Podstawy apletów

Co to jest aplet, cykl życia apletu, metody apletu, metoda paint(), czcionki, kolory, parametryzowanie apletu, układ współrzędnych ekranowych, klasa Graphics i jej metody.

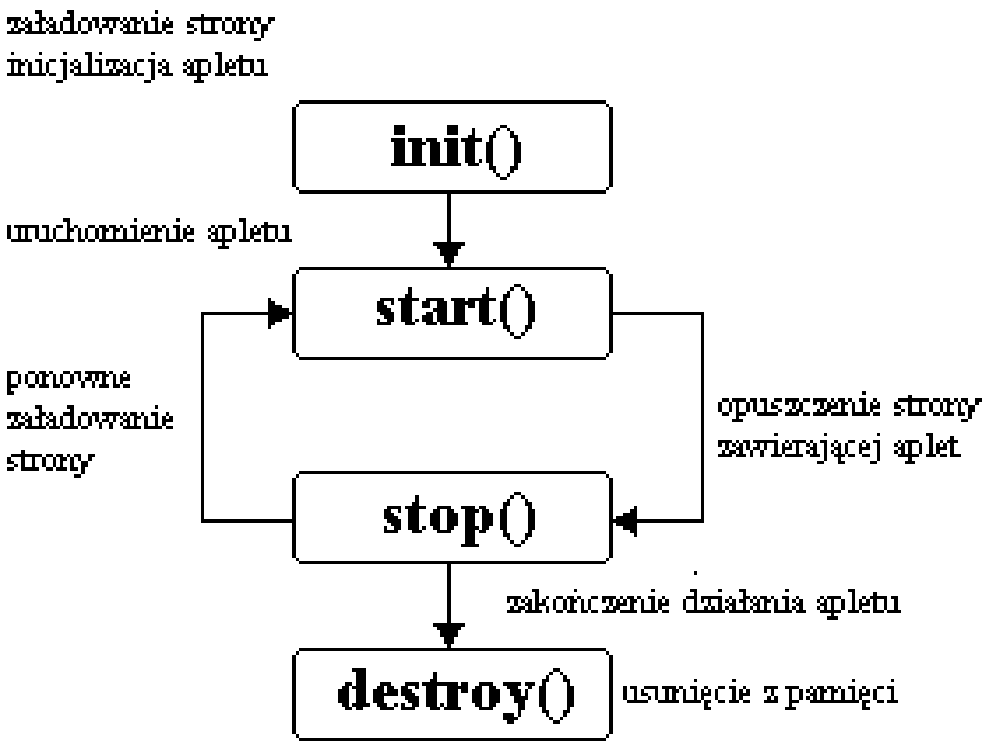
FUW

Co to jest aplet?

- **Aplet to obiekt Javy - wyposażony w graficzny interfejs i osadzony w dokumencie HTML.**
- **Klasy składające się na aplet są dynamicznie ładowane poprzez sieć i uruchamiane lokalnie przez przeglądarkę.**
- **Aplety w odróżnieniu od aplikacji nie posiadają metody main(). Główna klasa każdego apletu musi być klasą publiczną, dziedziczyć z predefiniowanej klasy java.applet.Applet, a jej nazwa musi być taka jak nazwa pliku, w którym się znajduje.**
- **Poprzez dziedziczenie aplet uzyskuje szereg metod, które są zdefiniowane w klasie nadrzędnej, a w szczególności:**
  - **możliwość komunikowania się z przeglądarką,**
  - **zdolność do obsługi graficznego interfejsu użytkownika.**

Cykl życia apletu

W odróżnieniu od aplikacji aplet nie zaczyna działania od metody main(). Cykl życia apletu wyznaczają cztery metody zdefiniowane w klasie Applet: init(), start(), stop() i destroy().



Metody apletu są zdefiniowane jako puste:

```
public void init()
{
}
```

Domyślnie metody te po prostu nic nie robią i jeśli nie są używane, to nie trzeba ich przedefiniowywać.

