Kapitola 1. Grafika

Obsah

Poznámka k literatuře:

- informací v TIJ3 je velmi málo a zastaralé
- informace v Tut nejsou úplně nejlépe organizovány

Základní charakteristika našeho snažení:

- aktivita, kdy sami chceme vykreslit nějaký grafický obrazec (úsečku, kružnici, atd.) nebo vypsat text
  - nejedná se o vykreslování jednotlivých komponent, např. zobrazování tlačítek apod.
    - to zajišťuje Swing automaticky

1.1. Jak se komponenty překreslují

- začíná se od hierarchicky nejvyššího kontejneru (často `JFrame`) a postupuje k nižším
  - o překreslování se většinou nemusí žádat, stačí jen komponentu změnit
    - např. `setText()` způsobí přepsání textu a i případnou změnu velikosti komponenty
  - to je zařízeno tak, že změna vzhledu komponenty způsobí automaticky vyvolání metody `repaint()` z `java.awt.Component`
    - ta zařídí, že požadavek na vykreslení je umístěn do fronty čekajících a vykreslí se metodou `paintComponent()`, jakmile na něj přijde řada

    - navíc, změní-li se pozice nebo velikost komponenty, je před `repaint()` zavolána ještě `revalidate()`
- je třeba si uvědomit, že ve `Swingu` jsou všechna volání součástí jednoho vlákna včetně volání `repaint()` a včetně reakcí na události
  - pokud jedna část trvá dlouho, ostatní musejí čekat, až doběhne
- aby bylo vykreslování co nejplynulejší (netrhané), používá se implicitně metoda dvojího bufferování (*double-buffered*)
  - komponenty se překreslují do bufferu v pozadí a až když je hotov, jednorázově se přepne na obrazovku
- existuje jednoduchý recept na vylepšení rychlosti vykreslování – nastavit neprůhledné komponenty (těch je prakticky v programu naprostá většina, ne-li všechny) na skutečně neprůhledné (*opaque*)
  - pro to slouží metoda *setOpaque(boolean isOpaque)* z `javax.swing.JComponent`
  - standardní nastavení je totiž `false` (= průhledné) což při vykreslování způsobí, že se musí kompletně vykreslit i komponenty, které jsou pod průhlednou komponentou
    - u některých komponent ale nastavení záleží na použitém L&F
  - je-li komponenta průhledná, nevykresluje se její pozadí nastavené pomocí `setBackground()`
    - naopak okraje komponenty, jsou-li vykreslovány, se vykreslují vždy

1.2. Grafické možnosti API

- jsou založené na primitivní (= minimální možnosti) třídě `java.awt.Graphics`
  - od JKD 1.2 je k dispozici potomek `java.awt.Graphics2D`
- metody třídy `Graphics2D` umožňují snadno:
  - kreslit čáry libovolné tloušťky
  - vyplňovat obrysy barvami, přechody, texturami
  - posouvat, rotovat, zmenšovat, zkosit grafická primitiva i text
  - pracovat s obrázky atp.

1.2.1. Kam kreslit a jak

- lze kreslit na jakýkoliv objekt zděděný od `javax.swing.JComponent`
  - nelze kreslit na `JFrame`
- pro kreslení je třeba překrýt metodu

```
protected void paintComponent(Graphics g)
```

- to znamená, že musíme vytvořit vlastní třídu zděděnou od `JComponent` (abstraktní třída)

- prakticky většinou dědíme od `JPanel`

- parametrem je grafický objekt (grafický kontext), který se získá „automaticky“
  - o jeho vytvoření se nemusíme starat, protože tuto metodu nikdy přímo nevoláme
    - vyvolává se metodou `repaint()`, která dokáže vynutit okamžik kreslení

1.2.2. Souřadnicový systém

- počátek souřadnic [0, 0] (typu `int` = počet pixelů) je v levém horním rohu
  - souřadnice `x` (šířka, *width*) roste doprava
  - souřadnice `y` (výška, *height*) roste dolů
- max. hodnota `x` a `y` (velikost prostoru, kam lze kreslit = „plátna“) je dána velikostí komponenty

  `x = getWidth()-1, y = getHeight()-1`
  - odečtení 1 je nutné, protože souřadnice jsou od 0
- protože se u komponenty mohou vyskytnout okraje, je vhodné použít

```
Insets insets = getInsets();
int minX = insets.left;
int minY = insets.top;
int maxX = getWidth() - insets.left - insets.right-1;
int maxY = getHeight() - insets.top - insets.bottom-1;
```

