1 Простое приложение с отрисовкой

В рамках данного задания познакомимся с тем, как с использованием готовых библиотек можно что-нибудь визуализировать на экране. Далее по ходу курса пойдут более сложные и объёмные задачи. Пока же просто создадим оконное приложение и порисуем на нём с помощью фреймворка.

Использовать для примеров в этой главе будем фреймворк Swing (Java). Он достаточно прост в ознакомлении и к тому же прекрасно подходит для текущей задачи.

Разберём нарочито примитивный пример. Заодно повторим некоторые вещи, которые студенты уже должны знать, если доучились до курса по КГ. Код из примеров можно найти в проекте по ссылке:

https://github.com/kv1tr4vn/CGVSU/tree/main/Task1/SimpleDrawingApp1

Начнем с того, что создадим файл *Main.java* с методом main(), а также пустой класс DrawingPanel. Напомним, что для создания экземпляров классов при объектно-ориентированном подходе используются конструкторы. В нашем случае он пока будет пустым, но это не помешает вызвать его внутри метода main():

```
1  class DrawingPanel {
2      // empty constructor
3      public DrawingPanel() {
4      }
5  }
6
7  public class Main {
8      public static void main(String[] args) {
9          // constructor call
10         new DrawingPanel();
11     }
12 }
```

Если все настроено правильно, после запуска можно будет увидеть надпись:

```
Process finished with exit code 0
```

Метод main() после своей работы вернул код 0. Это означает, что приложение отработало без ошибок.

Теперь превратим класс DrawingPanel в окошко, которое откроется при вызове конструктора. К счастью, сложную логику для этого писать не придётся. Мы унаследуем класс от библиотечного JFrame и вызовем его методы для настройки в конструкторе.

Будущий класс для отрисовки должен выглядеть так:

```
1  import javax.swing.*;
2
```

```
3  class DrawingPanel extends JFrame  {
4      private final int BACKGROUND_WIDTH = 800;
5      private final int BACKGROUND_HEIGHT = 800;
6
7      public DrawingPanel() {
8          setTitle("Drawing panel");
9          setSize(BACKGROUND_WIDTH, BACKGROUND_HEIGHT);
10         setVisible(true);
11         setDefaultCloseOperation(DISPOSE_ON_CLOSE);
12     }
13 }
```

Здесь на 8-й строчке задается надпись вверху окна, метод setSize() устанавливает его размер. 11-я строка нужна для того, чтобы завершать процесс при закрытии приложения.

Теперь при запуске откроется будущий холст. Часть задачи решена, осталось научиться на нем рисовать.

Для того, чтобы получить окно приложения, мы воспользовались наследованием. Вы, вероятно, помните, что таким образом можно в классе-потомке обращаться к части функционала класса-родителя. Но теперь нам требуется, чтобы потомок был не просто точным клоном своего отца, но и при наличии папиных глазок умел что-то делать иначе. Так, мы планируем переопределить метод paint(), отвечающий за то, что конкретно рисуется в окне. У JFrame он, конечно, останется старым, а вот у нашего класса будет реализовывать другую логику. Некоторые студенты здесь могут вспомнить, что такая возможность давать одноименным вещам разные реализации часто называется полиморфизмом.

Переопределение метода может выглядеть так:

```
1      @Override
2      public void paint(Graphics g) {
3          Graphics2D g2d = (Graphics2D) g;
4
5          g2d.setColor(Color.BLACK);
6          g2d.drawOval(200, BACKGROUND_HEIGHT / 2,
             BACKGROUND_WIDTH / 10, BACKGROUND_WIDTH / 10);
7          g2d.setColor(Color.GREEN);
8          g2d.fillOval(200, BACKGROUND_HEIGHT / 2,
             BACKGROUND_WIDTH / 10, BACKGROUND_WIDTH / 10);
9      }
```

Разберёмся, что тут происходит. Для начала обратите внимание на аннотацию *@Override* на первой строке. Ее использование является правилом хорошего тона и обычно прописано в стиле кода. Несмотря на то, что переопределение сработает

и без таких подписей, аннотация не только улучшит читаемость, но и спасет от багов в некоторых ситуациях. Допустите опечатку в названии paint() из примера и посмотрите, что произойдет.