FUW

Metody apletu

- **Metoda init()**
  - Wykonywanie apletu zaczyna się od metody *init()*.
  - Metoda ta powinna być wywoływana tylko raz w ciągu całego życia apletu, zaraz po załadowaniu strony z apletem.
  - Umieszcza się w niej kod potrzebny do zainicjowania apletu, np. nadający zmiennym wartości początkowe, związany z ustawieniem czcionek, kolorów, koloru tła ekranu, itp.
- **Metoda start()**
  - Metoda ta jest wywoływana zaraz po powrocie z metody *init()*.
  - Używa się jej do uruchamiania wątków, animacji, wysyłania odpowiednich komunikatów do obiektów pomocniczych lub odtwarzania dźwięków związanych z załadowaniem apletu.
  - Metoda ta może być wielokrotnie wywoływana podczas cyklu życia apletu, na przykład przy każdym powrocie na stronę z apletem, gdy staje się ona stroną bieżącą w przeglądarce.
  - Zwykle jeśli wyposażymy aplet w metodę *start()* powinniśmy go wyposażyć również w metodę stop.

Metody apletu

- **Metoda stop()**
  - Metody tej zwykle używa się do zatrzymania wątków oraz do przeprowadzenia czynności porządkowych.
  - Metoda *stop()* jest wywoływana, gdy okno przeglądarki, w której znajduje się aplet, jest minimalizowane do ikony, zasłaniane przez inne okno, lub gdy otwierana jest w nim nowa strona WWW.
- **Metoda destroy()**
  - Metoda *destroy()* jest wywoływana, gdy aplet kończy swoje działanie – tuż przed usunięciem apletu z pamięci.
  - Jest ona rzadko używana, ponieważ Java posiada mechanizm automatycznego zwalniania nieużywanych zasobów pamięci.
  - Poprzez przedefiniowanie (*override*) metody *destroy()* można zrealizować zatrzymanie dowolnego wątku lub usunięcie obiektów utworzonych przez aplet.

Metoda paint()

Metoda paint() jest zdefiniowana w java.awt.Component, która jest klasą nadrzędną dla klasy java.awt.Applet. Metoda ta służy do wyświetlania tekstu i grafiki na aplecie.

```
public void paint(Graphics g)
{
      g.drawString("Hello World", 20, 20);
}
```

Jedynym parametrem metody paint() jest referencja do obiektu klasy Graphics, który reprezentuje tzw. graficzny kontekst urządzenia (Graphics Device Context).

Graficzny kontekst jest swoistym urządzeniem wyjściowym apletu, tworzonym przez przeglądarkę. Jest on przekazywany do metody paint() wtedy, gdy zachodzi potrzeba pisania lub rysowania na wyjściu apletu (ekran, bufor pamięci).

Czcionki

Pismo jest obiektem klasy Font, której konstruktor wymaga podania trzech parametrów: nazwy kroju pisma, stylu i rozmiaru czcionki, np.

```
public void paint(Graphics gDC)
{
    Font font = new Font("Helvetica", Font.PLAIN, 16);
    gDC.setFont(font);
    gDC.drawString("Hello", 20, 20);
}
```

Standardowe czcionki:

- Helvetica
- Courier
- Dialog - domyślna
- DialogInput
- TimesRoman
- ZapfDingBats

Styl czcionek:

- Font.BOLD
- Font.PLAIN
- Font.ITALIC

Rozmiar:

- 9-48 punktów

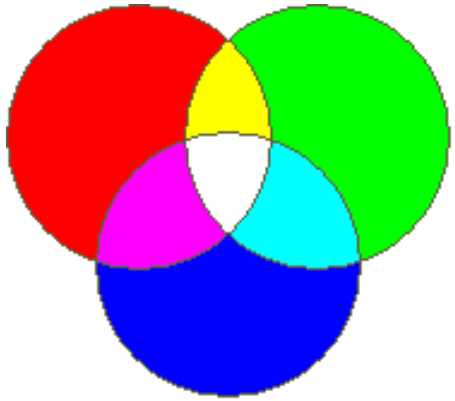
Kolory

Kolor jest obiektem klasy Color. Instrukcja tworzenia koloru ma postać:

```
Color nazwa_koloru = new (r, g, b);
```

Parametry r, g, b są liczbami całkowitymi z zakresu 0..255 i reprezentują składowe RGB (R-red, G-green, B-blue) definiowanego koloru (teoretycznie można otrzymać w ten sposób 24 miliony kolorów).

```
public void paint(Graphics gDC)
{
   Color c = new Color(128, 20,80);
   gDC.setColor(c);
   gDC.drawString("Hello", 20, 20);
}
```



Do ustawienia koloru można też użyć następującej konstrukcji:

```
   gDC.setColor(new Color(128, 20,80));
```

