

Инструментарий: GUI – графический интерфейс пользователя

Учебник 1_JDK



Оглавление

6. Инструментарий: GUI – графический интерфейс пользователя	3
В предыдущем разделе	3
6.1. Почему именно Swing?	4
6.2. JFrame: Главный класс окна	4
6.2.1. Создание окна	4
6.2.2. Заккрытие окна, завершение приложения	5
6.2.3. Свойства окна	6
6.2.4. Вопросы для самопроверки	8
6.3. Компоненты окна	8
6.3.1. Кнопка	8
6.3.2. Компоновщики (менеджеры размещений)	10
6.3.3. Панель для размещения JPanel	12
6.4. Многооконное приложение, взаимосвязи	14
6.4.1. Структура	14
6.4.2. Окно с настройками игры и обработчики кнопок	16
6.4.3. Последовательность выполнения программы	17
6.4.4. Вопросы для самопроверки	19
6.5. Основная панель с игрой	20
6.5.1. Рисование	20
6.5.2. Обработчик мышки	22
6.5.3. Логика игры	23
6.5.4. Последние приготовления	27
Практическое задание	29
Термины, определения и сокращения	30


6. Инструментарий: GUI – графический интерфейс пользователя

В предыдущем разделе

- ООП;
- процедурное программирование;
- исключения;
- устройство языка Java;
- устройство платформы для создания и запуска приложений на JVM-языках; базовые средства ввода-вывода;
- базовые терминальные приложения;
- алгоритмические задачи, не требующие сложных программных решений.

В этом разделе

Интерфейс пользователя – это важно, поскольку это именно то, что видят пользователи в первую очередь.

 Это самое сложное занятие начального этапа, но не потому что оно содержит очень сложную информацию, а потому что заставит под необычным углом взглянуть на те принципы ООП, которые уже известны.

Будут рассмотрены вопросы создания окон, менеджеров размещений, элементов графического интерфейса, и обработчиков событий.

- Swing;
- Асинхронность;
- Параллельность;
- Окно;
- Компонент;
- компоновщик;
- Панель;
- События;
- Обработчик

6.1. Почему именно Swing?

Вместо вступления следует кратко ответить на самый популярный вопрос.

- ✗ это популярный и современный фреймворк;
- ✗ пригодится любому программисту на Java; поможет лучше понять ООП;
- ✓ работа композиции из объектов;
- ✓ обмен информацией между объектами;
- ✓ явно использует ссылочную природу данных;
- ✓ улучшает запоминание базовых взаимосвязей объектов; без искусственных примеров (в результате будет разработана простая игра, крестики-нолики с графическим интерфейсом).

Почему не JavaFX? Фреймворк был выведен из стандартной библиотеки языка, начиная с Java 9, и достаточно сложен для базового знакомства с графическими библиотеками.

Почему не LibGDX? Фреймворк является надстройкой над Swing, объясняя его всё равно необходимо будет объяснять Swing/AWT.

IntelliJ IDEA — написана на Swing.

6.2. JFrame: Главный класс окна

6.2.1. Создание окна

В этом разделе происходит изучение программирования и ООП, на примере и с использованием фреймворка Swing, поэтому необходимо сосредоточиться на объектах, их свойствах и взаимосвязях.



Изучение фреймворка Swing – не основная задача раздела, поэтому важно помнить, что не нужно зазубривать все названия всех компонентов.

Окно графического интерфейса, как и любая другая программа – это класс и объекты. В листинге 1 создаётся новый класс с названием GameWindow. Для

начала, необходимо получить доступ к методам, содержащимся в библиотеке, для этого применяется наследование от класса `JFrame`, и создаётся конструктор.

```

1 package ru.gb.jdk.one.online;
2 import javax.swing.*;
3 public class GameWindow extends JFrame {
4     GameWindow() {
5         //...
6     }
7 }

```

Листинг 1: Класс окна

В основном классе программы просто создаётся новый объект вновь описанного класса с окном. С точки зрения программирования и ООП не происходит ничего значительно отличающегося от котиков и пёсиков. Если запустить получившееся приложение, оно должно запуститься, и сразу завершиться, это признак верно написанного кода.

```


1 package ru.gb.jdk.one.online;
2 public class Main {
3     public static void main(String[] args) {
4         new GameWindow();
5     }
6 }

```

Листинг 2: Создание нового окна

6.2.2. Заккрытие окна, завершение приложения

Большая часть свойств окна не меняется за всё время существования окна и задаётся в конструкторе, то есть окно при создании будет наделено какими-то свойствами, которые в некоторых случаях можно будет поменять во время работы приложения. Самое не очевидное на первый взгляд то, что в Swing при нажатии на крестик в углу окна программа не завершается. По умолчанию, все создаваемые окна – невидимые. Это сделано потому что есть возможность создавать сколько угодно окон для приложения и такое поведение значительно снижает риск неожиданного поведения.

 Все окна по умолчанию невидимые. Нажатие на крестик по умолчанию делает окно невидимым, а не завершает программу.

Для того, чтобы программа закрылась, необходимо принять решение, какое окно в ней будет главным. Чтобы программа завершалась при закрытии главного окна (обычно, это первое, что делается при создании одно оконных приложений) экземпляру `JFrame` устанавливается свойство `DefaultCloseOperation`. То есть устанавливается, что нужно сделать, когда это (главное) окно закроется. В это свойство записывается константа `EXIT_ON_CLOSE`. Если этого не сделать, то будет использовано поведение по умолчанию, окно сделается невидимым, и приложение не завершится.

```
1 setDefaultCloseOperation(EXIT_ON_CLOSE);
```

Листинг 3: Завершение приложения по закрытию окна

6.2.3. Свойства окна

Добавив констант¹ с шириной, высотой и положением окна по осям относительно рабочего стола, становится возможным настроить размеры и положение окна в конструкторе.

```
1 private static final int WINDOW_HEIGHT = 555;
2 private static final int WINDOW_WIDTH = 507;
3 private static final int WINDOW_POSX = 800;
4 private static final int WINDOW_POSY = 300;
5
6 GameWindow() {
7     setDefaultCloseOperation(EXIT_ON_CLOSE);
8     setLocation(WINDOW_POSX, WINDOW_POSY);
9     setSize(WINDOW_WIDTH, WINDOW_HEIGHT);
10
11     setVisible(true);
12 }
```

Листинг 4: Базовые свойства окна

Установка всех начальных свойств окна осуществляется вызовом соответствующих сеттеров в конструкторе. По умолчанию окно – невидимое, поэтому для его демонстрации в конструкторе вызывается метод `setVisible` с передаваемым аргументом `true`.

¹ Считается хорошим тоном не держать в коде никаких «магических цифр», чтобы не думать, что они значат и почему они именно такие, и что будет если их поменять.

