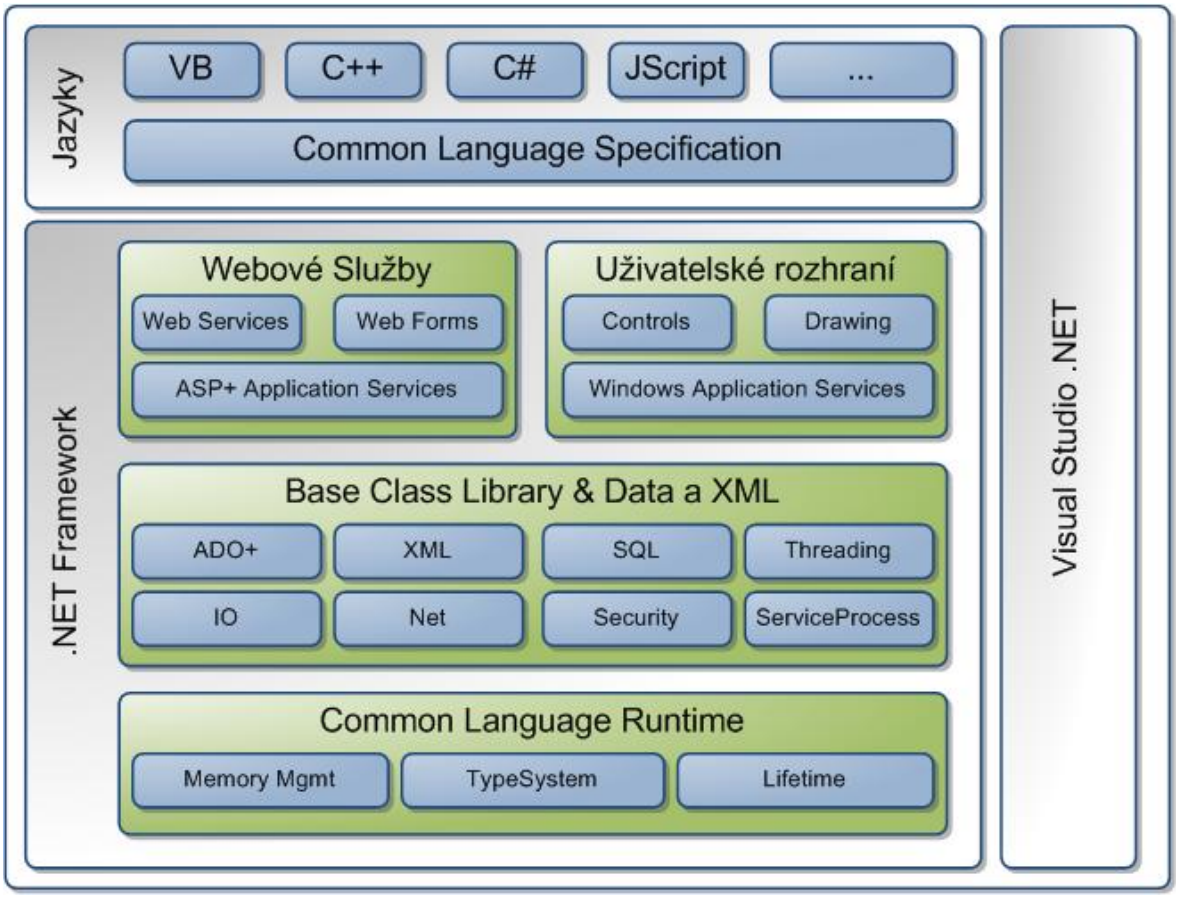
nástroje pro vývoj ve wxWidgets. K dispozici je zde celá řada tříd v jazyce C++. Této problematice se více věnuje [2].

2 TECHNOLOGIE MS .NET FRAMEWORK

2.1 Architektura .NET Framework

.NET Framework výrazně zjednodušuje vývoj aplikací pro operační systém MS Windows a je často přirovnáván k virtuálnímu stroji a jazyku Java od společnosti SUN. Toto přirovnání je zde určitě na místě, protože obě technologie mají velmi podobnou základní filosofii. Aplikace totiž běží na určitém virtuálním stroji, kde jsou za běhu překládány pro danou hardwarovou platformu.

První verze platformy .NET byla společností Microsoft uvedena v roce 2002. V současné době .NET Framework existuje již ve verzi 3.5.



Obr. 1. Struktura .NET Framework a navazující jazyky s vývojovým prostředím [6]

Technologie .NET (obr. 1) se skládá ze tří základních částí:

- Programovacích jazyk
- Vlastního jádro .NET Framework
- Vývojového prostředí Visual Studio .NET

Pro vytváření aplikací má programátor k dispozici celou řadu programovacích jazyků. Použít lze jak základní programovací jazyky pro .NET Framework (Visual Basic, C++, C#, JScript), tak i celou řadu programovacích jazyků třetích stran (např. IronPython, což je implementace jazyka Python v prostředí .NET).

Všechny jazyky jsou definovány nad společnou vrstvou nazvanou *Common Language Specification (CLS)*. CLS specifikuje základní pravidla pro programovací jazyky, ve kterých se vytvářejí aplikace založené na technologii .NET. V této specifikaci jsou definovány mimo jiné základní datové typy nebo třídy. Tím je docíleno, že například datový typ *Byte* definuje osmibitové číslo ve všech jazycích.

Jádro .NET Framework se skládá z několika částí:

- Webové služby
- Uživatelské rozhraní
- Base Class Library & Data a XML
- Common Language Runtime

První blok, tedy webové služby, obsahuje třídy a služby pro vývoj WWW stránek založených na technologii ASP.NET.

Uživatelské rozhraní definuje ovládací prvky, třídy pro kreslení a další služby vztahující se přímo k aplikacím pro operační systém MS Windows.

Base Class Library & Data a XML obsahuje definici základních tříd .NET Frameworku a tříd pro práci s daty (správa databází, XML, …).

Poslední část jádra .NET Frameworku tvoří Common Language Runtime (zkráceně CLR). Jedná se o běhové prostředí a sadu knihoven, které musí být nainstalovány na každém počítači, na kterém má být spuštěna aplikace založená na technologii .NET. Překlad kódu je blíže popsán v následující kapitole.

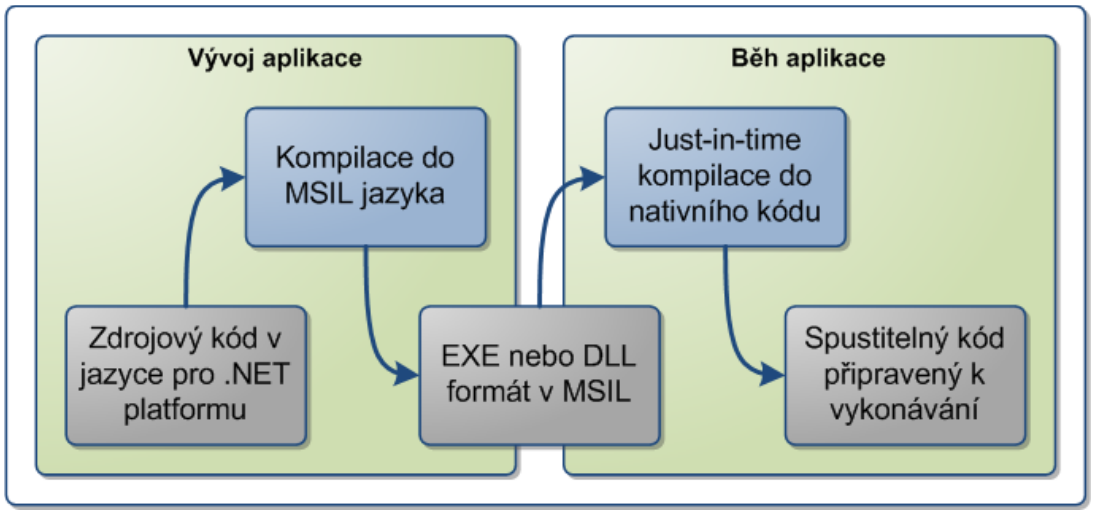
Poslední částí technologie .NET je vlastní vývojové prostředí MS Visual Studio .NET, tomu je však věnována samostatná kapitola (kapitola 4.1).

2.2 Překlad zdrojového kódu a MSIL

Všechny programy, které jsou napsané v libovolném programovacím jazyce z rodiny .NET, kompilátor přeloží do jednotného jazyka – **Microsoft Intermediate Language** (MSIL). MSIL je procesorově nezávislý kód podobný assembleru. Takto přeložený

program je přenositelný mezi různými hardwarovými platformami a až při spuštění na dané platformě je program přeložen tzv. Just-in-time kompilátorem do nativního kódu. K dispozici jsou 3 různé druhy Just-in-time překladačů [6]:

- **Překlad v době instalace** – v tomto případě se nejedná o skutečný Just-in-time překlad. Překlad je prováděn už v době instalace, čímž je odstraněno zpoždění, které se vyskytuje u následujících dvou metod. Nevýhodou je omezení přenositelnosti již nainstalované aplikace.
- **Just-in-time překladač** – kód MSIL je kompletně přeložen při každém spuštění aplikace. Mezi výhody lze zařadit optimalizaci kódu při překladu komponenty. Nevýhodou je zpomalení zavedení aplikace do operační paměti. Princip tohoto překladu je zobrazen na obr. 2.
- **Ekonomický Just-in-time překladač** – Jedná se o podobný princip překladu, jako v předešlém případě, avšak se dvěma rozdíly. Prvním rozdílem je vypnutí všech optimalizačních algoritmů. Druhý rozdíl spočívá v částečném překladu programu – jsou přeloženy pouze ty funkce, které jsou právě třeba pro běh programu.



Obr. 2. Překlad zdrojového kódu a jeho spuštění [6]

2.3 Historie verzí .NET Framework

První verzí platformy .NET Framework byla verze 1.0, která byla uvedena v lednu roku 2002. O rok později následovala verze 1.1, která přinesla řadu změn. Mimo jiné podporu IPv6 (Internet Protocol version 6) nebo .NET Compact Framework určené pro mobilní zařízení.

V roce 2005 byla uvedena na trh druhá hlavní verze .NET Framework (označená jako 2.0). Tato verze není zpětně kompatibilní s verzí 1.0 a 1.1. Největší změnou je podpora 64bitových procesorů. Došlo také k velkému množství změn v API a v ovládacích prvcích pro ASP.NET.

Významnějších změn doznala technologie **.NET Framework** ve verzi **3.0** (někdy označována také jako WinFX), která byla uvedena s příchodem operačního systému Windows Vista. Tato verze přinesla zcela nový pohled na vývoj aplikací (obr. 3). Základním stavebním kamenem jsou služby operačního systému (Base Operating System Services), nad kterým jsou vybudovány 4 základní pilíře technologie .NET:

- Windows Presentation Foundation
- Windows Communication Foundation
- Windows Workflow Foundation
- Windows Cardspace

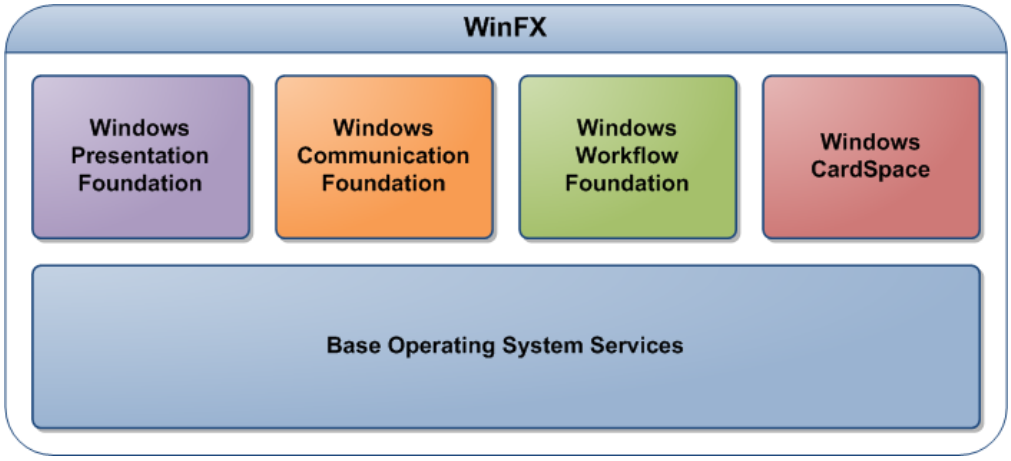
Windows Presentation Foundation (WPF) je knihovna pro práci s uživatelským rozhraním. Jejím úkolem je oddělení návrhu vzhledu aplikace od samotného programového kódu. WPF využívá jazyk XAML a veškeré prvky uživatelského rozhraní jsou definovány vektorově.

Knihovna Windows Communication Foundation (WCF) umožňuje komunikaci mezi jednotlivými aplikacemi. Uplatnění nachází zejména v oblasti webových služeb.

Windows WorkFlow Foundation obsahuje nástroje pro definování průběhu práce. Umožňuje jednoduše vytvářet schémata procesů jak v tradičních programovacích jazycích, tak i v deklarativním jazyku XAML.

Posledním pilířem platformy .NET 3.0 je Windows CardSpace. Tato knihovna nabízí jednotný přístup pro ověřování identity uživatele.

Ostatní vlastnosti a také většina knihoven vycházejí z verze 2.0.



Obr. 3. Struktura WinFX

Zatím poslední verzí technologie .NET je **MS .NET Framework 3.5**. Největšími novinkami této verze je dotazovací jazyk LINQ a programovací jazyk C# ve verzi 3.0.

Na závěr této kapitoly je vhodné uvést přehled všech zmíněných verzí doplněný o informace o kompatibilitě s operačními systémy a o vývojových prostředích, ve kterých lze vyvíjet aplikace v dané verzi MS .NET Framework (tab. 1).

Tab. 1. Přehled verzí MS .NET Framework

Verze	Datum uvedení	Operační systém	Vývojová prostředí
1.0	5. 1. 2002	Windows 98 Windows ME Windows NT 4.0 Windows 2000 Windows XP	MS Visual Studio .NET (2002)
1.1	1. 4. 2003	Windows 98 Windows ME Windows NT 4.0 Windows 2000 Windows XP Windows Server 2003	MS Visual Studio .NET 2003
2.0	7. 11. 2005	Windows 98 Windows ME Windows NT 4.0 Windows 2000 Windows XP Windows Server 2003	MS Visual Studio 2005 MS Visual Studio 2008
3.0	6. 11. 2006	Windows XP SP2 Windows Server 2003 SP1 Windows Vista Windows Server 2008	MS Visual Studio 2005 MS Visual Studio 2008
3.5	19. 11. 2007	Windows XP SP2 Windows Server 2003 SP1 Windows Vista Windows Server 2008	MS Visual Studio 2008

3 PROGRAMOVACÍ JAZYK C#

Jak už bylo zmíněno v předchozích kapitolách, jedním z hlavních programovacích jazyků v technologii MS .NET Framework je jazyk C# („C sharp“). Tento programovací jazyk byl nově vyvinut pro prostředí MS .NET Framework a vychází z jazyků C++ a Java.

Nyní se krátce podívejme na základní vlastnosti jazyka C# (většina vlastností vychází z vlastností platformy .NET): [3], [6] a [19]

- Jedná se o čistě **objektově orientovaný** jazyk.
- Je **case-sensitive**, tj. rozlišuje malá a velká písmena (například *promenna* a *Promenna* jsou 2 rozdílné pojmy).
- Obsahuje nativní podporu komponentového programování.
- Používá jednoduchou dědičnost s možností násobné implementace rozhraní.
- Vedle členských dat a metod využívá vlastnosti a události.
- Využívá **Garbage collection** (jedná se o automatickou funkci platformy .NET, která zajišťuje automatické uvolňování paměti).
- Podporuje zpracování chyb formou výjimek - **exceptions** (dojde-li k výskytu chyby, je vytvořen příslušný objekt výjimky a chyba zobrazena v okně).
- Zajišťuje typovou bezpečnost a správu verzí.
- Podporuje atributové programování.
- Zajišťuje hlubokou integraci se stávajícím kódem na binární i zdrojové úrovni.

3.1 Struktura programu

Bývá zvykem, že v úvodu popisu programovacího jazyka bývá uvedena ukázka programu „Hello world“, a tato kapitola nebude výjimkou. Na tomto oblíbeném programu bude totiž ukázána základní struktura programu.

Pozn.: Pro většinu ukázek v této i v následujících kapitolách budou použity konzolové aplikace.

```
1 using System;
2
3  namespace helloworld
4  {
5        class Program
6        {
7              static void Main(string[] args)
8              {
```

```
 9                     Console.WriteLine("Hello world!");
10               }
11          }
12    }
```

Nyní je vhodné popsat výše uvedený kód aplikace „Hello world“. Jedinou funkcí tohoto programu je výpis textu „Hello world!“ do okna konzole.

Na prvním řádku je umístěn příkaz ***using***, který importuje jmenný prostor *System*, který obsahuje mimo jiné objekt *Console* (tento objekt je použit dále v tomto programu).

Kód této aplikace je umístěn ve jmenném prostoru helloworld, který uvozuje příkaz ***namespace*** na třetím řádku programu. O řádek níže je definována třída *Program*. Tato třída je zároveň jedinou třídou tohoto jmenného prostoru. Pro definici třídy slouží klíčové slovo ***class***.

Každá aplikace v jazyce C# musí mít metodu **Main** (sedmý řádek ukázkového programu). Do této metody jako parametr vstupuje pole řetězců (*string[] args*) a jedná se o funkci bez návratové hodnoty (klíčové slovo ***void*** před názvem funkce). Funkce Main v tomto případě obsahuje pouze jeden příkaz, a to výpis textu do konzole (*Console.WriteLine("Hello world!");*).

3.1.1 Komentáře

Komentáře slouží pro vepisování poznámek do programu programátorem, nijak neovlivňují chod programu ani jeho rychlost.

V C# lze používat 2 základní typy komentářů, a to jednořádkové a víceřádkové. Jednořádkový komentář se uvozuje dvěma lomítky. Pro víceřádkový komentář slouží znaky /* na začátku komentáře a znaky ***/** na konci komentáře.

Použití komentářů je ukázáno na níže uvedeném příkladě:

```
1 // Konzolová aplikace HELLO WORLD
2 using System;
3
4  namespace helloworld
5  {
6         class Program
7         {
8               static void Main(string[] args)
9               {
```

```
10                      /*
11                      Výpis textu do okna konzole
12                      Text: „Hello world!"
13                      */
14                      Console.WriteLine("Hello world!");
15              }
16          }
17  }
```

3.2 Datové typy

Datové typy v jazyce C# lze rozdělit do dvou základních skupin:

- Hodnotové
- Referenční

První skupinou datových typů jsou **hodnotové typy** (value types). Do této skupiny patří všechny číselné datové typy, typ char a ostatní struktury. U těchto jednoduchých typů se jejich hodnota ukládá přímo do proměnné – místa v paměti určené pro uložení hodnoty.

Druhou skupinou jsou **referenční typy** (reference types). Do této skupiny patří typ String a všechny třídy. Na rozdíl od hodnotových typů se jejich hodnota uloží do oblasti paměti nazývané halda. Do proměnné se uloží pouze adresa paměti, kde je hodnota uložena – reference.

Základní datové typy přehledně shrnuje níže uvedená tabulka (tab. 2).

Tab. 2. Datové typy

C# Typ	CTS Typ	Velikost	Rozsah
sbyte	System.SByte	8 b	- 128 až 127
byte	System.Byte	8 b	0 až 255
short	System.Int16	16 b	- 32768 až 32767
ushort	System.UInt16	16 b	0 až 65535
int	System.Int32	32 b	- 2147483648 až 2147483647
uint	System.UInt32	32 b	0 až 4294967295
long	System.Int64	64 b	- 9223372036854775808 až 9223372036854775807
ulong	System.UInt64	64 b	0 až 18446744073709551615
char	System.Char	16 b	jeden 16bitový znak Unicode
float	System.Single	32 b	$\pm 1.5 \times 10^{-45}$ až $\pm 3.4 \times 10^{38}$ (přesnost na 7 desetinných míst)
double	System.Double	64 b	$\pm 5.0 \times 10^{-324}$ až $\pm 1.7 \times 10^{308}$ (přesnost na 15 až 16 desetinných míst)
decimal	System.Decimal	128 b	1.0×10^{-28} až 7.9×10^{28} (přesnost na 28 až 29 desetinných míst)
bool	System.Boolean	1 b	true / false
string	System.String	neomezeně	omezeno pouze velikostí dostupné paměti pro řetězce Unicode

3.3 Modifikátory

Pomocí modifikátorů lze ovlivňovat viditelnost a chování proměnných, konstant, metod a tříd. Seznam modifikátorů i s jejich popisem shrnuje následující tabulka (tab. 3). [3]

Tab. 3. Modifikátory

Modifikátor	Popis
public	přístup i vně příslušné třídy
private	přístup pouze zevnitř třídy (výchozí nastavení pro atributy objektu)
internal	přístup omezen na aktuální objekt
protected	přístup pouze uvnitř třídy nebo z tříd, které jsou od dané třídy odvozeny
abstract	označení tříd, od nichž není možné vytvářet instance (nutno odvodit další třídu
const	označení konstant (hodnoty není možné měnit)
event	deklarace události
extern	označení externí deklarace identifikátoru (např. přístup k metodám deklarovaným v DLL)
override	přepsání již deklarovaných metod při odvozování nové třídy
readonly	vlastnost, jejíž hodnota je mimo danou třídu určena pouze ke čtení
sealed	od takto označené třídy nelze odvozovat další třídy

3.4 Třídy a objekty

Ve výčtu vlastností jazyka C# je zmíněno, že se jedná o silně objektový jazyk. Tato kapitola je proto věnována základním pojmům a principům objektového programování.

3.4.1 Základní pojmy

Třída je abstraktní pojem, je to pouze šablona, která obsahuje atributy a metody objektu. Aby bylo možné objekt používat, je jej nutné definovat, tj. vytvořit instanci třídy.

Jak už bylo zmíněno, každá třída má své atributy a metody. **Atribut** (neboli datová položka objektu) je konstanta nebo proměnná, která je vázaná ve třídě. Atribut uchovává informace o stavu objektu. Každý atribut musí být definován svým datovým typem a názvem. Dalšími členy třídy jsou **metody**. Jedná se o funkce, které slouží k provádění určitých činností v rámci programu. Metody slouží pro komunikaci mezi objekty a pro nastavení nebo získávání informací o stavu objektu. Každá definice metody musí obsahovat:

- Název metody
- Vstupní parametry
- Tělo metody
- Návratovou hodnotu metody

Zvláštní metodou třídy je **konstruktor**. Jedná se o metodu bez návratové hodnoty, která má stejný název jako třída. Konstruktor je volán při vytváření daného objektu. Každá třída musí obsahovat alespoň jeden konstruktor. [4]

Opakem konstruktoru je destruktor. Destruktor je volán vždy při rušení objektu. Destruktor má stejný zápis jako konstruktor s tím rozdílem, že před jeho názvem je tilda (~). [3]

Základní vlastnosti objektově orientovaného přístupu:

- Zapouzdření (encapsulation)
- Dědičnost (inheritance)
- Polymorfismus (polymorphism)

První zmíněnou vlastností je **zapouzdření** (obr. 4). To znamená, že objekt má některé své členy (metody nebo atributy) skryty před svým okolím. Členy přístupné okolí se nazývají rozhraní objektu. [19]

Obr. 4. Zapouzdření

Dědičnost umožňuje odvozování potomků tříd. Potomek přebírá atributy a metody bázové třídy (třídy, ze které je odvozen), navíc může mít i nové vlastní atributy a metody.

Poslední zmiňovanou vlastností je **polymorfismus**, což znamená, že stejně pojmenovaná metoda se stejným počtem parametrů může v různých třídách provádět různé akce. Může se také vyskytovat situace, kdy jsou dvě stejně pojmenované metody s různým počtem parametrů v rámci jedné třídy.

3.4.2 Jmenné prostory

Jmenné prostory mají podobnou funkci jako adresáře – slouží pro logické rozdělení programu. Je to oblast, která obsahuje tematicky uspořádané třídy. Tato oblast není omezena na jeden soubor.

Pro jmenné prostory je vyhrazeno rezervované slovo *namespace*. Pokud je třeba využívat třídy z určitého jmenného prostoru, lze jej do programu implementovat pomocí klíčového slova *using*.

Platforma .NET Framework nabízí celou řadu jmenných prostorů (některé z nich jsou zmíněny v kapitole 3.9), které obsahují předdefinované třídy a datové typy.

3.4.3 Definice třídy

Obecná syntaxe definice třídy má v jazyce C# následující podobu:

```
1  [modifikatory] class jmenotridy [: jmenobazovetridy]
2 {
3         // deklarace atributů:
4         [modifikatory] datovytyp jmenoprvnihoatributu;
```

```
5         [modifikatory] datovytyp jmenodruhehoatributu;
6         …
7
8         // deklarace metod:
9         public jmenotridy ([parametry])
10        {
11              // telo konstruktoru
12        }
13
14        [modifikatory] typvysledku jmenometody ([parametry])
15        {
16              // telo metody
17        }
18
19        …
20
21  }
```

Pozn. Nepovinné parametry jsou uvedeny v hranatých závorkách.

Při vytváření objektu z výše uvedené třídy je nutné postupovat následovně:

```
1   [modifikátory] jmenotridy jmenoobjektu = new
    jmenotridy([parametry])
```

Při provádění výše uvedeného příkazu je vytvořen objekt a volán jeden z jeho konstruktorů (pokud je více konstruktorů, jsou rozlišeny pomocí jejich parametrů).

3.4.4 Příklad použití objektů a tříd

Následující ukázka demonstruje použití třídy, která bude reprezentovat grafický prvek v podobě bitové mapy. [5]

Nejprve je nutné nadefinovat třídu:

```
2  using System;
3  using System.Drawing;
4  using System.Windows.Forms;
5
6  namespace Tridy_priklad
7  {
8   public class BitováMapa
9   {
10        private string CestaKBitmapě;
11        private Bitmap Bitmapa;
```

```
12        public BitováMapa()
13        {
14              CestaKBitmapě = Environment.CurrentDirectory +
15                    @"\Západ slunce.bmp";
16              NačístBitovouMapu(CestaKBitmapě);
17        }
18        public void NačístBitovouMapu(string Cesta)
19        {
20              CestaKBitmapě = Cesta;
21              Bitmapa = new Bitmap(Cesta);
22        }
23        public void ZobrazitBitovouMapu()
24        {
25              Form Formulář = Form.ActiveForm;
26              Graphics GrafickýObjekt = Formulář.CreateGraphics();
27              GrafickýObjekt.DrawImage(Bitmapa, 10, 10, 200, 150);
28              GrafickýObjekt.Dispose();
29        }
30  }
31 }
```

Následně je možné vytvořit instanci této třídy v programu:

```
1 …
2  // Založení nové instance třídy BitováMapa.
3  BitováMapa bm = new BitováMapa();
4  // Zobrazení bitové mapy na ploše formuláře.
5  bm.ZobrazitBitovouMapu();
6 …
```

Pomocí dědičnosti je možné z třídy *BitováMapa* odvodit novou třídu pojmenovanou *BitováMapa2*:

```
1 …
2  public class BitováMapa2 : BitováMapa
3  {
4        private int rozmerX;
5        private int rozmerY;
6        public BitováMapa2():Base()
7        {
8              // implementace dodatečných operací:
9
10       }
```

```
11
12          public void NačístBitovouMapu(string Cesta)
13          {
14                  // Implementace metody z bázové třídy:
15                  base. NačístBitovouMapu(string Cesta);
16
17                  // Implementace nových operací:
18                  rozmerX = Bitmapa.Width;
19                  rozmerY = Bitmapa.Weight;
20          }
21
22          // Implementace nových metod:
23
24          public int ziskejX()
25          {
26                  return rozmerX;
27          }
28
29          public int ziskejY()
30          {
31                  return rozmerY;
32          }
33
34    }
35 …
```

Výsledkem výše uvedeného kódu bude definice třídy *BitováMapa2*, která bude odvozena z bázové třídy *BitováMapa*. Nová třída bude oproti původní obsahovat navíc vlastnosti *rozmerX* a *rozmerY*, metody *ziskejX* a *ziskejY* a patřičné úpravy původních metod.

3.5 Pole, struktury a výčtové typy

3.5.1 Pole

Pole je datová struktura, která obsahuje určitý počet proměnných. Tyto proměnné jsou nazývány prvky pole. Prvky pole jsou indexovány a pomocí těchto indexů je později možné prvky z pole získávat. V jazyce C# musí být všechny prvky pole stejného typu a jsou indexovány od nuly. [19]

V jazyce C# lze vytvářet několik typů polí:

- **jednorozměrná pole** – tato pole si lze představit jako vektor hodnot.
- **vícerozměrná pole** – najdou využití zejména pro uložení hodnot z tabulky.
- **nestejnoměrná pole** – vícerozměrná pole, kde každý řádek má jiný počet sloupců.

V případě jednorozměrného pole lze provést inicializaci následujícím způsobem:

```
1 // inicializace pole:
2 int[] i = new int[3];
3 // následuje naplnění pole:
4 i[0] = 1;
5 i[1] = 3;
6 i[2] = 5;
```

Popřípadě je možné spojit naplnění pole s jeho inicializací:

```
1 // inicializace pole:
2 int[] i = new int[] {1, 3, 5};
```

V případě vícerozměrného pole je situace velmi podobná, což znázorňuje následující příklad:

```
1 // inicializace dvourozměrného pole:
2 int[,] pole2D = new int[2,3];
3 // následuje naplnění pole:
4 Pole2D[0,0] = 4;
5
6 // inicializace trojrozměrného pole:
7 int[,,] pole3D = new int[2,2,2];
8 // následuje naplnění pole:
9 Pole3D[0,0,0] = 4;
```

Nestejnoměrná pole (v originále *jagged arrays*) jsou ve své podstatě pole polí. Jejich nejčastější použití je vytvoření 2D pole, kde každý řádek má jiný počet sloupců. Takovéto pole si lze představit i jako ekvivalent slučování buněk v tabulce (tab. 2).

