**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ**

**УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»**

**(СПбГУТ)**

Факультет: Инфокоммуникационных сетей и систем

Кафедра: Защищенных систем связи

Дисциплина: Разработка защищенных сетевых приложений**)**

**Отчет по курсовой работе**

«Разработка приложения по расчету оплаты за время нахождения в антикафе»

Выполнил студент группы:

Шишикин Иван Александрович, ИБС-23

Проверил преподаватель:

Шариков Павел Иванович, ассистент кафедры ЗСС

**Содержание**

[Введение 3](#_Toc466903759)

1. [Формулировка поставленной задачи 4](#_Toc466903760)

2. [Описание библиотек](#_Toc466903762) 4

2.1. “javax.swing” [4](#_Toc466903762)

2.2. “java.awt” [6](#_Toc466903762)

2.3. [“java.time” 8](#_Toc466903762)

2.4. "[java.util.List” и “java.util.ArrayList” 11](#_Toc466903762)

2.4.1. [“java.util.List” 12](#_Toc466903762)

2.4.2. [“java.util.ArrayList” 13](#_Toc466903762)

3. [Структурное описание 14](#_Toc466903763)

3.1. [Класс “Table” 14](#_Toc466903762)

3.2. [Класс “Main” 20](#_Toc466903762)

3.3. [Класс “Menu\_GUI” 23](#_Toc466903762)

3.4. [Класс “Statistics\_GUI” 26](#_Toc466903762)

3.5. [Класс “Tables\_GUI” 33](#_Toc466903762)

4. [Тестинг программы 38](#_Toc466903764)

[Заключение 43](#_Toc466903770)

[Список используемых источников 44](#_Toc466903771)

**Введение**

В наше время антикафе становятся неотъемлемой частью городской культуры, предоставляя уникальное пространство для неформальных встреч и творческого времяпрепровождения. С увеличением популярности таких заведений возникает потребность в эффективном управлении и оптимизации бизнес-процессов. В этом контексте разработка приложения на языке программирования Java для управления антикафе приобретает особую актуальность.  
**Целью** данной работы является разработка приложения, способного автоматизировать процессы управления антикафе и обеспечить эффективный расчет оплаты за время пребывания посетителей.  
**Задачи:**

1. Выбор столика: реализовать функционал выбора столика для новых посетителей, без учёта их количества.
2. Отслеживание времени: создать систему отслеживания времени пребывания гостей, фиксируя момент начала и завершения посещения.
3. Статистика по текущему состоянию:

* Предоставить информацию о занятых столиках.
* Отображать время, проведенное гостями за каждым столиком.
* Рассчитать суммарную стоимость на текущий момент.

1. Статистика по архивным данным:

* Отображение общей выручки заведения.
* Расчет среднего времени, в течение которого столик был занят. Определение предпочтительных и прибыльных столиков.

**Актуальность** данного исследования обусловлена динамичным развитием сферы общественного питания и необходимостью внедрения инноваций для оптимизации бизнес-процессов. Приложение, созданное на языке Java, может служить мощным инструментом для управления и анализа, улучшая общий опыт посетителей и способствуя успешному развитию бизнеса в данной области. В результате использования разработанного приложения, антикафе сможет повысить операционную эффективность, улучшить качество обслуживания и укрепить свою конкурентоспособность в современной рыночной среде.

1. **Формулировка поставленной задачи**

В антикафе 10 столиков. Задача включает в себя разработку приложения с интерфейсом, которое сможет обеспечить пользователя возможностью отметить стол как занятый или освободить его, если он уже занят. Приложение должно обеспечивать возможность просмотра статистики в целом и для каждого стола отдельно, а также рассчитывать, какой из столиков чаще всего выбирали, сколько в среднем он был занят по времени и сколько он принёс прибыли за всё время.

1. **Описание библиотек**
   1. **“*javax.swing*”**

Библиотека “*javax.swing*” является частью “*Java API*” и предоставляет мощные инструменты для создания графического интерфейса пользователя (GUI) в приложениях, разработанных на языке программирования Java.

Включая различные классы и методы, “*javax.swing*” обеспечивает разработчиков возможностью создавать привлекательные и функциональные пользовательские интерфейсы. Она предлагает готовые компоненты, такие как окна, кнопки, поля ввода, таблицы, списки и другие, которые могут быть легко настроены и использованы в приложении.

Класс “*JFrame*” является ключевым компонентом библиотеки “*javax.swing*” и представляет собой основное окно приложения. Он предоставляет возможности для установки заголовка окна, управления его размерами и расположением на экране. Также “*JFrame*” обрабатывает события, связанные с окном, такие как закрытие или сворачивание.

Класс “*JButton*” представляет собой компонент кнопки в пользовательском интерфейсе. Он позволяет добавлять кнопки с текстом и устанавливать обработчики событий для реагирования на нажатия. Кнопки широко используются для инициирования действий и взаимодействия с пользователем.

Компонент ”*JTextField*” представляет текстовое поле, в которое пользователь может вводить текст. Он предоставляет методы для получения введенного текста и установки ограничений на ввод. ”*JTextField*” может быть использован для сбора информации от пользователя или выполнения поиска.

“*JTable*” является компонентом, позволяющим отображать данные в виде таблицы с рядами и столбцами. Он предоставляет методы для заполнения таблицы данными, а также функциональность для редактирования и сортировки данных. “*JTable*” является чрезвычайно полезным инструментом при работе с большими объемами информации или просмотре результатов запросов к базе данных.

“*JLabe*”: Этот компонент используется для отображения текста или изображения, предоставляя возможность создания меток или надписей в GUI. “*JLabe*” может быть настроен для отображения текста разных размеров и стилей, а также изображений из файлов или ресурсов приложения.

“*JCheckBox*”: Этот компонент представляет собой флажок, который может быть установлен или снят пользователем. Используется, когда нужно предоставить пользователю возможность выбрать одну или несколько опций из предложенного списка.

“*JRadioButton*” и “*JRadioButtonGroup*”: “*JRadioButton*” также представляет собой флажок, но в отличие от “*JCheckBox*”, он предназначен для выбора только одной опции из группы связанных радиокнопок. “*JRadioButtonGroup*” используется для группировки радиокнопок, чтобы они работали вместе и обеспечивали взаимоисключающий выбор.

“*JComboBox*”: Этот компонент представляет раскрывающийся список, который позволяет пользователю выбирать одну опцию из предложенного набора. “*JComboBox*” может быть заполнен заранее заданными элементами или динамически обновляться на основе введенных данных.

“*JTextArea*”: Это компонент многострочного текстового поля, которое позволяет пользователю вводить и редактировать текст большого объема. “*JTextArea*” поддерживает прокрутку и может быть настроен с определенным количеством строк и столбцов.

“*JScrollPane*”: Этот компонент представляет собой контейнер с прокручиваемой областью, который может использоваться для добавления прокрутки к компонентам, например, к таблицам или текстовым областям, если их содержимое не помещается на экране.

Библиотека “*javax.swing*” также поддерживает много других компонентов, интерфейсов и классов, предоставляющих различные возможности создания и управления GUI-элементами.

Компоненты можно кастомизировать путем изменения атрибутов, добавления слушателей событий и настройки внешнего вида с помощью платформонезависимой системы раскраски (Look and Feel).

В заключение, библиотека “*javax.swing*” является важным инструментом для разработчиков Java, давая им возможность создавать красочные и интерактивные пользовательские интерфейсы. Она обеспечивает гибкость и расширяемость, позволяя создавать приложения, которые соответствуют требованиям и предпочтениям пользователей.

* 1. **“*java.awt*”**

“*java.awt*” (“Abstract Window Toolkit”) - это библиотека, предоставляемая платформой Java, которая представляет основные инструменты для создания графического пользовательского интерфейса (GUI). Она обеспечивает набор классов и методов для рисования элементов интерфейса, управления событиями и компоновки компонентов на экране.