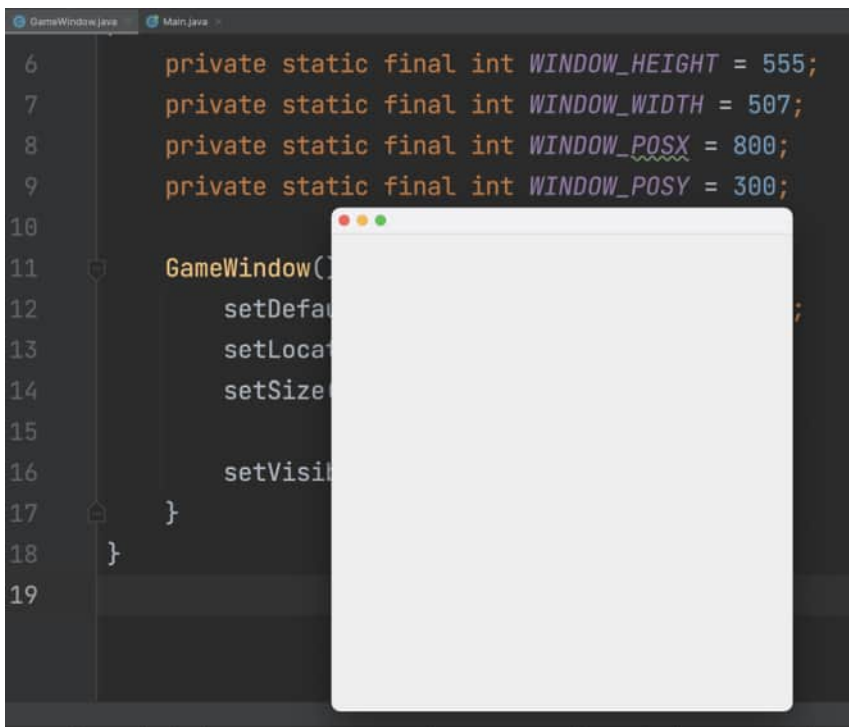


Рис. 1: Пустое окно указанного размера

Окно – это всегда отдельный поток программы, внутри которого работает бесконечный цикл. В объекте окна существует очередь сообщений, которую цикл опрашивает и выполняет.

```

1 public static void main(String[] args) {
2     new GameWindow();
3     System.out.println("Method main() is over");
4 }
  
```

Листинг 5: Пример сообщения, показывающего многопоточность

Запустив программу и внимательно изучив результат, очевидно, что окно создалось, в консоли видно, что работа метода `main` закончилась, а окно всё равно выполняется, его, при желании, можно подвигать, изменить размер и так далее. Это и есть наглядная демонстрация многопоточности. Таким образом, получается, что когда создаётся новое окно – нет необходимости его ни в какой контейнер помещать, ни думать, как оно будет взаимодействовать с пользователем, оно создастся и будет жить своей жизнью. **Инкапсуляция.** Если появится необходимость что-то ещё выполнить в методе `main`, запрета на написание действий не существует и действия будут выполняться параллельно, асинхронно.

6.2.4. Вопросы для самопроверки

1. Почему в классе `GameWindow` доступны методы фреймворка?
 - (a) Из-за устройства фреймворка `Swing`;
 - (b) из-за наследования от `JFrame`;
 - (c) из-за импорта классов `Swing`.
2. Чтобы создать пустое окно в программе нужно
 - (a) импортировать библиотеку `Swing`;
 - (b) создать класс `MainWindow`;
 - (c) создать класс-наследник `JFrame`.
3. Свойства окна, такие как размер и заголовок возможно задать
 - (a) написанием методов в классе-наследнике;
 - (b) вызовом методов в конструкторе;
 - (c) созданием констант в первых строках класса.

6.3. Компоненты окна

6.3.1. Кнопка

Для того, чтобы точно ничего не перепутать в процессе разработки, возможно придать окну больше индивидуальности, задав заголовок и запретив пользователю изменять его размеры, для игры в крестики-нолики это будет важно, чтобы красиво отображалось поле. Для этого вызовем методы `setTitle()` и `setResizable()`, соответственно.



Элементы графического интерфейса – это хорошо знакомые кнопки, текстовые поля, надписи (лейблы), и тому подобные.

За кнопки отвечает класс `JButton`, при создании экземпляра есть возможность сразу в конструкторе задать надпись, которая будет отображаться на кнопке. Сразу создадим несколько кнопок, например, «выход». Кнопки недостаточно просто создать, поскольку неизвестно, где они должны находиться. Одну из созданных кнопок добавим на окно – для этого внутри конструктора необходимо

воспользоваться методом `add()`, который требует в качестве аргумента передать ему какой-то `Component`. Все кнопки, лейблы и прочие элементы интерфейса – это наследники класса `Component`.

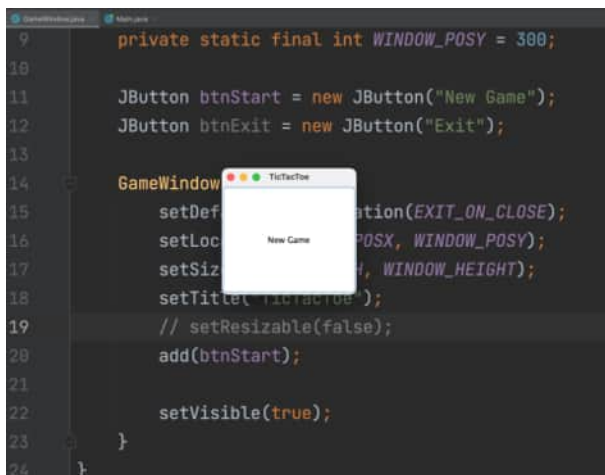
```

1  JButton btnStart = new JButton("New Game");
2  JButton btnExit = new JButton("Exit");
3
4  GameWindow() {
5      setDefaultCloseOperation(EXIT_ON_CLOSE);
6      setLocation(WINDOW_POSX, WINDOW_POSY);
7      setSize(WINDOW_WIDTH, WINDOW_HEIGHT);
8      setTitle("TicTacToe");
9      setResizable(false);
10     add(btnStart);
11
12     setVisible(true);
13 }

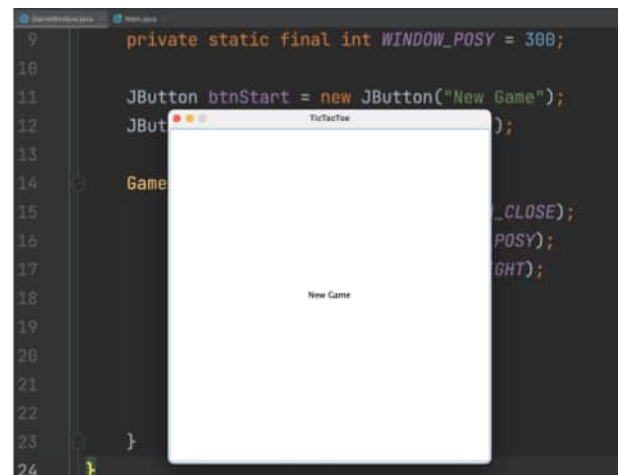
```

Листинг 6: Компонент «Кнопка»

После добавления в конструкторе кнопки, при запуске приложения видно, что она заняла всё окно приложения. Если убрать вызов метода `setResizable()`, то также возможно удостовериться, что при изменении размеров окна, размер кнопки также будет меняться.



а)



б)

Рис. 2: Изменение размеров кнопки в следствие изменения размеров окна

При попытке добавить вторую кнопку на это же окно очевидно (рис. 3), что вторая кнопка полностью перекрывает первую.

```

1  add(btnExit);

