Metody programowania

Wprowadzenie do platformy Java

Dr inż. Andrzej Grosser

Częstochowa, 2013

Spis treści

1. Tworzenie graficznego interfejsu użytkownika cz. 1

Ten i następny rozdział stanowią krótkie wprowadzenie do tworzenia graficznego interfejsu użytkownika przy użyciu Javy. Niniejszy Rozdział zawiera informacje, w jaki sposób utworzyć okno i rysować w nim zarówno w postaci tekstu jak i prostej grafiki. Do napisania rozdziału użyto książki "Core Java"[1] i dokumentacji Javy.

1.1. Wprowadzenie do Swinga

Biblioteka standardowa Javy zawierała początkowo tylko podstawową bibliotekę graficznego interfejsu użytkownika - Abstract Window Toolkit (AWT). Biblioteka ta deleguje tworzenie elementów interfejsu użytkownika do natywnych zestawów kontrolek systemowych. Teoretycznie umożliwiało to zachowanie dla aplikacji wyglądu charakterystycznego dla danej platformy systemowej, w praktyce jednak powodowało powstawanie wielu problemów. Przede wszystkim zbiór kontrolek dla różnych systemów nie był taki sam - niektóre platformy dostarczają bogaty zbiór takich kontrolek, inne zaś dużo bardziej ubogi, co wymuszało określenie wspólnego zbioru. Ponadto różne implementacje AWT zawierały inne błędy, co powodowało konieczność debugowania aplikacji na wszystkich obsługiwanych platformach ("zaimplementuj raz i debuguj wszędzie").

Rozwiązaniem tego problemu było użycie innego podejścia, polegającego na rysowaniu kontrolek użytkownika na czystym oknie. Ten sposób wymagał od systemu jedynie dostarczenia okien i API do rysowania na tych oknach, co zmniejszało zależność biblioteki GUI od systemu i jednocześnie redukowało prawdopodobieństwo występowania zależnych od platformy systemowej błędów.

Opisywana w poprzednim akapicie metoda znalazła zastosowania przy implementacji biblioteki Swing. Biblioteka ta stała się rozszerzeniem Javy 1.1, zaś częścią biblioteki standardowej w Javie SE 1.2. Jest ona nadbudową AWT.

1.2. Tworzenie ramki

Okno najwyższego poziomu jest nazywane w Javie ramką (ang. *frame*). Klasą w Swing, która implementuje tego rodzaju okno jest `JFrame` (Biblioteka zawiera klasę `Frame`, lecz jest ona częścią AWT). Ta klasa jest jednym z wyjątków w Swing - okno nie jest rysowane przez mechanizmy biblioteczne, lecz bazuje na systemowym managerze okien. Wypełnia on elementy dodatkowe okna (np. ikony, pasek tytułowy, przyciski minimalizacji, maksymalizacji itp.).

Poniżej zamieszczono prostą aplikację wyświetlającą czyste okno na ekranie komputera:

```
1  package firstgui;
2
3  import java.awt.EventQueue;
4  import javax.swing.JFrame;
5
6  class MainFrame extends JFrame {
7      public MainFrame(String title) {
8          super(title);
9          setSize(640, 480);
10     }
11 }
12
13 class FirstGui {
14     public static void main(String[] args) {
15         EventQueue.invokeLater(new Runnable() {
16             public void run() {
17                 MainFrame frame = new MainFrame("Pierwsza GUI");
18                 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
19                 frame.setVisible(true);
20             }
21         });
22     }
23 }
```

Pierwsze trzy linijki kodu zawierają w kolejności deklarację pakietu oraz import klas EventQueue i JFrame (proszę zwrócić uwagę, że ze względów historycznych klasy Swing znajdują się w pakiecie zaczynającym się od nazwy javax, który zazwyczaj jest zarezerwowany dla rozszerzeń biblioteki Javy). Zamieszczona dalej właściwa implementacja dzieli

się na dwie klasy - pierwsza (MainFrame) implementuje główne okno aplikacji, natomiast druga klasa (FirstGui) jest odpowiedzialna za uruchomienie aplikacji.