1.2.3. Práce s grafickým kontextem

- objekt třídy `Graphics` z historických důvodů předávaný do `paintComponent()` téměř nikdy nepoužíváme přímo
  - přetypovává se na `Graphics2D`

    `Graphics2D g2 = (Graphics2D) g;`
- tato třída má mnohem větší možnosti
- nese si s sebou následující defaultní nastavení renderovacích atributů
  - `Paint` – barva popředí komponenty (černá)
  - `Font` – font popisu komponenty (`Dialog`, `PLAIN`, 12 pt)
  - `Stroke` – typ čáry – tloušťka 1, nepřerušovaná čára
  - `Transform` – žádná transformace

  podrobnosti a ukázky viz v `tutorial\2d\overview\rendering.html`

Poznámka

1. objekt třídy `Graphics` předávaný do `paintComponent()` je pak také předáván do automaticky volané metody `paintBorder()`

   pokud tedy změníme nějaké nastavení, promítne se i do případného ohraničení

   pokud to nechceme, používáme:

```
Graphics2D g2 = (Graphics2D) g.create(); // kopie
// nějaké nastavení nad kopií
g2.translate(x, y);
...
g2.dispose(); // zrušení kopie
```

2. občas je vidět jako první příkaz metody `paintComponent()` volání `super.paintComponent(g);`
   - tím se zajistí, že se nejdříve vykreslí vše, co bylo požadováno v předkovi
   - pokud na kreslící komponentu jen jednorázově kreslíme, pak je tento příkaz zbytečný
   - má-li komponenta reagovat na `repaint()`, je tento příkaz vhodný (viz dále)

Příklad 1.1. Ukázka vytvoření kreslící komponenty a starého (z `Graphics`) a nového (z `Graphics2D`) způsobu kreslení

```
import javax.swing.*;
import java.awt.*;
import java.awt.geom.*;

class MojeKresliciKomponenta extends JPanel {
  MojeKresliciKomponenta() {
    setOpaque(true);
  }

  public Insets getInsets() {
    return new Insets(10, 10, 10, 10);
  }

  protected void paintComponent(Graphics g) {
    Insets insets = getInsets();
    int minX = insets.left;
    int minY = insets.top;
    int maxX = getWidth() - insets.left - insets.right - 1;
    int maxY = getHeight() - insets.top - insets.bottom - 1;
    // stary zpusob z Graphics
    g.setColor(Color.yellow);
    g.fillRect(minX, minY, maxX, maxY);
    Graphics2D g2 = (Graphics2D) g;
    // novy zpusob z Graphics2D
    g2.setPaint(Color.black);
    g2.draw(new Line2D.Double(0, 0, maxX, maxY));
  }
}
public class GrafikaZaklad {
  public static void main(String[] args) {
    JFrame oknoF = new JFrame("GrafikaZaklad");
    oknoF.add(new MojeKresliciKomponenta());
    oknoF.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    oknoF.setSize(250, 250);
    oknoF.setVisible(true);
  }
}
```

1.3. Základní grafická primitiva

- z `java.awt.geom`
- vykreslují se obrysově přednastavenou barvou pomocí

```
g2.draw(Shape s)
```

- nebo vyplněně (jsou-li uzavřená) pomocí

```
g2.fill(Shape s)
```


- od `Shape` jsou zděděny např.
  - přímka – `Line2D`
  - obdélníkové tvary – `Rectangle2D`, `RoundRectangle2D`, `Ellipse2D`, `Arc2D`
  - křivky – `CubicCurve2D`, `QuadCurve2D`
  - obecný tvar – `GeneralPath`

  `tutorial\2d\overview\shapes.html`
- toto jsou ale abstraktní třídy, které mají vždy dvě vnitřní třídy `.Double` a `.Float`
  - např. `Line2D.Double` a `Line2D.Float`
  - ty používají souřadnice `x` a `y` buď typu `double` nebo `float`
- prakticky je vykreslení přímky např.

```
g2.draw(new Line2D.Double(0.0, 0.0, 10.0, 10.0));
```

  Poznámka

  Přestože parametry `Line2D.Double()` mají být typu `double`, je možné používat typ `int`, protože dojde k automatické konverzi na `double`.

```
g2.draw(new Line2D.Double(0, 0, 10, 10));
```