```

Листинг 7: Вторая кнопка

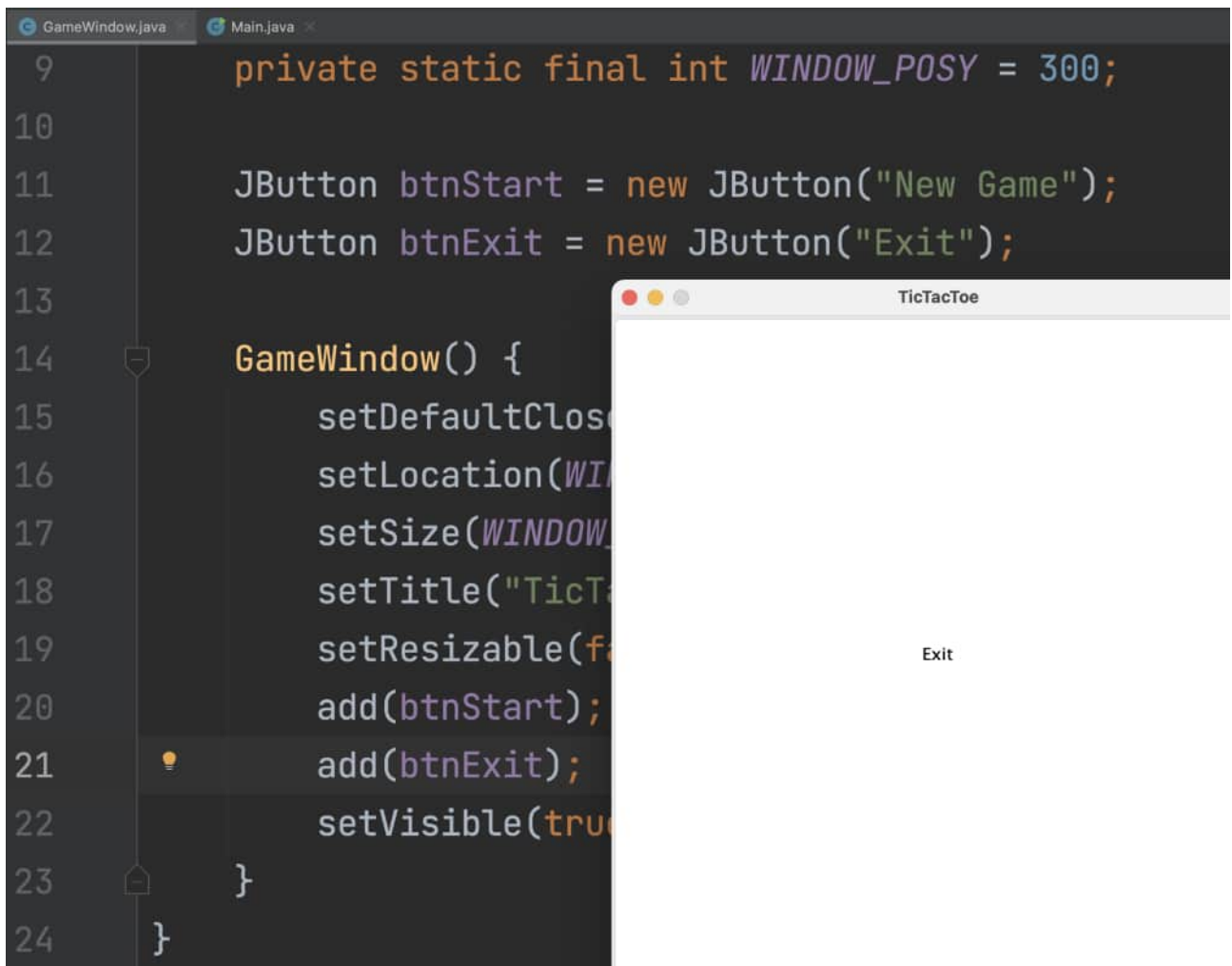


Рис. 3: Перекрывание компонентов

6.3.2. Компоновщики (менеджеры размещений)

Перекрывание происходит из-за использования компоновщика, или, как их ещё называют, менеджера размещений.



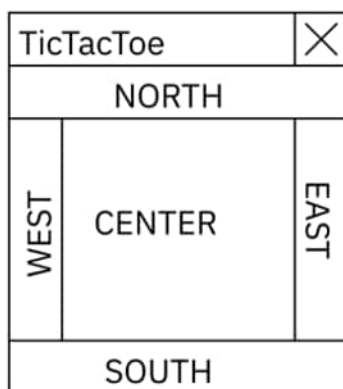
Менеджеры размещений нужны для того, чтобы не думать каждый раз о том, как изменится размер и координаты конкретного элемента, допустим, при изменении окна и не писать сложное поведение вложенных компонентов чтобы просто отобразить то, что привычно пользователю.

Компоновщики активно используются в любом программировании графических интерфейсов в любых языках программирования, от C++ до JavaScript, потому что это достаточно удобный механизм, берущий на себя значительный пласт работы.

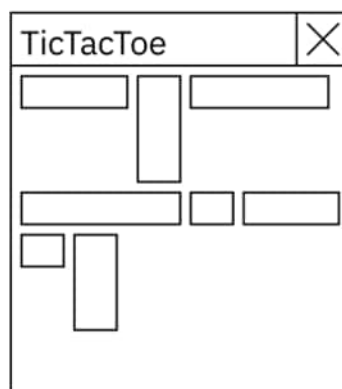
Использование компоновщиков позволяет эффективно управлять и размещать компоненты в окне или панели пользовательского интерфейса, обеспечивая гибкость и адаптивность приложения к изменениям размеров и расположения компонентов на экране. Компоновщик – это специальный объект, который помещается на некоторые (RootPaneContainer²) компоненты и осуществляет автоматическую расстановку добавляемых в него компонентов, согласно правилам. Компоновщиков существует несколько типов, каждый из которых предоставляет свои специфические возможности и алгоритмы расположения. Компоновщик выбирается в зависимости от требуемых задач и желаемого внешнего вида интерфейса.

- BorderLayout (по умолчанию);
- BoxLayout;
- CardLayout;
- FlowLayout;
- GridBagLayout;
- GridLayout;
- GroupLayout;
- SpringLayout.

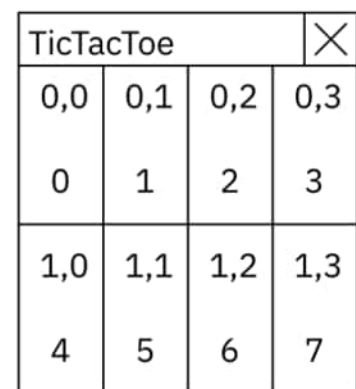
По умолчанию в Swing используется компоновщик BorderLayout (рис. 4а). Он располагает всё, что ему передаётся в центре, но также у него есть ещё четыре положения, маленькие области по краям.



(a) BorderLayout



(b) FlowLayout



(c) GridLayout

Рис. 4: Популярные менеджеры размещений

Если какая-то область не занята компонентом, она автоматически уменьшается до нулевых размеров, оставляя место другим компонентам. Поэтому, если необходимо какой-то компонент расположить не в центре, это нужно явно указать при

² docs.oracle.com: Компоновщики

добавлении. На первый взгляд, это немного не очевидно, поэтому лучше запомнить, что при добавлении надо указать ещё один параметр, константу, например, `BorderLayout.SOUTH`. `FlowLayout` будет располагать элементы друг за другом слева направо, сверху вниз. Компоновщик-сетка `GridLayout` при создании принимает на вход число строк и столбцов и располагает компоненты в получившейся сетке.



Основная идея, которую надо понять, это не названия компоновщиков, а то, что в Swing вся работа происходит через компоновщики – `Layout`, которые каждый по-своему располагают элементы в окне.

6.3.3. Панель для размещения `JPanel`

Разнообразие требований к разработке графических интерфейсов может привести к необходимости создания бесконечного числа компоновщиков. Поэтому разработчики библиотеки Swing придумали использовать не только компоненты сами по себе, но и группы элементов, которые располагаются на так называемых панелях (`JPanel`). Главная особенность панелей в том, что внутри каждой панели возможно использовать свой собственный компоновщик. `JPanel` – это по умолчанию невидимый прямоугольник, на котором может находиться собственный компоновщик. Например, становится доступным создание для окна панели с кнопками, а остальное пространство оставить под другие важные вещи. В листинге 8 описан код создания панели, добавление её в нижнюю часть основного экрана, расположение внутри панели компоновщика и двух кнопок. Важно, что на экран добавляются не кнопки по отдельности, а компонент, на который предварительно добавили кнопки.

```

1  GameWindow() {
2      setDefaultCloseOperation(EXIT_ON_CLOSE);
3      setLocation(WINDOW_POSX, WINDOW_POSY);
4      setSize(WINDOW_WIDTH, WINDOW_HEIGHT);
5      setTitle("TicTacToe");
6      setResizable(false);
7
8      JPanel panBottom = new JPanel(new GridLayout(1, 2));
9      panBottom.add(btnStart);
10     panBottom.add(btnExit);
11     add(panBottom, BorderLayout.SOUTH);
12     setVisible(true);
13 }