Implementacja klasy MainFrame jest bardzo prosta, ogranicza się do implementacji dziedziczenia właściwości po klasie JFrame i dostarczenia konstruktora, w którym wysłano do konstruktora klasy nadrzędnej tytuł okna i ustawiono rozmiar okna (rozmiar domyślny - 0,0 jest niezbyt użyteczny).

Klasa FirstGui dostarcza elementu niezbędnego do uruchomienia aplikacji - metodę main(). Ponieważ wszystkie elementy biblioteki Swing muszą być konfigurowane z wątku obsługi zdarzeń (nazwa wynika z tego, że są do niego przekazywane zdarzenia - np. obsługa myszy czy klawiatury), dlatego kod zamieszczony w funkcji main musiał wykonać instrukcje w tym wątku (wątki będą omówione w ostatnim rozdziale, w tym momencie omawiany kod należy po prostu przyjąć w zamieszczonej postaci). Uruchomione instrukcje są odpowiedzialne za utworzenie ramki z odpowiednim napisem na belce, ustawienie zachowania aplikacji po zamknięciu ramki - w tym przypadku zamknięcie całego programu (domyślnie ramka jest ukrywana, ale program działa dalej) i ustawieniu widoczności ramki (ramka bez tej instrukcji nie jest widoczna na ekranie). Po wykonaniu wszystkich instrukcji funkcja main kończy swoje działania, nie jest to jednak koniec działania aplikacji, gdyż jest uruchomiony wątek obsługi komunikatów. Działa on aż do momentu, gdy ramka zostanie zamknięta lub zostanie wywołana metoda System.exit.

1.3. Właściwości ramki

Klasa JFrame ma tylko kilka metod pozwalających określić jak ramka ma wyglądać. Najważniejszymi z nich są:

- `setLocation()` i `setBounds()` - pozwalają na określenie pozycji ramki na ekranie,
- `setIconImage()` - informuje, która ikona ma być wyświetlona przy belce tytułowej,
- `setTitle()` - określa tytuł okna,
- `setResizable()` - określa czy rozmiar okna może być zmieniany przez użytkownika.

Obiekt klasy JFrame składa się z czterech powierzchni:

- powierzchni podstawowej (ang. *root pane*) - główna powierzchnia zawierająca pozostałe powierzchnie,

- powierzchni warstwy (ang. *layered pane*) - ustawia pozycję swojej zawartości, którym jest powierzchnia zawartości i opcjonalne menu,
- powierzchni szklanej (ang. *glass pane*) - domyślnie ukryta powierzchnia pozwalająca na wyłapywanie zdarzeń lub rysowanie (nazwa wynika z tego, że zachowuje się jak szyba ustawiona nad innymi komponentami),
- powierzchni zawartości (ang. *content pane*) - zawiera widzialne komponenty (z wyjątkiem menu).

1.4. Rysowanie i malowanie w Swing

Od Javy 1.2 biblioteka standardowa zawiera klasy umożliwiające implementację wielu operacji graficznych z uwzględnieniem między innymi skalowania (wcześniejsze wersje biblioteki standardowej wprowadzały jedynie podstawowe operacje). Do rysowania kształtów konieczne jest użycie obiektu klasy `Graphics2D`, która jest podklasą `Graphics`. Ta relacja dziedziczenia jest o tyle istotna, gdyż metody odpowiedzialne za rysowanie (np. `paintComponent()`) przyjmują obiekt Graphics. Uzyskanie potrzebnego do rysowania obiektu Graphics2D wymaga rzutowania:

```
1 @Override
2 public void paintComponent(Graphics g) {
3     Graphics2D g2d = (Graphics2D) g;
4 }
```

Figury geometryczne z bibliotek Java 2D są prezentowane w postaci klas:

- Point2D - punkt,
- Line2D - linia,
- Path2D - ścieżka składająca się z segmentów (w tym krzywoliniowych),
- QuadCurve2D, CubicCurve2D - krzywe,
- Rectangle2D - prostokąt,
- Ellipse2D - elipsa,
- Arc2D - wycinek elipsy.

Wszystkie te klasy implementują interfejs `Shape`.