- vyplnění obdélníka

```
g2.fill(new Rectangle2D.Double(0.0, 0.0, 10.0, 10.0));
```

- objekty primitiv nemusíme pokaždé vytvářet znovu (časově náročné), mají dostatečné množství nastavovacích metod, např.

```
setLine(double X1, double Y1, double X2, double Y2)
setLine(Line2D jinaLine)
setLine(Point2D p1, Point2D p2)
```

1.3.1. Porovnání starého a nového způsobu kreslení

- třída `Graphics` (od které je zděděna `Graphics2D`) dává pro vykreslování primitiv k dispozici použitelné metody typu `drawLine()`, `drawRect()`, `fillRect()`, ...
  - většinou to ale není výhodné, protože `fill()` a `draw()` z `Graphics2D` dokáží využít polymorfismu
- dále je třeba si uvědomit, že „starým způsobem“ vykreslená primitiva byly jen „pixely na obrazovce“
  - primitiva z `java.awt.geom` jsou kromě toho navíc objekty v paměti s mnoha užitečnými metodami, např.:
    - ♦ `boolean intersectsLine(Line2D l)`
      - tj. test, zda úsečka kříží jinou úsečku

- ♦ nebo `int outcode(double x, double y)`
  - – která pro obdélník určí, kde testovaný bod leží – uvnitř nebo vně (vlevo, vpravo, nahoře, dole)
- ♦ vytváříme-li aplikaci, která pouze jednorázově nevykresluje, jsou tyto metody velkou pomocí

1.4. Nastavení parametrů barvy a čáry

- jedná se o barvu obrysu či výplně a čáru obrysu
  - jsou to parametry typu rozhraní `Paint` (barva) a `Stroke` (čára) nastavované metodami třídy `Graphics2D`

    `void setPaint(Paint paint)` a `void setStroke(Stroke s)`

1.4.1. Rozhraní `Paint`

- toto rozhraní implementují třídy:
  - `Color` – jednolitá barva
  - `GradientPaint` – barevný přechod
  - `TexturePaint` – textura
- vlastnost se nastavuje pomocí `g2.setPaint(Paint p);`
- `Color` má mnoho možností (viz dokumentace)
  - typické použití je `Color(int r, int g, int b)` pro vytvoření libovolné neprůhledné *true-color* barvy
  - nebo použití jedné ze 13 již přednastavených „základních“ konstantních barev
    - ♦ `black`, `blue`, `cyan`, `darkGray`, `gray`, `green`, `lightGray`, `magenta`, `orange`, `pink`, `red`, `white`, `yellow`
    - ♦ použití je `Color.red` a lze použít i velká písmena `Color.RED`
  - dále lze používat i průhledné barvy (*transparency*) atd.
- `GradientPaint` má typický způsob použití:

```
GradientPaint(float x1, float y1, Color color1,
              float x2, float y2, Color color2)
```

  - např. postupný vodorovný přechod od červené ke žluté:

```
GradientPaint redToYellow =
         new GradientPaint(x, y, Color.red,
                           x + sirka, y, Color.yellow);
```


1.4.2. Rozhraní `Stroke`

- implementuje třída `BasicStroke`
- vlastnost se nastavuje pomocí `g2.setStroke(Stroke s);`
  - lze nastavit zejména tloušťku čáry (*width*)
    - ♦ typické použití: `new BasicStroke(2.0f);`

      Výstraha

      parametr je typu `float`, takže pouze výraz `(2.0)` způsobí chybu při překladu
- pokud je čára přerušovaná, lze ji libovolně nakonfigurovat použitím třídy:

```
BasicStroke(float   width,
            int     cap,
            int     join,
            float   miterlimit,
            float[] dash,
            float   dash_phase)
```

- `cap` tvar zakončení
  -  `CAP_BUTT` – pravoúhlé bez přesahu
  -  `CAP_ROUND` – zakulacené s přesahem
  -  `CAP_SQUARE` – pravoúhlé s přesahem
- `join` – typ napojení dvou čar
  - `JOIN_BEVEL` – useknuté
  -  `JOIN_MITER` – do špičky
  - `JOIN_ROUND` – do kulata
- `miterlimit` – maximální délka špičatého napojení
- `dash` – vzor přerušované čáry
  - přerušovaná: `float[] dash = {10.0f};`
  - čerchovaná: `float[] dash = {10.0f, 3.0f, 2.0f, 3.0f};`
- `dash_phase` – offset v počtu bodů, kdy má začít přerušovaná čára

```
float[] dash = {10.0f};
BasicStroke dashed =
        new BasicStroke(1.0f,
                        BasicStroke.CAP_BUTT,
```

```
                        BasicStroke.JOIN_MITER,
                        10.0f, dash, 0.0f);
```

