Język JAVA podstawy programowania

Jacek Rumiński

Wykład 5, część 2

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

Budowa GUI: obsługa zdarzeń

Oprócz wyświetlenia komponentu konieczne jest stworzenie odpowiedniego sterowania związanego z danym komponentem. Programista musi napisać kod związany z obsługą zdarzenia, np. wciśnięcia przycisku, ruchu myszki, itd.

Najprostszy model obsługi zdarzeń można przedstawić jako obsługę przerwań programowych:

- nazywamy przerwania (nazwy/identyfikatory),
- piszemy procedurę (kod) jaka ma zostać wykonana jeśli dane (nazwane) przerwanie się pojawi (powiązanie przerwanie-obsługa),
- działający system asynchronicznie odbiera przerwania, rozpoznaje je (po nazwach/identyfikatorach) i obsługuje (czyli wykonuje powiązaną procedurę, lub nic nie robi jeśli takiej procedury nie ma).

Ten uproszczony schemat można zrealizować w formie pętli nasłuchiwania zdarzeń i instrukcji switch(PRZERWANIE_ID) identyfikującej przerwania i wywołującej powiązaną procedurę.

Budowa GUI: obsługa zdarzeń

Podobny model obsługi zdarzeń wykorzystywany był w większości systemów i języków programowania (np. Win API). Również w pierwszych wersjach Javy wykorzystano taki model wprowadzając dwie metody action(Event zdarzenie, Object zrodlo) i handleEvent(). Nadpisując taką metodę można wywoływać różne funkcje obsługi zdarzeń w zależności od rodzaju (źródła) zdarzenia reprezentowanego przez obiekt klasy Object (switch lub if). Obiekt klasy Event dostarcza informacji o okolicznościach wystąpienia zdarzenia.

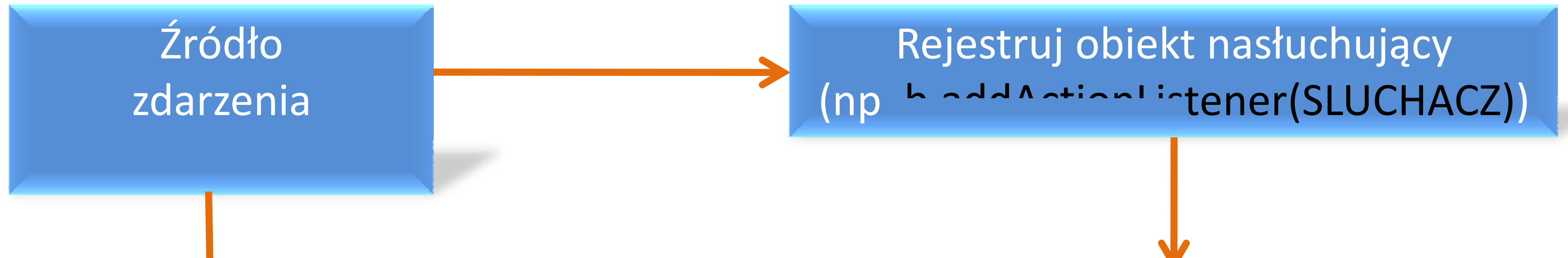
Niemniej bardzo szybko wprowadzono nowy model obsługi zdarzeń. Dlaczego? Otóż poprzedni model jest typowo proceduralny (strukturalny). Tworząc model obiektowy chcemy, żeby również obsługa zdarzeń związana była z obiektami, a metody obsługi zdarzeń mogły być wielokrotnie wykorzystywane tak, jak inne metody

