Język JAVA
podstawy programowania

Jacek Rumiński

Wykład 5, część 4

Plan wykładu:

1. Wprowadzenie do grafiki w Javie
2. Budowa GUI: komponenty, kontenery i układanie komponentów
3. Budowa GUI: obsługa zdarzeń
4. Grafika wektorowa – porysujmy sobie
5. Reprezentacja koloru
6. Grafika rastrowa - obrazy
7. Obsługa czcionek

Obraz – klasa Image

Możliwości pozyskiwania, tworzenia i przetwarzania obrazów w Javie są bardzo duże. Począwszy od pierwszych wersji w ramach biblioteki AWT poprzez Java2D, a skończywszy na dedykowanej bibliotece JAI (Java Advanced Imaging API) język JAVA dostarcza wiele narzędzi do tworzenia profesjonalnych systemów syntezy, przetwarzania, analizy i prezentacji obrazów.

W początkowych wersjach bibliotek graficznych Javy podstawą pracy z obrazami była klasa java.awt.Image. Obiekty tej abstrakcyjnej klasy nadrzędnej uzyskiwane są w sposób zależny od urządzeń (nie ma możliwości wywołania new Image(…), musimy stosować metody, które zwracają obiekt klasy Image). Podstawowa metoda zwracająca obiekt typu Image to getImage(). Inna możliwość uzyskania obiektu klasy Image to utworzenie go za pomocą metody createImage().

Obraz – klasa Image

Metoda getImage() związana jest z klasą Toolkit, natomiast dla apletów z klasą Applet. Wywołanie metody getImage() polega albo na podaniu ścieżki dostępu (jako String) lub lokalizatora URL do obrazu przechowywanego w formacie GIF, PNG lub JPEG. Przykładowo:

Image obraz = Toolkit.getDefaultToolkit.getImage("jacek.gif");
- zwraca obiekt obraz na podstawie obrazu przechowywanego w pliku jacek.gif w bieżącej ścieżce dostępu

Image obraz = Toolkit.getDefaultToolkit.getImage(new URL("http://www-med.eti.pg.gda.pl/~jwr/icons/jwrs4.gif");
- zwraca obiekt obraz na podstawie obrazu przechowywanego w pliku jwrs4.gif na serwerze www-med w danych podkatalogach.

Istnieją jeszcze inne metody "otwierania obrazów", ale o tym dalej.

```
import java.awt.*;
import javax.swing.*;
public class OtworzObraz extends JFrame{
  Image img;
  public OtworzObraz(String txt){
    super(txt);
    img = Toolkit.getDefaultToolkit().getImage("c:\\java\\kody_zrodlowe\\maly_motyl.jpg");
    try {//pętla w której czekamy na skonczenie produkcji obrazu dla obiektu img
          MediaTracker mt = new MediaTracker(this);
          mt.addImage(img, 0);            mt.waitForID(0);
    } catch (Exception e) {}
    int iw = img.getWidth(this);        int ih = img.getHeight(this);          setSize(iw,ih);
  }//koniec OtworzObraz()
  public void paint(Graphics g){              g.drawImage(img,0,0,this);     }
  public static void main(String a[]){
              //Warto dodać utworzenie nowego wątka i wywołać go poprzez SwingUtilities
              OtworzObraz oo=new OtworzObraz("Demonstracja tworzenia obiektu Image");
              oo.setVisible(true);
  }//koniec main()
}//koniec class
```

Obraz – klasa Image

Aby utworzyć obiekt klasy Image należy posłużyć się metodą createImage(). Argument tej metody to obiekt implementujący interfejs ImageProducer, czyli twórcę obrazu. W bibliotekach Javy możemy znaleźć klasę MemoryImageSource. Klasa ta dostarcza szereg konstruktorów, w których podaje się: typ modelu kolorów (ColorModel) lub wykorzystuje się domyślny RGB, rozmiar tworzonego obrazu, macierz (tablicy) z wartościami pikseli. Przykładowo w następujący sposób można wygenerować własny obiekt typu Image:

```
Image img = createImage(new MemoryImageSource(w, h, pix, 0, w));
```

Tworzony jest obraz w domyślnym systemie RGB o szerokości *w*, wysokości *h*, na podstawie tablicy próbek *pix*, bez przesunięcia w tej tablicy z *w* elementami w linii.