Некоторые основные компоненты и классы “*java.awt*” включают:

“*Component*”: Это базовый класс для всех графических компонентов, таких как кнопки, текстовые поля, панели и другие элементы интерфейса. Он предоставляет основные методы для рисования, обработки событий и управления компонентами.

“*Component*”: Этот класс представляет контейнер, который может содержать другие компоненты “*Component*”. Он помогает в организации компонентов в иерархическую структуру и управлении их компоновкой на экране.

“*Graphics*”: Этот класс предоставляет методы для рисования примитивов, текста и изображений на компонентах и контейнерах. Он используется для создания пользовательской графики и кастомных элементов интерфейса.

“*Layout Manager*”: Это интерфейс и набор классов, которые определяют различные алгоритмы компоновки компонентов внутри контейнера. Они обеспечивают автоматическую адаптацию компонентов к изменениям размеров контейнера и организацию элементов интерфейса в соответствии с заданными правилами.

“*Event*”: Этот класс и связанные классы предоставляют механизмы для обработки и генерации событий в интерфейсе. Он позволяет реагировать на пользовательские действия, такие как щелчок мыши или нажатие клавиши.

Рисование и графика: “*java.awt*” предоставляет классы и методы для создания элементов графики, таких как линии, формы, изображения и текст. Есть возможность рисовать примитивные формы, устанавливать цвета и шрифты, а также создавать пользовательские компоненты с использованием методов рисования.

Управление событиями: “*java.awt*” включает механизмы для обработки событий пользовательского взаимодействия, таких как щелчки мыши, нажатия клавиш и перемещения указателя. Возможно, регистрировать слушателей событий и реагировать на них, выполняя определенные действия или обновляя элементы интерфейса.

Компоновка компонентов: “*java.awt*” предоставляет базовые механизмы для управления компоновкой компонентов на экране. Предоставляет возможность использовать различные менеджеры компоновки, такие как “*FlowLayout*”, “*BorderLayout*” и “*GridLayout*”, чтобы организовать компоненты в структурированный и автоматически адаптирующийся макет.

Графические преобразования и анимация: “*java.awt*” поддерживает различные графические преобразования, такие как масштабирование, поворот и смещение, а также предоставляет возможности для создания простых анимаций. Существует возможность изменять размеры и положение компонентов, создавать эффекты движения и создавать преобразования графических объектов.

Работа с изображениями: “*java.awt*” предоставляет классы для загрузки, отображения и манипулирования изображениями. Существует возможность использовать эти классы для работы с различными форматами изображений, изменения их размеров, применения эффектов и рисования изображений на компонентах интерфейса.

Библиотека “*java.awt*” предоставляет базовый набор инструментов для создания простых графических интерфейсов в Java. Однако, для более продвинутого и богатого визуального опыта, рекомендуется использовать более новую и мощную библиотеку “*java.swing*”, которая основана на “*java.awt*” и предоставляет больше функций.

* 1. **“*java.time*”**

“*java.time*” вводит новую модель для работы с датами и временем в языке Java. Библиотека разработана таким образом, чтобы обеспечить простое и интуитивно понятное API для выполнения частых операций, связанных с датами и временем, таких как создание, форматирование, сравнение и арифметические операции.

Основные классы, предоставляемые библиотекой, включают:

“*LocalDate*”: Представляет только дату без времени и часового пояса, например, 2021-09-30. Этот класс предоставляет методы для получения информации о дне, месяце, годе, а также для выполнения операций с датами, таких как прибавление/вычитание дней, месяцев или лет.

“*LocalTime*”: Представляет только время без даты и часового пояса, например, 10:15:30. Класс “*LocalTime*” предоставляет методы для получения часов, минут, секунд и миллисекунд времени, а также для выполнения операций, таких как прибавление/вычитание времени или форматирование.

“*LocalDateTime*”: Класс “*LocalDateTime*” комбинирует дату и время без учета часового пояса, например, 2021-09-30T10:15:30. Он предоставляет удобное API для работы с датой и временем одновременно, а также для выполнения операций с периодами и интервалами времени.

“*Instant*”: Представляет момент времени в мировом координированном времени (UTC). Он может быть использован для измерения времени с высокой точностью и для сравнения моментов времени.

“*Duration*” и “*Period*”: Классы “*Duration*” и “*Period*” используются для представления временных интервалов и периодов. “*Duration*” предназначен для точного измерения продолжительности времени в часах, минутах, секундах и миллисекундах, а “*Period*” представляет периоды в годах, месяцах и днях.

“*ZonedDateTime*”: Представляет комбинацию даты, времени и часового пояса. Этот класс позволяет выполнять операции с датой и временем с учетом часовых поясов, таких как преобразования между разными часовыми поясами или вычисление разницы между двумя моментами времени в разных часовых поясах.

“*ZoneId*” и “*ZoneOffset*”: “*ZoneId*” представляет часовой пояс и предоставляет информацию о смещении относительно времени UTC, а “*ZoneOffset*” представляет только смещение времени относительно UTC без информации о часовом поясе.

Immutable (Неизменяемость): Классы в “*java.time*” являются неизменяемыми, что означает, что они не могут быть изменены после создания. Вместо этого, операции, modifing (модифицирования) возвращают новые экземпляры класса с изменениями. Это важно при работе с датами и временем, потому что гарантирует, что объекты будут безопасными для использования в многопоточной среде и предотвращает неожиданное изменение значений.

Поддержка форматирования: Библиотека “*java.time*” предоставляет классы для форматирования даты и времени. С помощью класса “*DateTimeFormatter*” возможно форматировать даты и времена в соответствии с заданным шаблоном или преобразовывать строки в объекты классов “*LocalDate*”, “*LocalTime*” и т.д.

Поддержка сериализации: Классы в “*java.time*” могут быть сериализованы и десериализованы с использованием стандартных механизмов сериализации в Java. Это позволяет сохранять объекты даты и времени в файлы, передавать их по сети и использовать в различных реализациях приложений.

Большой выбор операторов: Библиотека “*java.time*” предлагает широкий спектр операторов для выполнения различных операций с датами и временем. Некоторые из них включают вычисление разницы между двумя моментами времени, прибавление или вычитание периода из даты или времени, операции сравнения, извлечение конкретных значений, проверку на попадание в определенный диапазон и многое другое.

Множество дополнительных классов: Кроме основных классов, описанных выше, библиотека “*java.time*” также предоставляет ряд дополнительных классов, которые облегчают работу с датами и временем. Некоторые из них включают “*Year*”, “*Month*”, “*YearMonth*”, “*MonthDay*”, “*Instant*”, “*OffsetDateTime*” и другие.

“*java.time*” также обеспечивает множество методов для выполнения операций с датами и временем, таких как проверка на попадание в високосный год, вычисление разницы между датами и временем, форматирование и парсинг строкового представления даты и времени, работа с различными часовыми поясами и другие.

Благодаря новой модели библиотеки “*java.time*” код, связанный с датами и временем, становится более простым, понятным и безопасным. Это улучшает производительность и разработку приложений, требующих работы с датами и временем.

* 1. **“*java.util.List*” и “java.util.ArrayList”**

являются часто используемыми инструментами в программировании на языке Java. Они предоставляют удобные средства для работы со списками данных и обеспечивают динамическое управление их размером.

“*ArrayList*” является реализацией интерфейса “*List*” и представляет собой динамический массив. Его главное преимущество заключается в том, что он автоматически расширяется или сжимается в зависимости от количества добавленных или удаленных элементов. “*ArrayList*” позволяет добавлять, получать, изменять и удалять элементы по индексу, а также выполнять другие операции со списком, такие как проверка наличия элемента, определение размера списка и создание массива на основе списка.

Интерфейс “*List*” определяет основные методы и свойства, присущие спискам данных. Вместе с “*ArrayList*”, этот интерфейс предоставляет гибкость и множество функций для работы с упорядоченными списками. “*List*” дополнительно обеспечивает возможность создания подсписков, поиск элементов по значению, а также сортировку списка.

