Tvorba uživatelských komponent v knihovně Swing

5

V tomto semináři se seznámíme s tvorbou vlastních *grafických uživatelských prvků* (někdy též *komponent*) v knihovně Swing. V jednotlivých krocích si ukážeme základní možnosti, které knihovna Swing pro tvorbu vlastních komponent nabízí.

1 Jednoduchá pasivní komponenta

V knihovně Swing všechny grafické uživatelské prvky dědí ze třídy `JComponent`, která obstarává zakladní funkcionalitu společnou všem komponentám a poskytuje informace z okolního prostředí jako jsou rozměry komponenty nebo události spojené s touto komponentou.

Jako první si vytvoříme třídu dědící z `JComponent`.

```
package cz.upol.zp4jv.lecture05;
import javax.swing.JComponent;
public class MyComponent extends JComponent {
}
```

V dalším kroce bychom měli definovat vzhled komponenty. K tomu slouží chráněná metoda `void paintComponent(Graphics g)`, která je zavolána, kdykoliv je potřeba komponentu vykreslit, např. při vytvoření okna, jeho změně, apod. Argument této metody (objekt třídy `Graphics`) slouží jako „plátno", na které může komponenta vykreslit svůj vzhled.

Abychom viděli, jaké jsou skutečné rozměry komponenty, vykreslí naše ukázková komponenta nejdříve přes celou svou plochu „kříž" šedé barvy.

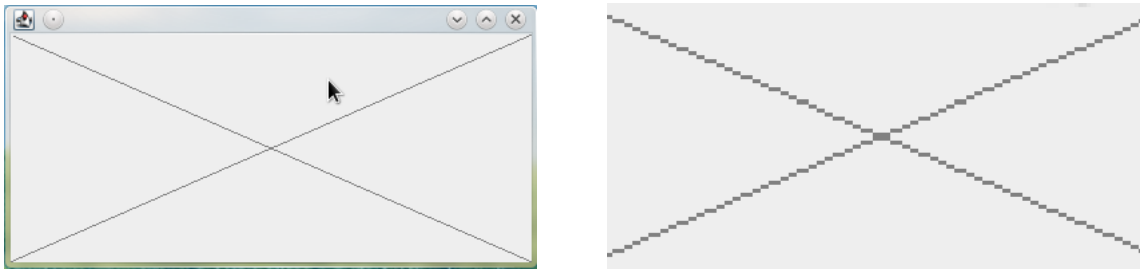
```
protected void paintComponent(Graphics g) {
   super.paintComponent(g);

   g.setColor(Color.GRAY);
   g.drawLine(0, 0, this.getWidth() - 1, this.getHeight() - 1);
   g.drawLine(0, this.getHeight() - 1, this.getWidth() - 1, 0);
}
```

Nejdříve je zvolena šedá barva a následně je vykreslena čára z levého horního rohu do pravého a čára z levého dolního do pravého horního rohu. Metody `int JComponent.getWidth()` a `int JComponent.getHeight()` nám poskytují informaci o velikosti komponenty.

Nyní již mámé komponentu, kterou můžeme vyzkoušet. Vytvoříme si okno, jehož obsahem bude nově vytvořená komponenta, a necháme okno zobrazit.

```
public class ComponentTest extends JFrame {
        public ComponentTest() {
                setDefaultCloseOperation(EXIT_ON_CLOSE);
                // pouziti nove komponenty
                getContentPane().add(new MyComponent());
                setPreferredSize(new Dimension(400, 200));
                pack();
        }
        public static void main(String[] args) {
                JFrame form = new ComponentTest();
                form.setVisible(true);
        }
}
```

Jak okno vypadá ukazuje Obrázek 1 (vlevo).



Obrázek 1: Vykreslení základní komponenty (vlevo), přiblížení (vpravo)

Pokud se na něj podíváme podrobněji, viz Obrázek 1 (vpravo), zjistíme, že čáry jsou „zubaté". Je to v důsledku tzv. aliasingu.[1]

Abychom se tohoto nepříjemného efektu zbavili, musíme změnit vlastnosti „plátna", na které vykreslujeme.

První krok je trochu neintuitivní. V metodě `paintComponent` musíme přetypovat argument z typu `Graphics` na `Graphics2D`, což je třída dědící z `Graphics`, která nabízí širší škálu metod, kterými lze manipulovat s obrazem.[2] Část těchto metod umožňuje definovat vlastnosti vykreslovaných objektů, např. zapnout anti-aliasing, který vykreslované objekty vhodně vyhladí.