Tab. 4. Ukázka struktury nestejnoměrného pole

0,0	0,1	0,2	0,3	0,4	0,5
1,0		1,1		1,2	
2,0			2,1		

Nyní je vhodné uvést, jak pomocí nestejnoměrného pole vytvořit strukturu, kterou znázorňuje výše zobrazená tabulka:

```
1 // definice řádků:
2 int[][] pole = new int[3][];
3
4 // počty sloupců v jednotlivých řádcích:
5 pole[0] = new int[6];
6 pole[1] = new int[3];
7 pole[2] = new int[2];
```

3.5.2 Struktury

Struktury jsou velmi podobné třídám. Stejně jako třídy obsahují data i metody a stejně jako třídy mohou mít svůj konstruktor. Jsou zde však 3 rozdíly:

- **Příkaz, kterým se struktura vytváří.**
  Klíčové slovo **class** je zde nahrazeno klíčovým slovem **struct**.
- **Způsob práce s pamětí.**
  Zatímco třídy jsou odkazové typy (při vytvoření nového objektu je rezervována paměť a odkaz na tuto paměť je uložen do proměnné představující objekt), struktury řadíme do skupiny hodnotových typů.
- **Konstruktor a definice výchozích hodnot.**
  Struktury narozdíl od tříd nemohou obsahovat konstruktor bez parametrů, ani nesmí mít definovanou výchozí hodnotu atributů.

Struktury jsou vhodné pro menší a často používané objekty.

Použití struktur je možné demonstrovat na zadání souřadnice bodu na obrazovce. Nejprve definice struktury:

```
1 public struct Point
2  {
3        public int x;
4        public int y;
5
6        public Point(int a, int b)
7        {
8              x = a;
9              y = b;
10       }
```

```
11  }
```

Následuje ukázka použití struktury v programu:

```
1 …
2 Point souradnice;
3 souradnice.x = 2;
4 souradnice.y = 3;
5  …
```

3.5.3 Výčtové typy

Výčtový typ si lze představit jako množinu hodnot stejného datového typu. V jazyce C# se označují rezervovaným slovem *enum*.

Obecná syntaxe výčtového typu vypadá následovně:

```
1 [modifikátor] enum identifikátor [: typ]
2 {
3  Hodnota1, Hodnota2, …, HodnotaN
4 };
```

Pozn. Nepovinné parametry jsou uvedeny v hranatých závorkách.

Příkladem použití výčtového typu mohou být například dny v týdnu:

```
1 public enum Dnyvtydnu
2 {
3  Pondeli, Utery, Streda, Ctvrtek, Patek, Sobota, Nedele
4 };
```

Často uživatel požaduje, aby byly hodnoty číslovány od jiného čísla než od nuly, například u dnů v týdnu, které byly zmíněny výše, je vhodné, aby první hodnota byla 1:

```
1 public enum Dnyvtydnu
2 {
3  Pondeli = 1, Utery, Streda, Ctvrtek, Patek, Sobota, Nedele
4 };
```

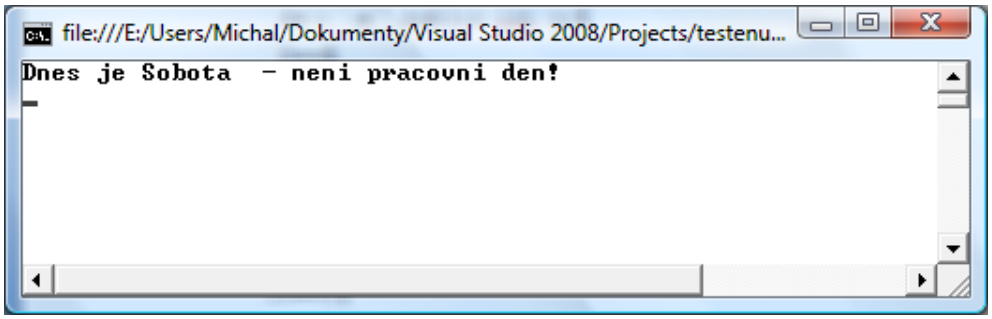
Další možností je přiřazení vlastních hodnot každému prvku výčtového typu, což lze použít kupříkladu u počtu dní v měsíci:

```
1 public enum Mesice
2 {
3  Leden = 31, Unor = 28, Brezen = 31, Duben = 30, Kveten = 31, Cerven
  = 30, Cervenec = 31, Srpen = 31, Zari = 30, Rijen = 31, Listopad =
  30, Prosinec = 31
4 };
```

Při práci s výčty se používá zápis ve tvaru *typ.konstanta*, tedy například pro dny v týdnu:

```
1 Dnyvtydnu dnes;
2 dnes = Dnyvtydnu.Sobota;
3
4 if (dnes == Dnyvtydnu.Sobota || dnes == Dnyvtydnu.Nedele)
5  {
6        Console.WriteLine("Dnes je {0}  - neni pracovni den!",dnes);
7  }
```

Jak je patrné z následujícího obrázku (obr. 5), výsledkem výše uvedené ukázky je výpis textu „Dnes je Sobota – není pracovní den!“.



Obr. 5. Výsledek činnosti programu s ukázkou použití výčtového typu

3.6 Operátory

Operátor je symbolické vyjádření elementární funkce s jednou nebo dvěma vstupními proměnnými, který vrací jednu výstupní hodnotu.

Operátory lze dělit do skupin dle několika aspektů. První možností rozdělení operátorů je dělení dle počtu vstupních hodnot. Lze je tedy rozdělit do následujících skupin:

- **Unární operátory** – mají pouze jednu vstupní hodnotu
- **Binární operátory** – mají 2 vstupní hodnoty
- **Ternární operátory** – mají 3 vstupní hodnoty (jediným ternárním operátorem v jazyce C# je operátor podmínky)

Další možností, jak lze dělit operátory, je rozdělení dle jejich účelu (tento pohled na operátory bude použit i v následujících kapitolách). Při tomto pohledu bude rozdělení operátorů následující:

- **Operátor přiřazení**
- **Matematické operátory**

- **Relační operátory**
- **Logické operátory**
- **Operátor podmínky**

3.6.1 Operátor přiřazení

Nejdůležitějším a zároveň nejpoužívanějším operátorem ve všech programovacích jazycích je operátor přiřazení. Jeho význam je prostý – přiřazuje hodnotu nebo výsledek výpočtu určité proměnné. Pro tento účel se v jazyce C# používá znak „=“.

3.6.2 Matematické operátory

Matematické operátory jsou nejspíše nejznámější skupinou operátorů, definují totiž základní matematické operace – sčítání, odčítání, násobení dělení a modulo (neboli zbytek po dělení). Dále lze do této skupiny zařadit i 2 unární operátory, a to inkrementaci (zvýšení hodnoty o 1) a dekrementaci (snížení hodnoty o 1).

Stručný přehled matematických operátorů shrnuje níže uvedená tabulka (tab. 5).

Tab. 5. Matematické operátory

Operátor	Význam	Příklad použití

+	Součet	`a = 3;`
		`b = 2;`
		`c = a + b;`
		`// výsledek: c = 5`

-	Rozdíl	`a = 3;`
		`b = 2;`
		`c = a - b;`
		`// výsledek: c = 1`

*	Součin	`a = 3;`
		`b = 2;`
		`c = 3 * 2;`
		`// výsledek: c = 6`

/	Podíl	`a = 4;`
		`b = 2;`
		`c = a / b;`
		`// výsledek: c = 2`

%	Modulo (zbytek po dělení)	`a = 3;`
		`b = 2;`
		`c = a % b;`
		`// výsledek: c = 1`

++	Inkrementace	`a = 1;`
		`a++;`
		`// výsledek: a = 2`

--	Dekrementace	`a = 2;`
		`a--;`
		`// výsledek: a = 1`

Pozn. V příkladech použití v následující tabulce se uvažuje, že proměnné *a*, *b* a *c* jsou definovány jako celočíselné proměnné.

K výše uvedeným matematickým operátorům jazyk C# nabízí určitou formu zjednodušení, kterou jsou složené operátory. Jedná se o operátory, které jsou složeny z operátoru přiřazení a daného matematického operátoru. Pomocí jednoho operátoru tak lze provést výpočet a výsledek uložit do proměnné.

Výčet složených operátorů je obsahem následující tabulky (tab. 6).

Tab. 6. Složené operátory

Operátor	Význam
+=	Součet a přiřazení (x+=y odpovídá výrazu x = x + y)
-=	Rozdíl a přiřazení (x-=y odpovídá výrazu x = x - y)
*=	Součin a přiřazení (x*=y odpovídá výrazu x = x * y)
/=	Podíl a přiřazení (x/=y odpovídá výrazu x = x / y)
%=	Modulo (zbytek po dělení) a přiřazení (x+=y odpovídá výrazu x = x % y)

3.6.3 Relační operátory

Relační operátory slouží pro porovnání dvou hodnot. Své uplatnění najdou zejména v oblasti řízení toku programu (viz. kapitola č. 3.7). Návratovou hodnotou je logická hodnota *true* nebo *false*.

Seznam relačních operátorů i s jejich příklady použití lze opět nalézt v přehledné tabulce (tab. 7).

Tab. 7. Relační operátory

Operátor	Význam
==	Rovnost
!=	Nerovnost
<	Menší než
>	Větší než
<=	Menší nebo rovno
>=	Větší nebo rovno

3.6.4 Logické operátory

Logické operátory se používají k provádění logických nebo bitových operací nad hodnotami. Tyto operátory lze ještě dělit do dvou podskupin:

- Spojovací operátory
- Bitové operátory

První skupina, tedy **spojovací operátory**, najde své využití zejména tehdy, když potřebujeme spojit několik výrazů, tj. zejména při stanovování podmínek v oblasti řízení běhu programu. Do této skupiny řadíme pouze 2 operátory:

- Logická konjunkce (význam „a zároveň“ vyjádřený znaky „&&“)
- Logickou disjunkce (význam „nebo“ vyjádřený znaky „||“))

Formu zápisu těchto operátorů a hodnoty, jakých nabývá výsledek, shrnují následující dvě tabulky (tab. 8 a tab. 9).

Tab. 8. Logická konjunkce

Logická konjunkce		
Operátor: &&		
A	**B**	**C = A && B**
True	True	True
True	False	False
False	True	False
False	False	False

Tab. 9. Logická disjunkce

Logická disjunkce		
Operátor: ||		
A	**B**	**C = A || B**
True	True	True
True	False	True
False	True	True
False	False	False

U této skupiny operátorů stojí za zmínku ještě jeden logický operátor, který sice do této skupiny nelze jednoznačně zařadit, ale často se využívá spolu s těmito operátory. Jedná se o operátor negace, pro jehož zápis používáme znak „!“. Funkci tohoto operátoru shrnuje tab. 10.

Tab. 10. Negace

Negace	
Operátor: !	
A	**B = A!**
True	False
False	True

Druhou skupinou logických operátorů jsou **operátory bitové**. Slouží pro provádění bitových operací – provádějí operace bit po bitu na svých operandech. Význam jednotlivých operátorů opět shrnuje tabulka (tab. 11). Bitové operace lze provádět u celočíselných, výčtových a logických datových typů.

Tab. 11. Logické bitové operátory

Operátor	Význam	Příklad
&	bitový AND	`000101 AND 010100` `// Výsledek: 000100`
|	bitový OR	`000101 OR 010100` `// Výsledek: 010101`
^	bitový XOR	`000101 XOR 010100` `// Výsledek: 010001`
>>	bitový posun doprava	`000101 << 2` `// Výsledek: 010100`
<<	bitový posun doleva	`000101 >> 2` `// Výsledek: 000001`

Pozn. Ve sloupci „Příklad“ se v tomto případě nejedná o zápis v jazyce C#, ale pouze o ukázku principu výpočtu.

3.6.5 Operátor podmínky

Operátor podmínky může v určitých případech nahradit příkaz *if* (jedná se o příkaz pro řízení běhu programu, tomuto příkazu se věnuje kapitola č. 3.7.1). Jak už bylo zmíněno v úvodu kapitoly o operátorech, jedná se o jediný ternární operátor v jazyce C#, má tedy 3 vstupní hodnoty.

Podmíněný výraz má tvar `b ? x : y` a postup jeho vyhodnocení vypadá následovně:

1. vyhodnotí se logický výraz *b*
2. pokud je výraz *b* pravdivý, je vyhodnocen výraz *x*, v opačném případě je vyhodnocen výraz *y*
3. hodnota výrazu, který je vyhodnocen, je výsledkem tohoto podmíněného výrazu

Použití podmíněného výrazu nejlépe vystihuje následující příklad:

```
1 int x = 5;
2 int y = 6;
3 int b = (x < y) ? x : y;
```

Po provedení tohoto ukázkového programu bude v proměnné *b* uložena hodnota 5.

3.7 Řízení toku programu

Pro řízení toku programů se obvykle používají 3 základní typy příkazů. Jedná se o podmínky, cykly a skoky.

Podmínky slouží pro větvení programu dle vyhodnocení pravdivosti určitého výrazu. V jazyce C# máme k dispozici celkem 3 příkazy pro konstrukci podmínky:

- IF
- IF-ELSE-IF
- SWITCH

Cykly umožňují opakování určitého bloku příkazů na základě vyhodnocení logického výrazu. Stejně jako u podmínek, i zde je několik možností výběru příkazu. Jsou to tyto čtyři:

- FOR
- WHILE
- DO … WHILE
- FOREACH

Skoky nevyhodnocují žádnou podmínku, ale pevně určují přesun na určitý řádek v rámci programu. Skoky mají jediného zástupce, a to příkaz GOTO.

V následujících několika kapitolách jsou popsány všechny zmíněné příkazy.

3.7.1 Podmínka IF

Příkaz IF uživateli nabízí nejjednodušší možnost tvorby podmínek. Umožňuje pouze větvení na základě vyhodnocení jednoho logického výrazu. Jeho obecná syntaxe je následující:

```
1 if (podmínka)
2  příkaz (příkazy)
3 else
4  příkaz (příkazy)
```

Jak tento příkaz funguje, je nejlépe vidět na následující ukázce, kde se testuje, zda je číslo rovno nule:

```
1 int cislo;
2 bool rovnonule;
3 cislo = 5;
4 if (cislo == 0)
5  {
6        rovnonule = true;
7        Console.WriteLine("cislo je rovno nule!");
```

```
 8  }
 9 else
10  {
11        rovnonule = false;
12        Console.WriteLine("cislo neni rovno nule!");
13  }
```

3.7.2 Podmínka IF-ELSE-IF

Při vytváření programů může dojít k situaci, kdy programátor potřebuje vyhodnotit více stavů určité proměnné. V tomto případě lze použít modifikaci podmínky IF, která umožňuje vícenásobné větvení. Tento příkaz má následující syntaxi:

```
1 if (podmínka 1)
2  příkaz (příkazy)
3 else if (podmínka 2)
4  příkaz (příkazy)
5 …
6 else if (podmínka N)
7  příkaz (příkazy)
8 else
9  příkaz (příkazy)
```

Možnosti použití příkazu opět nejlépe vystihne jednoduchý příklad. V tomto případě dojde k rozšíření příkladu z minulé kapitoly. Opět se bude vyhodnocovat hodnota proměnné, ale budou se rozlišovat 3 stavy: kladné číslo, nula a záporné číslo.

```
 1 int cislo;
 2 cislo = 5;
 3 if (cislo > 0)
 4  {
 5        Console.WriteLine("cislo je kladne!");
 6  }
 7 else if (cislo < 0)
 8  {
 9        Console.WriteLine("cislo je zaporne!");
10  }
11 else
12  {
13        Console.WriteLine("cislo je rovno nule!");
14  }
```

3.7.3 Podmínka SWITCH

Poslední možností použití podmínek je příkaz SWITCH. Tento příkaz vytváří určitou alternativu k příkazu IF-ELSE-IF, který byl popsán v předchozí kapitole. SWITCH však umožňuje vyhodnocovat více stavů určité proměnné.

Nejprve je vhodné zmínit obecnou syntaxi:

```
switch (proměnná)
{
 case hodnota1:
        (příkazy)
        break;
 case hodnota2:
        (příkazy)
        break;
 …
 case hodnotaN:
        (příkazy)
        break;
 default:
        (příkazy)
        break;
}
```

Jak je patrné z předchozího zápisu, příkaz SWITCH pro svou funkci využívá další tři pomocné příkazy. Prvním z nich je příkaz **case**. Tento příkaz uvozuje každou hodnotu, které může nabývat vyhodnocovaná proměnná. Následuje příkaz **break**. Tímto příkazem musí být ukončen každý blok case, jinak dochází k tzv. propadávání – po provedení bloku case se vykonávají příkazy u všech dalších case, dokud program nenarazí na příkaz break. Posledním použitým příkazem je **default**. Tento příkaz uvozuje blok programu, který se vykonává pouze v případě, že daná proměnná nenabývá žádné z hodnot uvedených u příkazů case.

Opět je vhodné uvést jednoduchou ukázku (program vyhodnocuje číslo dne v týdnu a následně vypíše jeho název – v tomto případě „Patek“):

```
int cislodnevtydnu;
string Nazevdnevtydnu;
cislodnevtydnu = 5;
switch (cislodnevtydnu)
{
```

```
 6  case 1:
 7        Nazevdnevtydnu = "Pondeli";
 8        break;
 9  case 2:
10        Nazevdnevtydnu = "Utery";
11        break;
12  case 3:
13        Nazevdnevtydnu = "Streda";
14        break;
15  case 4:
16        Nazevdnevtydnu = "Ctvrtek";
17        break;
18  case 5:
19        Nazevdnevtydnu = "Patek";
20        break;
21  case 6:
22        Nazevdnevtydnu = "Sobota";
23        break;
24  case 7:
25        Nazevdnevtydnu = "Nedele";
26        break;
27  default:
28        Nazevdnevtydnu = "Chyba: neplatna hodnota";
29        break;
30 }
31 Console.WriteLine(Nazevdnevtydnu);
```

3.7.4 Cyklus FOR

Cyklus FOR umožňuje opakování určitého bloku příkazů. Před každým provedením těchto příkazů se testuje platnost podmínky. Ze všech cyklů, které zde budou zmíněny, tento příkaz nabízí nejflexibilnější možnosti nastavení.

Jeho obecná syntaxe vypadá následovně:

```
1 for (inicializátor; podmínka; krok)
2 příkaz (příkazy)
```

Jak je patrné ze zápisu syntaxe příkazu, tento příkaz má následující tři parametry:

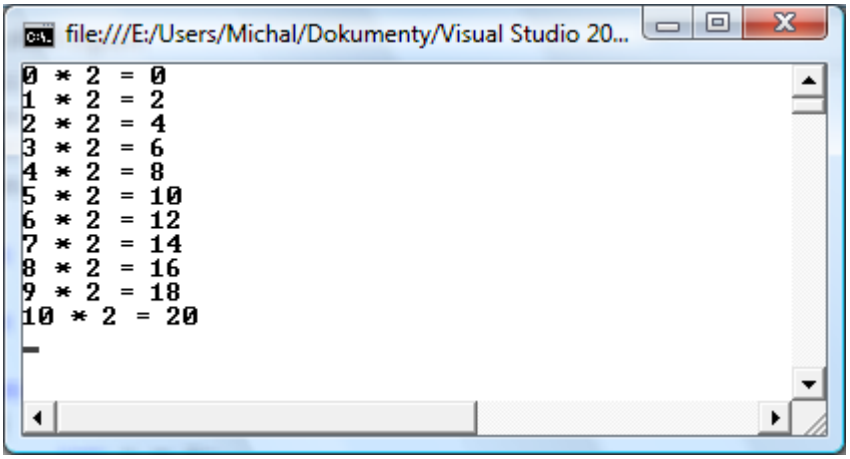
- Inicializátor – zde se inicializuje lokální proměnná, která slouží jako počítadlo cyklu.
- Podmínka - výraz, který se kontroluje pro každou další iteraci cyklu.

- Krok – výraz, který se vyhodnotí po každé operaci (většinou se jedná o zvýšení čítače cyklu).

Použití cyklu FOR demonstruje níže uvedená ukázka (výstup tohoto programu znázorňuje obr. 6):

```
1 int nasobek = 2;
2
3 for (int i = 0; i <= 10; i++)
4   {
5         Console.WriteLine("{0} * {1} = {2}",i, nasobek, i*nasobek);
6   }
```

Pozn. Výše uvedená ukázka neobsahuje základní kostru konzolové aplikace, ta je uvedena v kapitole 3.1.



Obr. 6. Výstup ukázkového programu pro cyklus FOR

3.7.5 Cyklus WHILE

Další možností použití cyklů je příkaz WHILE. Tento cyklus se provádí na základě vyhodnocení podmínky, která je uvedena v hlavičce cyklu. Podmínka se vyhodnocuje před použitím příkazu, tudíž může dojít k situaci, kdy blok příkazů uvedený v těle tohoto cyklu nebude proveden ani jednou.

Cyklus WHILE má následující syntaxi:

```
1 while (podmínka)
2 příkaz (příkazy)
```

Pro ukázku funkce tohoto cyklu byl zvolen stejný program jako u příkazu FOR. Výstup tohoto programu bude stejný jako v předchozím případě (viz. obr. 6).

```
1 int i = 0;
2 int nasobek = 2;
```

```
3
4 while (i <= 10)
5  {
6        Console.WriteLine("{0} * {1} = {2}",i, nasobek, i*nasobek);
7        i++;
8 }
```

Pozn. Výše uvedená ukázka neobsahuje základní kostru konzolové aplikace, ta je uvedena v kapitole 3.1.

3.7.6 Cyklus DO … WHILE

Cyklus DO … WHILE je velmi podobný cyklu WHILE. Je zde však jedna zásadní změna – podmínka je uvedena až na konci cyklu. Tato změna je příčinou odlišného způsobu vyhodnocování podmínky: nejprve se provede blok příkazů a následně je vyhodnocena podmínka. Na rozdíl od cyklu WHILE je zde kód vždy proveden alespoň jednou.

Obecná syntaxe je následující:

```
1 do
2 příkaz (příkazy)
3 while (podmínka);
```

Opět následuje ukázka stejného programu, který byl použit v předchozích příkladech:

```
1 int i = 0;
2 int nasobek = 2;
3
4 do
5  {
6        Console.WriteLine("{0} * {1} = {2}",i, nasobek, i*nasobek);
7        i++;
8  }
9 while (i <= 10);
```

Pozn. Výše uvedená ukázka neobsahuje základní kostru konzolové aplikace, ta je uvedena v kapitole 3.1.

3.7.7 Cyklus FOREACH

Cyklus FOREACH se liší od předchozích příkazů. Tento cyklus nevyhodnocuje podmínku, ale slouží k procházení polí nebo tzv. kolekcí (např. ArrayList). Tento cyklus je novinkou, která byla převzata z jazyka Visual Basic. Jeho syntaxe vypadá následovně:

```
1 foreach (datovytyp identifikator in seznam)
2 příkaz (příkazy)
```

Funkci tohoto příkazu nejlépe vystihne následující příklad (program vypíše všechny prvky pole):

```
1 int[] pole = new int[] { 0, 1, 2, 3, 5, 8, 13 };
2
3 foreach (int i in pole)  příkaz (příkazy)
4   {
5         Console.WriteLine(i);
6   }
```

Pozn. Výše uvedená ukázka neobsahuje základní kostru konzolové aplikace, ta je uvedena v kapitole 3.1.

3.7.8 Skok GOTO

Příkaz GOTO umožňuje napevno nadefinovat skok v rámci programu na řádek, který je označen návěštím (identifikátor následovaný dvojtečkou).

Skok lze použít například pomocí následujícího zápisu:

```
1 goto Label1;
2         Console.WriteLine("Tento příkaz se neprovede!");
3 Label1:
4         Console.WriteLine("Program pokračuje od tohoto řádku!");
```

Tento příkaz má několik omezení. Nelze jej používat pro skoky do bloků kódu, který je uvnitř cyklu, a nelze vyskočit z metody. **Používání tohoto příkazu se nedoporučuje.**

3.7.9 Příkazy BREAK a CONTINUE

Poslední dva příkazy, které budou zmíněny v kapitole „Řízení toku programu“, lze používat ve všech cyklech, které zde byly zmíněny.

Prvním příkazem je příkaz **break**, který ukončí vykonávání právě prováděného cyklu. Tento příkaz se používá také u příkazu SWITCH.

Druhým příkazem je příkaz **continue**. Má podobný význam jako *break*, ale ukončí pouze vykonávání následující iterace. To znamená, že se začne vykonávat následující průchod cyklem.

3.8 Výjimky a zpracování chyb

Ve většině programů může dojít při běhu k určitému chybovému stavu.

V technologii .NET Framework se tyto chyby částečně eliminují již při překladu, avšak je zde řada chyb, které nelze předem detekovat. Pro tyto situace má platforma .NET výjimky.

Nejčastější chybové situace, které nelze předem detekovat, jsou:

- Snaha o otevření neexistujícího souboru
- Nedostatek paměti, či jiných systémových zdrojů
- Dělení nulou
- Indexování mimo povolené meze

Dojde-li k výskytu některé z chyb, je vygenerován objekt výjimky a ta může být zobrazena v okně. Následující tabulka shrnuje seznam několika předdefinovaných výjimek (tab. 12).

Tab. 12. Předdefinované výjimky z knihovny .NET Framework

Výjimka	Popis
Exception	Základní třída pro všechny výjimky.
SystemException	Základní třída pro všechny druhy výjimek generovaných za běhu aplikace.
IndexOutOfRangeException	Generována při indexování pole mimo povolené meze.
NullReferenceException	Generována při pokusu pracovat s neinicializovanou funkcí.
InvalidOperationException	Generována metodou, která není ve stavu volání zpracovat.
ArgumentException	Základní třída pro všechny výjimky způsobené chybnými argumenty.
ArgumentNullException	Generována metodou, která neočekává parametr s nulovou hodnotou.
InteropException	Výjimka generovaná chybou mimo runtime prostředí .NET Framework.
ComException	Chyby ve formě HRESULT jsou v .NET aplikacích transformovány do této výjimky.
SEHException	Strukturovaná výjimka Win32.
DivideByZeroException	Generována při pokusu o dělení nulou.
FileNotFoundException	Generována při pokusu o přístup k neexistujícímu souboru.

3.8.1 Příkaz TRY-CATCH

Pro zachytávání výjimek slouží příkaz TRY-CATCH. První část tohoto příkazu, blok *try*, obsahuje příkazy, které se mají provést a u kterých má být kontrolováno, zda není vyvolána výjimka. Následuje minimálně jeden blok *catch*, který zpracovává výjimky.

Použití příkazu TRY-CATCH lze ilustrovat následujícím programem, který bude zpracovávat chybu při dělení nulou.

```
1  class Matematika
2  {
```

```
 3
 4         static int Podil (int a, int b)
 5         {
 6               int n = 0;
 7               try
 8               {
 9                     n = a/b;
10               }
11               catch(DivideByZeroException e)
12               {
13                     // dělení  nulou …
14                     Console.WriteLine("Nulou dělit nelze!");
15               }
16               catch(Exception e)
17               {
18                     // ostatní výjimky …
19                     Console.WriteLine("Výjimka: {0}",e.Message);
20               }
21
22               return n;
23
24         }
25
26         public static void Main()
27         {
28               int a = Console.ReadLine();
29               int b = Console.ReadLine();
30               int x = Podil(a,b);
31         }
32
33 }
```

3.8.2 Blok FINALLY

Příkaz TRY-CATCH, který byl popsán v předešlé kapitole, lze doplnit ještě o blok *finally*. Příkazy umístěné v tomto bloku se provedou vždy, tedy i při vyvolání výjimky.

3.9 Vybrané metody MS .NET Framework

V této kapitole budou shrnuty nejpoužívanější metody platformy .NET rozdělené dle oblasti použití.

3.9.1 Konzolové aplikace

Metody pro práci v konzolových aplikacích již byly několikrát použity v příkladech v předchozích kapitolách. Jedná se zejména o funkce pro načtení dat nebo pro výpis textu do konzole.

Metody pro práci s konzolovými aplikacemi jsou definovány ve třídě *Console*, která se nachází ve jmenném prostoru *System*. Přehled nejpoužívanějších metod je obsahem níže uvedené tabulky (tab. 13).

Tab. 13. Nejpoužívanější metody třídy Console

Metoda	Popis
Clear	Vymazání obsahu okna konzole
Read	Načtení znaku z okna konzole
ReadLine	Načtení řádku textu z okna konzole
ReadKey	Načtení znaku nebo funkční klávesy
Write	Výpis textu do okna konzole
WriteLine	Výpis řádku do okna konzole

Pozn. Třída *Console* samozřejmě obsahuje mnohem více metod. Kompletní přehled všech metod a atributů této třídy je k dispozici na webu MSDN[3].

3.9.2 Matematické funkce

Metody z oblasti matematiky jsou definovány ve třídě *Math*, která se nachází ve jmenném prostoru *System*. Přehled nejpoužívanějších metod je obsahem následující tabulky (tab. 14).

[3] Dokumentace této třídy je k dispozici na adrese < http://msdn2.microsoft.com/en-us/library/system.console.aspx>

Tab. 14. Nejpoužívanější metody třídy Math

Metoda	Popis
Abs	Absolutní hodnota
Exp	Mocnina Eulerova čísla
Log	Přirozený logaritmus nebo logaritmus o zadaném základu
Log10	Logaritmus o základu 10
Pow	Mocnina
Round	Zaokrouhlení
Sqrt	Odmocnina
Sin	Sinus
Cos	Cosinus
Tan	Tangens

Pozn. Třída *Math* samozřejmě obsahuje mnohem více metod. Kompletní přehled všech metod a atributů této třídy je k dispozici na webu MSDN[4].

3.9.3 Práce s textem

Stejně jako i jiné programovací jazyky obsahují jazyky z rodiny .NET celou řadu metod pro práci s textovými řetězci. Metody z této oblasti jsou definovány ve třídě *String*, která se nachází ve jmenném prostoru *System*. Přehled nejpoužívanějších metod je obsahem následující tabulky (Tab. 15).

Tab. 15. Nejpoužívanější atributy a metody třídy String

Atribut / Metoda	Popis
Length	Počet znaků
Replace	Nahrazení části textu
SubString	Vrátí podřetězec z daného řetězce
ToLower	Převod textu na malá písmena
ToUpper	Převod textu na velká písmena
Trim	Odebrání mezer na začátku i na konci textu
TrimEnd	Odebrání mezer na konci textu
TrimStart	Odebrání mezer na začátku textu

Pozn. Třída *String* samozřejmě obsahuje mnohem více metod. Kompletní přehled všech metod a atributů této třídy je k dispozici na webu MSDN[5].

