Лабораторная работа №7
«Программа с графическим пользовательским интерфейсом»

Скоробогатов С.Ю.

14 апреля 2016 г.

1 Цель работы

Приобретение навыков разработки программ с графическим пользовательским интерфейсом на основе библиотеки swing.

2 Исходные данные

Продемонстрируем создание приложения с графическим пользовательским интерфейсом в IntelliJ IDEA на примере простой программы, рисующей окружность заданного радиуса.

2.1 Заготовка приложения с пустым окном

Запустим IntelliJ IDEA и создадим пустой проект. В окне структуры проекта в контекстном меню для папки `src` выберем пункт `New|Gui Form` и в появившемся диалоговом окне создания формы (см. рис. 1) введём название формы – `PictureForm`.

Форма в терминологии графического пользовательского интерфейса – это именованный макет окна. IntelliJ IDEA содержит редактор форм, позволяющий визуально размещать на форме *компоненты* графического пользовательского интерфейса – панели, кнопки, поля ввода, и т.д. При этом позиционирование элементов на форме управляется *менеджером позиционирования*, который можно выбрать из выпадающего списка в окне создания формы. Мы оставим менеджер позиционирования, предлагаемый по умолчанию, а именно – `GridLayoutManager`.

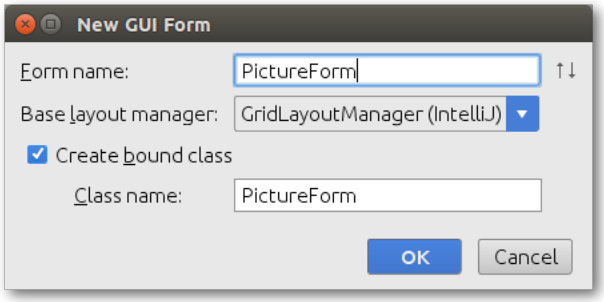

Рис. 1: Диалоговое окно New GUI Form.

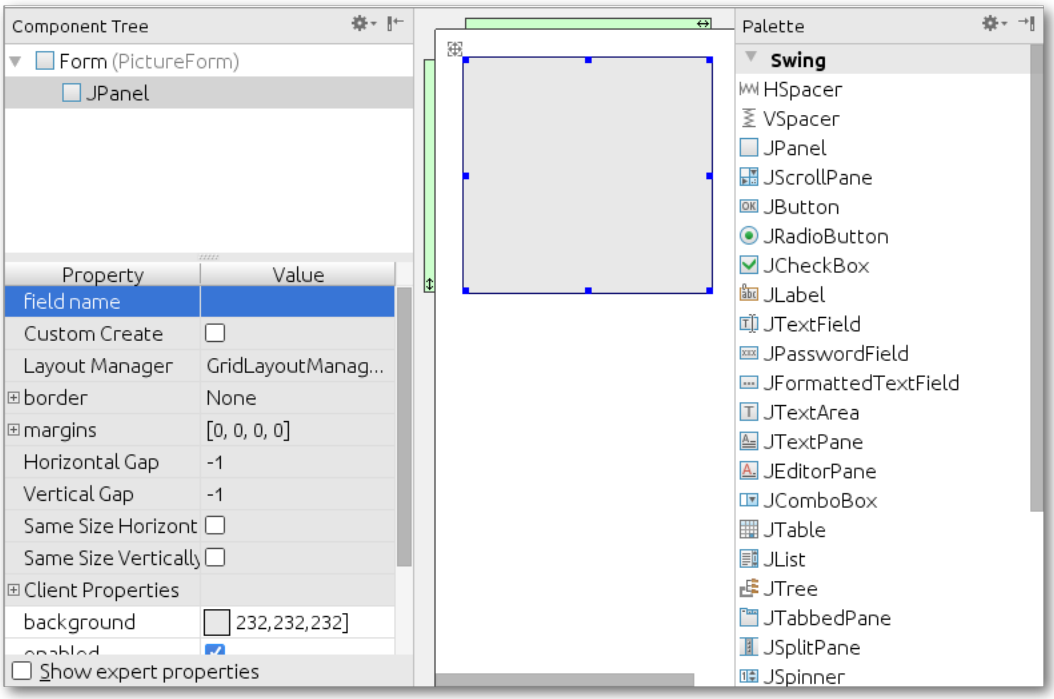

Рис. 2: Редактор форм.