Język JAVA – obrazy cyfrowe


```
import java.awt.event.*;  import java.awt.image.*;
import java.awt.*; import javax.swing.*;
public class ObrazJedi extends Frame {
   Image ob;
   ObrazJedi () {    super ("Obraz");     setSize(400, 430);     ob=stworzObraz();    }//ObrazJedi
   public Image stworzObraz(){
      int w = 300;  //szerokość obrazu
      int h = 300;   //wysokość obrazu
      int pix[] = new int[w * h]; //tablica wartości próbek
      int index = 0;
      for (int y = 0; y < h; y++) {//generacja przykładowego obrazu
         int red = (y * 255) / (h - 1);
         for (int x = 0; x < w; x++) {
            int blue = (x * 255) / (w - 1);
            pix[index++] = (255 << 24) | (red << 16) | blue;
         }
      }
      Image img = createImage(new MemoryImageSource(w, h, pix, 0, w));
      //tworzony jest obraz w RGB o szerokości w, wysokości h,
      //na podstawie tablicy prUbek pix, bez przesuniĺcia w tej tablicy z w elementami w linii
      return img;     }//stworzObraz()
```

Język JAVA – obrazy cyfrowe


```
public void paint (Graphics g) {
      Insets insets = getInsets();
      g.translate (insets.left, insets.top);
      g.drawImage(ob,50,50,this); //wyrysuj obraz od punktu 50,50
   }//paint()

   public static void main (String [] args) {
            SwingUtilities.invokeLater(new Runnable() {//interfejs
                public void run() {//metoda wątku
                   Frame f = new ObrazJedi();
                   f.addWindowListener(new WindowAdapter(){
                       public void windowClosing(WindowEvent e){
                           System.out.println("Dziekujemy za prace z programem...");
                           System.exit(0);
                       }//windowClosing()
                    });
                    f.setVisible(true);
                  }//run()
               });//invokeLater()
   }//main()
}//koniec public class ObrazJedi extends Frame
```

Obraz – klasa Image

Mając obiekt typu *Image* można obraz z nim związany wyświetlić (narysować) używając jednej z metod *drawImage*(). W najprostszej metodzie *drawImage*() podając jako argument obiekt typu *Image*, współrzędne {x, y} oraz obserwatora (*ImageObserver*, często w ciele klasy komponentu, w którym się rysuje odwołanie do obserwatora jest wskazaniem aktualnego obiektu komponentu - *this*) możemy wyświetlić obrazu w dozwolonym do tego elemencie (np. w JFrame). Inna wersja metody *drawImage*() umożliwia skalowanie wyświetlanego obrazu poprzez podanie dodatkowych argumentów: szerokość (*width*) i wysokość (*height*).

Jeśli obraz został utworzony można uzyskać informacje o jego wymiarach posługując się metodami klasy Image: *getWidth*() oraz *getHeight*(). Aby metody te zwróciły konkretne wartości dane obrazu (Image) muszą być w pełni wczytane lub wytworzone. W razie problemów należy użyć klasę MediaTracker (opis w dokumentacji).

Język JAVA – obrazy cyfrowe


```
import java.awt.*;     import javax.swing.*;
public class SkalujObraz extends JFrame{
  Image img;  int width=200;  int height=200;
  public SkalujObraz(String txt){
    super(txt);
    img = Toolkit.getDefaultToolkit().getImage("c:\\java\\kody_zrodlowe\\maly_motyl.jpg");
    try {//pętla w której czekamy na skonczenie produkcji obrazu dla obiektu img
        MediaTracker mt = new MediaTracker(this);  mt.addImage(img, 0);  mt.waitForID(0);
     } catch (Exception e) {}
    int iw = img.getWidth(this);
    int ih = img.getHeight(this);
    JOptionPane.showMessageDialog(null, "Oryginalna szer. obrazu="+iw+", wys.="+ih);
    setSize(width,height);
  }//SkalujObraz()
  public void paint(Graphics g){     g.drawImage(img,0,0,width,height,this);   }//paint()
  public static void main(String a[]){
    SwingUtilities.invokeLater(new Runnable() {//interfejs
     public void run() {//metoda wątku
           SkalujObraz so=new SkalujObraz("Demonstracja skalowania obiektu Image");
           so.setVisible(true);
     }//run()          });//invokeLater()    }//main    }//class SkalujObraz
```

Obraz – klasa BufferedImage

Java2D API rozszerza, a zarazem zmienia koncepcję pracy z obrazami w *Javie*. Podstawową klasą jest w tej koncepcji klasa *BufferedImage*, będącą rozszerzeniem klasy Image z dostępnym buforem danych.