```

Листинг 8: Создание и применение панели

JPanel позволяет также осуществлять рисование и взаимодействие с пользователем. Основные графические интерактивности в демонстрационном приложении будут сделаны именно на панели. Такую панель с достаточно большой функциональностью логично выделить в отдельный класс. В случае игры в крестики-нолики это будет карта поля сражения (листинг 9). В описании конструктора для простоты панель перекрашивается в чёрный цвет (строка 8), чтобы увидеть, что панель создаётся без ошибок.

```

1 package ru.gb.jdk.one.online;
2
3 import javax.swing.*;
4 import java.awt.*;
5
6 public class Map extends JPanel {
7     Map() {
8         setBackground(Color.BLACK);
9     }
10 }

```

Листинг 9: Создание панели с полем боя

Естественно, панель также недостаточно просто создать (листинг 10, строка 8), но нужно её кудато разместить. Например, на основной экран (строка 14). Поскольку не была указана сторона экрана, панель заняла всё свободное место на окне, кроме юга, где расположилась панель с кнопками.

```

1 GameWindow() {
2     setDefaultCloseOperation(EXIT_ON_CLOSE);
3     setLocation(WINDOW_POSX, WINDOW_POSY);
4     setSize(WINDOW_WIDTH, WINDOW_HEIGHT);
5     setTitle("TicTacToe");
6     setResizable(false);
7
8     Map map = new Map();
9
10    JPanel panBottom = new JPanel(new GridLayout(1, 2));
11    panBottom.add(btnStart);
12    panBottom.add(btnExit);
13    add(panBottom, BorderLayout.SOUTH);
14    add(map);
15    setVisible(true);
16 }

```

Листинг 10: Добавление панели с полем боя

6.4. Многооконное приложение, взаимосвязи

6.4.1. Структура

Полученных знаний достаточно, чтобы начать описывать так называемую бизнес-логику. Созданная панель Map будет выполнять функции поля боя, поэтому логично расположить в ней метод `startNewGame()`, начинающий новую игру. В качестве параметров метод должен принимать какие-то начальные настройки самой игры. Например, будут два режима игры, компьютер против игрока и игрок против игрока, размер поля, и сразу не будем привязываться к квадратному полю 3x3, для полей больше, чем 3x3 понадобится выигрышная длина, то есть число крестиков или ноликов, расположенных подряд на одной прямой для победы той или иной стороны. В теле метода сразу будет установлена так называемая заглушка, чтобы знать, что метод вызывается и все параметры передаются верно.

```
1 void startNewGame(int mode, int fSzX, int fSzY, int wLen) {
2     System.out.printf("Mode: %d;\nSize: x=%d, y=%d;\nWin Length: %d",
3         mode, fSzX, fSzY, wLen);
4 }
```

Листинг 11: Метод начала новой игры

Если сразу описать архитектуру проекта, его будет проще наполнять логикой и расширять, чем если писать последовательно, удерживая общую картину в голове. Итоговое приложение приложение будет работать в двух окнах: первое – стартовое, где будут задаваться настройки поля и производиться выбор режима игры; второе – основное, где будет происходить собственно игра. Основное окно уже написано, и при его закрытии происходит выход из программы. Для создания второго окна необходимо написать ещё один класс, названный, например, `SettingsWindow`, наследник `JFrame`. Конструктор второго окна будет принимать экземпляр игрового окна. В первую очередь это сделано для передачи параметров игры, а во-вторых, чтобы красиво отцентрировать его относительно основного.

```
1 package ru.gb.jdk.one.online;
2
3 import javax.swing.*;
4
5 public class SettingsWindow extends JFrame {
6     SettingsWindow(GameWindow gameWindow) {
7
8     }
9 }
```

Листинг 12: Заготовка окна настроек новой игры

В основном окне `GameWindow` понадобится два поля, одно класса `SettingsWindow` чтобы иметь возможность экземпляра этого окна показывать когда появится необходимость и второе – это панель `Map`. В основном окне при создании экземпляра окна настроек в него передаётся `this`.



Обратите внимание, на этот способ применять `this`, когда необходимо передать в метод ссылку на объект, который вызывает этот метод, фактически, основное окно передаёт себя.

```

1 Map map;
2 SettingsWindow settings;
3
4 GameWindow() {
5     setDefaultCloseOperation(EXIT_ON_CLOSE);
6     setLocation(WINDOW_POSX, WINDOW_POSY);
7     setSize(WINDOW_WIDTH, WINDOW_HEIGHT);
8     setTitle("TicTacToe");
9     setResizable(false);
10
11     map = new Map();
12     settings = new SettingsWindow(this);
13     // ...
14 }
```

Листинг 13: Создание окна настроек в основном окне

На рисунке 5 первый черновик диаграммы классов разрабатываемого приложения³. Создание идеальной диаграммы классов или модели данных не входит в цели курса, более важно понятное объяснение того, что в данный момент программируется. Буквами F обозначены экземпляры `JFrame`, буквой P `Jpanel`, а A это `Application`, то есть основной класс приложения. На диаграмме видно, что основное приложение создаёт основное окно, на которое добавлена панель `Map` и которое время от времени будет обращаться к `SettingsWindow` за настройками новой игры.

³ [Исходный код PlantUML](#)

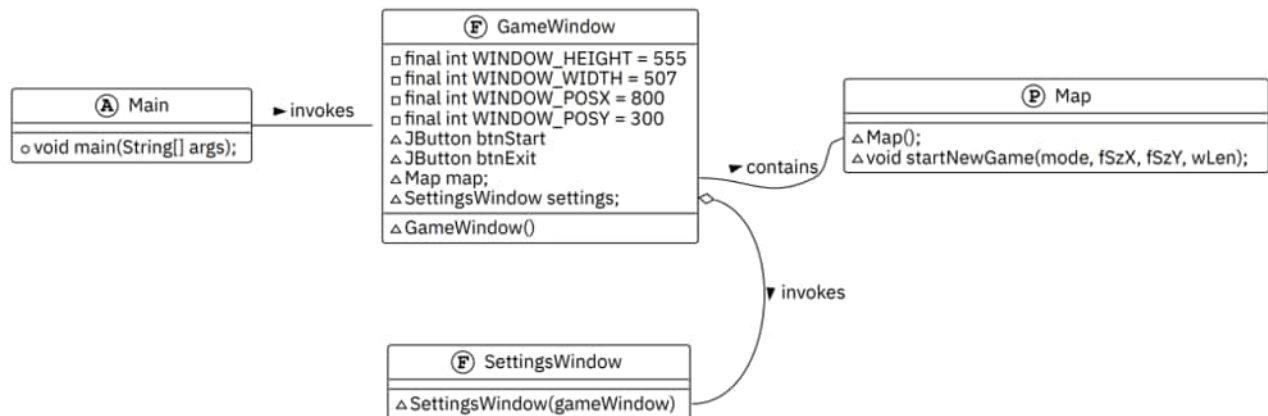


Рис. 5: Диаграмма классов приложения

6.4.2. Окно с настройками игры и обработчики кнопок

Окно настроек игры на данный момент будет представлено одной кнопкой старта игры, вызывающей метод старта игры с одним зафиксированным набором настроек – игра против компьютера, поле 3x3, чтобы выиграть необходимо собрать 3 крестика (или нолика) подряд. В данный момент окно создаётся в координатах (0,0) и имеет размер (0,0), то есть в левом верхнем углу экрана видно только кнопки свернуть, развернуть, закрыть. В конструкторе окна задаются его размеры и то, что его местоположение должно быть относительным главному окну. Аналогично основному окну добавлена кнопка подтверждения правильности настроек и старта игры.

```

1 public class SettingsWindow extends JFrame {
2     private static final int WINDOW_HEIGHT = 230;
3     private static final int WINDOW_WIDTH = 350;
4
5     JButton btnStart = new JButton("Start new game");
6     SettingsWindow(GameWindow gameWindow) {
7         setLocationRelativeTo(gameWindow);
8         setSize(WINDOW_WIDTH, WINDOW_HEIGHT);
9
10        add(btnStart);
11    }
12 }
  