Во время выполнения программы создание окна по макету и управление этим окном осуществляет объект класса `JFrame` библиотеки swing. При этом все размещённые на форме компоненты контролируются объектами соответствующих библиотечных классов, ссылки на которые удобно хранить в объекте нашего собственного класса, про который говорят, что он *связан* с формой. IntelliJ IDEA предлагает автоматически создать за нас этот класс, выбрав для него имя, совпадающее с названием формы.

Нажав кнопку OK в окне создания формы, мы получим два новых пункта в структуре проекта – форму `PictureForm.form` и связанный с ней класс `PictureForm`. При этом для формы будет открыт редактор форм.

Редактор форм состоит из трёх частей (рис. 2): слева располагается дерево компонент формы, посередине размещается изображение формы, а справа – палитра компонент.

Дерево компонент отражает отношение вложенности на множестве компонент формы. В корне дерева располагается сама форма. Другие вершины соответствуют либо так называемым *панелям*, которые выглядят как прямоугольные области, либо *элементам управления* – кнопкам, полям ввода и т.п. Панели служат контейнерами, в которые вкладываются другие компоненты. В элементы управления ничего не может быть вложено, поэтому они могут располагаться только в листьях дерева.

Изначально в дереве компонент присутствует только наша форма и единственная вложенная в неё панель. Если мы хотим добавить на форму новый компонент, мы должны «перетащить» его из палитры на изображение формы.

Выделив компонент в дереве, мы можем просматривать и редактировать его *свойства* в списке свойств, расположенном внизу дерева компонент. Свойства управляют внешним видом и поведением компоненты. Среди них особняком стоит свойство `field name`, через которое мы можем назначить имя полю связанного с формой класса. В этом поле во время работы программы будет храниться ссылка на объект, соответствующий компоненте.

Выделим в дереве панель и дадим ей имя – `mainPanel`. Кроме того, зададим размеры полей вокруг панели, установив для свойств `margins.top`, `margins.left`, `margins.bottom` и `margins.right` значение `10`. Переключившись в текстовый редактор для редактирования

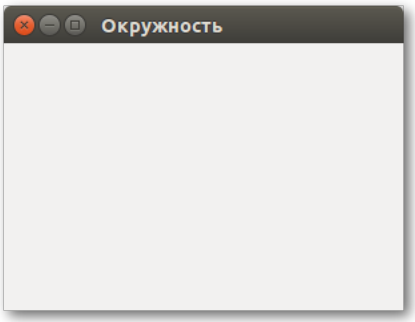

Рис. 3: Окно приложения при первом запуске.

исходного кода класса `PictureForm`, мы увидим, что в него автоматически добавилось поле `mainPanel`:

```
1  public class PictureForm {
2      private JPanel mainPanel;
3  }
```

Так как наше приложение будет содержать единственную форму, нам будет удобно, чтобы класс `PictureForm` был главным классом нашей программы. IntelliJ IDEA может автоматически создать для нашего класса метод `main`. Для этого переместим курсор в текстовом редакторе на имя класса и в главном меню выберем пункт `Code|Generate`. Выбрав пункт `Form main()` в появившемся контекстном меню, мы увидим, что исходный код класса `PictureForm` изменился:

```
1  import javax.swing.*;
2
3  public class PictureForm {
4      private JPanel mainPanel;
5
6      public static void main(String[] args) {
7          JFrame frame = new JFrame("PictureForm");
8          frame.setContentPane(new PictureForm().mainPanel);
9          frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
10         frame.pack();
11         frame.setVisible(true);
12     }
13 }
```

В строчке 7 создаётся объект класса `JFrame`, соответствующий рамке окна нашего приложения. Его конструктор получает в качестве параметра имя окна, которое будет отображаться в его заголовке. Мы назовём наше окно «Окружность».

В строчке 8 создаётся объект нашего класса `PictureForm` и его главная панель регистрируется в качестве содержимого рамки окна.