Обе библиотеки широко используются в Java-проектах всех типов и масштабов. Они предоставляют удобные методы для управления списками данных, что делает их особенно полезными в сценариях, где требуется динамическое добавление, удаление и изменение элементов списка. “*ArrayList*” и “*List*” также эффективно работают с большими объемами данных, обеспечивая высокую производительность.

* + 1. **“*java.util.List*”**

“*List*” является интерфейсом в Java и представляет собой коллекцию элементов, упорядоченных по индексам. Он расширяет интерфейс “*Collection*” и определяет дополнительные операции для работы с упорядоченным списком.

Методы:

*add(E element)*: Добавляет элемент заданного типа E в конец списка.

*add(int index, E element)*: Вставляет элемент на указанной позиции в списке.

*get(int index)*: Возвращает элемент по указанному индексу.

*set(int index, E element)*: Заменяет элемент на указанной позиции новым элементом.

*remove(int index)*: Удаляет элемент по указанному индексу.

*size()*: Возвращает текущий размер списка.

*isEmpty()*: Проверяет, является ли список пустым.

*clear()*: Удаляет все элементы из списка.

*contains(Object obj)*: Проверяет, содержится ли указанный объект в списке.

*indexOf(Object obj)*: Возвращает индекс первого вхождения указанного объекта в списке.

*lastIndexOf(Object obj)*: Возвращает индекс последнего вхождения указанного объекта в списке.

*subList(int fromIndex, int toIndex)*: Возвращает новый список, содержащий элементы от “*fromIndex*” до “*toIndex*” (не включая “*toIndex*”).

* + 1. **“*java.util.ArrayList*”**

“*ArrayList*” является реализацией интерфейса “*List*” в Java и представляет собой динамический массив, который может изменять свой размер по мере добавления и удаления элементов. В отличие от обычного массива, “*ArrayList*” автоматически расширяет или сжимает свою емкость при необходимости.

Методы:

*add(E element)*: Добавляет элемент заданного типа E в конец списка “*ArrayList*”.

*add(int index, E element)*: Вставляет элемент на указанной позиции в списке “*ArrayList*”.

*get(int index)*: Возвращает элемент по указанному индексу.

*set(int index, E element)*: Заменяет элемент на указанной позиции новым элементом.

*remove(int index)*: Удаляет элемент по указанному индексу.

*size()*: Возвращает текущий размер списка “*ArrayList*”.

*isEmpty()*: Проверяет, является ли “*ArrayList*” пустым.

*clear()*: Удаляет все элементы из списка.

*contains(Object obj)*: Проверяет, содержится ли указанный объект в “*ArrayList*”.

*indexOf(Object obj)*: Возвращает индекс первого вхождения указанного объекта в “*ArrayList*”.

*lastIndexOf(Object obj)*: Возвращает индекс последнего вхождения указанного объекта в “*ArrayList*”.

*toArray()*: Возвращает массив, содержащий все элементы из “*ArrayList*”.

1. **Структурное описание**
   1. **Класс “*Table*”**

Атрибуты:

1. “*tariff*” - Статическая переменная, представляющая стоимость за минуту.
2. “*sessions*” - Список для хранения информации о каждой сессии, включая ее длительность и общую стоимость.
3. “*availability*” - Булева переменная, указывающая, доступен ли столик или занят.
4. “start\_time” - Записывает время начала текущей сессии.

Методы:

1. “*new\_session()*” - Начинает новую сессию, обновляя доступность и время начала.

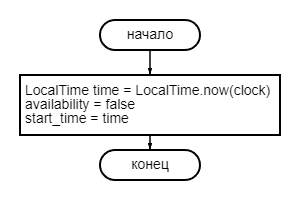


Рис. 1 блок-схема метода “*new\_session()*” класса “*Table*”

1. “*close\_session()*” - Завершает текущую сессию, обновляя доступность и записывая детали сессии.

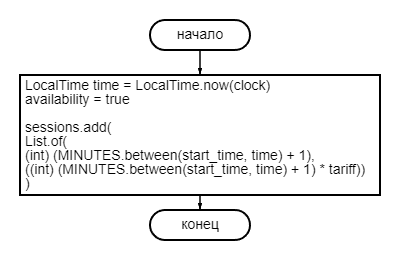


Рис. 2 блок-схема метода “*close\_session()*” класса “*Table*”

1. “*is\_available()*” - Проверяет, доступен ли в данный момент столик.
2. “get\_current\_time()” - Вычисляет прошедшее время с начала текущей сессии.

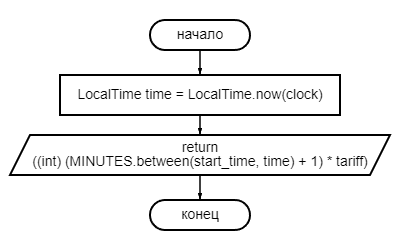


Рис. 3 блок-схема метода “*get\_current\_time()*” класса “*Table*”

1. *“get\_current\_price()”* - Метод для вычисления и возврата цены текущей сессии.
2. *“get\_number\_of\_sessions()”* - Метод для получения общего количества сессий.
3. *“get\_average\_session\_time()”* - Метод для вычисления и возврата средней длительности сессии.

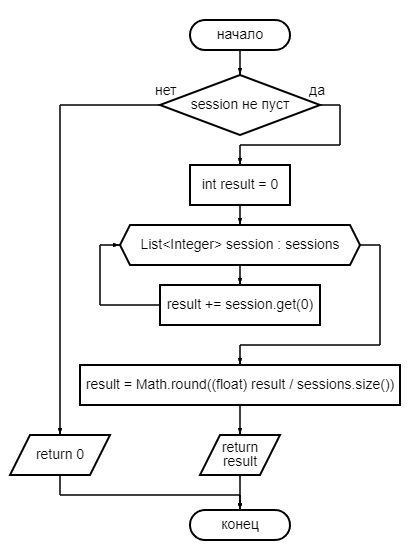


Рис. 4 блок-схема метода “*get\_average\_session\_time()*” класса “*Table*”

1. *“get\_amount\_of\_money\_earned()”* - Метод для вычисления и возврата общей суммы заработанных денег со всех сессий.

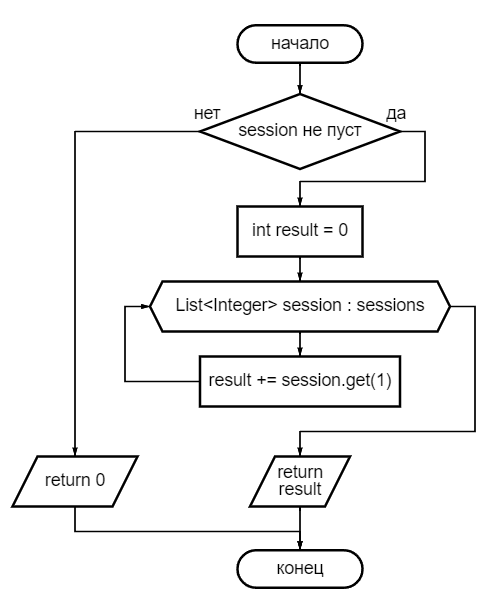


Рис. 5 блок-схема метода “*get\_amount\_of\_money\_earned()*” класса “*Table*”

Класс предоставляет функциональность для управления столиком в антикафе, отслеживая сессии, их длительность и стоимость.