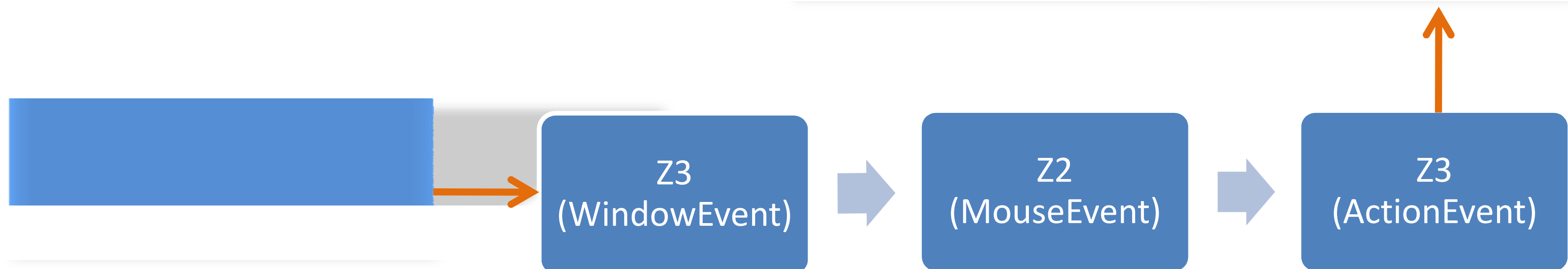
Budowa GUI: obsługa zdarzeń

Od wersji Javy 1.1 wprowadza się nowy system przekazywania i obsługi zdarzeń określający obiekty jako nasłuchujące zdarzeń (*listeners*) i jako generujące zdarzenia (*sources*). W tym nowym modelu obsługi zdarzeń komponent "odpala" ("*fire*") zdarzenie. Każdy typ zdarzenia jest reprezentowany przez określoną klasę.
W nowym systemie można nawet stworzyć własne typy zdarzeń.

Zdarzenie wygenerowane przez komponent jest odbierane przez właściwy element nasłuchu związany z komponentem generującym zdarzenie.
Jeżeli w ciele klasy nasłuchu danych zdarzeń istnieje metoda obsługi tego zdarzenia, to zostanie ona wykonana. Generalnie więc można rozdzielić źródło zdarzeń i obsługę zdarzeń.

Budowa GUI: obsługa zdarzeń

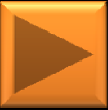




Język JAVA – GUI - zdarzenia


```
import java.awt.*; import javax.swing.*;
import java.awt.event.*; //Koniecznie należy pamiętać o importowaniu pakietu event
public class ZdarzeniaJedi extends JFrame {
  //JTextField jako pole obiektu, bo tf musi być widoczne w metodzie obsługi zdarzeń
  private JTextField tf;
  public void init() {
        JButton bJEDI, bSITH;
        tf = new JTextField(50);
        tf.setFont(new Font(Font.DIALOG, Font.BOLD, 20));
        add(tf, BorderLayout.NORTH);
        JPanel jp=new JPanel();
        bJEDI = new JButton("JEDI");
        bSITH = new JButton("SITH");
        jp.add(bJEDI);
        jp.add(Box.createRigidArea(new Dimension(50,0))); //wstaw przerwę
        jp.add(bSITH);
        add(jp,BorderLayout.CENTER);
        //dodajemy obiekty nasłuchujące zdarzenia
        bJEDI.addActionListener(new Postac("Jedi"));
        bSITH.addActionListener(new Postac("Sith"));
  } // koniec public void init()
```

c.d.n.



```
 public static void main(String []a){
             ZdarzeniaJedi zj=new ZdarzeniaJedi();
             zj.init();
             zj.setSize(500,150);
             zj.setVisible(true);
 } //koniec main()
/* Klasa wewnętrzna - ma dostęp do pól i metod klasy zewnętrznej;
  * implementuje interfejs ActionListener. Twórcy klasy JButton
  * przewidzieli obsługę zdarzenia przyciśnięcia przycisku poprzez
  * przygotowanie metody actionPerformed() w interfejsie ActionListener  */
 class Postac implements ActionListener {
             String rodzaj;
             Postac(String s){
                          rodzaj=s;
             } //koniec Postac()
  public void actionPerformed(ActionEvent e) {
   tf.setText(rodzaj);
  } //koniec actionPerformed()
 } // koniec class
}//koniec class ZdarzeniaJedi
```

Budowa GUI: obsługa zdarzeń

Realizując obsługę zdarzeń musimy rozważyć:

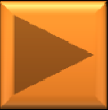
1. Jaki rodzaj zdarzenia może wystąpić – czyli obiekt jakiej klasy zostanie utworzony.
2. Jaki interfejs zdarzeń chcemy imlementować oraz jakie konkretne zdarzenia związane są z metodami w danym interfejsie. W dokumentacji Javy (klas Java API) możemy zobaczyć jaką metodę addXXXListener możemy wykorzystać dla danego komponentu. Znając nazwę interfejsu "nasłuchiwacza" (Listener) można uzyskać szczegółowy opis metod, np.:

KeyListener	**keyPressed(KeyEvent)**
KeyAdapter	**keyReleased(KeyEvent)**
	keyTyped(KeyEvent)
MouseListener	**mouseClicked(MouseEvent)**
MouseAdapter	**mouseEntered(MouseEvent)**
	mouseExited(MouseEvent)
	mousePressed(MouseEvent)
	mouseReleased(MouseEvent)

Język JAVA – GUI - zdarzenia


```
import java.awt.*;
import java.awt.event.*;
class Ekran extends Canvas{ //Canvas - pole graficzne
  public String s="Witam"; //początkowa wartość pola
            private Font f;
            Ekran (){
            super();    f = new Font("Times New Roman",Font.BOLD,16);//ustaw czcionkę
            setFont(f);
            addKeyListener(new KeyAdapter(){
                        public void keyPressed(KeyEvent ke){//wciśnięto przycisk klawiatury
                          s=new String(ke.paramString());//parametry zdarzenia jako String
                          repaint();//wywołaj metodę paint() - czyli wyświetl nową wartość s
                        }            });
            addMouseListener(new MouseAdapter(){
                        public void mousePressed(MouseEvent me){//wciśnięto przycisk myszki
                          s=new String(me.paramString());//parametry zdarzenia jako String
                          repaint();//wywołaj metodę paint() - czyli wyświetl nową wartość s
                        }            });
  }//koniec Ekran()
  public void paint(Graphics g){          g.drawString(s,10,180);          }//koniec paint()
}// koniec class Ekran
```

c.d.n.



```
public class KomunikatorJedi extends Frame  {
        KomunikatorJedi (String nazwa){
                super(nazwa);
        }//koniec KomunikatorJedi()
        public static void main(String args[]){
                KomunikatorJedi okno = new KomunikatorJedi("Komunikator");
                okno.setSize(1000,400);
                Ekran e = new Ekran();
                okno.add(e);
                okno.addWindowListener(new WindowAdapter(){
                        public void windowClosing(WindowEvent e){
                                System.out.println("Dziękujemy za pracę...");
                                System.exit(0);
                        }
                });
                okno.setVisible(true);
        }//koniec main()
}// koniec public class KomunikatorJedi
```

Budowa GUI: obsługa zdarzeń – trochę komplikacji

Obsługę zdarzeń dla środowiska graficznego (GUI) realizuje oddzielny wątek działający w tle (jeden z wątków maszyny wirtualnej). Program z "main()" to oddzielny i inny wątek. Żeby prawidłowo obsługiwać zdarzenia (realizować dostęp do komponentów) związane z GUI trzeba odwołać się do głównej kolejki zdarzeń (wątek dystrybucji zdarzeń, EDT-Event Dispatch Thread). Jeśli się tego nie zrobi, wówczas (szczególnie przy złożonych programach) może dojść do zakleszczenia (jeden wątek blokuje drugi, a drugi pierwszy - program "wisi"). Zatem każdy wątek (w tym main()), który odwołuje się do komponentów GUI trzeba powiązać z EDT (umieścić zadanie w kolejce wykonywanych czynności względem GUI).

Zadanie to realizuje statyczna metoda SwingUtilities.invokeLater(NASZ_WATEK).