Stworzenie figury na ekranie wymaga utworzenia stosowanego obiektu a następnie narysowania go na ekranie z wykorzystaniem obiektu Graphics2D:

```
Ellipse2D circ = new Ellipse2D.Double(10.0, 10.0, 20.0, 20.0);
g2D.draw(circle);
```

Współrzędne i rozmiary figur są wyznaczane z wykorzystaniem liczb zmiennoprzecinkowych. Początek układu współrzędnych użytkownika jest w lewym górnym rogu. Współrzędne x-owe są zwiększane w kierunku na prawo, natomiast y-owe w kierunku w dół.

W celu uniknięcia konieczności wykonywania wielu uciążliwych konwersji (konwersja z double do float musi być jawna) projektanci biblioteki zadecydowali, że będą implementowane dwie wersje figur - jedna dla liczb podwójnej precyzji, druga dla liczb pojedynczej precyzji. Są one implementowane w postaci statycznych klas wewnętrznych Double i Float (żeby było jeszcze ciekawiej dziedziczą one pod swoim obiekcie figury - co umożliwiło pominięcie po lewej stronie nazwy klasy wewnętrznej - Eclipse2D zamiast Eclipse2D.Double).

Wybór koloru używanego do rysowania lub malowania jest wykonywany za pośrednictwem funkcji `setPaint()`. Przyjmuje ona obiekt klasy implementującej interfejs `Paint`, jedną z takich klas jest `Color`. Kolor można ustawić przekazując stałą dosłowną zdefiniowaną w klasie Color lub korzystając z konstruktora klasy Color podając kod RGB:

```
Ellipse2D circ = new Ellipse2D.Double(10,10,10,10);
g2d.setPaint(Color.YELLOW);
g2d.fill(circ);
Rectangle2D rect = new Rectangle2D.Double(20,20,10,10);
g2d.setPaint(new Color(155,140,0));
g2d.draw(rect);
```

Zamieszczony powyżej kod spowoduje namalowanie na ekranie wypełnionych kolorem żółtym koła i kwadratu o ciemnopomarańczowej obwódce.

Do nadawania stylów liniom zewnętrznym figur (konturom) można użyć metody `setStroke()`. Wymieniona metoda przyjmuje obiekt klasy implementującej interfejs `Stroke`. Najprostszym sposobem nadania stylu jest użycie obiektu klasy `BasicStroke()`, konstruktory tej klasy umożliwiające uzyskanie linii o zadnie grubości, stylu końca linii, stylu połączenia pomiędzy segmentami i stylów kreskowania. Na przykład:

```
Ellipse2D circ = new Ellipse2D.Double(10,10,10,10);
g2d.setStroke(new BasicStroke(4));
g2d.draw(circ);
Rectangle2D rect = new Rectangle2D.Double(20,20,10,10);
g2d.setStroke(new BasicStroke(2, BasicStroke.CAP_BUTT,
    BasicStroke.JOIN_ROUND);
```

```
6 g2d.draw(rect);
```

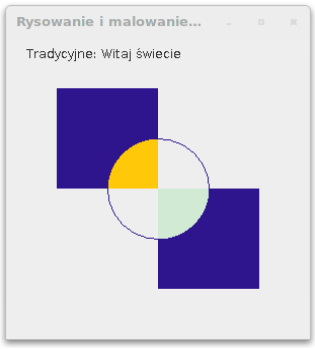
Wyświetlany jest tutaj okrąg jest rysowany z użyciem linii o grubośći 4 jednostek i kwadrat z wykorzystaniem linii o grubości dwóch jednostek oraz zaokrąglonym stylu połączenia pomiędzy segmentami.

Tekst można na ekranie wyświetlić korzystając z metody `drawString()`. Należy do niej przekazać wymagany łańcuch znaków i współrzędne od których zacznie się rysowanie. Na przykład:

```
1 g2d.drawString("Witaj␣świecie", 10, 10);
```

W wyniku działania zamieszczonego powyżej kodu zostanie narysowany na ekranie napis Witaj świecie zaczynając od współrzędnych 10, 10 (wyznacza się w ten sposób początek linii bazowej[1] pierwszego znaku).

Przykładową aplikację korzystającą z mechanizmów rysowania Swing zaprezentowano na rysunku 1.1. Program prezentuje, w jaki sposób narysować tekst na ekranie i wyświetlić kontury oraz wypełnione figury.



