

ZÁKLADNÍ 2D VEKTOROVÁ GRAFIKA

Vektorová grafika

Programování
vektorové grafiky

Základní
primitiva

POČÍTAČOVÁ GRAFIKA

■ Tiskoviny, Reklama



POČÍTAČOVÁ GRAFIKA

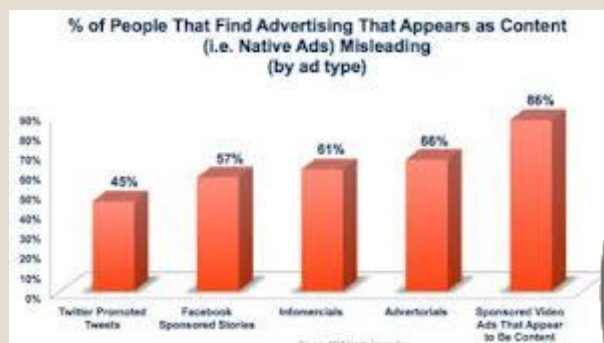
■ Média, televize, film, hry, ...

The screenshot shows a web browser window with the URL <http://voyo.nova.cz/product/serialy/31498-camelot-10-dil>. The page features a large image of the main characters from the TV series Camelot, with a play button overlay and the text "Přehrát video". To the right, there is a section titled "CAMELOT - 10. Díl" with a description: "Arthur se snaží ubránit zlodákům v průmysku Bardon. Morgan mezitím vede Syblu a své následovníky ke Camelotu. Po cestě dostane uspokojivé informace, které ji usvedčí v jejím přesvědčení, že trůn má na dosah ruky." Below this, there is a list of cast members and their roles, including Jamie Campbell Bower, Tamsin Egerton, Joseph Fiennes, Claire Forlani, Eva Green, Peter Mooney, Diarmuid Murtagh, Clive Standen, Philip Winchester, and Sinéad Cusack. The page also includes a star rating of 5 stars, a release date of 2011, and a duration of 50 minutes. At the bottom, there is a section for "SOUVISEJÍCÍ VIDEA" and a chat window.



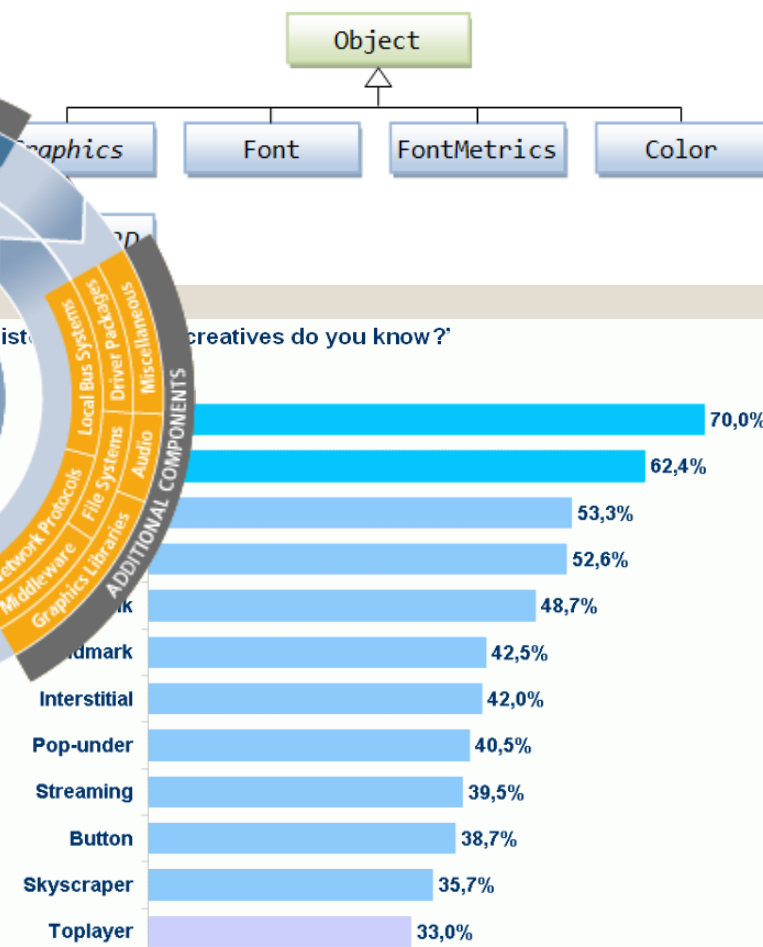
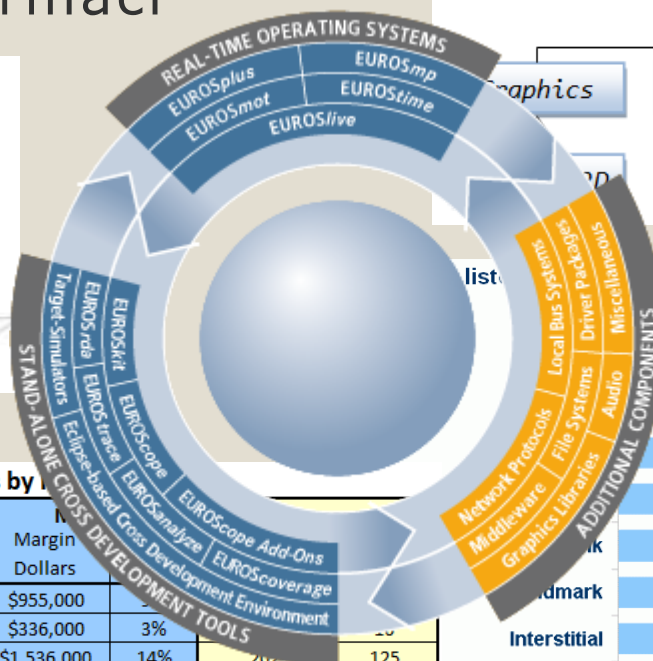
POČÍTAČOVÁ GRAFIKA

■ Zpracování informací



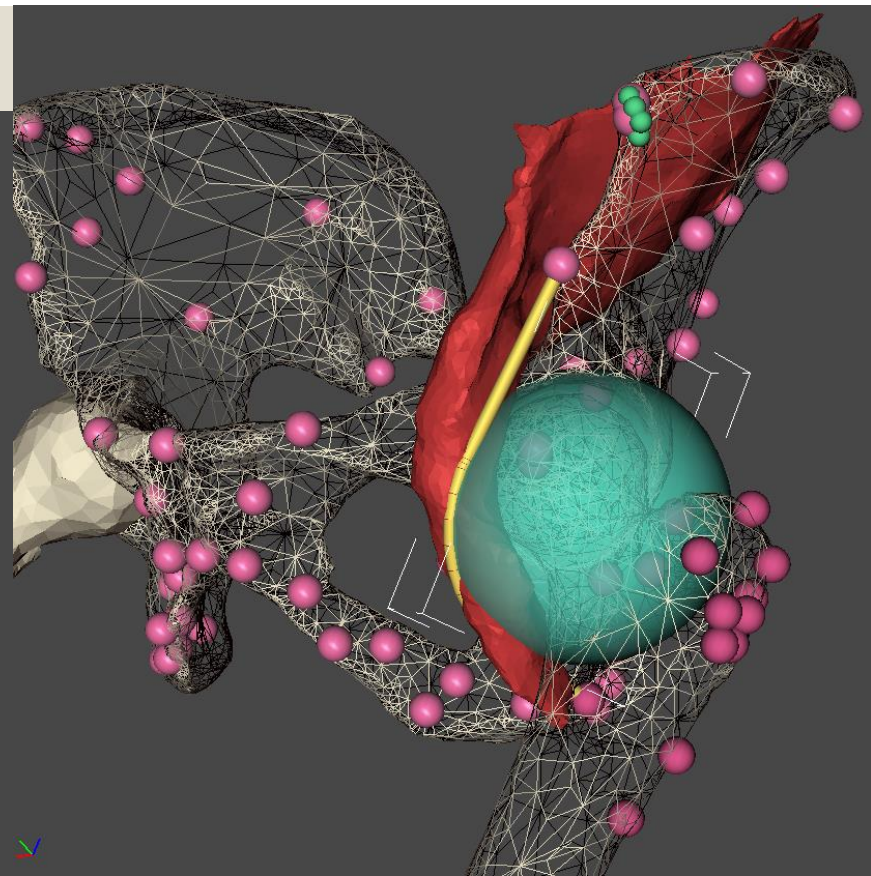
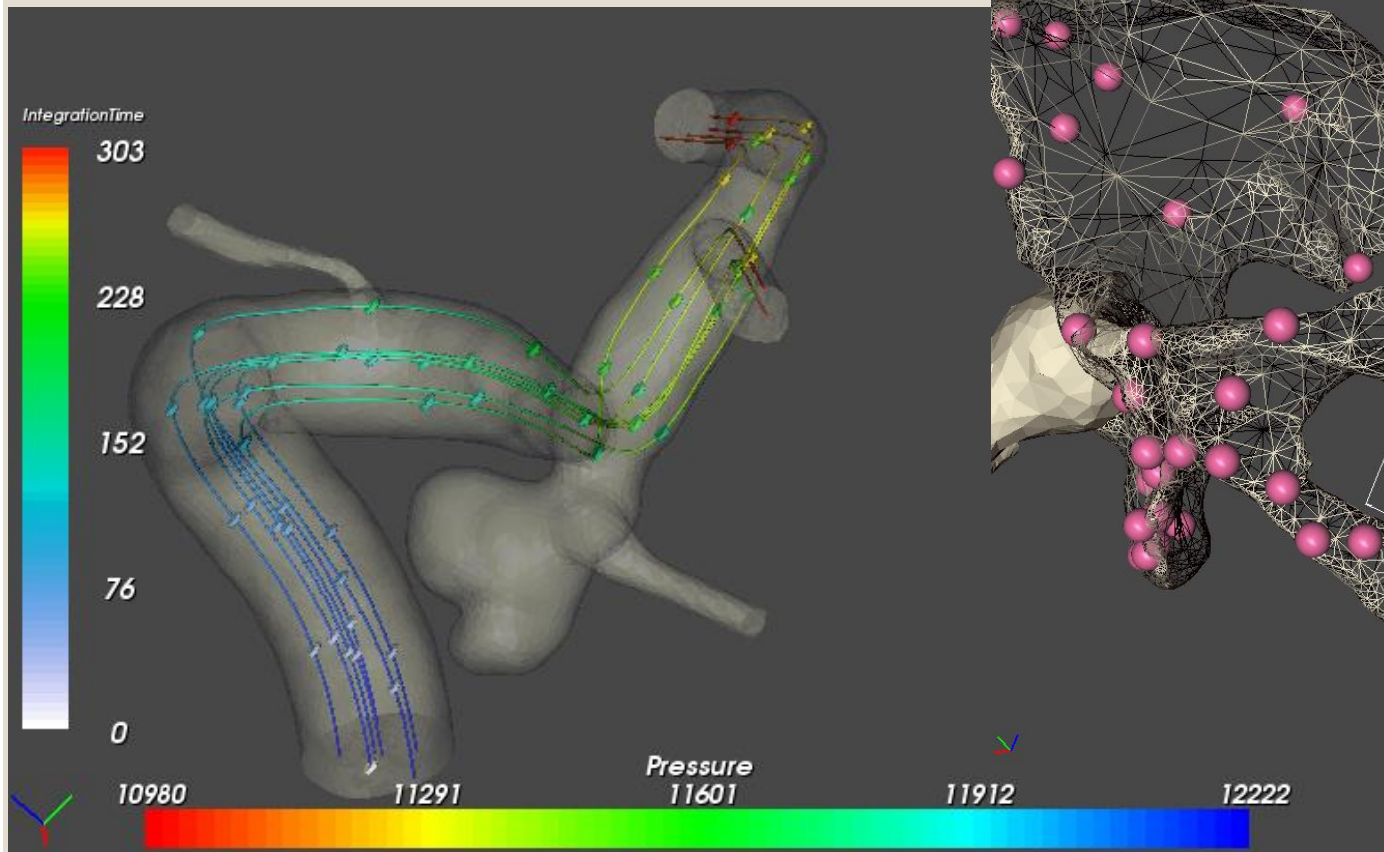
Top 10 Domestic Routes by Revenue

From	To	Revenue Dollars	Revenue Percent	Margin Dollars	Margin Percent	Passengers	Flights
Atlanta	New York	\$3,602,000	8.09%	\$955,000	26.5%	125	125
Chicago	New York	\$4,674,000	10.50%	\$336,000	7.2%	125	125
Columbus (Ohio)	New York	\$2,483,000	5.58%	\$1,536,000	61.9%	125	125
New York	Detroit	\$12,180,000	27.35%	\$2,408,000	19.8%	177	35
New York	Washington	\$6,355,000	14.27%	\$1,230,000	19.3%	186	36
New York	Philadelphia	\$3,582,000	8.04%	-\$716,000	-20.0%	125	-25
New York	San Francisco	\$3,221,000	7.23%	\$1,856,000	57.6%	590	340
New York	Phoenix	\$2,846,000	6.39%	\$1,436,000	50.5%	555	280
New York	Toronto	\$2,799,000	6.29%	\$1,088,000	39.0%	450	175
New York	Seattle	\$2,792,000	6.27%	\$467,000	16.7%	448	75
Total Domestic routes		\$44,534,000		\$10,596,000		272	53



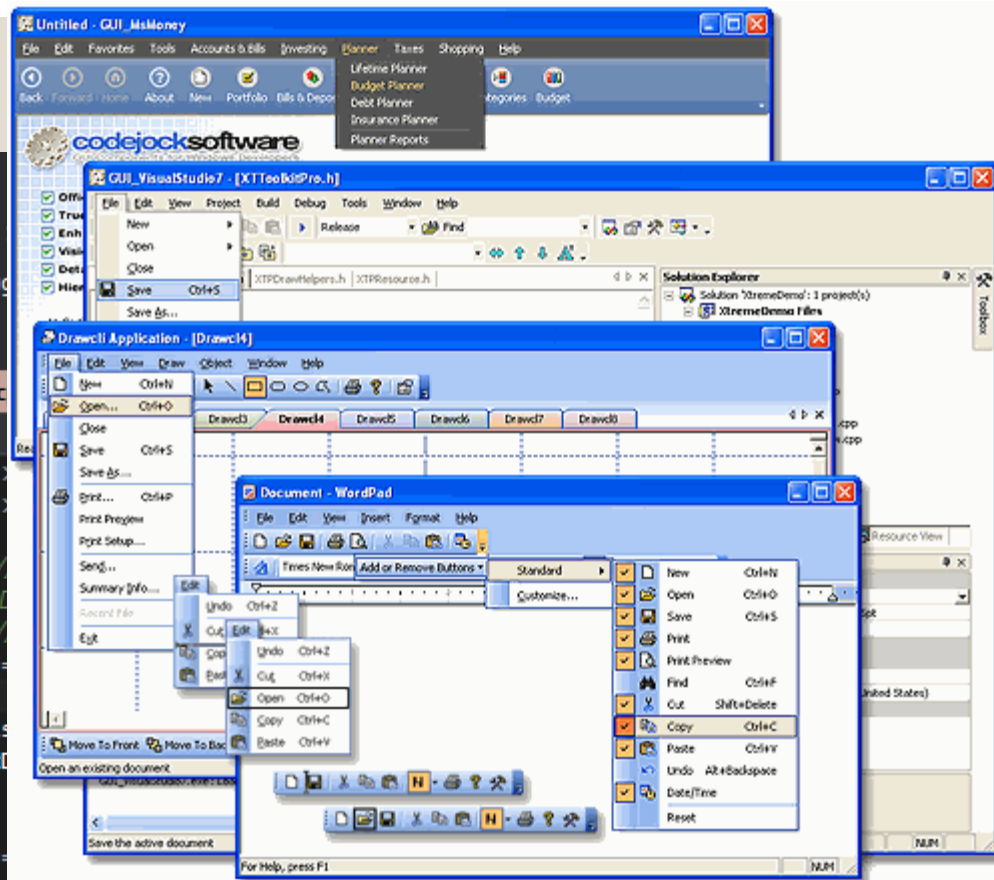
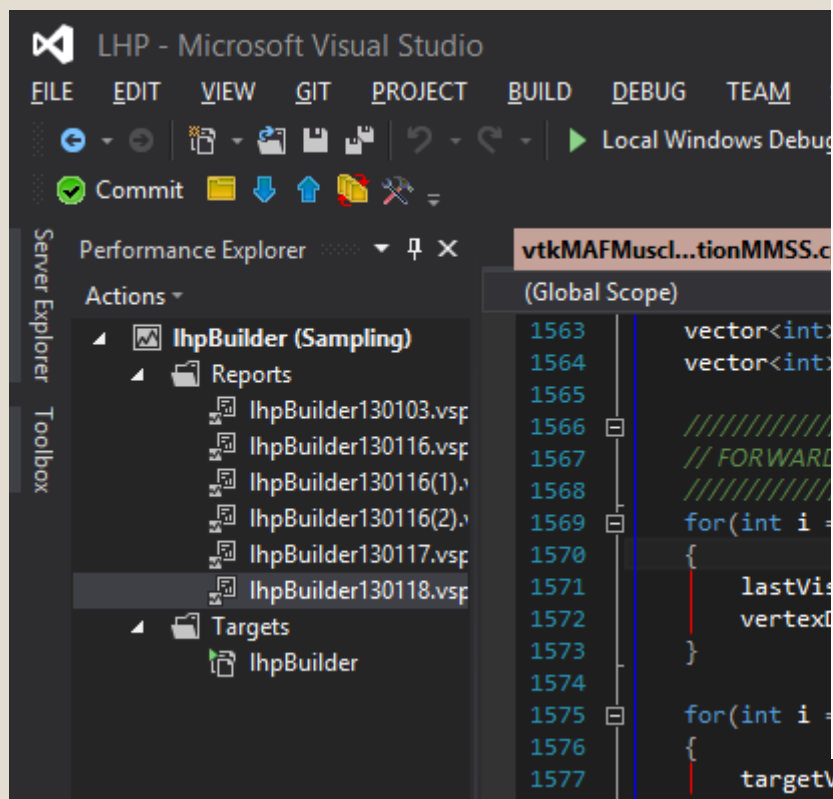
POČÍTAČOVÁ GRAFIKA

- Věda, virtuální realita, ...



POČÍTAČOVÁ GRAFIKA

■ Aplikace 😊



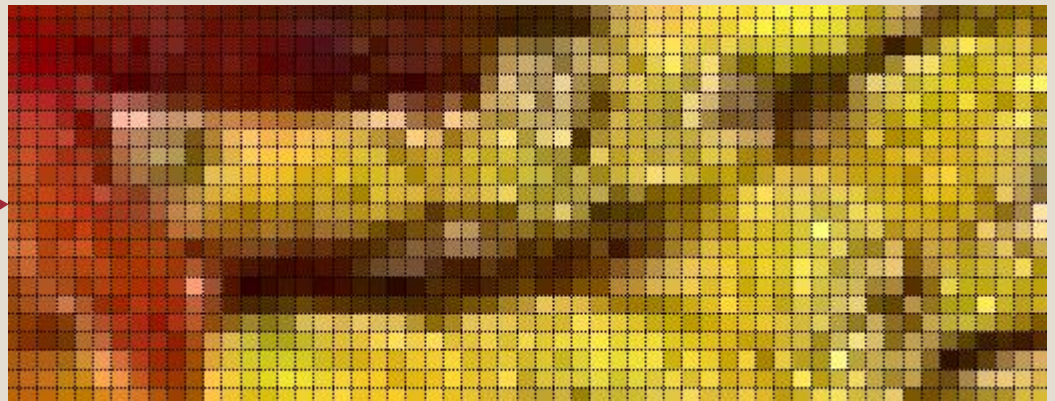
POČÍTAČOVÁ GRAFIKA

- Vyžaduje používání sofistikovaných nástrojů pro přípravu „grafických dat“
 - Adobe Photoshop, Autodesk Maya, AutoCAD, Corel Draw, Adobe Illustrator, ...
- Vyžaduje programování grafiky
 - Typicky požadována interaktivní grafika
 - Typicky alespoň 50 snímků za vteřinu
=> nutnost efektivních algoritmů
 - Programování základních algoritmů
 - Viz např. KIV/ZPG, KIV/KPG, KIV/APG, KIV/VD, KIV/ZVI, ...
 - Programování aplikací s využitím knihoven
 - GDI+, OpenGL, DirectX, WPF, VTK, ITK, ...
 - KIV/UPG 😊 a samozřejmě další



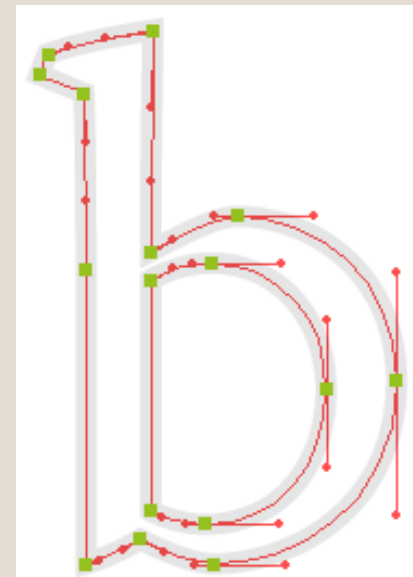
BITMAPOVÁ A VEKTOROVÁ GRAFIKA

- Bitmapová (rastrová grafika)
 - Obrázek složen z 2D matice diskrétních pixelů
 - Pixel = Picture Element = obrazový element
 - Pixel nese informaci o barevné hodnotě
 - může se měnit v čase => video
 - Většina současných zařízení
 - Digitální fotoaparát, scanner, LCD display, stolní tiskárny, ...
 - Vysoké nároky na velikost úložiště