[4] Dokumentace této třídy je k dispozici na adrese <http://msdn2.microsoft.com/en-us/library/system.math.aspx>
[5] Dokumentace této třídy je k dispozici na adrese <http://msdn2.microsoft.com/en-us/library/system.string.aspx>

3.10 Windows Forms

Knihovna tříd Windows Forms je součástí technologie MS .NET Framework a umožňuje uživateli vývoj grafického rozhraní aplikací v MS Windows. Potřebné třídy lze nalézt ve jmenném prostoru *System.Windows.Forms*.

Základním prvkem většiny aplikací v operačním systému MS Windows je formulář. Hlavní formulář aplikace je označován jako *okno aplikace*. V případě, že je formulář zobrazen jako odezva na určitou událost, hovoří se o *dialogu*.

3.10.1 Dialogy a předávání dat

Pokud se formulář zobrazí jako reakce na požadavek uživatele na nějakou službu, jedná se o dialog.

Lze rozlišit dva základní typy dialogů:

- Modální
- Nemodální

Modální dialogy zastaví veškeré uživatelské interakce s aplikací. Aktivní je pouze daný dialog. Pro zobrazení modálního dialogu se používá metoda *ShowDialog()*, což v praxi může vypadat následovně:

```
1 …
2 DialogProNastaveniParametru dlg = new DialogProNastaveniParametru()
3 DialogResult res = dlg.ShowDialog();
4 …
```

V případě nemodálních dialogů může uživatel přistupovat i k hlavnímu oknu nebo k jiným dialogům. Použití je velmi podobné předchozímu příkladu, s tím rozdílem, že místo metody *ShowDialog()* je použita metoda *Show()*.

```
1 …
2 DialogProNastaveniParametru dlg = new DialogProNastaveniParametru()
3 DialogResult res = dlg.Show();
4 …
```

Jak je patrné z obou předchozích příkladů, metody *Show()* a *ShowDialog()* vracejí hodnotu typu *DialogResult*. *DialogResult* může nabývat následujících hodnot:

- *Abort* (odezva na tlačítko *Přerušit*)
- *Cancel* (odezva na tlačítko *Storno*)

- *Ignore* (odezva na tlačítko *Přeskočit*)
- *No* (odezva na tlačítko *Ne*)
- *None* (žádná návratová hodnota)
- *OK* (odezva na tlačítko *OK*)
- *Retry* (odezva na tlačítko *Opakovat*)
- *Yes* (odezva na tlačítko *Ano*)

Návratovou hodnotu tlačítka určuje atribut DialogResult. U uživatelsky vytvářených ovládacích prvků, které mají ukončit práci s dialogem, je tedy nutné tuto hodnotu nastavit. V opačném případě je odeslána hodnota *None*. Metoda takového ovládacího prvku může vypadat například následovně:

```
1 void okButton_Click(object sender, EventArgs e)
2 {
3 this.DialogResult = DialogResult.OK;
4 this.Close();
5 }
```

Poslední (a nejdůležitější) věcí, kterou je třeba u dialogu vyřešit, je předávání dat. Jelikož všechny ovládací prvky mají soukromé atributy (*private*), nelze k nim přistupovat běžnou cestou. Určitým řešením by byla možnost změny atributů na veřejné. Tento postup se však nedoporučuje.

V případě potřeby předávání dat je nutné ve třídě dialogu vytvořit atribut, který bude předávat hodnoty ovládacím prvkům pomocí metod *get* a *set*.

```
1 public float SouradniceX
2 {
3         get
4         {
5               return float.Parse(txtX.Text);
6         }
7         set
8         {
9               txtX.Text = value.ToString();
10        }
11 }
```

Kromě uživatelských dialogů nabízí technologie .NET také řadu standardních dialogů, které jsou definovány ve jmenném prostoru *System.Windows.Forms*. Lze tedy používat následující předdefinované dialogy:

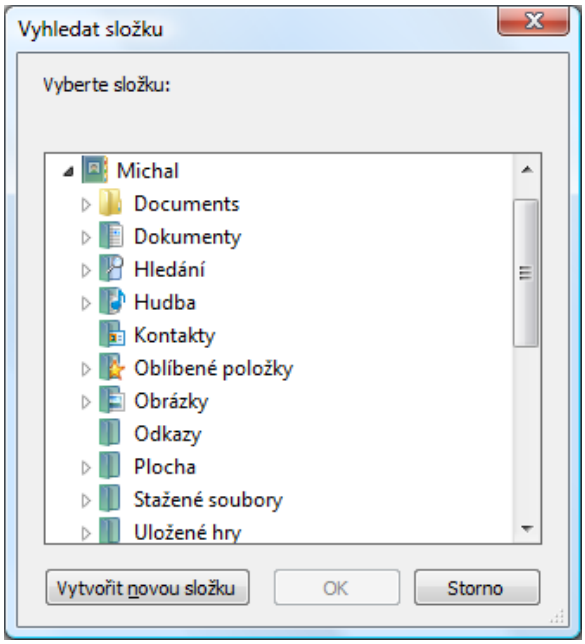
- *ColorDialog* – dialog pro výběr barvy
- *FolderBrowserDialog* - dialog pro výběr složky
- *FontDialog* – nastavení typu a vlastností písma
- *OpenFileDialog* – dialog pro otevření souboru
- *SaveFileDialog* – dialog pro uložení souboru
- *PageSetupDialog* – nastavení vlastností stránky pro tisk
- *PrintDialog* – dialog tisku
- *PrintPreviewDialog* – náhled stránky

Použití jednoho z výše uvedených standardních dialogů (dialogu pro výběr složky) ilustruje následující příklad [1]:

```
1  using System;
2  using System.Windows.Forms;
3
4  namespace FolderDialog
5  {
6     class Program
7     {
8        [STAThread]
9        static void Main(string[] args)
10       {
11           Console.Title = "Dialog pro výběr složky";
12
13          FolderBrowserDialog fbd = new FolderBrowserDialog();
14          fbd.SelectedPath = "C:\\";
15          // fbd.RootFolder = Environment.SpecialFolder.Personal;
16          fbd.ShowNewFolderButton = true;
17          fbd.Description = "Vyberte složku:";
18          if (fbd.ShowDialog() == DialogResult.OK)
19          {
20             Console.WriteLine(fbd.SelectedPath);
21          }
22
23          Console.WriteLine();
24          Console.WriteLine("Stiskni Enter");
```

```
25            Console.ReadLine();
26        }
27    }
28 }
```

Po spuštění tohoto programu je otevřen dialog pro výběr složky (obr. 7) a vybraná cesta je vypsána do okna konzole.



Obr. 7. Dialog pro výběr složky

Použití standardních dialogů je detailněji popsáno v [9] a v [12].

3.10.2 Základní ovládací prvky

Každý formulář může obsahovat jeden nebo více ovládacích prvků. Windows Forms jich nabízí celou škálu, od těch základních, jako například textové pole, až po méně typické, mezi kterými lze najít například kalendář.

Ovládací prvky lze rozdělit do několika základních kategorií:

- Akční ovládací prvky
- Hodnotové ovládací prvky
- Ovládací prvky pro seznam
- Kontejnerové ovládací prvky

Do skupiny akčních ovládacích prvků lze zařadit ovládací prvky, které na základě konání uživatele (nejčastěji kliknutí) vyvolají v aplikaci určitou akci. Nejpoužívanějším zástupcem této kategorie je tlačítko (*button*).

Hodnotové ovládací prvky slouží k zobrazení (a někdy i k editaci) určité hodnoty. Nejpoužívanějšími zástupci jsou *Label* (pro zobrazení textové hodnoty) a *PictureBox* (pro zobrazení obrázku).

Ovládací prvky pro seznam lze rozdělit do dvou skupin. První skupina nabízí pouze zobrazení seznamu hodnot. Uživatel také může z tohoto seznamu jednu nebo i více hodnot vybrat. Do této skupiny lze zařadit například otevřený seznam (*ListBox*) nebo pole se seznamem (*ComboBox*). Druhou skupinou jsou ovládací prvky, které umožňují i editaci položek v seznamu. Typickým zástupcem této skupiny je mřížka dat (*DataGrid*).

Poslední kategorii ovládacích prvků tvoří kontejnerové ovládací prvky, které umožňují do kontejneru seskupovat ovládací prvky a v něm je uspořádat. Do této skupiny jsou zařazeny pouze tři ovládací prvky: rámeček skupiny prvků (*GroupBox*), panel (*Panel*) a listovací rámeček (*TabControl*).

3.11 Práce s grafikou

Knihovna Windows Forms obsahuje celou řadu ovládacích prvků, ale i tak se může stát, že programátor nenajde vhodný ovládací prvek pro požadovaný způsob zobrazení stavu aplikace. V tomto případě přichází v úvahu použití nástrojů pro kreslení.

Kreslit lze na obrazovku, do souboru nebo na tiskárnu. Ve všech těchto případech jsou základními prvky barva, štětec, pero a písmo.

Všechny třídy, které se týkají kreslení, jsou obsaženy ve jmenném prostoru *System.Drawing*, který je implementován nad GDI+ (Graphics Device Interface +). S jeho předchůdcem (označeným GDI) se bylo možno setkat již v předchozích verzích operačního systému Windows a stejně jako GDI+ poskytoval abstrakci nad obrazovkami a tiskárnami, což vedlo ke zjednodušení vytváření GUI.

3.11.1 Kreslení

Ať už uživatel bude chtít kreslit na obrazovku, či na jiné médium, ve všech případech bude zacházet s podkladovou abstrakcí, třídou *Graphics*, která se nachází ve jmenném prostoru *System.Drawing*.

Jedním ze způsobů, jak lze získat objekt grafiky, je zavolání metody *CreateGraphics*, čímž se vytvoří objekt grafiky sdružený s formulářem. Protože objekt grafiky drží podkladový prostředek, který spravuje operační systém, je nutné, aby byl tento prostředek uvolněn po dokončení kreslení. Pro uvolnění podkladové grafiky se zde nabízejí dvě možnosti. První možností je zavolání metody *Dispose*. Alternativou je napsání programu tak, aby metoda *Dispose* byla zavolána automaticky. Tuto možnost znázorňuje následující příklad, na kterém je vidět i to, jak kreslení pracuje.

```
1 bool drawEllipse = false;
2
3 public DrawingForm()
4 {
5 InitializeComponent();
6 }
7
8 void drawEllipseButton_Click(object sender, EventArgs e)
9 {
10        using( Graphics g = this.CreateGraphics() )
11        {
12        if (!drawEllipse)
13        {
14        g.FillEllipse(Brushes.DarkBlue, this.ClientRectangle);
15        }
16        else
17        {
18        g.FillEllipse(SystemBrushes.Control, this.ClientRectangle);
19        }
20        drawEllipse = !drawEllipse;
21        }
22 }
```

Pokud je použita metoda ve výše uvedené podobě, programátor narazí na zásadní problém. Tímto problémem je změna velikosti formuláře a posunutí formuláře mimo pracovní plochu (obr. 8a).

Pro tento případ však existuje řešení, a to událost *Paint*. Tato událost je volána vždy, když je požadováno překreslení okna, což nastává právě ve zmiňovaných případech.

Úpravou předchozí ukázky lze získat program, který bude obsahovat událost *Paint*. Ta už se postará o překreslení formuláře ve výše uvedených případech.

```
1 bool drawEllipse = false;
2
3 public DrawingForm()
4 {
5 InitializeComponent();
6 }
7
8 private void button1_Click(object sender, EventArgs e)
9  {
10        drawEllipse = !drawEllipse;
11        this.Invalidate(true);
12 }
13
14 private void DrawingForm_Paint(object sender, PaintEventArgs e)
15  {
16        if (!drawEllipse) return;
17        Graphics g = e.Graphics;
18        g.FillEllipse(Brushes.DarkBlue, this.ClientRectangle);
19  }
```

Ve výše uvedeném příkladě je událost *Paint* vyvolána metodou *Invalidate*. Kreslení je jednou z nejnáročnějších operací, operační systém proto nejdříve zpracuje jiné události, jako například vstup z klávesnice nebo pohyb myši, což může způsobit prodlevu při překreslování. Tuto prodleva lze odstranit doplněním metody *Update*, což by mohlo vypadat následovně:

```
1 private void button1_Click(object sender, EventArgs e)
2  {
3        drawEllipse = !drawEllipse;
4        this.Invalidate(true);
5        this.Update();
6 }
```

Stejného výsledku se dosáhne také nahrazením metody *Invalidate* metodou *Refresh*:

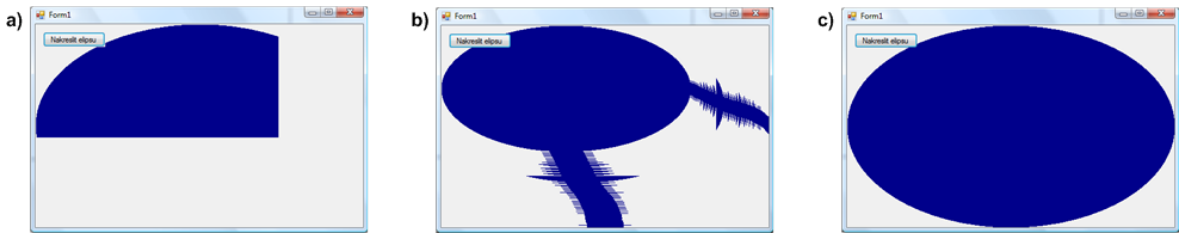
```
1 private void button1_Click(object sender, EventArgs e)
2  {
3        drawEllipse = !drawEllipse;
4        this.Refresh();
5 }
```

Předchozími úpravami sice bylo docíleno toho, aby se formulář překresloval při změně rozměrů a při posunutí, ale výsledek stále není ideální. Překreslována je totiž pouze nově zobrazená část formuláře (obr. 8b). Pro překreslení celého formuláře je třeba do výše uvedeného kódu doplnit ještě jeden řádek:

```
1 this.SetStyle(ControlStyles.ResizeRedraw, true);
```

Výsledný program (obr. 8c) tak bude vypadat následovně:

```
1  bool drawEllipse = false;
2
3  Public DrawingForm()
4  {
5  InitializeComponent();
6  this.SetStyle(ControlStyles.ResizeRedraw, true);
7  }
8
9  private void button1_Click(object sender, EventArgs e)
10   {
11         drawEllipse = !drawEllipse;
12         this.Refresh();
13 }
14
15 private void DrawingForm_Paint(object sender, PaintEventArgs e)
16   {
17         if (!drawEllipse) return;
18         Graphics g = e.Graphics;
19         g.FillEllipse(Brushes.DarkBlue, this.ClientRectangle);
20   }
```



Obr. 8. Okno s nakreslenou elipsou po zmenšení a následném zvětšení

a) základní způsob kreslení

b) použití události Paint

c) *použití události Paint a nastavení překreslování celého formuláře*

3.11.2 Barvy

První věcí, kterou je třeba při kreslení nastavit, je barva vykreslovaného prvku. Technologie .NET Framework nabízí několik možností definice barvy:

- Použití známé barvy
- Použití barvy operačního systému
- Definice barvy přes metody struktury *Color*

Použití známé barvy je nejjednodušším způsobem definice barvy. Ve struktuře Color je předdefinována řada základních barev a přistupuje se k nim následujícím zápisem:

```
1 Color.OznačeníBarvy
```

Za *OznačeníBarvy* se dosadí anglický ekvivalent názvu barvy. Přehled některých základních barev shrnuje následující tabulka (tab. 16).

Tab. 16. Některé vybrané předdefinované barvy

Označení barvy	Barva
Black	Černá
White	Bílá
Gray	Šedá
Yellow	Žlutá
Blue	Modrá
Red	Červená
Green	Zelená
Chocolate	Čokoládová
Cyan	Azurová
...	

Druhý způsob definice barvy je velmi podobný. Opět se jedná o množinu předdefinovaných barev. V tomto případě jsou ale využity barvy prostředí operačního systému. Barva se v tomto případě zapisuje v následujícím tvaru:

```
1 SystemColor.OznačeníPrvku
```

Za *OznačeníPrvku* se dosadí anglický ekvivalent názvu části prostředí operačního systému. Přehled některých prvků prostředí shrnuje níže uvedená tabulka (tab. 17).

Tab. 17. Některé vybrané části prostředí operačního systému

Označení prvku	Popis
Control	Ovládací prvek
Desktop	Pracovní plocha
Menu	Menu
ScrollBar	Posuvník
Window	Okno
...	

Posledním způsobem definice barvy je definice pomocí jednotlivých složek (červená, zelená, modrá a průhlednost). Tento způsob poskytuje největší svobodu ve volbě barvy ze zde uvedených způsobů. Barva se nadefinuje následujícím příkazem:

```
1 Color.FromArgb( [alpha] , red , green , blue )
```

První parametr příkazu (*alpha*) definuje průhlednost a je nepovinný. Následují jednotlivé barevné složky v pořadí červená (*red*), zelená (*green*) a modrá (*blue*). Všechny parametry nabývají celočíselných hodnot z intervalu od 0 do 255.

3.11.3 Štětce

Dalším parametrem kreslení je štětec. Štětec definuje výplň kresleného objektu a je definován ve třídě *Systém.Drawing.Brush*. Lze je rozdělit dle druhu na:

- Barevné štětce (*SolidBrush*)
- Texturové štětce (*TextureBrush*)
- Šrafovací štětce (*HatchBrush*)
- Gradientní štětce
    - Štětce s lineárním gradientem (*LinearGradientBrush*)
    - Štětce s gradientem založeným na cestě (*PathGradientBrush*)

Barevné štětce

Barevný štětec vyplní oblast ohraničenou kresleným tvarem jednou barvou. Definice barevných štětců je velmi podobná definici barev z předchozí kapitoly.

První možností je použití známé barvy. V tomto případě je v programu uveden štětec ve tvaru *System.Drawing.Brushes.OznačeníBarvy*. Za *OznačeníBarvy* se dosadí například jedna z barev, která je uvedena v tab. 16 na straně 57.

Druhou možností je použití barvy prostředí operačního systému. Toho lze docílit použitím štětce zapsaného ve tvaru *System.Drawing.SystemBrushes.OznačeníPrvku*. Za *OznačeníPrvku* se dosadí například jedna z barev, která je uvedena v tab. 17 na straně 58.

Pokud si tvůrce programu nevystačí s předdefinovanými štětci, tak se mu nabízí ještě třetí možnost – vytvoření vlastního štětce. Vlastní štětec lze vytvořit pomocí konstruktoru, v jehož parametru bude objekt typu *Color*. Vytvoření takového štětce může vypadat následovně:

```
1 Brush modryStetec = new SolidBrush(Color.FromArgb(0,0,255));
```

Texturové štětce

Štětec *TextureBrush* vyplňuje určitou oblast obrázkem. Konstruktoru objektu *TextureBrush* je nutné předat dva parametry. Prvním parametrem je obrázek a druhým způsob jeho opakování (tab. 18). Obecný zápis takového konstruktoru vypadá následovně:

```
1 TextureBrush(Image image, WrapMode wrapMode)
```

Tab. 18. Hodnoty parametru WrapMode u texturového štětce [24]

Hodnota	Význam
WrapMode.Clamp	Bez opakování.
WrapMode.Tile	Vyskládá do nekonečna nezměněné stejné obrázky.
WrapMode.TileFlipX	Vyskládá obrázky do nekonečna, převrátí každý druhý obrázek horizontálně.
WrapMode.TileFlipY	Vyskládá obrázky do nekonečna, převrátí každý druhý obrázek vertikálně.
WrapMode.TileFlipXY	Kombinace obou předchozích.

Šrafovací štětce

Šrafovacím štětcem (*HatchBrush*) se vyplňuje prostor jedním z 56 zabudovaných dvoubarevných vzorků. Pro vytvoření šrafovacího štětce lze použít následující konstruktor:

```
1 HatchBrush(HatchStyle HatchStyle, Color Color1, Color Color1)
```

Prvním parametrem je parametr HatchStyle (tab. 19), který určuje typ šrafování. Následují dva parametry definující barvy, které jsou použity pro vybrané šrafování.

Tab. 19. Některé hodnoty parametru HatchStyle

Hodnota	Význam
Cross	Šrafování do kříže (horizontálně a vertikálně)
Horizontal	Horizontální šrafování
Vertical	Vertikální šrafování
Percent25	25-procentní šrafování
Percent75	75-procentní šrafování
...	

Gradientní štětce

Gradientní štětce jsou v .NET Framework dva. Prvním je lineární gradientní štětec (*LinearGradientBrush*) a druhým štětec s gradientem založeným na cestě (*PathGradientBrush*). V obou případech je vytvořen plynulý přechod dvou barev s tím rozdílem, že se barva mění dle nadefinované cesty.

Lineární gradientní štětec lze nadefinovat následovně:

```
1 LinearGradientBrush prechod = new
  LinearGradientBrush(this.ClientRectangle, Color.Black, Color.Azure,
  LinearGradientMode.Horizontal);
```

3.11.4 Pera

Pera jsou definována ve třídě System.Drawing.Pen a lze jimi ovlivnit vzhled ohraničení kresleného objektu.

Každé pero má několik základních vlastností:

- Síla čáry
- Barva nebo štětec
- Vzorek pro přerušované čáry
- Tvar začátku a konce čáry

V konstruktoru pera se jako první parametr zadává barva nebo štětec, které byly popsány v předchozích dvou kapitolách. Druhým parametrem je tloušťka pera.

Po vytvoření instance pera již lze nastavovat další vlastnosti. První užitečnou vlastností je vzorek pro přerušovanou čáru. Jedná se o atribut *DashStyle*, který může nabývat hodnot z tab. 20.

Tab. 20. Hodnoty parametru DashStyle

Hodnota	Význam
Solid	Plná čára
Dash	Přerušovaná čára
DashDot	Čerchovaná
DashDotDot	Čerchovaná s dvěma tečkami
Dot	Tečkovaná čára
Custom	Vlastní nastavení dle atributu *DashPattern*

Druhou užitečnou vlastností je tvar začátku a konce čáry. Ten lze nastavit pomocí atributů *StartCap* a *EndCap*, které nabývají hodnot z výčtového typu *LineCap*, jehož hodnoty jsou obsahem následující tabulky (tab. 21).

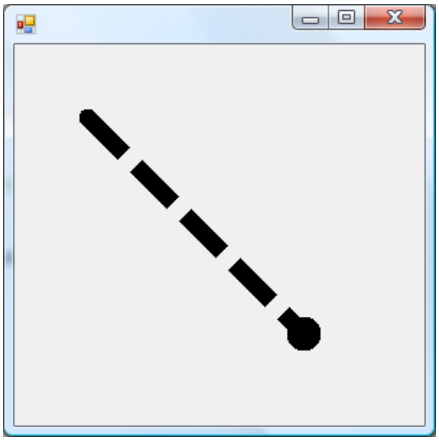
Tab. 21. Hodnoty výčtového typu LineCap

Hodnota	Význam
ArrowAnchor	Šipka s kotvou
DiamantAnchor	Káro s kotvou
Flat	Ploché zakončení
NoAnchor	Zakončení bez kotvy
Round	Kulaté zakončení bez kotvy
RoundAnchor	Kulaté zakončení s kotvou
Square	Čtvercové zakončení bez kotvy
SquareAnchor	Čtvercové zakončení s kotvou
Triangle	Trojúhelníkové zakončení bez kotvy
Custom	Vlastní nastavení

Pozn. Kotva znamená, že dané zakončení přesahuje šířku pera.

Na závěr této kapitoly je vhodné zmínit příklad na použití pera. Jedná se o jednoduchý příklad, jehož výsledkem je vykreslení čáry na plochu formuláře.

```
1  using (Graphics g = this.CreateGraphics())
2  {
3          using (Pen pen = new Pen(Color.Black, 12))
4          {
5                pen.EndCap = LineCap.RoundAnchor;
6                pen.StartCap = LineCap.Round;
7                pen.DashStyle = DashStyle.Dash;
8                g.DrawLine(pen, 50, 50, 200, 200);
9          }
10 }
```



Obr. 9. Výsledek ukázkového programu pro vykreslení čáry

3.11.5 Základní tvary

V knihovně GDI+ je k dispozici celá řada tvarů, které lze rozdělit do dvou skupin: křivky a plošné útvary. Pokud si uživatel nevybere z předdefinovaných tvarů, má možnost si vytvořit i vlastní tvar definováním cesty.

Přehled základních tvarů shrnuje následující tabulka (tab. 22).

Tab. 22. Některé předdefinované tvary pro kreslení v GDI+

Typ objektu	Obrys	Výplň
Čára	*DrawLine*	-
Křivka	*DrawCurve*	-
Beziérova křivka	*DrawBezier*	-
Výseč	*DrawArc*	-
Uzavřená křivka	*DrawClosedCurve*	*FillClosedCurve*
Obdélník	*DrawRectangle*	*FillRectangle*
Elipsa	*DrawEllipse*	*FillEllipse*
Mnohoúhelník	*DrawPolygon*	*FillPolygon*

Pozn. Příkazy uvedené ve sloupcích „Obrys“ a „Výplň“ jsou metody objektu *Graphics*, jejich přesná syntaxe je uvedena v dokumentaci MSDN[6].

V případě obrysů je kreslený objekt definován perem a umístěním (okrajové body nebo obdélník, do kterého má být kresba umístěna). U výplní je kreslený objekt definován štětcem a umístěním (okrajové body nebo obdélník, do kterého má být kresba umístěna).

[6] Dokumentace této třídy je k dispozici na adrese <http://msdn2.microsoft.com/en-us/library/system.graphics.aspx>

Několik příkladů už bylo uvedeno v předchozích kapitolách. Prvním příkladem byla elipsa z kapitoly 3.11.1:

```
1 Graphics g = e.Graphics;
2 g.FillEllipse(Brushes.DarkBlue, this.ClientRectangle);
```

Druhý příklad, který se objevil v předchozí kapitole, obsahoval příkaz pro vykreslování čáry:

```
1 g.DrawLine(pen, 50, 50, 200, 200);
```

3.11.6 Kreslení textu

Jednou z možností využití knihovny GDI+ je i zobrazení textu. Pro tento účel slouží metoda *DrawString*, která je definována ve třídě *Graphics*.

Obecná syntaxe této metody se dá zapsat následovně:

```
1 Graphics.DrawString( Text , Font, Stetec , [Umisteni] , [Format] )
```

Pozn. Nepovinné parametry jsou uvedeny v hranatých závorkách.

Prvním parametrem metody pro nakreslení textu je parametr *Text*, jehož obsahem je textový řetězec, který má být vypsán. Tento parametr je povinný. Dalším povinným parametrem je *Font*. Jedná se o objekt, jehož obsahem jsou formátovací atributy (například velikost písma nebo rodina písma). Poslední (v pořadí třetí parametr) definuje štětec, který má být použit (štětcům se věnuje kapitola 3.11.3).

Zbylou dvojici parametrů tvoří nepovinné parametry. Prvním z nich je umístění vykreslovaného textu. Zde se uživateli nabízí několik možností, jak jej definovat:

- Bod definovaný jako struktura *Point*
- Souřadnice *X* a *Y*
- Obdélník definovaný jako struktura *Rectangle*

Druhým (a zároveň posledním) nepovinným parametrem je formát řetězce, ve kterém se definuje mimo jiné zkracování delších řetězců nebo zarovnání.

Třída Font

Jak už bylo zmíněno v předcházejících řádcích, jedním ze zadávaných parametrů je instance třídy Font.

Vzhled daného písma se nastaví v konstruktoru, který vypadá následovně:

```
1 Font(FontFamily, [FontSize,] [FontStyle,] [GraphicsUnit,] Byte,
  Boolean)
```

Pozn. Nepovinné parametry jsou uvedeny v hranatých závorkách.

Prvním parametrem konstruktoru je *FontFamily*. Zde se specifikuje rodina písma, která má být použita. Lze zadat buď tzv. generickou rodinu písma (tab. 23) nebo konkrétní písmo (například *Arial* nebo *Times New Roman*).

Tab. 23. Generické rodiny písem

Název	Popis
Monospace	Neproporcionální písmo (Ve výchozím nastavení bude použit *Curier New*)
SansSerif	Bezpatkové písmo (Ve výchozím nastavení bude použit *Microsoft Sans Serif*)
Serif	Patkové písmo (Ve výchozím nastavení bude použit *Times New Roman*)

Za rodinou písma následuje velikost písma a styl písma. Styl lze nadefinovat pomocí výčtového typu FontStyle. Pokud je nutné použití více stylů zároveň (například tučné a podtržené písmo), lze je spojit pomocí znaku „ | “. Styly, které je možno použít, shrnuje následující tabulka (tab. 24).