Predefiniowane kolory

Klasa Color zawiera kilkanaście standardowych wartości kolorów. Są one zdefiniowane jako stałe klasowe static final Color - zatem można je używać bez tworzenia obiektów klasy Color. Poniższy fragment kodu ustawia kolor tła na pomarańczowy oraz kolor pisma na purpurowy:

```
public void paint(Graphics gDC)
{
  setBackground(Color.orange);
  gDC.setColor(Color.magenta);
  gDC.drawString("Hello", 20, 20);
}
```

Nazwy kolorów

- black
- blue
- cyan
- darkGray
- gray
- green
- lightGray
- magenta
- orange
- pink
- red
- white
- yellow

Parametryzowanie apletu

Poprzez sparametryzowanie apletu można zmieniać jego wygląd i sposób działania już na poziomie HTML. Parametry można przekazywać ze strony HTML do apletu za pomocą specjalnych znaczników <PARAM> umieszczonych wewnątrz znacznika <APPLET>.

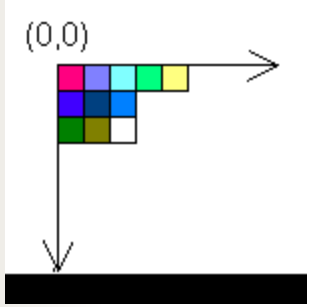
```
<applet code="Napis.class" width="200" height="200">
<param name="text" value="Hello World">
<param name="size" value="24">
</applet>
```

Do odczytu parametru służy metoda getParameter()

```
String napis = getParameter("text");
if ( napis == null )  napis = "Hello";
```

- **Metoda getParameter() zawsze zwraca String. Jeśli jest to liczba, należy dokonać odpowiedniej konwersji.**
- **W przypadku, kiedy parametr nie jest określony w pliku HTML, przyjmowana jest wartość domyślna.**

Układ współrzędnych ekranowych



Obszar apletu można postrzegać jako macierz pikseli, z których każdy może wyświetlać jeden lub więcej kolorów.

- **Początek układu współrzędnych ekranowych (0,0) znajduje się w lewym górnym rogu okna apletu.**
- **Zwiększanie wartości współrzędnych następuje w miarę przesuwania się w prawo i w dół.**
- **Maksymalne wartości współrzędnych wynoszą:**
  - **int mx = getSize().width;**
  - **int my = getSize().height;**

Klasa Graphics

- Klasa Graphics z pakietu **java.awt** - jest klasą bazową dla wszystkich graficznych kontekstów urządzenia. Jej metody umożliwiają:
  - uzyskiwanie i ustalanie własności kontekstu graficznego
  - wypisywanie tekstów,
  - rysowanie linii, prostokątów, owali, łuków i wielokątów,
  - wyświetlanie obrazów - obiektów klasy Image.

```
public void paint(Graphics g)
{
     g.setFont(new Font("Courier", Font.PLAIN, 24));
     g.setColor(Color.red);
     g.drawString("Hello", 20, 20);
 }
```

Metody klasy Graphics

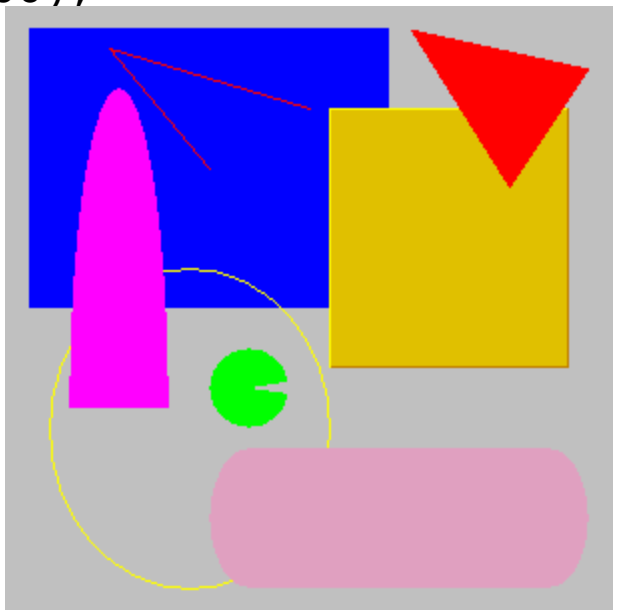
Metoda	Opis
drawLine(int x1, int y1, int x2, int y2)	Linia prosta: (x1, y1) – punkt początkowy, (x2, y2) - końcowy
drawRect (int x, int y, int w, int h)	Prostokąt: x, y - współrzędne lewego górnego rogu prostokąta; w, h - szerokość i wysokość
fillRect (int x, int y, int w, int h)	Prostokąt wypełniony bieżącym kolorem.
drawOval(int x, int y, int w, int h)	Owal (wewnątrz prostokąta)
drawOval(int x, int y, int w, int h	Owal wypełniony kolorem.
draw3DRect(int x, int y, int w, int h, boolean b)	Prostokąt trójwymiarowy - wypukły, gdy b=true.
fill3DRect(int x, int y, int w, int h, boolean b)	Prostokąt trójwymiarowy wypełniony kolorem.