BITMAPOVÁ A VEKTOROVÁ GRAFIKA

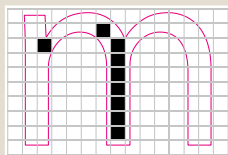
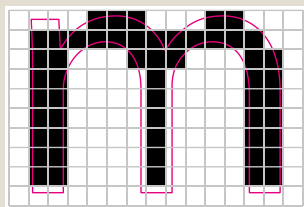
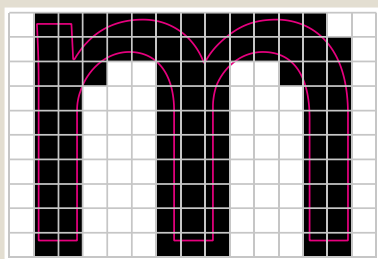
- Vektorová (objektově orientovaná) grafika
 - Obrázek složen z matematicky definovatelných objektů
 - Např. úsečka, kružnice, křivka, ...
 - Objekty popsány různými vlastnostmi
 - Např. umístění v prostoru, poloměr, barva, ...
 - Podporováno minimem zařízení
 - Plotter, řezací plotter, CNC stroje, ...
 - Typicky nutná konverze na bitmapovou grafiku
 - Jen těžko ovlivnitelné
 - Nároky na velikost úložiště závisí na obsahu



BITMAPOVÁ A VEKTOROVÁ GRAFIKA

■ Vektorová grafika

- Lze vykreslit kvalitně v „libovolné“ velikosti



Název písma: MS Sans Serif
1234567890.:; ' " (!?) +-*/=

12 Příliš žluťoučký kůň úpěl ďábelské ódy

18 Příliš žluťoučký kůň úpěl ď

24 Příliš žluťoučký kůň

36 Příliš žluťouč

48 Příliš žluťo

60 Příliš žlu

72 Příliš ž

Název písma: Arial
Verze: Version 6.81
Rozložení OpenType, Digitálně podepsáno

abcdefghijklmnopqrstuvwxy
1234567890.:; ' " (!?) +-*/=

12 Příliš žluťoučký kůň úpěl ďábelsk

18 Příliš žluťoučký kůň

24 Příliš žluťoučký

36 Příliš žluťo

48 Příliš žlu

60 Příliš ž

72 Příliš

BITMAPOVÁ A VEKTOROVÝ GRAFIKA

- Vektorová grafika
 - Snadná úprava
 - Lze pracovat s objekty nezávisle
- Často kombinace obojího
 - Vektorový obraz, ve kterém jedním z objektů je vložený bitmapový obraz (nativní nebo jiné rozlišení)

PROGRAMOVÁNÍ APLIKACÍ S GRAFICKÝM VÝSTUPEM

- Úloha: zobrazit vektorovou, bitmapovou, případně smíšenou grafiku na daném zařízení
 - Např. na obrazovce nebo tiskárně
- Přímé řešení: použít funkcionality poskytovanou ovladačem zařízení (dodán výrobcem)
- Problémy:
 - Některá zařízení umí zpracovávat jen bitmapovou grafiku => vektorovou grafiku musíme zkonvertovat
 - Rozhraní ovladačů pro různé typy zařízení se mohou lišit
 - Různý souřadný systém => složitý kód, abychom dosáhli stejného výsledku

PROGRAMOVÁNÍ APLIKACÍ S GRAFICKÝM VÝSTUPEM

Fyzické zařízení
(např. tiskárna)

Fyzická oblast
A4 (210x297 mm)
např. 4960x7015 px

„Plátno“

Zobrazitelná oblast

Podporovaný
rozsah souřadnic

$X = 3720$,
Šířka = 705

Fyzické zařízení
(např. obrazovka)

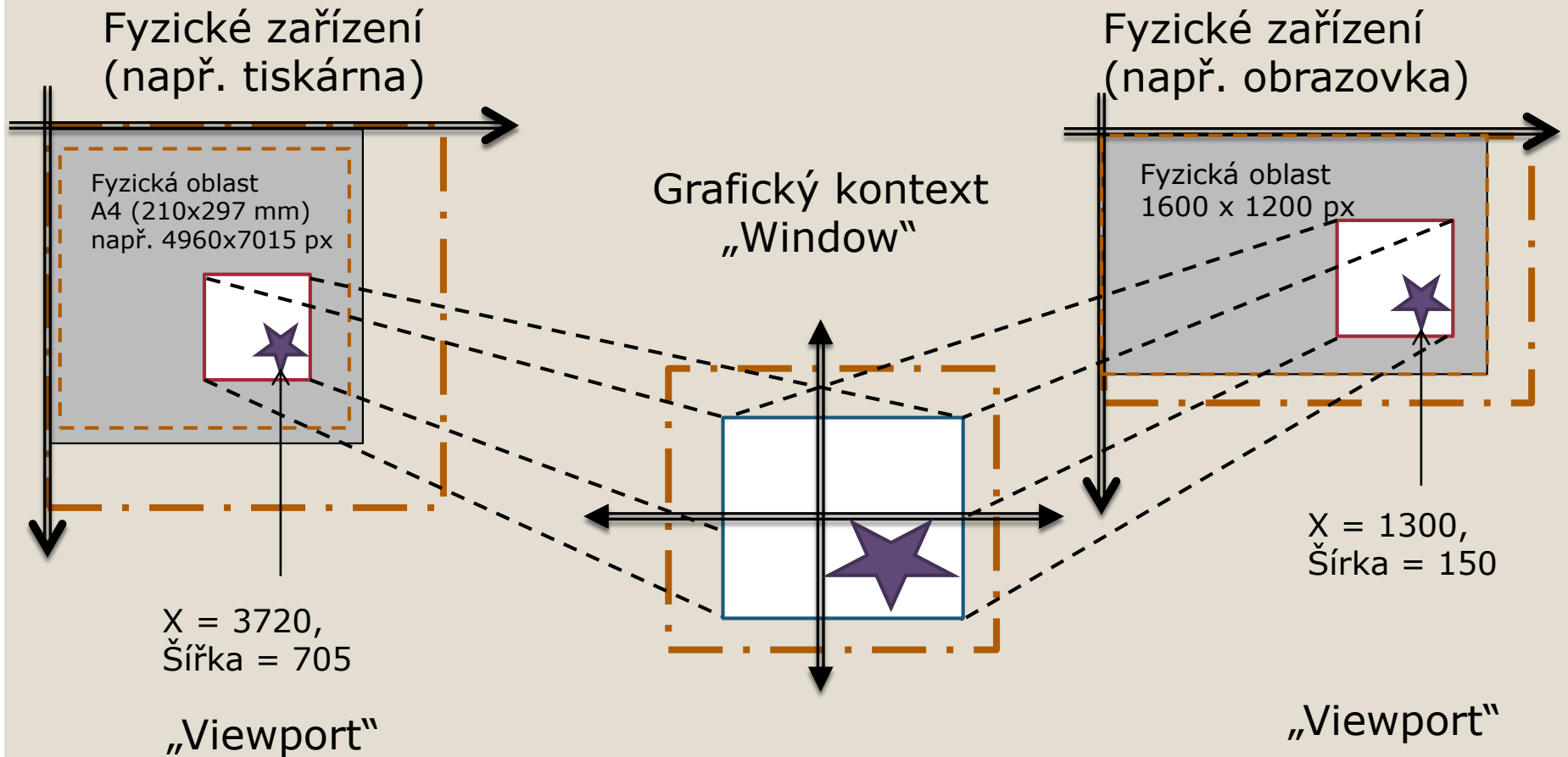
Fyzická oblast
1600 x 1200 px

$X = 1300$,
Šířka = 150

PROGRAMOVÁNÍ APLIKACÍ S GRAFICKÝM VÝSTUPEM

- Nepřímé řešení: „abstraktní“ grafický kontext
 - Taktéž zvaný jako „device context“
- Grafický kontext poskytuje jednotný přístup k libovolnému zařízení (pro grafický výstup)
- Programátor o vytvoření požádá operační systém
- Dvě možnosti:
 - Prostřednictvím API OS
 - WINAPI funkce *CreateDC*
 - Nutno specifikovat ovladač zařízení
 - WINAPI funkce *GetDC*
 - Poskytne již vytvořený grafický kontext (vytvořený nějakou utilitou)
 - Prostřednictvím nějaké dostupné knihovny
 - Např. MFC, VCL, AWT, SWING, Qt, wxWidgets, ...
 - PPA1: DrawingTools

PROGRAMOVÁNÍ APLIKACÍ S GRAFICKÝM VÝSTUPEM



GRAFICKÝ KONTEXT

- Grafický kontext má
 - Identifikátor poskytnutý operačním systémem
 - Windows: HDC
 - Vlastní souřadný systém
 - Označovaný též jako logický systém nebo „user space“
 - Nastavenou tzv. „Window“ – „Viewport“ transformaci
 - Mapování na oblast v souřadném systému zařízení
 - Zahrnuje posun a změnu měřítka
 - Stav
 - Aktuálně zvolené pero (obrysy) a štětec (výplň)
 - Aktuální poloha pera
 - Ořezová oblast
 - ...
 - Asociovány funkce pro zobrazení smíšené grafiky
 - Např. WINAPI funkce *LineTo*(HDC hdc, int xEnd, int yEnd)

GRAFICKÝ KONTEXT

- Grafický kontext bývá „zapouzdřen“ v grafických knihovnách, tj. jedná se obvykle o třídu

Knihovna	Jazyk	OS	Grafický kontext
GDI	nativní	Windows	WINAPI funkce
GDI+	C++	Windows	Graphics (Gdiplus)
MFC	C++	Windows	CDC, CClientDC
wxWidgets	C++	multiplatformní	wxDC, wxClientDC, wxPaintDC
WinForms	.NET	Windows, Mac OS X, Linux,	Graphics (System.Drawing)
WPF	.NET	Windows	DrawingContext (System.Windows.Media), Graphics (System.Drawing)
Qt	C++, Javascript	multiplatformní	QPainter

GRAFICKÝ KONTEXT

- Grafický kontext bývá „zapouzdřen“ v grafických knihovnách, tj. jedná se obvykle o třídu

Knihovna	Jazyk	OS	Grafický kontext
AWT	Java	Multiplatformní	Graphics, Graphics2D (java.awt)
Swing	Java	Multiplatformní	Graphics, Graphics2D (java.awt)
JavaFX	Java	Multiplatformní	Scene (javafx.scene)
DrawingTool	Java	Multiplatformní	DrawingTool (ppa1)
HTML5	JavaScript	Multiplatformní	CanvasRenderingContext2D

PROGRAMOVÁNÍ APLIKACÍ S VÝSTUPEM NA OBRAZOVKU

- Umístění a velikost „viewport“ určeno uživatelem
 - Možnost mít zobrazeno více aplikací najednou na monitoru
- Aplikace typicky musí umožnit interakci s uživatelem
 - myš, klávesnice, ...
 - Např. aplikaci lze ukončit
- Různé aplikace musí mít standardní základní chování
- Řešení: operační systém nabízí základní chování

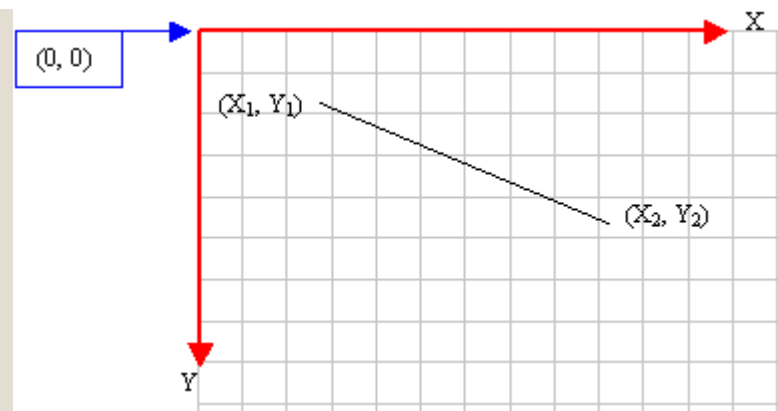
PROGRAMOVÁNÍ APLIKACÍ S VÝSTUPEM NA OBRAZOVKU

- Základem aplikace je okno => tzv. okenní aplikace
- Okno obsluhováno operačním systémem
- Programátor o vytvoření okna požádá OS
- Dvě možnosti:
 - Prostřednictvím API OS
 - Např. WINAPI funkce *CreateWindow*
 - Prostřednictvím nějaké dostupné knihovny
 - Např. MFC, VCL, AWT, SWING, Qt, wxWidgets, ...
 - PPA1: DrawingTools

PROGRAMOVÁNÍ APLIKACÍ S VÝSTUPEM NA OBRAZOVKU

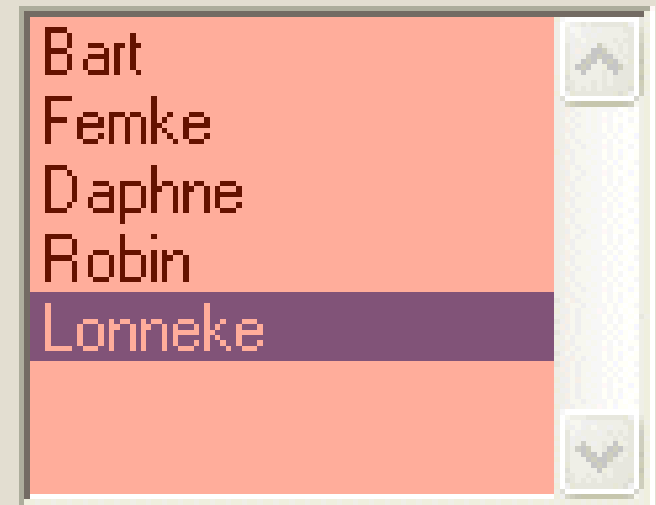
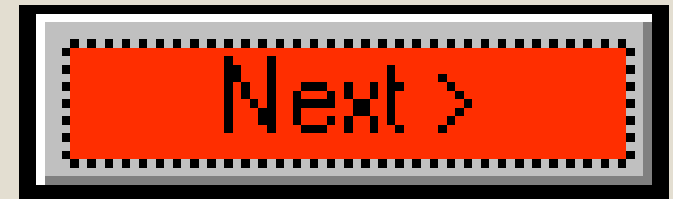
■ Okno má

- Pozici na obrazovce
 - Resp. pozici vůči svému rodiči
- Šířku a výšku
- Název (titulek)
- Další atributy
 - typ okna, průhlednost, ...
- Identifikátor
 - Přidělen operačním systémem
 - Windows: HWND
- **Grafický kontext pro obrazovku**
 - Vytvořen automaticky operačním systémem
 - Počátek souřadnic $(0,0)$ odpovídá levému hornímu rohu okna

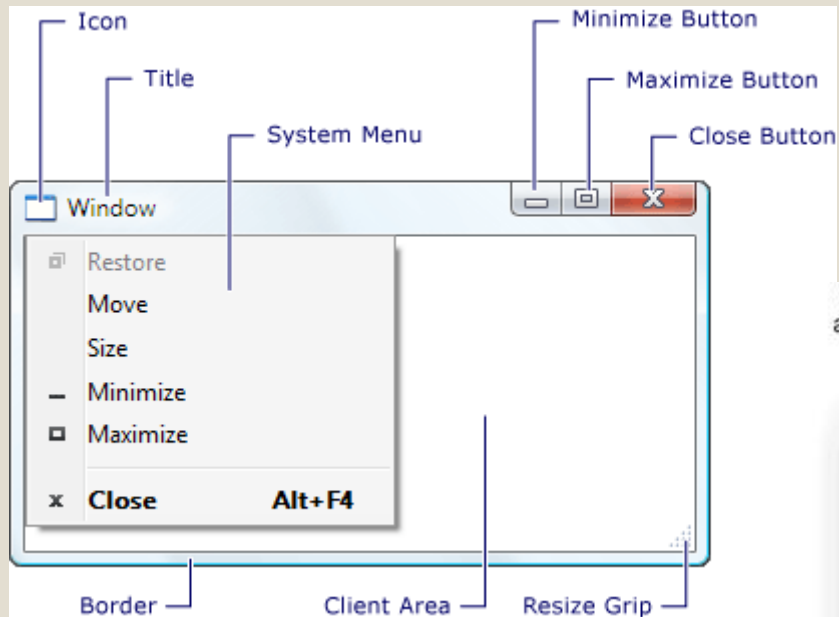


PROGRAMOVÁNÍ APLIKACÍ S VÝSTUPEM NA OBRAZOVKU

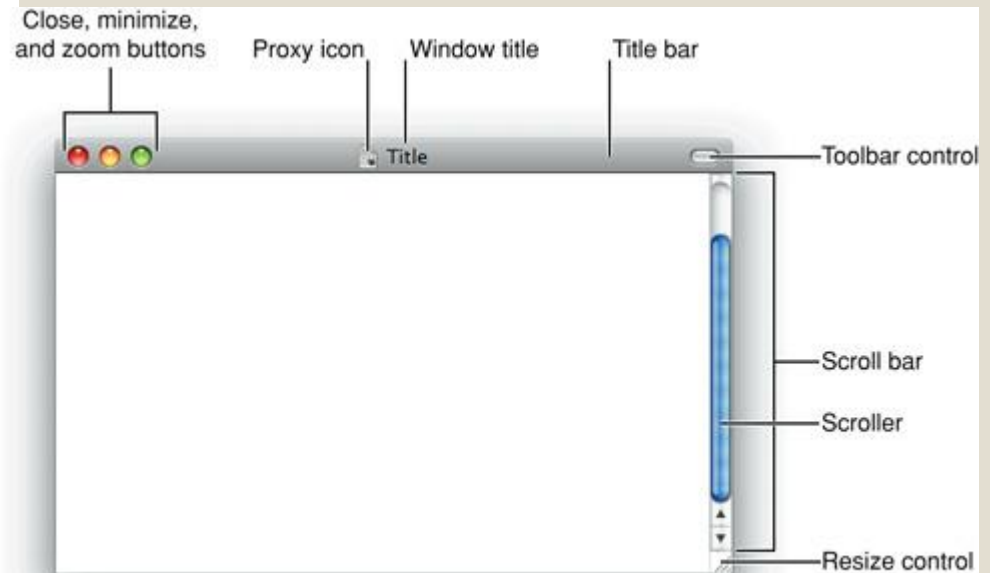
- Okno obsahuje grafický obsah
- Dvě části:
 - Rámeček (ohraničení)
 - „non-client“ area
 - Klientská oblast



PROGRAMOVÁNÍ APLIKACÍ S VÝSTUPEM NA OBRAZOVKU



■ Hlavní okno aplikace



PROGRAMOVÁNÍ APLIKACÍ S VÝSTUPEM NA OBRAZOVKU

- Rámeček vykreslován obvykle systémem, resp. knihovnou, která je použita
- Klientská oblast obsahuje obvykle
 - Komponenty (tlačítka, zaškrtnutá, seznamy, ...)
 - O vykreslení se postará kód komponent
 - Vizualizace (graf, náčrt, ...)
 - Programátor musí napsat kód pro vykreslení
 - Někdy definován grafický kontext přímo pro klientskou oblast
 - Grafický kontext získán od systému pro okno
 - Do něj se vykresluje
 - Obsah grafického subsystému (DirectX, XNA, OpenGL)
 - Vektorová primitiva (čáry, křivky, ...)
 - Bitmapový obrázek, ...