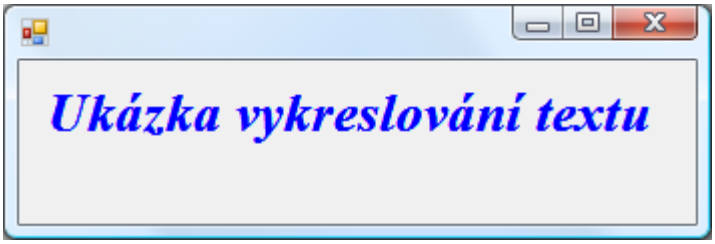
Tab. 24. Hodnoty výčtového typu FontStyle

Hodnota	Význam
Regular	Normální písmo
Bold	Tučné písmo
Italic	Kurzíva
Underline	Podtržení
Strikeout	Přeškrtnutí

Následně je možno definovat ještě obrazové jednotky (pixel – *GraphicsUnit.Pixel*, palec – *GraphicsUnit.Inch*, milimetr – *GraphicsUnit.Milimeter*, …),

Závěrem je vhodné uvést stručný příklad (jehož výsledek je zobrazen na obr. 10):

```
1 using (Graphics g = this.CreateGraphics())
2 {
3         Font novyFont = new
  Font(FontFamily.GenericSerif,20,FontStyle.Bold | FontStyle.Italic);
4         g.DrawString("Ukázka vykreslování
  textu",novyFont,Brushes.Blue,10,10);
5 }
```



Obr. 10. Výsledek ukázkového programu pro vykreslování textu

4 VÝVOJOVÉ NÁSTROJE

4.1 MS Visual Studio

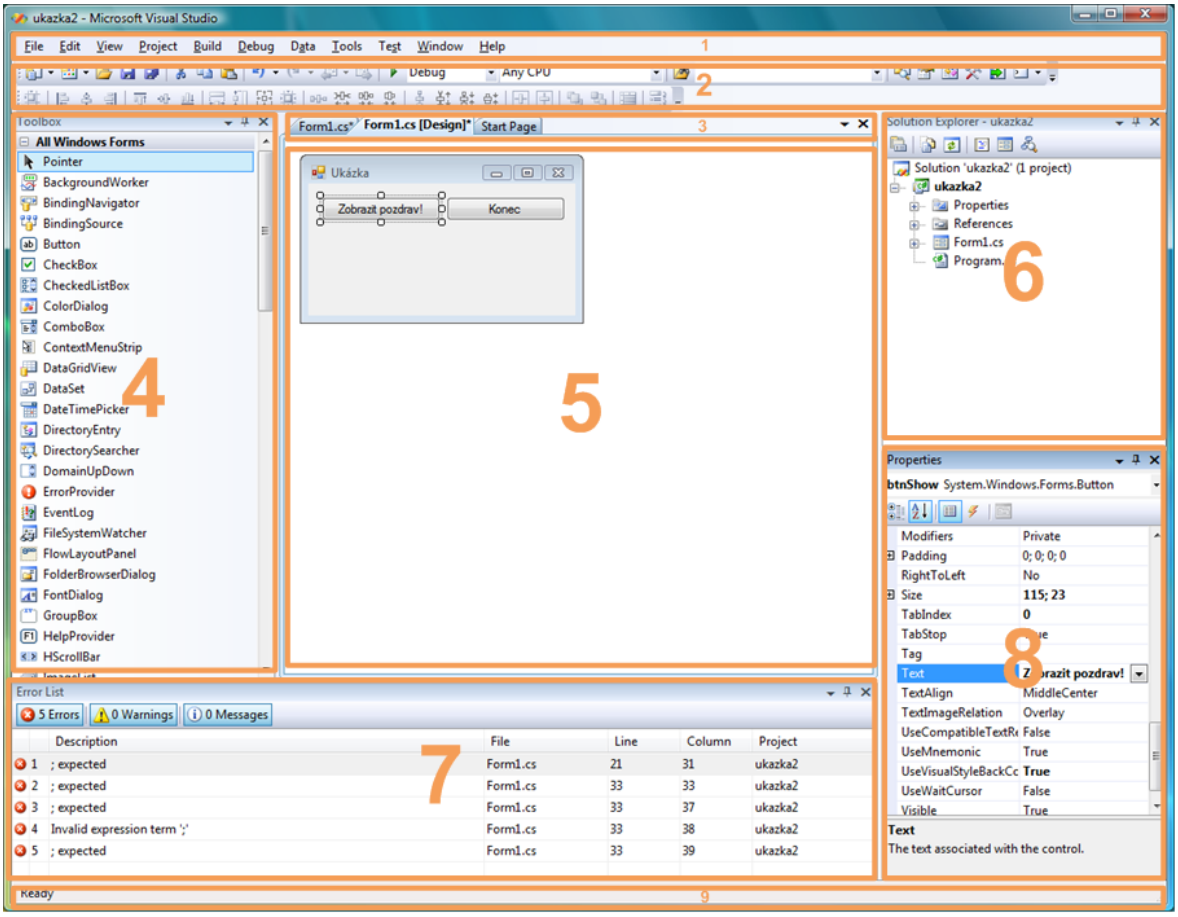
Nejznámějším a také nejrozšířenějším vývojovým nástrojem pro vývoj aplikací postavených na technologii MS .NET Framework je MS Visual Studio, které je aktuálně k dispozici ve verzi 2008[7].

Vývojové prostředí MS Visual Studio nabízí tyto funkce:

- **Textový editor** – umožňuje psát kód v jazycích C#, C++, Visual Basic, J#. Textový editor nabízí řadu nástrojů pro usnadnění psaní kódu. Například barevně označuje syntaxi, automaticky formátuje kód (odsazení řádku, …), podtržením označuje některé chyby v syntaxi nebo nabízí použití technologie IntelliSense (zobrazení seznamu názvů tříd, proměnných, polí a metod dle počátečních znaků).
- **Editor vizuálního návrhu** – umožňuje navrhování formulářů a rozmístění objektů uživatelského rozhraní.
- **Pomocná okna** – umožňují zobrazovat a upravovat určité parametry projektu (některá základní pomocná okna jsou popsána v kapitole 4.1.1, která se věnuje popisu okna aplikace).
- **Možnost překladu v prostředí** – integrace překladače přímo do prostředí MS Visual Studia.
- **Integrovaný ladící program** – umožňuje krokování a ladění programu přímo ve vývojovém prostředí.
- **Integrovaná nápověda MSDN** – přímý přístup k dokumentaci stisknutím klávesy F1 nad daným prvkem přímo z rozhraní programu (Nápovědě MSDN se detailněji věnuje kapitola 4.1.4).
- **Přístup k dalším programům** – například přístup k webovému prohlížeči MS Internet Explorer.

[7] MS Visual Studio 2008 bylo vydáno 19. 11. 2007

4.1.1 Okno aplikace



Obr. 11. Okno aplikace MS Visual Studio 2008

Okno aplikace MS Visual Studio obsahuje stejně jako okna většiny jiných aplikací menu (1)[8] (menu je detailněji popsáno v kapitole 4.1.2), nástrojovou lištu (2) a stavový řádek (9).

Největší část okna zabírá *pracovní plocha* (5), kde je zobrazen aktuálně otevřený soubor buď v textovém režimu (obr. 12), nebo v režimu návrhu (obr. 13). Nad pracovní plochou se nacházejí *záložky* (3) s možností přepínat mezi otevřenými soubory. Po okrajích pracovní plochy jsou poskládány jednotlivé pomocné bloky.

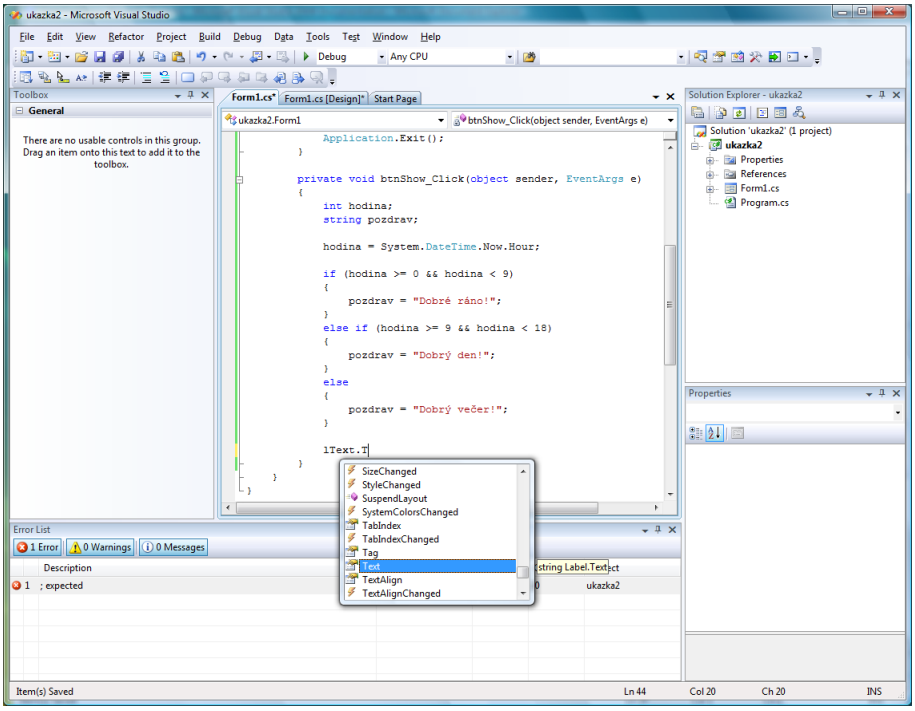
V levé části okna se nachází *Toolbox* (4). V tomto bloku jsou umístěny základní stavební prvky aplikací, zejména ovládací prvky formulářů.

[8] Čísla v závorkách uvedená v popisu okna aplikace *MS Visual Studio* odpovídají oranžovým blokům na Obr. 11

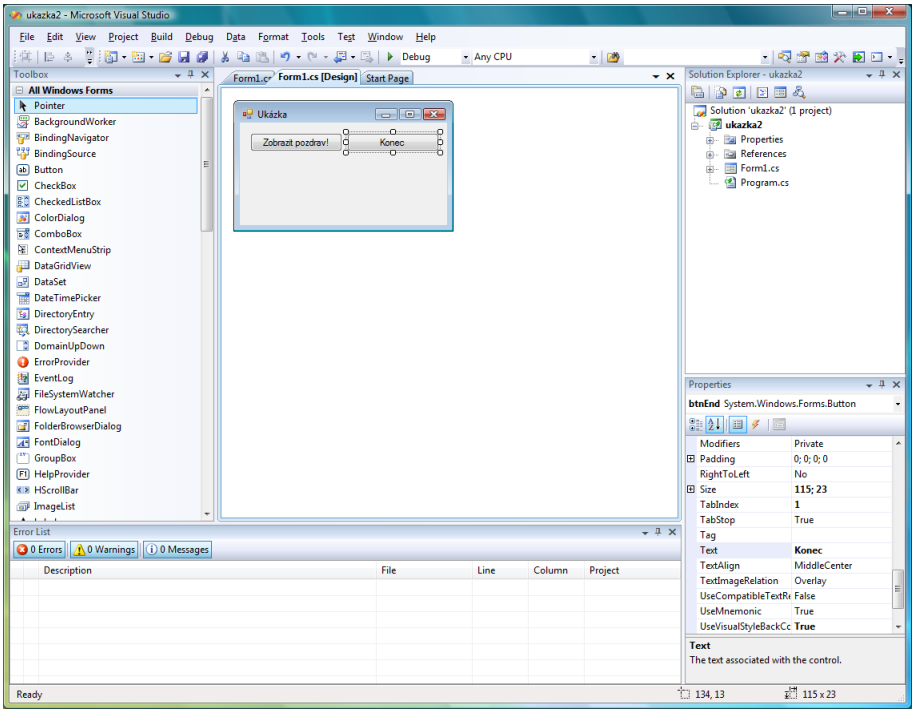
V pravé části okna jsou umístěny dva důležité bloky: *Solution explorer* (6) a *Properties* (8). *Solution explorer* (neboli „Průzkumník řešení“) zobrazuje strukturu vytvořeného projektu. Najdeme zde zejména seznam souborů zahrnutých do projektu. V bloku *Properties* lze nastavit vlastnosti jednotlivých objektů.

Posledním pomocným blokem je Error list (7). Error list je umístěn ve spodní části okna a zobrazuje chybové zprávy, které jsou rozděleny do tří kategorií:

- Chyba (Error) – nejvyšší důležitost. Pokud projekt obsahuje chybu, nelze sestavit a spustit.
- Varování (Warning) – nižší důležitost. Projekt lze sestavit a spustit, ale může docházet k chybnému chování programu.
- Informační zprávy (Messages) – nejnižší důležitost. Zprávy mají pouze informativní charakter.



Obr. 12. MS Visual Studio 2008 – soubor otevřený v textovém režimu



Obr. 13. MS Visual Studio 2008 – soubor otevřený v návrhovém režimu

4.1.2 Menu

Struktura menu vývojového nástroje MS Visual Studio:

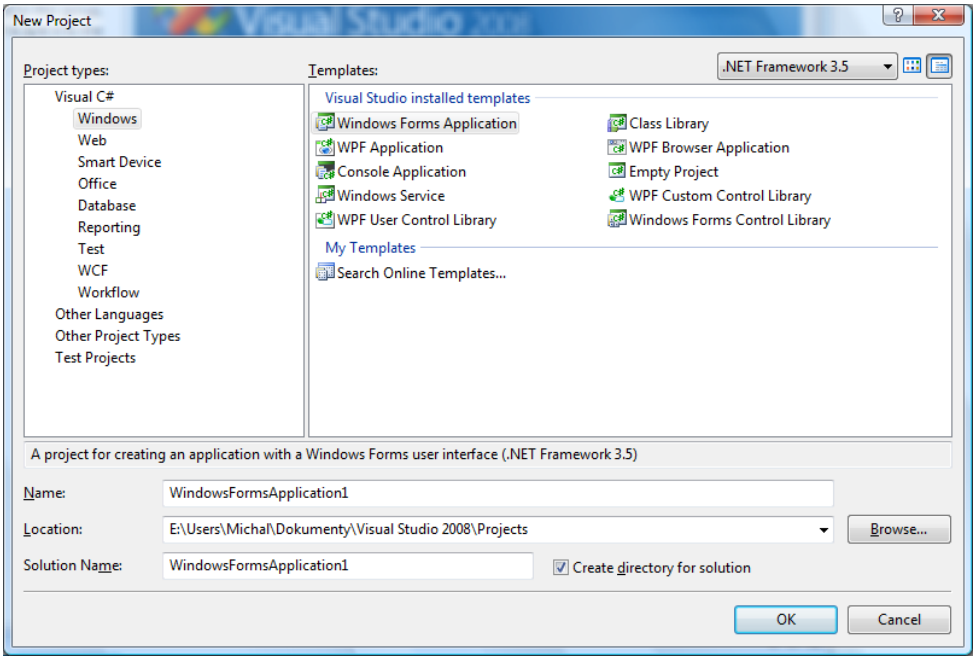
- **File** (Soubor) – práce se soubory – vytváření, otevírání a zavírání projektů nebo souborů.
- **Edit** (Upravit) – obsahuje operace vpřed a zpět, operace pro práci se schránkou, funkce najít a nahradit.
- **View** (Zobrazit) – obsahuje přepínání mezi textovým editorem a návrhářem, nastavení zobrazení jednotlivých pomocných oken.
- **Refactor**[9] (Refaktorizace) – extrahování metod z existujícího kódu a další funkce.
- **Project** (Projekt) – přidávání nových položek do projektu (formuláře, třídy, …) a nastavení vlastností projektu.
- **Build** (Sestavení) – konfigurace sestavení a vlastní sestavení řešení nebo projektu.
- **Debug** (Ladění) – nastavení a obsluha ladění programu.
- **Data** (Data) – práce s databázemi a jinými datovými sklady.
- **Format**[10] – nastavení formátovacích vlastností ovládacích prvků formuláře.

[9] Dostupné pouze v režimu textového editoru

- **Tools** (Nástroje) – připojení k zařízení nebo databázi, doplňky, makra, vlastnosti vývojového prostředí a další.
- **Test** (Test) – správa a provádění testování výsledného programu.
- **Window** (Okno) – nastavení zobrazení okna a přepínání mezi otevřenými soubory.
- **Help** (Nápověda) – přístup k nápovědě, dokumentaci MSDN a internetovým zdrojům.

4.1.3 Vytvoření projektu

Nový projekt lze vytvořit výběrem položky *„Project …“*, která se nachází v podmenu *„New“* … v menu *„File“*. Následně se zobrazí dialogové okno pro výběr šablony projektu a nastavení parametrů (obr. 14).



Obr. 14. Dialogové okno „New Project“

[10] Dostupné pouze v režimu návrháře

V případě použití Visual Studia 2008 je v horní části okna možné vybrat verzi .NET Frameworku, která má být použita[11]. Uživateli je nabídnuto několik možností:

- MS .NET Framework 2.0
- MS .NET Framework 3.0
- MS .NET Framework 3.5

Další část tohoto okna je tvořena výběrem šablony. V levé části je zobrazen seznam kategorií, pravou část tvoří výpis šablon z aktuální kategorie.

Následuje možnost nastavení následujících parametrů:

- Jméno projektu
- Umístění projektu
- Jméno řešení (řešení může být tvořeno několika projekty)
- Možnost vytvoření samostatné složky pro dané řešení

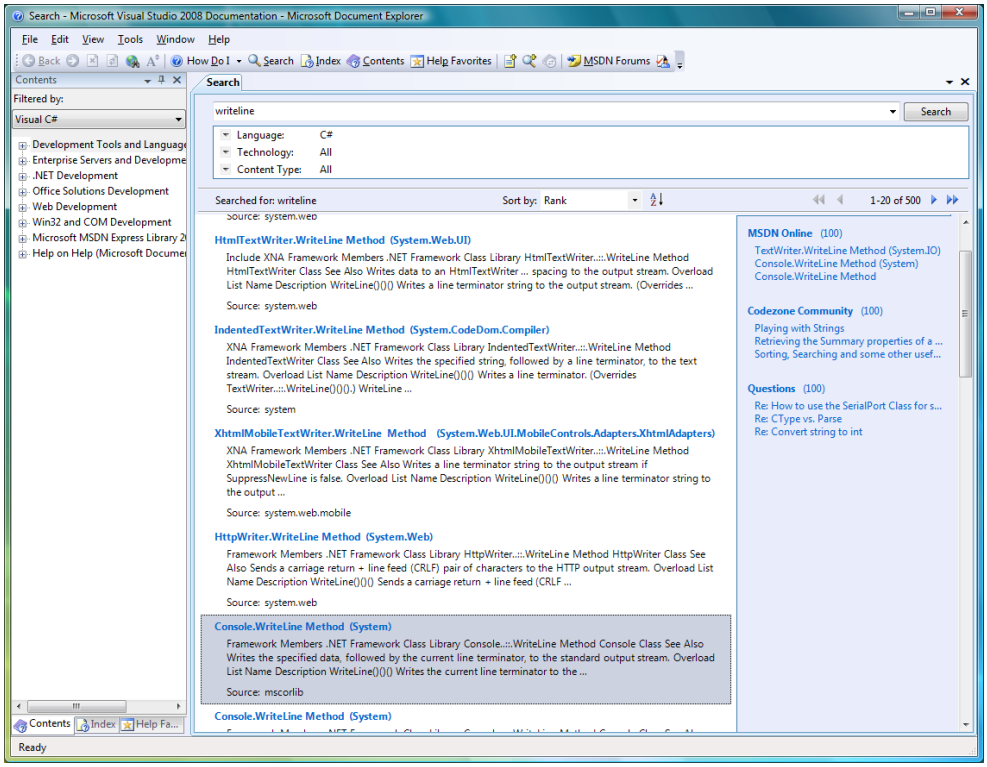
Na vysvětlení všech šablon projektu zde není dostatečný prostor, je ale vhodné zmínit ty nejčastěji používané:

- **Windows Forms Application** - vytvoření aplikace, kde je grafické rozhraní definováno pomocí technologie Windows Forms (technologii Windows Forms se věnuje kapitola 3.10)
- **WPF Application**[12] – vytvoření aplikace, kde je grafické rozhraní definováno pomocí technologie XAML (Windows Presentation Foundation je součástí .NET Framework 3.0)
- **Console Application** – vytvoření konzolové aplikace
- **Windows Service** – vytvoření služby systému MS Windows
- **Empty Project** – vytvoření prázdného projektu

[11] Tato možnost není k dispozici v MS Visual Studio 2005. V tomto případě je automaticky použito MS .NET Framework 2.0

[12] K dispozici pouze v MS Visual Studio 2008

4.1.4 Nápověda



Obr. 15. MSDN Library for Visual Studio

Nespornou výhodou vývojového prostředí MS Visual Studio je nápověda. Spolu s aplikací je totiž možno nainstalovat kompletní dokumentaci, která je přiložena na instalačním DVD.

Nápověda je distribuována pod názvem *MSDN Library for Visual Studio* a obsahuje kompletní nápovědu k aplikaci a kompletní dokumentaci platformy .NET včetně dokumentace programovacích jazyků. Samozřejmostí je řada příkladů použití a možnost vyhledávání (obr. 15). Vyhledávat lze v lokální nápovědě i v online zdrojích na Internetu.

4.2 MS Visual Studio Express

Mimo běžného vývojového nástroje MS Visual Studio nabízí společnost Microsoft také tzv. Express verzi[13]. Její výhodou je to, že je zcela zdarma i pro komerční využití.

MS Visual Studio Express se však liší ve dvou vlastnostech.

[13] K dispozici ke stažení na <http://www.microsoft.com/express/>

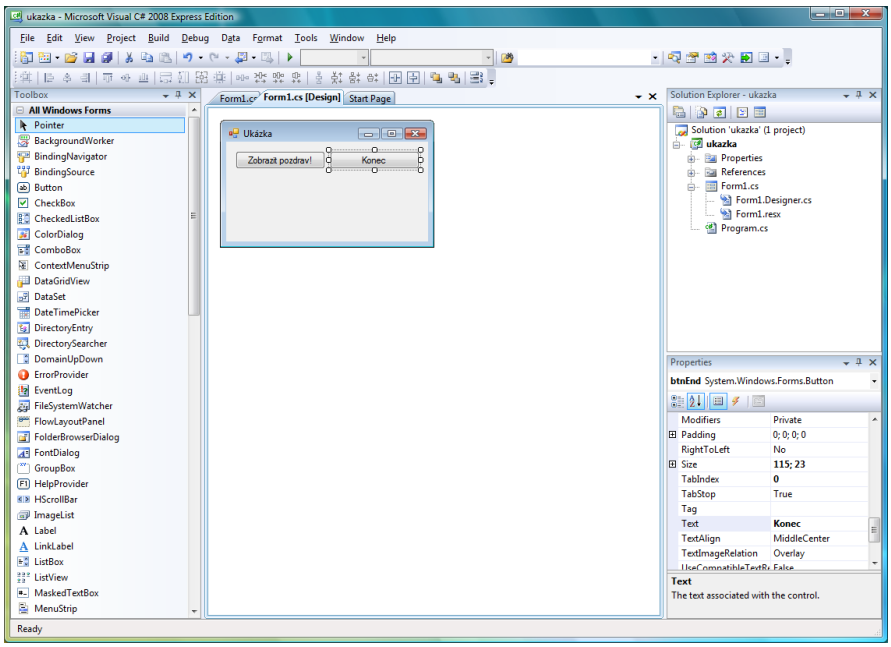
Prvním rozdílem je rozdělení Visual Studia na několik produktů dle programovacích jazyků. Na webu společnosti Microsoft lze nalézt tyto produkty:

- Visual Basic 2008 Express Edition
- Visual C# 2008 Express Edition
- Visual Web Developer Express Edition
- Visual C++ Express Edition
- SQL Server Express Edition

Druhý rozdíl spočívá v omezení funkčnosti oproti plnému MS Visual Studio. V Express verzi nelze využívat následující funkce [17]:

- Nástroje pro podporu vzdálených databází
- Rozšiřitelné IDE prostředí s možností doinstalace standardních IDE nadstaveb typu XY add-on, SDK, atd.
- Návrhář tříd
- Podpora mobilních zařízení
- 64-bitový překladač

Jak je patrné z níže uvedeného obrázku (obr. 16), pracovní prostředí tohoto vývojového nástroje je téměř totožné s MS Visual Studio 2008, proto zde nebude znovu popisováno.



Obr. 16. Okno aplikace MS Visual Studio 2008 Express Edition

4.3 Další vývojové nástroje

Kromě vývojových prostředí od společnosti Microsoft, která byla popsána na předcházejících stránkách, existuje i řada alternativních nástrojů, které jsou mnohdy zdarma. Uživatelské rozhraní je u těchto nástrojů velmi podobné, proto zde není dále popisováno.

Je ale vhodné zmínit příklady těchto vývojových prostředí:

- **SharpDevelop**[14] – open source vývojové prostředí pro C#.
- **Turbo C# Explorer** – vývojové prostředí od společnosti Borland, které není nadále vyvíjeno.
- **MonoDevelop**[15] – vývojové prostředí projektu Mono (možnost použití technologie .NET v operačních systémech GNU/Linux).

[14] Aktuální verze k dispozici na adrese <http://www.sharpdevelop.com/OpenSource/SD/>
[15] Domovská stránka projektu na adrese <http://www.monodevelop.com/Main_Page>

5 DIGITALIZACE PLOŠNÉHO GRAFU KŘIVKY

5.1 Obecné principy digitalizace

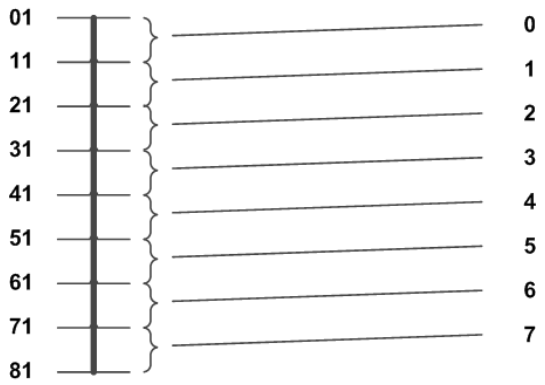
Digitalizace je proces, ve kterém dochází k přechodu od spojité funkce f(x, y) k diskrétní funkci I(x, y), a to jak v definičním oboru této funkce, tak v jejím oboru hodnot. [10]

Digitalizace má dvě základní fáze:

- Kvantování
- Vzorkování

5.1.1 Kvantování

Principem kvantování je diskretizace oboru hodnot obrazové funkce (obr. 17). Obor hodnot se rozdělí na intervaly, jimž je pak přidělena jedna zástupná hodnota (viz. obr. 17). [10]



Obr. 17. Kvantování

Pro výběr zástupné hodnoty lze zvolit jeden z několika přístupů:

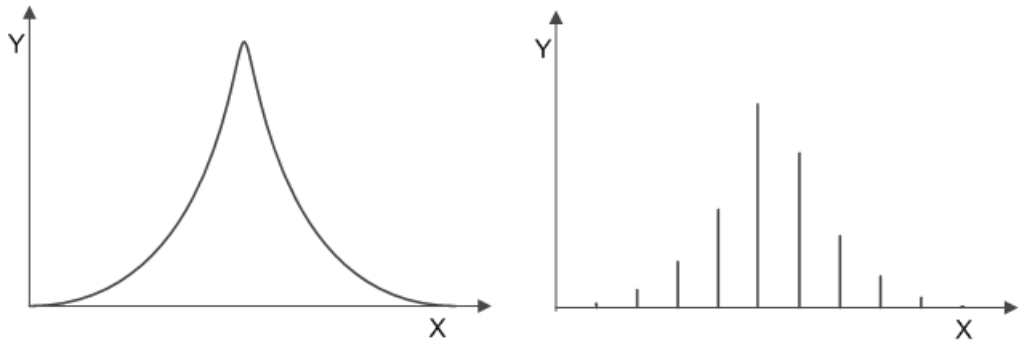
- Průměr hodnot z celého intervalu
- Vážený průměr
- Medián
- Průměr z hodnot na okraji intervalu

Při kvantování může docházet k tzv. kvantizační chybě, která způsobuje náhlé změny barev, a jimi způsobený výskyt hran.

5.1.2 Vzorkování

Vzorkování (*sampling*) představuje zjišťování hodnot v pravidelném intervalu. Lze tak chápat i přechod od spojitého na diskrétní případ (obr. 18), čímž se získá nová funkce I(x, y). Zde se vychází z toho, že pixel není bod, ale plocha o nenulové velikosti, kterou reprezentuje jedna hodnota. V tomto případě se mluví o bodovém vzorkování (*point sampling*).

Méně často se lze setkat s tzv. plošným vzorkováním, kdy se hodnota přiřazuje celé větší oblasti. V tomto případě se však jedná o mnohem výpočetně náročnější metodu, je totiž použita integrace na ploše. Z tohoto důvodu se v praxi plošné vzorkování aproximuje několika body, které jsou rozprostřeny uvnitř plochy funkce f(x, y). Tato metoda se nazývá *supersampling*. [8]



Obr. 18. Vzorkování

Pro frekvenci vzorkování platí Shannonův vzorkovací teorém, který říká, že vzorkovací frekvence má být větší než dvojnásobek nejvyšší frekvence obsažené v signálu. [10]

Obecně se dá říct, že čím je vzorkovací frekvence vyšší, tím je reprezentace původního signálu přesnější. Vyšší frekvence vzorkování však může způsobit větší paměťovou náročnost aplikace.

5.2 Digitalizace plošného grafu křivky

Digitalizace plošného grafu křivky využívá základní principy digitalizace, které byly popsány v předchozích kapitolách. Algoritmy jsou pouze aplikovány na digitalizaci průběhu křivky v rámci grafu. Jednotlivým algoritmům této digitalizace se věnují kapitoly 8.3 a 8.4.

II. PRAKTICKÁ ČÁST

6 EXISTUJÍCÍ SOFTWARE

Pro digitalizaci grafů existuje řada programů. Některé z nich jsou šířeny zdarma, dokonce i s otevřeným kódem.

