

Лабораторная работа №6

«Программа с графическим пользовательским интерфейсом»

Цель работы

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

Исходные данные

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

Заготовка приложения с пустым окном

Запустим IntelliJ IDEA и создадим пустой проект. В окне структуры проекта в контекст-ном 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 визуальнo 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.

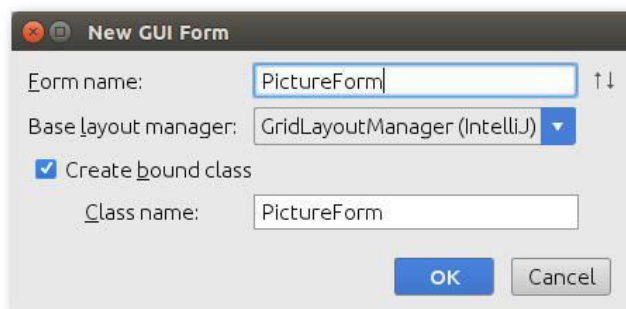


Рисунок 1 – Диалоговое окно New GUI Form

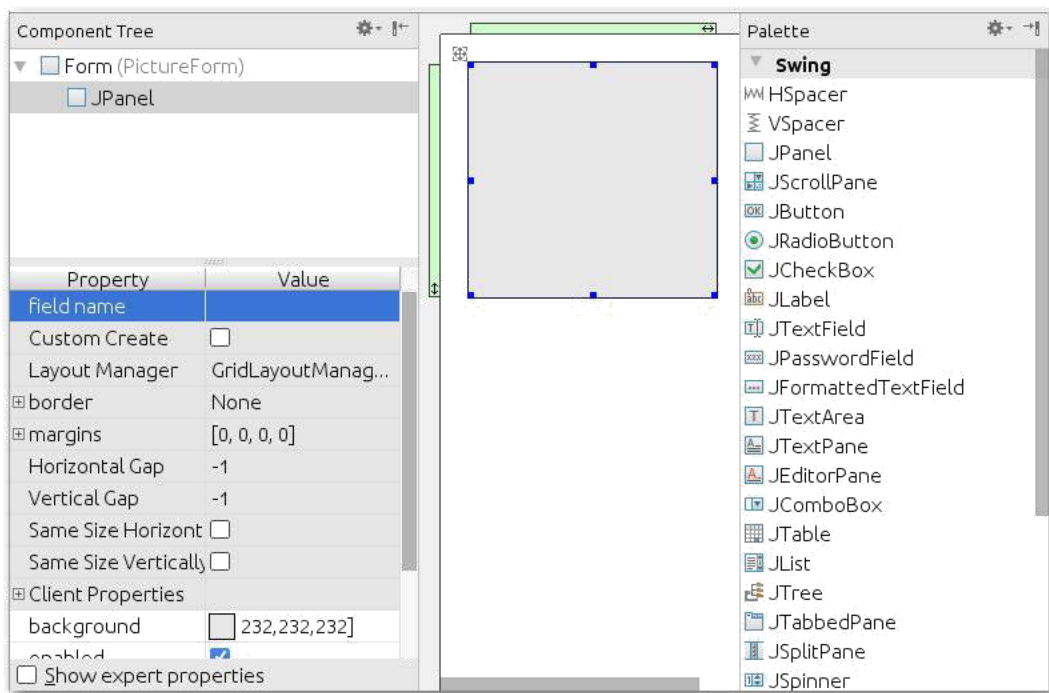


Рисунок 2 – Редактор форм

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

```
public class PictureForm {  
  
    private JPanel mainPanel ;  
  
}
```

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()` появившемся контекстном меню, мы увидим, что исходный код класса `PictureForm` изменился:

```
import javax.swing.*;  
  
public class PictureForm {  
  
    private JPanel mainPanel;  
  
    public static void main (String[] args) {  
  
        JFrame frame = new JFrame("PictureForm") ;  
  
        frame.setContentPane (new PictureForm().mainPanel) ;  
  
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE) ;  
  
        frame.pack();  
  
        frame.setVisible(true);  
  
    }  
  
}
```

В примере кода приведенного выше создается объект класса `JFrame`, соответствующий рамке окна нашего приложения. Его конструктор получает в качестве параметра имя окна, которое будет отображаться в его заголовке. Мы назовем наше окно «Окружность».

