Grafika rastrowa i wektorowa w Javie
(podstawowe elementy API Java 2D)

1. Informacje ogólne (grafika rastrowa/wektorowa)

Java 2D API jest rozszerzeniem pakietu AWT (Abstract Windowing Toolkit), umożliwiającym tworzenie grafiki dwuwymiarowej – rysowanie obiektów geometrycznych, prostych, tekstów oraz przetwarzanie obrazów rastrowych (wspierane są formaty graficzne takie jak: *bmp*, *jpg*, *gif* itd.). Jedne z podstawowych klas Java 2D API, to klasy abstrakcyjne *Graphics* i *Graphics2D*.

java.awt.Graphics:

Klasa ta implementuje dostęp do kontekstu graficznego komponentów/obiektów graficznych Javy, umożliwiając tworzenie grafiki. Ogólnie mechanizm Javy rysowania na komponentach graficznych (czyli takich, które umożliwiają wyświetlanie grafiki) takich jak: JApplet, JFrame, JPanel i innych (obecnie używane są komponenty biblioteki *Swing*), polega na przesłonięciu metod: paintComponent() lub paint(). Obie metody należy wywoływać z argumentem typu *Graphics*. Obiekt *graphics* jest tzw. 'płótnem', którego zawartość wyświetlana jest na ekranie monitora. Obiekt ten przekazywany jest przez JVM (nie można go samemu zdobyć – jednak można wymusić ponowne wywołanie tych metod poprzez uruchomienie metody *repaint()*).

java.awt.Graphics2D:

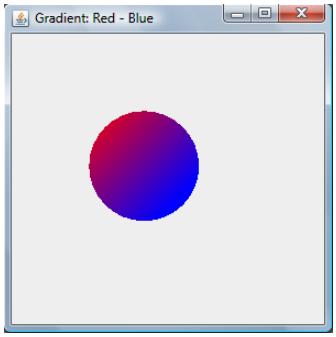
W rzeczywistości obiekt przekazywany metodom *paintComponent()* czy *paint()* jest obiektem klasy *Graphics2D*. Klasa ta jest rozszerzeniem klasy *Graphics* (tzn. dziedziczy po niej) udostępniającym większe możliwości tworzenia grafiki. Aby z skorzystać z tych możliwości należy rzutować obiekt *Graphics* na *Graphics2D*.

Jako przykład poniżej przedstawiono kod prostego programu, rysującego prosty obiekt geometryczny, przy wykorzystaniu gradientu (technika umożliwiająca płynne przejście pomiędzy dwoma kolorami – w przykładzie czerwonym i niebieskim).

```
import java.awt.Color;
import java.awt.GradientPaint;
import java.awt.Graphics;
import java.awt.Graphics2D;
import javax.swing.JFrame;
import javax.swing.JPanel;


public class Gradient {
	JFrame ramka;

	public static void main(String[] args) {
		Gradient grad = new Gradient();
		grad.GUI();
	}
```

```
        public void GUI(){

                ramka = new JFrame();
                ramka.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                ramka.setTitle("Gradient: Red - Blue");

                MojPanelRysunkowy panelR  = new MojPanelRysunkowy();

                ramka.getContentPane().add(panelR);
                ramka.setSize(300, 300);
                ramka.setVisible(true);
        }
}

class MojPanelRysunkowy extends JPanel{
        public void paintComponent(Graphics g){
                super.paintComponent(g);

                Graphics2D g2d = (Graphics2D) g;
                GradientPaint gradient = new GradientPaint(70, 70,
Color.RED, 150, 150, Color.BLUE);
                g2d.setPaint(gradient);
                g2d.fillOval(70, 70, 100, 100);
        }
}
```

Efekt uruchomienia programu:



Zwróć uwagę na: użyte pakiety i ich klasy, sposób w jaki dodawany jest komponent *JPanel* do ramki, cel wykorzystania obiektu *Graphics2D* i sposób pozyskania tego obiektu.