V následujících několika kapitolách budou popsány tyto programy:

- Plot Digitizer
- xyExtract Graph Digitizer
- GetData Graph Digitizer
- Logic Graph Digitizing Software

6.1 Plot Digitizer

Prvním programem určeným na digitalizaci plošného grafu je Plot Digitizer. Tento program, který má za sebou 4 roky vývoje, byl napsán v jazyce JAVA. Jeho největší výhodou je fakt, že je šířen zdarma[16], a to včetně zdrojových kódů. Další výhodou je multiplatformnost. Plot Digitizer je totiž k dispozici pro operační systémy MS Windows, GNU/Linux i MacOs. [18]

Obr. 19. Program Plot Digitizer se zobrazeným dialogem „O Programu“

[16] K dispozici ke stažení na oficiálních stránkách projektu: < http://plotdigitizer.sourceforge.net/index.html >

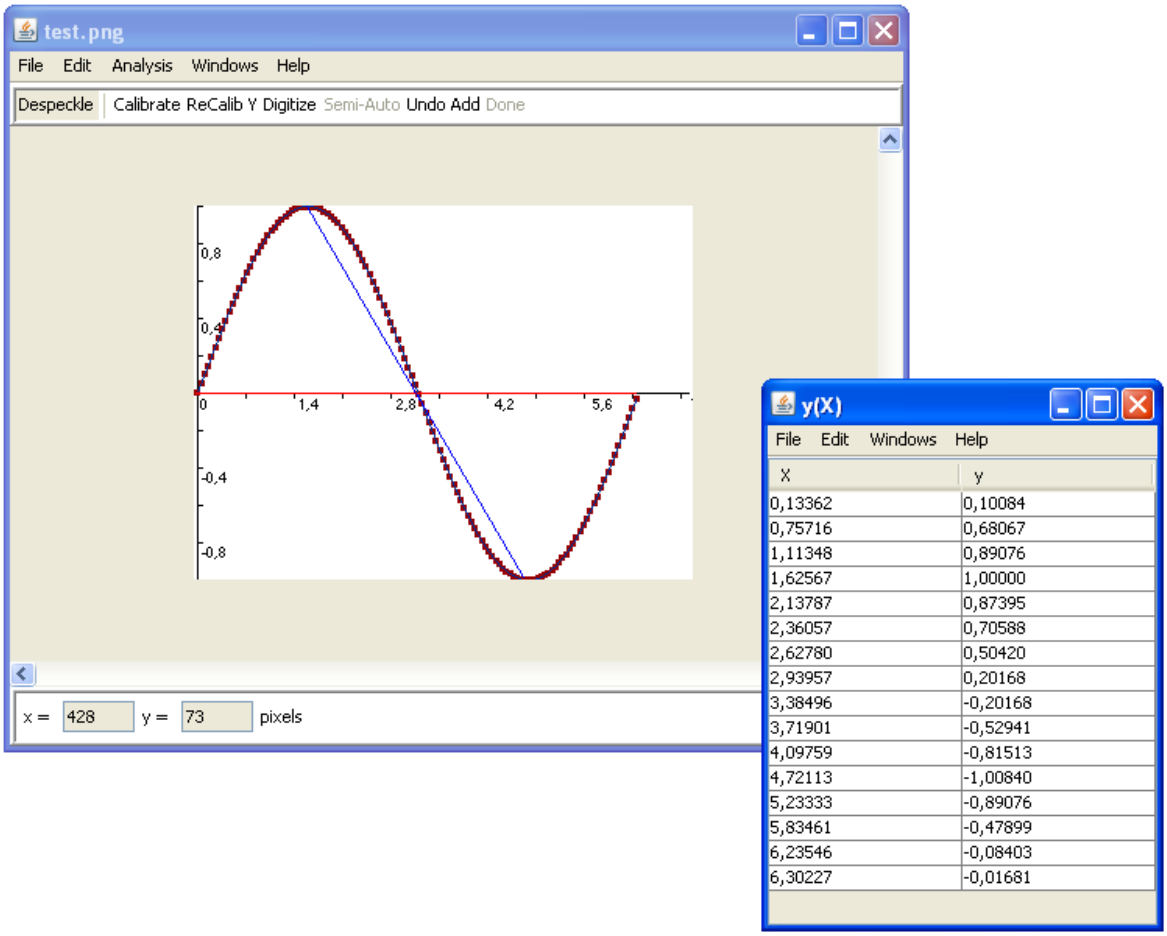
Další řádky se již budou věnovat popisu prostředí programu. Po spuštění přivítá uživatele poměrně jednoduché prostředí programu (obr. 19), které se skládá z menu, plochy pro zobrazení grafu a stavového řádku.

Nyní je vhodné se krátce podívat na ovládání tohoto programu. Graf lze načíst pouze z rastrového obrázku (menu *File* → *Open*). Následně je graf zobrazen na ploše aplikace.

Digitalizace bohužel neprobíhá zcela automaticky, ale je nutné nastavit několik parametrů. Prvními dvěma parametry jsou minimální a maximální hodnota na ose X. Tyto hodnoty se nastaví kliknutím myši na daná místa v grafu a zapsáním hodnot, kterých tyto body nabývají. Následně se stejným způsobem nastaví i minimální a maximální hodnota Y.

Dále zbývají ještě 2 parametry, a to symbolické označení nezávislé proměnné (hodnoty na ose X) a závislé proměnné (hodnoty na ose Y).

Po nastavení všech parametrů se může přistoupit k samotné digitalizaci. Jak už bylo zmíněno, digitalizace neprobíhá zcela automaticky. Jednotlivé body grafu, které mají být přidány do vektoru hodnot, musí uživatel vybrat myší. Následně kliknutím na tlačítko „Done“, dojde k uložení hodnot do tabulky. Tyto hodnoty lze následně uložit do textového dokumentu ve formátu CSV. Výsledek digitalizace je zobrazen na obr. 20.



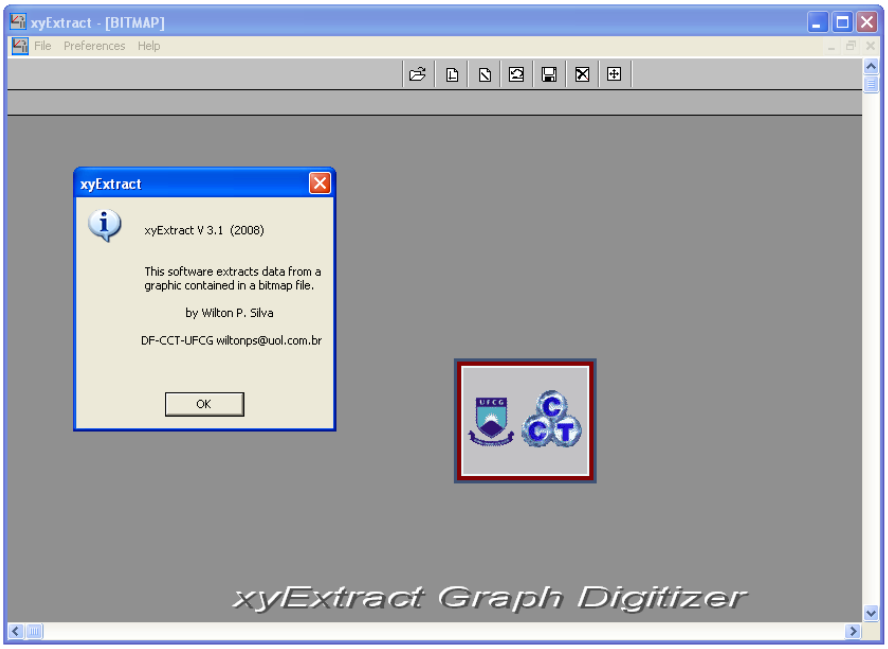
Obr. 20. Výsledek digitalizace provedené programem Plot Digitizer

6.2 xyExtract Graph Digitizer

Další možností digitalizace grafu je použití programu *xyExtract Graph Digitizer*. Ovládání programu je shodné s *Plot Digitizerem*, který byl popsán v předchozí kapitole. Jsou zde však dva rozdíly. Prvním rozdílem je mírně odlišné uživatelské rozhraní (obr. 21). Druhý rozdíl je mnohem důležitější. Tímto rozdílem je cena. Ta byla u tohoto programu stanovena na 25 USD[17].

Nyní však již k samotnému ovládání programu. Po spuštění přivítá uživatele poměrně jednoduché prostředí programu (obr. 21), které se skládá z menu, nástrojové lišty s tlačítky a plochy pro zobrazení grafu.

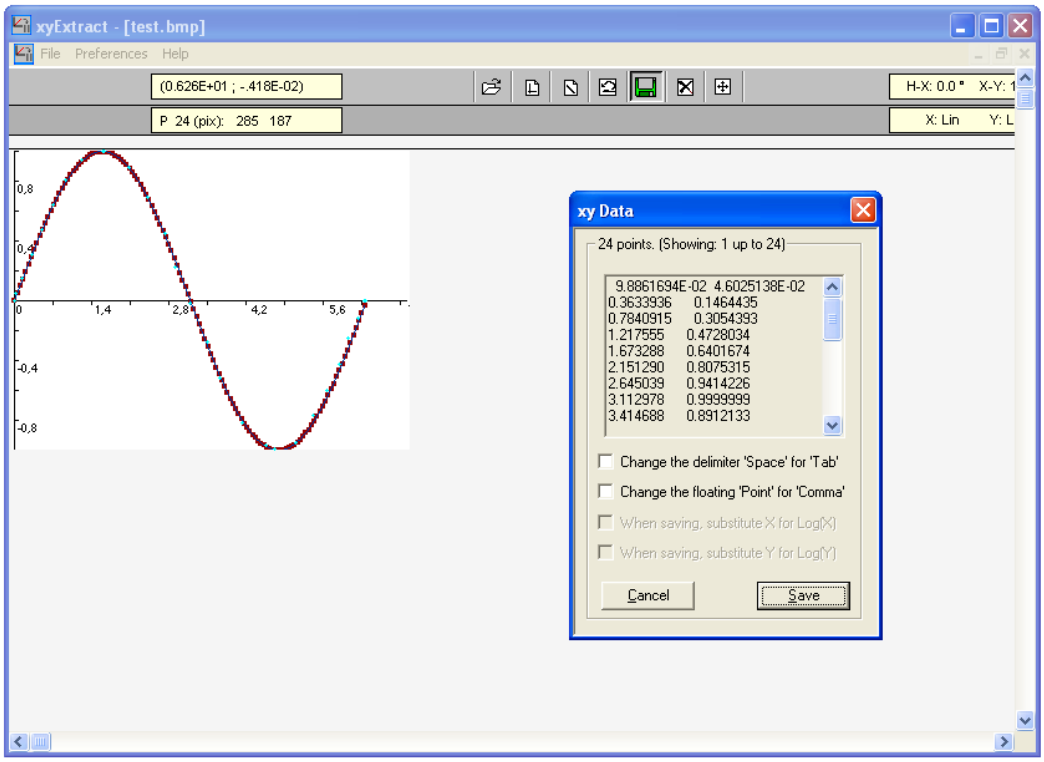
[17] Demoverze omezená na 40 spuštění je k dispozici ke stažení na oficiálních stránkách projektu: < http://www.download3000.com/download-count-home-4178.html >



Obr. 21. Program xyExtract Graph Digitizer se zobrazeným dialogem „O Programu“

Graf lze načíst pouze z rastrového obrázku ve formátu BMP (menu *File* → *Open*). Následně je graf zobrazen na ploše aplikace.

Jak už bylo zmíněno, digitalizace probíhá stejně jako u programu *Plot Digitizer*. Nejprve je nutné nastavit maximální a minimální hodnotu na osách X a Y. Následně je možné přejít k výběru jednotlivých bodů grafu. Body, které mají být přidány do vektoru hodnot, musí uživatel vybrat myší. Následně kliknutím na tlačítko „Done“ dojde k uložení hodnot do tabulky. Tyto hodnoty lze následně uložit do textového dokumentu ve formátu CSV. Výsledek digitalizace je zobrazen na obr. 22. [23]



Obr. 22. Výsledek digitalizace provedené programem xyExtract Graph Digitizer

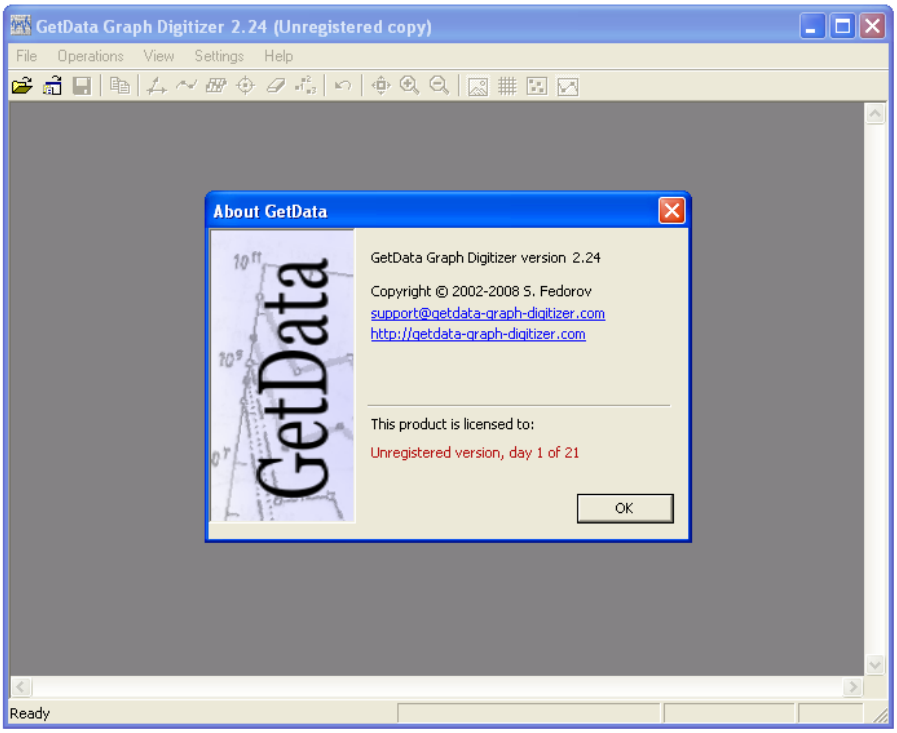
6.3 GetData Graph Digitizer

V pořadí třetím testovaným programem je program *GetData Graph Digitizer*. Stejně jako u předchozího programu, i v tomto případě se jedná o placený program. Jeho cena se pohybuje od 30 do 50 USD za jednu licenci[18].

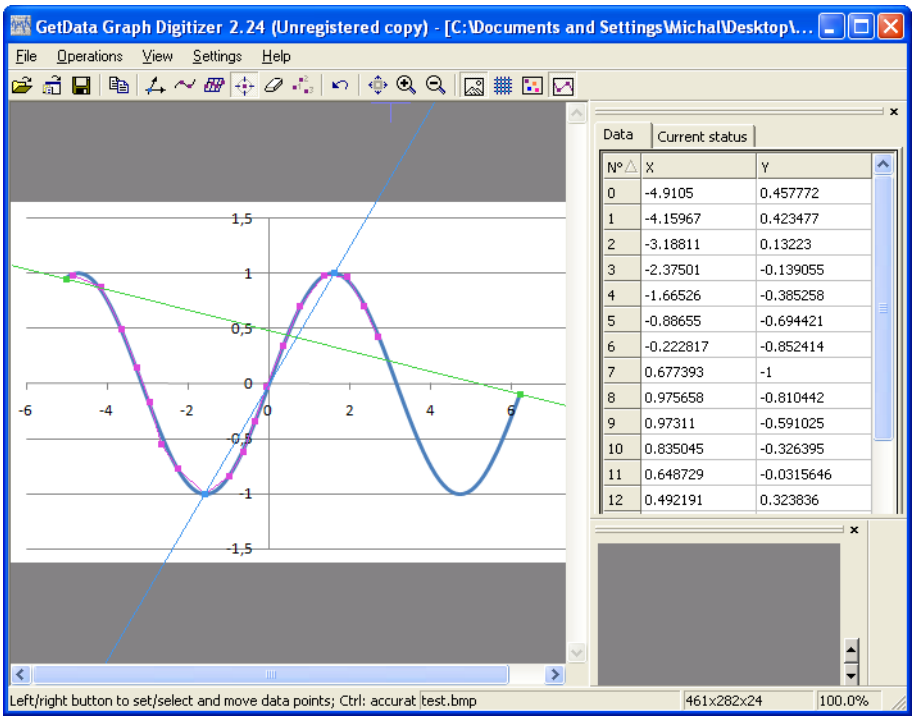
Po spuštění programu přivítá uživatele okno (obr. 23) s typickým rozložením ovládacích prvků, které je v různých obdobách použito i u programů popsaných v předchozí kapitole.

Samotná digitalizace má opět stejnou filosofii. Nejprve je nutné nadefinovat maximální a minimální hodnoty X a Y. Následně lze provést digitalizaci výběrem jednotlivých bodů grafu. Výsledek je poté zobrazen v přehledné tabulce (obr. 24). Narozdíl od předchozích aplikací zde program uživateli nabízí mnohem bohatší možnosti exportu dat. Kromě textových souborů je zde možnost exportu dat do tabulek MS Excel nebo do souborů XML. [13]

[18] Podrobný ceník včetně možnosti stažení testovací verze je k dispozici na adrese < http://getdata-graph-digitizer.com/index.php >



Obr. 23. Program GetData Graph Digitizer se zobrazeným dialogem „O Programu“



Obr. 24. Výsledek digitalizace provedené programem GetData Graph Digitizer

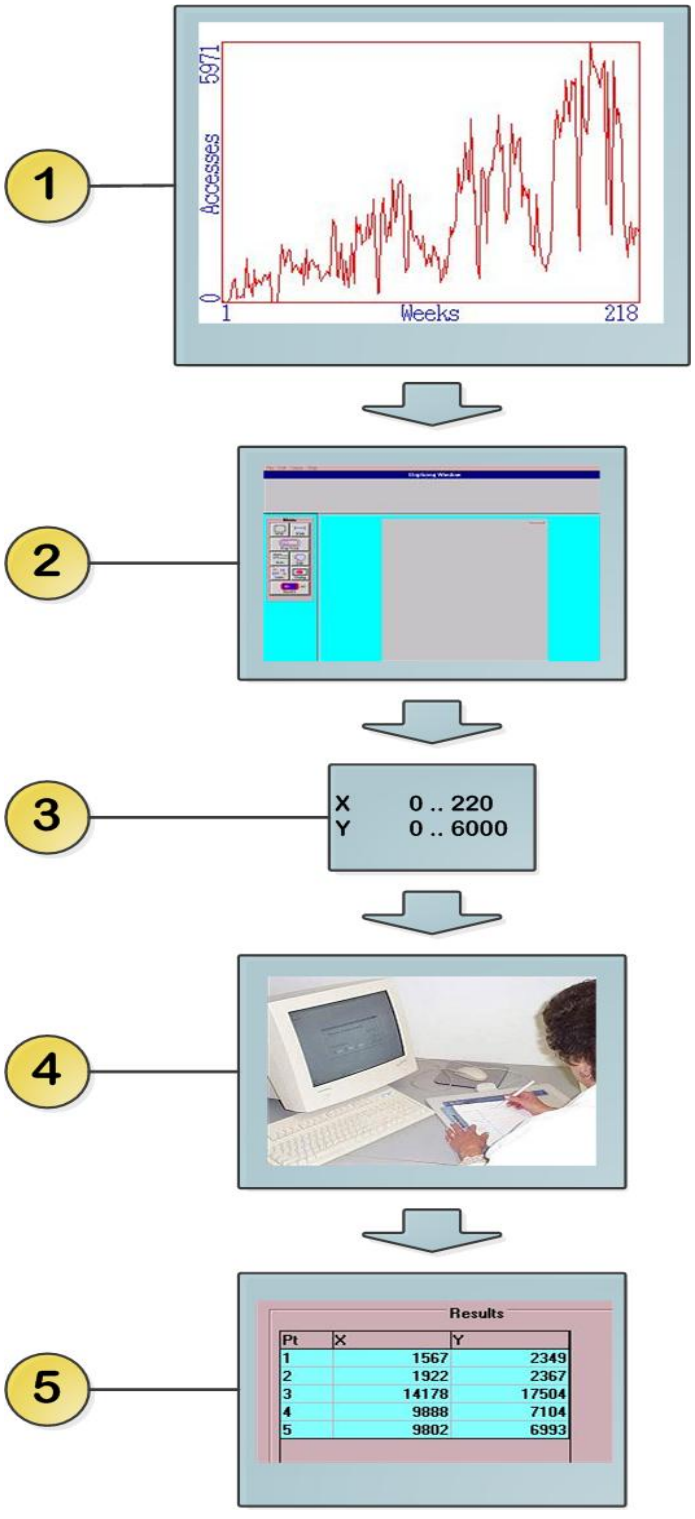
6.4 Logic Graph Digitizing Software

Posledním programem pro digitalizaci grafů, který zde bude zmíněn, je Logic Graph Digitizing Software. Tento program sice nebylo možno vyzkoušet v praxi, ale je natolik zajímavý, že je vhodné jej uvést v tomto krátkém přehledu. Jako vstupní zařízení je zde totiž využit tablet.

Postup digitalizace shrnuje schéma na obr. 25. Jako zdroj je použit graf v tištěné podobě (1)[19]. Následně je spuštěn program pro obsluhu tabletu (2) a jsou nastaveny rozsahy pro obě osy (3). Po nastavení všech parametrů je už možno na tabletu zachytit jednotlivé body grafu (4) a uložit je do tabulky (5).

Nevýhodou tohoto programu je nutnost použití tabletu a také jeho cena, která je výrobcem stanovena na 200 USD. [15]

[19] Čísla uvedená v závorkách odpovídají číslům na Obr. 25



Obr. 25. Postup digitalizace pomocí programu Logic Graph Digitizing Software

7 VYTVOŘENÉ KNIHOVNY A APLIKACE

V rámci této diplomové práce byla vytvořena aplikace pro digitalizaci grafu a s ní související dvě knihovny:

- *PlotPaint2D (*pro vykreslování grafů)
- *PlotDigitizer2D* (pro digitalizaci grafů)

7.1 Knihovna PlotPaint2D

První z vytvořených knihoven je knihovna *PlotPaint2D* pro vykreslování grafů. Důvodem pro vznik této knihovny je fakt, že .NET Framework neobsahuje knihovny pro tvorbu grafů.

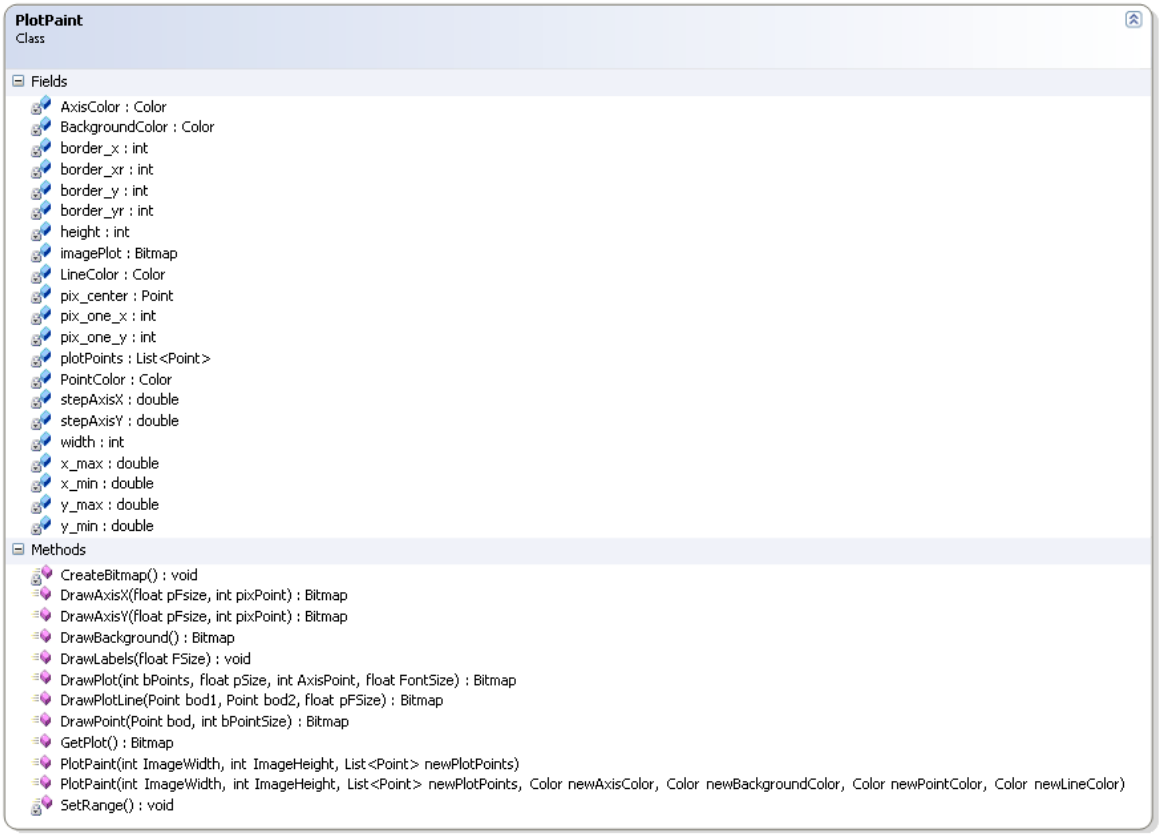
7.1.1 Distribuce a požadavky na vývojové prostředí

PlotPaint2D je distribuována formou jedné knihovny pojmenované *PlotPaint2D.dll*.

Tuto knihovnu lze používat ve vývojovém prostředí MS Visual Studio ve verzi 2005 nebo 2008 v operačním systému MS Windows s nainstalovanou knihovnou .NET Framework ve verzi 3.0 nebo 3.5.

7.1.2 Struktura tříd

Knihovna *PlotPaint2D* obsahuje pouze jedinou třídu, a to *PlotPaint* (obr. 26). Tato třída slouží pro vykreslení bodového grafu v osách *0xy*. Výsledný graf je uložen do instance třídy *Bitmap* (tato třída se nachází ve jmenném prostoru *System.Drawing*) a následně může být vykreslen nebo jinak zpracován.



Obr. 26. Struktura třídy PlotPaint

PlotPaint uživateli nabízí dvě možnosti volání konstruktoru. Prvním konstruktorem je metoda *PlotPaint(int ImageWidth, int ImageHeight, List<System.Windows.Point> newPlotPoints, Color newAxisColor, Color newBackgroundColor, Color newPointColor, Color newLineColor)*. Prvními dvěma vstupními parametry této metody jsou rozměry výsledné bitmapy v pixelech (celočíselné parametry *ImageWidth* a *ImageHeight*). Následuje seznam bodů, které mají být vykresleny v grafu. V tomto případě se jedná o pole, jehož prvky jsou body s reálnými souřadnicemi X a Y (kapitola 8.2). Poslední čtyři parametry obsahují definici barev (zapsány jako objekty třídy *System.Drawing.Color*, jejímuž popisu se věnuje kapitola 3.11.2). Barvy jsou zde definovány v následujícím pořadí:

- Barva pro vykreslení os souřadnic a jejich popisu (*newAxisColor*) – ve výchozím nastavení černá barva [A = 255, R = 0, G= 0, B = 0][20]

[20] V závorce uvedeny hodnoty jednotlivých parametrů struktury *System.Drawing.Color*. Význam parametrů: A – alfa kanál, R – červená složka, G – zelená složka, B – modrá složka.

- Barva pozadí (*newBackgroundColor*) – ve výchozím nastavení bílá barva [A = 255, R = 255, G= 255, B = 255][20]
- Barva pro vykreslení bodů grafu (*newPointColor*) – ve výchozím nastavení červená barva [A = 255, R = 153, G= 0, B = 0][20]
- Barva pro vykreslení čáry grafu (*newLineColor*) – ve výchozím nastavení modrá barva [A = 255, R = 0, G= 51, B = 153][20]

Druhá možnost použití konstruktoru vypadá velmi podobně, rozdíl je jen v tom, že uživatel nemusí definovat barvy. V tomto případě je graf vykreslen ve výchozích barvách (viz. předchozí odstavec). Tato metoda se zapisuje ve tvaru *PlotPaint(int ImageWidth, int ImageHeight, List<System.Windows.Point> newPlotPoints).*

Dále je vhodné zmínit i další metody a atributy této třídy. Všechny atributy shrnuje následující tabulka (tab. 25).

Tab. 25. Atributy třídy PlotPaint

Modifikátor	Datový typ	Název	Popis
private	*Color*	*AxisColor*	Barva pro vykreslování os souřadnic a jejich popisu
private	*Color*	*BackgroundColor*	Barva pozadí grafu
private	*Int32*	*border_x*	Levý okraj grafu v bitmapě
private	*Int32*	*border_xr*	Pravý okraj grafu v bitmapě
private	*Int32*	*border_y*	Horní okraj grafu v bitmapě
private	*Int32*	*border_yr*	Dolní okraj grafu v bitmapě
private	*Int32*	*height*	Výška generované bitové mapy
private	*Bitmap*	*imagePlot*	Objekt pro uložení vygenerovaného grafu
private	*Color*	*LineColor*	Barva čáry grafu
private	*Point*[21]	*pix_center*	Umístění průniku souřadnic X a Y na bitové mapě
private	*Int32*	*pix_one_x*	Počet pixel připadajících na změnu hodnoty na ose X o 1
private	*Int32*	*pix_one_y*	Počet pixel připadajících na změnu hodnoty na ose X o 1
private	*List<Point>*[21]	*plotPoints*	Pole pro uchování bodů pro vykreslení grafu
private	*Color*	*PointColor*	Barva pro vykreslení bodů v grafu
private	*double*	*stepAxisX*	Hodnota připadající na jednu jednotku na ose X
private	*double*	*stepAxisY*	Hodnota připadající na jednu jednotku na ose Y
private	*Int32*	*width*	Šířka generované bitové mapy
private	*double*	*x_max*	Maximální X-ová hodnota
private	*double*	*x_min*	Minimální X-ová hodnota
private	*double*	*y_max*	Maximální Y-ová hodnota
private	*double*	*y_min*	Minimální Y-ová hodnota

K úplnosti popisu třídy *PlotPaint* chybí již jen metody. Ty lze rozdělit do dvou skupin – na soukromé (*private*) a veřejné (*public*).

Soukromé metody jsou pouze dvě. První z nich, *CreateBitmap()*, slouží pouze pro vytvoření instance objektu bitové mapy. Druhá metoda je pojmenována *SetRange()* a jejím účelem je nastavení rozsahu a dalších vlastností vykreslovaného grafu. Obě zmíněné metody jsou volány automaticky konstruktorem objektu.

[21] Třída *Point* ze jmenného prostoru *System.Windows*

Zbývající metody jsou již všechny veřejné a ve všech případech se bude jednat o funkce s návratovou hodnotou typu *Bitmap*. První metoda slouží pouze pro vyplnění pozadí bitové mapy a zapisuje se ve tvaru *DrawBackground()*. Další dvojice (metody *DrawAxisX(Single pFsize, Int32 pixPoint*) a *DrawAxisY(Single pFsize, Int32 pixPoint)*) slouží pro vykreslení os X a Y. Obě funkce obsahují 2 parametry v tomto pořadí:

- Tloušťka čáry v pixelech.
- Velikost značek na ose v pixelech.

Metoda *DrawLabels(float FSize)* doplní do grafu popisky o parametrem definované velikosti písma.

O vykreslení vlastního bodu se postarají metody *DrawPoint(System.Windows.Point bod, Int32 bPointSize)* a *DrawPlotLine(System.Windows.Point bod1, System.Windows.Point bod2, Single pFSize).* První z nich vykreslí jeden bod s definovaným umístěním (System.Windows.Point) a velikostí bodu (v pixelech). Druhá zmíněná funkce slouží pro vykreslení úsečky, která je definována dvěma krajními body (System.Windows.Point) a tloušťkou čáry.