PROGRAMOVÁNÍ APLIKACÍ S VÝSTUPEM NA OBRAZOVKU

```
INT WINAPI WinMain(...)
{
    HWND          hWnd;
    MSG           msg;
    WNDCLASS      wndClass;

    //Registrace okna a jeho správy
    wndClass.lpszClassName = "GettingStarted";
    wndClass.lpfnWndProc   = WndProc;
    ...

    RegisterClass(&wndClass);

    //Vytvoření okna a jeho zobrazení
    hWnd = CreateWindow("GettingStarted",...);

    ShowWindow(hWnd, iCmdShow);
    UpdateWindow(hWnd);

    //Smyčka zpráv: správa okna
    while(GetMessage(&msg, NULL, 0, 0))
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }

    return msg.wParam;
} // WinMain
```

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT msg, ...)
{
    HDC          hdc;
    PAINTSTRUCT  ps;

    //Správa událostí okna
    switch(msg)
    {
    case WM_PAINT:
        //Získání grafického kontextu okna
        hdc = BeginPaint(hWnd, &ps);
        OnPaint(hdc);
        EndPaint(hWnd, &ps);
        return 0;

    ...
    } // WndProc
```

```
VOID OnPaint(HDC hdc)
{
    //Kreslení prostřednictvím Grafického
    kontextu
    SelectObject(hdc,
        GetStockObject(DC_PEN));
    SetDCPenColor(hdc, RGB(0, 0, 255));
    MoveToEx(hdc, 0, 0, NULL);
    LineTo(hdc, 200, 100);
}
```

PROGRAMOVÁNÍ APLIKACÍ S VÝSTUPEM NA OBRAZOVKU

```
package FirstJavaFX;

import javafx.application.Application;
import javafx.scene.*;
import javafx.scene.paint.*;
import javafx.scene.shape.*;
import javafx.stage.Stage;

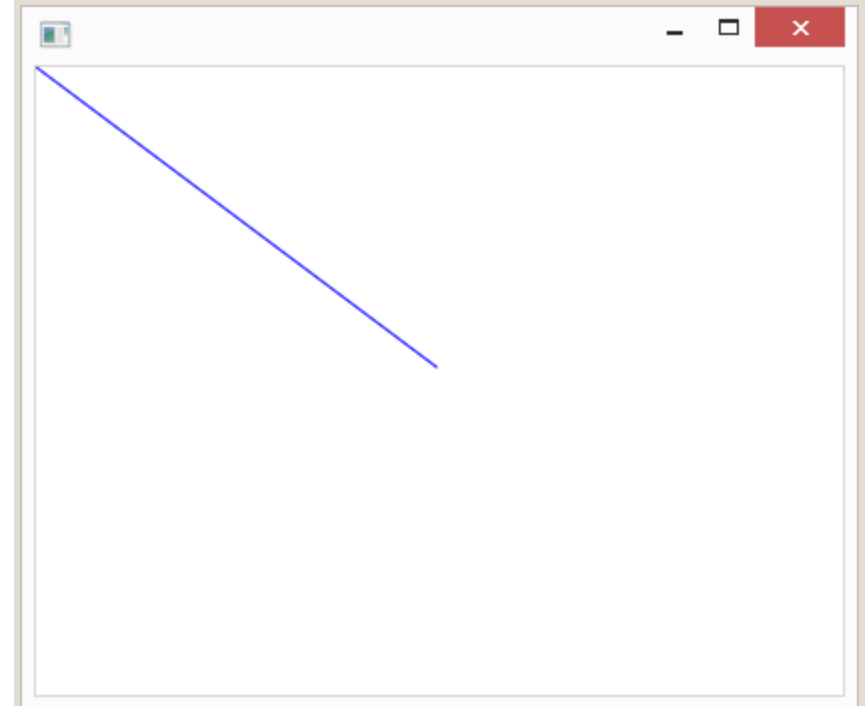
public class FirstJavaFX extends Application {

    /**
     * @param args
     */
    public static void main(String[] args) {
        launch(args); //registrace a vytvoření okna
    }

    @Override
    public void start(Stage stage) throws Exception {
        //Kreslení skládáním na grafický kontext
        Group root = new Group();
        Line line = new Line(0, 0, 200, 150);
        line.setStroke(Color.BLUE);
        root.getChildren().add(line);

        //Získání grafického kontextu okna
        Scene scene = new Scene(root, 500, 500);
        stage.setScene(scene);

        //Zobrazení okna
        stage.show();
    }
}
```



KRESLENÍ GRAFICKÝCH OBJEKTŮ

- Grafické objekty složeny z grafických primitiv (např. úsečka, kružnice, ...)
- Dva základní přístupy
 - Přímý přístup: vykresluji postupně primitiva a vizuálně skládám požadovaný výsledek (malíř)
 - Nepřímý přístup: nadefinuji geometrický objekt složením z primitiv a poté nechám objekt vykreslit jako celek

KRESLENÍ GRAFICKÝCH OBJEKTŮ

- Přímý přístup
 - Kresebně orientovaný
 - Jednodušší, ale omezené možnosti
 - Dvě varianty:
 - Pixelový přístup
 - Logický přístup



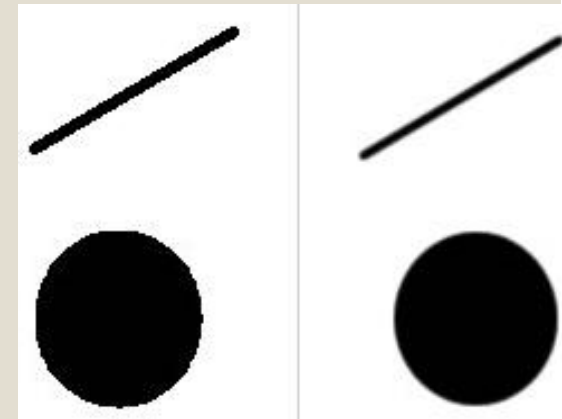
KRESLENÍ GRAFICKÝCH OBJEKTŮ

■ Pixelový

- Nejstarší, velmi rychlý, vhodný např. pro LED reklamní panely
- Práce přímo s pixely (celočíselné hodnoty)
- Práce prostřednictvím velmi jednoduchých metod (funkcí)
 - Např. setPixel, line, ...
- Zubatý vzhled
- Např. GDI

■ Logický

- Délkové jednotky libovolné (reálná čísla)
- Převod na pixely proveden automaticky grafickým systémem
- Práce prostřednictvím kreslících příkazů
- Kvalitní vzhled, lze nakreslit téměř cokoliv
- Vyžaduje dostatečně výkonný hardware
- Např. GDI+, Java.awt.Graphics2D, ...



KRESLENÍ GRAFICKÝCH OBJEKTŮ

- Nepřímý přístup

- Objektově (na scéně) orientovaný
- Složitější, ale mohu s objekty před vykreslením pracovat
- Vhodné pro interaktivní grafiku, množinové operace, ...
- Výpočetně náročné, nelze nakreslit snadno vše
- Např. JavaFX, WPF



OKENNÍ APLIKACE V JAVĚ

- Java nezávislá na platformě => systémový přístup „není možný“, je třeba užít nějakou knihovnu
- AWT
 - Prvotní knihovna
 - Omezená funkčnost
 - Jmenný prostor: `java.awt`
- **Swing**
 - Rozšiřuje AWT
 - Jmenný prostor: `javax.swing`

OKENNÍ APLIKACE V JAVĚ

■ JavaFX

- Nástupce Swing
- Jmenný prostor: `javafx.*`
- Vektorová grafika realizována primárně skládáním objektů
- Podpora multimediálních prvků
- Java Windows Presentation Foundation (WPF)

■ DrawingTool

- Postaveno nad Swing
- Zjednodušuje vytvoření okna
- Vhodné pro začátečníky z PPA1, jinak k ničemu

OKENNÍ APLIKACE VE SWINGU

- Hlavní okno = třída `javax.swing.JFrame`
- Vykreslení grafického obsahu
 - Virtuální metoda `void paint(Graphics g)`
 - Metodu třeba přepsat => vytvořit vlastní třídu odděděnou od třídy `javax.swing.JFrame`

```
import java.awt.Graphics;
import java.awt.Graphics2D;
import javax.swing.JFrame;
public class MojeOkno extends JFrame {
    public void paint(Graphics g){
        Graphics2D g2 = (Graphics2D)g;
        /** Kreslení voláním metod třídy Graphics2D*/
    }
}
```

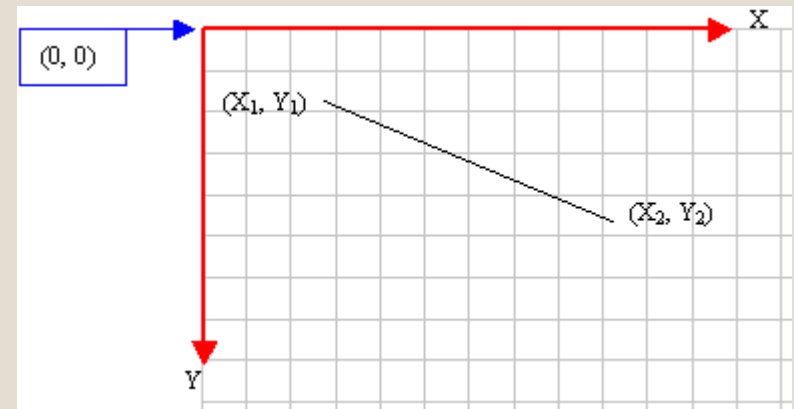
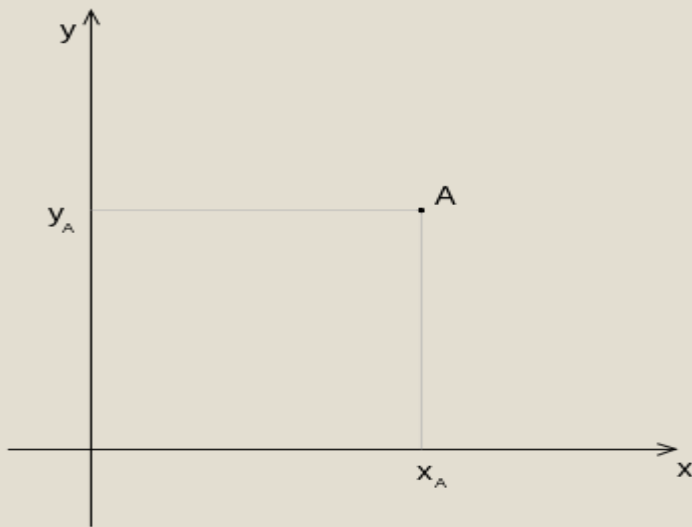
OKENNÍ APLIKACE VE SWINGU

■ Vytvoření okna

```
public static void main(String[] args) {  
    MojeOkno okno = new MojeOkno();  
    okno.setTitle("Muj nazev okna");  
    okno.setSize(640, 480);  
    okno.setLocation(0, 0);  
    //okno.setLocationRelativeTo(null);  
  
    okno.setDefaultCloseOperation(EXIT_ON_CLOSE);  
    okno.setVisible(true);  
}
```

SOUŘADNÝ SYSTÉM

- Prakticky výhradně kartézský systém souřadnic
 - $(0, 0)$ buď v levém dolním rohu obrazu nebo levém horním rohu obrazu (např. pro obrazovku)
 - Levý horní roh = tradiční pozice počátku



DÉLKOVÉ JEDNOTKY

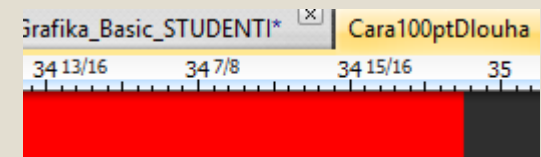
■ Pixel (px)

- Typicky jediné jednotky akceptované výstupními zařízeními
- Problém: fyzická velikost pixelu závisí na zařízení
- Řešení: zavedení a používání jiných jednotek
 - Nutný převod z jiných jednotek na pixely
 - Často skryto programátorovi

■ Fyzické jednotky

- Milimetr, centimetr
- Palec (inch, in)
 - 1 inch = 25.4 mm
- DTP bod (Point, pt, b)
 - 1 pt = 1/72 palce
- Device-independent pixel (DIP)
 - 1 DIP = 1/96 palce