Grafikę wektorową w Java 2D można tworzyć przy użyciu pakietu '*java.awt.geom*'. W pakiecie tym zdefiniowane są takie klasy jak: *Line2D*, *Rectangle2D*, *RoundRectangle2D*, *Arc2D*, czy też *Ellipse2D*. Każdy z tych 'prymitywów' geometrycznych może być rysowany w różnym trybie (takim jak na przykład tryb XOR). Tryb rysowania można ustawić za pomocą obiektu *Graphics2D*, natomiast o charakterystykę (wygląd) samego rysowanego obiektu, można zadbać poprzez użycie metody *setStroke()* oraz obiektu klasy *BasicStroke*.

Poniższy kod prezentuje przykład użycia trybu rysowania XOR, w stosunku do obiektu *Rectangle2D*.

"…

```
g2d.setXORMode( new Color( 255, 255, 255 ));
        g2d.setStroke( new BasicStroke( 5 ) );
  Rectangle2D.Double  rectangle = new Rectangle2D.Double(40, 40, 60, 90);
  g2d.draw( rectangle );

  g2d.draw( rectangle );
```

…"

Pytanie: Jaki będzie efekt wykonania tego kodu ?; Co nam daje rysowanie w trybie XOR ?

Oprócz podstawowych kształtów geometrycznych, Java 2D oferuje obiekty umożliwiające rysowanie krzywych parametrycznych (*QuadCurve2D* lub *CubicCurve2D*).

W przypadku grafiki rastrowej (przetwarzanie obrazów), należy zwrócić uwagę na dwie podstawowe klasy, służące do obsługi obrazów. Mianowicie *java.awt.Image* i *java.awt.image.BufferedImage* .

java.awt.Image:
Jest to klasa nadrzędna dla wszystkich klas, które przeznaczone są do realizacji operacji na obrazach rastrowych. Zakłada się reprezentację obrazu jako macierz elementów (pikseli).

java.awt.image.BufferedImage:
Klasa ta jest rozszerzeniem klasy *Image*, umożliwiająca reprezentację i przetwarzanie informacji obrazowej w pamięci komputera. *BufferedImage* reprezentuje obraz poprzez udostępnienie bufora zawierającego dane obrazowe (tzn. informacji o lokalizacji pikseli i ich koloru – przeważnie używany jest model kolorów RGB). Obraz przygotowywany (przetwarzany) jest w obiekcie *BufferedImage*, który następnie wyświetla już gotowy obraz w określonym komponencie graficznym Javy. *BufferedImage* jest również obiektem typu *Image*, a więc można go używać we wszystkich metodach klas *Graphics* i *Graphics2D*, które przyjmują jako argument obiekt typu *Image* (na przykład metoda *drawImage()*).

Poniżej przedstawiono prosty przykład użycia obiektu *BufferedImage* (kod demonstruje konwersję obrazu w ‘skalę szarości’, poprzez zmianę wartości kolorów pikseli).

„
…

```
BufferedImage  image;
image = new BufferedImage( 300, 300, BufferedImage.TYPE_INT_RGB);
```

…

```
int  color;
byte gray;
int  i, j;
int  height = image.getHeight();
int  width  = image.getWidth();
for ( i = 0; i < height; i++)
   for ( j = 0; j < width; j++)
   {
        gray = (byte)(j % 256);
     color = byte2RGB( gray, gray, gray );
        image.setRGB( j, i, color );
   }

g2d.drawImage( image, 60, 60, null );
```

… „ (*g2d jest obiektem typu Graphics2D*)

Zwróć uwagę na: wywołanie konstruktora klasy *BufferedImage* (przekazane argumenty), zmianę wartości koloru piksela na pozycji *j, i*.

Dodatkowo, zastanów się nad: Jak uzyskać dostęp do poszczególnych składowych koloru modelu RGB ?, Jak użyć przeciążoną wersję metody setRGB() ?