Metoda *DrawPlot(Int32 bPoints, Single pSize, Int32 AxisPoint, float FontSize)* bude ze zde zmiňovaných metod používána nejčastěji. Jedná se totiž o komplexní metodu, která automaticky provede všechny výše zmíněné metody pro vykreslení grafu. Na místa jejích parametrů je nutné dosadit následující hodnoty v pořadí, v jakém jsou zde uvedeny:

- Velikost bodů grafu (pokud bude 0, tak budou vykreslovány pouze čáry)
- Tloušťka čáry
- Velikost značek na osách
- Velikost písma

Poslední metoda, označená jako *GetPlot()*, najde využití zejména v případech, kdy uživatel potřebuje opětovně získat výslednou bitovou mapu. Tato funkce pouze vrací objekt typu *Bitmap*.

7.1.3 Ukázka použití

Kapitolu o knihovně PlotPaint2D zakončí krátká ukázka, která demonstruje vykreslení grafu. V ukázce se předpokládá existence pole s definicí jednotlivých bodů (*List<System.Windows.Point>*), které je v ukázce pojmenováno *body*, a formuláře s objektem typu *PictureBox* pojmenovaným jako *pbGraf*.

```
1 …
2 using PlotPaint2D;
3 …
4 Bitmap bmpGraf
5 PlotPaint Grafkvykresleni = new PlotPaint(pbGraf.Width,
  pbGraf.Height, body, Color.Black, Color.White, Color.Red,
  Color.Blue);
6 bmpGraf = Grafkvykresleni.DrawPlot(4,1,3,7);
7 pbGraf.Image = bmpGraf;
8 …
```

7.2 Knihovna PlotDigitizer2D

Druhou knihovnou, která byla vytvořena v rámci této diplomové práce, je knihovna *PlotDigitizer2D*. Tato knihovna řeší hlavní úkol této diplomové práce, a to samotnou digitalizaci grafu.

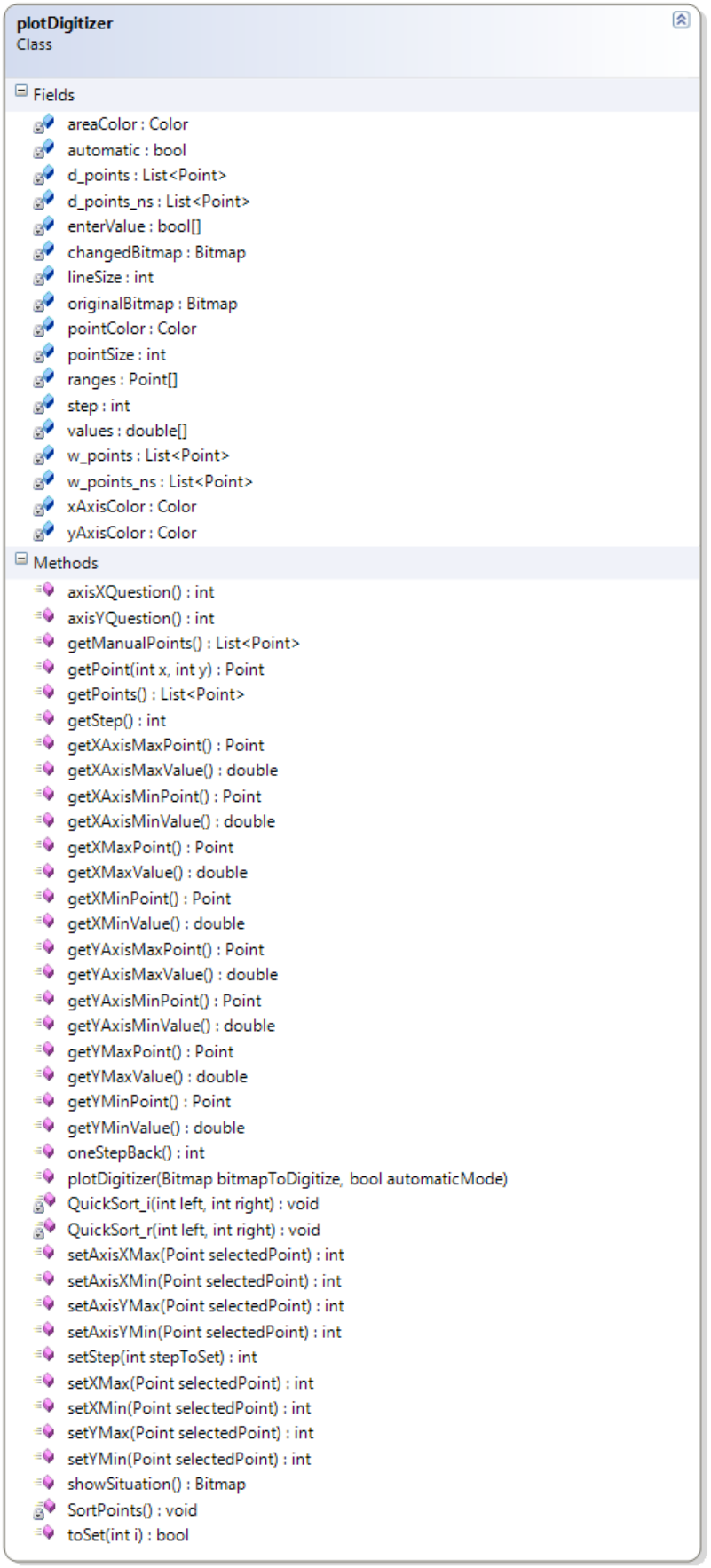
7.2.1 Distribuce a požadavky na vývojové prostředí

PlotDigitizer2D je distribuována formou jedné knihovny pojmenované *PlotDigitizer2D.dll*.

Tuto knihovnu lze používat ve vývojovém prostředí MS Visual Studio ve verzi 2005 nebo 2008 v operačním systému MS Windows s nainstalovanou knihovnou .NET Framework ve verzi 3.0 nebo 3.5.

7.2.2 Struktura tříd

Knihovna *PlotDigitizer* opět obsahuje pouze jednu třídu. Třída se v tomto případě jmenuje *plotDigitizer* (obr. 27). Tato třída slouží pro získání souřadnic bodů křivky, která je zobrazena v grafu v osách *0xy*. Výsledné hodnoty jsou uloženy do kolekce typu *List<System.Windows.Point>*. Takto získané hodnoty mohou být dále zpracovány.



Obr. 27. Struktura třídy plotDigitizer

Třída *plotDigitizer* uživateli nabízí pouze jednu variantu volání konstruktoru. Tou je metoda *plotDigitizer(Bitmap bitmapToDigitize, bool automaticMode).* Na místo prvního vstupního parametru je třeba doplnit instanci třídy Bitmap, jejímž obsahem je rastrový obrázek obsahující graf k digitalizaci. Na místo druhého parametru je třeba umístit logickou proměnnou (*bool*). Ta nabývá hodnoty *true* v případě automatické digitalizace nebo hodnoty *false* v případě ručního výběru bodů.

Dále je vhodné zmínit i další metody a atributy této třídy. Všechny atributy shrnuje následující tabulka (tab. 26).

Tab. 26. Atributy třídy PlotDigitizer

Modifikátor	Datový typ	Název	Popis
private	*Color*	*areaColor*	Barva neaktivní oblasti grafu.
private	*bool*	*automatic*	Logická hodnota, která určuje, zda proběhne automatická digitalizace (*true*) nebo bude použit ruční výběr bodů (*false*).
private	*List<Point>*[22]	*d_points*	Seřazený vektor souřadnic bodů k digitalizaci.
private	*List<Point>*[22]	*d_points_ns*	Neseřazený vektor souřadnic bodů k digitalizaci.
private	*double[]*	*values*	Vektor okrajových hodnot (význam jednotlivých hodnot vysvětlen dále v textu).
private	*Bitmap*	*changedBitmap*	Rastrový obrázek s grafem určeným k digitalizaci, který prošel dalším zpracováním pro zjednodušení automatické digitalizace.
private	*Bitmap*	*originalBitmap*	Původní rastrový obrázek s grafem určeným k digitalizaci.
private	*Color*	*pointColor*	Barva uživatelem vybraných bodů k digitalizaci.
private	*Point[]*[22]	*ranges*	Vektor okrajových souřadnic (význam jednotlivých hodnot vysvětlen dále v textu).
private	*int*	*step*	Ukazatel aktuálního kroku.
private	*int*	*lineSize*	Tloušťka spojnice uživatelem vybraných bodů k digitalizaci.
private	*int*	*pointSize*	Velikost uživatelem vybraných bodů k digitalizaci.
private	*List<Point>*[23]	*w_points*	Seřazený vektor reálných souřadnic digitalizovaných bodů.
private	*List<Point>*[23]	*w_points_ns*	Neseřazený vektor reálných souřadnic digitalizovaných bodů.
private	*Color*	*xAxisColor*	Barva uživatelem vyznačeného úseku osy X.
private	*Color*	*yAxisColor*	Barva uživatelem vyznačeného úseku osy Y.
private	*bool[]*	*enterValue*	Vektor logických hodnot, který určuje, zda má být daná hodnota vyžadována (význam jednotlivých hodnot vysvětlen dále v textu).

[22] V tomto případě použita třída *Point* ze jmenného prostoru *System.Drawing*.
[23] V tomto případě použita třída *Point* ze jmenného prostoru *System.Windows*.

Jak je patrné z výše uvedené tabulky, atributy *enterValue, ranges* a *values* jsou tvořeny vektory hodnot. Význam jednotlivých pozic v těchto polích vysvětluje tab. 27.

Tab. 27. Význam jednotlivých hodnot ve vektorech values, ranges, enterValue

Index	Popis
0	Počátek osy X (pokud je zobrazena)
1	Konec osy X (pokud je zobrazena)
2	Počátek osy Y (pokud je zobrazena)
3	Konec osy Y (pokud je zobrazena)
4	Minimální hodnota pro X
5	Maximální hodnota pro X
6	Minimální hodnota pro Y
7	Maximální hodnota pro Y

K úplnosti popisu třídy *PlotDigitizer* chybí již jen metody. Ty lze rozdělit do dvou skupin – na soukromé (*private*) a veřejné (*public*).

Soukromé metody jsou tři. První z nich, *SortPoints()*, slouží k seřazení položek v polích *d_points* a *w_points*. Pro řazení těchto hodnot byl vybrán algoritmus QuickSort (viz. kapitola 8.2.1), ten je definován ve zbylých dvou soukromých metodách. První slouží pro řazení dat v poli souřadnic v obrázku a nese jméno *QuickSort_i(int left, int right)*. Druhá je určena pro řazení reálných souřadnic grafu. Tato metoda je označena *QuickSort_r(int left, int right).*

Zbývající metody jsou již všechny veřejné a vzhledem k jejich většímu množství budou shrnuty v přehledné tabulce (tab. 28).

Tab. 28. Veřejné metody třídy PlotDigitizer

Datový typ návratové hodnoty	Název a parametry	Popis metody	Popis parametrů	
			Parametr	**Popis**
int	*axisXQuestion()*	Zobrazení dotazu, zda je v grafu zobrazena osa X. Návratovou hodnotou je index následujícího kroku.		
int	*axisYQuestion()*	Zobrazení dotazu, zda je v grafu zobrazena osa Y. Návratovou hodnotou je index následujícího kroku.		
List<Point>[24]	*getManualPoints()*	Vrátí pole všech získaných bodů.		
Point[24]	*getPoint(int x, int y)*	Vrátí reálné souřadnice bodu grafu na základě zadaných souřadnic bodu v obrázku.	*int x*	x-ová souřadnice bodu
			int y	y-ová souřadnice bodu
List<Point>[24]	*getPoints()*	Provedení automatické digitalizace grafu. Návratovou hodnotou je pole reálných souřadnic bodů grafu.		
int	*getStep()*	Vrátí index aktuálního kroku.		
Point[25]	*getXAxisMaxPoint()*	Vrátí souřadnice koncového (s maximální hodnotou) bodu na ose X.		
double	*getXAxisMaxValue()*	Vrátí X-ovou hodnotu v koncovém (s maximální hodnotou) bodě na ose X.		

[24] V tomto případě použita třída *Point* ze jmenného prostoru *System.Windows*.
[25] V tomto případě použita třída *Point* ze jmenného prostoru *System.Drawing*.

Datový typ návratové hodnoty	Název a parametry	Popis metody	Popis parametrů	
			Parametr	**Popis**
Point[25]	*getXAxisMinPoint()*	Vrátí souřadnice počátečního (s minimální hodnotou) bodu na ose X.		
double	*getXAxisMinValue()*	Vrátí X-ovou hodnotu v počátečním (s minimální hodnotou) bodě na ose X.		
Point[25]	*getXMaxPoint()*	Vrátí souřadnice maximální X-ové hodnoty.		
double	*getXMaxValue()*	Vrátí maximální X-ovou hodnotu.		
Point[25]	*getXMinPoint()*	Vrátí souřadnice minimální X-ové hodnoty.		
double	*getXMinValue()*	Vrátí minimální X-ovou hodnotu.		
Point[25]	*getYAxisMaxPoint()*	Vrátí souřadnice koncového (s maximální hodnotou) bodu na ose Y.		
double	*getYAxisMaxValue()*	Vrátí Y-ovou hodnotu v koncovém (s maximální hodnotou) bodě na ose Y.		
Point[25]	*getYAxisMinPoint()*	Vrátí souřadnice počátečního (s minimální hodnotou) bodu na ose Y.		
double	*getYAxisMinValue()*	Vrátí Y-ovou hodnotu v počátečním (s minimální hodnotou) bodě na ose Y.		
Point[25]	*getYMaxPoint()*	Vrátí souřadnice maximální Y-ové hodnoty.		
double	*getYMaxValue()*	Vrátí maximální Y-ovou hodnotu.		
Point[25]	*getYMinPoint()*	Vrátí souřadnice minimální Y-ové hodnoty.		
double	*getYMinValue()*	Vrátí minimální Y-ovou hodnotu.		
int	*setAxisXMax(Point selectedPoint)* [25]	Nastavení maximální hodnoty na ose X (koncový bod osy X). Návratovou hodnotou je index následujícího kroku.	*Point selectedPoint*	Souřadnice vybraného bodu obrázku
int	*setAxisXMin(Point selectedPoint)* [25]	Nastavení minimální hodnoty na ose X (počáteční bod osy X). Návratovou hodnotou je index následujícího kroku.	*Point selectedPoint*	Souřadnice vybraného bodu obrázku

Datový typ návratové hodnoty	Název a parametry	Popis metody	Popis parametrů	
			Parametr	**Popis**
int	*setAxisYMax(Point selectedPoint)*[25]	Nastavení maximální hodnoty na ose Y (koncový bod osy Y). Návratovou hodnotou je index následujícího kroku.	*Point selectedPoint*	Souřadnice vybraného bodu obrázku
int	*setAxisYMin(Point selectedPoint)*[25]	Nastavení minimální hodnoty na ose Y (počáteční bod osy Y). Návratovou hodnotou je index následujícího kroku.	*Point selectedPoint*	Souřadnice vybraného bodu obrázku
int	*setStep(int stepToSet)*	Nastavení indexu kroku, který má být vykonáván. Návratovou hodnotou je výsledný index kroku.	*int stepToSet*	Index kroku, který má být nastaven
int	*setXMax(Point selectedPoint)*[25]	Nastavení maximální X-ové hodnoty. Návratovou hodnotou je index následujícího kroku.	*Point selectedPoint*	Souřadnice vybraného bodu obrázku
int	*setXMin(Point selectedPoint)*[25]	Nastavení minimální X-ové hodnoty. Návratovou hodnotou je index následujícího kroku.	*Point selectedPoint*	Souřadnice vybraného bodu obrázku
int	*setYMax(Point selectedPoint)*[25]	Nastavení maximální Y-ové hodnoty. Návratovou hodnotou je index následujícího kroku.	*Point selectedPoint*	Souřadnice vybraného bodu obrázku
int	*setYMin(Point selectedPoint)*[25]	Nastavení minimální Y-ové hodnoty. Návratovou hodnotou je index následujícího kroku.	*Point selectedPoint*	Souřadnice vybraného bodu obrázku
Bitmap	*showSituation()*	Vrátí náhled obrázku se zakreslenými body, které byly digitalizovány.		
bool	*toSet(int i)*	Vrátí informaci, zda má být vyžadováno zadání hodnoty označené daným indexem.	*int i*	index hodnoty

7.2.3 Použití v režimu manuální digitalizace

V případě manuální digitalizace uživatel vybírá jednotlivé body, jejichž hodnoty mají být programem získány a dále zpracovány.

Nyní je vhodné se podívat, jak se v tomto případě postupuje.

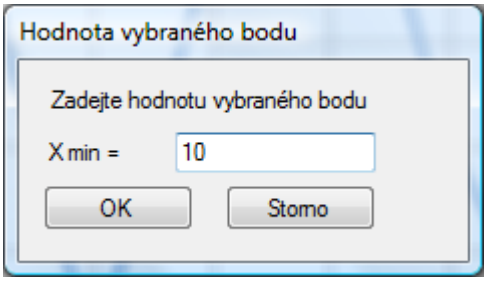
Na počátku je zavolán konstruktor, jehož prvním parametrem je objekt, který obsahuje obrázek s grafem, a druhý parametr je logická hodnota *false*. Volání konstruktoru tedy může vypadat následovně:

```
1 pdDigitalizace = new plotDigitizer(originalImage, false);
```

Následně je třeba definovat rozsahy hodnot souřadnic X a Y. Pro tuto akci je vhodné použít událost, kterou vyvolá kliknutí na dané místo obrázku (událost *Click*). Postupně je třeba zavolat 4 metody, jako například v níže uvedeném kódu:

```
2 pdDigitalizace.SetXMin(new System.Drawing.Point(0,100));
3 pdDigitalizace.SetXMax(new System.Drawing.Point(100,100));
4 pdDigitalizace.SetYMin(new System.Drawing.Point(0,100));
5 pdDigitalizace.SetYMax(new System.Drawing.Point(0,0));
```

Při volání každé z výše uvedených funkcí je zobrazen dialog pro zadání skutečné hodnoty X nebo Y (obr. 28).



Obr. 28. Dialog pro zadání hodnoty vybraného bodu

Následně již je možné vybírat body v obrázku (opět pomocí události *Click*) a program těmto bodům přiřadí skutečné souřadnice.

```
6 Bod = pdDigitalizace.getPoint(new System.Drawing.Point(50,50));
```

Tyto souřadnice jsou uloženy do kolekce, ze které je možné je získat pomocí následujícího řádku kódu:

```
7 Body = pdDigitalizace.getManualPoints();
```

7.2.4 Použití v režimu automatické digitalizace

V režimu automatické digitalizace jsou body grafu získány programem automaticky. Je nutné nastavit pouze krajní body grafu.

Pro správnou funkci této metody digitalizace musí být splněny dva předpoklady:

- Graf neobsahuje více průběhů.
- Mřížka grafu je méně výrazná než samotný průběh grafu.

Před spuštěním digitalizace je nutné nastavit umístění os X a Y voláním metod

```
1 pdDigitalizace.SetAxisXMin(new System.Drawing.Point(0,100));
2 pdDigitalizace.SetAxisXMax(new System.Drawing.Point(100,100));
```

pro osu X a metod

```
3 pdDigitalizace.SetAxisYMin(new System.Drawing.Point(0,100));
4 pdDigitalizace.SetAxisYMax(new System.Drawing.Point(0,0));
```

pro osu Y.

Pokud není zobrazena některá z os X a Y, je třeba volat jednu z dvojic metod

```
5 pdDigitalizace.SetXMin(new System.Drawing.Point(0,100));
6 pdDigitalizace.SetXMax(new System.Drawing.Point(100,100));
```

nebo

```
7 pdDigitalizace.SetYMin(new System.Drawing.Point(0,100));
8 pdDigitalizace.SetYMax(new System.Drawing.Point(0,0));
```

Při volání každé z výše uvedených funkcí je zobrazen dialog pro zadání skutečné hodnoty X nebo Y (obr. 28).

Poté, co jsou nastaveny všechny výše uvedené parametry, je možné spustit automatickou digitalizaci:

```
9 Body = pdDigitalizace.getPoints();
```

7.2.5 Příklad použití

Následující dvě kapitoly nastínily použití pouze přibližně. Pro lepší pochopení principu digitalizace grafu je vhodné seznámit se s podrobnějším příkladem. Tento příklad je, vzhledem ke svému většímu rozsahu, pouze na disku CD-ROM, který je přiložen k této diplomové práci.

7.3 Aplikace pro digitalizaci grafu

Program pro digitalizaci grafu aplikuje jednotlivé vytvořené knihovny v praxi. Představuje kompletní aplikaci, která umožňuje vytvoření grafu z ručně zadaných hodnot, nebo digitalizaci grafu. Následně je možné hodnoty a na jejich základě vykreslený graf exportovat do souboru.

7.3.1 Požadavky na hardware a software

Před samotným popisem aplikace je vhodné zmínit její požadavky na hardware a software. Jelikož se nejedná o nijak náročnou aplikaci, jsou tyto požadavky shodné s požadavky běhového prostředí MS .NET Framework, které musí být nainstalováno na počítači, na kterém má být aplikace používána. [16]

Minimální požadavky:

- Procesor: 400 MHz, Pentium kompatibilní
- Operační systém:
  - MS Windows Server 2003 SP 1 s doinstalovaným MS .NET Framework 3.0
  - MS Windows XP SP 2 s doinstalovaným MS .NET Framework 3.0
  - MS Windows Vista
- RAM: 96 MB
- Pevný disk: min. 500 MB
- Rozlišení monitoru: 800×600 px

Doporučené požadavky:

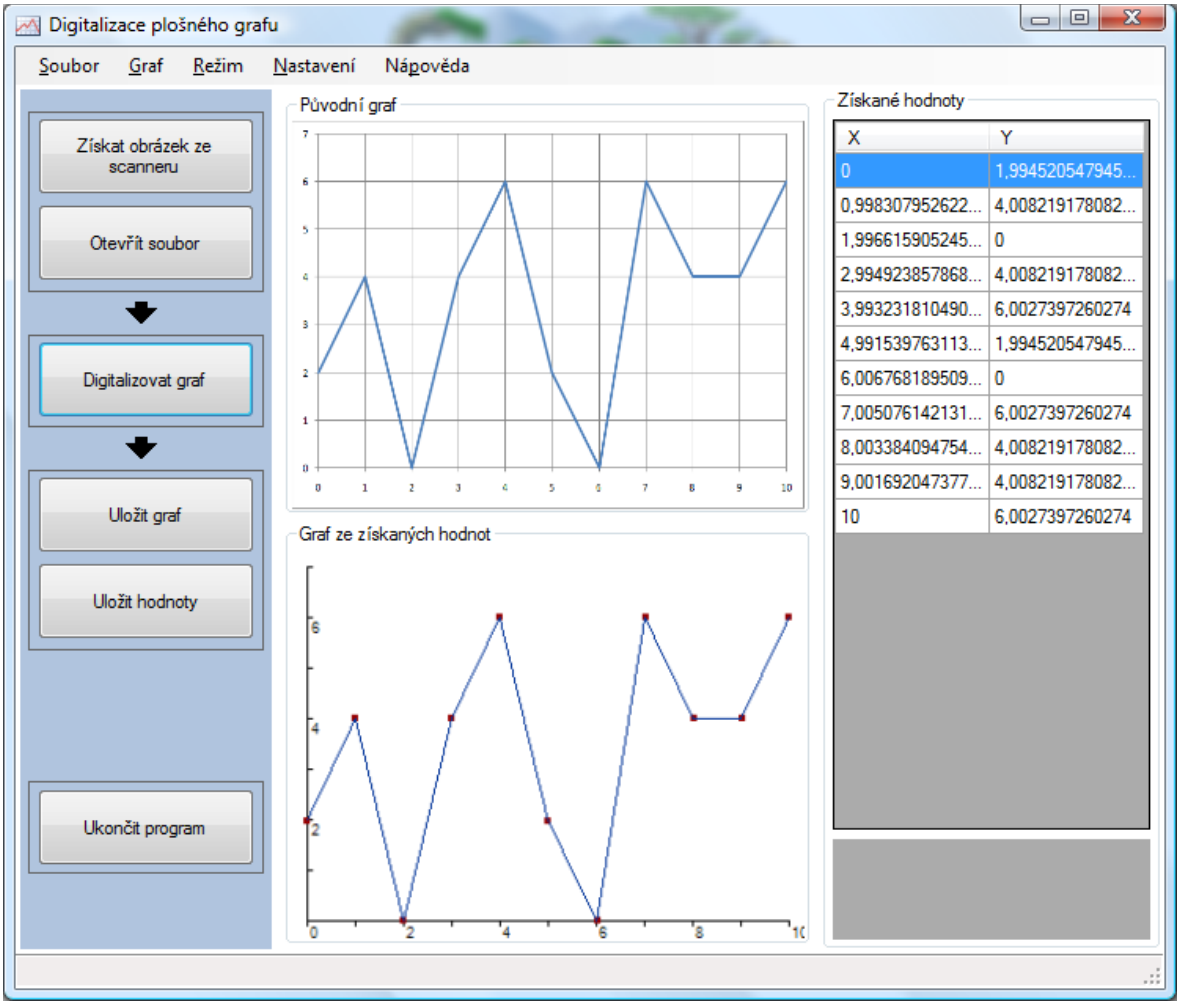
- Procesor: 1 GHz, Pentium kompatibilní
- Operační systém:
  - MS Windows Server 2003 SP 1 s doinstalovaným MS .NET Framework 3.0
  - MS Windows XP SP 2 s doinstalovaným MS .NET Framework 3.0
  - MS Windows Vista
- RAM: 256 MB
- Pevný disk: min. 500 MB
- Rozlišení monitoru: 1024×768 px
- Scanner (pouze v případě použití operačního systému Windows XP)

7.3.2 Instalace a spuštění programu

Před instalací programu je nutné ověřit, zda je na klientském počítači nainstalováno prostředí MS .NET Framework ve verzi 3 nebo vyšší (tuto kontrolu je nutné provést pouze v případě použití operačního systému MS Windows XP. Operační systém MS Windows Vista již tuto knihovnu obsahuje ve výchozím nastavení).

Po ověření přítomnosti výše uvedené knihovny již je možné nainstalovat vlastní aplikaci. Aplikace je distribuována formou archivu ZIP (konkrétně *digitalizacegrafu.zip*), který stačí rozbalit do libovolné složky. Poté již stačí jen spustit aplikaci otevřením souboru *digitalizacegrafu.exe*.

7.3.3 Popis pracovního prostředí



Obr. 29. Náhled prostředí aplikace „Digitalizace plošného grafu“

Pracovní prostředí aplikace „Digitalizace plošného grafu“ se skládá z menu, stavového řádku a vlastního prostředí aplikace. V levé části okna jsou zobrazena tlačítka, která představují nejčastěji používané funkce v pořadí, tak jak mají být volány.

Ve středu okna je prostor pro náhled původního grafu a náhled grafu vykresleného na základě získaných hodnot.

Napravo je zobrazena tabulka, ve které jsou umístěny jednotlivé souřadnice grafu.

7.3.4 Struktura menu

Menu programu má následující strukturu:

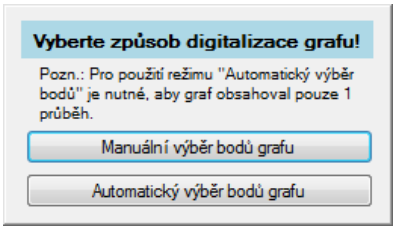
- **Soubor**
  - **Otevřít** – výběr a otevření souboru.
  - **Získat ze scanneru**[26] - získání obrazu ze scanneru, či jiného záznamového zařízení.
  - **Uložit graf** – uložení grafu, který byl vykreslen ze získaných hodnot, do souboru.
  - **Uložit hodnoty** – uložení získaných hodnot do souboru.
  - **Konec** – ukončení programu.
- **Graf**
  - **Digitalizovat graf**[27] - spuštění průvodce pro digitalizaci grafu (viz. kapitoly 7.3.5 a 7.3.6).
  - **Zobrazit graf**[28] - zobrazení grafu na základě zadaných hodnot.
- **Režim**
  - **Digitalizace grafu** – přepnutí programu do režimu „Digitalizace grafu“.
  - **Návrhář grafu** – přepnutí programu do režimu „Návrhář grafu“.
- **Předdefinované hodnoty**[28] – položky tohoto menu slouží pro vyplnění tabulky předdefinovanými hodnotami. Položka přítomna pro testovací účely.
  - **Náhodné hodnoty – funkce náhodných hodnot v intervalu <-8;8>.**
  - **Sin(x)** – funkce Sinus(x).
  - **Ln(x)** – přirozený logaritmus čísla x.

[26] Položka přístupná pouze v operačním systému Windows XP.
[27] Zobrazeno pouze v režimu „Digitalizace grafu“.
[28] Zobrazeno pouze v režimu „Návrhář grafu“.

- **Nastavení**
  - **Barvy** – nastavení barev pro vykreslení grafu.
- **Nápověda**
  - **O programu** – zobrazení informací o programu a počítači, na kterém je program nainstalován.