Далее, следуя по коду, создается объект нашего класса `PictureForm` и его главная панель регистрируется в качестве содержимого рамки окна. Отображение окна и запуск диспетчера, который будет передавать компонентам окно сообщения мыши и клавиатуры, осуществляется в методе `frame.setVisible(true)`.

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

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.

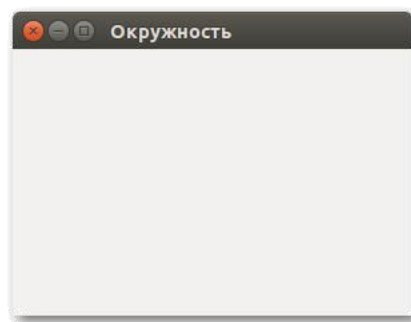


Рис. 3: Окно приложения при первом запуске.

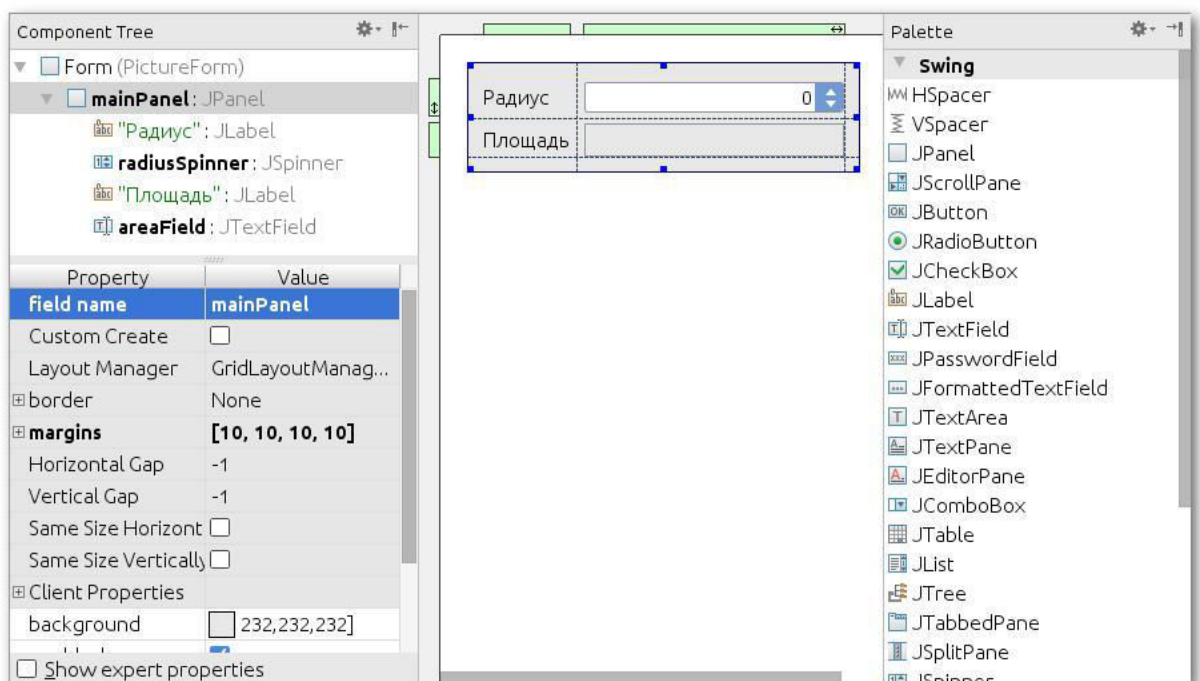


Рис. 4: Размещение компонентов на форме.