Upravená metoda `paintComponent` vypadá následovně:

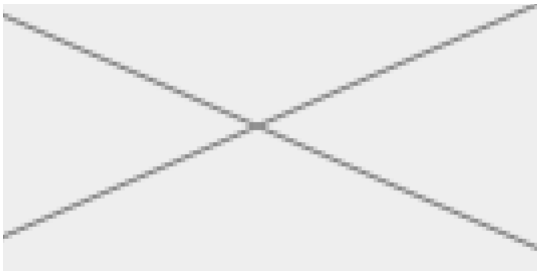
```
protected void paintComponent(Graphics g) {
    super.paintComponent(g);
```

[1] `https://en.wikipedia.org/wiki/Aliasing`

[2] Toto přetypování je bezpečné, protože knihovna Swing předává do metody `paintComponent` objekt třídy `Graphics2D`, ale z historických důvodů a z důvodů zpětné kompatibility zůstává v metodě argument typu `Graphics`.

```
    Graphics2D g2d = (Graphics2D) g;

    // aktivace anti-aliasingu
    g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);

    g2d.setColor(Color.GRAY);
    g2d.drawLine(0, 0, this.getWidth() - 1, this.getHeight() - 1);
    g2d.drawLine(0, this.getHeight() - 1, this.getWidth() - 1, 0);
}
```

Jak vypadá vykreslení komponenty ilustruje Obrázek 2.



Obrázek 2: Vykreslení komponenty se zapnutným anti-aliasingem

Úkol: Vyzkoušejte si další metody třídy Graphics2D, zejména ty, které umožňují vykreslovat různé geometrické útvary.

2 Reakce na události myši

V řadě případů očekáváme, že budou jednotlivé grafické uživatelské prvky reagovat na akce uživatele, ať už vyvolané myší nebo klávesnicí. Nejdříve si ukážeme, jak reagovat na události vyvolané myší, a v následující kapitole si ukážeme reakce na stisk kláves.

Události myši lze v knihovně Swing rozdělit do tří kategorií:

(i) jednorázové události (kliknutí, najetí myší nad komponentu, apod.),

(ii) pohyb myší (přesun myši, tažení),

(iii) pohyb kolečkem.

Jednorázové události lze odchytit pomocí třídy implementující rozhraní `MouseListener`, která je navázána na danou komponentu pomocí metody `addMouseListener`. Jedná se o podobný princip, s jakým jsme se setkali např. u tlačítek (`JButton.addActionListener`). V tomto případě však `MouseListener` obsluhuje hned několik různých typů událostí současně. Podle typu události je vyvolána příslušná metoda, které jsou jejím argumentem předány bližší informace o dané události, např. souřadnice, kde k události došlo, číslo tlačítka myši, apod.

```
interface MouseListener {
   public void mouseClicked(MouseEvent e);
   public void mousePressed(MouseEvent e);
   public void mouseReleased(MouseEvent e);
   public void mouseEntered(MouseEvent e);
   public void mouseExited(MouseEvent e);
}
```

Abychom si ukázali tuto funkcionalitu, rozšíříme ukázkovou komponentu o schopnost zapamatovat si, kde do ní bylo kliknuto.

Jednotlivé kliknutí zobrazíme jako kolečka, proto si zavedeme pomocnou metodu `drawPoint`:

```
/** na misto dane bodem "p" vykresli kolecko o velikosti POINT_SIZE */
private void drawPoint(Graphics2D g2d, Point p) {
   g2d.fillOval(p.x - POINT_SIZE / 2, p.y - POINT_SIZE / 2, POINT_SIZE, POINT_SIZE);
}
```

A pomocnou konstantu:

```
/** velikost vykreslovanych bodu */
private static final int POINT_SIZE = 20;
```

Dále si zavedeme atribut, který bude obsahovat seznam míst, kam bylo kliknuto:

```
/** seznam vykreslovanych bodu */
private LinkedList<Point> points = new LinkedList<Point>();
```

Odchytávání jednotlivých událostí je vhodné nastavit v konstruktoru:

```
public MyComponent() {
    this.addMouseListener(new MouseListener() {
        @Override public void mouseReleased(MouseEvent e) { }
        @Override public void mousePressed(MouseEvent e) { }
        @Override public void mouseExited(MouseEvent e) { }
        @Override public void mouseEntered(MouseEvent e) { }

        @Override public void mouseClicked(MouseEvent e) {
            points.add(new Point(e.getX(), e.getY()));
            repaint();
        }
    });
}
```

Metoda `mouseClicked` reaguje na kliknutí myší. V tomto případě uloží do seznamu `points` souřadnice, na nichž došlo ke kliknutí, a následně dojde k překreslení komponenty zavoláním metody `JComponent.repaint()`.