Отображение окна и запуск диспетчера, который будет передавать компонентам окна сообщения мыши и клавиатуры, осуществляется в строчке 11.

Теперь мы можем откомпилировать и запустить наше приложение. На рис. 3 показано, как его окно будет выглядеть на экране.

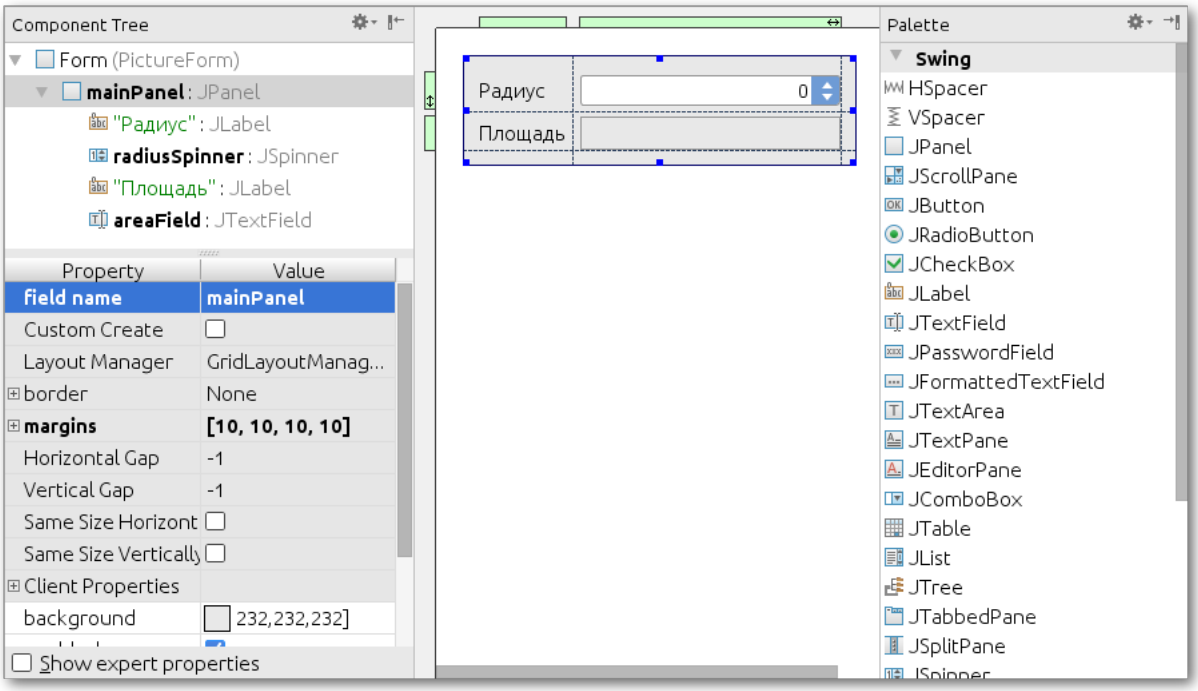

Рис. 4: Размещение компонентов на форме.

2.2 Добавление элементов управления

Перетащим из палитры компонентов на изображение формы две «метки» (`JLabel`), один «счётчик» (`JSpinner`) и одно текстовое поле (`JTextField`) так, как показано на рис. 4. Выбранный нами менеджер позиционирования предлагает размещать элементы управления в сетке, состоящей из рядов и колонок. При этом ряды и колонки добавляются автоматически по мере надобности, а их размеры определяются размерами элементов, содержащихся в ячейках сетки. Обратите внимание на то, что на форму автоматически добавляются компоненты `HSpacer` и `VSpacer`, позволяющие организовать увеличение полей панели при растягивании главного окна. Нам эта функциональность не нужна, поэтому мы эти компоненты просто удалим.

Метки нужны для размещения около элементов управления поясняющего текста. Этот поясняющий текст задаётся свойством `text` метки. Мы установим для текстов наших двух меток значения «Радиус» и «Площадь».

Через элемент управления «счётчик» пользователь сможет задавать радиус окружности. Дадим полю, на которое будет отображаться наш «счётчик», имя `radiusSpinner`.