Описание кода подробнее:

1. *protected static int tariff;* - Объявление статической переменной “*tariff*” типа “*int*”, которая хранит тариф для антикафе.
2. *private final List<List<Integer>> sessions = new ArrayList<>();* - Создание списка “*sessions*” для хранения информации о сессиях. Каждая сессия представлена списком, содержащим длительность и заработок.
3. *public boolean availability = true;* - Объявление булевой переменной “*availability*”, указывающей, доступен ли стол в данный момент.
4. *private final Clock clock = Clock.systemUTC();* - Создание объекта “*clock*” типа “*Clock*” для управления временем, инициализированного системным временем в формате UTC.
5. *private LocalTime start\_time = LocalTime.now(clock);* - Создание объекта “*start\_time*” типа “*LocalTime*” для хранения времени начала текущей сессии, инициализированного текущим временем.
6. *public void new\_session() - -* Метод для начала новой сессии, обновляющий доступность и устанавливающий время начала.
7. *LocalTime time = LocalTime.now(clock);* - Создание объекта “*time*” типа “*LocalTime*” с текущим временем.
8. *availability = false;* - Установка флага “*availability*” “*false*”, обозначая, что стол теперь занят.
9. *start\_time = time;* - Обновление времени начала текущей сессии.
10. *public void close\_session()* - Метод для закрытия текущей сессии, обновляющий доступность и добавляющий детали сессии в список.
11. *LocalTime time = LocalTime.now(clock);* - Создание объекта “*time*” типа “*LocalTime*” с текущим временем.
12. *availability = true;* - Установка флага “*availability*” “*true*”, обозначая, что стол теперь свободен.
13. *sessions.add(List.of((int) (MINUTES.between(start\_time, time) + 1), ((int) (MINUTES.between(start\_time, time) + 1) \* tariff)));* - Добавление информации о текущей сессии (длительность и заработок) в список “sessions”.
14. *public boolean is\_available()* - Метод для проверки доступности стола в текущий момент.
15. *return availability;* - Возврат значения переменной “availability”, указывающей, доступен ли стол.
16. *public int get\_current\_time()* - Метод для вычисления и возврата длительности текущей сессии.
17. *LocalTime time = LocalTime.now(clock);* - Создание объекта “*time*” типа “*LocalTime*” с текущим временем.
18. *return (int) MINUTES.between(start\_time, time);* - Возврат длительности текущей сессии в минутах.
19. *public int get\_current\_price()* - Метод для вычисления и возврата цены текущей сессии.
20. *LocalTime time = LocalTime.now(clock);* - Создание объекта “*time*” типа “*LocalTime*” с текущим временем.
21. *return ((int) (MINUTES.between(start\_time, time) + 1) \* tariff);* - Возврат стоимости текущей сессии, учитывая тариф.
22. *public int get\_number\_of\_sessions()* - Метод для получения общего количества сессий.
23. *return sessions.size();* - Возврат общего количества сессий из списка “*sessions*”.
24. *public int get\_average\_session\_time()* - Метод для вычисления и возврата средней длительности сессии.
25. *if (!sessions.isEmpty()) {* - Проверка, не пуст ли список “sessions”.
26. *int result = 0;* - Инициализация переменной “*result*” для суммирования длительностей сессий.
27. *for (List<Integer> session : sessions) { result += session.get(0); }* - Суммирование длительностей каждой сессии.
28. *result = Math.round((float) result / sessions.size());* - Вычисление средней длительности и округление к целому числу.
29. *return result;* - Возврат средней длительности сессии.
30. *public int get\_amount\_of\_money\_earned()* - Метод для вычисления и возврата общей суммы заработанных денег со всех сессий.
31. *if (!sessions.isEmpty()) {* - Проверка, не пуст ли список “*sessions*”.
32. *int result = 0;* - Инициализация переменной “*result*” для суммирования заработка сессий.
33. *for (List<Integer> session : sessions) { result += session.get(1); }* - Суммирование заработка каждой сессии.
34. *return result;* - Возврат общей суммы заработанных денег со всех сессий.
    1. **Класс “*Main”***

Атрибуты:

1. “tables\_list” - Список объектов класса “Table”, представляющих столики в антикафе.
2. “menu, tables\_menu, statistics” - Объекты GUI (графического интерфейса пользователя) для меню, списка столов и статистики соответственно.

Методы:

1. “*main(String[] args)*” - Основной метод, инициализирующий приложение. Устанавливает тариф, создает 10 столов и запускает GUI-окна для меню, списка столов и статистики.

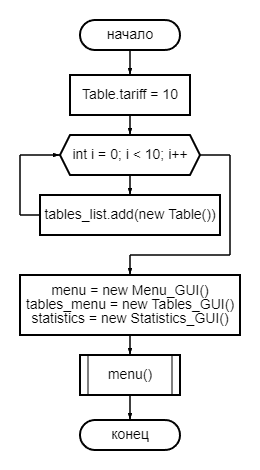


Рис. 6 блок-схема метода “*main()*” класса “*Main*”

1. “*menu()”, “tables\_list()”, “statistics()”* - Методы для управления видимостью различных GUI-окон в приложении.

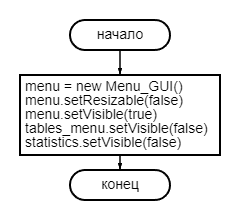


Рис. 7 блок-схема метода “*menu()*” класса “*Main*”

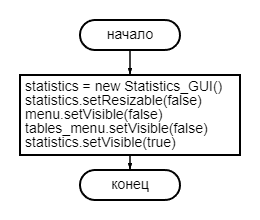


Рис. 8 блок-схема метода “*tables\_list()*” класса “*Main*”

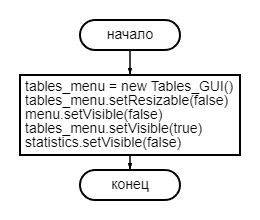


Рис. 9 блок-схема метода “*statistics()*” класса “*Main*”

Класс “*Main*” показывает организацию главного потока приложения и взаимодействие с GUI-компонентами.

Описание кода подробнее:

1. *static List<Table> tables\_list = new ArrayList<>(); -* Статический список столов.
2. *static Menu\_GUI menu; -* Объект для графического интерфейса меню.
3. *static Tables\_GUI tables\_menu;* - Объект для таблиц.
4. *static Statistics\_GUI statistics; -* Объект для статистики.
5. *public static void main(String[] args) {* - Точка входа в приложение.
6. *Table.tariff = 10;* - Установка тарифа для столов.
7. *for (int i = 0; i < 10; i++) { tables\_list.add(new Table()); }* - Инициализация списка столов.
8. *menu = new Menu\_GUI(); tables\_menu = new Tables\_GUI(); statistics = new Statistics\_GUI();* - . Инициализация графического интерфейса меню, таблиц и статистики.
9. *menu();* - Запуск основного меню.
10. *public static void menu() { -* Метод для отображения главного меню
11. *menu = new Menu\_GUI(); -* Создание объекта графического интерфейса меню.
12. *menu.setResizable(false);* - Запрет изменения размеров окна.
13. *menu.setVisible(true);* - Отображение меню.
14. *tables\_menu.setVisible(false);* - Скрытие таблиц.
15. *statistics.setVisible(false);}* - Скрытие статистики.
16. *public static void tables\_list()* { - Метод для отображения списка столов.
17. *tables\_menu = new Tables\_GUI();* - Создание объекта графического интерфейса списка столов.
18. *tables\_menu.setResizable(false);* - Запрет изменения размеров окна.
19. *menu.setVisible(false);* - Скрытие меню
20. *tables\_menu.setVisible(true);* Отображение списка столов.
21. *statistics.setVisible(false);* - Скрытие статистики.
22. public static void statistics() { - Метод для отображения статистики.
23. *statistics = new Statistics\_GUI();* - Создание объекта графического интерфейса статистики.
24. *statistics.setResizable(false);* - Запрет изменения размеров окна.
25. *menu.setVisible(false);* - Скрытие меню.
26. *tables\_menu.setVisible(false);* - Скрытие списка столов.
27. statistics.setVisible(true);}} - Отображение статистики.
    1. **Класс “Menu\_GUI”**

Конструктор:

1. Устанавливает заголовок окна, размеры и расположение.
2. Инициализирует контейнер и устанавливает менеджер компоновки “*GridLayout*” с двумя строками и одним столбцом.
3. Создает две кнопки ("Список столов" и "Статистика") и добавляет их в контейнер.
4. Добавляет слушателей событий для кнопок.