Rys. 1.1. Wygląd okna aplikacji

Pełen kod źródłowy prezentowanego programu wygląda następująco:

```
1 import javax.swing.*;
2 import java.awt.*;
3 import java.awt.geom.Arc2D;
4 import java.awt.geom.Ellipse2D;
5 import java.awt.geom.Rectangle2D;
6
7 public class MainApp {
8     public static void main(String[] args) {
```

[1]Linia bazowa to niewidoczna linia, na której opierają się części dolne znaków takich jak e, a czy k.

```
9           EventQueue.invokeLater(new Runnable() {
10              public void run() {
11                  JFrame frame = new MainFrame();
12                  frame.setTitle("Rysowanie i malowanie w Javie");
13                  frame.setDefaultCloseOperation(JFrame.
                        EXIT_ON_CLOSE);
14                  frame.setVisible(true);
15              }
16          }
17          );
18      }
19  }
20
21  class MainFrame extends JFrame {
22      public MainFrame() {
23          add(new MyComponent());
24          pack();
25      }
26  }
27
28  class MyComponent extends JComponent {
29      private void paintTextHello(Graphics2D g2d) {
30          g2d.setPaint(Color.BLUE);
31          g2d.drawString("Tradycyjne: Witaj świecie", 20,20);
32      }
33
34      private void paintShapes(Graphics2D g2d) {
35
36          int midX = 200;
37          int midY = 200;
38
39          Rectangle2D rect1 = new Rectangle2D.Double(midX/2 - 50,
                midY/2 - 50, 100, 100);
40          g2d.setPaint(new Color(46, 20, 141));
41          g2d.fill(rect1);
42          Rectangle2D rect2 = new Rectangle2D.Double(midX/2 + 50,
                midY/2 + 50, 100, 100);
43          g2d.fill(rect2);
44          //g2d.setPaint();
45          Ellipse2D circle = new Ellipse2D.Double(midX/2,midY
                /2,100,100);
46          g2d.draw(circle);
47          Arc2D arc1 = new Arc2D.Double(circle.getBounds(),90, 90,
                Arc2D.PIE);
```

```
48          g2d.setPaint(new Color(255, 200, 8));
49          g2d.fill(arc1);
50          Arc2D arc2 = new Arc2D.Double(circle.getBounds(),270,
                90,Arc2D.PIE);
51          g2d.setPaint(new Color(209,234,211));
52          g2d.fill(arc2);
53      }
54
55      @Override
56      public void paintComponent(Graphics g) {
57          Graphics2D g2d = (Graphics2D)g;
58          paintTextHello(g2d);
59          paintShapes(g2d);
60      }
61
62      @Override
63      public Dimension getPreferredSize() {
64          return new Dimension(300,300);
65      }
66  }
```

Pierwsze linie to importy klas używanych w programie. Pierwsza klasa MainApp dostarcza implementacji funkcji main(), w której utworzone jest główne okno programu (wykonywane są podobne czynności jak w pierwszym przykładzie z tego rozdziału). Kolejna klasa MainFrame jest odpowiedzialna za główne okno programu. Prezentowany kod ogranicza się do konstruktora, w którym dodawany jest obiekt komponentu (`add()`) i ustawiany preferowany rozmiar dla tej kontrolki (metodą `pack()`). Ostatnia klasa MyComponent dziedzicząca po JComponent dostarcza implementacji dwóch metod publicznych - paintComponent() i getPreferredSize(). Metoda paintComponent() jest odpowiedzialna za narysowanie tekstu i figur. Te zadania są oddelegowane do pomocniczych, prywatnych metod paintTextHello() i paintShapes(). Metoda paintTextHello() rysuje na ekranie niebieski tekst - Tradycyjne: Witaj świecie. Metoda paintShapes() kreśli na ekranie odpowiednie figury geometryczne - dwa wypełnione prostokąty, kontur okręgu i dwa wypełnione wycinku koła. Metoda getPreferredSize() zwraca preferowany rozmiar kontrolki wykorzystywanej tutaj do rysowania.

Literatura

[1] C. S. Horstmann and G. Cornell. *Core Java Volume I–Fundamentals*. Prentice Hall, 2011.