Metody klasy Graphics

Metoda	Opis
drawArc(**int x, int y, int w, int h, int alfa, int beta**)	Łuk pokrywający określony prostokąt; alfa – kąt określający początek łuku, beta – rozmiar kąta.
fillArc(**int x, int y, int w, int h, int alfa, int beta**)	Łuk wypełniony bieżącym kolorem.
drawPolygon(int[] x, int[] y, int n)	Wielokąt: tablice punktów x i y określają współrzędne kolejnych wierzchołków wielokąta, n - to liczba punktów.
fillPolygon(int[] x, int[] y, int n)	Wielokąt wypełniony kolorem.
drawPolyline(int x[], int y[], int n)	Linia łamana - otwarta

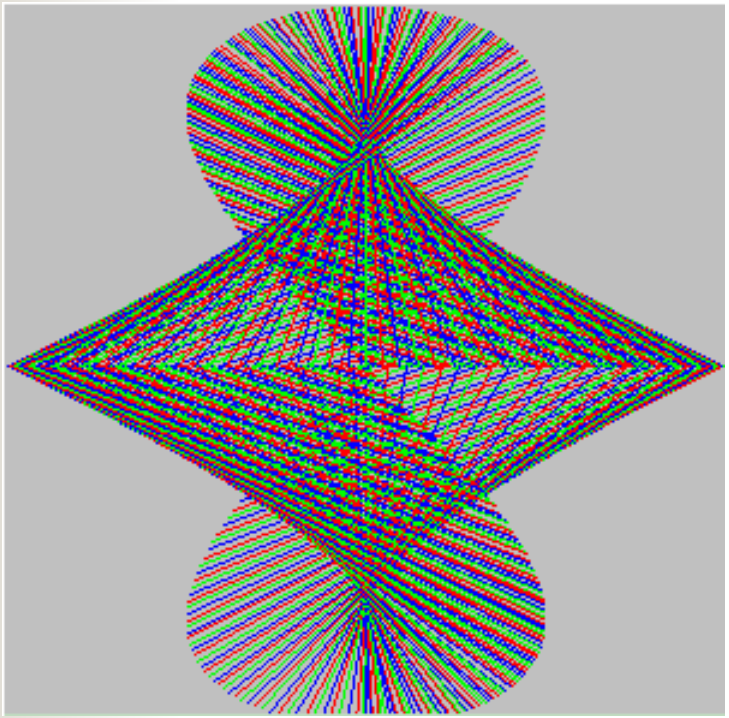
Prezentacja kształtów

```
public void paint(Graphics g)
{
 g.setColor(Color.blue);
 g.fillRect(10, 10, 180, 140);
 g.setColor(Color.orange);
 g.fill3DRect(160, 50, 120, 130, true);
 g.drawOval(20, 130, 140, 160);
 g.fillRoundRect(100, 220, 190, 70, 40, 90);
 g.fillArc(30, 40, 50, 320, 0, 180);
 g.fillArc(100, 170, 40, 40, 10, 340);
   int[] x = {200,250, 290};
   int[] y = {10,90, 30};
 g.fillPolygon(x, y, 3);
   int[] a = {100,50, 150};
   int[] b = {80,20, 50};
 g.drawPolyline(a, b, 3);
}
```



Przykład - serwetka

Korzystając z metody drawLine() można tworzyć w prosty sposób ciekawą grafikę. Wystarczy odpowiednio dobrać wzory określające współrzędne punktów końcowych rysowanych linii.