Вложенные классы:

1. “*tables\_menu*” - Реализует интерфейс “*ActionListener*” для обработки событий кнопки "Список столов". Вызывает метод “*Main.tables\_list()*” при активации.
2. “*statistics\_menu*” - Реализует интерфейс “*ActionListener*” для обработки событий кнопки "Статистика". Вызывает метод “*Main.statistics()*” при активации.

Класс “*Menu\_GUI*” описывает окно меню, где пользователь может выбирать между просмотром списка столов и статистики.

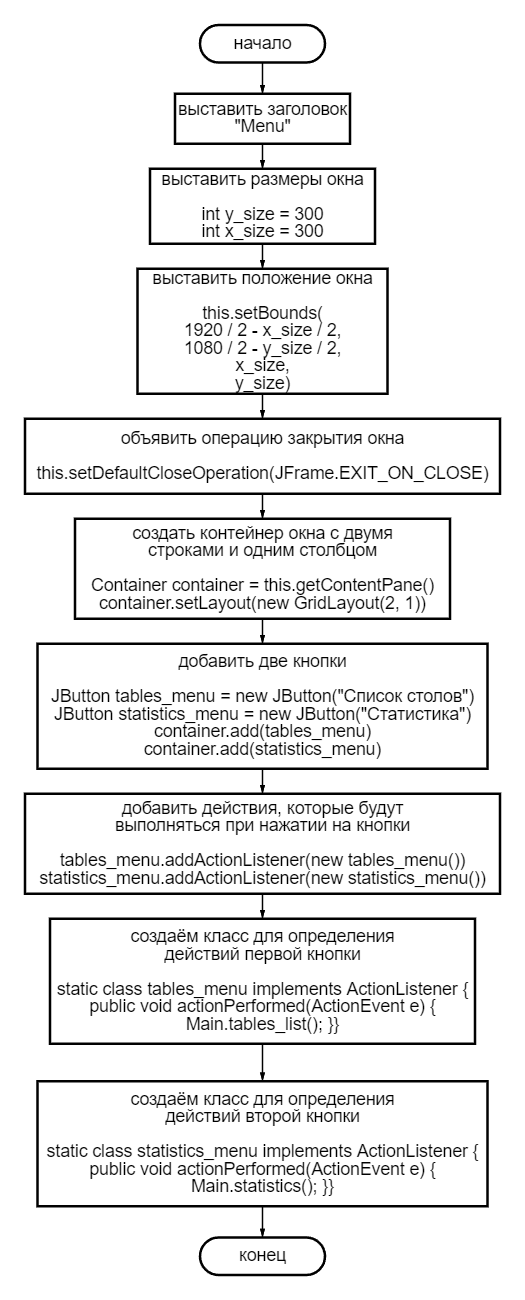


Рис. 10 блок-схема класса “*Menu\_GUI*”

Описание кода подробнее:

1. *public Menu\_GUI()* { - Объявление конструктора класса “*Menu\_GUI*”.
2. *super("Menu");* - Вызов конструктора суперкласса (“*JFrame*”) с передачей заголовка окна.
3. *int y\_size = 300;* - Определение переменной “*y\_size*” для высоты окна.
4. *int x\_size = 300*; - Определение переменной “*x\_size*” для ширины окна.
5. *this.setBounds(1920 / 2 - x\_size / 2, 1080 / 2 - y\_size / 2, x\_size, y\_size);*-Установка размеров и положения окна на экране.
6. *this.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);* - Установка операции при закрытии окна (выход из приложения).
7. *Container container = this.getContentPane();* - Получение контейнера (панели содержимого) для компоновки компонентов.
8. *container.setLayout(new GridLayout(2, 1));* - Установка менеджера компоновки “*GridLayout*” (2 строки, 1 столбец) для контейнера.
9. *JButton tables\_menu = new JButton("Список столов");* - Создание кнопки "Список столов".
10. *JButton statistics\_menu = new JButton("Статистика");* - Создание кнопки "Статистика".
11. *container.add(tables\_menu);* - Добавление кнопки "Список столов" в контейнер.
12. *container.add(statistics\_menu);* - Добавление кнопки "Статистика" в контейнер.
13. *tables\_menu.addActionListener(new tables\_menu());* - Установка слушателя событий для кнопки "Список столов".
14. *statistics\_menu.addActionListener(new statistics\_menu());* - Установка слушателя событий для кнопки "Статистика".
15. *}* - Завершение конструктора класса “*Menu\_GUI*”.
16. *static class tables\_menu implements ActionListener {* - Объявление внутреннего статического класса “*tables\_menu*”, реализующего интерфейс “*ActionListener*”.
17. *public void actionPerformed(ActionEvent e) {* - Реализация метода “*actionPerformed*” интерфейса “*ActionListener*”.
18. *Main.tables\_list();* - Вызов статического метода “*tables\_list*” из класса “*Main*”.
19. *static class statistics\_menu implements ActionListener {* - Объявление внутреннего статического класса “*statistics\_menu*”, реализующего интерфейс “*ActionListener*”.
20. *public void actionPerformed(ActionEvent e) {* - Реализация метода “*actionPerformed*” интерфейса “*ActionListener*”.
21. *Main.statistics();* - Вызов статического метода “*statistic*s” из класса “*Main*”.
    1. **Класс “Statistics\_GUI”**

Конструктор:

1. Устанавливает заголовок окна, размеры и расположение.
2. Инициализирует контейнер и устанавливает менеджер компоновки “*GridLayout*” с 55 строками и 1 столбцом.
3. Создает текстовые поля для каждого столика с информацией о заработанной сумме, среднем времени занятости и количестве сессий.
4. Вычисляет общую сумму заработка.
5. Находит столик с максимальной суммой заработка и максимальным количеством сессий.
6. Создает текстовые поля для общей статистики и кнопку для возвращения в главное меню.
7. Добавляет все компоненты в контейнер и устанавливает слушателя событий для кнопки "Вернуться в главное меню".

Вложенный класс:

“*back*” - Реализует интерфейс “*ActionListener*” для обработки событий кнопки "Вернуться в главное меню". Вызывает метод “*Main.menu()*” при активации.

Класс “*Statistics\_GUI*” предоставляет пользователю обзор статистики по столикам и возможность вернуться в главное меню.

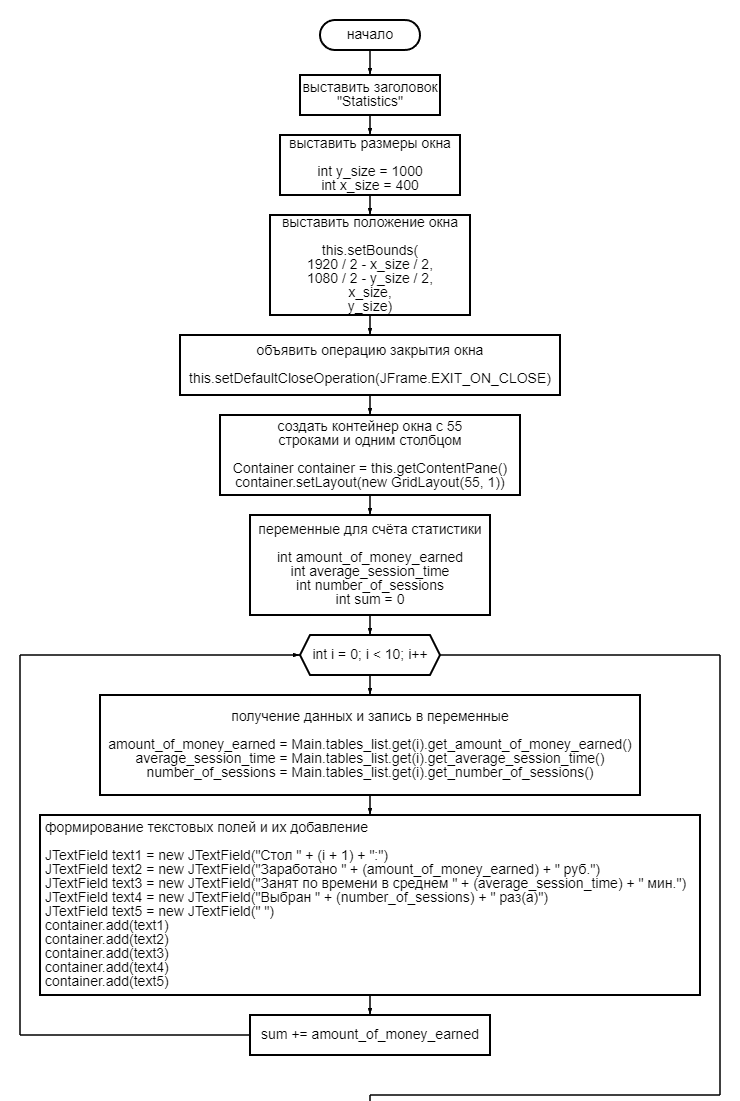


Рис. 11 блок-схема класса “*Statistics\_GUI*” (часть 1)