Obiekt *BufferedImage* może być stworzony bezpośrednio w pamięci i użyty do przechowywania i przetwarzania danych obrazu uzyskanego z pliku lub poprzez URL.

Obraz *BufferedImage* może być wyświetlony poprzez użycie obiektów klasy *Graphics2D*. Obiekt *BufferedImage* zawiera dwa istotne obiekty: obiekt danych - *Raster* oraz model kolorów *ColorModel*. Klasa Raster umożliwia zarządzanie danymi obrazu. Na obiekt tej klasy składają się obiekty *DataBuffer* oraz *SampleModel*.

Obraz – klasa BufferedImage

DataBuffer stanowi macierz wartości próbek obrazu, natomiast SampleModel określa sposób interpretacji tych próbek.
Przykładowo dla danego piksela próbki (RGB) mogą być przechowywane w trzech różnych macierzach (*banded interleved*) lub w jednej macierzy w formie przeplatanych próbek (*pixel interleved*) dla różnych komponentów (R1G1B1R2G2B2...).
Rolą *SampleModel* jest określenie jakiej formy użyto do zapisu danych w macierzach.
Najczęściej nie tworzy się bezpośrednio obiektu Raster lecz wykorzystuje się efekt działania obiektu *BufferedImage*, który rozbija *Image* na *Raster* oraz *ColorModel*. Niemniej istnieje możliwość stworzenia obiektu *Raster* poprzez stworzenie obiektu *WritableRaster* i podaniu go jako argumentu w jednym z konstruktorów klasy

Obraz – klasa BufferedImage

No dobrze, ale jak używać klasy BufferedImage? Istnieją dwie podstawowe metody:

1. Przegrać dane obrazu z obiektu klasy Image do BufferedImage,
2. Otworzyć plik z obrazkiem metodą bezpośrednio tworzącą obiekt klasy BufferedImage (pakiet javax.imageio).

Pierwsza metoda wymaga trzech kroków:

- Posiadania obiektu klasy Image (poprzez otwarcie pliku lub utworzenie obrazu),
- Utworzenie bufora danych (BufferedImage) o rozmiarach właściwych dla obrazu z obiektu Image,
- Przegraniu (wrysowaniu) danych obiektu Image do bufora obiektu BufferedImage.

Język JAVA – obrazy cyfrowe


```
import java.awt.*;  import java.awt.image.*;  import javax.swing.*;
public class BuforowanyObraz extends JFrame{
  BufferedImage bi;
  public BuforowanyObraz(String txt){
    super(txt); Image img;
    img=Toolkit.getDefaultToolkit().getImage("c:\\java\\kody_zrodlowe\\maly_motyl.jpg");
    try {//pętla w której czekamy na skonczenie produkcji obrazu dla obiektu img
          MediaTracker mt = new MediaTracker(this);
          mt.addImage(img, 0);            mt.waitForID(0);
    } catch (Exception e) {}
    int iw = img.getWidth(this);        int ih = img.getHeight(this);
    bi = new BufferedImage(iw,ih,BufferedImage.TYPE_INT_RGB);//utwórz bufor obrazu
    Graphics2D g2=bi.createGraphics();//pobierz kontekst graficzny
    g2.drawImage(img,0,0,this); //wrysuj obraz img do bufora
    g2.setColor(Color.red);//ustaw kolor do rysowania
    g2.setStroke(new BasicStroke(6f));//ustaw grubość linii
    g2.drawLine(0,0,iw-1,ih-1);//wrysuj linię do bufora
    JOptionPane.showMessageDialog(this, "Szer. obrazu="+iw+",
wys.="+ih,"Informacja",JOptionPane.INFORMATION_MESSAGE);
    setSize(iw,ih);
  }//BuforowanyObraz()
```

```
  public void paint(Graphics g){
            Graphics2D g2d=(Graphics2D)g;
            g2d.drawImage(bi,null,0,0);
  }//paint()
  public static void main(String a[]){
            SwingUtilities.invokeLater(new Runnable() {//interfejs
                public void run() {//metoda wątku
                        BuforowanyObraz so=new BuforowanyObraz("Demonstracja tworzenia
obiektu BufferedImage");
                        so.setVisible(true);
                }//run()
            });//invokeLater()
  }//main
}//class BuforowanyObraz
```