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 конструктора класса, связанного с формой.

Исходный текст класса PictureForm после всех проведённых манипуляций должен принять следующий вид:

```
import javax.swing.*;

public class PictureForm {

    private JPanel mainPanel ;

    private JSpinner radiusSpinner;

    private JTextFieldarea Field;

    public PictureForm ( ) {

        radius Spinner.setValue(20);

    }

    public static void main (String[ ] args) {

        JFrame frame = new JFrame("Окружность");

        frame.setContentPane (new PictureForm().mainPanel) ;

        frame.set DefaultCloseOperation(JFrame.EXIT_ON_CLOSE) ;

        frame.pack();

        frame.set Visible(true) ;

    }

}
```

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 `ChangeListener`, и в конструкторе класса `PictureForm` появится вызов метода `addChangeListener` для объекта `radiusSpinner`. Этот метод получает в качестве параметра объект класса, реализующего функциональный интерфейс `ChangeListener`. 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:

```
public PictureForm ( ) {  
    radius Spinner.addChangeListener (new ChangeListener ( ) {  
        public void stateChanged ( ChangeEvent e ) {  
            int radius = (int)radiusSpinner.getValue();  
            double area = Math.PI*radius*radius;  
            areaField.setText(String.format("%.2f",area));  
        }  
    } );  
    radiusSpinner.setValue(20);  
}
```

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

Рисование на форме

Компонент, внутри которого можно рисовать, изначально в палитре компонент не присутствует. Однако мы можем создать собственный компонент, унаследовав его класс от компонента `JPanel` и переопределив его метод перерисовки:

```
import javax.swing.*;  
import java.awt.*;  
public class CanvasPanel extends JPanel {  
    protected void paintComponent (G raphics g) {  
        super.paintComponent (g);  
    }  
}
```

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 приложения 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

```
import javax.swing.*;
import java.awt.*;

public class CanvasPanel extends JPanel {
    private int radius = 20;

    public void setRadius(int r) {
        radius = r;
        repaint();
    }

    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.setColor(Color.RED);
        g.drawOval(10,10,radius,radius);
    }
}
```

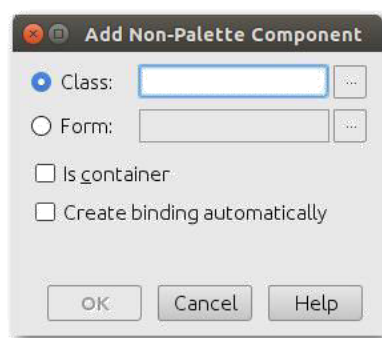


Рис. 5: Диалоговое окно Add Non-Palette Component.

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 ОК. Далее даём 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 окружности к событию изменения радиуса. Для этого изменим обработчик соответствующего события следующим образом:

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

Задание

В течение лабораторной работы нужно разработать программу, рисующую на экране одно из изображений, перечисленных в таблицах 1 – 7. 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 – 7 латинскими буквами, представляют собой неотрицательные целые числа. Когда в описании изображения говорится о выборе цвета, подразумевается выбор из нескольких predetermined альтернатив (например, красный, зелёный или синий).

Дополнительная информация

Если возникнет проблема с компиляцией формы, то необходимо воспользоваться по очереди одной из ниже приведенных рекомендаций.