Budowa GUI: obsługa zdarzeń – trochę komplikacji

KAŻDĄ aplikację graficzną powinniśmy wywoływać w sposób następujący:

```
public class KazdaAplikacja extends JFrame {
 public static void main(String[] args) {
  SwingUtilities.invokeLater(new Runnable() {//interfejs
   public void run() {//run to główna metoda wątku -main() dla wątku
        KazdaAplikacja ka= new KazdaAplikacja ();
        //ustawienia
        ka.setVisible(true);
   }
  });//koniec invokeLater()
 }//koniec main
 //tu konstruktor, metody i pola naszej klasy
}//koniec class KazdaAplikacja
```

Budowa GUI: obsługa zdarzeń – trochę komplikacji

Należy również pamiętać o tym, że jeśli chcemy zrealizować długotrwałe zadania, które jest wywoływane z GUI (np. wciskamy przycisk, a obsługa zdarzenia wywołuje metodę łamania hasła; metoda działa długo) to zablokujemy EDT, czyli nie mamy wówczas możliwości sterowania GUI (obsługa zdarzeń jest zawieszona, aż długie zadanie zakończy się…).

Każde takie długie zadanie należy wykonywać w oddzielnym wątku. Pomocna może być wówczas klasa SwingWorker.

Ale wątki to jeden z tematów objętych kursem dla bardziej zaawansowanych.

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

Pakiet **AWT** zarówno w wersjach wcześniejszych jak i w wersji 2 wyposażony jest w klasę Graphics, a od wersji 2 dodatkowo w klasę Graphics2D. Klasy te zawierają liczne metody umożliwiające tworzenie i zarządzanie grafiką w Javie. Podstawą pracy z grafiką jest tzw. kontekst graficzny, który jako obiekt posiada właściwości konkretnego systemu prezentacji np. panelu. W **AWT** kontekst graficzny jest dostarczany do komponentu poprzez następujące metody:

- paint,
- paintAll,
- update,
- print,
- printAll,

Język JAVA – grafika wektorowa

Obiekt graficzny (kontekst) zawiera informacje o stanie grafiki potrzebne dla podstawowych operacji wykonywanych przez metody Javy. Zaliczyć tu należy następujące informacje:

- obiekt komponentu, który będzie obsługiwany,
- współrzędne obszaru rysowania oraz obcinania,
- aktualny kolor,
- aktualne czcionki,
- aktualna funkcja operacji na pikselach logicznych (XOR lub Paint),
- aktualny kolor dla operacji XOR.

Posiadając obiekt graficzny można wykonać szereg operacji rysowania np.: Graphics g;

g.drawLine(int x1, int y1, int x2, int y2) - rysuje linię,

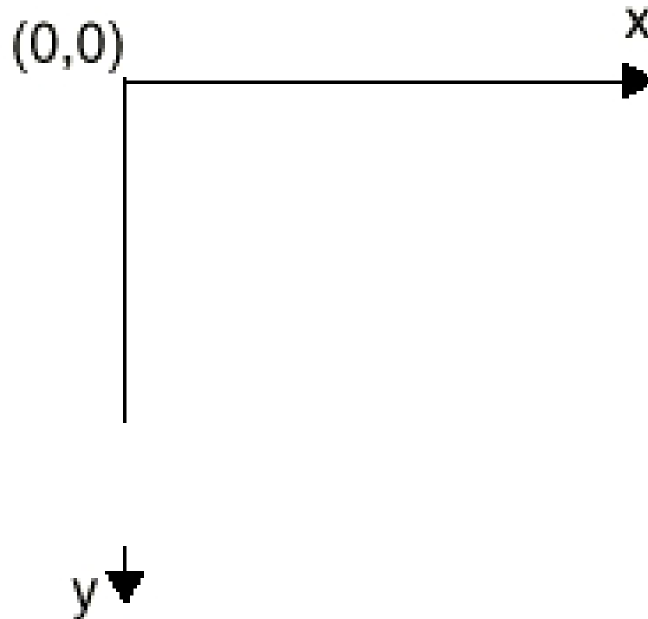
g.drawRect(int x, int y, int width, int height) - rysuje prostokąt,

g.drawString(String str, int x, int y) - rysuje tekst,

g.drawImage(Image img, int x, int y, Color bgcolor, ImageObserver observer) - wyświetla obraz

Język JAVA – grafika wektorowa

W celu rysowania elementów grafiki konieczna jest znajomość układu współrzędnych w ramach, którego wyznacza się współrzędne rysowania. Podstawowym układem współrzędnych w Javie jest układ użytkownika, będący pewną abstrakcją układów współrzędnych dla wszystkich możliwych urządzeń. Układ użytkownika definiowany jest w sposób następujący.



