Поток-демон (из-за вызова метода **sleep(10000)**) не успел завершить выполнение своего кода до завершения основного потока приложения, связанного с методом **main()**. Базовое свойство потоков-демонов заключается в возможности основного потока приложения завершить выполнение потока-демона (в отличие от обычных потоков) с окончанием кода метода **main()**, не обращая внимания на то, что поток-демон еще работает. Если уменьшать время задержки потока-демона, то он может успеть завершить свое выполнение до окончания работы основного потока.

Потоки в графических приложениях

Добавить анимацию в апплет можно при использовании потоков. Поток, ассоциированный с апплетом, следует запускать тогда, когда апплет становится видимым, и останавливать при сворачивании браузера. В этом случае метод **repaint()** обновляет экран, в то время как программа выполняется. Поток создает анимационный эффект повторением вызова метода **paint()** и отображением графики в новой позиции.

```
/* пример # 7 : освобождение ресурсов апплетом: GraphicThreadsDemo.java */
package chapt14;
import java.awt.Color;
import java.awt.Container;
import java.awt.Graphics;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;

public class GraphicThreadsDemo extends JFrame {
      JPanel panel = new JPanel();
      Graphics g;
      JButton btn = new JButton("Добавить шарик");
      int i;

      public GraphicThreadsDemo() {
            setBounds(100, 200, 270, 350);
            Container contentPane = getContentPane();
            contentPane.setLayout(null);
            btn.setBounds(50, 10, 160, 20);
            contentPane.add(btn);
            panel.setBounds(30, 40, 200, 200);
            panel.setBackground(Color.WHITE);
            contentPane.add(panel);
            btn.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent ev) {
                        new BallThread(panel).start();
                        i++;
                        repaint();
                  }
```

```
            });
      }
      public static void main(String[] args) {
            GraphicThreadsDemo frame =
                        new GraphicThreadsDemo();
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.setVisible(true);
      }
      public void paint(Graphics g){
            super.paint(g);
         g.drawString("Количество шариков: " + i, 65, 300);
      }
}
class BallThread extends Thread {
      JPanel panel;
      private int posX, posY;
      private final int BALL_SIZE = 10;
      private double alpha;
      private int SPEED = 4;

      BallThread(JPanel p) {
            this.panel = p;
            //задание начальной позиции и направления шарика
            posX = (int)((panel.getWidth() - BALL_SIZE)
                                    * Math.random());
            posY = (int)((panel.getHeight() - BALL_SIZE)
                                    * Math.random());
            alpha = Math.random() * 10;
      }
      public void run() {
            while(true) {
                  posX += (int)(SPEED * Math.cos(alpha));
                  posY += (int)(SPEED * Math.sin(alpha));
            //вычисление угла отражения
            if( posX >= panel.getWidth() - BALL_SIZE )
                  alpha = alpha + Math.PI - 2 * alpha;
            else if( posX <= 0 )
                        alpha = Math.PI - alpha;
            if( posY >= panel.getHeight() - BALL_SIZE )
                        alpha = -alpha;
            else if( posY <= 0 )
                        alpha = -alpha;
                  paint(panel.getGraphics());
            }
      }
      public void paint(Graphics g) {
      //прорисовка шарика
      g.setColor(Color.BLACK);
```

```
        g.fillArc(posX, posY, BALL_SIZE, BALL_SIZE, 0, 360);
        g.setColor(Color.WHITE);
        g.drawArc(posX + 1, posY + 1, BALL_SIZE,
                                BALL_SIZE, 120, 30);
              try {
                    sleep(30);
              } catch(InterruptedException e) {
                    e.printStackTrace();
              }
        //удаление шарика
        g.setColor(panel.getBackground());
        g.fillArc(posX, posY, BALL_SIZE, BALL_SIZE, 0, 360);
        }
}
```

Рис.14.2.Потоки в апплетах

При вызове метода **`stop()`** апплета поток перестает существовать, так как ссылка на него устанавливается в **`null`** и освобождает ресурсы. Для следующего запуска потока необходимо вновь инициализировать ссылку и вызвать метод **`start()`** потока.

Методы synchronized

Очень часто возникает ситуация, когда несколько потоков, обращающихся к некоторому общему ресурсу, начинают мешать друг другу; более того, они могут повредить этот общий ресурс. Например, когда два потока записывают информацию в файл/объект/поток. Для предотвращения такой ситуации может использоваться ключевое слово **`synchronized`**. Синхронизации не требуют только атомарные процессы по записи/чтению, не превышающие по объему 32 бит.