```

Листинг 14: Окно настроек новой игры

Далее необходимо «оживить» кнопки на окнах, это делается специальной конструкцией, синтаксис которой пока что придётся запомнить. Синтаксически всё написанное уже понятно и оговорено – у объекта кнопки `btnExit` вызывается метод добавления к этому объекту некоторого слушателя действия. Какое может у кнопки

быть самое очевидное действие? Нажатие. В аргумент метода добавления передаётся некий новый объект класса «слушатель действия», у которого переопределяется метод «действие произошло». Кнопка старта игры будет делать видимым окно с будущими настройками. В листинге 15 показаны обработчики кнопок старта новой игры и завершения приложения, находящихся на основном окне программы. Эти обработчики необходимо поместить внутрь конструктора основного окна.

```

1 btnExit.addActionListener(new ActionListener() {
2     @Override
3     public void actionPerformed(ActionEvent e) {
4         System.exit(0);
5     }
6 });
7
8 btnStart.addActionListener(new ActionListener() {
9     @Override
10    public void actionPerformed(ActionEvent e) {
11        settings.setVisible(true);
12    }
13 });

```

Листинг 15: Обработчики нажатий на кнопки основного окна

6.4.3. Последовательность выполнения программы

В основном окне понадобится метод, инициализирующий новое игровое поле, поскольку прямой вызов по кнопке старта новой игры на окне настроек метода в панели основного окна противоречит инкапсуляции. Зачем мы так делаем, казалось бы усложняем? Но нет: панель находится на основном окне, а кнопка начала игры будет находиться на окне настроек, которое не может «знать», какие на основном окне есть панели. Или может оказаться, что дальше нет никакого интерфейса, а игра происходит по сети.



В этом и есть суть ООП, когда один объект максимально отделён от другого, и каждому из них вообще не важно, как реализован другой.

Соответственно, когда в окне настроек нажата кнопка «начать игру», обработчик вызывает метод главного окна, а главное окно в свою очередь уже знает, что оно разделено на панели, и вынуждает панель Мар начинать (рисунок 6)⁴. Для чего нужен промежуточный метод? Чтобы не делать лишних связей между классами. Это

⁴ Исходный код диаграммы в PlantUML

логично с точки зрения инкапсуляции. Одно окно не должно никак управлять панелью на другом окне.

```

1 SettingsWindow(GameWindow gameWindow) {
2     setLocationRelativeTo(gameWindow);
3     setSize(WINDOW_WIDTH, WINDOW_HEIGHT);
4     btnStart.addActionListener(new ActionListener() {
5         @Override
6         public void actionPerformed(ActionEvent e) {
7             gameWindow.startNewGame(0, 3, 3, 3);
8             setVisible(false);
9         }
10    });
11    add(btnStart);
12 }

```

Листинг 16: Обработчик кнопки старта новой игры

В окне настроек описан обработчик нажатия на единственную кнопку, из этого обработчика вызывается единственный доступный метод – «старт новой игры» на основном окне. По факту нажатия, также, целесообразно спрятать окно настроек. Из метода основного класса `startNewGame()` вызывается `map.startNewGame()` класса мэп.

```

1 void startNewGame(int mode, int fSzX, int fSzY, int wLen) {
2     map.startNewGame(mode, fSzX, fSzY, wLen);
3 }

```

Листинг 17: Цепочка вызовов методов старта новой игры

Ещё раз цепочка вызовов (рис. 6):

1. основное окно делает окно настроек видимым;
2. окно настроек говорит основному, что пора начинать игру;
3. основное окно в свою очередь знает, как именно надо игру начинать и просит панель стартовать. Если всё сделано верно, в терминале появится вывод из заглушки на панели Map. Mode: 0; Size: x=3, y=3; Win Length: 3

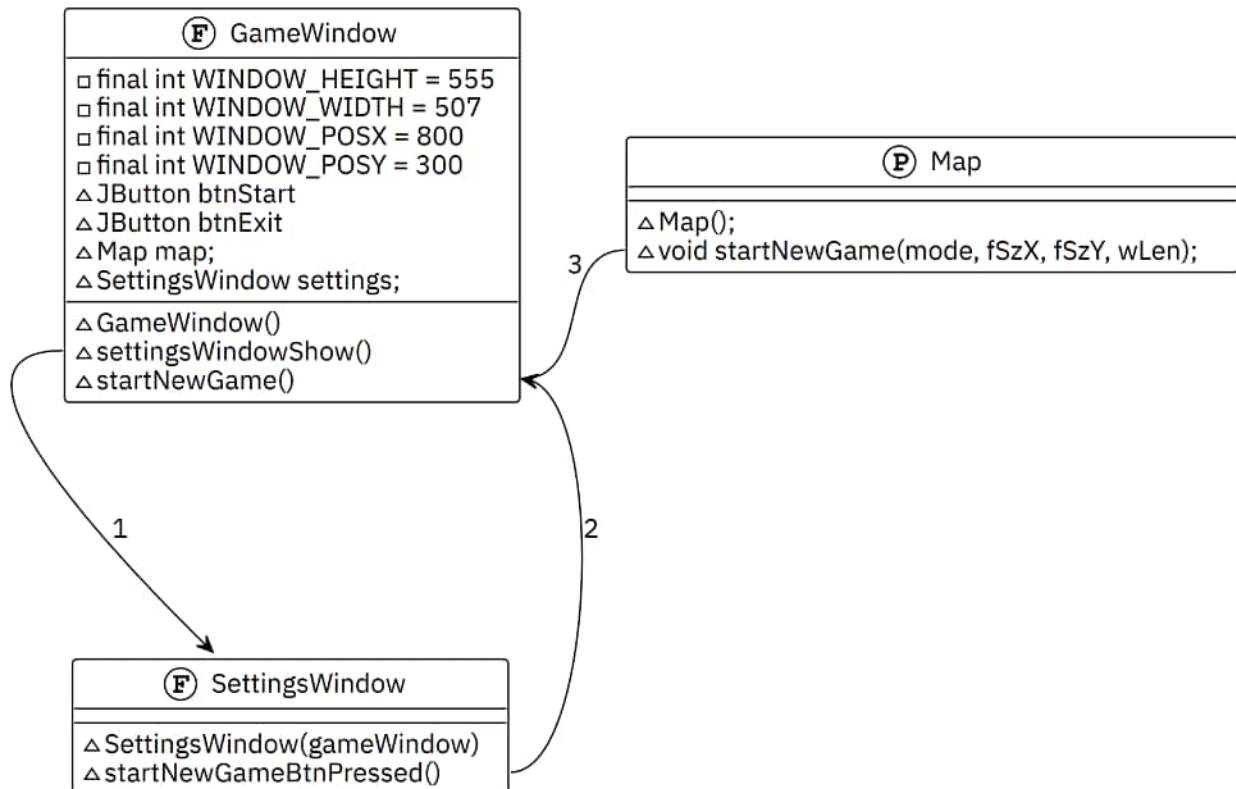


Рис. 6: Цепочка вызовов при старте новой игры

6.4.4. Вопросы для самопроверки

1. Менеджер размещений - это

- (a) сотрудник, занимающийся разработкой интерфейса;
- (b) объект, выполняющий расстановку компонентов на окне приложения;
- (c) механизм, проверяющий возможность отображения окна в ОС.

2. Экземпляр JPanel позволяет

- (a) применять комбинации из компоновщиков
- (b) добавить к интерфейсу больше компонентов
- (c) создавать группы компонентов
- (d) всё вышеперечисленное


3. Для выполнения кода по нажатию кнопки на интерфейсе нужно

- (a) создать обработчик кнопки и вписать код в него
- (b) переопределить метод нажатия у компонента кнопки
- (c) использовать специальный класс “слушателя” кнопок

6.5. Основная панель с игрой

6.5.1. Рисование

Далее всё будет происходить на панели с полем для игры. Для рисования самой панели фреймворком определён метод `paintComponent()`. Этот метод вызывается фреймворком когда что-то происходит, например, когда основное окно перекрывается другим, перемещается на другой экран, или если его развернуть из свёрнутого состояния, вызывается он гораздо реже, чем это необходимо для логики игры. Для описания игрового процесса необходимо перерисовывать компонент по каждому клику мышкой и по каждому действию оппонента.