Obraz – klasa BufferedImage

Druga metoda uzyskania obiektu klasy BufferedImage to wykorzystanie biblioteki javax.imageio. Biblioteka ta daje możliwości odczytu i zapisu plików w formatach graficznych (JPEG, GIF, PNG, …). Najprościej z pliku można uzyskać obiekt klasy BufferedImage, a

```
(...) import javax.imageio.*;
//Read i write …

BufferedImage bi;
(…)

bi=ImageIO.read(new FileInputStream("motyl.jpg"));

ImageIO.write(bi, "GIF", new File("motyl.gif"));
```

Obraz – klasa BufferedImage

Wyświetlenie obrazu obiektu *BufferedImage* odbywa się poprzez wykorzystanie metod *drawImage*() zdefiniowanych dla klasy *Graphics2D* (np. *g2.drawImage*(*bi*,*null*,*null*);).

Java2D i klasa *BufferedImage* umożliwiają liczne korekcje graficzne, dodawanie elementów graficznych i metody przetwarzania obrazu. Korekcje graficzne i dodawanie elementów graficznych polega na rysowaniu w stworzonym kontekście graficznym dla obiektu *BufferedImage*. Przetwarzanie obrazu odbywa się głównie poprzez wykorzystanie klas implementujących interfejs *BufferedImageOp* a mianowicie: *AffineTransformOp*, *BandCombineOp*, *ColorConvertOp*, *ConvolveOp*, *LookupOp*, oraz *RescaleOp*.

Do najczęściej wykorzystywanych operacji należą przekształcenia geometryczne sztywne (afiniczne) - aplikacja *AffineTransformOp* - oraz

Język JAVA – obrazy cyfrowe


```
import java.awt.*; import java.awt.image.*; import javax.swing.*;
import javax.imageio.*;import java.io.*;
public class FiltrujObraz extends JFrame{
  BufferedImage bi;
  public FiltrujObraz(String txt){
    super(txt);
    try{
      bi=ImageIO.read(new FileInputStream("c:\\java\\kody_zrodlowe\\maly_motyl.jpg"));
    }catch (Exception e){System.out.println("BLAD "+e);}
    setSize(bi.getWidth(),bi.getHeight());
    bi=convertBIType();
    Kernel kernel = new Kernel(3, 3,  new float[] {   0, -1, 0, 0, 0, 0, 0, 1, 0 });
    ConvolveOp op = new ConvolveOp(kernel,ConvolveOp.EDGE_NO_OP, null);
    bi = op.filter(bi, null);
   }//FiltrujObraz()
  public BufferedImage convertBIType(){
    BufferedImage bnew;
    bnew = new BufferedImage(bi.getWidth(), bi.getHeight(), BufferedImage.TYPE_INT_RGB);
    Graphics2D g2 = bnew.createGraphics();
    g2.drawImage(bi, 0, 0, null);              bi=bnew;              return bi;
   }//convertBIType
```

```
public void paint(Graphics g){
        Graphics2D g2d=(Graphics2D)g;
        g2d.drawImage(bi,null,0,0);
}//paint()
public static void main(String a[]){
 SwingUtilities.invokeLater(new Runnable() {//interfejs
  public void run() {//metoda wątku
        FiltrujObraz so=new FiltrujObraz("Demonstracja filtracji");
        so.setVisible(true);
  }//run()
 );//invokeLater()
}//main
```

Obrazy – jak pobrać wartości pikseli?

1. Dla obiektu klasy Image img:

Należy zastosować klasę PixelGrabber:

```
int[] pixels = new int[w * h];
PixelGrabber pg = new PixelGrabber(img, x, y, w, h, pixels, 0, w);
(…) pg.grabPixels(); (…)
```

gdzie: *x, y*-współrzędne lewego górnego narożnika prostokąta skąd chcemy pobrać dane macierzy obrazu; *w, h* – szerokość i wysokość prostokąta, *0*-przesunięcie w docelowej tablicy danych (odkąd składujemy dane), *w* – liczba pikseli w wierszu.

2. Dla obiektu klasy BufferedImage bi:

-Jedną z metod bi.getRGB(x,y) – dostajemy wartość int (na kolejnych bajtach ARGB),

-Pobrać obiekt klasy Raster (bi.getData()), a później wiele możliwości, np. getPixel(), getSample(), itd.

