**Basics**

In this part of the Java 2D games tutorial, we will write about some basics needed to create games. We create a basic application, paint a donut, and display a picture.

**About**

This is Java 2D games tutorial. It is aimed at beginners. This tutorial will teach you the basics of programming 2D games in Java programming language and Swing GUI toolkit. Images used in this tutorial can be downloaded [here](http://zetcode.com/img/gfx/javagames/images.zip).

**Application**

We will show the skeleton of each of the games in this tutorial.

Board.java

package com.zetcode;

import javax.swing.JPanel;

public class Board extends JPanel {

public Board() {}

}

The Board is a panel where the game takes place.

Application.java

package com.zetcode;

import java.awt.EventQueue;

import javax.swing.JFrame;

public class Application extends JFrame {

public Application() {

initUI();

}

private void initUI() {

add(new Board());

setSize(250, 200);

setTitle("Application");

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setLocationRelativeTo(null);

}

public static void main(String[] args) {

EventQueue.invokeLater(new Runnable() {

@Override

public void run() {

Application ex = new Application();

ex.setVisible(true);

}

});

}

}

This is the entry point of the game. Here we have the main method.

add(new Board());

Here we put the Board to the center of the JFrame container.

setSize(250, 200);

This line sets the size of the window.

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

This will close the application when we click on the close button. It is not the default behaviour.

setLocationRelativeTo(null);

Passing null to the setLocationRelativeTo() method centers the window on the screen.

public static void main(String[] args) {

EventQueue.invokeLater(new Runnable() {

@Override

public void run() {

Application ex = new Application();

ex.setVisible(true);

}

});

}

We create an instance of our code example and make it visible on the screen.

**Donut**

The objects on the board are either images or are drawn with the painting tools provided by the Java 2D API. In the next example, we draw a donut shape.

Board.java

package com.zetcode;

import java.awt.BasicStroke;

import java.awt.Color;

import java.awt.Dimension;

import java.awt.Graphics;

import java.awt.Graphics2D;

import java.awt.RenderingHints;

import java.awt.geom.AffineTransform;

import java.awt.geom.Ellipse2D;

import javax.swing.JPanel;

public class Board extends JPanel {

@Override

public void paintComponent(Graphics g) {

super.paintComponent(g);

drawDonut(g);

}

private void drawDonut(Graphics g) {

Graphics2D g2d = (Graphics2D) g;

RenderingHints rh

= new RenderingHints(RenderingHints.KEY\_ANTIALIASING,

RenderingHints.VALUE\_ANTIALIAS\_ON);

rh.put(RenderingHints.KEY\_RENDERING,

RenderingHints.VALUE\_RENDER\_QUALITY);

g2d.setRenderingHints(rh);

Dimension size = getSize();

double w = size.getWidth();

double h = size.getHeight();

Ellipse2D e = new Ellipse2D.Double(0, 0, 80, 130);

g2d.setStroke(new BasicStroke(1));

g2d.setColor(Color.gray);

for (double deg = 0; deg < 360; deg += 5) {

AffineTransform at

= AffineTransform.getTranslateInstance(w/2, h/2);

at.rotate(Math.toRadians(deg));

g2d.draw(at.createTransformedShape(e));

}

}

}

The painting is done inside the paintComponent() method.

Graphics2D g2d = (Graphics2D) g;

The Graphics2D class extends the Graphics class. It provides more sophisticated control over geometry, coordinate transformations, colour management, and text layout.

private void drawDonut(Graphics g) {

...

}

It is a good programming practice to delegate the actual painting to a specific method.

RenderingHints rh

= new RenderingHints(RenderingHints.KEY\_ANTIALIASING,

RenderingHints.VALUE\_ANTIALIAS\_ON);

rh.put(RenderingHints.KEY\_RENDERING,

RenderingHints.VALUE\_RENDER\_QUALITY);

g2d.setRenderingHints(rh);

The rendering hints are used to make the drawing smooth.

Dimension size = getSize();

double w = size.getWidth();

double h = size.getHeight();

We get the height and the width of the window. We need them to center the donut shape on the window.

Ellipse2D e = new Ellipse2D.Double(0, 0, 80, 130);

g2d.setStroke(new BasicStroke(1));

g2d.setColor(Color.gray);

Here we create the ellipse.

for (double deg = 0; deg < 360; deg += 5) {

AffineTransform at

= AffineTransform.getTranslateInstance(w/2, h/2);

at.rotate(Math.toRadians(deg));

g2d.draw(at.createTransformedShape(e));

}

Here the ellipse is rotated 72 times to create a donut shape.

Donut.java

package com.zetcode;

import java.awt.EventQueue;

import javax.swing.JFrame;

public class DonutExample extends JFrame {

public DonutExample() {

initUI();

}

private void initUI() {

add(new Board());

setSize(330, 330);

setTitle("Donut");

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setLocationRelativeTo(null);

}

public static void main(String[] args) {

EventQueue.invokeLater(new Runnable() {

@Override

public void run() {

DonutExample ex = new DonutExample();

ex.setVisible(true);

}

});

}

}

This is the main class.

**Drawing an image**

When we create computer games we often work with images. In the next example we load an image and paint it on the board. If you cannot locate the image file, have a look at the [Displaying image in Java tutorial](http://zetcode.com/java/displayimage/).

Board.java

package com.zetcode;

import java.awt.Dimension;

import java.awt.Graphics;

import java.awt.Image;

import javax.swing.ImageIcon;

import javax.swing.JPanel;

public class Board extends JPanel {

private Image bardejov;

public Board() {

initBoard();

}

private void initBoard() {

loadImage();

int w = bardejov.getWidth(this);

int h = bardejov.getHeight(this);

setPreferredSize(new Dimension(w, h));

}

private void loadImage() {

ImageIcon ii = new ImageIcon("bardejov.png");

bardejov = ii.getImage();

}

@Override

public void paintComponent(Graphics g) {

g.drawImage(bardejov, 0, 0, null);

}

}

We pain an image of a town on the board. The image is drawn inside the paintComponent() method.

ImageIcon ii = new ImageIcon("bardejov.png");

We create an ImageIcon.

bardejov = ii.getImage();

We get an Image out of the ImageIcon.

g.drawImage(bardejov, 0, 0, null);

We draw the image on the window.

int w = bardejov.getWidth(this);

int h = bardejov.getHeight(this);

setPreferredSize(new Dimension(w, h));

We determine the width and height of the image. The preferred size of the board panel is set to the dimensions of the image. In cooperation with the JFrame's pack() method, the window is just big enough to show the image.

Image.java

package com.zetcode;

import java.awt.EventQueue;

import javax.swing.JFrame;

public class ImageExample extends JFrame {

public ImageExample() {

initUI();

}

private void initUI() {

add(new Board());

pack();

setTitle("Bardejov");

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setLocationRelativeTo(null);

}

public static void main(String[] args) {

EventQueue.invokeLater(new Runnable() {

@Override

public void run() {

ImageExample ex = new ImageExample();

ex.setVisible(true);

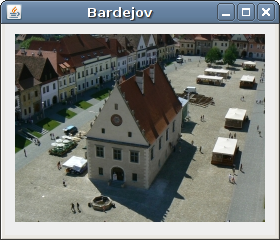
}

});

}

}

This is the main class of the example.

 Figure: Image