А внутри самого метода здесь описывается отрисовка фигур. Для этого сначала экземпляр класса Graphics приводится к типу Graphics2D, так как у последнего больше возможностей. А затем для рисования контура эллипса вызывается метод drawOval() и для его заполнения — fillOval(). Метод setColor() и прочие, настраивающие кисть, распространяют результат своей работы на весь код ниже до тех пор, пока не будет новая смена настроек.

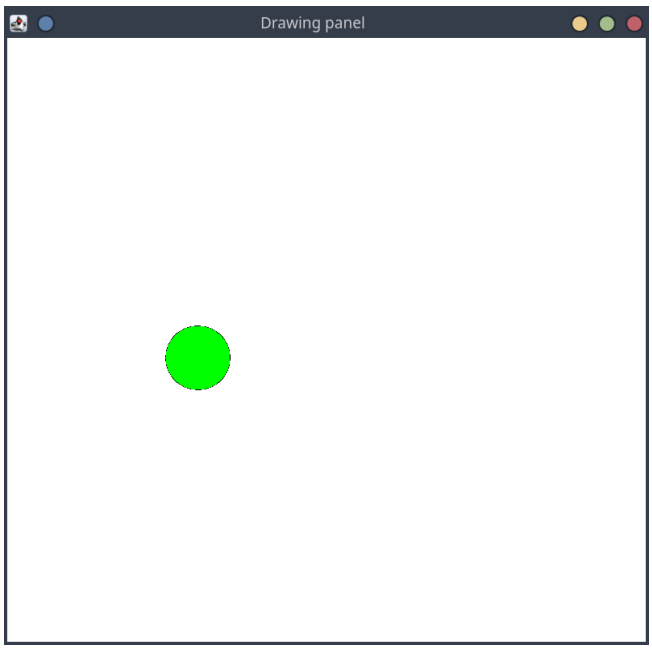

Рис. 1: Окно с отрисовкой

Таким образом мы сообщаем фреймворку, что хотим отрисовать зеленый круг с черной обводкой по заданным координатам. Результат работы программы изображён на рисунке 1.

Graphics2D позволяет гораздо больше, чем рисование кружков. Прямоугольники, прямые и кривые линии, дуги, надписи, различные настройки кисти. Об этом много информации в сети. Начать можно, например, с документации:

https://docs.oracle.com/javase/7/docs/api/java/awt/Graphics2D.html

Также могут помочь чьи-нибудь готовые примеры.

Итак, мы смогли нарисовать фигуру в открытом окошке. Попробуем сделать проект более интересным и заставим круг двигаться. Код теперь будет выглядеть примерно так:

```
1 import javax.swing.*;
2 import java.awt.*;
3 import java.awt.event.ActionEvent;
4 import java.awt.event.ActionListener;
5
6 class DrawingPanel extends JFrame implements
    ActionListener {
7     private final int BACKGROUND_WIDTH = 800;
8     private final int BACKGROUND_HEIGHT = 800;
```

```
9       private final int TIMER_DELAY = 1000;
10      private final Timer timer = new Timer(TIMER_DELAY,
          this);
11      private int ticksFromStart = 0;
12
13      public DrawingPanel() {
14          setTitle("Drawing panel");
15          setSize(BACKGROUND_WIDTH, BACKGROUND_HEIGHT);
16          setVisible(true);
17          setDefaultCloseOperation(DISPOSE_ON_CLOSE);
18          timer.start();
19      }
20
21      @Override
22      public void paint(Graphics g) {
23          Graphics2D g2d = (Graphics2D) g;
24
25          g2d.setColor(Color.WHITE);
26          g2d.fillRect(0, 0, BACKGROUND_WIDTH,
              BACKGROUND_HEIGHT);
27
28          g2d.setColor(Color.BLACK);
29          g2d.drawOval(ticksFromStart, BACKGROUND_HEIGHT /
              2, BACKGROUND_WIDTH / 10, BACKGROUND_WIDTH / 10)
              ;
30          g2d.setColor(Color.GREEN);
31          g2d.fillOval(ticksFromStart, BACKGROUND_HEIGHT /
              2, BACKGROUND_WIDTH / 10, BACKGROUND_WIDTH / 10)
              ;
32      }
33
34      @Override
35      public void actionPerformed(ActionEvent e) {
36          if (e.getSource() == timer) {
37              repaint();
38              ++ticksFromStart;
39          }
40      }
41  }
```

Разберемся, что здесь происходит. Помимо того, что класс уже наследуется от JFrame (6-я строка), также добавляются слова *implements ActionListener*. Здесь

речь идет уже не о наследовании, а о реализации интерфейса - класса, который не может иметь своих экземляров и нужен лишь для того, чтобы определить в коде возможное поведение. С точки зрения теории у интерфейса нет полей, нет реализации методов. В нашем случае он нужен для того, чтобы указать, что DrawingPanel будет иметь метод actionPerformed().