Poznámka

Používáme-li základní primitiva a tenké čáry, je vhodné zapnout antialiasing, který často významně vylepší vzhled obrázku

```
g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                    RenderingHints.VALUE_ANTIALIAS_ON);
```


Příklad 1.2. Ukázka grafických primitiv

```
import java.awt.*;
import java.awt.geom.*;
import javax.swing.*;

public class Primitiva2D extends JPanel {
  public void paintComponent(Graphics g) {
    Graphics2D g2 = (Graphics2D) g;
    g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                        RenderingHints.VALUE_ANTIALIAS_ON);

    Dimension d = getSize();
    int gridWidth = d.width / 6;
    int gridHeight = d.height / 2;

    g2.setPaint(Color.white);
    g2.fill(new Rectangle2D.Double(0, 0, d.width, d.height));

    Font font = new Font("SansSerif", Font.PLAIN, 8);
    FontMetrics fontMetrics = g2.getFontMetrics(font);

    g2.setPaint(Color.lightGray);
    g2.draw3DRect(0, 0, d.width - 1, d.height - 1, true);
    g2.draw3DRect(3, 3, d.width - 7, d.height - 7, false);
    g2.setPaint(Color.black);

    int x = 5;
    int y = 8;
    int rectWidth = gridWidth - 2 * x;
    int stringY = gridHeight - 3 - fontMetrics.getDescent();
    int rectHeight = stringY - fontMetrics.getMaxAscent() - y - 5;

    // Line2D
    g2.draw(new Line2D.Double(x, y+rectHeight-1,
                              x + rectWidth, y));
    g2.drawString("Line2D", x, stringY);
    x += gridWidth;

    // Rectangle2D
    g2.setStroke(new BasicStroke(2.0f));
    g2.draw(new Rectangle2D.Double(x, y, rectWidth, rectHeight));
    g2.drawString("Rectangle2D", x, stringY);
    x += gridWidth;

    // RoundRectangle2D
//    float vzor[] = {10.0f, 3.0f, 2.0f, 3.0f};
    float vzor[] = {10.0f};
    BasicStroke dashed = new
                 BasicStroke(1.0f,
                             BasicStroke.CAP_BUTT,
                             BasicStroke.JOIN_MITER,
                             10.0f, vzor, 0.0f);
```

```
g2.setStroke(dashed);
g2.draw(new RoundRectangle2D.Double(x, y, rectWidth,
                                    rectHeight, 10, 10));
g2.drawString("RoundRectangle2D", x, stringY);
x += gridWidth;

// Arc2D
g2.setStroke(new BasicStroke(8.0f,
                             BasicStroke.CAP_BUTT,
                             BasicStroke.JOIN_MITER));
g2.draw(new Arc2D.Double(x, y, rectWidth, rectHeight,
                         90, 135, Arc2D.OPEN));
g2.drawString("Arc2D", x, stringY);
x += gridWidth;

// Ellipse2D
g2.setStroke(new BasicStroke(2.0f));
g2.draw(new Ellipse2D.Double(x, y,
                             rectWidth, rectHeight));
g2.drawString("Ellipse2D", x, stringY);
x += gridWidth;

// GeneralPath uzavrena - polygon
int x1Points[] = {x, x+rectWidth, x, x+rectWidth};
int y1Points[] = {y, y+rectHeight, y+rectHeight, y};
GeneralPath polygon = new
                    GeneralPath(GeneralPath.WIND_EVEN_ODD,
                                x1Points.length);
polygon.moveTo(x1Points[0], y1Points[0]);
for (int i = 1;  i < x1Points.length;  i++ ) {
  polygon.lineTo(x1Points[i], y1Points[i]);
}
polygon.closePath();
g2.draw(polygon);
g2.drawString("GeneralPath", x, stringY);

// druhy radek
x = 5;
y += gridHeight;
stringY += gridHeight;

// GeneralPath otevrena - polyline
int x2Points[] = {x, x+rectWidth, x, x+rectWidth};
int y2Points[] = {y, y+rectHeight, y+rectHeight, y};
GeneralPath polyline = new
                     GeneralPath(GeneralPath.WIND_EVEN_ODD,
                                 x2Points.length);
polyline.moveTo (x2Points[0], y2Points[0]);
for (int i = 1;  i < x2Points.length;  i++ ) {
  polyline.lineTo(x2Points[i], y2Points[i]);
}
g2.draw(polyline);
g2.drawString("GeneralPath otevřená", x, stringY);
```



```
x += gridWidth;

// Rectangle2D
g2.setPaint(Color.red);
g2.fill(new Rectangle2D.Double(x, y, rectWidth, rectHeight));
g2.setPaint(Color.black);
g2.drawString("Rectangle2D", x, stringY);
x += gridWidth;

// RoundRectangle2D
GradientPaint redToYellow = new GradientPaint(x, y, Color.red,
                                x + rectWidth, y, Color.yellow);
g2.setPaint(redToYellow);
g2.fill(new RoundRectangle2D.Double(x, y, rectWidth,
                                    rectHeight, 10, 10));
g2.setPaint(Color.black);
g2.drawString("RoundRectangle2D", x, stringY);
x += gridWidth;

// Arc2D
g2.setPaint(Color.red);
g2.fill(new Arc2D.Double(x, y, rectWidth, rectHeight,
                         90, 135, Arc2D.OPEN));
g2.setPaint(Color.black);
g2.drawString("Arc2D", x, stringY);
x += gridWidth;

// Ellipse2D
redToYellow = new GradientPaint(x, y, Color.red,
                                x+rectWidth, y,Color.yellow);
g2.setPaint(redToYellow);
g2.fill (new Ellipse2D.Double(x, y, rectWidth, rectHeight));
g2.setPaint(Color.black);
g2.drawString("Ellipse2D", x, stringY);
x += gridWidth;

// GeneralPath
int x3Points[] = {x, x+rectWidth, x, x+rectWidth};
int y3Points[] = {y, y+rectHeight, y+rectHeight, y};
GeneralPath filledPolygon = new
                          GeneralPath(GeneralPath.WIND_EVEN_ODD,
                                      x3Points.length);
filledPolygon.moveTo(x3Points[0], y3Points[0]);
for (int i = 1;  i < x3Points.length;  i++ ) {
  filledPolygon.lineTo(x3Points[i], y3Points[i]);
}
filledPolygon.closePath();
g2.setPaint(Color.red);
g2.fill(filledPolygon);
g2.setPaint(Color.black);
g2.draw(filledPolygon);
g2.drawString("GeneralPath", x, stringY);
}
```

```
  public static void main(String s[]) {
    JFrame f = new JFrame("Primitiva2D");
    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    f.getContentPane().add(new Primitiva2D());
    f.setSize(750, 300);
    f.show();
  }
}
```