Текстовое поле нам понадобится для демонстрации работоспособности «счётчика». Дадим ему имя `areaField`. Подразумевается, что пользователь будет менять радиус, и при этом в текстовом поле будет выводиться площадь круга, ограниченного окружностью. Так как нам не надо, чтобы пользователь мог редактировать площадь круга, мы выберем значение `false` для свойства `editable` текстового поля (снимем «галочку»).

Чтобы установить начальное значение радиуса, создадим в классе `PictureForm` конструктор, в котором вызовем метод `setValue` у «счётчика», передав ему значение 20. Обратите внимание на то, что к моменту вызова конструктора в поле `radiusSpinner` уже будет волшебным образом содержаться ссылка на объект компонента. Дело в том, что IntelliJ IDEA автоматически генерирует для нашего класса метод под названием `$$$setupUI$$$()`, в котором создаются объекты компонентов, и вызов этого метода невидимо для нас добавляется в начало котструктора класса, связанного с формой.

Исходный текст класса `PictureForm` после всех проведённых манипуляций должен при-

нять следующий вид:

```
1  import javax.swing.*;
2
3  public class PictureForm {
4      private JPanel mainPanel;
5      private JSpinner radiusSpinner;
6      private JTextField areaField;
7
8      public PictureForm() {
9          radiusSpinner.setValue(20);
10     }
11
12     public static void main(String[] args) {
13         JFrame frame = new JFrame("Окружность");
14         frame.setContentPane(new PictureForm().mainPanel);
15         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
16         frame.pack();
17         frame.setVisible(true);
18     }
19 }
```

Теперь нам осталось лишь привязать вычисление площади круга к событию изменения значения «счётчика». Для этого на изображении формы кликнем правой кнопкой мыши по компоненту `radiusSpinner`, и в контекстном меню выберем пункт `Create Listener`. В результате мы увидим список событий, которые может порождать «счётчик». Выберем в этом списке событие `ChangeListner`, и в конструкторе класса `PictureForm` появится вызов метода `addChangeListener` для объекта `radiusSpinner`. Этот метод получает в качестве параметра объект класса, реализующего функциональный интерфейс `ChangeListner`. Нам будет предложено реализовать метод `stateChanged` этого интерфейса, записав в этот метод код, который должен быть выполнен при каждом изменении значения «счётчика».

Текущее значение «счётчика» возвращает его метод `getValue`. Текст в текстовом поле меняется его методом `setText`. Тогда конструктор класса `PictureForm` должен приобрести вид

```
10     public PictureForm() {
11         radiusSpinner.addChangeListener(new ChangeListener() {
12             public void stateChanged(ChangeEvent e) {
13                 int radius = (int)radiusSpinner.getValue();
14                 double area = Math.PI*radius*radius;
15                 areaField.setText(String.format("%.2f", area));
16             }
17         });
18         radiusSpinner.setValue(20);
19     }
```

Запустив наше приложение, убедимся, что при изменении радиуса значение площади меняется.

2.3 Рисование на форме

Компонент, внутри которого можно рисовать, изначально в палитре компонент не присутствует. Однако мы можем создать собственный компонент, унаследовав его класс от ком-

понента `JPanel` и переопределив его метод перерисовки:

```
1  import javax.swing.*;
2  import java.awt.*;
3
4  public class CanvasPanel extends JPanel {
5      protected void paintComponent(Graphics g) {
6          super.paintComponent(g);
7      }
8  }
```

Класс нашего компонента называется `CanvasPanel`. Мы переопределили в нём метод перерисовки `paintComponent`. Этот метод автоматически вызывается всякий раз, когда нужно отрисовать на экране компонент (например, при запуске приложения, или после того, как частично или полностью закрытое другими окнами окно приложение получает фокус). В настоящее время наша реализация этого метода ничего полезного не делает, а только лишь вызывает метод `paintComponent` родительского класса для того, чтобы закрасить прямоугольную область, занимаемую компонентом.

Рисование в методе `paintComponent` организовано через объект класса `Graphics`, который передаётся ему в качестве параметра. В классе `Graphics` реализован набор графических примитивов таких, как рисование отрезка линии, окружности, текста и т.п. Обратите внимание, что координаты для этих графических примитивов задаются относительно верхнего левого угла компонента `CanvasPanel`.