Причем, если в случае с наследованием от JFrame мы могли и не переопределять метод paint(), потому что он есть у родителя, то вот метод интерфейса мы определить обязаны, ведь у ActionListener его реализации нет.

Сам метод нам нужен для того, чтобы ловить события (ActionEvent) от таймера, который будет посылать их раз в секунду. Количество таких событий (тиков) будем записывать в переменную ticksFromStart, а после использовать при перерисовке изображения для обновления координат фигуры.

После запуска приложения, которое теперь пришло в полное соответствие с проектом из репозитория, можно увидеть, как зеленый круг отрисуется, а после начнет медленно двигаться вправо.

Полученное приложение очень примитивное и имеет кучу проблем с точки зрения программирования. Сейчас весь код свален вместе, его сложно читать и еще сложнее вести дальнейшую разработку. Перед началом работы над собственной задачей посмотрите также проект:

https://github.com/kv1tr4vn/CGVSU/tree/main/Task1/SimpleDrawingApp2

Второе приложение рисует фигурку потяжелее и выносит часть функционала в отдельный класс. Также полотно DrawingPanel теперь является не самим окошком, а виджетом на нем, что позволит легко докидывать кнопочки, меню и другие элементы в будущем. Код все еще далек от идеала. Например, не помешало бы вынести куда-нибудь константы, используемые при рисовании. О подобных вещах вам теперь предстоит задуматься самостоятельно. Здесь проведена первичная работа по улучшению, поэтому на этот проект желательно опираться при выполнении своей задачи.

Условие задачи

Отталкиваясь от разобранных примеров, создайте приложение с осмысленной картинкой, используя готовую библиотеку. Важно, чтобы это был не абстрактный набор квадратиков, а что-то разумное и цельное. Необходимо включить в приложение большинство примитивов, таких как:

- ◇ Прямые и кривые линии;
- ◇ Прямоугольники, эллипсы (контуры и заполнения);
- ◇ Дуги;
- ◇ Текстовые надписи;
- ◇ др.

Не стоит забывать и про возможность рисовать различными цветами, стилями.

Код тоже должен быть осмысленным, ведь его написание — такой же творческий процесс, как и рисование. Красота внутри приложения повлияет на оценку.

Создайте грамотную архитектуру, не лепите все вызовы примитивов в кучу. Рисунок состоит из нескольких составляющих, которые следует разносить по смысловым блокам, методам. Используйте также условные операторы, операторы цикла, константы и все прочее, что позволит сделать код более читаемым.

Не стесняйтесь создавать отдельные классы для каких-то моделей с вашего холста: машин, домиков, солнышка и пр. Это не только улучшит читаемость, но и позволит легко использовать какую-то сущность несколько раз, с разными настройками.

Желательно, чтобы рисунок был не просто фигурой на белом холсте, но оказался как-то вписан в контекст, фон. Например, если делаете машину, добавьте дорогу, облачка, птиц и т.д.

Возможные варианты задач:

- ◇ Транспортные средства (автомобиль, самолёт, ж/д поезд, речное или морское судно);
- ◇ Пейзажи, натюрморты и прочие жанры живописи. Правда, и тут лучше избегать абстракционизма;
- ◇ Различные устройства, интерьеры, экстерьеры;
- ◇ Герои фильмов, мультфильмов, книг, игр;
- ◇ Любые другие осмысленные цензурные рисунки, удовлетворяющие требованиям законодательства РФ и другим нормативным документам.

Можно ограничиться и статическим изображением. Но с использованием разумной анимации или каким-то интерактивом: взаимодействием с помощью мыши, клавиатуры, оценка будет выше. Некоторые студенты умудряются на базе этого упражнения делать простенькие игры.

Это первое ознакомительное задание в курсе, так что не стоит ради него убиваться. Но, конечно, сложность выбранной вами темы повлияет на оценку. Если сомневаетесь, лучше обсудить свой вариант с преподавателем по практике.