Poznámka

pokud chceme tvar vyplnit a navíc orámovat, musíme použít nejdříve `fill()` a pak `draw()`

1.5. Křivky jako grafická primitiva

- existují dvě
  - `QuadCurve2D` – křivka určená dvěma koncovými body a jedním řídícím bodem
    - má jen jedno zakřivení
    - `tutorial\2d\display\Quad.html`
  - `CubicCurve2D` – Beziérova křivka určená dvěma koncovými body a dvěma řídícími body
    - má dvě zakřivení
    - `tutorial\2d\display\Cubic.html`

1.5.1. Obecná křivka – `GeneralPath`

- může být uzavřená (polygon) nebo otevřená (polyline)
- typicky se vytváří tak, že se nejdříve definují pole souřadnic `x` a `y`
- pak se vytvoří objekt třídy `GeneralPath`, kterému se určí způsob vyhodnocení, zda je nějaký bod uvnitř (*winding rule*) – většinou se použije `WIND_EVEN_ODD`
  - druhý parametr je počet bodů, ze kterých bude křivka skládat
- třetím krokem je nastavení výchozího bodu pomocí `moveTo()`
- pak následuje spojení bodů pomocí
  - `lineTo()` – spojení úsečkou
  - `quadTo()` – spojení křivkou
- poslední krok může být uzavření křivky pomocí `closePath()`

1.6. Afinní transformace

- `Graphics2D` poskytuje možnosti transformací
  - `translate` – posun souřadnic
  - `rotate` – natočení
  - `scale` – změna měřítka
  - `shear` – zkosení
- tyto transformace lze slučovat
- lze je provést nad celým kreslícím prostorem jednotlivě, pomocí metod z `Graphics2D`
  - `translate()`
  - `rotate()`
  - `scale()`
  - `shear()`