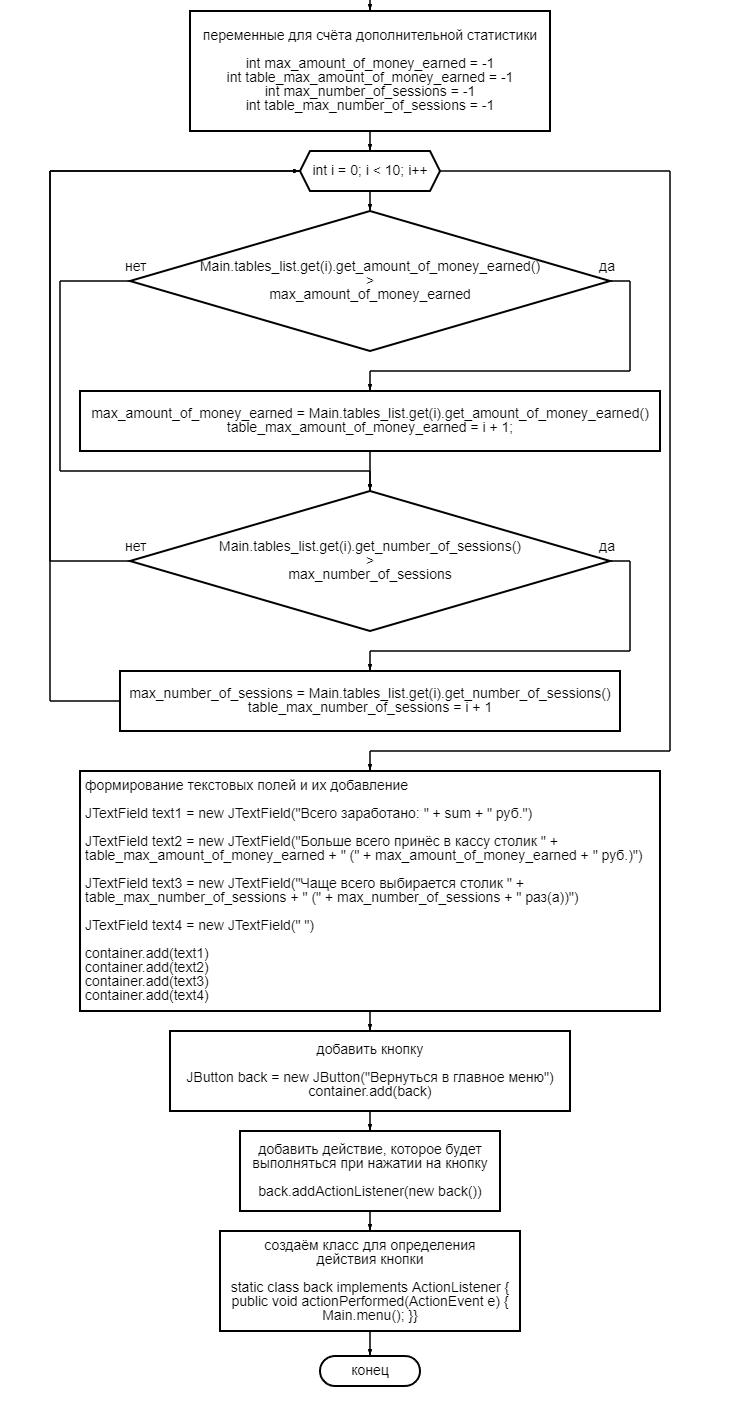


Рис. 12 блок-схема класса “*Statistics\_GUI*” (часть 2)

Описание кода подробнее:

1. *public class Statistics\_GUI extends JFrame {* - Объявление класса “*Statistics\_GUI*”, расширяющего “*JFrame*” для создания графического интерфейса статистики.
2. *public Statistics\_GUI() {* - Объявление конструктора класса “Statistics\_GUI”.
3. *super("Statistics");* - Вызов конструктора суперкласса (“*JFrame*”) с передачей заголовка окна.
4. *int y\_size = 1000;* - Определение переменной “*y\_size*” для высоты окна.
5. *int x\_size = 400;* - Определение переменной “*x\_size*” для ширины окна.
6. *this.setBounds(1920 / 2 - x\_size / 2, 1080 / 2 - y\_size / 2, x\_size, y\_size);* - Установка размеров и положения окна на экране.
7. *this.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);* - Установка операции при закрытии окна (выход из приложения).
8. *Container container = this.getContentPane();* - Получение контейнера (панели содержимого) для компоновки компонентов.
9. container.setLayout(new GridLayout(55, 1)); - Установка менеджера компоновки “*GridLayout*” (55 строк, 1 столбец) для контейнера.
10. *int amount\_of\_money\_earned;* - Объявление переменной для хранения заработанных денег.
11. *int average\_session\_time;* - Объявление переменной для хранения средней длительности сессии.
12. *int number\_of\_sessions;* - Объявление переменной для хранения количества сессий.
13. *int sum = 0;* - Инициализация переменной для подсчета общей суммы заработанных денег.
14. *for (int i = 0; i < 10; i++) {* - Начало цикла для обработки информации о каждом столе.
15. *amount\_of\_money\_earned =Main.tables\_list.get(i).get\_amount\_of\_money\_earned();* - Получение заработанных денег для текущего стола.
16. *average\_session\_time = Main.tables\_list.get(i).get\_average\_session\_time();* - Получение средней длительности сессии для текущего стола.
17. *number\_of\_sessions = Main.tables\_list.get(i).get\_number\_of\_sessions();* - Получение количества сессий для текущего стола.
18. *JTextField text1 = new JTextField("Стол " + (i + 1) + ":");* - Создание текстового поля для отображения информации о столе
19. *JTextField text2 = new JTextField("Заработано " + (amount\_of\_money\_earned) + " руб.");* - Создание текстового поля с информацией о заработанных деньгах.
20. *JTextField text3 = new JTextField("Занят по времени в среднем " + (average\_session\_time) + " мин.");* - Создание текстового поля с информацией о средней длительности сессии.
21. *JTextField text4 = new JTextField("Выбран " + (number\_of\_sessions) + " раз(a)");* - Создание текстового поля с информацией о количестве сессий.
22. *JTextField text5 = new JTextField(" ");* - Создание пустого текстового поля для разделения информации о столах.
23. *container.add(text1);* - Добавление текстового поля о номере стола в контейнер.
24. *container.add(text2);* - Добавление текстового поля с информацией о заработанных деньгах в контейнер.
25. *container.add(text3);* - Добавление текстового поля с информацией о средней длительности сессии в контейнер.
26. *container.add(text4);* - Добавление текстового поля с информацией о количестве сессий в контейнер.
27. *container.add(text5);* - Добавление пустого текстового поля в контейнер.
28. *sum += amount\_of\_money\_earned;* - Наращивание суммы заработанных денег.
29. *int max\_amount\_of\_money\_earned = -1;* - Инициализация переменной для отслеживания максимального заработка.
30. *int table\_max\_amount\_of\_money\_earned = -1;* - Инициализация переменной для хранения номера стола с максимальным заработком.
31. *int max\_number\_of\_sessions = -1;* - Инициализация переменной для отслеживания максимального количества сессий.
32. *int table\_max\_number\_of\_sessions = -1;* - Инициализация переменной для хранения номера стола с максимальным количеством сессий.
33. *for (int i = 0; i < 10; i++) {* - Начало второго цикла для поиска стола с максимальным заработком и максимальным количеством сессий.
34. *if (Main.tables\_list.get(i).get\_amount\_of\_money\_earned() > max\_amount\_of\_money\_earned){* - Проверка, является ли текущий заработок максимальным.
35. *max\_amount\_of\_money\_earned=Main.tables\_list.get(i).get\_amount\_of\_money\_earned();* - Обновление максимального заработка.
36. *table\_max\_amount\_of\_money\_earned = i + 1;* - Обновление номера стола с максимальным заработком.
37. *if (Main.tables\_list.get(i).get\_number\_of\_sessions() > max\_number\_of\_sessions) {* - Проверка, является ли текущее количество сессий максимальным.
38. *max\_number\_of\_sessions=Main.tables\_list.get(i).get\_number\_of\_sessions()*; - Обновление максимального количества сессий.
39. *table\_max\_number\_of\_sessions = i + 1;* - Обновление номера стола с максимальным количеством сессий.
40. *JTextField text1 = new JTextField("Всего заработано: " + sum + " руб.");* - Создание текстового поля с общей суммой заработанных денег.
41. *JTextField text2 = new JTextField("Больше всего принёс в кассу столик " + table\_max\_amount\_of\_money\_earned + " (" + max\_amount\_of\_money\_earned + " руб.)");* - Создание текстового поля с информацией о столе с максимальным заработком.
42. *JTextField text3 = new JTextField("Чаще всего выбирается столик " + table\_max\_number\_of\_sessions + " (" + max\_number\_of\_sessions + " раз(a))");* - Создание текстового поля с информацией о столе с максимальным количеством сессий.
43. *JTextField text4 = new JTextField(" ");* - Создание пустого текстового поля для разделения информации.
44. *container.add(text1);* - Добавление текстового поля с общей суммой заработанных денег в контейнер.
45. *container.add(text2);* - Добавление текстового поля с информацией о столе с максимальным заработком в контейнер.
46. *container.add(text3);* - Добавление текстового поля с информацией о столе с максимальным количеством сессий в контейнер.
47. *container.add(text4);* - Добавление пустого текстового поля в контейнер.
48. *JButton back = new JButton("Вернуться в главное меню");* - Создание кнопки "Вернуться в главное меню".
49. *container.add(back);* - Добавление кнопки в контейнер.
50. *back.addActionListener(new back());* - Установка слушателя событий для кнопки "Вернуться в главное меню".
51. *}*- Завершение конструктора класса Statistics\_GUI.
52. *static class back implements ActionListener {* - Объявление внутреннего статического класса back, реализующего интерфейс ActionListener.
53. *public void actionPerformed(ActionEvent e) {* - Реализация метода actionPerformed интерфейса ActionListener.
54. *Main.menu();* - Вызов статического метода menu из класса Main для возврата в главное меню.
    1. **Класс “Tables\_GUI**”