 Важно помнить, что метод `paintComponent()` не следует напрямую вызывать из кода. Этот метод должен вызываться только фреймворком. Для того чтобы запросить у фреймворка вызов этого метода тоже есть специальный метод.

Для дальнейшей разработки важно отделить стандартный метод рисования компонента от пользовательского рисования на этом компоненте, так называемую бизнес-логику. Для этого будет создан ещё один метод `void render(Graphics g)`, который будет вызываться из переопределённого `paintComponent()`. из самого `paintComponent()` вызов метода родительского класса удалять не следует, поскольку там, скорее всего, происходит что-то важное. Для вызова же метода фреймворка, необходимо в нужный момент сказать фреймворку что требуется перерисовать панель, фреймворк поставит метод `paintComponent()` в очередь сообщений окна, и когда очередь дойдёт до выполнения этого метода – окно выполнит перерисовку.

```

1 @Override
2 protected void paintComponent(Graphics g) {
3     super.paintComponent(g);
4     render(g);
5 }
6
7 private void render(Graphics g) { }
```

Листинг 18: Отделение бизнес-логики рисования

 Это действие полностью асинхронно и косвенно зависит от наших вызовов

Чтобы рисовать нужен объект класса Graphics, который умеет рисовать геометрические фигуры, линии, текст и тому подобное. Чтобы нарисовать поле для игры понадобится ширина и высота поля в пикселях. Их возможно узнать из свойств панели – ширины и высоты. Всё, что связано с размерами лучше вынести в переменные объекта, поскольку они понадобятся в других методах. Помимо ширины и высоты понадобятся переменные, в которых будет храниться высота и ширина каждой ячейки. Размеры каждой ячейки пригодятся для создания отступа одной линии от другой. Далее циклически делаются отступы и рисуются горизонтальные и вертикальные линии.

```

1 private int panelWidth;
2 private int panelHeight;
3 private int cellHeight;
4 private int cellWidth;
5
6 private void render(Graphics g) {
7     panelWidth = getWidth();
8     panelHeight = getHeight();
9     cellHeight = panelHeight / 3;
10    cellWidth = panelWidth / 3;
11
12    g.setColor(Color.BLACK);
13    for (int h = 0; h < 3; h++) {
14        int y = h * cellHeight;
15        g.drawLine(0, y, panelWidth, y);
16    }
17    for (int w = 0; w < 3; w++) {
18        int x = w * cellWidth;
19        g.drawLine(x, 0, x, panelHeight);
20    }
21    repaint();
22 }

```

Листинг 19: Разлиновка поля для игры

У многих разработчиков, в зависимости от используемой операционной системы после запуска этого кода не полностью или вовсе не рисуется разлиновка, это происходит из-за асинхронности рисования, скорее всего метод с линиями отработал позже того, как Swing нарисовал панель.

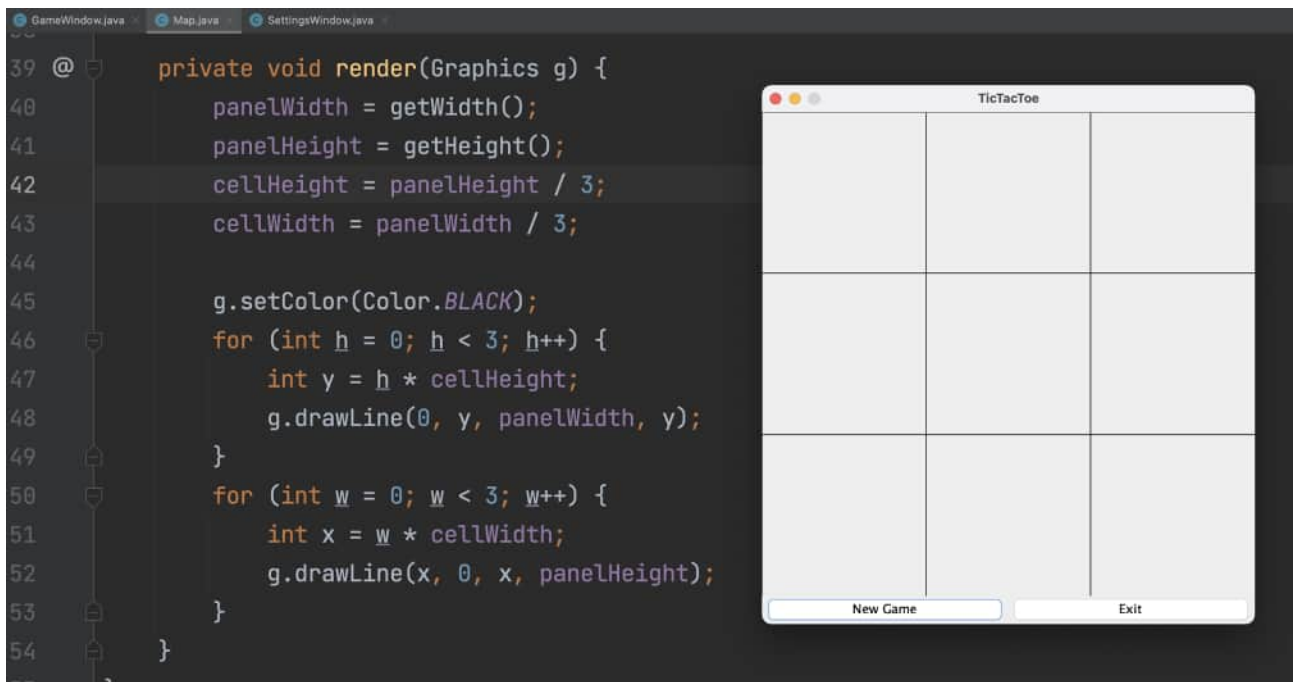


Рис. 7: Результат разлиновки поля для игры

Чтобы всё увидеть, необходимо заставить компонент панели полностью перерисоваться. Это делается вызовом метода `repaint()` из метода старта новой игры.

6.5.2. Обработчик мышки

Обработчик действий мышки очень похож на те обработчики, которые уже написаны. В конструкторе панели описывается метод добавления слушателя, в котором переопределяется метод `mouseReleased()`, то есть для приложения важно когда пользователь отпустит кнопку и аналогично методу отрисовки следует сразу отделить обработчик от основной исполняемой логики.

```

1 Map() {
2     addMouseListener(new MouseAdapter() {
3         @Override
4         public void mouseReleased(MouseEvent e) {
5             update(e);
6         }
7     });
8 }

```

Листинг 20: Обработчик отпускания кнопки мышки

Внутри метода обновления также принудительно вызывается метод перерисовки компонента, чтобы получился игровой цикл: старт – отрисовка – клик мыши – отрисовка – клик – отрисовка...

```

1 private void update(MouseEvent e) {
2     int cellX = e.getX() / cellWidth;
3     int cellY = e.getY() / cellHeight;
4     System.out.printf("x=%d, y=%d\n", cellX, cellY);
5     repaint();
6 }

```

Листинг 21: Обработчик отпускания кнопки мышки

В методе обновления из объекта `MouseEvent` получаются координаты клика, делятся на размер ячейки и тем самым получается номер ячейки, в которую произошёл клик.

6.5.3. Логика игры

Поскольку лекция про графические интерфейсы и ООП, а все используемые конструкции примитивны, код логики игры будет приведён без подробных пояснений.