Příklad 1.3.

```
public class Transformace extends JPanel {
  protected void paintComponent(Graphics g) {
    Graphics2D g2 = (Graphics2D) g;
    Dimension d = getSize();
    int sirka = d.width / 3;
    int vyska = d.height / 3;
    g2.setStroke(new BasicStroke(10.0f));
//    g2.translate(sirka, vyska);
//    g2.rotate(Math.toRadians(45));
    g2.rotate(Math.toRadians(45), 1.5 * sirka, 1.5 * vyska);
//    g2.scale(0.5, 0.5);
//    g2.shear(0.5, 0.0);
    g2.setPaint(Color.yellow);
    g2.draw(new Rectangle2D.Double(sirka, vyska,
                                   sirka, vyska));
    g2.setPaint(Color.black);
    g2.draw(new Line2D.Double(sirka, 2 * vyska,
                              2 * sirka, vyska));
  }
```

- druhou možností je připravit si objekt třídy `AffineTransform` a pak pomocí

  `g2.transform(AffineTransform Tx)`

  tuto transformaci provést najednou

1.7. Interakce s uživatelem

- potřebujeme-li překreslit grafiku, aniž bychom měnili pozici a/nebo velikost základní kreslící komponenty musíme při reakci na událost volat metodu `repaint()` z kreslící komponenty

Příklad 1.4.

```
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import javax.swing.*;

class Kresleni extends JPanel {
  int pocet = 1;
  protected void paintComponent(Graphics g) {
//    super.paintComponent(g);
    Graphics2D g2 = (Graphics2D) g;
    Dimension d = getSize();
    int y = d.height;
    int krok = d.height / 20;
    g2.setStroke(new BasicStroke(10.0f));
    g2.setPaint(Color.yellow);
    g2.draw(new Line2D.Double(d.width / 2, y, d.width / 2,
                              y - (krok * pocet)));
  }
}

public class Klikani {
  Kresleni kresleni;
  Klikani() {
    JFrame oknoF = new JFrame("Klikani");
    kresleni = new Kresleni();
    JPanel jPN = new JPanel();
    JButton tBT = new JButton("+1");
    tBT.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        kresleni.pocet++;
        kresleni.repaint();
      }
    });
    jPN.add(tBT);
    oknoF.add(jPN, BorderLayout.NORTH);
    oknoF.add(kresleni, BorderLayout.CENTER);
    oknoF.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    oknoF.setSize(250, 250);
    oknoF.setVisible(true);
  }

  public static void main(String[] args) {
    new Klikani();
  }
}
```

1.8. Interakce uživatele s jednotlivými grafickými objekty

- občas potřebujeme, aby grafické objekty reagovaly na události od myši
  - grafická primitiva obsahují metodu `contains()`, která zjistí, zda jsme uvnitř objektu
    - pak stačí jen obsloužit události od `MouseListener` (kliknutí) a/nebo `MouseMotionListener` (pohyb)

Příklad 1.5.

```
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import javax.swing.*;

class Kresleni extends JPanel implements MouseListener,
                                    MouseMotionListener {
  int pocet;
  Color barva;
  Rectangle2D cara;
  Kresleni() {
    setOpaque(true);
    pocet = 1;
    barva = Color.yellow;
    cara = new Rectangle2D.Double();
    this.addMouseListener(this);
    this.addMouseMotionListener(this);
  }

  protected void paintComponent(Graphics g) {
    super.paintComponent(g);
    Graphics2D g2 = (Graphics2D) g;
    Dimension d = getSize();
    int y = d.height;
    int krok = d.height / 20;
    g2.setPaint(barva);
    cara.setRect(d.width / 2, y - (krok * pocet),
                 10, (krok * pocet));
    g2.fill(cara);
  }

  public void mouseClicked(MouseEvent e) {
    if (cara.contains(e.getX(), e.getY())) {
      if (barva.equals(Color.yellow)){
        barva = Color.red;
      }
      else {
        barva = Color.yellow;
      }
    }
    repaint();
  }
  public void mouseEntered(MouseEvent e) { }
  public void mouseExited(MouseEvent e) { }
  public void mousePressed(MouseEvent e) { }
  public void mouseReleased(MouseEvent e) { }

  public void mouseDragged(MouseEvent e) { }
  public void mouseMoved(MouseEvent e) {
    if (cara.contains(e.getX(), e.getY())) {
      barva = Color.green;
```

```
    }
    else {
      barva = Color.blue;
    }
    repaint();
  }
}
```