Cykliczną zmianę kolorów kolejnych linii można uzyskać, pisząc:

```
if (i%3==1)
  gDC.setColor(Color.red);
else if (i%3==2)
  gDC.setColor(Color.green);
else
  gDC.setColor(Color.blue);
```

Kod źródłowy serwetki

```
public void paint(Graphics gDC)
{
  int x1,y1,x2,y2;
  int xs = getSize().width / 2;
  int ys = getSize().height / 2;
  for (int i=0; i<400; i++)
  {
    x1=(int)( xs*Math.sin(i/40.0)*Math.cos(i/40.0));
    y1=(int)( xs*Math.cos(i/40.0));
    x2=(int)(-2*xs*Math.sin(i/40.0)*Math.cos(i/40.0));
    y2=(int)(-xs/400*Math.cos(i/40.0));
    gDC.drawLine(x1+xs, y1+ys, x2+xs, y2+ys);
  }
}
```

Przykład - dywan

Tworzenie dywanów ma charakter eksperymentalny. Tajemnica uzyskania ciekawego wzoru tkwi w dobraniu postaci funkcji, która wylicza kolor na podstawie współrzędnych rysowanego punktu.

Algorytm rysowania dywanu korzysta z dwóch pętli. Zewnętrzna pętla wyznacza wartości kolejnych współrzędnych poziomych; dla każdej z nich pętla wewnętrzna rysuje od góry do dołu kolejne punkty. Kolor punktu jest obliczany za pomocą funkcji, których argumentami są wyrażenia zależne od położenia punktu.

Kod źródłowy dywanu

```
public void paint(Graphics gDC)
{
    int mx = getSize().width, my = getSize().height;
    int r, g, b;
    for ( i=5; i < mx-5; i++)
```

Przykład – interferencja fal

Sumowanie fal o zbliżonych amplitudach i o zbliżonych długościach pozwala uzyskać charakterystyczny rysunek tzw. dudnień.

```
import java.awt.*;
import java.applet.*;
public class Drgania extends Applet
{
   int width, height;  // rozmiary apletu
   public void init()
   {
      width = getSize().width;
      height=getSize().height;
      setSize(width, height);
      setBackground(Color.orange);
   }
```

interferencja fal

```
public void paint(Graphics g)
{
    int x, xp, yp, y, ys, y1, y2;
    double A = (double) height/3;
    double f =36;
    ys = height/2;
    double T1 = 1.14*width, T2 = width;
    xp = 0; yp = ys; g.setColor(Color.blue);
    for (x=0; x<=width; x++)
    {
      y1=ys-(int)(A*Math.sin(f*Math.PI/T1*x));
      y2=ys-(int)(A*Math.sin(f*Math.PI/T2*x));
      y =(y1+y2)/2;
      g.drawLine(xp, yp, x, y);
      xp = x;  yp = y;
    }
 }
}
```

Zadanie 1

Wykonaj aplet rysujący trójkąt wypełniony szarym kolorem. Trójkąt powinien stopniowo rozjaśniać się aż do osiągnięcia koloru białego.

Wskazówki

1. W kolorze szarym składowe r, g, b mają taką samą wartość: r=80, g=80 i b=80.
2. Aby efekt był zauważalny, należy spowolnić działanie apletu, np. pisząc pętlę opóźniającą ;

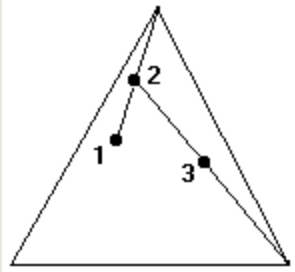
Rozwiązanie

```
import java.awt.*;
import java.applet.*;

public class Trojkat extends Applet
{
     public void paint(Graphics g)
     {
          int x[] = {60, 130, 220};
          int y[] = {240,50,200};
          for (int i=80; i<256; i++)
          {
             g.setColor(new Color(i, i,
i));
             g.fillPolygon(x, y, 3);
             for (int j=0; j<1e6; j++);
          }
     }
}
```

Zadanie 2

Zadaniem naszym będzie wykonanie apletu tworzącego trójkąt Sierpińskiego.



Trójkąt Sierpińskiego powstaje jako zbiór punktów generowanych w wyniku błądzenia przypadkowego wewnątrz trójkąta.

Algorytm postępowania jest prosty. Błądzenie rozpoczynamy w dowolnym punkcie P_1 wewnątrz trójkąta. Jest to tzw. punkt wiodący. W każdym kolejnym ruchu losujemy jeden z wierzchołków trójkąta i w środku odcinka łączącego ten wierzchołek z punktem wiodącym umieszczamy nowy punkt wiodący.