Poniżej przedstawiono listę metod (pomijając argumenty, które należy przekazać do tych metod) biblioteki AWT, które mogą przydać się do ćwiczenia 1:

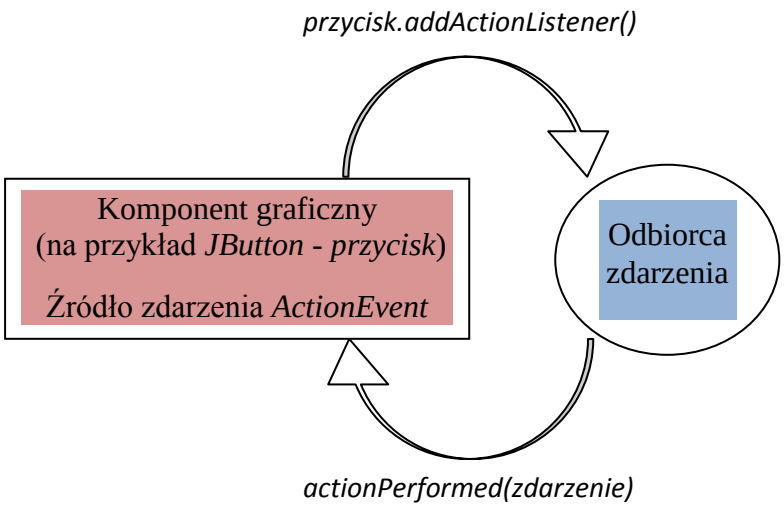
Dla obiektów klas *Graphics*, *Graphics2D*:

draw(prymityw geometryczny) - rysowanie obiektów geometrycznych,
drawString() – rysowanie stringów,
setPaint() – ustawienie koloru rysowania, wypełniania,
setXORMode() – ustawienie trybu XOR,
drawImage() – rysowanie obrazu,
setPaintMode() – ustawienie trybu rysowania na domyślny ('normalny'),
setColor() – ustawienie koloru rysowania,
fill() – wypełnianie kształtu,
Dla obiektu reprezentującego obraz rastrowy:
setRGB() – ustawienie wartości koloru pikseli
itd.

Więcej informacji można znaleźć w dokumentacji JavaSE 6 (API Specification), adres: **http://java.sun.com/javase/6/docs/api/** (lub w nowszej wersji)

2. Interakcja z użytkownikiem (obsługa zdarzeń w Javie)

W Javie proces zdobywania i obsługi zdarzeń generowanych przez użytkownika nazywany jest *obsługa zdarzeń*. Istnieje wiele różnych typów zdarzeń, jednak większość z nich związana jest z operacjami wykonywanymi przez użytkownika na elementach graficznego interfejsu użytkownika (przeważnie komponenty biblioteki Swing). W dużym uogólnieniu obsługa zdarzeń w Javie odbywa się tak jak pokazano na rysunku poniżej.



Źródło zdarzeń	**Odbiorca**
Komponenty graficzne Javy są tzw. źródłami zdarzeń. Na przykład (powyżej) przycisk jest źródłem zdarzeń *ActionEvent*. W klasie *JButton* jest zaimplementowana metoda *addActionListener()*, którą zainteresowane obiekty odbiorem zdarzenia, generowanego przez przycisk muszą użyć. Poprzez metodę *addActionListener()*, można dodać (zarejestrować) do listy zarejestrowanych odbiorców zdarzenia generowanego przez przycisk, kolejnego odbiorcę tego zdarzenia. Kiedy użytkownik wykona akcję (‘kliknie przycisk’), ten ‘zgłasza’ zdarzenie, wywołując metody *actionPerformed()* wszystkich odbiorców zarejestrowanych na liście.	Odbiorcą zdarzenia może być dowolny obiekt. Jednak aby obiekt mógł ‘reagować’ na zdarzenia *ActionEvent*, musi zostać również obiektem nasłuchującym akcje. Czyli jego klasa powinna implementować **interfejs** *ActionListener*. Implementacja tego interfejsu wiąże się z koniecznością przesłonięcia wszystkich metod tego interfejsu – czyli w tym przypadku jedyną metodę tego interfejsu – *actionPerformed()*. Oprócz tego obiekt ten musi się ‘zarejestrować’ na liście obiektów zainteresowanych akcją przycisku. ‘Rejestracja’ polega na wywołaniu metody *addActionListener()* przycisku, przekazując obiekt (*do obiektów odwołujemy się w Javie poprzez referencje*) jako parametr tej metody. Metoda ta ‘przyjmie’ każdy obiekt, którego klasa implementuje interfejs *ActionListener*. Sposób reakcji na zdarzenie implementuje się w metodzie *actionPerformed()*. Dodatkowo do tej metody jako argument przesyłany jest również obiekt (typu *ActionEvent*), który reprezentuje zdarzenie (przechowuje dane o zdarzeniu).