DEMO - MPO

Plan wykładu:

1. Wprowadzenie do grafiki w Javie
2. Budowa GUI: komponenty, kontenery i układanie komponentów
3. Budowa GUI: obsługa zdarzeń
4. Grafika wektorowa – porysujmy sobie
5. Reprezentacja koloru
6. Grafika rastrowa - obrazy
7. Obsługa czcionek

Czcionki w Javie

W Javie można korzystać z różnych typów czcionek, które związane są z daną platformą na jakiej pracuje Maszyna Wirtualna. Dostęp do czcionek odbywa się poprzez trzy typy nazw: nazwy logiczne czcionek, nazwy czcionek, nazwy rodziny czcionek.
Nazwy logiczne czcionek to nazwy zdefiniowane dla Javy. Możliwe są następujące nazwy logiczne czcionek w Javie:
Dialog, DialogInput, Monospaced, Serif, SansSerif, oraz Symbol.

Nazwy logiczne są odwzorowywane na nazwy czcionek powiązane z czcionkami dla danego systemu. Odwzorowanie to występuje w pliku font.properties znajdującego się w katalogu lib podkatalogu jre. W pliku tym zdefiniowano również sposób kodowania znaków, dlatego definiowane są różne pliki font.properties z odpowiednim rozszerzeniem np. font.properties.pl (rozszerzenie zależy od lokalizacji – Locale – dla Polski „pl”).

Czcionki w Javie

Podstawowe klasy związane z czcionkami to:

1. Font – umożliwia zdefiniowanie czcionki, jaka ma być użyta w komponencie graficznym, np.

```
(...) Component c; (...)
Font f = new Font ("Arial", Font.PLAIN, 24);
c.setFont(f);
```

Gdzie: *"Arial"* – nazwa czcionki natywnej lub logicznej, *Font.PLAIN* – stała określająca krój (BOLD, ITALIC), *12*-rozmiar czcionki.

2. FontMetrics – klasa umożliwiająca uzyskanie informacji wymiarach czcionki/znaków w danej czcionce, np. metody getHight(), charWidth(), itp.

3. GraphicsEnvironment – pobranie parametrów konfiguracji, np. tablicy nazw dostępnych czcionek, tablicy dostępnych czcionek:

```
GraphicsEnvironment.getLocalGraphicsEnvironment().getAvailableFontFamilyNames();
GraphicsEnvironment.getLocalGraphicsEnvironment().getAllFonts();
```

Język JAVA – czcionki


```
import java.awt.*;  import java.awt.event.*;
public class Czcionki extends Frame  {
  public String s[];  private Font czcionki[];  private Font f;
  Czcionki (String nazwa){
    super(nazwa);  f = new Font("Verdana",Font.BOLD,12);
    s=GraphicsEnvironment.getLocalGraphicsEnvironment().getAvailableFontFamilyNames();
    czcionki=GraphicsEnvironment.getLocalGraphicsEnvironment().getAllFonts();
   }//Czcionki()
  public static void main(String args[]){
    Czcionki okno = new Czcionki("Lista czcionek");
    okno.setSize(600,500);    okno.setLayout(new GridLayout(2,1));
    List lista1 = new List(1, false);
    for (int i=0; i<okno.s.length; i++){         lista1.add(okno.s[i]);          }
    lista1.setFont(okno.f);                      okno.add(lista1);
    List lista2 = new List(1, false);
    for (int i=0; i<okno.czcionki.length; i++){   lista2.add(okno.czcionki[i].toString());          }
    lista2.setFont(okno.f);         okno.add(lista2);
    okno.addWindowListener(new WindowAdapter(){
     public void windowClosing(WindowEvent e){
       System.out.println("Dziekujemy za prace z programem...");   System.exit(0);  }    });
    okno.setVisible(true);         }//main()                }// koniec public class Czcionki
```

Co dalej?

Przedstawiona część wykładu dotycząca grafiki w Javie to wybrane przeze mnie elementy, które uważam za kluczowe (możliwości są dużo większe, np. Java3D, Java Media Framework, Java Speech/Sound API). Zainteresowanych odsyłam do zasobów Internetu oraz dedykowanych podręczników.

Teraz – zapraszam na kolejny wykład nr 6, w którym poznamy możliwości operacji wejścia/wyjścia, czyli coś o strumieniach.

Zapraszam na wykład 6