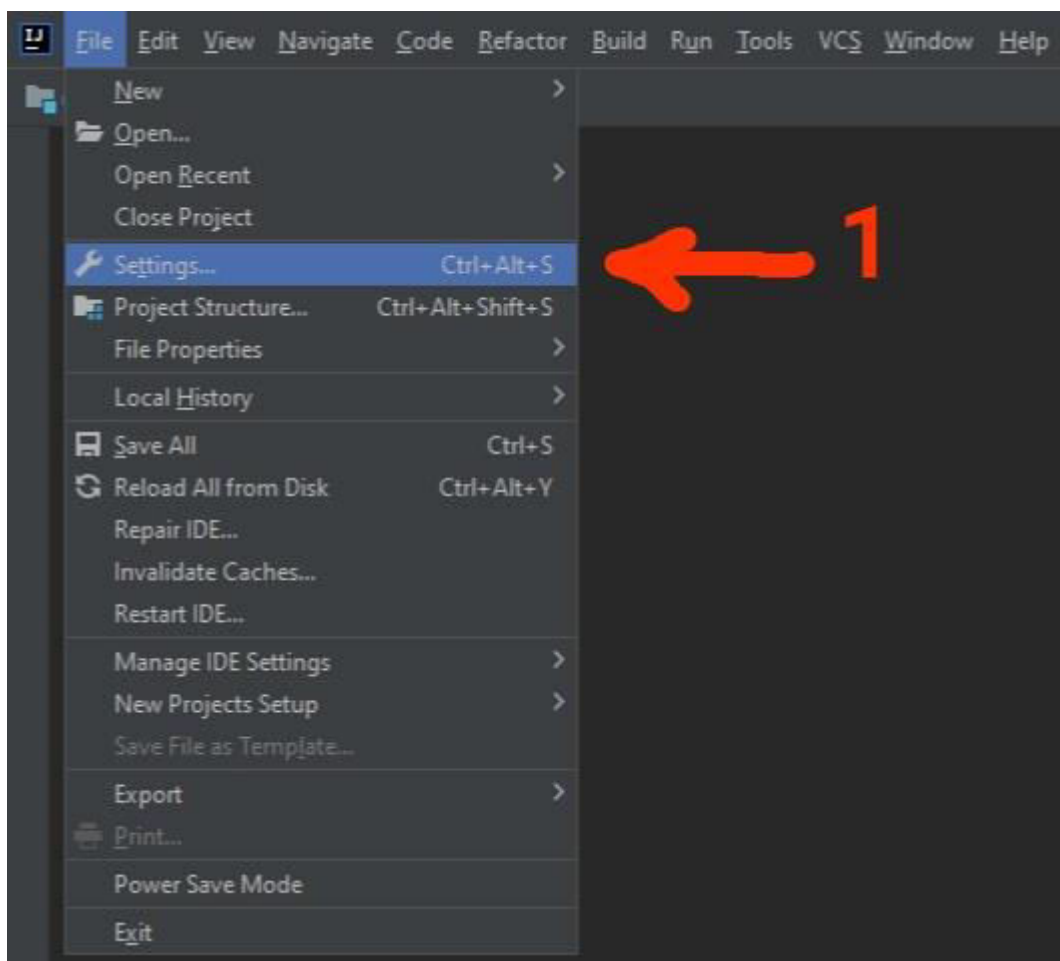
Рекомендация 1

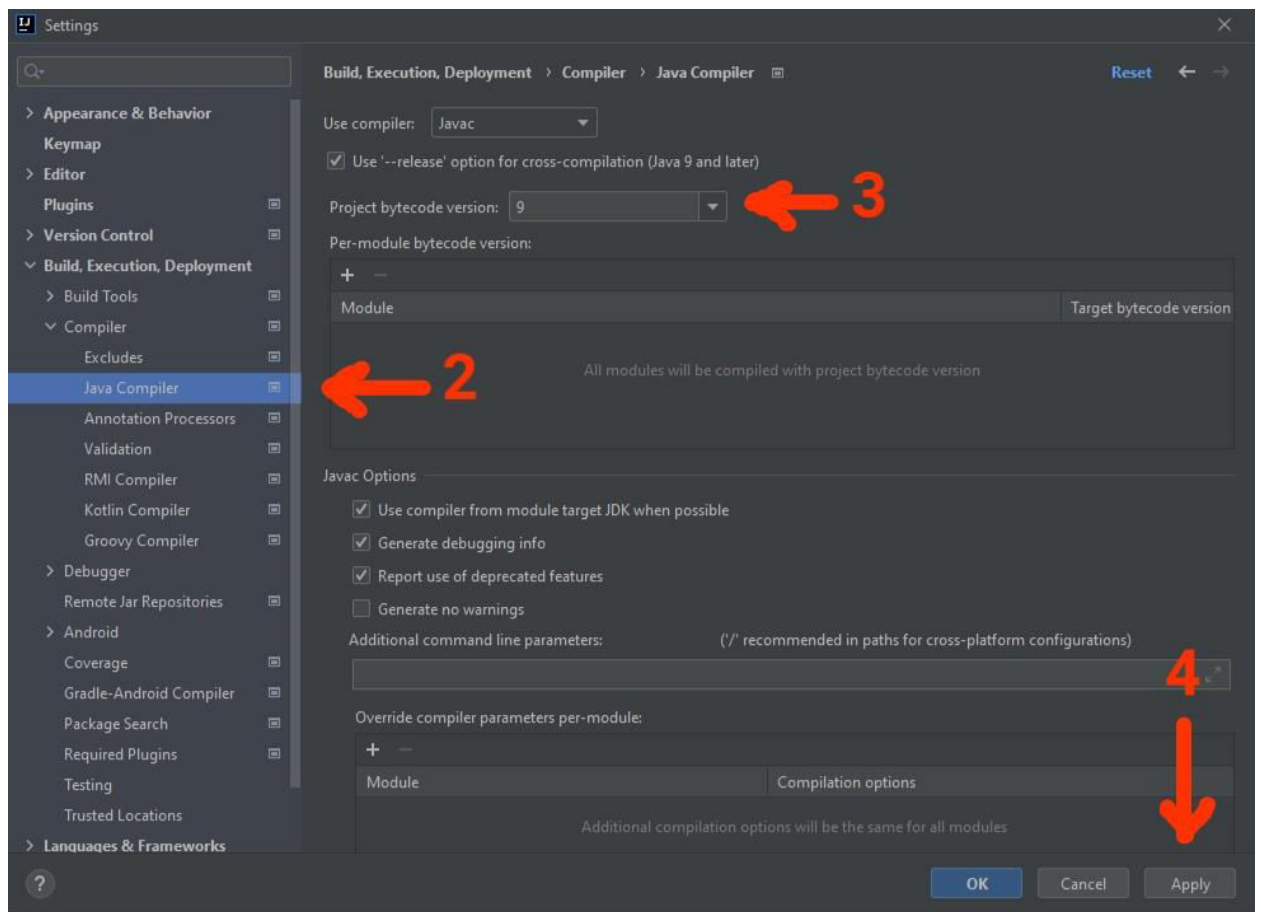
Подразумевается, что эта инструкция по хронологии идёт сразу после пятой иллюстрации данного методического пособия. Для возможности добавления холста в нашу форму для начала нужно выбрать соответствующие версии компилятора и проекта.

Для этого нужно перейти по следующему пути в главном меню настроек компилятора:

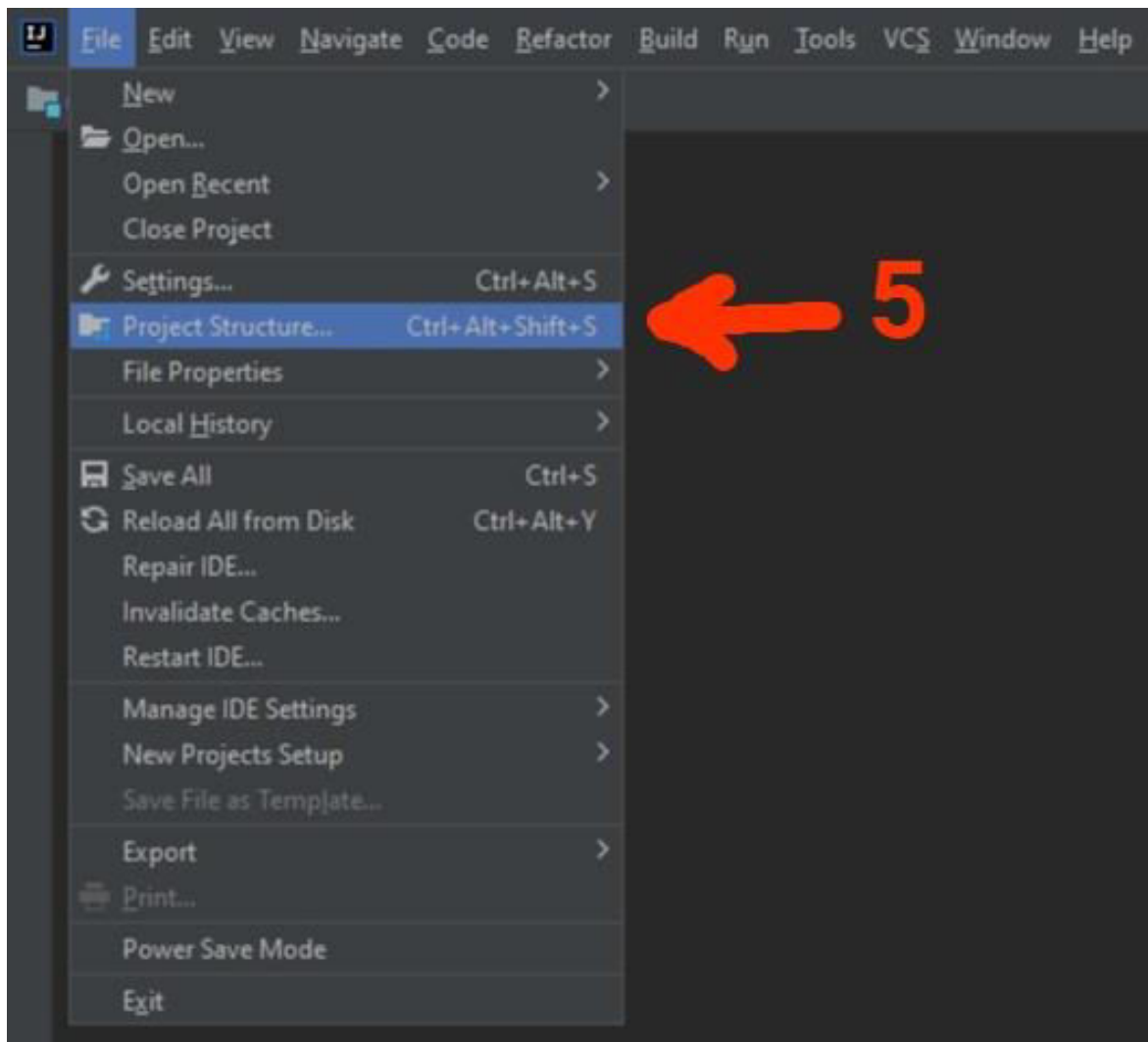
- {File → Settings... → Build, Execution, Deployment → Compiler → Java Compiler}.
- Далее в окошке «Project bytecode version:» выбираем цифру 9 и применяем настройки(Apply).
- Теперь переходим по этому пути главного меню: {File → Project Structure... → Project Settings → Project}
- и в окне с надписью «Language level:» выбираем вариант «9 — Modules, private methods in interfaces etc.».
- Теперь откомпилируем наш класс...