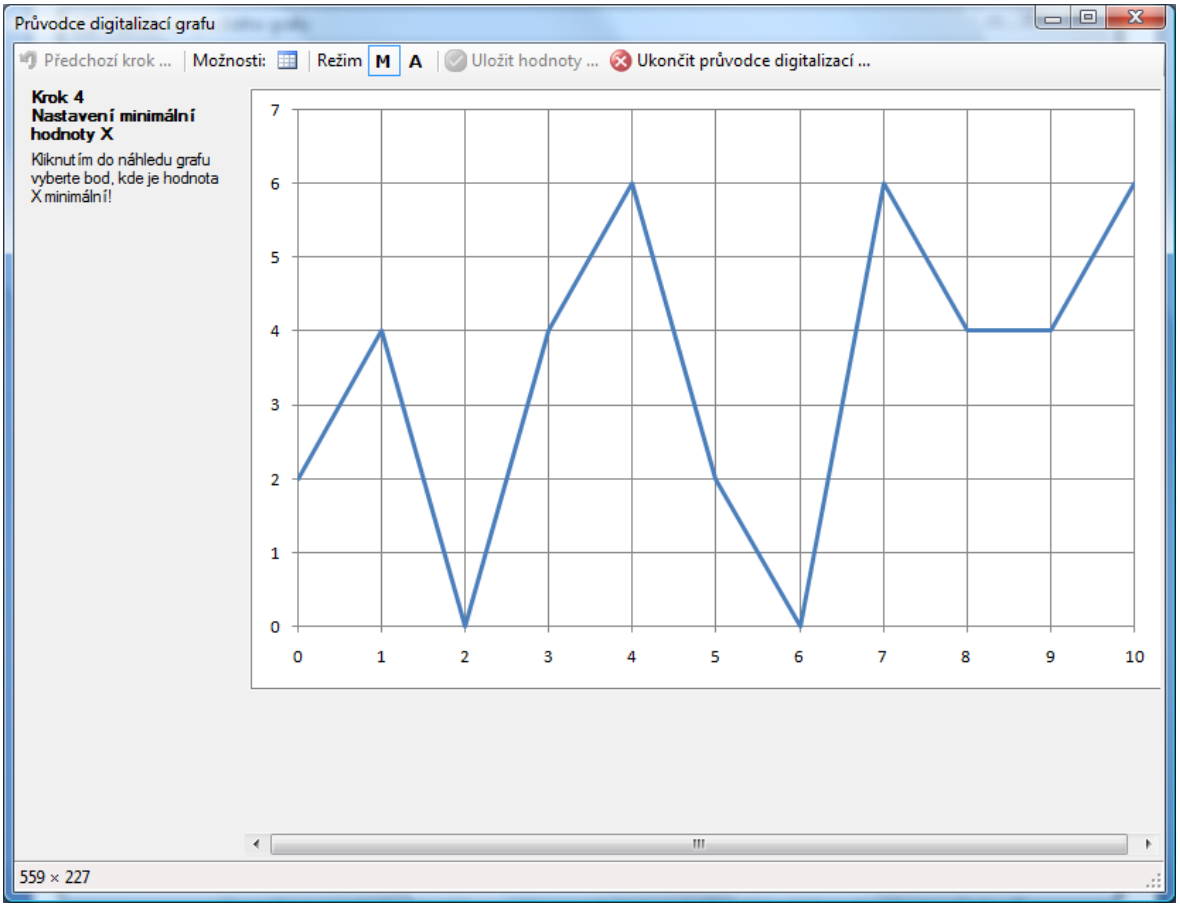
7.3.5 Použití v režimu manuální digitalizace

Do režimu manuální digitalizace uživatel vstoupí kliknutím na tlačítko „Digitalizovat graf“ a v následně zobrazeném okně (obr. 30) „Manuální výběr bodů grafu“.



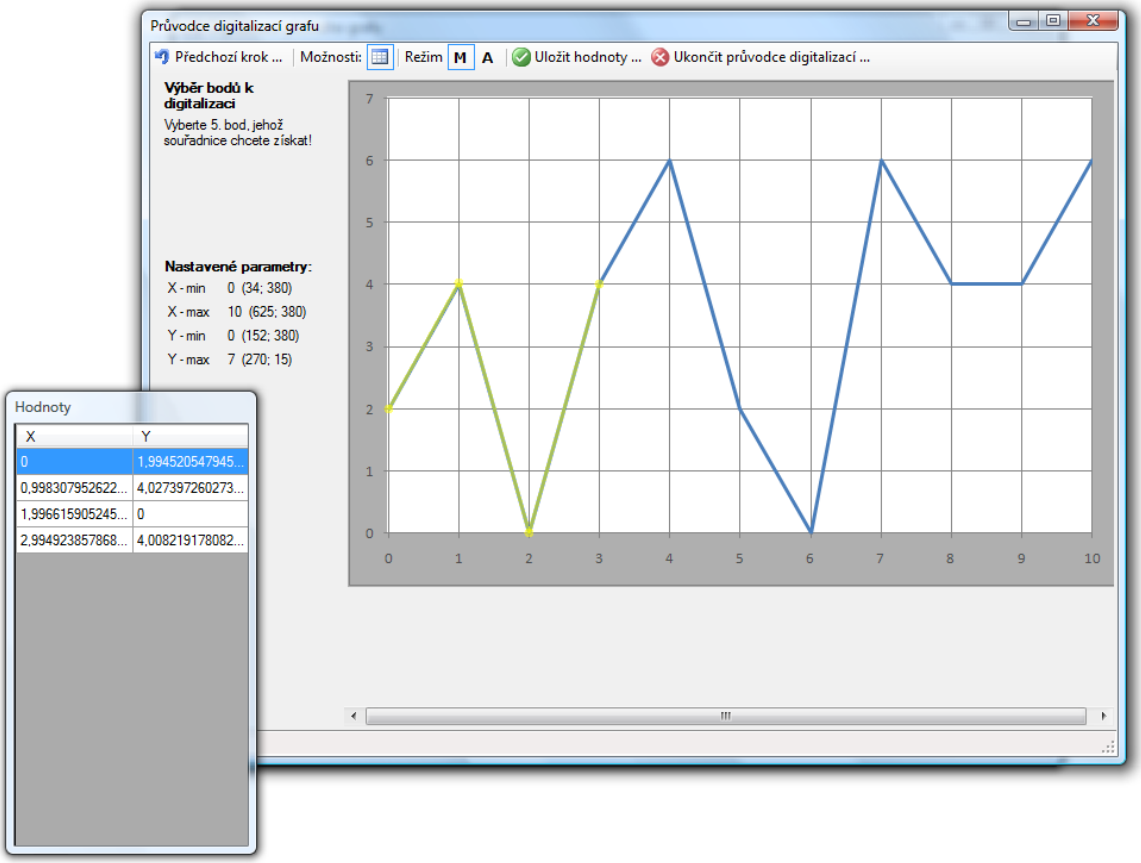
Obr. 30. Výběr způsobu digitalizace grafu

Po spuštění průvodce (obr. 31) je nutné postupně definovat minimální a maximální hodnoty na osách X a Y.



Obr. 31. Průvodce digitalizací grafu

Po zadání výše uvedených parametrů již je možné vybírat jednotlivé body k digitalizaci (obr. 32).



Obr. 32. Průběh manuální digitalizace grafu

Po výběru všech bodů lze průvodce ukončit kliknutím na tlačítko „Uložit hodnoty“.

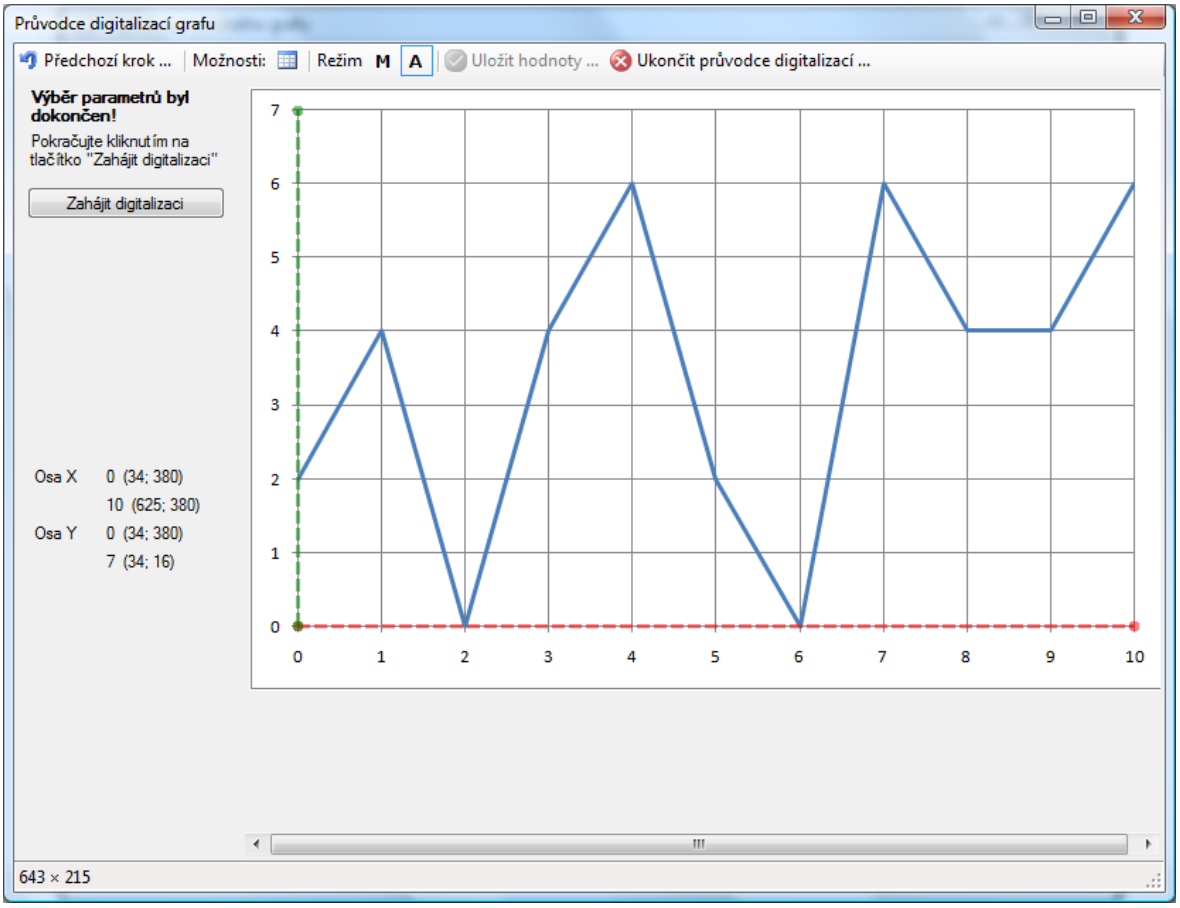
7.3.6 Použití v režimu automatické digitalizace

Do režimu automatické digitalizace uživatel vstoupí kliknutím na tlačítko „Digitalizovat graf“ a v následně zobrazeném okně (obr. 30) „Automatický výběr bodů grafu“. Pro správnou funkci tohoto režimu musí být splněny dva předpoklady:

- Graf neobsahuje více průběhů.
- Mřížka grafu je méně výrazná než samotný průběh grafu.

Na začátku je uživatel dotázán, zda graf obsahuje osu X. Pokud ano, je třeba vyznačit její počátek a konec a zadat jejich hodnoty. Pokud osa není zobrazena, je uživatel později vyzván k zadání maximální a minimální hodnoty X, stejně jako u manuální digitalizace. Stejný postup se opakuje i pro osu Y.

Po zadání potřebných hodnot je v aplikaci zobrazeno tlačítko „Zahájit digitalizaci“ (obr. 33). Po kliknutí na toto tlačítko je provedena digitalizace a zobrazena výsledná tabulka.



Obr. 33. Průběh automatické digitalizace grafu

7.3.7 Použití v režimu návrháře

Do režimu návrháře se aplikace přepne výběrem položky *Návrhář grafu* z menu *Režim*.

Po přepnutí do režimu návrháře se pod prostorem pro tabulku zobrazí 4 tlačítka (obr. 34), která umožní ruční přidávání a odebírání hodnot.

Obr. 34. Tlačítka pro ruční návrh grafu

Následně je možné graf zobrazit kliknutím na tlačítko *Zobrazit graf* v levé liště nebo v menu *Graf*.

7.3.8 Export dat

Ať už v režimu digitalizace nebo v režimu návrháře uživatel získá graf a tabulku. Tato data je vhodné exportovat do souboru, aby mohla být použita pro další zpracování.

Graf

V případě zobrazeného grafu jsou data ukládána do souboru formou rastrového obrázku. Uživateli se nabízí několik formátů obrázků, jejichž seznam shrnuje následující tabulka (tab. 29).

Tab. 29. Formáty obrázků, do nichž lze exportovat graf

Název	Popis
Bmp	Obrázek ve formátu bitmap (BMP).
Gif	Obrázek ve formátu Graphics Interchange Format (GIF).
Jpeg	Obrázek ve formátu Joint Photographic Experts Group (JPEG).
Png	Obrázek ve formátu W3C Portable Network Graphics (PNG).
Tiff	Obrázek ve formátu Tagged Image File Format (TIFF).

Tabulka

Pro export dat zobrazených v tabulce se uživateli nabízejí dvě možnosti:

- Soubor TXT
- Soubor CSV

První možnost uloží data do textového souboru. Hodnoty jsou v tomto souboru odděleny tabulátorem. Tato možnost je vhodná zejména v případě, kdy uživatel chce hodnoty archivovat a zobrazovat pouze v jednoduchých textových editorech (například *Poznámkový blok* v operačním systému MS Windows).

Při použití druhé možnosti (tj. soubor CSV) jsou data opět uložena do textového souboru. V tomto případě však jsou hodnoty oddělené středníkem. Použití této možnosti je vhodné, pokud uživatel chce dále zpracovávat data v tabulkovém procesoru (např. MS Excel). Většina tabulkových procesorů totiž podporuje otevírání souborů ve formátu CSV.

8 TECHNOLOGIE A ALGORITMY POUŽITÉ PŘI VÝVOJI PROGRAMU

Následující stránky se budou věnovat popisu některých pokročilejších algoritmů, které byly použity při vývoji jednotlivých knihoven a aplikace pro digitalizaci grafu.

8.1 Získání obrazových dat ze scanneru

Prvním řešeným problémem byla komunikace se scannerem. Technologie .NET Framework totiž nenabízí funkce pro komunikaci s tímto druhem zařízení.

V prostředí operačního systému MS Windows je však možno použít technologii Windows Image Acquision (WIA).

8.1.1 Popis a struktura technologie WIA

WIA byla uvedena s příchodem operačního systému MS Windows 98 a poskytuje API a ovladače pro komunikaci se scannery, webkamerami a jinými zařízeními pro záznam obrazu.

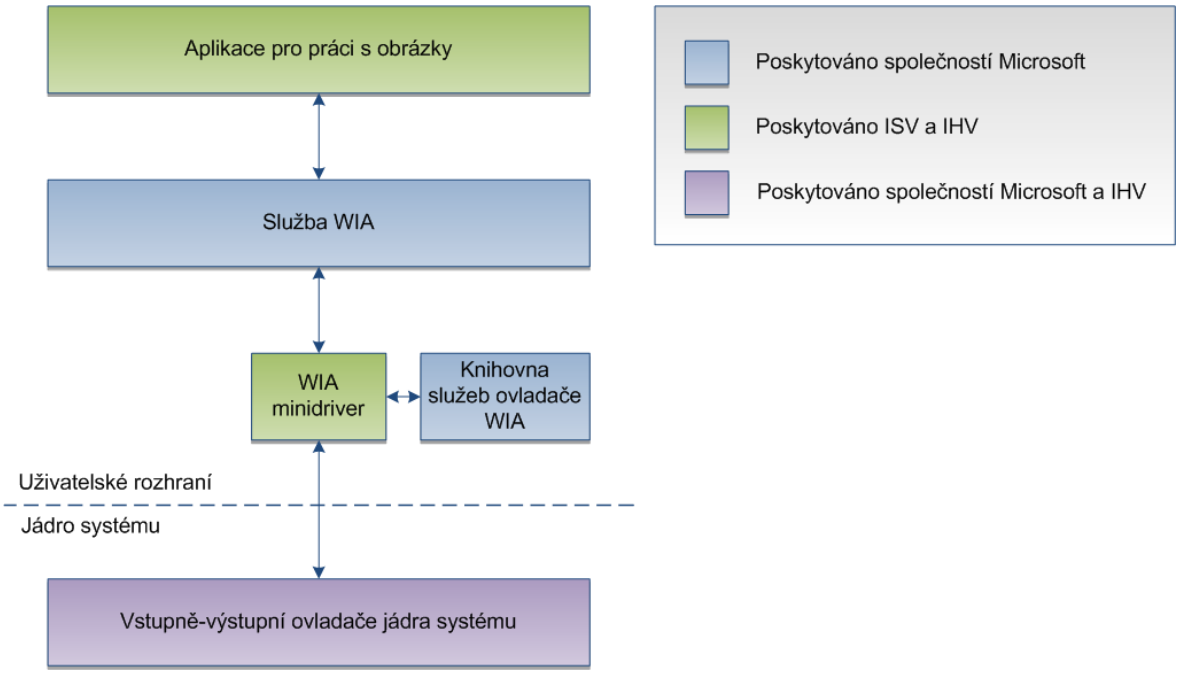
Aktuální verzi WIA je možno používat v operačním systému MS Windows XP. Pro MS Windows Vista již WIA není k dispozici.

WIA umožňuje aplikacím [22]:

- Chod ve stabilním a robustním prostředí.
- Minimalizaci problémů součinnosti.
- Získávání informací o aktuálně dostupných zařízeních na úpravu obrazu.
- Současnou komunikace s více zařízeními.
- Získávání informací o parametrech zařízení.
- Získávání dat ze zařízení standardizovaným a výkonným mechanismem.
- Upozornění na široké množství událostí na daném zařízení.

Dle [14] se technologie WIA skládá z několika součástí (obr. 35), které lze rozdělit také podle dodavatele dané součásti. Samotnou službu WIA a k ní náležící knihovnu ovladačů poskytuje společnost Microsoft. Tyto dvě komponenty samotné by však nestačily pro zajištění správné komunikace mezi uživatelskou aplikací a daným zařízením. Proto je nutná i podpora ze strany dodavatele hardwaru (IHV). Ten musí do ovladačů svého zařízení implementovat tzv. *WIA minidriver*, který implementuje do ovladačů zařízení

podporu technologie WIA. Samozřejmě je nutné napojení na jádro systému MS Windows a na danou aplikaci pro práci s obrázky, kterou vytváří poskytovatel softwaru (ISW).



Obr. 35. Struktura knihovny WIA

8.1.2 Použití WIA pro získání obrázku ze scanneru

Použití technologie WIA nejlépe ilustruje následující komplexní příklad:

```
1  WiaClass wiaManager = null;          // WIA manager COM object
2  CollectionClass wiaDevs = null;      // WIA devices collection COM
   object
3  ItemClass wiaRoot = null;            // WIA root device COM object
4  CollectionClass wiaPics = null;      // WIA collection COM object
5  ItemClass wiaItem = null;            // WIA image COM object
6
7  try
8  {
9         wiaManager = new WiaClass();        // create COM instance of
   WIA manager
10
11        wiaDevs = wiaManager.Devices as CollectionClass;
    // call Wia.Devices to get all devices
12
13        if ((wiaDevs == null) || (wiaDevs.Count == 0))
14        {
```

```
15              MessageBox.Show(this, "Nebyl nalezen žádný scanner, či
   jiné zobrazovací zařízení!", "WIA", MessageBoxButtons.OK,
   MessageBoxIcon.Stop);
16              return;
17        }
18
19        object selectUsingUI = System.Reflection.Missing.Value;
          // = Nothing
20        wiaRoot = (ItemClass)wiaManager.Create(ref selectUsingUI); //
   let user select device
21        if (wiaRoot == null)    // nothing to do
22        return;
23
24        // this call shows the common WIA dialog to let the user
   select a picture:
25        wiaPics = wiaRoot.GetItemsFromUI(WiaFlag.SingleImage,
   WiaIntent.ImageTypeColor) as CollectionClass;
26        if (wiaPics == null)
27        return;
28
29        bool takeFirst = true;  // this sample uses only one single
   picture
30        foreach (object wiaObj in wiaPics)  // enumerate all the
   pictures the user selected
31        {
32              if (takeFirst)
33              {
34                    DisposeImage();   // remove previous picture
35                    wiaItem =
   (ItemClass)Marshal.CreateWrapperOfType(wiaObj, typeof(ItemClass));
36                    imageFileName = Path.GetTempFileName();   //
   create temporary file for image
37                    Cursor.Current = Cursors.WaitCursor;     // could
   take some time
38                    this.Refresh();
39                    wiaItem.Transfer(imageFileName, false);   //
   transfer picture to our temporary file
40                    pbOriginal.Image = Image.FromFile(imageFileName);
    // create Image instance from file
41                    takeFirst = false;      // first and only one
   done.
42              }
43              Marshal.ReleaseComObject(wiaObj);
    // release enumerated COM object
44        }
```

```
45  }
46  catch (Exception ee)
47  {
48        MessageBox.Show(this, "Získání obrazových dat ze scanneru
    selhalo:\r\n" + ee.Message, "WIA", MessageBoxButtons.OK,
    MessageBoxIcon.Stop);
49        System.Windows.Forms.Application.Exit();
50  }
51  finally
52  {
53        if (wiaItem != null)
54              Marshal.ReleaseComObject(wiaItem);        // release WIA
    image COM object
55        if (wiaPics != null)
56              Marshal.ReleaseComObject(wiaPics);        // release WIA
    collection COM object
57        if (wiaRoot != null)
58              Marshal.ReleaseComObject(wiaRoot);  // release WIA root
    device COM object
59        if (wiaDevs != null)
60              Marshal.ReleaseComObject(wiaDevs);  // release WIA
    devices collection COM object
61        if (wiaManager != null)
62              Marshal.ReleaseComObject(wiaManager);     // release WIA
    manager COM object
63        Cursor.Current = Cursors.Default;   // restore cursor
64  }
```

8.2 Uchovávání vektorů souřadnic

Při práci s plošnými grafy je nutné uchovávat souřadnice jednotlivých bodů. Tyto souřadnice jsou tvořeny reálnými hodnotami X a Y.

Pro uložení souřadnice jednoho bodu se nabízí struktura *Point* ze jmenného prostoru *System.Windows* (knihovna *WindowsBase.dll*). Tato struktura přesně splňuje výše uvedené požadavky, tj. je tvořena dvojicí reálných hodnot X a Y. Tím je zatím vyřešena pouze záležitost jednoho bodu. Jak už bylo ale zmíněno výše, nutné je uchovávat vektory hodnot. Pro uchování vektorů je v .NET Framework implementována třída *List<T>*, která byla zařazena do jmenného prostoru *System.Collections.Generic*.

Při implementaci objektu *List<T>* je nutné v definici objektu dosadit za *T* jméno datového typu, struktury nebo třídy. Následně je možné s tímto objektem pracovat pomocí jeho jednotlivých metod. Ty nejpoužívanější shrnuje tab. 30.

Tab. 30. Nejpoužívanější metody třídy List<T>

Metoda	Popis
Add	Přidání nové položky
Clear	Odstranění všech položek
Exists	Zjištění zda daná položka je již v poli obsažena
Remove	Odstranění určité položky
RemoveAll	Odstranění všech položek splňujících určitou podmínku

Závěrem je vhodné uvést ukázku použití. V tomto případě byl tento objekt využit pro uchování jednotlivých bodů definovaných funkcí *Sin(x)*:

```
1 …
2 using System.Collections.Generic;
3 using System.Windows;
4 …
5 List<System.Windows.Point> body = new List<System.Windows.Point>();
6 …
7 double i = 0;
8 while(i<=(2*Math.PI))
9 {
10 body.Add(new System.Windows.Point(i, Math.Sin(i)));
11 i=i+0.05;
12 }
13 …
```

8.2.1 Řazení dat ve vektoru souřadnic

Pro zjednodušení vykreslování grafu a přehlednější zobrazení tabulky hodnot jsou data průběžně řazena. Toto řazení dat má svůj význam zejména při ručním přidávání dat ve vytvořené aplikaci pro digitalizaci grafu (více v kapitole 7.3.7).

Pro řazení dat byl vybrán algoritmus QuickSort. Encyklopedie Wikipedia o něm říká:„*Základní myšlenkou quicksortu je rozdělení řazené posloupnosti čísel na dvě přibližně stejné části. V jedné části jsou čísla větší a ve druhé menší, než nějaká zvolená hodnota (nazývaná pivot). Pokud je tato hodnota zvolena dobře, jsou obě části přibližně stejně velké. Pokud budou obě části samostatně seřazeny, je seřazené i celé pole. Obě části se pak rekurzivně řadí stejným postupem.*“. [21]

Ve srovnání s nejpoužívanějším algoritmem BubbleSort QuickSort nabízí menší časovou složitost, v optimálním případě $O(N \log N)$. Nejhorší časová složitost je stejná jako u

BubbleSortu - $O(N^2)$ (v případě BubbleSortu této hodnoty dosahuje i průměrná časová složitost). Více se těmto dvěma algoritmům věnuje [11], [21] a [20].

Při řazení vektoru souřadnic se QuickSort použije následovně:

```
1  private void QuickSort(int left, int right)
2  {
3        System.Windows.Point h = new System.Windows.Point();
4        double x = new double();
5        int i = new int();
6        int j = new int();
7        i = left;
8        j = right;
9        x = body[((left + right) - ((left + right) % 2)) / 2].X;
10
11       do
12       {
13           while (body[i].X < x) i++;
14           while (body[j].X > x) j--;
15
16           if (i <= j)
17           {
18               h = body[i]; body[i] = body[j]; body[j] = h;
19               i++; j--;
20           }
21
22       }
23       while (i <= j);
24
25       //  recursion
26       if (left < j) QuickSort(left, j);
27       if (i < right) QuickSort(i, right);
28 }
```

Pozn. Při volání této metody se za parametr *left* dosadí hodnota 0 a za *right* velikost pole.

8.3 Manuální digitalizace

Při manuální digitalizaci je načten obrázek do instance třídy *Bitmap*. Následně uživatel musí nastavit okrajové hodnoty na osách X a Y (to je provedeno kliknutím na obrázek a zadáním hodnoty pro daný bod).

Po zadání souřadnic je již možné provést samotnou digitalizaci. Ta se provádí výběrem bodů křivky grafu. Tyto body je nutné přepočítat na skutečné souřadnice grafu dle následujícího programu:

```
1 // výpočet souřadnice X:
2 rozsah_x = souradnice_x_max – souradnice_x_min;
3 pixelu_na_jednotku_x = rozsah_x / (hodnota_x_max - hodnota_x_min);
4 r_souradnice_x = hodnota_x_min + (i_souradnice_x - souradnice_x_min)
  / pixelu_na_jednotku_x;
5 // výpočet souřadnice Y:
6 rozsah_y = souradnice_x_min – souradnice_x_max;
7 pixelu_na_jednotku_y = rozsah_y / (hodnota_y_max - hodnota_y_min);
8 r_souradnice_y = hodnota_y_min + (souradnice_y_min - i_souradnice_y)
  / pixelu_na_jednotku_y;
```

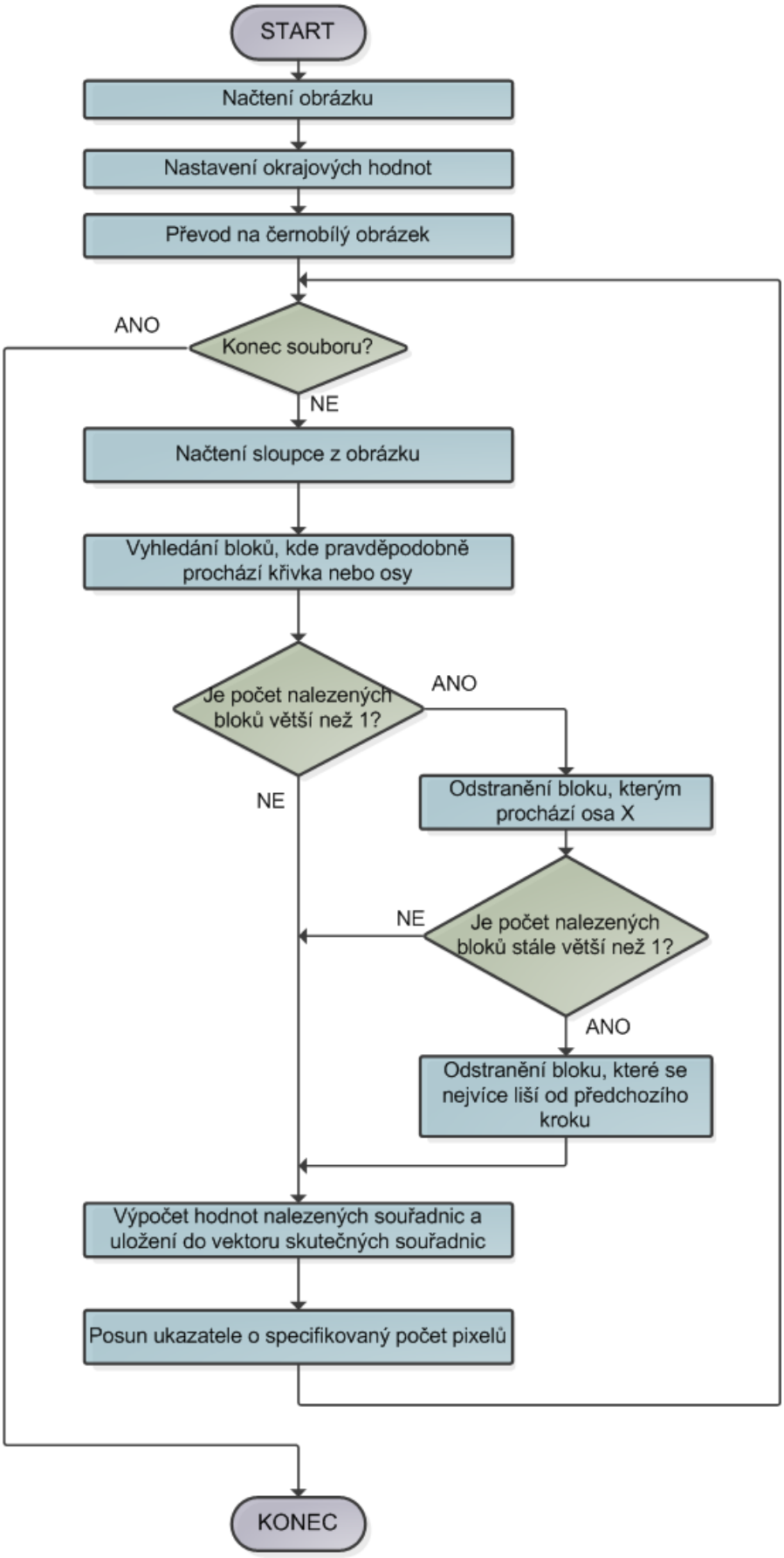
Pozn.: i_souradnice_x a i_souradnice_y jsou souřadnice obrázku, r_souradnice_x a r_souradnice_y jsou skutečné souřadnice grafu.