```
g2.setStroke(new BasicStroke(5,  
BasicStroke.CAP_BUTT, BasicStroke.JOIN_MITER));  
g2.setColor(Color.red);  
g2.drawLine(0, 0, 99, 0);
```



DÉLKOVÉ JEDNOTKY

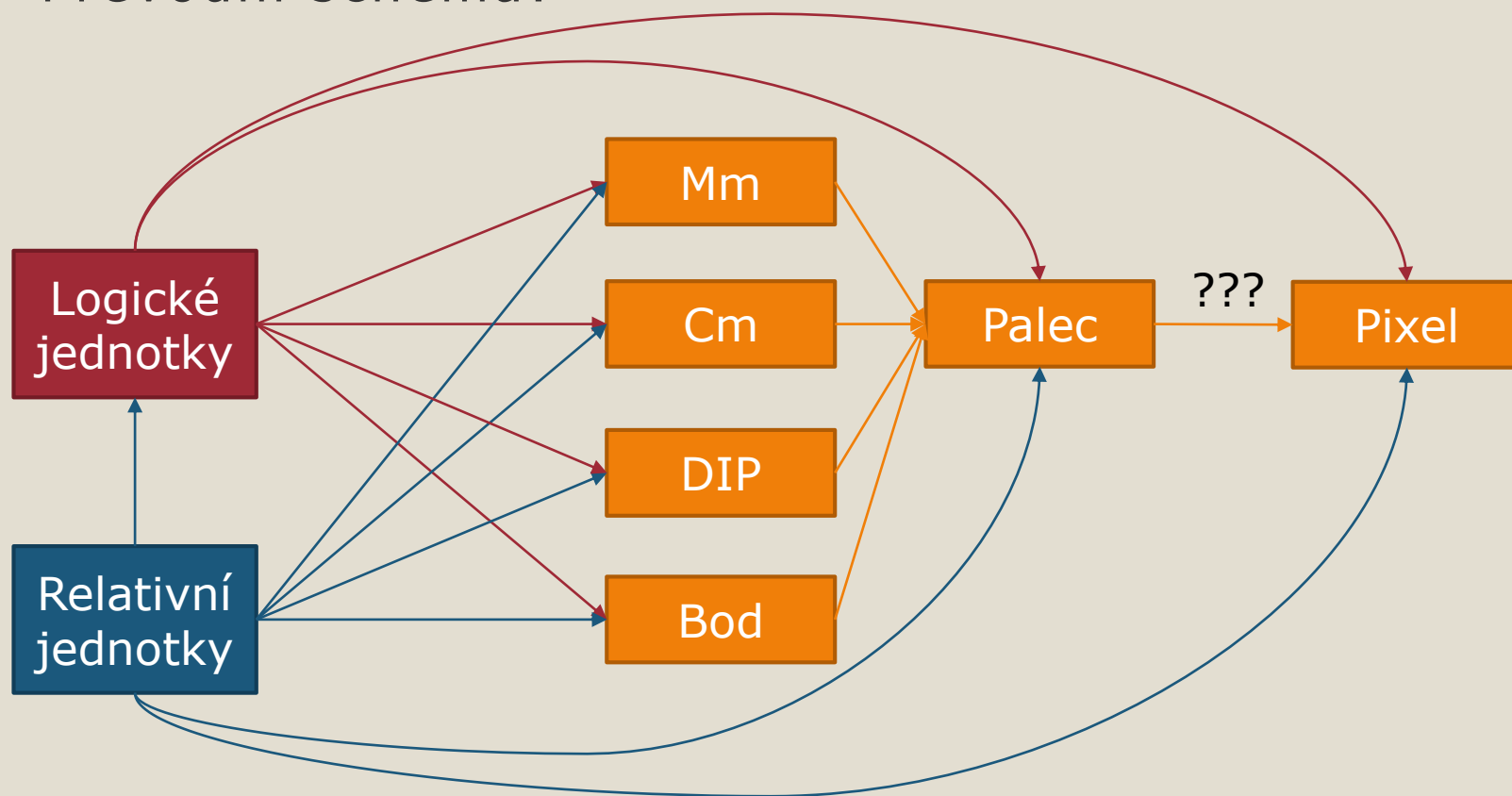
- Logické jednotky
 - Např. násobky 0.01 palce, 0.1 mm, ...
- Relativní jednotky
 - Procenta (např. celkové fyzické šíře obrazu)
 - E_m = velikost vztažená na velikost písmene M současně zvoleného písma

DÉLKOVÉ JEDNOTKY

- Grafické knihovny nepodporují nativně všechny možné jednotky => nutné přepočty v aplikaci
 - GDI a GDI+(rovněž .NET Graphics)
 - Standardní knihovny pro Windows rodinu
 - Logické jednotky
 - standardně nastaveno tak, že 1 logická jednotka = 1 pixel
 - Body pro velikosti písma
 - Java AWT, Swing
 - Pixely nebo body, body pro velikosti písma
 - Direct2D a Windows Presentation Foundation (WPF)
 - DIPs, body pro velikosti písma

DÉLKOVÉ JEDNOTKY

■ Převodní schéma:



DÉLKOVÉ JEDNOTKY

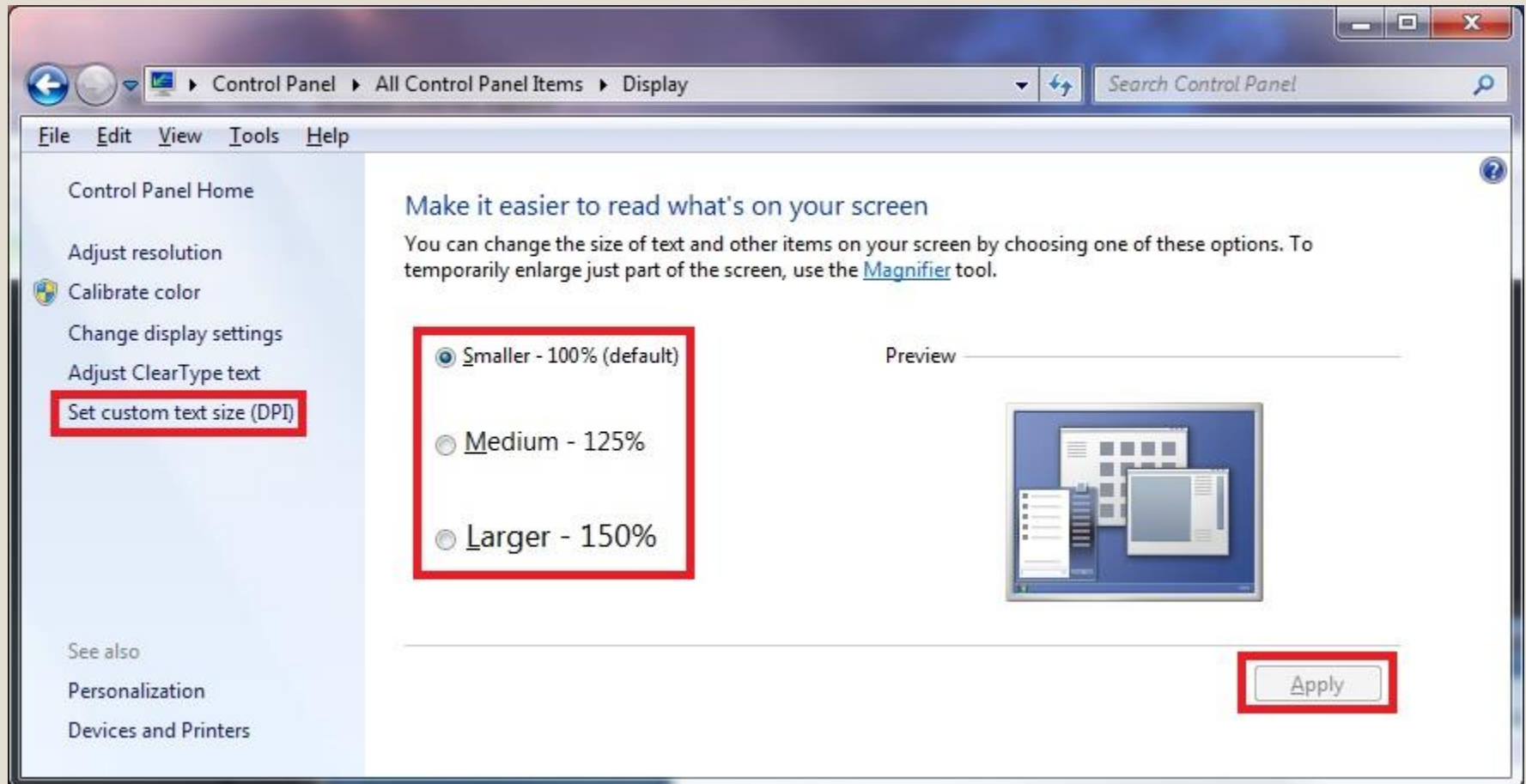
- Problém:
 - Fyzická velikost pixelu závisí na zařízení
- Pixels per inch (PPI)
 - Počet pixelů na palec
 - Jedná se o „obrazové rozlišení“
 - Typicky zaměňováno za DPI
 - Např. 150 DPI u scannerů obvykle znamená 150 PPI
- PPI dostupné pro „papír“, tj. tiskárny, scannery, ...
 - výstup měřitelný v SI jednotkách
 - 1 in = 25.4 mm

DÉLKOVÉ JEDNOTKY

- PPI není dostupné pro „obrazovku“
 - důvod:
 - velikost pixelu závisí na:
 - Fyzická velikost zařízení (např. 26" obrazovka)
 - Rozměrová velikost (též udáváno jako rozlišení) zařízení (např. 1600x1200)
 - důsledek: neexistuje možnost měřit výstup v jednotkách SI
 - řešení:
 - podvádíme a předpokládáme, že obrazovka má 96 PPI
 - 72 PPI v případě Mac OS
 - 1 in = 96 pixelů (při 96 PPI)



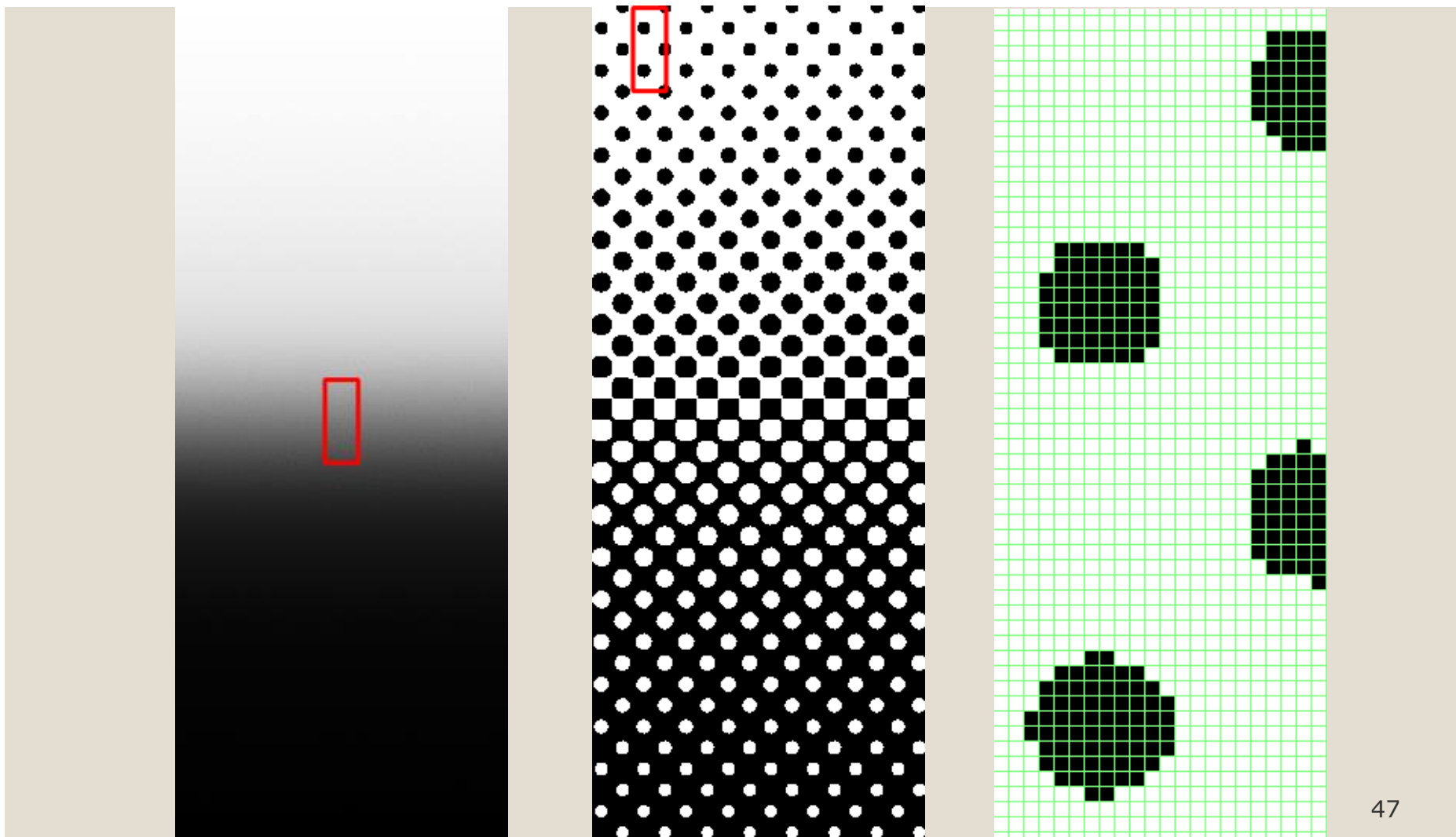
DÉLKOVÉ JEDNOTKY



DÉLKOVÉ JEDNOTKY

- Dots per inch (DPI)
 - Historicky starý a velmi rozšířený pojem
 - Aplikován na cokoliv související s rozlišením => nevyjadřuje vždy totéž
 - DPI uvedené pro jeden typ zařízení může znamenat úplně něco jiného, než DPI uvedené pro jiný typ zařízení
 - DPI pochází z tiskové problematiky
 - Pixel simulován několika puntíky (dots)
 - Jedná se tedy o „tiskové rozlišení“
 - DPI může být jiné pro osu x a y
- Lines per inch (LPI)
 - Pochází z tiskové problematiky
 - Počet obrazových „pixelů“ na palec

DÉLKOVÉ JEDNOTKY



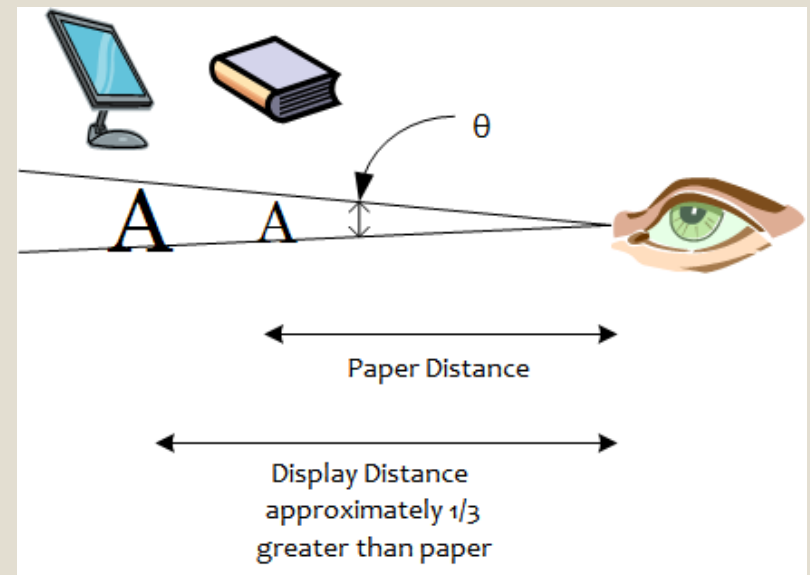
DÉLKOVÉ JEDNOTKY

- Monitory a DPI/PPI problém
 - Apple Macintosh 128K (1984)
 - Černobílá (1 bit) 9" obrazovka, zobrazovací plocha cca 7.11" x 4.75"
 - 512 x 342 pixelů
 - →72 PPI
 - Tiskárna Apple měla 72 DPI
 - Z konvenčních důvodů rozlišení monitoru udáváno jako 72 DPI
 - Význam: WYSIWYG



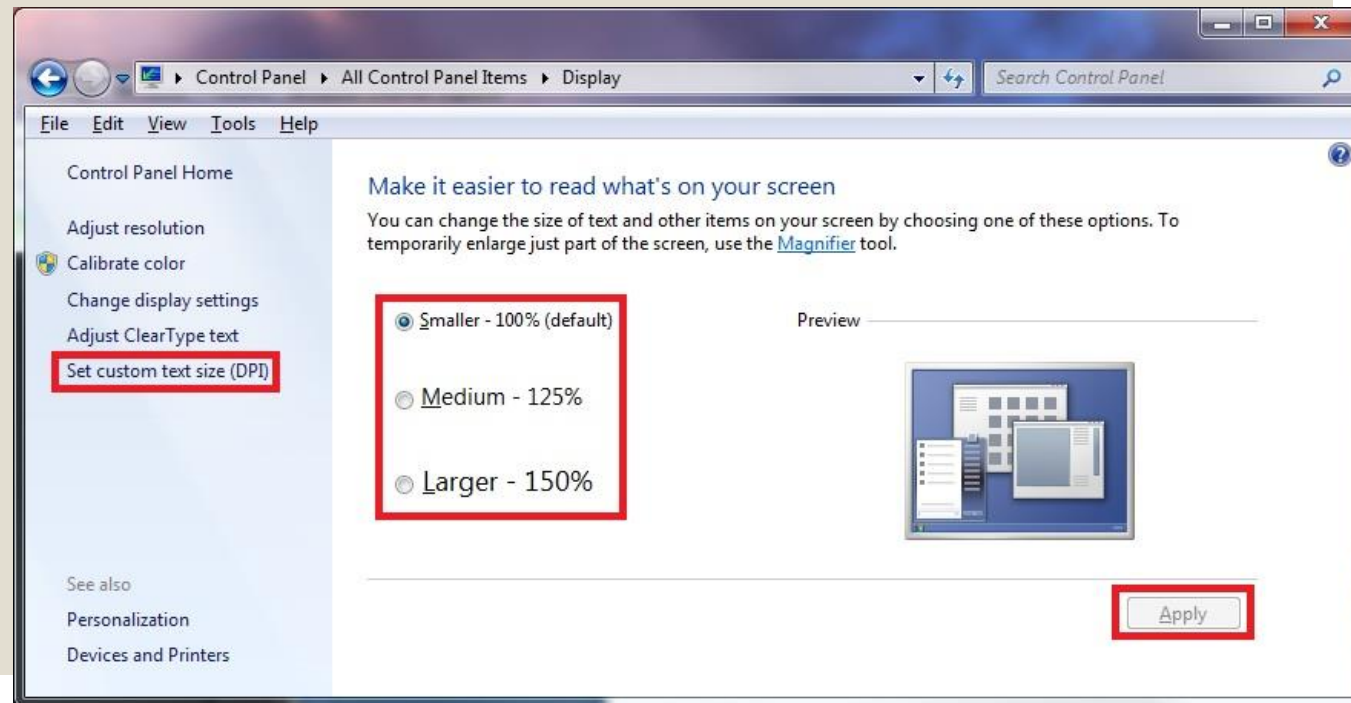
DÉLKOVÉ JEDNOTKY

- Microsoft: „při čtení textu z obrazovky je text cca o 1/3 dále od očí než při čtení téhož textu z papíru => pro dosažení stejného vjemu musí být text na obrazovce zobrazen o něco větší
 - $72 \text{ PPI} * (1 + 1/3) = 96 \text{ PPI}$
 - Normální rozlišení pod MS Windows stanoveno na 96 PPI



DÉLKOVÉ JEDNOTKY

- Dnes již DPI (PPI) pro monitory neodpovídá realitě
 - 96 DPI (PPI) neznamená, že čára o 96 pixelech bude zabírat 1 palec – její fyzickou délku nelze jednoduše stanovit
 - Klíčový je poměr, tj. nastavím-li pro display 120 DPI, bude to čára o 25% delší než v případě 96 DPI
 - MS Vista+ DPI již uživatel nenastavuje, nastavuje poměr



KRESLENÍ GRAFICKÝCH OBJEKTŮ

- Metody (funkce) pro vykreslení obrysu
 - Obvykle prefix „Draw“, např. *DrawRectangle*
 - Kreslí se aktuálně nastaveným perem
- Metody (funkce) pro vykreslení výplně
 - Obvykle prefix „Fill“, např. *FillRectangle*
 - Kreslí se aktuálně nastaveným štětcem
- Pro vykreslení vyplněného ohraničeného tvaru se obvykle zavolá napřed Fill, pak Draw

Text

Text



PERO

- Pero (Pen, Stroke) slouží pro vykreslení obrysu
- Lze definovat: šířka, styl čáry, způsob napojení a zakončení, barva
- Šířka
 - Zadávána v pixelech
 - Některé aplikace v bodech (např. MS Office)
 - 1 pt —————
 - **BONUSOVÁ OTÁZKA**

1 px

2 px

5 px

10 px

20 px

PERO

■ Styl čáry

- Plná (solid), čárkovaná (dashed), tečkovaná (dotted), ...
- Některé knihovny mají pevně nastaveno jiné (např. AWT) umožňují uživatelsky definovatelný vzor

———— — — — —

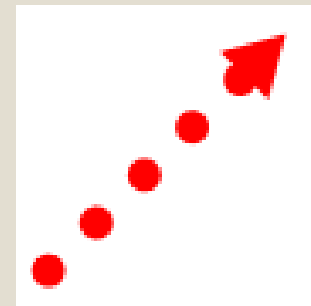
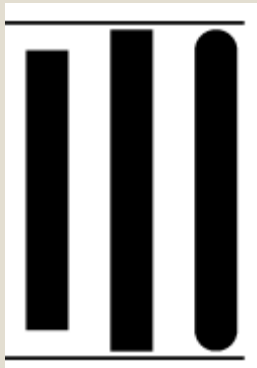
■ Způsob napojení čáry

- Určuje, jak se napojí na sebe dvě úsečky polygonu
- Často označováno jako miter, bevel, round, none
 - <http://www.html5canvastutorials.com/tutorials/html5-canvas-line-joins/>





PERO

- Způsob zakončení (cap)
 - Určuje, jak bude vypadat konec čáry
 - Má smysl jen pro čáry širší 1 px
 - Často flat (butt), square, round
 - Někdy též zakončení „roundanchor“, „arrowanchor“, ...
 - <http://www.html5canvastutorials.com/tutorials/html5-canvas-line-caps/>



PERO

■ Barva

- Obvykle plná barva, např. červená 
- Některé knihovny (např. WPF) umožňují gradientní přechod 
- Některé (např. WinForms) dokonce obrázky



PERO V JAVĚ

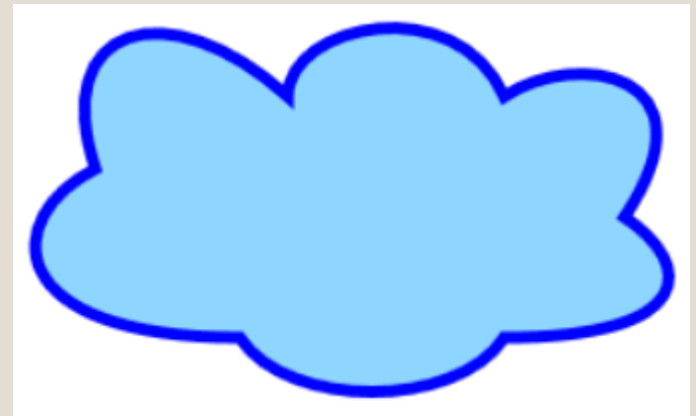
- Pero = třída BasicStroke
- Pero nastaveno v konstruktoru
 - Konstruktor přetížen
- Výběr pera: metoda setStroke
- Výber barvy pera: metoda setColor
- Příklad:

```
final static float dash1[] = {10.0f};  
BasicStroke dashed = new BasicStroke(5.0f,  
    BasicStroke.CAP_BUTT,  
    BasicStroke.JOIN_MITER,  
    10.0f, dash1, 0.0f);
```