Для работы понадобится генератор псевдослучайных чисел, символы, которыми будет обозначаться на поле игрок, компьютер и пустая ячейка, собственно поле и его размеры. Размеры – на будущее.

```

1 private static final Random RANDOM = new Random();
2 private final int HUMAN_DOT = 1;
3 private final int AI_DOT = 2;
4 private final int EMPTY_DOT = 0;
5 private int fieldSizeY = 3;
6 private int fieldSizeX = 3;
7 private char[][] field;

```

Листинг 22: Субъекты игры

Метод инициализации поля – создаётся новый массив и заполняется пустыми символами. Его вызов логично сразу разместить в метод старта новой игры.

```

1 private void initMap() {
2     fieldSizeY = 3;
3     fieldSizeX = 3;
4     field = new char[fieldSizeY][fieldSizeX];
5     for (int i = 0; i < fieldSizeY; i++) {
6         for (int j = 0; j < fieldSizeX; j++) {
7             field[i][j] = EMPTY_DOT;
8         }
9     }
10 }

```

Листинг 23: Инициализация карты

Когда кто-то (игрок или компьютер) будет совершать ход, будет важно, попал ли игрок в какую-то ячейку поля и пуста ли она, потому что нельзя ставить крестик поверх нолика и наоборот.

```

1 private boolean isValidCell(int x, int y) {
2     return x >= 0 && x < fieldSizeX && y >= 0 && y < fieldSizeY;
3 }
4
5 private boolean isEmptyCell(int x, int y) {
6     return field[y][x] == EMPTY_DOT;
7 }

```

Листинг 24: Попал ли игрок в пустую ячейку и в поле

Компьютер будет очень примитивный – он будет делать ход в случайные места на карте.

```

1 private void aiTurn() {
2     int x, y;
3     do {
4         x = RANDOM.nextInt(fieldSizeX);
5         y = RANDOM.nextInt(fieldSizeY);
6     } while (!isEmptyCell(x, y));
7     field[y][x] = AI_DOT;
8 }

```

Листинг 25: Ход компьютера

Очевидно, что учебные цели предполагают не только демонстрацию того, как надо делать, но также и демонстрацию того как делать не надо. Далее приведён код, который не следует допускать при работе над приложениями. Метод принимает на вход символ, который нужно проверить и проверяет - не победил ли он.

```

1 private boolean checkWin(char c) {
2     if (field[0][0]==c && field[0][1]==c && field[0][2]==c) return true;
3     if (field[1][0]==c && field[1][1]==c && field[1][2]==c) return true;
4     if (field[2][0]==c && field[2][1]==c && field[2][2]==c) return true;
5
6     if (field[0][0]==c && field[1][0]==c && field[2][0]==c) return true;
7     if (field[0][1]==c && field[1][1]==c && field[2][1]==c) return true;
8     if (field[0][2]==c && field[1][2]==c && field[2][2]==c) return true;
9
10    if (field[0][0]==c && field[1][1]==c && field[2][2]==c) return true;
11    if (field[0][2]==c && field[1][1]==c && field[2][0]==c) return true;
12    return false;
13 }

```

Листинг 26: Проверка победы одного из игроков

🔥 Всегда пишите с помощью циклов, потому что стоит захотеть изменить размер поля на 4x4 или 5x5 – размер и сложность этого метода будет расти в геометрической прогрессии.

И метод проверки поля на состояние ничьей. Ничья в крестиках-ноликах наступает, когда не победил ни игрок ни оппонент, и не осталось пустых клеток.

```

1 private boolean isMapFull() {
2     for (int i = 0; i < fieldSizeY; i++) {
3         for (int j = 0; j < fieldSizeX; j++) {
4             if (field[i][j] == EMPTY_DOT) return false;
5         }
6     }
7     return true;
8 }

```

Листинг 27: Проверка на ничью

Вся дальнейшая работа будет сконцентрирована на методах обновления игрового состояния и отрисовки игрового поля. В результате клика в ячейку необходимо проверить, валидная-ли ячейка, и можно-ли туда ходить. Если какое-то из условий не прошло, клик игнорируется, а если всё хорошо – делается ход.

```

1 int cellX = e.getX()/cellWidth;
2 int cellY = e.getY()/cellHeight;
3 if (!isValidCell(cellX, cellY) || !isEmptyCell(cellX, cellY)) return;
4 field[cellY][cellX] = HUMAN_DOT;
5
6 repaint();

```

Листинг 28: Ход игрока

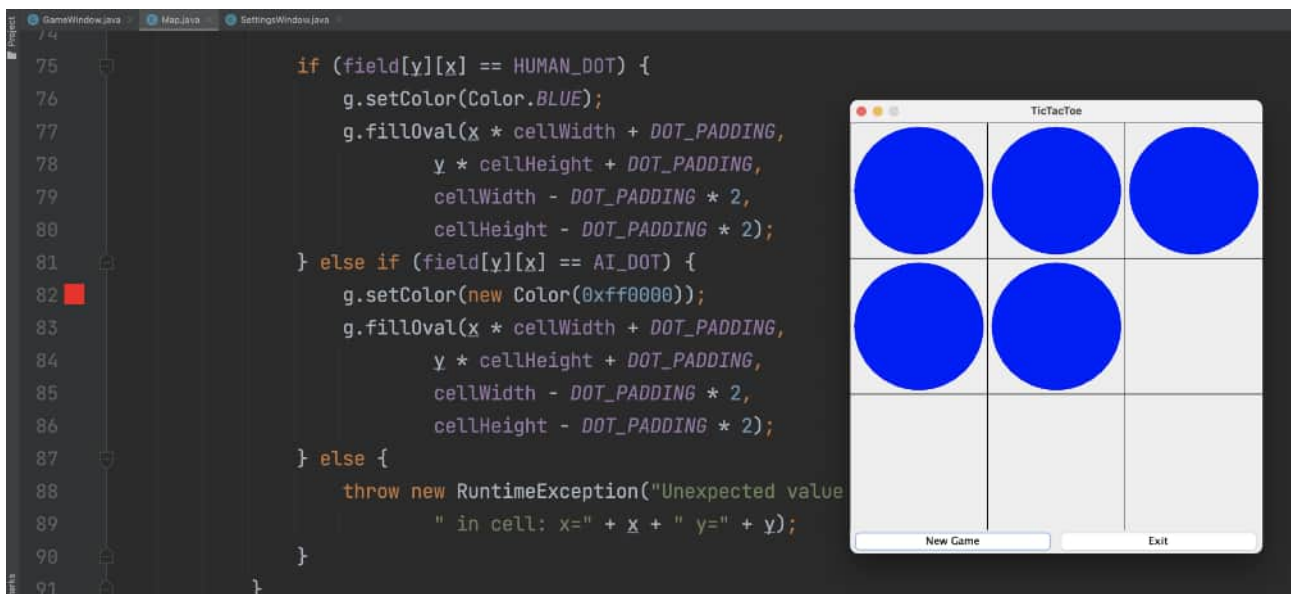
```

1  for (int y = 0; y < fieldSizeY; y++) {
2      for (int x = 0; x < fieldSizeX; x++) {
3          if (field[y][x] == EMPTY_DOT) continue;
4
5          if (field[y][x] == HUMAN_DOT) {
6              g.setColor(Color.BLUE);
7              g.fillOval(x * cellWidth + DOT_PADDING,
8                      y * cellHeight + DOT_PADDING,
9                      cellWidth - DOT_PADDING * 2,
10                     cellHeight - DOT_PADDING * 2);
11          } else if (field[y][x] == AI_DOT) {
12              g.setColor(new Color(0xff0000));
13              g.fillOval(x * cellWidth + DOT_PADDING,
14                      y * cellHeight + DOT_PADDING,
15                      cellWidth - DOT_PADDING * 2,
16                      cellHeight - DOT_PADDING * 2);
17          } else {
18              throw new RuntimeException("Unexpected value " + field[y][x] +
19                                     " in cell: x=" + x + " y=" + y);
20          }
21      }
22  }

```

Листинг 29: Отрисовка поля

Далее – непосредственно отрисовка. Сюда можно картинку вставлять, закрашивать квадраты, рисовать крестики и нолики. Для простоты будут рисоваться кружки. Методу объекта графики `g.fillOval()` в сигнатуре передаётся левая верхняя координата прямоугольника, в который затем будет вписан овал, его ширина и высота соответственно. Чтобы задать цвет – перед тем как рисовать необходимо изменить цвет объекта графики `g.setColor(Color.BLUE)`. Для человека далее будут рисоваться синие кружки, а для компьютера красные.