Будем хранить радиус окружности в поле `radius` класса `CanvasPanel`. Чтобы устанавливать значение этого поля, добавим в класс метод `setRadius`, который, кроме того, будет вызывать перерисовку компонента. В результате мы получим готовый класс компонента, выглядящий как

```
1  import javax.swing.*;
2  import java.awt.*;
3
4  public class CanvasPanel extends JPanel {
5      private int radius = 20;
6
7      public void setRadius(int r) {
8          radius = r;
9          repaint();
10     }
11
12     protected void paintComponent(Graphics g) {
13         super.paintComponent(g);
14         g.setColor(Color.RED);
15         g.drawOval(10, 10, radius, radius);
16     }
17 }
```

Теперь сначала откомпилируем наш класс, а затем добавим его на форму.

Для добавления на форму нашего компонента кликнем левой кнопкой мыши по пункту `Non-Palette Component` в палитре, а затем кликнем в том месте изображения формы, куда мы хотим поместить наш компонент. В результате откроется диалоговое окно, показанное на рис. 5.

Выбираем в качестве класса компонента класс `CanvasPanel` из состава нашего проекта, ставим галочку `Create binding automatically` и нажимаем кнопку OK. Далее даём

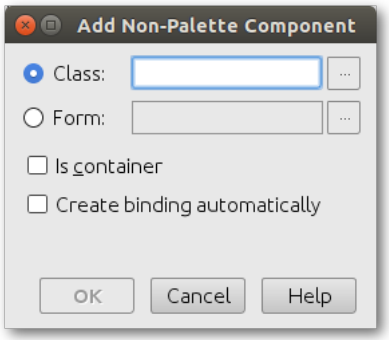

Рис. 5: Диалоговое окно Add Non-Palette Component.

имя `canvasPanel` полю, соответствующему нашему компоненту, и обязательно устанавливаем предпочтительные размеры компонента (свойство `Preferred Size`). Для красоты можно задать в качестве фона белый цвет, изменив значение свойства `background`.

Теперь осталось привязать перерисовку окружноси к событию изменения радиуса. Для этого изменим обработчик соответствующего события следующим образом:

```
1       radiusSpinner.addChangeListener(new ChangeListener() {
2           public void stateChanged(ChangeEvent e) {
3               int radius = (int)radiusSpinner.getValue();
4               canvasPanel.setRadius(radius);
5               double area = Math.PI*radius*radius;
6               areaField.setText(String.format("%.2f", area));
7           }
8       });
```

3 Задание

В течение лабораторной работы нужно разработать программу, рисующую на экране одно из изображений, перечисленных в таблицах 1 и 2. Программа должна иметь графический пользовательский интерфейс, через который пользователь может задавать параметры изображения. Изображение должно перерисовываться автоматически при изменении любого параметра.

Значения параметров, обозначенных в таблицах 1 и 2 латинскими буквами, представляют собой неотрицательные целые числа. Когда в описании изображения говорится о выборе цвета, подразумевается выбор из нескольких предопределённых альтернатив (например, красный, зелёный или синий).