Zbývá doplnit kód pro vykreslení jednotlivých bodů (modrou barvou) do metody `paintComponent`:

```
g2d.setColor(Color.BLUE);
for (Point p : points)
    drawPoint(g2d, p);
```

Všimněme si, že jsme u rozhraní `MouseListener` museli „implementovat" i metody, které nic nedělají, což je kód navíc. Tuto nepříjemnost lze obejít pomocí třídy `MouseAdapter`, která implementuje rozhraní `MouseListener` tak, že žádná z jeho metod nevykonává žádnou činnost. Díky tomu můžeme použít `MouseAdapter` na místech, kde se očekává `MouseListener`, a kde implementujeme pouze metody, které jsou pro nás relevantní, např. následovně, pomocí anonymní třídy.

```
public MyComponent() {
    this.addMouseListener(new MouseAdapter() {
        @Override public void mouseClicked(MouseEvent e) {
            points.add(new Point(e.getX(), e.getY()));
            repaint();
        }
    });
}
```

Analogicky můžeme zavést reakce na pohyb myši. Rozšíříme proto komponentu tak, aby se pod kurzorem myši ukazoval bod.

Zavedeme si nejdříve attribut:

```
/** bod nachazejici se pod kurzorem mysi */
private Optional<Point> currentPoint = Optional.empty();
```

Jako typ atributu jsme použili třídu `Optional<Point>`, která nám umožňuje rozlišovat, zda se kurzor myši nachází nad komponentou a případně na jakých souřadnicích.

Začneme odchytávat relevantní události, tj. události, když se kurzor myši objeví nad komponentou a když ji opustí:

```
@Override
public void mouseEntered(MouseEvent e) {
   currentPoint = Optional.of(new Point(e.getX(), e.getY()));
   repaint();
}

@Override
public void mouseExited(MouseEvent e) {
    currentPoint = Optional.empty();
    repaint();
}
```

A dále budeme reagovat na pohyb myši:

```
this.addMouseMotionListener(new MouseAdapter() {
    @Override
    public void mouseMoved(MouseEvent e) {
        currentPoint = Optional.of(new Point(e.getX(), e.getY()));
        repaint();
    }
});
```

V tomto případě jsme místo implementace rozhraní `MouseMotionListener` opět použili třídu `MouseAdapter`, aby kód byl kratší a přehlednější.

Zbývá dodělat vykreslení příslušného bodu v metodě `paintComponent`:

```
currentPoint.ifPresent(p -> {
    g2d.setColor(Color.RED);
    drawPoint(g2d, p);
});
```

3 Reakce na stisk kláves

Reakce na stisk kláves vyžadují složitější mechanismus pro zpracování událostí. U jednotlivých komponent se rozlišuje, zda jsou aktivní (mají tzv. *focus*) a za jakých okolností mají reagovat na stisk kláves. Máme-li například v okně dvě textová pole, dává smysl, aby v jeden okamžik pouze jedno z těchto textových polí reagovalo na stisk kláves, tj. byl do něj zadáván text. Na druhou stranu máme-li tlačítko typu „Ok", chceme, aby se při stisku „Enter" vyvolala událost navázaná na toto tlačítko bez ohledu na to, jaká komponenta je aktivní.

Tento problém Swing řeší tak, že existují tři mapování typu *klávesa* → *název-události* pro situace, kdy (i) komponenta má focus, (ii) komponenta je rodičem komponenty, která má focus, a (iii) komponenta je v okně, které má focus. Vedle toho mají komponenty ještě mapování *název-události* → *akce*, které definuje, jak se má reagovat na příslušné stisky kláves.

Použití budeme opět demonstrovat na naší ukázkové komponentě. Přidáme možnost stiskem klávesy „DELETE" odstranit poslední vložený bod.

Nejdříve pomocí metody `getInputMap()` získáme mapovaní typu *klávesa* → *název-události* pro situace, kdy komponenta má focus, a na klávesu „DELETE" navážeme událost jménem `"deletePoint"`.

```
getInputMap().put(KeyStroke.getKeyStroke(KeyEvent.VK_DELETE, 0), "deletePoint");
```

Zbývá na jménem `"deletePoint"` navázat kód, který se má vyvolat při dané události:

```
getActionMap().put("deletePoint", new AbstractAction() {
    @Override
    public void actionPerformed(ActionEvent e) {
```

```
        if (points.size() > 0) points.removeLast();
        repaint();
    }
});
```