6.5.4. Последние приготовления

По сути, осталось сделать две вещи – описать так называемую бизнес-логику, то есть в правильном порядке вызвать методы с логикой игры, избавиться от исключений и вывести сообщение об окончании игры. Для того, чтобы вывести результат, поверх игрового поля будет выводиться сообщение.

```

1 private int gameOverType;
2 private static final int STATE_DRAW = 0;
3 private static final int STATE_WIN_HUMAN = 1;
4 private static final int STATE_WIN_AI = 2;
5
6 private static final String MSG_WIN_HUMAN = "Победил игрок!";
7 private static final String MSG_WIN_AI = "Победил компьютер!";
8 private static final String MSG_DRAW = "Ничья!";

```

Листинг 30: Состояния игрового поля

В методе обновления уточняется, что когда пользователь поставил точку, необходимо проверить состояние поля на наличие победы или ничьей, дать возможность компьютеру поставить точку и сделать тоже самое.

```

1 // update
2 if (checkEndGame(HUMAN_DOT, STATE_WIN_HUMAN)) return;
3
4 aiTurn();
5 repaint();
6 if (checkEndGame(AI_DOT, STATE_WIN_AI)) return;
7 // end update
8
9 private boolean checkEndGame(int dot, int gameOverType) {
10     if (checkWin(dot)) {
11         this.gameOverType = gameOverType;
12         repaint();
13         return true;
14     }
15     if (isMapFull()) {
16         this.gameOverType = STATE_DRAW;
17         repaint();
18         return true;
19     }
20     return false;
21 }

```

Листинг 31: Логика обновления

В методе рендеринга, как только поле выведено и если игра закончилась необходимо вывести сообщение с одним из вариантов исхода игры. Для упрощения также следует завести классовую переменную с признаком окончания игры. Метод окончания игры рисует тёмно серый прямоугольник с жёлтой надписью о победе одного из игроков или ничьей в зависимости от состояния.

```

1  //render
2  if (isGameOver) showMessageGameOver(g);
3  // end render
4
5  private void showMessageGameOver(Graphics g) {
6      g.setColor(Color.DARK_GRAY);
7      g.fillRect(0, 200, getWidth(), 70);
8      g.setColor(Color.YELLOW);
9      g.setFont(new Font("Times new roman", Font.BOLD, 48));
10     switch (gameOverType) {
11         case STATE_DRAW:
12             g.drawString(MSG_DRAW, 180, getHeight() / 2); break;
13         case STATE_WIN_AI:
14             g.drawString(MSG_WIN_AI, 20, getHeight() / 2); break;
15         case STATE_WIN_HUMAN:
16             g.drawString(MSG_WIN_HUMAN, 70, getHeight() / 2); break;
17         default:
18             throw new RuntimeException("Unexpected gameOver state: " + gameOverType);
19     }
20 }

```

Листинг 32: Отрисовка сообщения об окончании игры

Далее в листинге 33 приведены несколько мелких правок в методах. Прочитав текст исключения при старте приложения становится ясно, что оно возникает, когда программа не может что-то поделить на ноль. Размеры поля до их инициализации равны нулю, поэтому понадобится ещё одна булева переменная – инициализирована ли игра.

- В конструкторе панели поле не инициализировано;
- В методе обновления нет смысла обрабатывать клики по неинициализированному полю или полю на котором закончилась игра;
- при старте новой игры игра перестаёт быть законченной, а поле становится инициализированным;
- рендеринг на неинициализированном поле не имеет смысла;
- в методе проверки на победу нужно добавить присвоение истинности булевой переменной с фактом окончания игры.

```

1 private boolean isGameOver;
2 private boolean isInitialized;
3
4 Map() {
5     isInitialized = false;
6 }
7
8 private void update(MouseEvent e) {
9     if (isGameOver || !isInitialized) return;
10 }
11
12 void startNewGame(int mode, int fSzX, int fSzY, int wLen) {
13     isGameOver = false;
14     isInitialized = true;
15 }
16
17 private void render(Graphics g) {
18     if (!isInitialized) return;
19 }
20
21 private boolean checkEndGame(int dot, int gameOverType) {
22     if (checkWin(dot)) {
23         isGameOver = true;
24     }
25     if (isMapFull()) {
26         isGameOver = true;
27     }
28     return false;
29 }

```

Листинг 33: Незначительные добавления в методах панели

Результат запуска получившегося приложения представлен на рисунке 8.



(a) Победа компьютера



(b) Победа игрока



(c) Ничья

Рис. 8: Результаты игры

Практическое задание

1. Полностью разобраться с кодом.

2. Переделать проверку победы, чтобы она не была реализована просто набором условий.
3. Попробовать переписать логику проверки победы, чтобы она работала для поля 5x5 и количества фигур 4.
4. ** Доработать искусственный интеллект, чтобы он мог примитивно блокировать ходы игрока, и примитивно пытаться выиграть сам.

Термины, определения и сокращения

Swing библиотека для создания графического интерфейса для программ на языке Java. Swing разработан компанией Sun Microsystems и содержит ряд графических компонентов (Swing widgets), таких как кнопки, поля ввода, таблицы и тому подобные.

Асинхронность вид программирования, позволяющий вынести выполняемые задачи отдельными блоками кода. Применяется в сервисах, где предыдущее действие тормозит следующее. Синхронный процесс выполняется поэтапно. Пользователь совершает действие, ждёт, пока программа обработает часть кода, переходит к другому блоку. При использовании асинхронности, код убирает операцию, блокирующую следующие действия.

Компонент независимый модуль программы, предназначенный для повторного использования. Часто компоненты объединяют по общим признакам и организуют в соответствии с определёнными правилами и ограничениями. Например, компоненты графического интерфейса.

Компоновщик (layout manager) в библиотеке Java Swing отвечает за расположение и организацию компонентов (например, кнопок, текстовых полей, панелей). Он определяет, как компоненты будут выравниваться, размещаться и изменяться при изменении размеров окна.

Обработчик это механизм, с помощью которого приложение может перехватить события, такие как сообщения, действия мыши и нажатия клавиш. Функция, которая перехватывает события определенного типа, называется процедурой-обработчиком. Процедура-обработчик может действовать для каждого получаемого события, а затем изменить или отменить событие.

Окно это прямоугольная область экрана, в котором приложение отображает информацию и получает реакцию от пользователя. Одновременно на экране может отображаться несколько окон, в том числе, окон других приложений, однако лишь одно из них может получать реакцию от пользователя – активное окно.

Пользователь использует клавиатуру, мышь и прочие устройства ввода для взаимодействия с приложением, которому принадлежит активное окно. Панель

Панель в библиотеке Java Swing представляет собой контейнер, который используется для группировки и организации других компонентов в пользовательском интерфейсе. Она представляет собой область с фиксированным размером на которую можно добавить другие компоненты, такие как кнопки, текстовые поля или изображения.

Параллельность способ организации компьютерных вычислений, при котором программы разрабатываются, как набор взаимодействующих вычислительных процессов, работающих асинхронно и при этом одновременно. Параллельное программирование – это техника программирования, которая использует преимущества многоядерных или многопроцессорных компьютеров и является подмножеством более широкого понятия многопоточности (multithreading).

События это действия или случаи, возникающие в программируемой системе, о которых система сообщает для того, чтобы было возможно с ними взаимодействовать. Например, если пользователь нажимает кнопку на графическом интерфейсе, возможно ответить на это действие, отобразив информационное окно.