Конструктор:

1. Устанавливает заголовок окна, размеры и расположение.
2. Инициализирует контейнер и устанавливает менеджер компоновки “*GridLayout*” с 6 строками и 2 столбцами.
3. Создает кнопки для каждого столика с информацией о его статусе (свободен или занят), текущем времени и стоимости.
4. Устанавливает цвет фона кнопок в зависимости от их статуса (зеленый для свободных и красный для занятых).
5. Добавляет кнопки в контейнер и устанавливает слушателей событий для каждой кнопки столика.
6. Создает кнопку "Вернуться в главное меню" и текстовое поле для отображения общей суммы занятых столов.
7. Добавляет кнопку и текстовое поле в контейнер и устанавливает слушателя событий для кнопки "Вернуться в главное меню".

Вложенные классы:

1. *table\_button* - Реализует интерфейс “*ActionListener*” для обработки событий кнопок столов. Запускает новую сессию, если столик свободен, или закрывает текущую сессию, если столик занят. Затем вызывает метод “*Main.menu()*”.
2. *Back* - Реализует интерфейс “*ActionListener*” для обработки событий кнопки "Вернуться в главное меню". Вызывает метод “*Main.menu()*”.

Класс “Tables\_GUI” предоставляет пользователю возможность взаимодействовать со столиками, начинать и заканчивать сессии, а также возвращаться в главное меню.

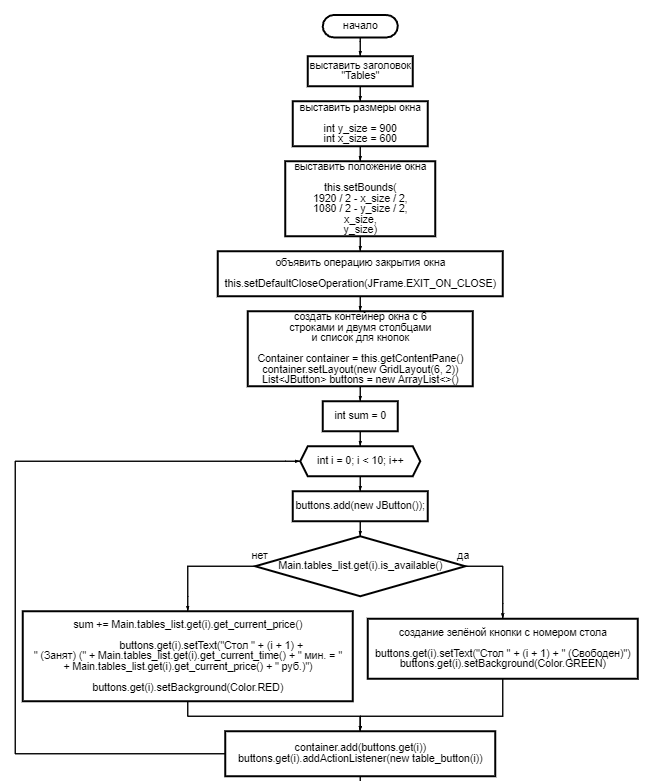


Рис. 13 блок-схема класса “*Tables\_GUI*” (часть 1)

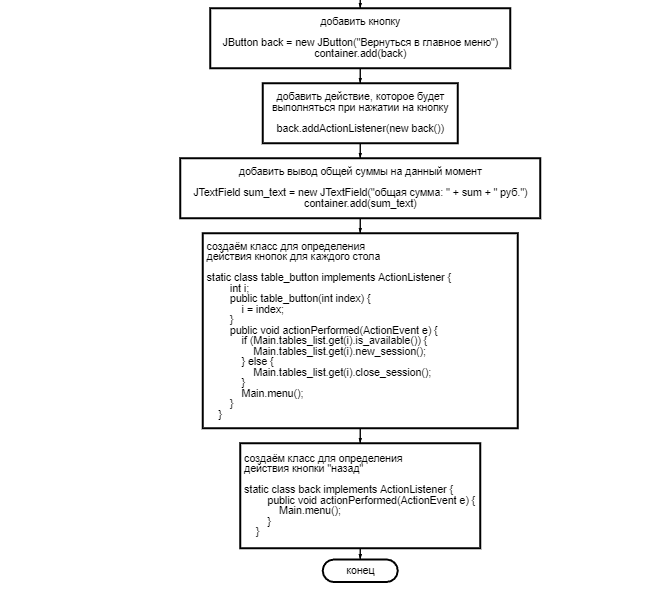


Рис. 14 блок-схема класса “*Tables\_GUI*” (часть 2)

Описание кода подробнее:

1. *public class Tables\_GUI extends JFrame {* - Объявление класса “*Tables\_GUI*”, который расширяет класс “*JFrame*” для создания графического интерфейса списка столов.
2. *public Tables\_GUI() {* - Объявление конструктора класса “Tables\_GUI”.
3. *super("Tables") ;-* Вызов конструктора суперкласса (“*JFrame*”) с передачей заголовка окна.
4. *int y\_size = 900;* - Определение переменной “*y\_size*” для высоты окна.
5. *int x\_size = 600;* - Определение переменной “*x\_size*” для ширины окна.
6. *this.setBounds(1920 / 2 - x\_size / 2, 1080 / 2 - y\_size / 2, x\_size, y\_size);* - Установка размеров и положения окна на экране.
7. *this.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);* - Установка операции при закрытии окна (выход из приложения).
8. *Container container = this.getContentPane();* - Получение контейнера (панели содержимого) для компоновки компонентов.
9. *container.setLayout(new GridLayout(6, 2));* - Установка менеджера компоновки “*GridLayout*” (6 строк, 2 столбца) для контейнера.
10. *List<JButton> buttons = new ArrayList<>();* - Создание списка кнопок с использованием “*ArrayList*”.
11. *int sum = 0;* - Инициализация переменной для подсчета общей суммы заработанных денег.
12. *for (int i = 0; i < 10; i++) {* - Начало цикла для обработки информации о каждом столе.
13. *buttons.add(new JButton());* - Создание новой кнопки и добавление ее в список.
14. *if (Main.tables\_list.get(i).is\_available()) {* - Проверка, свободен ли текущий стол.
15. *buttons.get(i).setText("Cтол " + (i + 1) + " (Свободен)");* - Установка текста на кнопке для свободного стола.
16. *buttons.get(i).setBackground(Color.GREEN);* - Установка фона кнопки для свободного стола (зеленый).
17. *} else {* - Ветвь, выполняющаяся, если стол занят.
18. *sum += Main.tables\_list.get(i).get\_current\_price();* - Наращивание суммы заработанных денег.
19. *buttons.get(i).setText("Cтол " + (i + 1) + " (Занят) (" + Main.tables\_list.get(i).get\_current\_time() + " мин. = " + Main.tables\_list.get(i).get\_current\_price() + " руб.)");* - Установка текста на кнопке для занятого стола.
20. *buttons.get(i).setBackground(Color.RED);* - Установка фона кнопки для занятого стола (красный).
21. *container.add(buttons.get(i));* - Добавление кнопки в контейнер.
22. *buttons.get(i).addActionListener(new table\_button(i));* - Установка слушателя событий для кнопки, связывая ее с определенным столом.
23. *}* - Завершение цикла обработки столов.
24. *JButton back = new JButton("Вернуться в главное меню");* - Создание кнопки "Вернуться в главное меню".
25. *container.add(back);* - Добавление кнопки в контейнер.
26. *back.addActionListener(new back());* - Установка слушателя событий для кнопки "Вернуться в главное меню".
27. *JTextField sum\_text = new JTextField("общая сумма: " + sum + " руб.");* - Создание текстового поля с общей суммой заработанных денег.
28. *container.add(sum\_text);* - Добавление текстового поля в контейнер.
29. *}* - Завершение конструктора класса

**Тестинг программы**

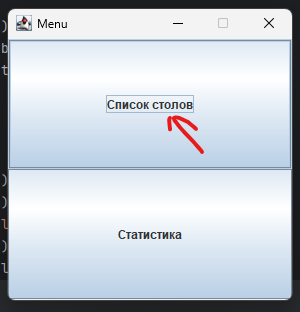


Рис. 15 главное меню (выбрать меню списка столов)



Рис. 16 меню столов (выбрать 1 стол)



Рис. 17 меню столов (выбрать 6 стол)



Рис. 18 меню столов (выбрать 1 стол)



Рис. 19 меню столов (вернуться в главное меню)

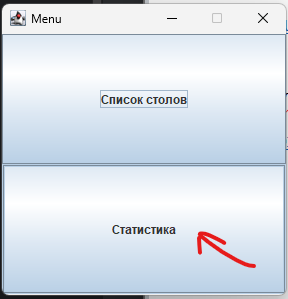


Рис. 20 меню столов (выбрать показ статистики)

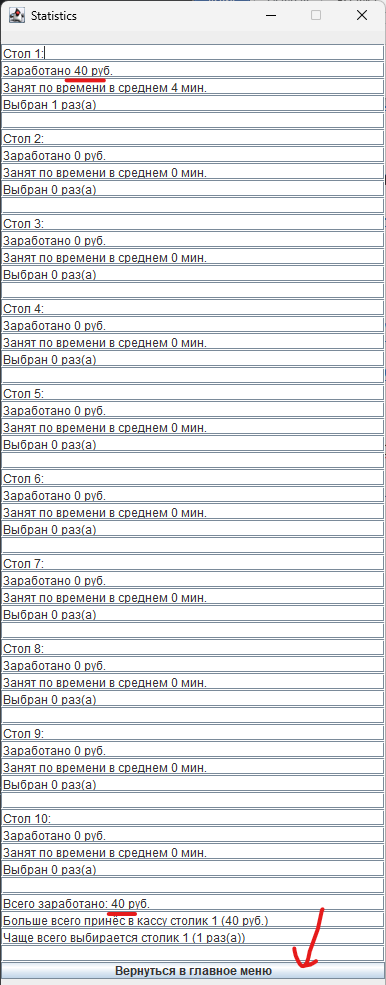


Рис. 21 показ статистики (вернуться в главное меню)

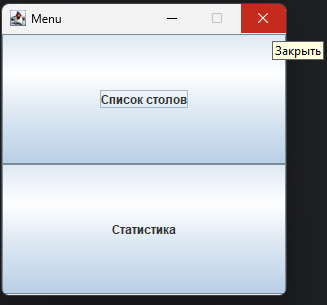


Рис. 22 закрытие окна приложения



Рис. 23 результат закрытия окна приложения

**Заключение**

Разработка приложения для управления антикафе представляет собой важный этап в совершенствовании современных методов управления в индустрии общественного питания. Приложение создано с целью предоставить владельцам антикафе мощный инструмент для эффективного ведения бизнеса и повышения качества обслуживания клиентов.

Класс “*Table*” играет ключевую роль в этом проекте, обеспечивая детальное отслеживание и управление каждым столиком. Гибкость тарифа за минуту позволяет владельцам антикафе настраивать ценовую политику в соответствии с текущими трендами рынка и потребностями клиентов. Этот класс становится фундаментом для создания интеллектуальной системы управления, способной адаптироваться к изменяющимся условиям и предоставлять ценные аналитические данные.

Интерфейс пользователя, разработанный на базе библиотеки “*Swing*”, обеспечивает простоту использования и интуитивно понятный доступ ко всем функциональным возможностям приложения. Это создает комфортное взаимодействие как для владельцев антикафе, так и для посетителей.

Аналитика и статистика, предоставляемая приложением, открывают владельцам новые возможности для принятия обоснованных решений. Отслеживание популярности столиков, средней продолжительности сессий и общей выручки становится ключевым элементом стратегии управления.

Созданное приложение не только автоматизирует операционные процессы антикафе, но и открывает путь для внедрения инноваций и оптимизации бизнес-процессов. Оно создает перспективы для более эффективного управления заведением, улучшения клиентского опыта и укрепления позиций на рынке общественного питания.

**Список используемых источников**

1. Библиотека Swing // Java-online.ru URL: <https://java-online.ru/libs-swing.xhtml> (дата обращения: 15.12.2023).
2. События и слушатели // Java-online.ru URL: <https://java-online.ru/java-listener.xhtml> (дата обращения: 15.12.2023).
3. Что такое JFrame в Java? // uchet-jkh.ru URL: <https://uchet-jkh.ru/i/cto-takoe-jframe-v-java> (дата обращения: 15.12.2023).
4. AWT Краткое руководство // CoderLessons.com URL: <https://coderlessons.com/tutorials/java-tekhnologii/uznaite-awt/awt-kratkoe-rukovodstvo> (дата обращения: 15.12.2023).
5. JTextField, JTextArea, JTextPane // Java-online.ru URL: <https://java-online.ru/swing-jtextfield.xhtml> (дата обращения: 15.12.2023).