Таблица 1: Варианты изображений

1	Прямоугольный треугольник выбранного цвета с катетами a и b, вписанный в окружность.
2	Параллелепипед в изометрической проекции со сторонами a, b и c с выбором, рисовать или не рисовать невидимые линии.
3	«Шахматная» доска размера $m \times n$ с выбором, чёрной или белой является верхняя левая клетка.
4	Одна из двух возможных (по выбору пользователя) развёрток тетраэдра со стороной a.
5	Окружности с радиусами a и b, центры которых расположены на расстоянии d друг от друга, а точки пересечения по желанию пользователя могут быть отмечены красными кружками.
6	Треугольник выбранного цвета со сторонами a, b и c.
7	Тетраэдр в изометрической проекции со стороной a и с выбором, рисовать или не рисовать невидимые линии.
8	Выбранная пользователем фигура из тетриса с длиной a стороны каждого из составляющих фигуру квадрата.
9	Набор из n прямоугольников, закрашенных через один по желанию пользователя, которыми аппроксимируется интеграл функции $y = x \cdot \sin \frac{a \cdot x}{b}$ на интервале от $-\pi$ до π при интегрировании методом средних прямоугольников.
10	Выбранная пользователем грань игральной кости (сторона грани – a, радиус «точки» на грани – $a/10$).
11	Правильный n-угольник со стороной a выбранного цвета.
12	Куб в изометрической проекции со строной a и с выбором, вписывать или не вписывать в него куб, вершины которого располагаются в центре граней описанного куба.
13	n-конечная звезда, по желанию пользователя закрашенная.
14	Набор из n трапеций, закрашенных через одну по желанию пользователя, которыми аппроксимируется интеграл функции $y = x \cdot \cos \frac{a \cdot x}{b}$ на интервале от $-\pi$ до π при интегрировании методом трапеций.
15	Квадрат со стороной a, по желанию пользователя заштрихованный под углом x градусов с шагом d.
16	«Пифагоровы штаны» с катетами a и b, закрашенные по желанию пользователя.
17	Правильная n-угольная прямая призма выбранного цвета в изометрической проекции с высотой h, равной стороне основания.
18	Спираль, закрученная в выбранную пользователем сторону, с количеством витков n и расстоянием между витками d.
19	Три окружности радиусов a, b и c, касающиеся друг друга. По желанию пользователя окружности закрашиваются разными цветами.
20	Круг радиуса r, по желанию пользователя заштрихованный под углом x градусов с шагом d.
21	Сектор круга радиуса r с длиной дуги l, который по выбору пользователя может быть закрашен.
22	Кубик рубика размера $n \times n \times n$ в изометрической проекции, который по выбору пользователя может быть или не быть раскрашен случайными цветами.
23	График функции $y = ax^2/b$ на интервале от 0 до 10, к которому проведена касательная выбранного из нескольких альтернатив цвета в точке x_0.
24	Круглый «смайлик» или «хмурик» (по выбору пользователя) радиуса r.
25	n закрашенных по желанию пользователя касающихся соседей окружностей, центры которых равномерно распределены по окружности радиуса r.

Таблица 2: Варианты изображений

26	«Паучок» с диаметром тела d, у которого n ножкек и m глаз.
27	«Многоножка» с n ногами и двумя усами, изогнутая по синусоиде $y = A\sin\omega x$ на интервале от -2π до 2π. Здесь ω – вещественное число.
28	«Домик» с двускатной крышей, у которого n этажей, а на каждом этаже по m окон. Печное отопление – по желанию пользователя.
29	«Танк» выбранного пользователем цвета, у которого n башен с пушками.
30	«Кентавр» выбранного пользователем цвета, у которого n рук и m ног.
31	Красная планета диаметра d, вокруг которой расположены зелёные спутники диаметром $d/3$, расстояния орбит которых от поверхности планеты образуют последовательность Фибоначчи с начальным членом $f > d/3$. Спутники, которые поместились на картинку, не должны быть расположены на одной прямой. Орбиты спутников должны быть изображены синим цветом.
32	Монитор с диагональю d и соотношением сторон, задаваемым дробью a/b. По желанию пользователя на экране монитора может быть нарисован такой же монитор, причём на экране нарисованного монитора тоже рисуется монитор и т.д.
33	Бутылка, имеющая высоту h и диаметр d, с горлышком длины $h/3$ и диаметром $d/3$, заполненная жидкостью на k процентов.
34	Стакан, имеющий форму усечённого конуса. Диаметры низа и верха стакана – d_1 и d_2, высота стакана – h. Стакан заполнен жидкостью на k процентов.
35	«Дерево», изображаемое как ствол высоты h, от которого влево и вправо под углом α к стволу исходят ветки. Расстояния между ветками, если идти от верхушки вниз, образуют последовательность чисел Фибоначчи. Форма кроны дерева – конус с диаметром основания d.
36	«Паровоз» выбранного пользователем цвета с n колесами и m трубами.
37	Сосуд в виде шара диаметром d, заполненный на k процентов.
38	Лабиринт с одним входом и одним выходом, состоящий из квадратных клеток со стороной k и имеющий размер $m \times n$ клеток.
39	
40	