Pierwotne wersje AWT definiują kilka obiektów geometrii jak np. Point, Rectangle. Elementy te są bardzo przydatne dlatego, że, co jest właściwe dla języków obiektowych, nie definiujemy za każdym razem prostokąta za pomocą atrybutów opisowych (współrzędnych) lecz przez gotowy obiekt - prostokąt, dla którego znane są (różne metody) jego liczne właściwości:

```
Point p = new Point(x,y);
Rectangle r = new Rectangle(x,y,width,height);
```

Przykładowo metoda translate():

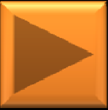
```
Insets insets = getInsets();
g.translate (insets.left, insets.top);
```

zmienia początek układu współrzędnych przesuwając go do aktywnego pola graficznego (bez ramek).

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

class PoleGraficzne extends Canvas{
    public void paint (Graphics g) {
        g.drawLine (5, 5, 195, 5);
        g.drawLine (5, 75, 5, 75);
        g.drawRect (25, 10, 50, 75);
        g.fillRect (25, 110, 50, 75);
        g.drawRoundRect (100, 10, 50, 75, 60, 50);
        g.fillRoundRect (100, 110, 50, 75, 60, 50);
        g.setColor(Color.red);
        g.drawString ("Test grafiki",50, 100);
        g.setColor(Color.black);
   }//koniec paint()
}//koniec class PoleGraficzne
```

Język JAVA – grafika wektorowa



```
public class RysunkiJedi extends JFrame {
    RysunkiJedi () {
        super ("Rysunki");
        add(new PoleGraficzne());
        setSize(200, 220);
    }//koniec RysunkiJedi()
    public static void main (String [] args) {
            SwingUtilities.invokeLater(new Runnable() {//interfejs
                    public void run() {//
                    RysunkiJedi r = new RysunkiJedi ();
                    r.setVisible(true);
                    r.addWindowListener(new WindowAdapter(){
                            public void windowClosing(WindowEvent e){
                            System.out.println("Dziekujemy za prace z programem...");
                            System.exit(0);
                            }
                    });
            }//koniec run()
      });//koniec invokeLater()
    }//koniec main()
}// koniec public class Rysunki extends Frame
```

Java2D API w sposób znaczny rozszerza możliwości graficzne **AWT**.
Po pierwsze umożliwia zarządzanie i rysowanie elementów graficznych o współrzędnych zmiennoprzecinkowych (float i double). Własność ta jest niezwykle przydatna dla różnych aplikacji m.in. systemów CAD. Ta podstawowa zmiana podejścia do rysowania obiektów graficznych i geometrycznych powoduje powstanie, nowych, licznych klas i metod.
W sposób szczególny należy wyróżnić tutaj sposób rysowania nowych elementów. Odbywa się to poprzez zastosowanie jednej metody:

```
Graphics2D g2;
g2.draw(Shape s);
```

Metoda draw() umożliwia narysowanie dowolnego obiektu implementującego interfejs Shape (kształt). Przykładowo narysowanie linii o współrzędnych typu float można wykonać w następujący sposób:

```
Line2D linia = new Line2D.Float(20.0f, 10.0f, 100.0f, 10.0f);
g2.draw(linia);
```

Oczywiście klasa Line2D implementuje interfejs Shape. **Java2D** wprowadza liczne klasy w ramach pakietu java.awt.geom, np:

Arc2D.Double
Arc2D.Float
CubicCurve2D.Double
CubicCurve2D.Float
Ellipse2D.Double
Ellipse2D.Float
Line2D
Line2D.Double
Line2D.Float
Point2D
Point2D.Double
Point2D.Float
Rectangle2D
itd.

W celu skorzystania z tych oraz innych dobrodziejstw jakie wprowadza **Java2D** należy skonstruować obiekt graficzny typu Graphics2D.
Ponieważ Graphics2D rozszerza klasę Graphics, to konstrukcja obiektu typu Graphics2D polega na:

```
Graphics2D g2 = (Graphics2D) g;
```

gdzie *g* jest obiektem graficznym otrzymywanym jak omówiono wyżej.
Uwaga! Argumentem metody paint() komponentów jest obiekt klasy Graphics, a nie Graphics2D.

Dodatkowe klasy w **AWT** wspomagające grafikę to BasicStroke oraz TexturePaint. Pierwsza z nich umożliwia stworzenie właściwości rysowanego obiektu takich jak np.: szerokość linii, typ linii.
Przykładowo ustawienie szerokości linii na 12 punktów odbywać się może poprzez zastosowanie następującego kodu:

```
grubaLinia = new BasicStroke(12.0f);
g2.setStroke(grubaLinia);
```

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

public class Rysunki2DJedi extends Frame {
    float[] dash = {3.0f,0.0f,3.0f};
    Rysunki2DJedi () {
        super ("Rysunki w Java2D");
        setSize(600, 600);
    }//koniec Rysunki2DJedi()
    public static void main (String [] args) {
        Frame f = new Rysunki2DJedi ();
        f.addWindowListener(new WindowAdapter(){
          public void windowClosing(WindowEvent e){
                System.out.println("Dziekujemy za prace z programem...");
                System.exit(0);
          }
        });
        f.setVisible(true);
    }//koniec main()
```

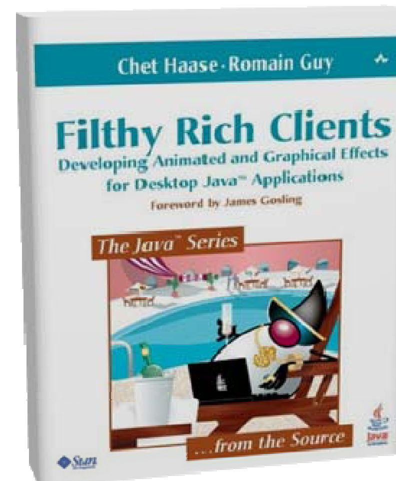
Język JAVA – grafika wektorowa

```
public void paint (Graphics g) {
    Graphics2D g2 = (Graphics2D) g;
    Insets insets = getInsets();
    g2.translate (insets.left, insets.top);
    System.out.println("Przesuniecie= "+insets.left+", "+insets.top);
    Ellipse2D glowa=new Ellipse2D.Float(20.0f, 20.0f, 580.0f-(insets.left+20.0f), 580.0f-
        (2*insets.top));
    BasicStroke grubaLinia = new BasicStroke(6.0f);
    g2.setStroke(grubaLinia);        g2.draw(glowa);
    g2.setColor(Color.blue);        g2.fillOval(175,150,50,50);        g2.fillOval(375,150,50,50);
    g2.setColor(Color.red);
    QuadCurve2D nos= new QuadCurve2D.Float(300, 220, 250, 300, 300,370);
    g2.draw(nos);
    BasicStroke kLinia = new
        BasicStroke(2f,BasicStroke.CAP_ROUND,BasicStroke.JOIN_ROUND,1f,dash,2f);
    g2.setStroke(kLinia);
    QuadCurve2D usta= new QuadCurve2D.Float(220, 420, 300, 490, 380,420);
    g2.draw(usta);
    usta= new QuadCurve2D.Float(220, 420, 300, 450, 380,420); g2.draw(usta);
}//koniec paint()
}// koniec public class Rysunki2DJedi extends Frame
```

Klasa TexturePaint umożliwia wypełnienie danego kształtu (Shape) określoną teksturą.

Do dodatkowych zalet grafiki w **Java2D** należy zaliczyć:
- sterowanie jakością grafiki (np. antyaliasing, interpolacje)
- sterowanie przekształceniami geometrycznymi (przekształcenia sztywne - afiniczne - klasa AffineTransform),
- sterowanie przeźroczystością elementów graficznych,
- bogate narzędzia do zarządzania czcionkami i rysowania tekstu,
-narzędzia do drukowania grafiki,
- i inne.

Zainteresowanym grafiką
i animacją polecam książkę
http://filthyrichclients.org/



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