FUW

Rozwiązanie

```
public void paint(Graphics g)
{
    int[]x = {5, 150, 295};
    int[]y = {230, 7, 230};
    int w;
    int a = (int)(5+Math.random()*290);  //P1
    int b = (int)(7+Math.random()*223);
    for (double k=0; k<1e5; k++)
    {
       w = (int)(Math.random()*3);
       a = (a+x[w])/2;
       b = (b+y[w])/2;
       g.drawLine(a, b, a, b);
     }
}
```

Zadanie 3

Wykonaj aplet rysujący wielokąt foremny o n wierzchołkach, wpisany w okrąg o promieniu r i środku S(xs, ys).

Wskazówki

Wierzchołki wielokąta foremnego są równomiernie rozłożone na okręgu, a ich współrzędne spełniają następujące równania parametryczne:

$$x = xs + r\cos\alpha, \quad y = ys + r\sin\alpha$$

gdzie α oznacza kąt łączący

FUW

Rozwiązanie

```
public void paint(Graphics g)
{
    int r=100;
    int n=8;
    int xs = 150, ys = 150;
    int xp=xs+r, yp=ys;
    double alfa = (double) 2*Math.PI/n;
    int x, y;
    for (int i=1; i<=n; ++i)
    {
      x = xs + (int)(r*Math.cos(i*alfa));
      y = ys - (int)(r*Math.sin(i*alfa));
      g.drawLine(xp, yp, x, y);
      xp = x; yp = y;
    }
}
```

Modyfikacja 1

Zmodyfikuj poprzedni aplet tak, aby uzyskać prostą animację: losowana jest liczba wierzchołków n (n>=3), a następnie aplet jest odświeżany.

Wskazówki

1. Instrukcja generująca liczbę n może mieć postać:

   ```
   int n = 3 + (int) (12*Math.random());
   ```

2. Do odrysowania apletu używa się metody repaint(). Ma ona kilka konstruktorów. Najczęściej jest używana jako metoda bezparametrowa. Może też pobierać czas (w ms), po którym aplet ma być odświeżony - np. repaint (500). Można również określić wymiary prostokąta, który ma zostać odświeżony - np. repaint(50, 50, 150, 150).

Rozwiązanie

```
public void paint(Graphics g)
{
    int x, y, r=100;
    int n=3+(int)(12*Math.random());
    int xs = 150, ys = 150;
    int xp=xs+r, yp=ys;
    double alfa = (double) 2*Math.PI/n;
    for (int i=1; i<=n; ++i)
    {
      x = xs + (int)(r*Math.cos(i*alfa));
      y = ys - (int)(r*Math.sin(i*alfa));
      g.drawLine(xp, yp, x, y);
      xp = x; yp = y;
    }
    repaint(500);
}
```

Modyfikacja 2

Zmodyfikuj poprzedni aplet tak, aby oprócz wielokąta rysowane były jego przekątne.

Wskazówki

1. Utwórz dwuwymiarową tablicę, w której zapamiętywane będą wierzchołki wielokąta - np.

```
   int[][] W = new int[MAX][2];
gdzie MAX - maksymalna liczba wierzchołków
```

2. W pętli od k=0 do k= n-1 wypełnij tablicę wierzchołków

```
 W[k][0] = xs+(int)(r*Math.cos(k*alfa));
 W[k][1] = ys-(int)(r*Math.sin(k*alfa));
```

Rozwiązanie - metoda paint()

```
final int MAX=50; //maks. liczba wierzch.
int[][] W = new int[MAX][2];
int r=100, n=3+(int)(12*Math.random());
int xs = 150, ys = 150;
double alfa = (double) 2*Math.PI/n;
for (int k=0; k<=n; ++k)
{
   W[k][0] = xs + (int)(r*Math.cos(k*alfa));
   W[k][1] = ys - (int)(r*Math.sin(k*alfa));
}
for (int i=0; i<n; ++i)
for (int j=i+1; j<n; ++j)
  g.drawLine(W[i][0],W[i][1],W[j][0],W[j][1]);
repaint(500);
```

Modyfikacja 3

Wprowadź do apletu kolory - kontur wielokąta ma być rysowany kolorem czerwonym, zaś jego przekątne kolorem niebieskim.

Wystarczy przed rysowaniem linii umieścić instrukcję

```
if ((j-i==1)||(i==0)&&(j==n-1))
{
   g.setColor(Color.red);
}
else
{
   g.setColor(Color.blue);
}
```