```
g2.setStroke(dashed);  
/* kreslení */
```

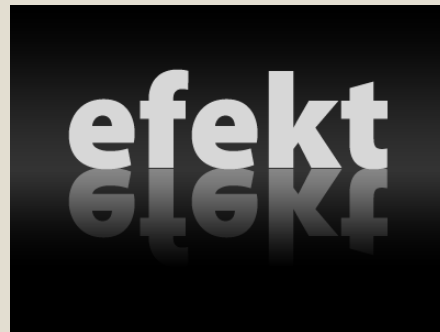
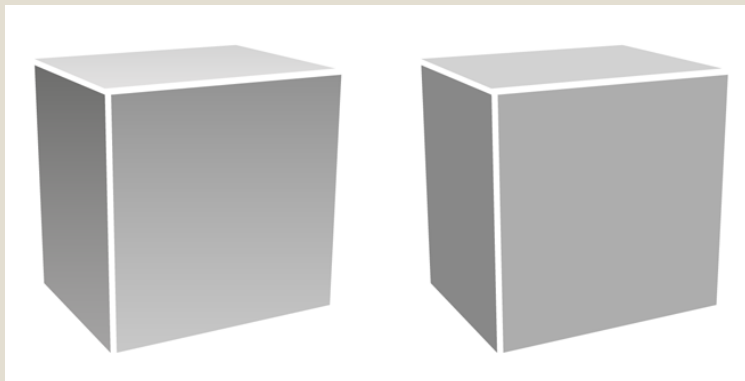
VÝPLŇ

- Výplň (brush, paint, fill) slouží pro vykreslení vnitřku
- Lze definovat: typ + další parametry dle typu
- Obvykle pro každý typ definována speciální třída
- Jednobarevná výplň
 - Označováno často jako „solid“
 - Nejjednodušší výplň
 - Postačí specifikovat barvu



VÝPLŇ

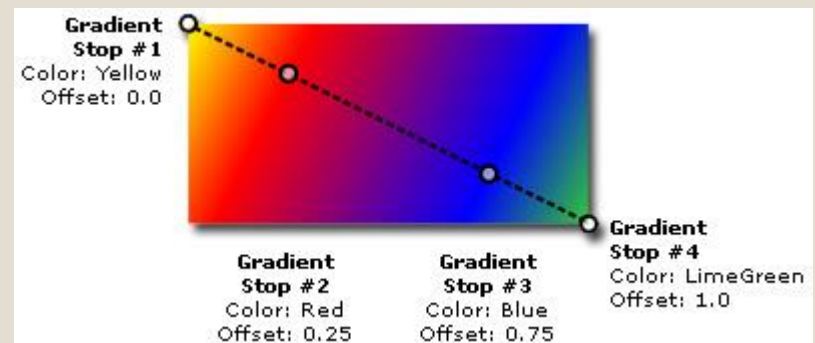
- Výplň lineárním gradientem



VÝPLŇ

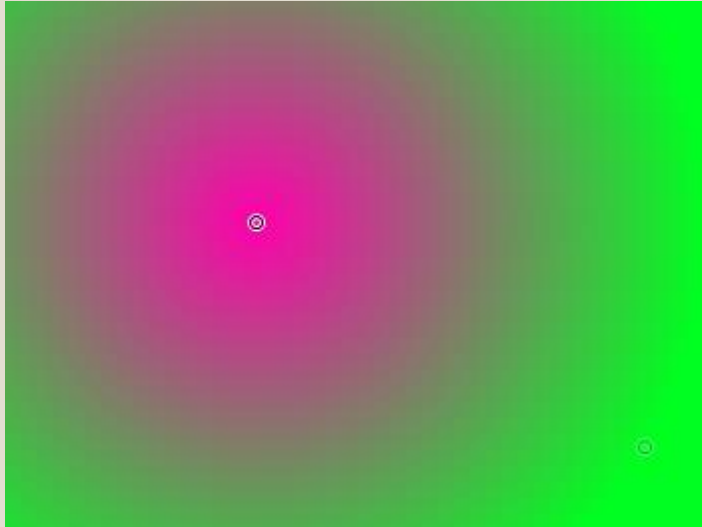
■ Výplň lineárním gradientem

- Barevný přechod mezi dvěma zadanými body
 - Počáteční a koncový bod
 - Může být zadán absolutně nebo relativně vůči čtverci 1x1
- Iso-čáry jsou přímky kolmé na spojnici těchto bodů
 - Iso-čára = místo, kde je stejná barva
- Nutno specifikovat barvy v obou bodech
- Některé knihovny umožňují definovat rovněž, jaká barva má být v dalším bodě na spojnici
 - Spojnice parametrizována na interval 0-1
 - Parametr offset

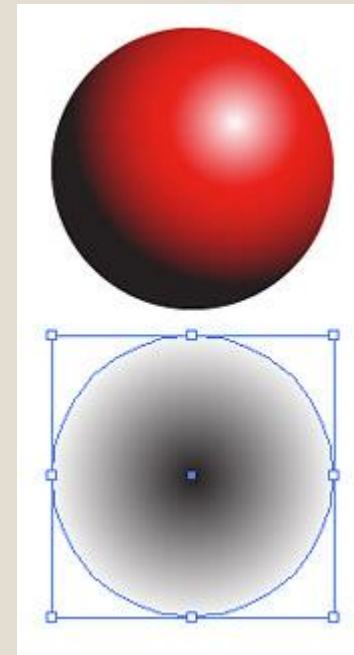


VÝPLŇ

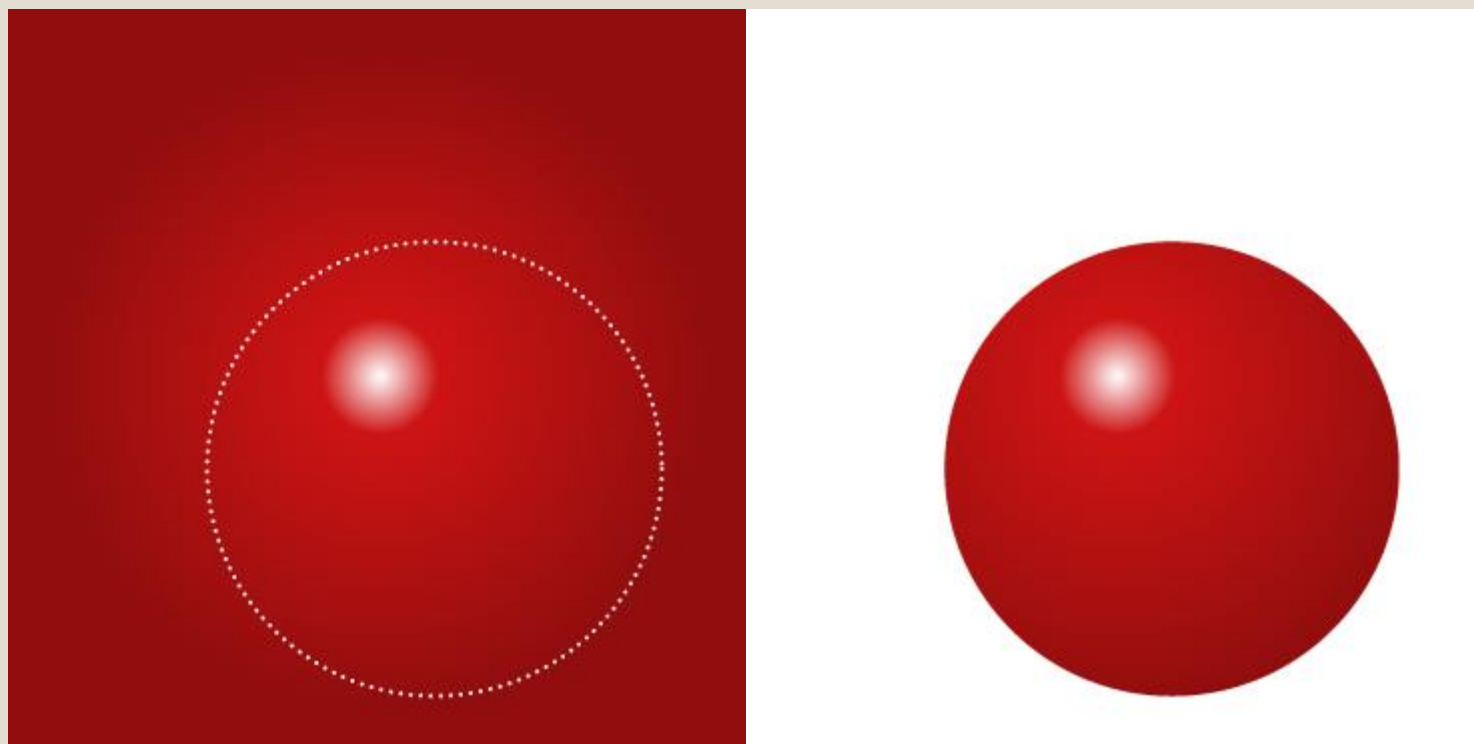
- Výplň radiálním gradientem
 - Obdoba výplně lineárním gradientem
 - Iso-čáry jsou kružnice se společným středem
 - Střed kružnice = počáteční bod



VÝPLŇ



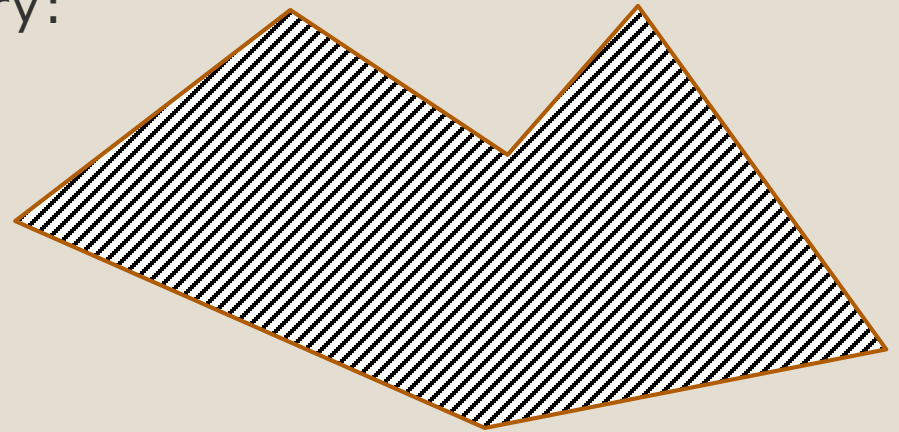
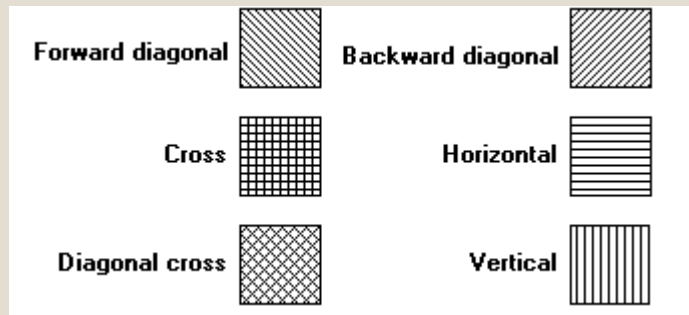
VÝPLŇ



VÝPLŇ

■ Výplň šrafováním

- Výplň jednobarevnými rovnoběžnými čarami pevného vzoru
- Význam pro technické výkresy (např. stavebnictví)
- Nejčastěji podporované vzory:

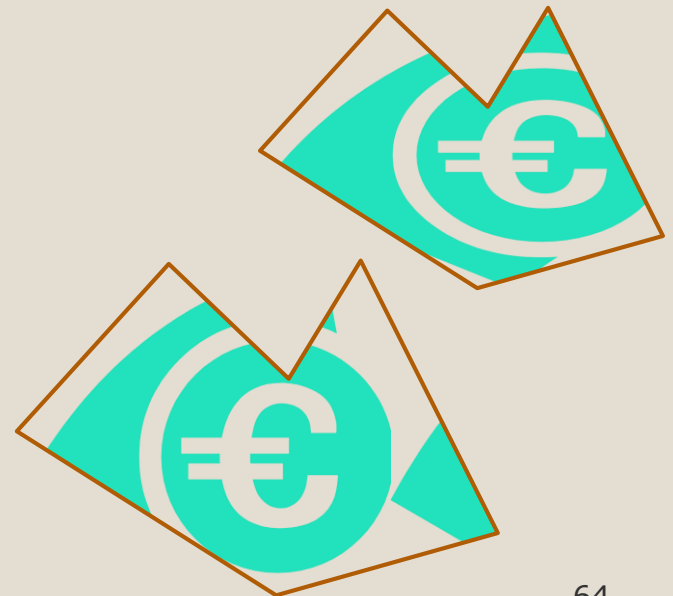
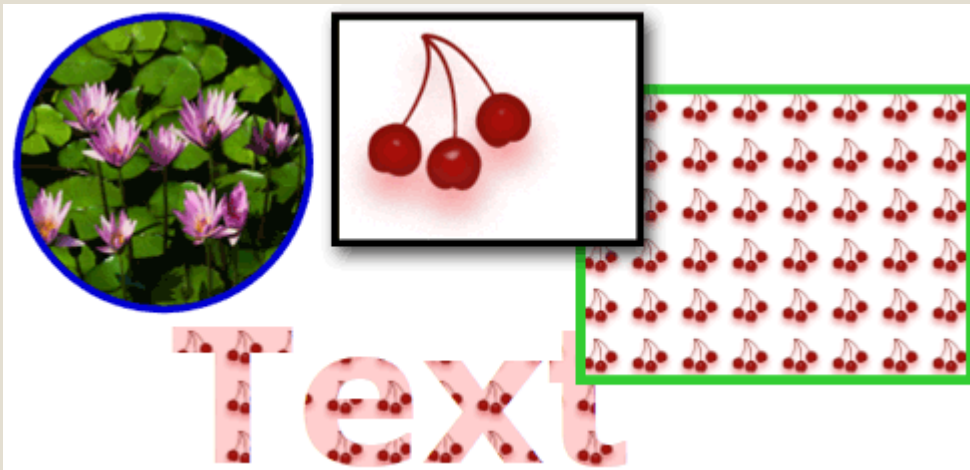


- Některé grafické knihovny tento styl štětce nepodporují, protože obdobného efektu lze dosáhnout štětcem s texturou
 - Např. Java AWT

VÝPLŇ

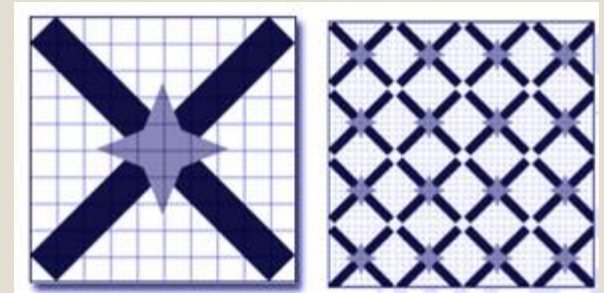
■ Výplň texturou

- Výplň provedena vzorem určeným bitmapovým obrázkem
 - Obrázek načten např. ze souboru
- Vzor může být zmenšen / zvětšen
 - Není nutné zachovat poměr stran
- Vzor se může opakovat (viz např. webové stránky)
- **OTÁZKA: ...**



VÝPLŇ

- Výplň kresbou („drawing“)
 - Obdoba výplně texturou
 - Vzor však určen příkazy pro vykreslení grafiky
 - Jedná se typicky o vektorový obraz
 - V podstatě je to rekurzivní přístup
 - Tento styl výplně obvykle nebývá v grafických knihovnách implementován
 - Lze vlastní implementace, ale není jednoduché



VÝPLŇ V JAVĚ

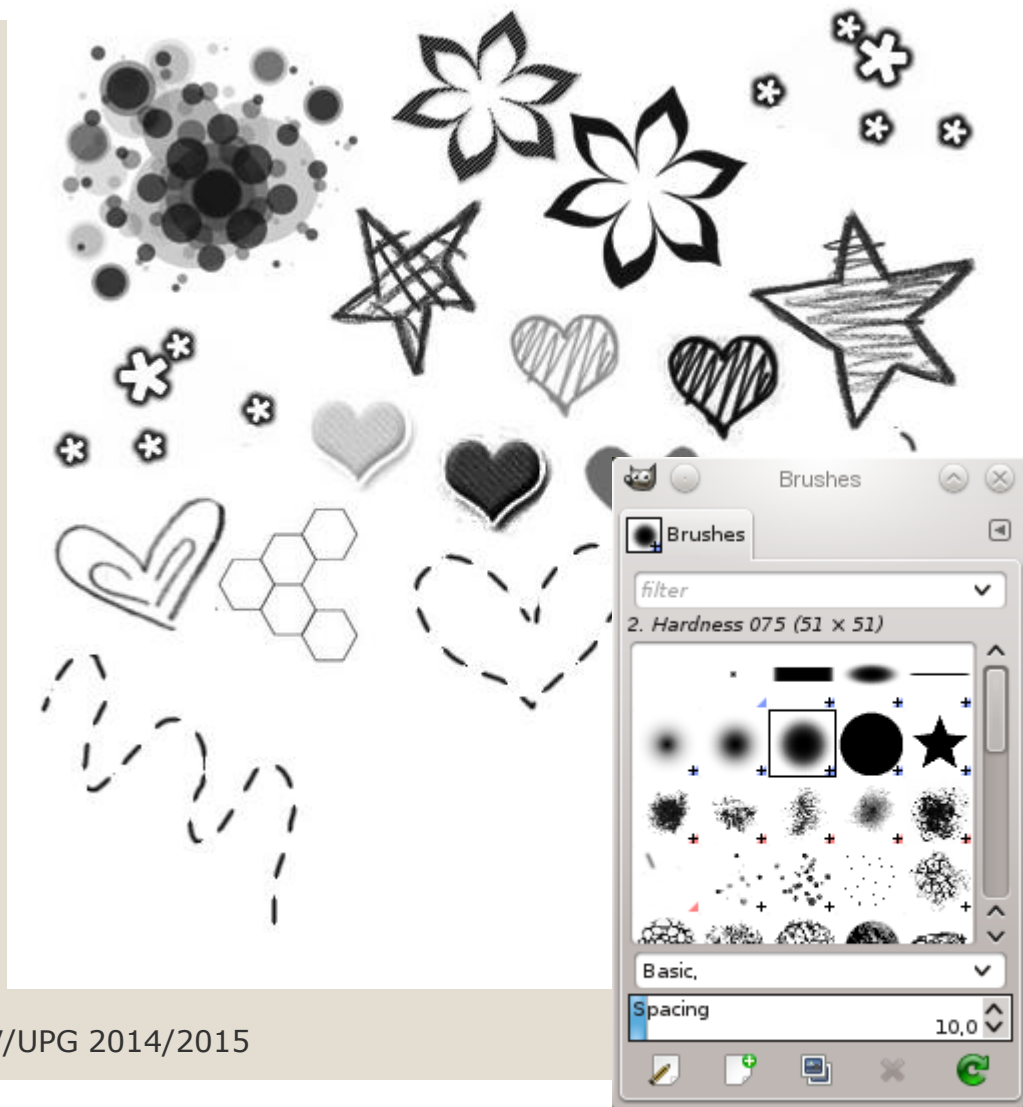
- Výplň = třída implementující rozhraní Paint
- Podporovány pouze tři typy výplně:
 - Jednobarevná výplň – třída *Color* (java.awt)
 - Lineární gradient – třída *GradientPaint*
 - Výplň texturou – třída *TexturePaint*
- Výběr výplně: metoda `setPaint`
- Příklad: **--- DEMO**

```
g2.setPaint(Color.RED);  
g2.fill(...)
```

```
GradientPaint gp = new GradientPaint(  
    0, 0, Color.WHITE,  
    0, 20, Color.BLACK, true);  
g2.setPaint(gp);  
g2.fillRect(20, 260, 300, 40);
```

ŠTĚTEC (BRUSH)

- Často zaměňované za výplň nebo pero
- Určeno pro efekty
- Grafické knihovny nativně většinou nepodporují



GRAFICKÁ PRIMITIVA

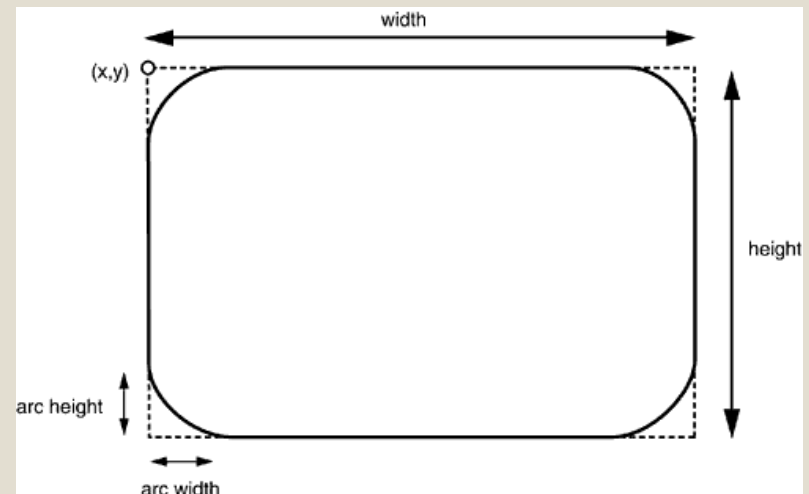
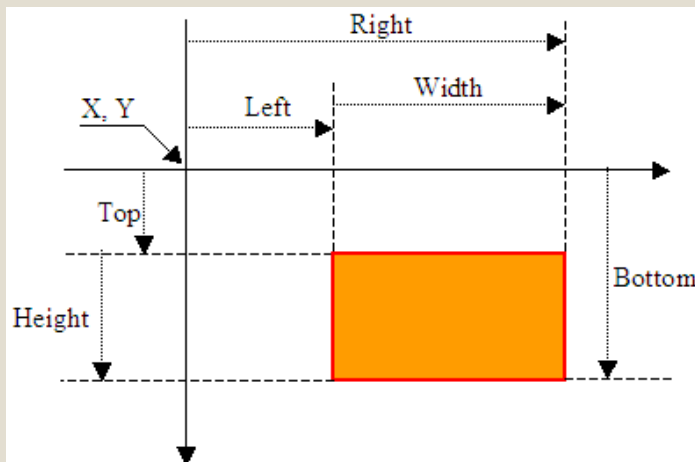
- Čára (úsečka)
- Kružnice, Kruh
- Elipsa
- Oblouk
 - Kruhový, eliptický
- Obdélník
 - Ostré nebo zakulacené rohy
- Polygon
- Křivka
- Bitmapový obraz
- ...

ČÁRA (ÚSEČKA)

- Určena: počáteční (x_1, y_1) a koncový bod (x_2, y_2)
 - Jedna metoda: `line(x1, y1, x2, y2)`
 - Dvě metody: `moveTo(x1, y1)` a `lineTo(x2, y2)`
- Základem vykreslování ostatních primitiv
 - Ostatní primitiva aproximována lomenou čarou
 - POZOR: lomená čára \neq skupina úseček
 - Viz způsob napojování
- Rasterizace úsečky, tj. převod na pixely
 - DDA algoritmus - KIV/ZPG
 - Nebo převod na oblast a výplň

OBDÉLNÍK

- Strany rovnoběžné s osami x, y
- Různé způsoby definování
 - Nejčastěji poloha levého horního rohu + šířka a výška
- Může mít zakulacené rohy
 - Nejen kvůli estetice, ale také prakticky (obrábění)
 - Zakulacení určeno poloměrem nebo velikostí poloos

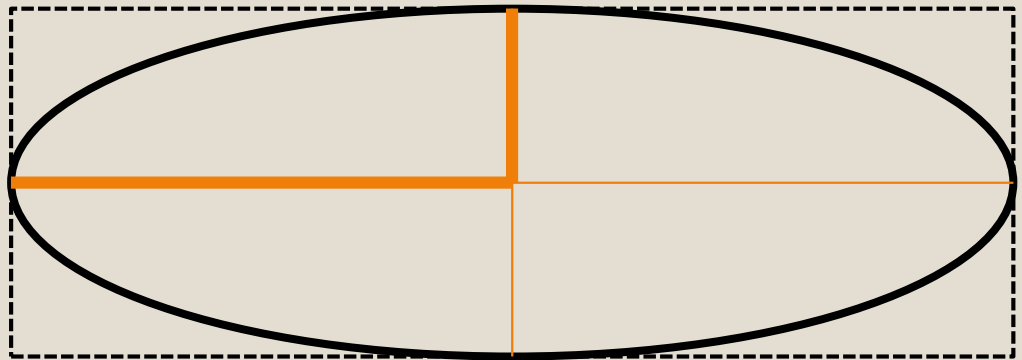


OBDÉLNÍK

- Obdélník s ostrými rohy a zaoblenými rohy odlišen často v grafických knihovnách názvem metody
 - Např. Java AWT: `drawRect` a `drawRoundedRect`

ELIPSA

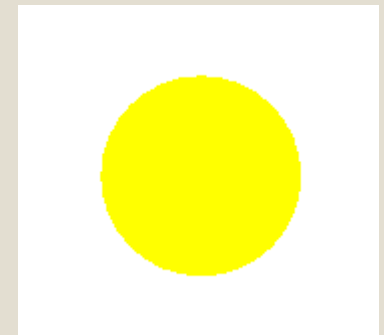
- Určena: umístění (x, y) a svou velikostí v osách x, y
- Možnosti umístění
 - Levý horní roh nejmenšího opsaného obdélníku => velikost = velikost stran toho obdélníku
 - Střed elipsy => velikost poloos
- Metoda: typicky přípona „ellipse“ nebo „oval“
 - Např. Java: drawOval



KRUŽNICE, KRUH

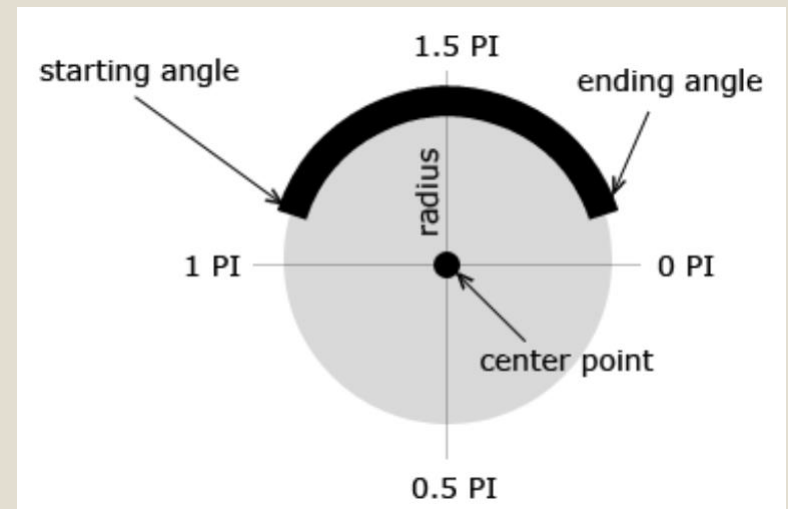
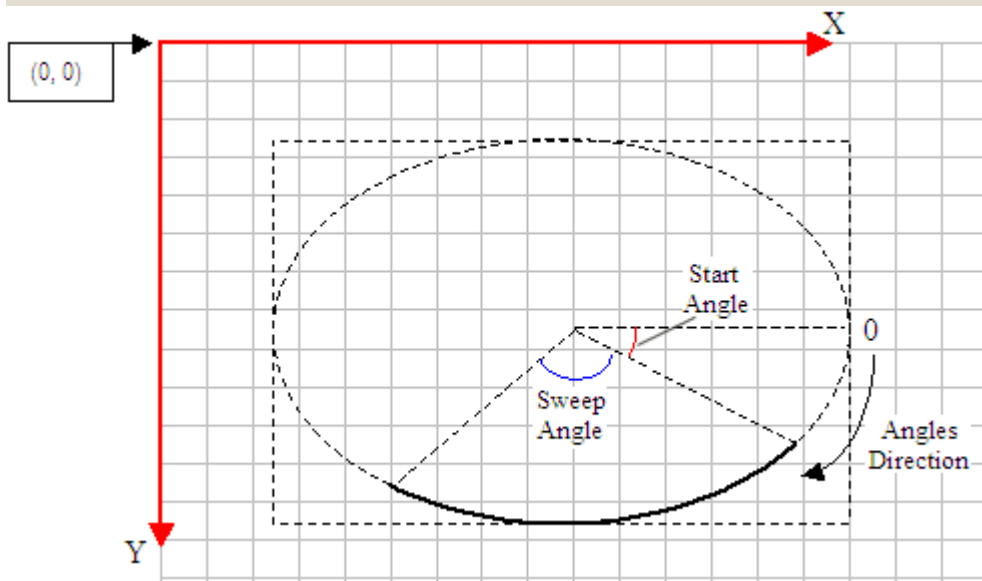
- Grafické knihovny často nemají pro kružnici (kruh) přímou podporu, protože kružnice je jen speciální případ elipsy se stejnou velikostí poloos
- Příklad:

```
g2.setColor(Color.YELLOW);  
g2.fillOval(cx - SMILE_RADIUS, cy - SMILE_RADIUS,  
            2*SMILE_RADIUS, 2*SMILE_RADIUS);
```



OBLOUK (ARC), VÝSEČ

- Část elipsy (resp. kružnice/kruhu) definována:
 - Stejně jako elipsa (resp. kružnice/kruh), ze které vychází
 - Počátečním úhlem („start angle“)
 - Koncovým úhlem („ending angle“) nebo úhlovou velikostí výseče („sweep angle“ nebo také „arc angle“)

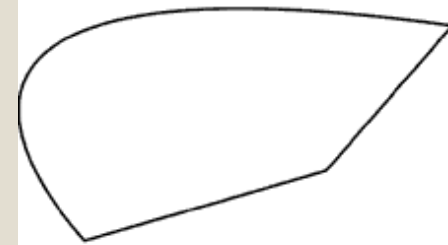


OBLOUK (ARC), VÝSEČ

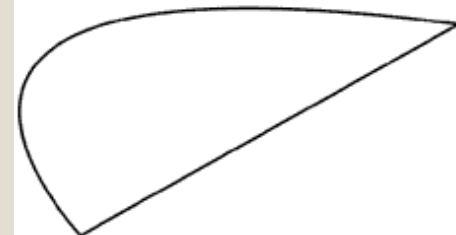
- Pozor: některé knihovny mají úhly po směru hodinových ručiček (většina), jiné proti směru, některé umožňují toto specifikovat
- Oblouk může být otevřený, nebo uzavřený, ať již přímo („chord“) nebo přes („střed“)
 - Knihovny mohou poskytovat různé metody pro rozlišení tohoto
 - Např. GDI+ *DrawArc* a *DrawPie*
- Java AWT: *drawArc*



Arc2D.OPEN



Arc2D.PIE



Arc2D.CHORD

KONEC

- Pokračování příště ...



ZÁKLADNÍ 2D VEKTOROVÁ GRAFIKA

Další primitiva

Bézierova křivka

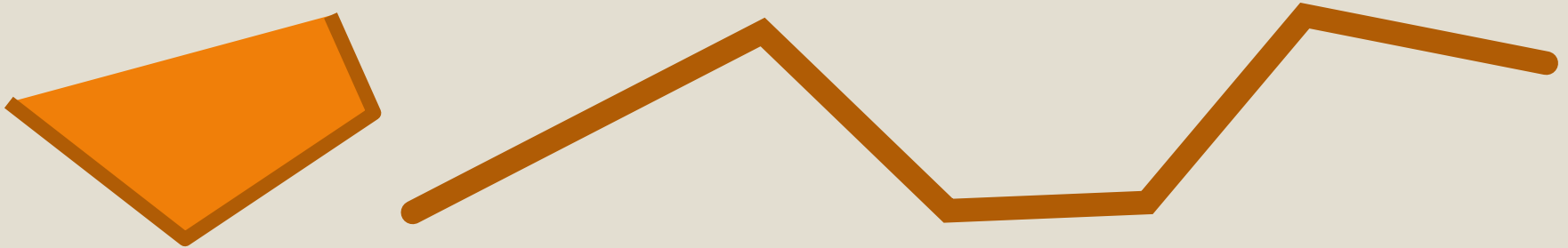
Práce s písmem

Vektorový obraz
na tiskárně

LOMENÁ ČÁRA

- Lomená čára = sekvence navazujících úseček
- V místě napojení vykresleno navázání dle zvoleného způsobu napojení pro aktuální pero
- Definováno posloupností vrcholů (uzlů)

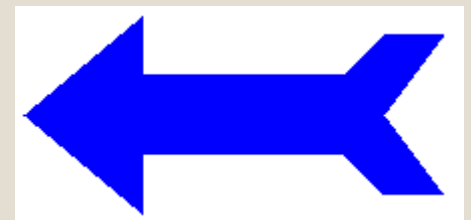
```
int x[] = {5,200,305,420,505,645};  
int y[] = {115,15,115,110,10,35};  
g2.drawPolyline (x, y, x.length);
```



POLYGON

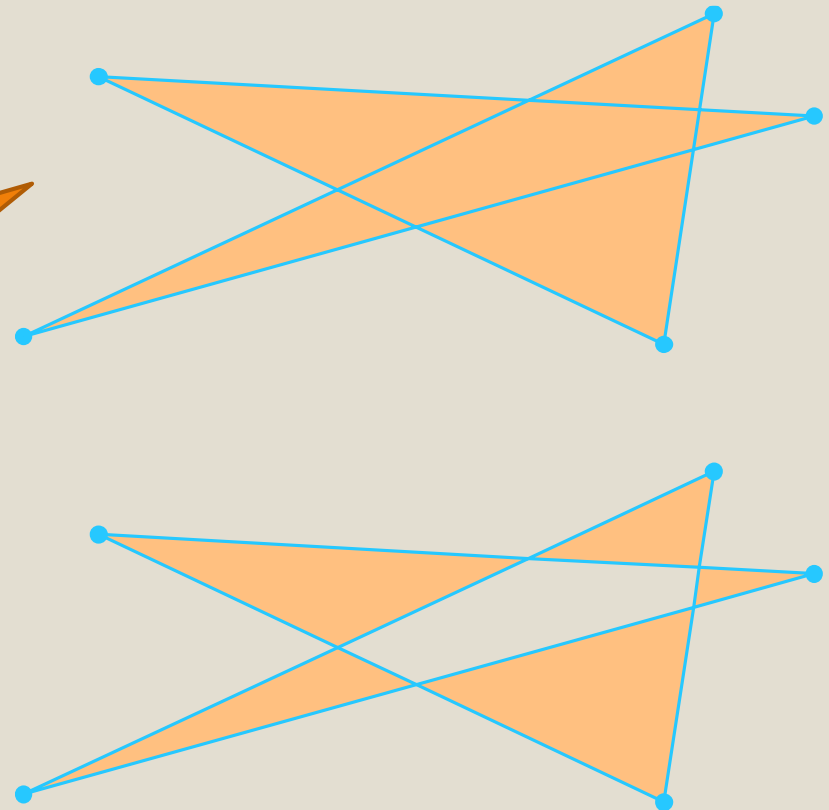
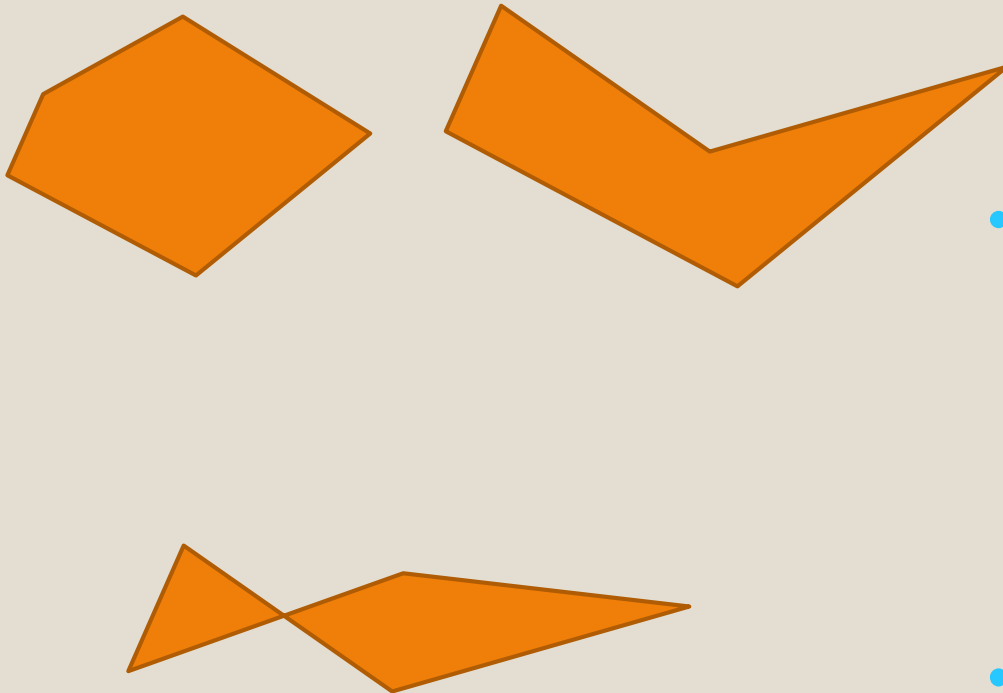
- Je to **uzavřená** lomená čára, jejíž poslední vrchol je totožný s prvním zadáním
 - Některé knihovny zadání tohoto totožného vrcholu vyžadují, jiné polygon automaticky uzavřou
 - Pozor: nezaměňovat za otevřenou lomenou čarou, jejíž poslední vrchol je totožný s prvním!
- Příklad:

```
int x[] = {10,70,70,170,190,220,  
           190,220,190,170,70,70};  
int y[] = {60,10,40,40,20,20,  
           60,100,100,80,80,110};  
g2.fillPolygon(x, y, x.length);
```



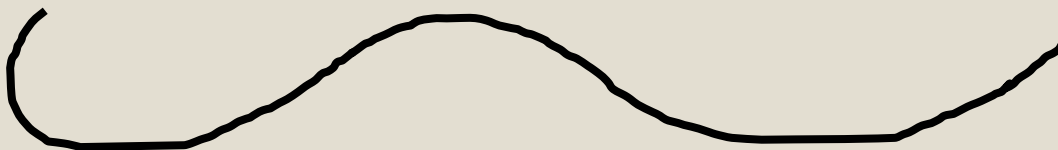
POLYGON

- Konvexní vs. konkávní polygon
- Protínající se polygon



ROVINNÁ KŘIVKA

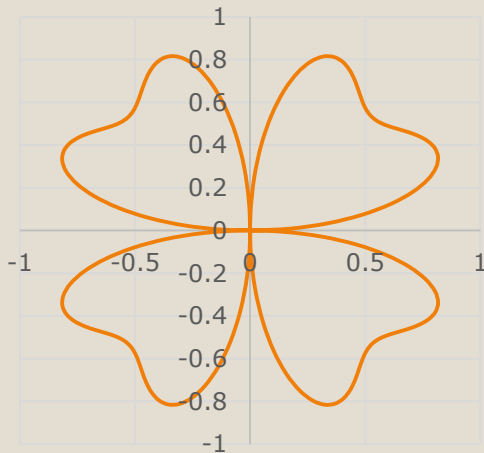
- Vektorová funkce $F(x, y)$, $x = f(t)$ a $y = g(t)$, taková, že funkce $f(t)$, $g(t)$ jsou pro proměnnou $t \in \langle a, b \rangle$ spojité
- Několik málo příkladů:
 - Úsečka AB: $x = A_x + (B_x - A_x) \cdot t$, $y = A_y + (B_y - A_y) \cdot t$, $t \in \langle 0, 1 \rangle$
 - Kružnice se středem v počátku a poloměrem r :
 $x = r \cdot \cos(t)$, $y = r \cdot \sin(t)$, $t \in \langle 0, 2\pi \rangle$
 - Sinusovka: $x = t$, $y = \sin(t)$, $t \in (-\infty, \infty)$
 - Obecná křivka:



ROVINNÁ KŘIVKA

■ Příklad

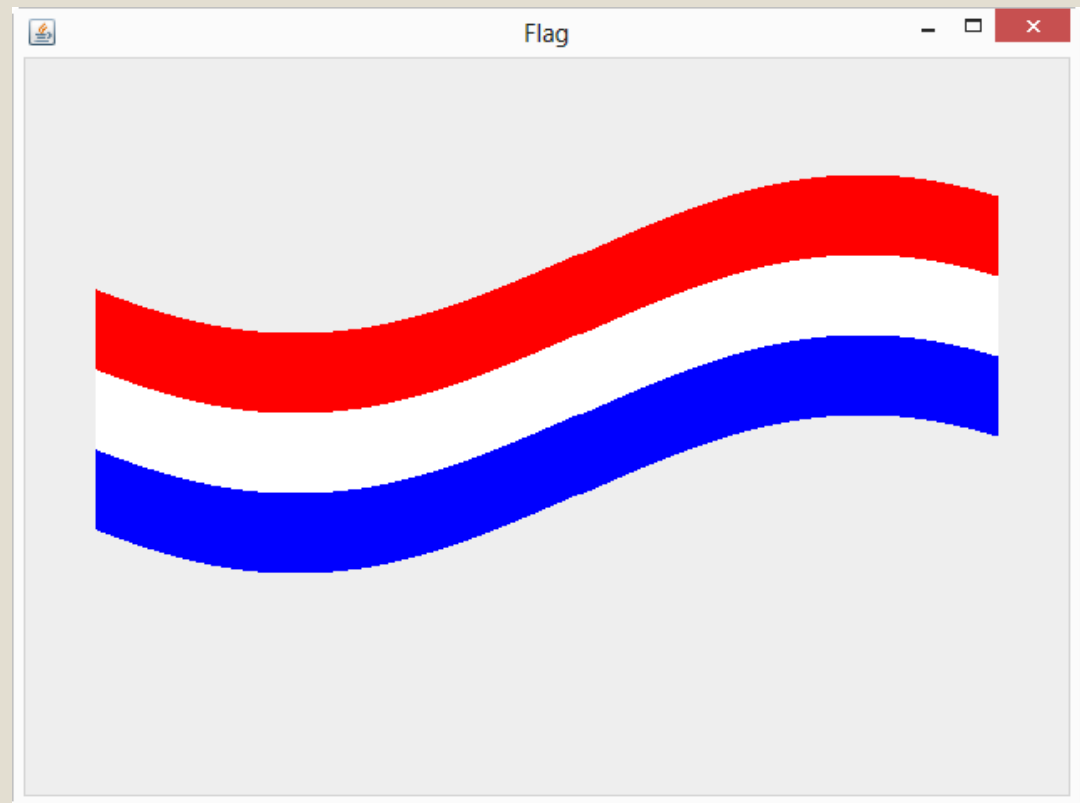
- Čtyřlístek
- Flag.java
- **BONUS**



$$x = r \cdot \cos t$$

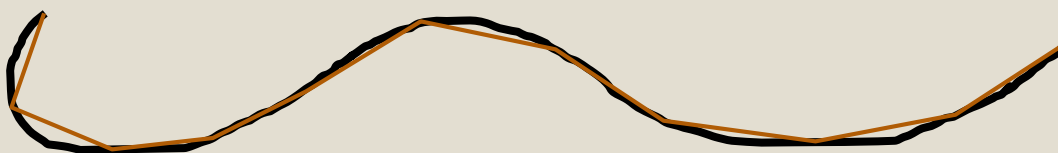
$$y = r \cdot \sin t$$

$$r = \sin 2t + \frac{1}{4} \cdot \sin 6t$$



ROVINNÁ KŘIVKA

- Rovinnou křivku lze nahradit lomenou čarou
- Uniformní rozklad
 - Vzdálenost mezi sousedními vrcholy (uzly) konstantní
 - Nejjednodušší: vzdálenost v parametru t
 - Někdy též Euklidovská vzdálenost ve 2D nebo častěji v projekci uzlů na osu (vzorkování)
 - Jsou-li uzly ve vzdálenosti 1 px, není možné výsledek na daném zařízení odlišit od skutečné křivky



ROVINNÁ KŘIVKA

- Rovinnou křivku lze nahradit lomenou čarou
- Neuniformní rozklad
 - Vzdálenost různá, ale snaha minimalizovat chybu
 - Je-li chyba menší 0.5 pixelu, není možné výsledek na daném zařízení odlišit od skutečné křivky

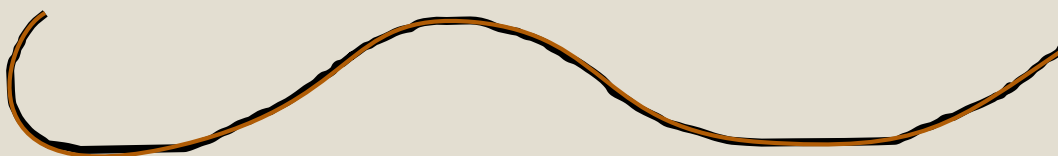


ROVINNÁ KŘIVKA

- Uniformní rozklad
 - Jednodušší, ale produkuje zbytečně mnoho uzlů
- Neuniformní rozklad
 - Složitější, uzlů méně (ale také hodně)
- Problém: algoritmy v zařízeních a grafických knihovnách podporují typicky jen omezený předem neznámý počet uzlů
 - Řádově tisíce
- Řešení: rozdělit lomenou čáru na více částí
 - Nelze univerzálně

ROVINNÁ KŘIVKA

- Rovinnou křivku lze nahradit po částech matematicky definovanou parametrickou elementární rovinnou křivkou
 - Typicky se jedná o kubickou funkci:
$$C(t) = a \cdot t^3 + b \cdot t^2 + c \cdot t + d, t \in \langle 0, 1 \rangle$$
 - Konstanty a, b, c, d určeny typem elementární křivky
 - Nejčastěji Bezier, Coons, NURBS, ...



ROVINNÁ KŘIVKA

- Výhoda přístupu:
 - Vektorové vyjádření
 - Lze měnit měřítko obrazu bez ztráty kvality
 - Převod na lomenou čáru proveden automaticky
 - Provádí se až na samý závěr => vysoká kvalita

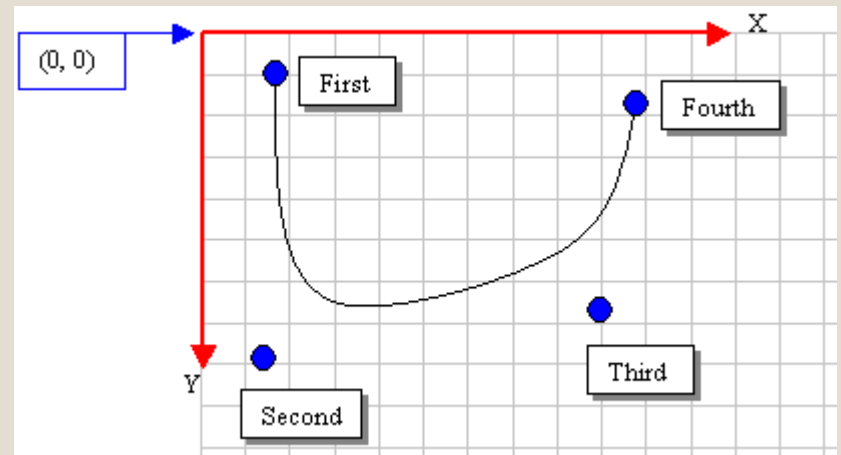
BEZIEROVA KUBIKA

- Definována čtyřmi parametry:
 - Počáteční a koncový bod, kterým křivka prochází
 - Dva pomocné body, které určují tvar křivky

```
CubicCurve2D cubcurve = new CubicCurve2D.Float(
```

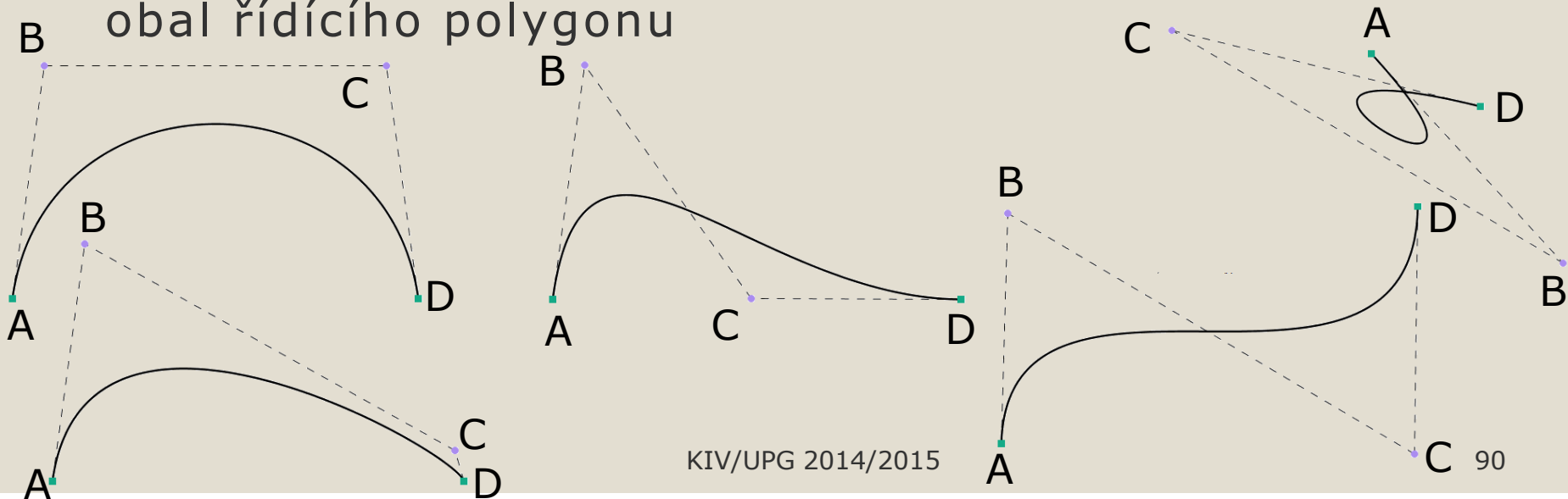
```
    1.7f, 1.0f, //First  
    1.2f, 8.0f, //Second  
    9.0f, 6.8f, //Third  
    9.8f, 1.8f); //Fourth
```