1.9. Práce s textem a fonty

- srovnáme-li popisy komponent a psaní textu do grafického okénka, pak jediná shodná věc je práce s nastavením fontů
  - třída `Font` a metody `getFont()` a `setFont()` fungují
  - dále však následují samé odlišnosti
- metody pro výpis textu:

```
drawString(String str, int x, int y)
```

- souřadnice `x` a `y` označují počáteční bod, odkud se bude text vypisovat
  - co je míněno pojmem „počáteční bod“?

```
g.drawString(s, 0, 0);
g.drawString(s, 0, 20);
g.drawString(s, 0 , d.height - 1);
```

- písmo na řádce je umístěno do tzv. **písmové osnovy**
  - tu tvoří několik vodorovných čar, nazývaných odborně dotažnice
  - tři základní z nich se jmenují **horní dotažnice** (*ascender line*), **základní dotažnice** neboli **účaří** (*baseline*) a **dolní dotažnice** (*descender line*)
    - ♦ význam *ascender line* je v typografii jiný – velikost velkých písmen
- metodě `drawString()` se předávají souřadnice levého bodu na základní dotažnici
  - to je zásadní rozdíl (a také častý zdroj výše zobrazených chyb) se zadáváním souřadnic grafických primitiv, u nichž se zadává vždy levý horní roh
- použitelnou (nikoliv přesnou!) hodnotu nutného posunutí první řádky textu lze získat `getSize()` třídy `Font`

  - udává celkovou velikost aktuálního fontu v typografických bodech

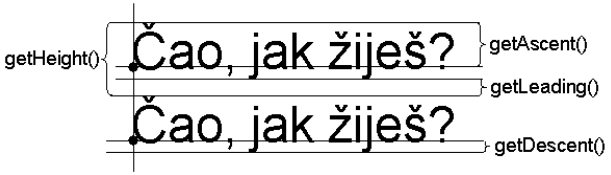
```
Font f = g.getFont();
int vel = f.getSize();
g.drawString(s, 0, 0 + vel);
g.drawString(s, 0 , d.height - vel);
```

1.9.1. Metrika fontu

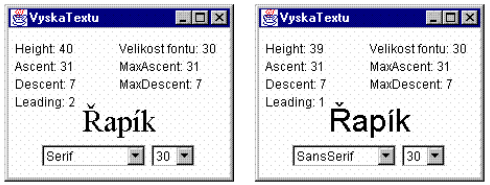
- pokud chceme detailnější informace o aktuálním fontu, musíme použít třídu `FontMetrics`
  - získáme i informace „šířkové“, kdy např. můžeme zjistit, zda se nějaký text v použitém fontu a v zadané velikosti ještě vejde na obrazovku
- instanci `FontMetrics` získáme pomocí `getFontMetrics()` třídy `Graphics2D`
  - metoda je přetížená

```
FontMetrics getFontMetrics(Font f)
```

    - ♦ vrátí metriku libovolného fontu
- `FontMetrics` má pouze metody, které vrací informace, nelze tedy pomocí nich nic měnit



- svislé rozměry fontu vrací `getAscent()`, `getDescent()`, `getLeading()` a `getHeight()`
  - `getHeight()` se typicky použije pro posun y-ové souřadnice při výpisu další řádky, tj. je to výška řádky
    - ♦ do těchto hodnot se započítávají i akcenty nad velkými písmeny
      - – toto nemusí být splněno pro všechny fonty
- změna různých rozměrů v závislosti na velikosti fontu a typu rodiny písma



- další skupina metod z této třídy umožňuje získat šířku znaku nebo řetězce v daném fontu
  - slouží pro vycentrování textu, zjištění, zda se vejde do okénka atp.
- šířka jednoho znaku `int charWidth(char ch)`
- šířka řetězce znaků braného jako celek

```
int stringWidth(String str)
```

- `int[] getWidths()` – vrátí šířky jednotlivých znaků pro prvních 256 znaků fontu
  - pomocí ní můžeme např. snadno najít „úsporné“ (tj. „štíhlé“) písmo
- `int getMaxAdvance()` – vrátí šířku nejširšího znaku ve fontu