Hodnoty získané dle výše uvedeného kódu jsou následně uloženy do vektoru souřadnic (jeho struktura je popsána v kapitole 8.2)

8.4 Automatická digitalizace

Algoritmus, který byl použit pro digitalizaci grafu, je poněkud rozsáhlejší, proto se tato kapitola soustřeďuje pouze na jeho stručný popis. Kompletní zdrojový kód lze najít na přiloženém disku CD-ROM.

Funkce použitého algoritmu je nejvíce patrná z níže uvedeného vývojového diagramu. Jednotlivé bloky diagramu jsou popsány na následujících stránkách.



Obr. 36. Vývojový diagram automatické digitalizace grafu

8.4.1 Načtení obrázku a nastavení výchozích hodnot

Načtení obrázku a nastavení výchozích hodnot probíhá podobně jako v případě manuální digitalizace grafu (kapitola 8.3). Je zde však jeden rozdíl. Pokud je zobrazena osa X nebo osa Y, je nutné místo okrajových bodů grafu nadefinovat okrajové body těchto os.

8.4.2 Převod na černobílý obrázek

Převod na černobílý obrázek probíhá ve dvou krocích. Nejprve je obrázek upraven na odstíny šedi dle následujícího vzorce [2]:

$$I = 0{,}299R + 0{,}587G + 0{,}114B$$

Druhým krokem je převod na černobílý obrázek s jednobitovou barevnou hloubkou. Toho je dosaženo následujícími podmínkami:

- Pokud je I větší než 128, je na pixel použita černá barva.
- Pokud je I menší než 128, je na pixel použita bílá barva.

8.4.3 Digitalizace

Nyní již k samotné digitalizaci. Oblast obrázku, která byla označena k digitalizaci, je procházena po sloupcích o šířce 3 pixely. V každém sloupci jsou vyhledány bloky, kudy by mohl procházet graf (bloky s černou barvou). Pokud je těchto bloků více, musí být vyloučeny všechny, které s největší pravděpodobností neodpovídají skutečnému průběhu grafu. To se děje dle následujícího postupu:

- Odstranění bloku, kterým prochází osa X.
- Odstranění všech bloků, které mají větší vzdálenost od bodu, který byl nalezen v předchozím kroku.
- Pokud i tak zůstane více bloků, je použit první nalezený blok.

Po projití všech sloupců jsou hodnoty přepočítány na skutečné souřadnice. To se děje stejně jako v případě manuální digitalizace grafu (kapitola 8.3). Takto získané hodnoty jsou následně uloženy do vektoru souřadnic (jeho struktura je popsána v kapitole 8.2).

8.4.4 Poznámky k použitému algoritmu

Algoritmus použitý pro automatickou digitalizaci má určitá omezení. Pro správnou funkci této metody totiž musí být splněny dva předpoklady:

- Graf neobsahuje více průběhů.

- Mřížka grafu je méně výrazná než samotný průběh grafu.

Ve výše uvedených případech nedá automatická digitalizace správný výsledek, a musí být použita digitalizace manuální (kapitola 8.3).

8.5 Zápis do souboru

8.5.1 Textové soubory

V případě textových souborů se o ukládání dat postará instance třídy *StreamWriter* ze jmenného prostoru *System.IO*.

Způsob práce s tímto objektem lze nejlépe ilustrovat následujícím příkladem:

```
1 using System.IO;
2 …
3 using (StreamWriter sw = new StreamWriter(dlgSaveVal.FileName))
4 {
5         sw.WriteLine("X\tY");
6         foreach (System.Windows.Point B in body)
7         {
8               sw.WriteLine(B.X.ToString() + "; " + B.Y.ToString());
9         }
10 }
```

Výše uvedená část programu slouží pro zápis hodnot do souboru ve formátu CSV (hodnoty oddělené středníkem).

Nejprve je nutné definovat jmenný prostor, jehož součástí je třída *StreamWriter* (1. řádek výše uvedeného zdrojového kódu). Následně je už možné vytvořit instanci této třídy (3. řádek) a dále ji používat.

Samotný zápis do souboru zajišťuje dvojice metod *Write* a *Writeline*. V tomto případě byla použita metoda *WriteLine* (řádky č. 5 a 8), která zapíše do souboru celý řádek textu (na konec řádku je vloženo zalomení řádku a další volání metody pro zápis vkládá text na nový řádek).

8.5.2 Obrázky

U ukládání obrázků je situace nejjednodušší. Ukládání totiž zajišťuje jediná metoda třídy Image. Její obecná syntaxe je následující:

```
1 public void Save(string filename, ImageFormat format)
```

Na místo parametru *filename* se doplní cesta k souboru, do kterého má být obrázek uložen, a na místo parametru *format* formát ukládaného souboru. Seznam podporovaných souborů shrnuje tab. 31.

Tab. 31. Formáty obrázků podporované objektem Image

Název	Popis
Bmp	Obrázek ve formátu bitmap (BMP).
Emf	Obrázek ve formátu enhanced metafile (EMF).
Exif	Obrázek ve formátu Exchangeable Image File (Exif).
Gif	Obrázek ve formátu Graphics Interchange Format (GIF).
Icon	Obrázek ve formátu Windows icon.
Jpeg	Obrázek ve formátu Joint Photographic Experts Group (JPEG).
Png	Obrázek ve formátu W3C Portable Network Graphics (PNG).
Tiff	Obrázek ve formátu Tagged Image File Format (TIFF).
Wmf	Obrázek ve formátu Windows metafile (WMF).

Konkrétní použití této metody tedy může vypadat například takto:

```
1 if (dlgSavePic.ShowDialog() == DialogResult.OK)
2 {
3         obrazek.Save(dlgSavePic.FileName, ImageFormat.Bmp);
4 }
```

ZÁVĚR

Cílem této diplomové práce bylo vytvoření programu pro automatickou digitalizaci plošného grafu z obrázku. Pro vývoj programu byla zvolena platforma .NET a programovací jazyk C#. Důvodem pro tuto volbu byl fakt, že se jedná o moderní a v dnešní době velmi populární platformu, která je neustále zlepšována. Na druhou stranu je zde menší nevýhoda, a tou je skutečnost, že takto vytvořenou aplikaci lze využívat pouze na operačním systému MS Windows (existuje sice projekt MONO, který umožňuje překlad MS .NET Framework aplikací pro GNU/Linux, neobsahuje však všechny funkce z nejnovějšího MS .NET Frameworku).

Vzhledem k tomu, že pro vývoj aplikace bylo zvoleno vývojové prostředí MS .NET Framework, věnovala se mu i první část této práce. Tato část byla rozdělena do tří kapitol:

- Technologie MS .NET Framework
- Programovací jazyk C#
- Vývojová prostředí

Druhá část práce se již věnovala samotnému problému digitalizace grafu. Nejprve byly rozebrány samotné principy digitalizace. Následovala část, kde byly vyzkoušeny již existující programy, kde bylo zjištěno, že již existuje velké množství řešení, avšak žádné nenabízí plně automatickou digitalizaci grafu.

Závěrem následoval samotný vývoj programu. Tomu jsou v této práci věnovány dvě kapitoly. První z nich popisuje program spíše z uživatelského pohledu (ovládání programu, použití vytvořených knihoven, …). Druhá rozebírá algoritmy, které byly použity při vývoji programu.

Při vývoji programu byly nejprve vyvinuty dvě knihovny - knihovna *PlotPaint2D* pro vykreslování grafů a knihovna *PlotDigitizer2D* pro digitalizaci grafů. Následně již mohl být vytvořen samotný program pro digitalizaci grafu, který využívá obě výše uvedené knihovny.

Důvodem pro vznik knihovny *PlotPaint2D* byla skutečnost, že MS .NET Framework neobsahuje žádné knihovny pro vykreslování grafu, proto musely být vytvořeny funkce, které využívají základní způsoby kreslení pomocí knihoven GDI+. Tato knihovna obsahuje metody pro vykreslování grafů na základě vektoru dvourozměrných souřadnic.

Druhá knihovna, *PlotDigitizer2D*, tvoří základní část této práce. Slouží totiž k samotné digitalizaci grafu. Ta může probíhat ve dvou režimech: v manuálním a v automatickém.

V obou případech je nejprve nutné vybrat krajní body grafu a nastavit jejich hodnoty. Další postup se již liší dle zvolené metody.

Automatický režim provede vyhledávání bodů křivky zcela samostatně. Pro správnou funkci této metody musí být splněny dva předpoklady:

- Graf neobsahuje více průběhů.
- Mřížka grafu je méně výrazná než samotný průběh grafu.

V případě, že výše uvedené předpoklady nebudou splněny, musí být provedena manuální digitalizace. Úvodní postup je stejný jako v předchozím případě, s tím rozdílem, že body k digitalizaci musí vybrat sám uživatel.

Jak je patrné z předchozích řádků, digitalizaci zatím nelze provést zcela bez zásahu uživatele. Vždy je nutné nastavit minimálně krajní body. Program lze však dále rozvíjet a lze uvažovat, že při implementaci funkcí, které fungují na bázi softwaru OCR, již bude možné zautomatizovat digitalizaci více, než je tomu teď. A protože algoritmy pro oba hlavní řešené problémy byly vyvíjeny jako samostatné knihovny, je možné je jednoduše použít i při vývoji dalších aplikací v této oblasti.

ZÁVĚR V ANGLIČTINĚ (CONCLUSION)

The aim of this master thesis was to make program for 2D curve automatic digitizing from figure. For development of this application was selected .NET platform. The reason for this choice was the fact, that it is modern and nowadays very popular platform, which is improved continual. But there is one disadvantage. This way developed application could be run only on MS Windows operation system (there exists MONO project though, which enable running this application at GNU/Linux, but there are not included all functions from the newest version of MS .NET Framework).

Regarding that it was selected MS .NET Framework for development this application, the first part documented this technology. This part was sectored to three chapters:

- MS .NET Framework technology
- C# programming language
- Development environments

The second part dealt with problem of plot digitizing. At first, there were analyzed principles of digitizing. In the following part, there were trying out existing applications. In this part it was found that there are many applications for this purpose, but none of them offered full automatic plot digitizing.

Finally came on own application development. In this thesis there are two chapters about it. First of them describes application from user view (control of the application, using of libraries, which were made within the frame of this thesis, …). The second part describes algorithms, which were used by the application development.

By the development of main application, two libraries were developed at first - *PlotPaint2D* library for painting plots and *PlotDigitizer2D* library for plot digitizing. Consequently the main application could be developed. This application is utilizing above-mentioned libraries.

The reason for origination of *PlotPaint2D* library was the fact that in the .NET Framework is not included any library for painting of 2D plots. Because of it the functions which utilize basic ways of painting by way of GDI+ libraries must be made. This library contains methods for painting 2D plots on the basis of vectors of two dimensional coordinates.

Second library, *PlotDigitizer2D*, forms the basic part of this thesis. Namely it serves the purpose of plot digitizing. It could be used in two modes: manual and automatic.

In both cases, there is necessary to select end points of plot and set its values. Next procedure is various according to selected method.

Automatic mode finds out points of the curve totally independently. For the right function of this method must be fulfilled two premises:

- The plot has not got more than one curve
- The matrix grid is fewer strong than the curve.

In the case of nonfulfilment of above-mentioned premises, manual digitizing must be taken over. Beginning part is the same that in previous case, but with the difference that user selects points for digitizing manually.

As it is evident from previous lines, digitizing could not be realized without user's intervention. It is always minimally required to set up end points of the plot. But application could be further developed and it is possible to think about implementation of OCR-based functions. After the implementation of these functions the digitizing could be fully automatic. And because the fact that these algorithms were developed as independent libraries, it is easy to use them to developing other applications in this area.

SEZNAM POUŽITÉ LITERATURY

Monografie:

[1] BAYER Jürgen. C# 2005 : velká kniha řešení. 1. vydání. Brno: Computer Press, a.s. 2007. 816 s. ISBN 978-80-251-1620-3

[2] BLIŽŇÁK Michal. Systémové programování. 1. vydání. Zlín: Univerzita Tomáše Bati ve Zlíně. 2005. 206 s. ISBN 80-7318-364-1

[3] ELLER Frank. C# - začínáme programovat. Praha: Grada Publishing, a.s. 2002. 1. vydání. 240 s. ISBN 80-247-0324-6

[4] GIANNINI Mario, KEOGH James. Objektově orientované programování bez předchozích znalostí. 1. vydání. Brno: Computer Press, a.s. 2006. 224 s. ISBN 80-251-0973-9

[5] HANÁK Ján. C# - praktické příklady. Praha: Grada Publishing, a.s. 2006. 1. vydání. 288 s. ISBN 80-247-0988-0

[6] KAČMÁR Dalibor. Programujeme .NET aplikace ve Visual Studiu .NET. 1. vydání. Brno: Computer Press, a.s. 2001. 344 s. ISBN 80-722-6569-5

[7] NAGEL Christian, EVJEN Bill, GLYNN Jay, SKINNER Morgan, WATSON Karli, JONES Allen. C# 2005 - programujeme profesionálně. 1. vydání. Brno: Computer Press, a.s. 2006. 1400 s. ISBN 80-251-1181-4

[8] POKORNÝ Pavel. Základy počítačové grafiky. 1. vydání. Zlín: Univerzita Tomáše Bati ve Zlíně. 2004. 120 s. ISBN 80-7318-161-4

[9] SELLS Chris. C# a WinForms. 1. vydání. Brno: Zoner Press. 2005. 645 s. ISBN 80-86815-25-0

[10] ŽÁRA Jiří, BENEŠ Bedřich, FELKEL Petr. Moderní počítačová grafika. 1. vydání. Brno: Computer Press, a.s. 1998. 448 s. ISBN 80-7226-049-9

Internetové zdroje:

[11] *Bubble sort* [online]. Wikipedia, the free encyclopedia. [cit. 2008-03-25]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Bubble_sort>

[12] *Dokumentace MSDN* [online]. Microsoft. [cit. 2008-03-16]. Dostupný z WWW: <http://msdn2.microsoft.com/cs-cz/library/default(en-us).aspx>

[13] *GetData Graph Digitizer*. [online]. GetData Graph Digitizer. [cit. 2008-04-07]. Dostupný z WWW: < http://getdata-graph-digitizer.com/ >

[14] *Imaging* [online]. Microsoft. [cit. 2008-03-23]. Dostupný z WWW: <http://www.microsoft.com/whdc/device/StillImage/default.mspx>

[15] Logic Graph Digitizing Software [online]. The Logic Group. [cit. 2008-04-07]. Dostupný z WWW: <http://www.logicgroup.com/Software/GraphDigi.htm>

[16] *Microsoft .NET Framework 3.0 Release Notes* [online]. Microsoft. [cit. 2008-04-19]. Dostupný z WWW: < http://msdn2.microsoft.com/en-us/windowsvista/bb188202.aspx>

[17] *Microsoft Visual Studio 2008 – porovnání verzí* [online]. Microsoft. [cit. 2008-02-19]. Dostupný z WWW: <http://www.microsoft.com/cze/msdn/produkty/vstudio/porovnani.mspx>

[18] *Plot Digitizer* [online]. Sourceforge.net. [cit. 2008-04-06]. Dostupný z WWW: <http://plotdigitizer.sourceforge.net/index.html>

[19] *Poznáváme C# a Microsoft .NET* [online]. Živě.cz. [cit. 2008-02-04]. Dostupný z WWW: < http://www.zive.cz/Programovani/C_CSHARP/sc-74/default.aspx>

[20] *Quick sort* [online]. Wikipedia, the free encyclopedia. [cit. 2008-04-06]. Dostupný z WWW: < http://en.wikipedia.org/wiki/Quicksort >

[21] *Quick sort* [online]. Wikipedie, otevřená encyklopedie. [cit. 2008-04-06]. Dostupný z WWW: < http://cs.wikipedia.org/wiki/Quicksort >

[22] *Windows Image Acquisition (WIA)* [online]. Microsoft. [cit. 2008-03-23]. Dostupný z WWW: < http://msdn2.microsoft.com/en-us/library/ms630368(VS.85).aspx >

[23] xyExtract Graph Digitizer. [online]. LAB Fit. [cit. 2008-04-07]. Dostupný z WWW: < http://www.angelfire.com/rnb/labfit/index.htm >

[24] *Základy počítačové grafiky v C# a GDI+ (2.)* [online]. PC Svět. [cit. 2008-03-16]. Dostupný z WWW: < http://www.pcsvet.cz/art/article.php?id=5609 >

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

API	Application Programming Interface (Rozhraní pro programování aplikací)
ASP	Active Server Pages
ASP.NET	Implementace ASP v MS .NET Framework
BMP	Windows Bitmap – formát souboru pro uložení rastrové grafiky
CLR	Common Language Runtime (Běhové prostředí .NET)
CLS	Common Language Specification (Společná jazyková specifikace)
CSV	Comma-separated values (hodnoty oddělené středníkem) – formát souboru
CTS	Common Type System (Společný typový systém)
GDI	Graphics Device Interface
GDI+	Graphics Device Interface Plus
GHz	Gigahertz – jednotka frekvence (1 GHz = 1 000 000 000 Hz)
GUI	Graphics User Interface (Grafické uživatelské rozhraní)
IHV	Independent Hardware Vendor
IPv6	Internet Protocol version 6
ISV	Independent Software Vendor
LINQ	Language Integrated Query
MB	jednotka informace (1MB = 1024^2 B)
MFC	Microsoft Foundation Classes
MHz	Megahertz – jednotka frekvence (1 MHz = 1 000 000 Hz)
MS	Microsoft
MSDN	Microsoft Developer Network
MSIL	Microsoft Intermediate Language
OCR	Optical character recognition (optické rozpoznávání znaků)
px	pixel (nejmenší jednotka rastrové grafiky)
SP	Service Pack (sada aktualizací, oprav a vylepšení programu)

USD	Americký dolar – měnová jednotka
WCF	Windows Communication Foundation
WIA	Windows Image Acquisition
WPF	Windows Presentation Foundation
XML	Extensible Markup Language – obecný značkovací jazyk

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA P1 – VYBRANÉ GRAFY, NA KTERÝCH BYL PROGRAM TESTOVÁN

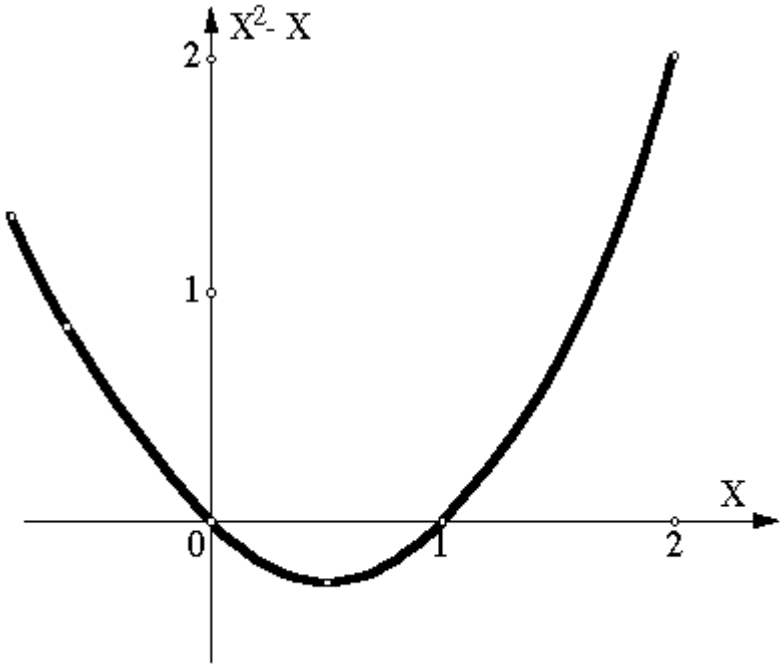
Graf 1:

Graf, který byl nalezen na Internetu.

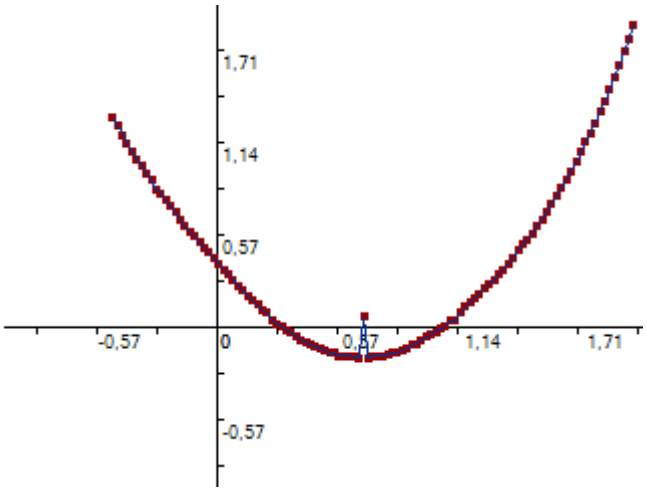
Komentář:

Až na jednu chybnou hodnotu bylo dosaženo přesného výsledku.

Originální graf



Výsledek digitalizace



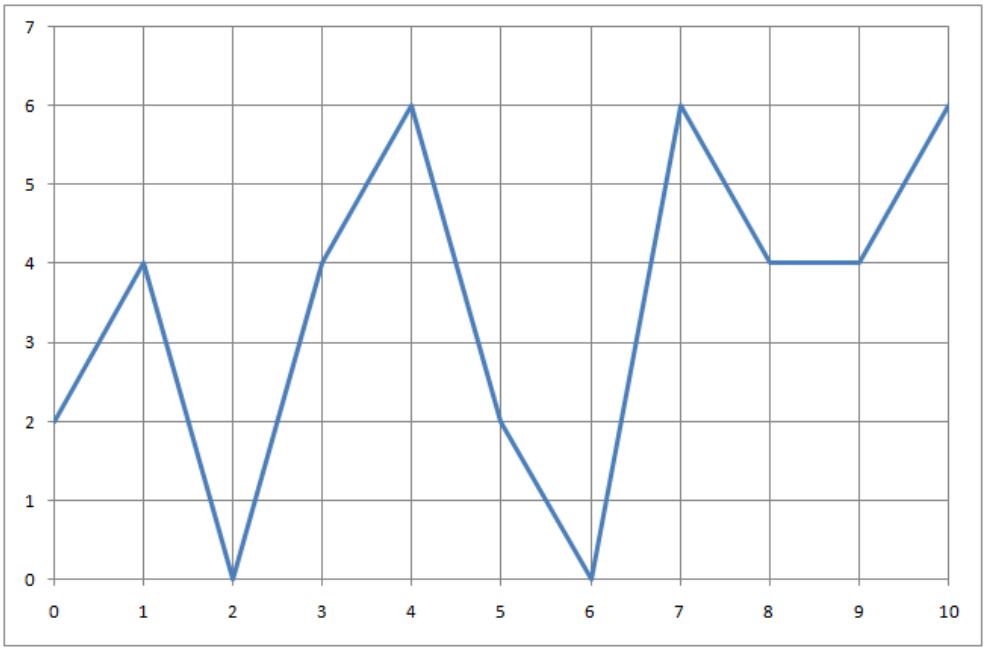
Graf 2:

Graf, který byl vygenerován aplikací MS Excel. Osy jsou na okraji grafu

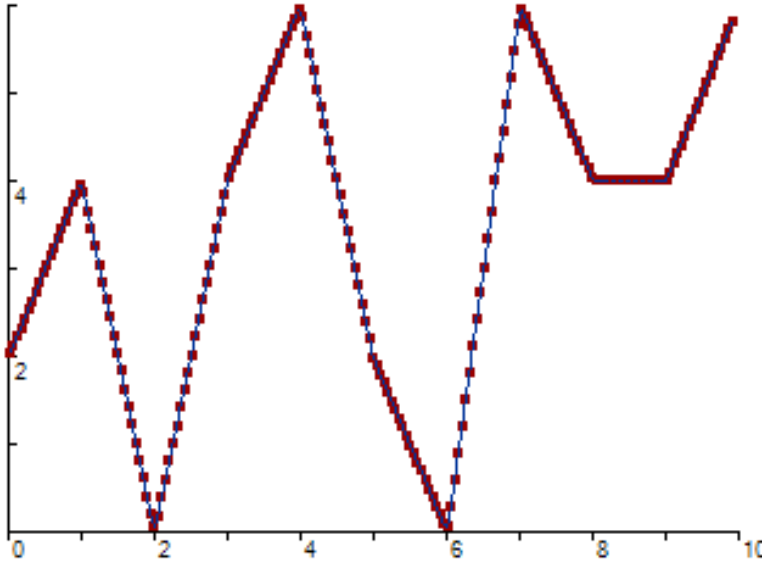
Komentář:

Bylo dosaženo přesného výsledku.

Originální graf



Výsledek digitalizace



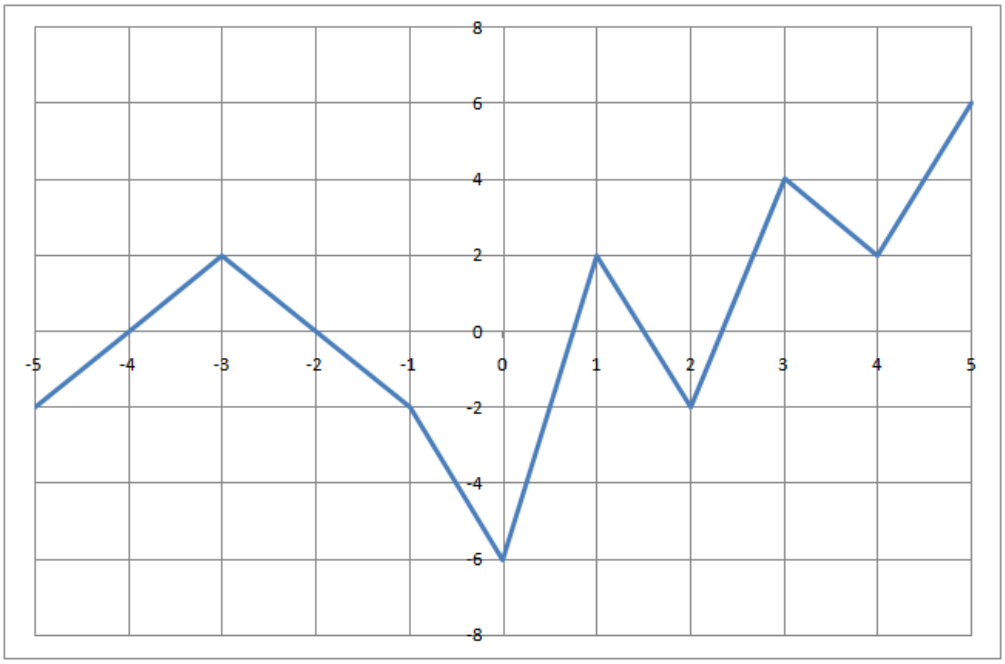
Graf 3:

Graf, který byl vygenerován aplikací MS Excel. Osy se protínají uprostřed grafu.

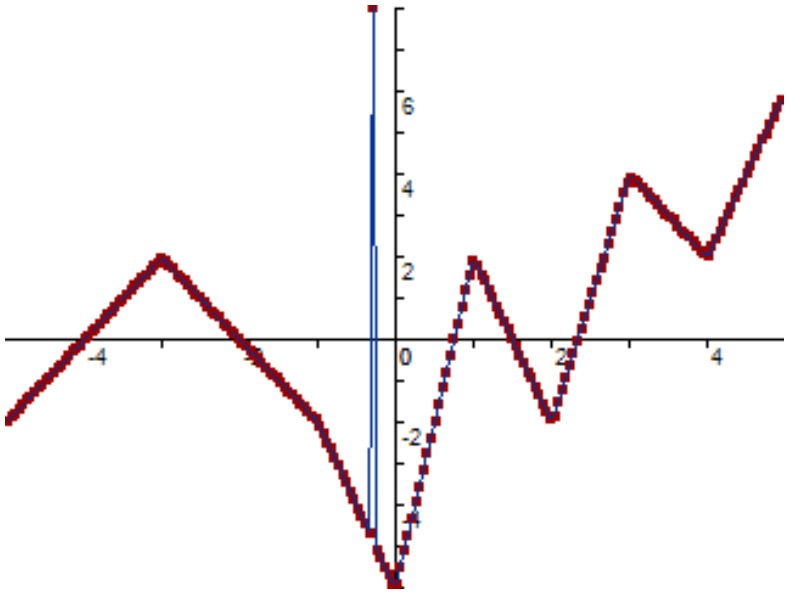
Komentář:

Až na jednu chybnou hodnotu bylo dosaženo přesného výsledku.

Originální graf



Výsledek digitalizace



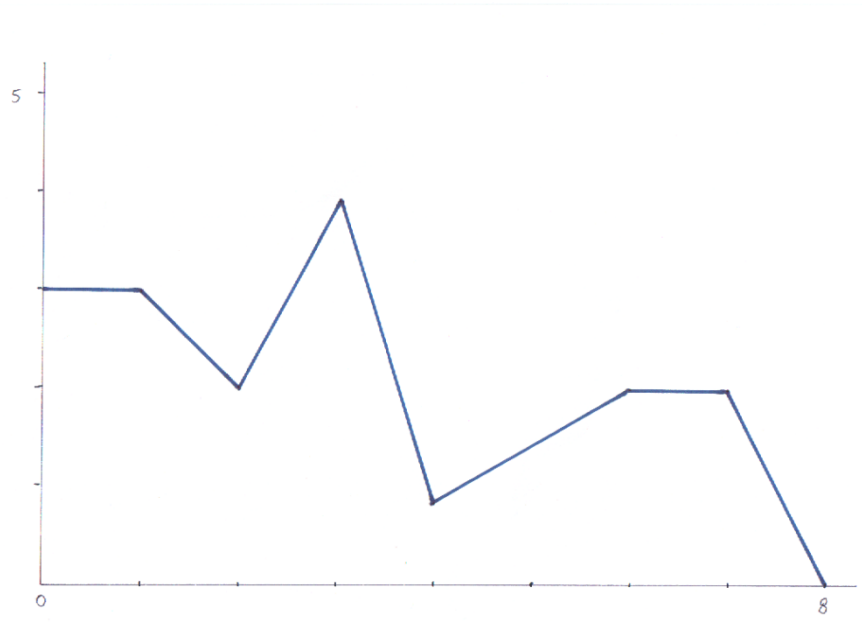
Graf 4:

Ručně kreslený graf na papíře.

Komentář:

Až na jednu chybnou hodnotu bylo dosaženo přesného výsledku.

Originální graf



Výsledek digitalizace