```
g2.draw(cubcurve);
```

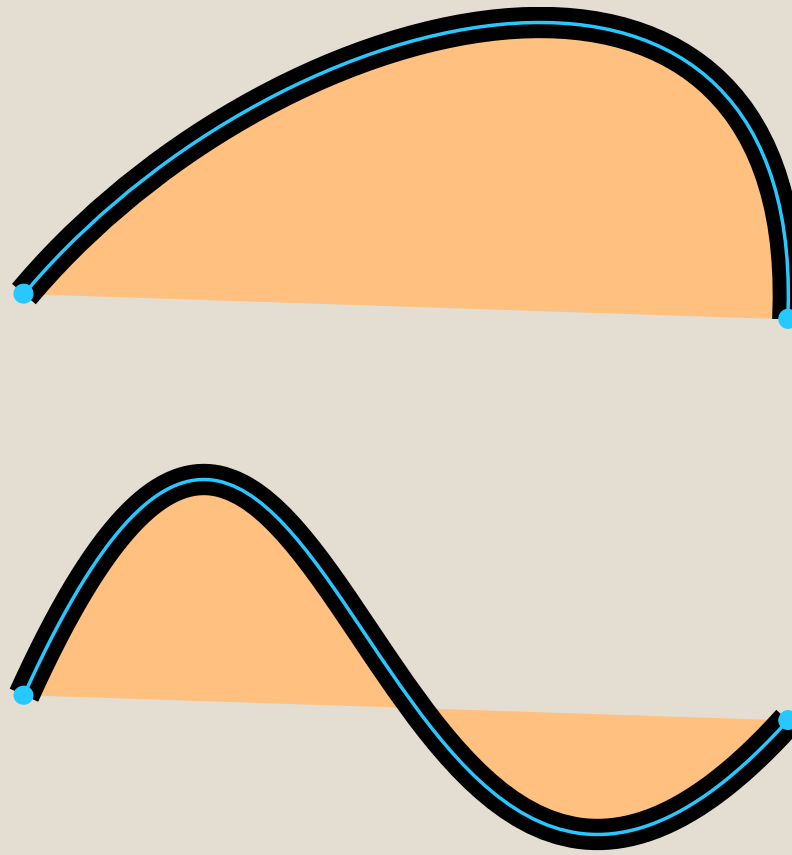


BEZIEROVA KUBIKA

- Pořadí bodů klíčové pro tvar
- Body tvoří tzv. řídící polygon ABCD
- Křivka prochází body A, D
- Křivka se přimyká k úsečkám AB a CD
 - Čím je úsečka delší, tím déle křivka „kopíruje“ úsečku
- Křivka nemá zběsilé tvary, protože neopustí konvexní obal řídícího polygonu



BEZIEROVA KUBIKA



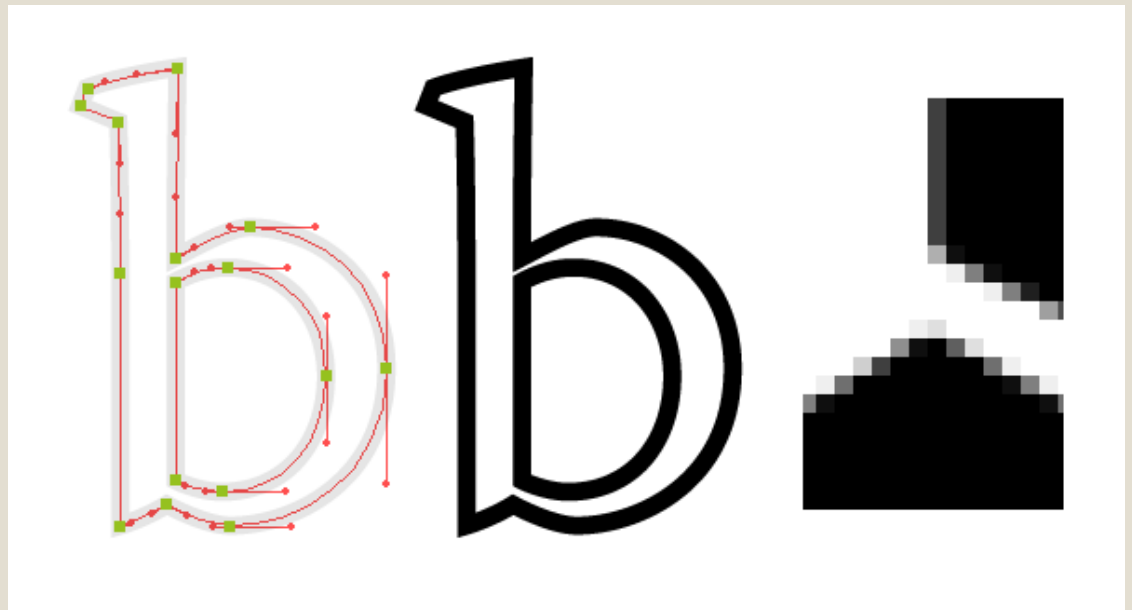
BEZIEROVA KŘIVKA

- Možnosti napojení Bezierových křivek:
 - Spojité (C0) – bod A navazující křivky musí být totožný s bodem D první (předchozí) křivky
 - Hladké (G1) – spojitě + bod B navazující křivky musí ležet na přímce určené úsečkou CD první křivky
 - Bod nesmí splynout s body CD první křivky
 - Hladké (C1) – G1 + vzdálenost AB navazující křivky je rovna vzdálenosti CD první křivky
 - Poloha bodů A,B navazující křivky je tedy pevně dána

BEZIEROVA KŘIVKA

■ Užití:

- Interpolace klíčových snímků v animacích
 - např. v Adobe Flash
- Většina dnes používaných fontů písma



PRÁCE S PÍSMEM

- Práce s písmem sestává ze tří úkonů:
 1. Vykreslování jednotlivých znaků
 - volba fontu, problematika kódování znaků, ...
 2. Řazení znaků do slova
 - problematika mezer mezi písmeny, psaní zleva doprava či zprava doleva apod.
 3. Řazení slov do řádků
 - problematika zalamování a zarovnání odstavce, ...

PRÁCE S PÍSMEM

- Pro práci s písmem je klíčová volba tzv. fontu písma
- Pojem font dnes používán ve třech významech, které je nutné od sebe odlišovat dle kontextu:
 - Původní definice: font = úplná sada znaků jednotného stylu a jedné velikosti, tj. obsahuje popisy tvarů „všech“ znaků
 - DTP definice: font = úplná sada znaků jednotného stylu, ale velikost nemusí být součástí (upraví se automaticky)
 - Např. Arial Normální (arial.ttf), System (vgasyse.fon)

PRÁCE S PÍSMEM

- Přenesená definice: font = úplná sada znaků sdílející společné prvky jednoho designu, tj. různý tvar pro stejné písmeno, ale „stejným rukopisem“ (např. jiný náklon)
 - Synonymum k pojmu „rodina písma“ (font family, typeface)
 - Např. Arial = Arial Černé (arialb.ttf), Arial Kurzíva (ariali.ttf), Arial Normální (arial.ttf), Arial Tučné Kurzíva (arialbi.ttf), nebo Arial Tučné (arialbd.ttf)
 - Arial Polozhuštěné = Arial Polozhuštěné (ARIALN.ttf), Arial Polozhuštěné Kurzíva (ARIALNI.ttf), Arial Polozhuštěné Tučné (ARIALNB.ttf), Arial Polozhuštěné Tučné Kurzíva (ARIALNBI.ttf),

PRÁCE S PÍSMEM

Arial

Select font to view more information.

Arial

[View font information](#)

Общий Ζαφεΐρι ÀÉÎÕÜ àéîõü

Arial Black

[View font information](#)

Five Wax Quacking Zephyrs

Arial Bold

[View font information](#)

Общий Ζαφεΐρι ÀÉÎÕÜ àéîõü

Arial Bold Italic

[View font information](#)

Общий Ζαφεΐρι ÀÉÎÕÜ àéîõü

Arial Italic

[View font information](#)

Общий Ζαφεΐρι ÀÉÎÕÜ àéîõü

PRÁCE S PÍSMEM

- Rodina písma je buď
 - Bezpatková (Sans Serif) nebo patková (Serif)
 - Např. Arial, Verdana vs. Times New Roman, Lucida Bright
- AaBbCc AaBbCc
- Proporcionální nebo neproporcionální
 - Různá / stejná šířka jednotlivých znaků
 - Např. Arial vs. Courier



PRÁCE S PÍSMEM

- Grafické knihovny typicky poskytují tři generické rodiny písem pro:
 - Bezpatkové písmo (Sans Serif)
 - Patkové písmo (Serif)
 - Neproporcionální písmo (Monospaced)
- Generická rodina se mapuje na skutečnou dle výchozího nastavení operačního systému

VÝBĚR FONTU

- Výběr fontu (3. definice) dán zejména zvyklostmi v oblasti, pro kterou je výstup určen
 - Např. matematické popisky patkovým písmem, kartografické bezpatkovým, strojařské výkresy verzálkami (velká písmena), ...
 - Neexperimentovat, nejsem-li odporník na DTP
- Výběr ovlivněn důležitými parametry fontu:
 - Typ počítačového formátu písma
 - Tvar glyfů (kresba písma)
 - Řez (kresebná varianta, někdy též označováno jako styl)
 - Množství podporovaných znaků
 - Dostupnost
 - Copyright

O0, lI1, vV

O0, lI1, vV

VÝBĚR FONTU

- Formát písma:
 - Rastrové písmo
 - Type 1
 - TrueType
 - OpenType
- Rastrové písmo
 - Historicky nejstarší
 - Pro každý znak bitmapa
 - Fixní velikost písma
 - Předdefinováno několik málo velikostí (např. 6)
 - Nedostanu to, co chci!
 - Přípona .fon

MS Sans Serif, 12

MS Sans Serif, 14

MS Sans Serif, 16

MS Sans Serif, 18

MS Sans Serif, 20

MS Sans Serif, 22

MS Sans Serif, 24

MS Sans Serif, 26

MS Sans Serif, 28

MS Sans Serif, 30

MS Sans Serif, 32

MS Sans Serif, 34

MS Sans Serif, 36

MS Sans Serif, 10

MS Sans Serif, 9

MS Sans Serif, 8

MS Sans Serif, 6

MS Sans Serif, 4

VÝBĚR FONTU

- Příklady rastrových písem:

Courier
Fixed Sys
MS Sans Serif
MS Serif
Small Fonts
System
Terminal

VÝBĚR FONTU

- Type1 písmo
 - Zavedeno společností Adobe v roce 1984
 - Určeno pro profesionální DTP
 - Součástí PostScript (PS)
 - Používá se např. pro komunikaci s laserovými tiskárnami
 - Glyfy definovány jako Bezierovy kvadratické křivky
 - **BONUS**
 - Přípona (typicky) .pfb (glyfy) + .pfm (metrika)
 - Některé grafické knihovny tyto písma nepodporují
 - Např. WPF, GDI+ (ale GDI písmo podporuje)
 - Při použití se zkonvertuje automaticky na něco jiného

VÝBĚR FONTU

■ Příklady Type1 píssem:

- Courier (PS)
- Helvetica
- Symbol: Σψμβολ
- Times
- ITC Zapf Dingbats: ITX Ζαπφ Δινγβατσ
- ITC Avant Garde Gothic
- ITC Bookman (Light)
- ITC Zapf Chancery
- New Century Schoolbook
- Palatino

VÝBĚR FONTU

■ TrueType písmo

- Zavedeno společnostmi Apple (a Microsoft) v roce 1991 z licenčních důvodů formátu Type1
- Glyfy definovány jako Bezierovy kvadratické křivky
- Masivní rozšíření na Mac OS a Windows platformách
- Unicode => písmo může obsahovat tisíce znaků
- Přípona .ttf nebo .tte

■ Příklady TrueType (TT) písem:

- Arial
- Courier New
- Times New Roman

VÝBĚR FONTU

■ OpenType písmo

- Zavedeno v roce 1997 (Microsoft + Apple)
- ISO standardizace v roce 2007
- Následovník TrueType
- Glyfy mohou být definovány jako TrueType nebo Bezierovy kubické křivky (Type 2)
- Rozšířené typografické možnosti (např. ligatury)
- Přípona .ttf nebo .otf
- Většina původních TT fontů převedena do OpenType (O)



VÝBĚR FONTU

■ Příklady OpenType (TT) píssem:

- Calibri
- Verdana
- *Monotype Corsiva*
- Microsoft Sans Serif (MS Reference Sans Serif)
- Arial
- Courier New
- Times New Roman
- ...
- <http://www.microsoft.com/typography/fonts/product.aspx?PID=164>

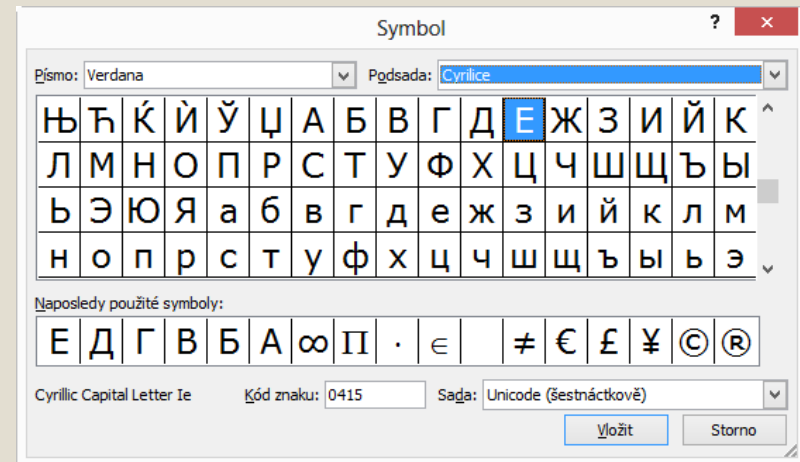
VÝBĚR FONTU

- Řez písma (styl):

- Normální
- **Tučné (Bold)**
- „Kurzíva“ (*Italic*)
- ***Tučné italikou***
- ...

- ## ■ Podporované znaky:

- Diakritika – Příšerně žluťoučký kůň úpěl ďábelské ódy
- Azbuka (Cyrilice) – АБВГДЕ ...
- Alfabeto – αβγδεζ ...
- Matematické symboly - $\pi \Sigma \sqrt{\infty} \neq \dots$
- Grafické symboly - — ∟ ∟ ⊥ ◇ ● ...
- ...



VÝBĚR FONTU

■ Dostupnost fontů

- Vybraný font nemusí být dostupný na všech zařízeních
- Např. „Times“ je dostupný prakticky všude, ale speciální písmo pro strojařské výkresy typicky není

■ Copyright

- Naprostá většina písem je komerčních a nelze je volně distribuovat, tj. „přibalit“ k aplikaci

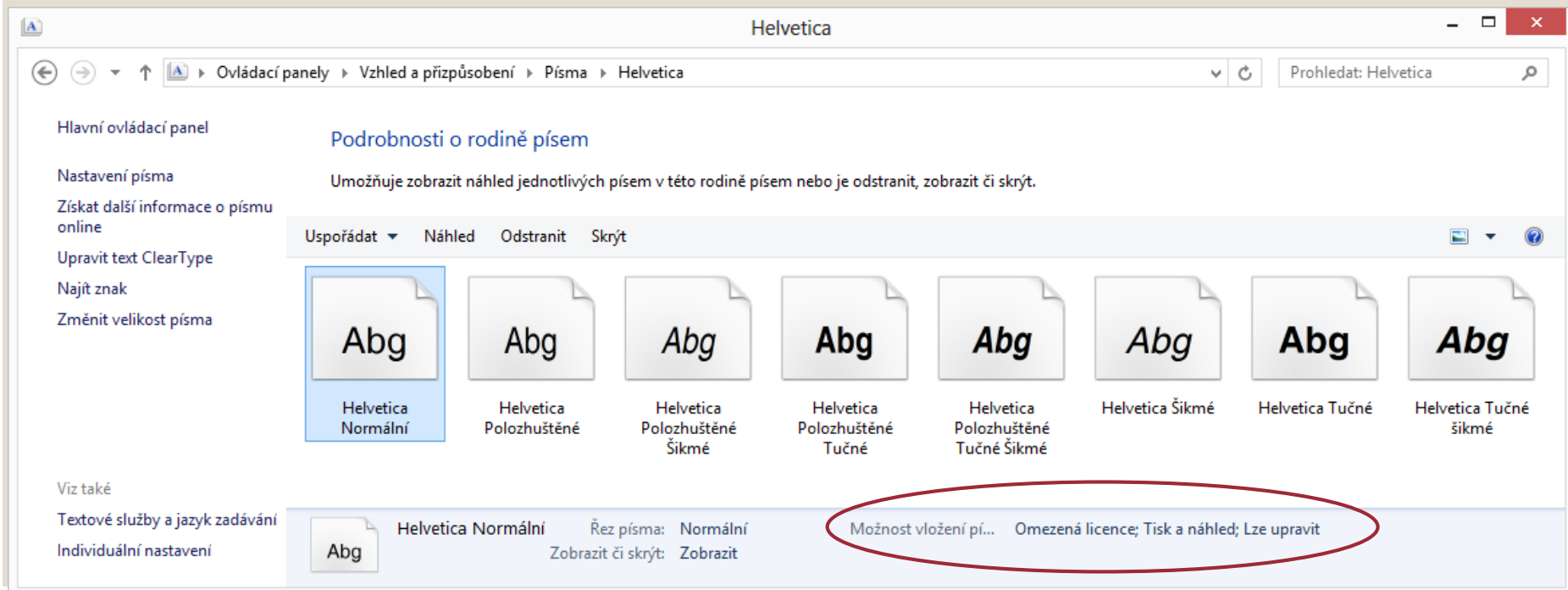
Q What can I do with the fonts supplied with Microsoft products?

A The fonts are governed by the same restrictions as the products they are supplied with. You are not allowed to copy, redistribute or reverse engineer the font files. For full details see the license agreement supplied with the product.

Some fonts may be embedded within document files. Embedding allows fonts to travel with documents. Embedded fonts can only be used to print, preview and in some cases edit the document in which they are embedded. Please see the [Embedding TrueType](#) page and the [TrueType font embedding FAQ](#) for details.

VÝBĚR FONTU

- Některé fonty nainstalované v rámci OS Windows nepatří společnosti Microsoft
- Microsoft poskytuje licenci na své vlastní fonty prostřednictvím společnosti Monotype Imaging
 - <http://catalog.monotypeimaging.com/>



PRÁCE S PÍSMEM V JAVĚ

- Písmo = třída `java.awt.Font`
 - Představuje font dle 1. definice, tj. jedná se již o konkrétní řez (styl) a konkrétní velikost
- Požadavky na písmo při konstrukci
 - Konstruktor přetížený
 - Minimální požadavky: rodina písma, velikost (v pt) a požadovaný řez
 - Volit raději větší velikost, ideálně dát uživateli volbu
- Výběr písma pro text: metoda `setFont`

PRÁCE S PÍSMEM V JAVĚ

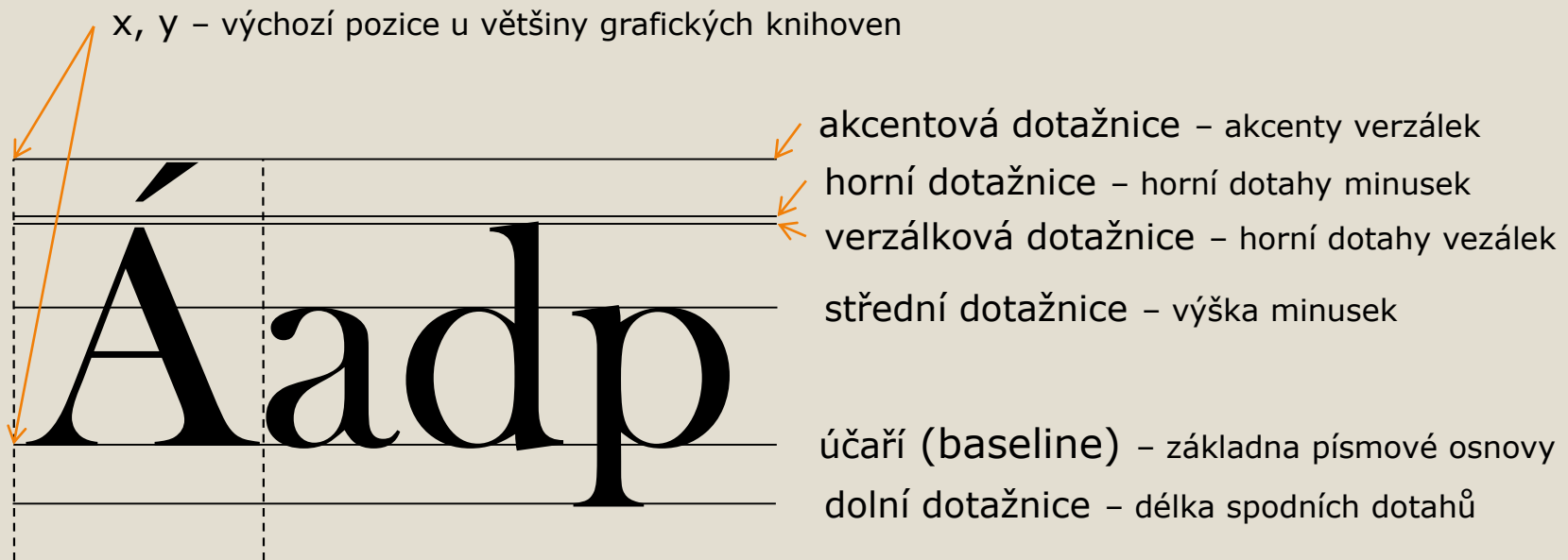
```
Font fnt = new Font("Calibri",  
                    Font.BOLD | Font.ITALIC, 12);
```

```
g2.setFont(fnt);  
g2.drawString("Hello World", x, y);
```

```
//x, y = pozice (v pixelech),  
//kam se má text zobrazit
```


PÍSMO ZBLÍZKA

- písmová osnova = soustava vodorovných prímek (dotažnic), která určuje výškové proporce písma



PÍSMO ZBLÍZKA

- malá písmena (minuskule)
- velká písmena (verzálky)
- kapitálky
- číslice
- interpunkční znaménka
 - verzálkové × minuskulní
- závorky
- samostatné akcenty
- speciální znaky

a b c
A B C
A B C
0 1 2 3 4 5 6 7 8 9
. , ; : ? !
([{ }])
˘ ˙ ˚
% # & ~ ½ (

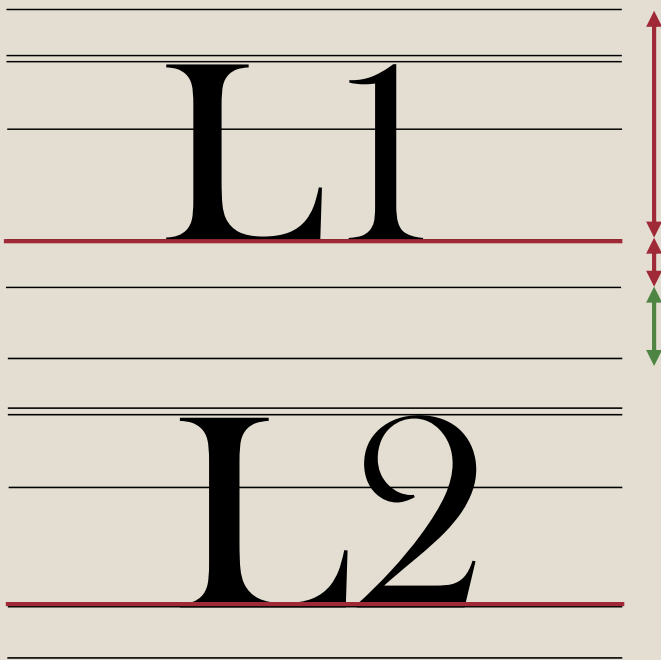
a b c
A B C
A B C
0 1 2 3 4 5 6 7 8 9
. , ; : ? !
([{ }])
˘ ˙ ˚
% # & ~ ½ @ \$



METRIKA PÍSMÁ

- Grafické knihovny poskytují metody pro měření textu, který je třeba zobrazit
 - Metriky se liší knihovna od knihovny

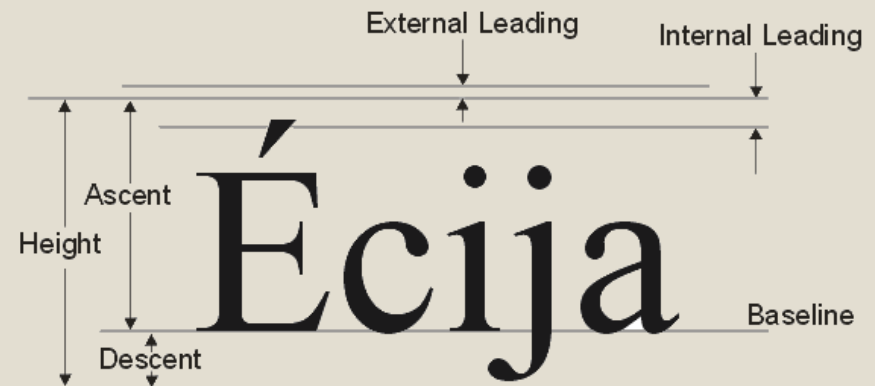
Line spacing (meziřádková mezera)



ascent

descent

leading (meziřádkový proklad)



METRIKA PÍSMO V JAVĚ

- Třída `FontMetrics` nebo `LineMetrics`
 - Instance poskytnuta grafickým kontextem
 - Poskytuje mj., kolik px zabere určitý text
 - Nemusí být přesné!