См. скриншоты ниже:

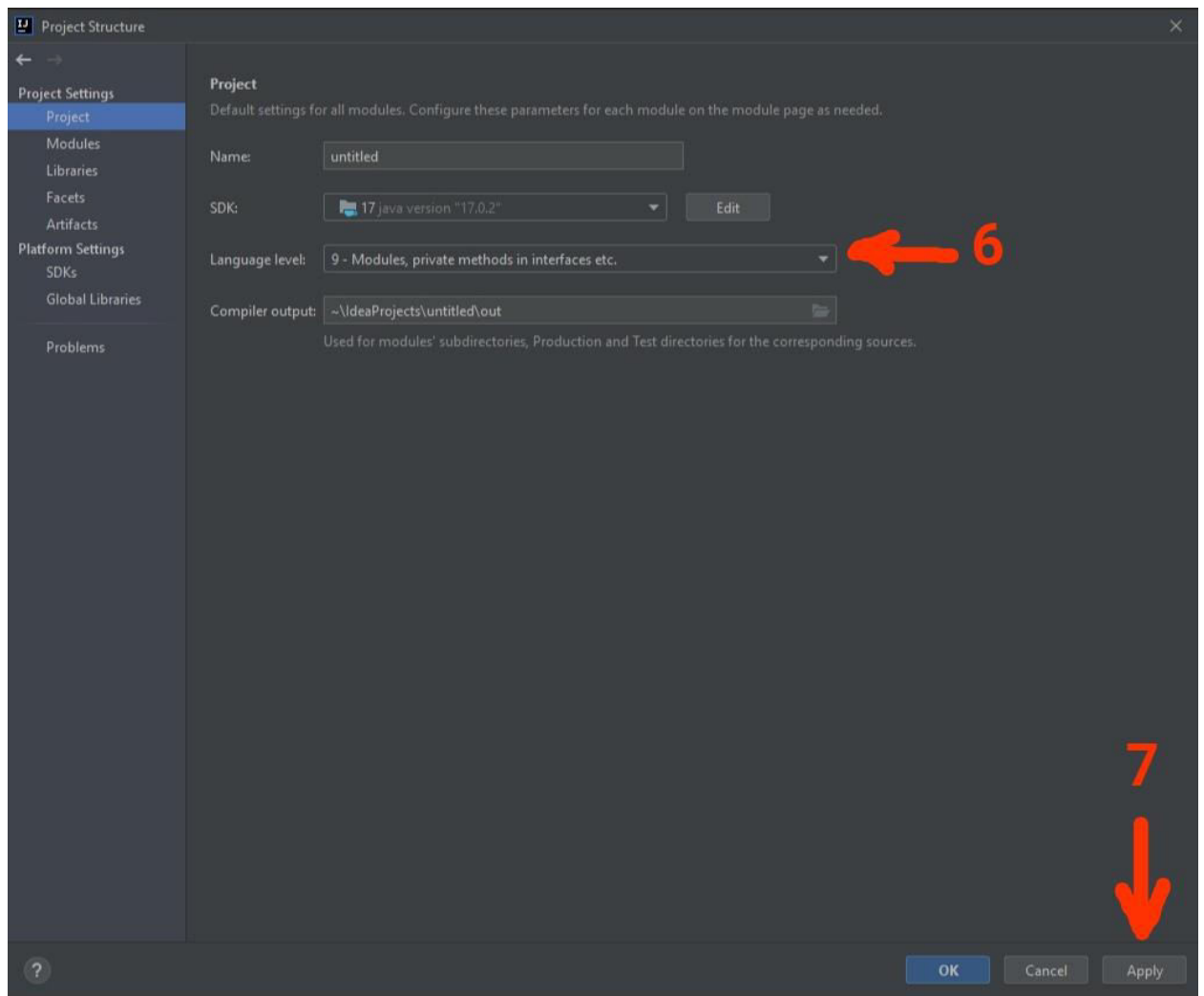




Шаг 2,3,4



Шаг 5



Шаг 6,7

Если проблема не решается, то необходимо попробовать сделать Рекомендацию2.

Рекомендация 2

Откатывать на бю версию, ставишь non palette - будет работать.