Sprawa jest bardzo podobna w przypadku obsługi zdarzeń generowanych przez ‘mysz’ (wynika to z tego, że model obsługi zdarzeń w Javie powstał na bazie jednego z wzorców projektowych -‘*obserwator*’).

Wystarczy aby odbiorcy zdarzeń implementowali odpowiednie interfejsy (na przykład *MouseListener* lub *MouseMotionListener*) i odpowiednio przesłonili metody tych interfejsów (w zależności od tego jak chcą zareagować na zdarzenie – na przykład *mouseClicked(), mousePressed(), mouseDragged()* itd.). Informacje o zdarzeniu przechowywane są w obiekcie *MouseEvent*, który jest przekazywany jako argument do powyższych metod. Można wykorzystać ten obiekt na przykład do odczytu bieżących wartości współrzędnych wskaźnika myszy.

Ćwiczenie 1

Zaprojektować graficzny interfejs użytkownika (GUI), za pomocą którego przy wykorzystaniu wybranych operacji Java 2D dla grafiki rastrowej i wektorowej, umożliwi się użytkownikowi interakcyjne wycinanie wybranych fragmentów obrazu rastrowego. Należy umożliwić wycinanie obrazów przy wykorzystaniu prostych figur geometrycznych poprzez obsługę zdarzeń generowanych przez mysz.

Projekt GUI powinien zawierać co najmniej poniższe interakcje z użytkownikiem:

- zaznaczanie dowolnych fragmentów obrazu przy wykorzystaniu prostych form geometrycznych, takich jak: elipsa, prostokąt, dowolny zamknięty wielokąt,
- poza polem, w którym załadowany zostanie obraz rastrowy, powinna być widoczna tabela z informacją o wszystkich segmentach i możliwością ich zaznaczenia - wizualizacja w postaci JList/JTable z własnym Rendererem (będzie przykład na wykładzie),
- należy zapewnić poprawną skalowalność interfejsu poprzez użycie odpowiednich 'menedżerów układu' (ang. Layouts),
- zapis i odczyt zaznaczonych fragmentów do pliku (XML lub format tekstowy),
- możliwość usuwania segmentów (po zaznaczeniu w tabeli).

Punkty dodatkowe:

- możliwość edycji (modyfikacji bieżącego zaznaczenia - np. rozszerzenie wykorzystanego prostokąta) segmentów (+2 dodatkowe punkty).

Uwaga:
Za brak opcji zaznaczenia poprzez dowolny wielokąt: -1 punkt,
Za brak tabeli: -1 punkt,
Za brak możliwości usuwania: -1 punkt,
Za brak zapisu: -1 punkt.

Przykładowe praktyczne zastosowanie - np. w diagnostyce medycznej:

Bardzo często w diagnostyce medycznej zachodzi problem przygotowania danych medycznych do analizy. Lekarze podejmują często subiektywne decyzje w odniesieniu do szeregu różnych czynników jak również i własnego doświadczenia. Jednym z aktualnie problematycznych zagadnień, jest przygotowanie 'wiarygodnego' zbioru danych medycznych w postaci zbioru komórek nowotworowych, pobranych z biopsji raka piersi. W ramach tzw. testu immunohistochemicznego, gotowe preparaty IHC poddawane są akwizycji cyfrowej przy wykorzystaniu dostępnych technologii mikroskopowych. Następnie, obrazy histopatologiczne przedstawiające konglomeracje komórek nowotworowych oceniane są przez eksperta patologa, celem określenia stopnia nasilenia ekspresji receptora HER-2/neu, warunkującego stopień agresywności nowotworu. Generalnie, istotne są tu dwie kategorie komórek: HER-2 nadekspresyjne i HER-2 nie nadekspresyjne.

A więc, przykładowym celem wprowadzenia opisanego GUI w ćwiczeniu nr 1, może stanowić wygenerowanie dużego zbioru danych histopatologicznych, przez różnych histopatologów. Wiarygodność danych może zostać określona poprzez wykorzystanie uznanych współczynniki zgodności - np. współczynniki Kappa (Cohena, Scotta).