```
FontMetrics metrics = graphics.getFontMetrics(font);  
int height = metrics.getHeight();  
int advance = metrics.stringWidth(text);
```



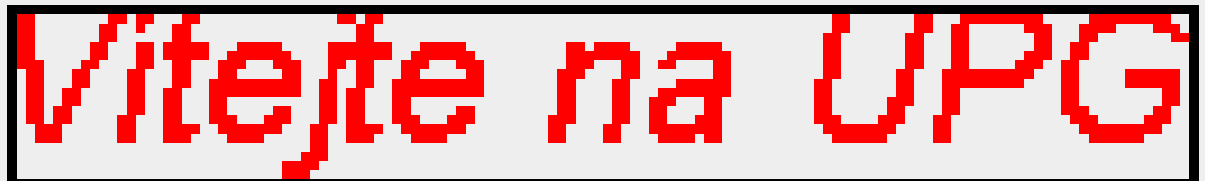
METRIKA PÍSMO V JAVĚ

■ Třída TextLayout

- Poskytuje mj. obdélník, do kterého se text vejde
- Umožňuje vykreslení textu

```
Point2D loc = new Point2D.Float(15, g2.getFontMetrics().getAscent());  
TextLayout layout = new TextLayout(text, g2.getFont(),  
                                g2.getFontRenderContext());  
layout.draw(g2, x, y); //vykreslení textu
```

```
Rectangle2D bounds = layout.getPixelBounds(  
    g2.getFontRenderContext(), x, y);  
g2.setColor(Color.BLACK);  
g2.draw(bounds);
```

A pixelated red text "Vítejte na UPG" is displayed inside a black rectangular border. The text is rendered in a pixelated font style, with each character composed of small red squares. The background of the text area is white, and the entire text is enclosed within a thick black rectangular frame.

PROSTRKÁNÍ - KERNING

- Prostrkání = proces, který zmenšuje mezeru mezi písmeny v textu tam, kde toto je možné
- Může být žádoucí nejen při zarovnávání do bloku, ale rovněž při běžném zobrazování
- Font musí prostrkání podporovat

VAL

VAL

AV Wa
No kerning

AV Wa
Kerning applied

PROSTRKÁNÍ V JAVĚ

```
Map<TextAttribute, Object> map =  
    new Hashtable<TextAttribute, Object>();  
map.put(TextAttribute.KERNING,  
    TextAttribute.KERNING_ON);
```

```
Font newFont = oldFont.deriveFont(map);  
g2.setFont(font);  
g2.drawString(text, x, y);
```

DALŠÍ TEXTOVÉ EFEKTY V JAVĚ

- Třída `TextAttribute` umožňuje analogickým způsobem přidat další efekty jako:
 - Podtržení textu
 - Přeskrtnutí textu
 - Horní a dolní index
 - ...

VÍCEŘÁDKOVÝ TEXT

- Dlouhý text vhodné zalomit na více řádek
 - Text rozdělen v místě mezery, čárky, středníku, pomlčky, otevírací závorky, ...
 - Alternativně lze text rozdělit uvnitř slova
 - Složitější, vyžaduje slovník
- Neproporcionální písmo
 - Lze zalamovat tak, aby počet znaků na řádce nepřesáhl předem daný limit (typicky 60-80)
- Proporcionální písmo
 - Nutno zalamovat na předem danou fyzickou šířku
 - Např. šířka oblasti, do které se text má vejít

VÍCEŘÁDKOVÝ TEXT

■ Algoritmus:

- Načítej slova z textu do pomocného řetězce a měř šířku, kterou by tento řetězec po zobrazení zabral
- Po překročení meze se vrať k nejbližšímu předchozímu místu, kde lze provést rozdělení
 - Pokud místo neexistuje, najdi nejbližší další
- Zobraz pomocný řetězec

■ Některé grafické knihovny poskytují jednoduchou implementaci

- Např. Java AWT má třídu `LineBreakMeasurer`

VÍCEŘÁDKOVÝ TEXT

```
LineBreakMeasurer brk = new LineBreakMeasurer(  
    text, g2.getFontRenderContext());  
  
int curY = y;  
while (brk.getPosition() < text.getEndIndex()) {  
    TextLayout layout2 = brk.nextLayout(wrpWidth);  
  
    curY += (layout2.getAscent());  
    layout2.draw(g2, x, curY);  
    curY += layout2.getDescent() +  
            layout2.getLeading();  
}
```

VÍCEŘÁDKOVÝ TEXT

- Čím delší řádek, tím větší mezera mezi řádky
 - Usnadňuje čtenáři hledání navazujícího textu
 - Pro 60-80 znaků na řádek obvykle vzdálenost mezi sousedními účarámi (baseline) 120% velikost písma
 - Např. pro 12 pt standardně 2.4 pt mezera
 - Velikost mezery pro řádkování: leading

ZAROVNÁNÍ TEXTU

■ Doleva

- Levé okraje řádků pod sebou
- Nejjednodušší
- Vhodné zejména pro
 - Krátké texty
 - Úzké sloupce
 - Špatně dělitelný text (URL, chemické vzorce)



■ Doprava

- Právě okraje řádků pod sebou
- Vhodné zejména pro
 - Velmi krátké texty (např. popisky, názvy kapitol)



ZAROVNÁNÍ TEXTU

■ Na osu (např. na střed)

- Osa nemusí být uprostřed
- Vhodné zejména pro
 - Velmi krátké texty (např. popisky, názvy kapitol)



■ Do bloku

- Levé i pravé okraje řádků pod sebou
 - Poslední řádek může být zarovnán také doleva, na osu nebo doprava
- Technicky nejsložitější
- Vhodné zejména pro
 - Delší (mnoha odstavcové) texty



ZAROVNÁNÍ TEXTU DO BLOKU

- Základní princip: zvětšit (resp. zmenšit) mezislovní mezery
 - Velikost mezery musí být z předem daného rozsahu
 - Mezery musí být na správném místě
- Časté typografické chyby
 - Neúnosně rozšířené nebo zúžené mezislovní mezery
 - Po sobě jdoucí řádky mají mezislovní mezery diametrálně odlišné
 - Řešení:
 - Úprava šířky sloupce, změna textu
 - Dělení slov
 - „Kerning“ (mezerky mezi písmeny)

Cerná je „barva“, kterou oko vnímá v případě, že z daného směru nepřichází žádné světlo. Tento nedostatek světla může být dán jak neexistencí žádného zdroje, tak tím, že příslušný materiál pohlcuje světlo všech barev, namísto aby některé barvy odrazil. Černý pigment může být také tvořen směsí několika pigmentů, z nichž každý pohlcuje některé barvy, čímž v souhrnu pohlcuje tak velkou část viditelného spektra, že se dá označit za černý (viz CMYK).

ZAROVNÁNÍ TEXTU DO BLOKU

- Nejjednodušší (greedy) algoritmus:
 1. Načítej slova z textu do pomocného řetězce, dokud není překročena povolená mez
 2. Pokud lze stáhnout šířku mezislovních mezer, aby to sedlo, udělej tak a pokračuj dalším řádkem
 3. Vrať poslední načtené slovo a zkontroluj, zda lze roztáhnout šířku mezislovních mezer, aby to sedlo.
 4. Pokud to lze, udělej tak a pokračuj dalším řádkem
 5. Nalezni, kde lze poslední načtené slovo (nebo jeho část) rozdělit, a první část vrať do pomocného řetězce s oddělovačem „-“ a kroky 2-5 zopakuj
 6. Není-li možné poslední načtené slovo (resp. jeho část) již rozdělit, aplikuj řešení 3
 - alternativně ještě lze vyzkoušet prostrkání (kerning) nebo uniformní zvětšení mezispicových mezer (tracking)

ZOBRAZENÍ NA TISKÁRNĚ

■ Elementy pro tisk stejné u většiny aplikací:

- Textové prvky
- Obrázek
- Čáry
- ...

■ Např. faktura

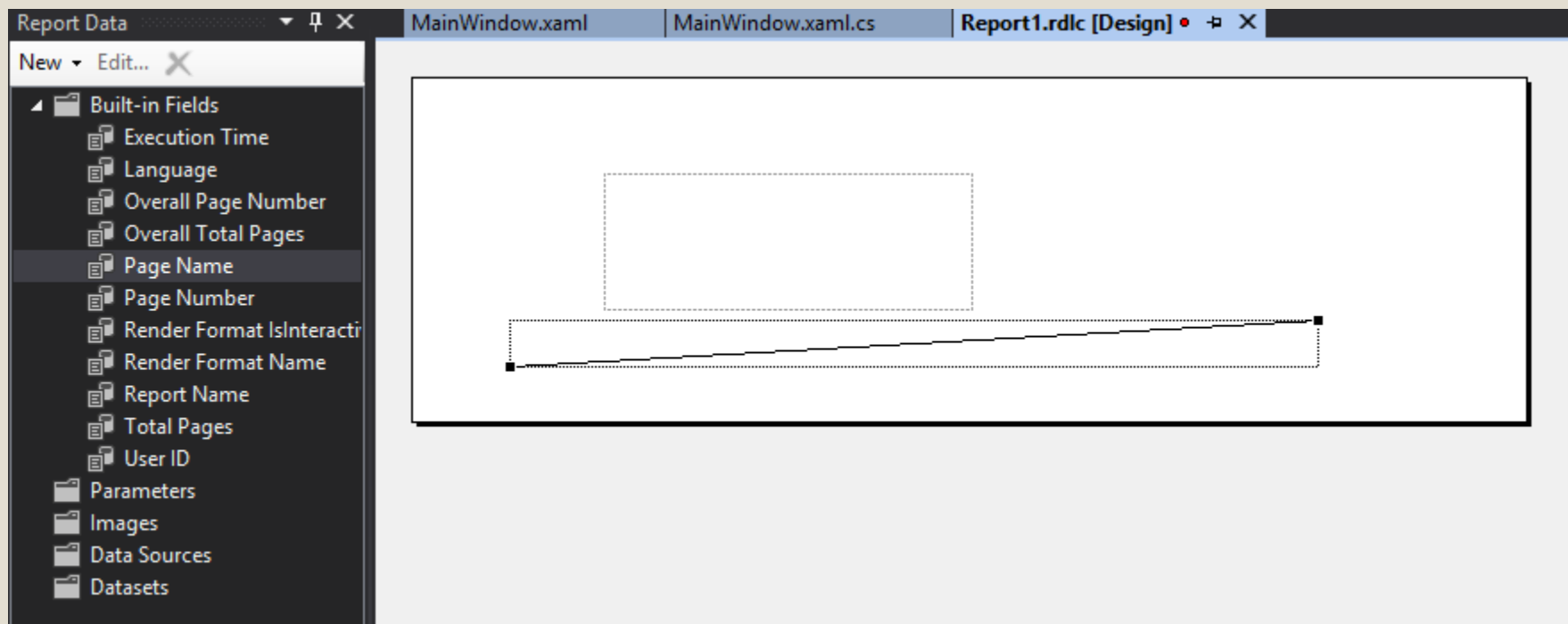
■ Data většinou z databáze

ČEZ, a. s.		Faktura - daňový doklad č. 235896																											
Dodavatel: ČEZ, a. s. Duhová 2/1444 14053 Praha 4		Variabilní symbol: 235896 Konstantní symbol: 308																											
IČO: 45274649 DIČ: CZ45274649 tel.: fax.: e-mail: OR: Městský soud v Praze, B 1581		Odběratel: IČO: 61672190 DIČ: CZ61672190 UNIPETROL, a.s. Klimentská 10 11005 Praha 1																											
Účet: IBAN:																													
Datum vystavení: 22.12.2005 Datum splatnosti: 5.1.2006 Datum zdan. plnění: 22.12.2005																													
<table border="1"><thead><tr><th>Označení položky</th><th>Množství</th><th>DPH %</th><th>Jedn. cena</th><th>Celkem bez DPH</th></tr></thead><tbody><tr><td>Dodávka elektřiny</td><td>1.00</td><td>19.00</td><td>255 820.00 Kč</td><td>255 820.00 Kč</td></tr><tr><td colspan="4">Celková částka bez DPH</td><td>255 820.00 Kč</td></tr><tr><td colspan="4">DPH</td><td>48 605.80 Kč</td></tr><tr><td colspan="4">Částka k úhradě</td><td>304 425.80 Kč</td></tr></tbody></table>					Označení položky	Množství	DPH %	Jedn. cena	Celkem bez DPH	Dodávka elektřiny	1.00	19.00	255 820.00 Kč	255 820.00 Kč	Celková částka bez DPH				255 820.00 Kč	DPH				48 605.80 Kč	Částka k úhradě				304 425.80 Kč
Označení položky	Množství	DPH %	Jedn. cena	Celkem bez DPH																									
Dodávka elektřiny	1.00	19.00	255 820.00 Kč	255 820.00 Kč																									
Celková částka bez DPH				255 820.00 Kč																									
DPH				48 605.80 Kč																									
Částka k úhradě				304 425.80 Kč																									
Razítko a podpis																													
http://www.faktura.cz/ - online faktura za pár vteřin																													
http://www.market.cz/ - Internetové stránky, které přinášejí zisk																													

ZOBRAZENÍ NA TISKÁRNĚ

- Většina aplikací proto dnes používá nějakou „reportovací“ knihovnu
- Knihovna poskytuje:
 - WYSIWYG vektorový editor obsahu jedno nebo více stránkového reportu
 - Uložen v interním formátu
 - Programové rozhraní pro:
 - Načtení reportu a jeho napojení na DB
 - Uložení naplněného reportu do HTML, PDF, ...
 - Vytisknutí naplněného reportu na tiskárnu
 - Programátor se o detaily spojené s tiskem nestará
- Např. Crystal Reports, JasperReports, ...

ZOBRAZENÍ NA TISKÁRNĚ



ZOBRAZENÍ NA TISKÁRNĚ

- Report nelze vyžít vždy
 - Drahá licence reportovací knihovny
 - Speciální požadavky
 - Data nejsou uložena v přístupné DB
- V případě MS Windows lze užít XPS
- Vlastní řešení: je třeba vytvořit grafický kontext (pro danou tiskárnu) a na něj kreslit
 - Grafické knihovny poskytují pomocné rutiny

ZOBRAZENÍ NA TISKÁRNĚ

- Obvyklá podpora ze strany knihovny:
 - Uživatelský dialog pro výběr tiskárny, formátu papíru, orientace (na výšku x na šířku), kvality tisku, ...
 - Vrací vytvořený grafický kontext
 - Metody pro zahájení a dokončení tisku dokumentu
 - Metody pro zahájení a dokončení tisku jedné strany

ZOBRAZENÍ NA TISKÁRNĚ

- WINAPI systémové řešení:
 - HDC hdcPrinter = CreateDC(...*ovladač tiskárny* ...) nebo PrintDlg(...)
 - StartDoc
 - StartPage
 - Kreslení na hdcPrinter
 - EndPage
 - EndDoc
 - ReleaseDC(hdcPrinter)

ZOBRAZENÍ NA TISKÁRNĚ

- Při kreslení je třeba myslet na limity:
 - Většina tiskáren nedokáže tisknout od kraje ke kraji
 - Minimálně 3 mm „bílý“ okraj
 - Papír se bude nějak sešívát
 - Vhodné nechat uživatele definovat vnitřní tiskovou plochu
- Kreslit efektivně!
 - Vytvoření zdrojů (pera, výplně, fonty, ...) společných pro všechny stránky před zahájením tisku

ZOBRAZENÍ NA TISKÁRNĚ V JAVĚ

■ Třída `PrinterJob`

- Metoda `printDialog()` pro výběr tiskárny
- Metoda `setPrintable` pro nastavení implementace pro tisk dokumentu o jedné nebo více stránkách
 - Musí implementovat rozhraní `Printable`
 - `Printable` má metodu `print(Graphics, ...)`
- Metoda `print()` pro zahájení tisku

ZOBRAZENÍ NA TISKÁRNĚ V JAVĚ

```
PrinterJob printJob = PrinterJob.getPrinterJob();
printJob.setPrintable(drawarea);

if (printJob.printDialog()) {
    try {
        printJob.print();
    } catch(PrinterException pe) {
        System.out.println("Error printing: " + pe);
    }
} //end if
```

ZOBRAZENÍ NA TISKÁRNĚ V JAVĚ

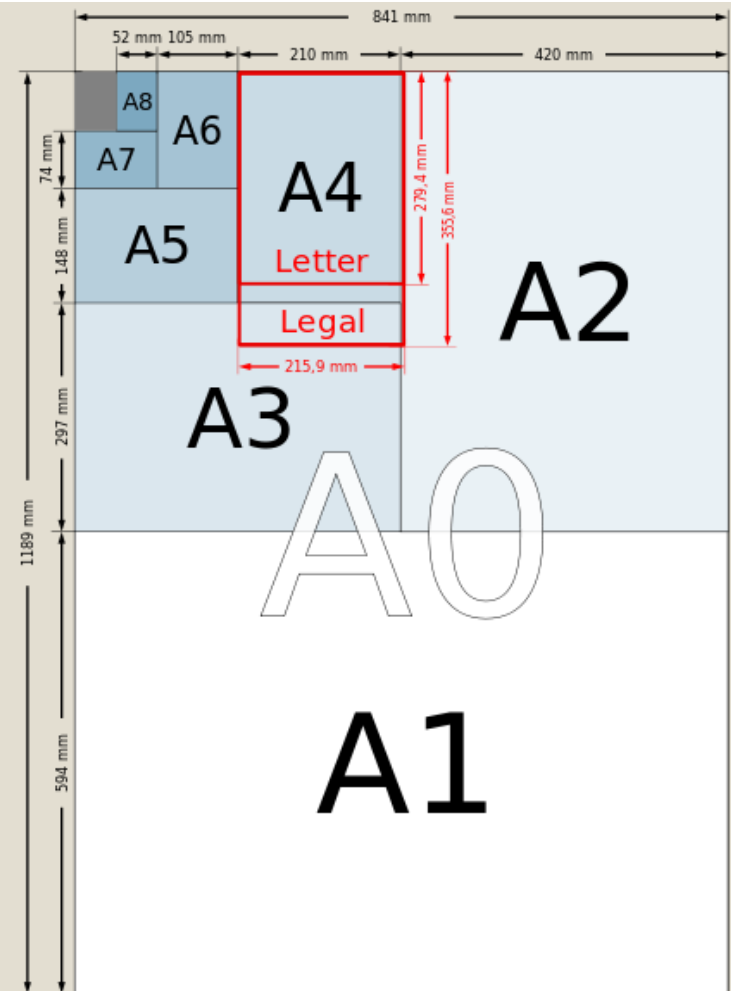
```
class TiskniDoc implements Printable {  
    public int print(Graphics g, PageFormat pageFormat,  
        int pageIndex) throws PrinterException {  
        if (pageIndex > 0){  
            return NO_SUCH_PAGE;  
        }  
  
        //tiskni na g stranku s indexem pageIndex  
  
        return PAGE_EXISTS;  
    }  
}
```

ZOBRAZENÍ NA TISKÁRNĚ V JAVĚ

- Třída `PageFormat` poskytuje informace o:
 - Velikost zvoleného papíru (v pt)
 - `getWidth()`, `getHeight()`
 - Tisknutelnou oblast (opět v pt) – limity tiskárny
 - `getImageableX()`, `getImageableY()`,
`getImageableWidth()`, `getImageableHeight()`
 - Orientaci papíru

FORMÁTY PAPÍRU

- Evropa
 - ISO 216 A, B, C
 - Nejběžnější A4
 - 210 x 297 mm
- USA
 - Nejběžnější ANSI A (Letter)
 - 8.5 x 11 palců, tj. 216 x 279
- Nikdy nepředpokládat jednu velikost papíru!



KONEC

- Příště: Pokročilá vektorová grafika