1.9.2. Použití fyzických fontů

- grafický kontext nám umožňuje použít všechny fonty, které jsou dostupné v našem počítači
  - musíme vytvořit instanci třídy `java.awt.GraphicsEnvironment` voláním její statické metody `getLocalGraphicsEnvironment()`
  - pak lze použít metodu `getAllFonts()`, která vrátí pole objektů třídy `Font`

```
GraphicsEnvironment ge =
     GraphicsEnvironment.getLocalGraphicsEnvironment();
Font[] fonty = ge.getAllFonts();
```

  - z tohoto pole lze získat již známými metodami jména jednotlivých fontů a ty pak použít
- druhou možností je použít metodu vracející jména rodin písem

```
String[] jmena = ge.getAvailableFontFamilyNames();
```

  - a pak pomocí metod `deriveFont()` vytvořit požadovaný řez
- pokusy ukazují, že se můžete spolehnout na všechny logické fonty

Příklad 1.6. Ukázka použití fontů pro popis grafu

```
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import javax.swing.*;

class Kresleni extends JPanel {
  static final int OKRAJ = 10;
  int pocet = 1;
  int krok, maxY;
  Kresleni() {
    setOpaque(true);
  }
  protected void paintComponent(Graphics g) {
    super.paintComponent(g);
    Graphics2D g2 = (Graphics2D) g;
    nakresliOsy(g2);
    Dimension d = getSize();
    g2.setStroke(new BasicStroke(30.0f,
                                 BasicStroke.CAP_BUTT,
                                 BasicStroke.JOIN_MITER));
    g2.setPaint(Color.yellow);
    int px = d.width / 2;
    int py = maxY - (krok * pocet);
    g2.draw(new Line2D.Double(px, maxY, px, py));
    vypisHodnotu(g2, px, py);
  }

  void nakresliOsy(Graphics2D g2) {
    int horniMez = (pocet + 10) / 10 * 10;
    Dimension d = getSize();
    maxY = d.height - OKRAJ;
    krok = (maxY - OKRAJ) / horniMez;
    g2.setStroke(new BasicStroke(1.0f));
    g2.setPaint(Color.black);
    g2.draw(new Line2D.Double(OKRAJ * 2, maxY,
                              d.width - OKRAJ, maxY));
    g2.draw(new Line2D.Double(OKRAJ * 2, OKRAJ,
                              OKRAJ * 2, maxY));
    FontMetrics fm = g2.getFontMetrics();
    for (int i = 0;  i <= horniMez;  i += 5) {
      int py = maxY - (krok * i);
      g2.draw(new Line2D.Double(OKRAJ * 2 - 3, py,
                                OKRAJ * 2 + 3, py));
      if (i % 10 == 0) {
        String s = "" + i;
        int sirka = fm.stringWidth(s);
        int posun = fm.getAscent() / 2;
        g2.drawString(s, OKRAJ * 2 - 5 - sirka, py + posun);
      }
    }
  }
```

```
  void vypisHodnotu(Graphics2D g2, int px, int py) {
    if (pocet == 0) {
      return;
    }
    g2.setPaint(Color.red);
    Font f = g2.getFont();
    Font fb = new Font(f.getFontName(), Font.BOLD, 18);
    FontMetrics fm = g2.getFontMetrics(fb);
    g2.setFont(fb);
    String s = "" + pocet;
    int sirka = fm.stringWidth(s) / 2;
    int posun = fm.getAscent() / 2;
    int y = (maxY - py) / 2 - posun;
    g2.drawString(s, px - sirka, maxY - y);
  }
}

public class KlikaniGraf {
  Kresleni kresleni;
  KlikaniGraf() {
    JFrame oknoF = new JFrame("KlikaniGraf");
    kresleni = new Kresleni();
    JPanel jPN = new JPanel();
    JButton tBT = new JButton("+1");
    tBT.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        kresleni.pocet++;
        kresleni.repaint();
      }
    });
    JButton resetBT = new JButton("Reset");
    resetBT.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        kresleni.pocet = 0;
        kresleni.repaint();
      }
    });
    jPN.add(tBT);
    jPN.add(resetBT);
    oknoF.add(jPN, BorderLayout.NORTH);
    oknoF.add(kresleni, BorderLayout.CENTER);
    oknoF.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    oknoF.setSize(250, 250);
    oknoF.setVisible(true);
  }

  public static void main(String[] args) {
    new KlikaniGraf();
  }
}
```