УЧЕБНОЕ ЗАВЕДЕНИЕ: GEEKBRAINS

Специальность: Программист Java

Цифровые профессии

**Дипломная работа**

**На тему «Простое банковское приложение на языке Java.**

**Проект IQKidBank»**

Подготовил:

студентка GeekBrains, гр. 5944

Загребина А.П.

Ссылка на GitHub:

https://github.com/AnnaBinoid/IQKidBank.git

Воронеж

2024

**Простое банковское приложение на языке Java.**

**Проект IQKidBank.**

## Оглавление

[Оглавление 1](#_Toc160907397)

[1. Введение 2](#_Toc160907398)

[1.1 Актуальность проекта 2](#_Toc160907399)

[1.2 Сложности разработки 3](#_Toc160907400)

[1.3 Архитектура проекта 4](#_Toc160907401)

[1.3.1 Use-Case 5](#_Toc160907402)

[1.3.2 UML 7](#_Toc160907403)

[1.3.3 Прочая документация 9](#_Toc160907404)

[1.4 Инструменты разработки 11](#_Toc160907405)

[1.4.1 Язык программирования Java 12](#_Toc160907406)

[1.4.2 Среда разработки IntelliJ Idea 13](#_Toc160907407)

[1.5 Среда окружения и вспомогательные ресурсы (допущения). 14](#_Toc160907408)

[1.5.1 Банковское ПО 15](#_Toc160907409)

[1.5.2 Дневник.ру 15](#_Toc160907410)

[1.5.3 Система контроля версий Git и онлайн-сервис Github 15](#_Toc160907411)

[2. Создание банковского приложения 17](#_Toc160907412)

[2.1 Классы Person и Customer 17](#_Toc160907413)

[2.1.1 Наследование классов 18](#_Toc160907414)

[2.1.2 Геттеры, сеттеры 19](#_Toc160907415)

[2.1.3 Конструкторы 21](#_Toc160907416)

[2.1.4 Переопределение методов базового класса Object 23](#_Toc160907417)

[2.1.4.1 Переопределение метода toString() для класса Person 23](#_Toc160907418)

[2.1.4.2 Переопределение метода equals() для класса Person 24](#_Toc160907419)

[2.1.4.3 Переопределение метода hashCode() для класса Person 25](#_Toc160907420)

[2.2 Класс Customer, наследованный от класса Person и имплементирующий интерфейс iMakeDeposite 26](#_Toc160907421)

[2.2.1 Оправданность использования интерфейса iMakeDeposite с точки зрения ООП 26](#_Toc160907422)

[2.2.2 Класс Customer 27](#_Toc160907423)

[2.2.2.1 ArrayList<String> transactions для записи транзакций и использование дженериков 28](#_Toc160907424)

[2.2.2.2 Конструктор класса Customer 29](#_Toc160907425)

[2.2.2.3 Метод addTransaction () класса Customer 30](#_Toc160907426)

[2.2.2.4 Метод trimToSize() класса Customer 30](#_Toc160907427)

[2.2.2.4.1 Формат даты в методе addTransaction() 31](#_Toc160907428)

[2.2.2.5 Метод deposite () класса Customer 32](#_Toc160907429)

[2.2.2.6 Метод withdraw() класса Customer 33](#_Toc160907430)

[2.3 Класс MarkStatus 35](#_Toc160907431)

[2.3.1 Константы (private final переменные) и класс Enum 36](#_Toc160907432)

[2.3.2 Scanner в Java 37](#_Toc160907433)

[2.3.3 Методы класса MarkStatus 38](#_Toc160907434)

[2.3.3.1 Метод createMarkArray() класса MarkStatus 39](#_Toc160907435)

[2.3.3.2 Метод setGradesByOne() класса MarkStatus и цикл do-while и оператор break 40](#_Toc160907436)

[2.3.3.3 Методы printList() и printMatrix() класса MarkStatus, цикл for 42](#_Toc160907437)

[2.3.3.4 Метод numberOfMarks() класса MarkStatus, конструкция ветвления if-else 44](#_Toc160907438)

[2.3.3.5 Метод setGradesCoefficient() класса MarkStatus 46](#_Toc160907439)

[2.3.3.6 Метод getCurrentGradesCoefficient() класса MarkStatus 47](#_Toc160907440)

[2.4 Класс BankAccount 49](#_Toc160907441)

[2.4.1 Пользовательский интерфейс и особенности его реализации в коммерческих приложениях 49](#_Toc160907442)

[2.4.2 Ещё немного о реализации оконных приложений 53](#_Toc160907443)

[2.4.3 Поля класса BankAccount 55](#_Toc160907444)

[Структура данных Map 56](#_Toc160907445)

[2.4.4 Методы класса BankAccount 57](#_Toc160907446)

[2.4.4.1 Метод showMenu(), строковые данные и пул строк. 57](#_Toc160907447)

[2.4.4.2 Метод enterByRegistration() проекта IQKidBank, конструкции while(true) и swich case 59](#_Toc160907448)

[2.4.4.2.1 «Вечный» цикл while(true) 59](#_Toc160907449)

[2.4.5 Слабые места реализации класса BankAccount 70](#_Toc160907450)

[2.5 Класс Main 70](#_Toc160907451)

[3. Заключение 72](#_Toc160907452)

[Список литературы и ссылки 74](#_Toc160907453)

## Введение

Проект IQKidBank разработан и реализован в рамках дипломного проекта по курсу «Программист Java. Цифровые профессии» компании Geek Brains студенткой Анной Петровной Загребиной в апреле 2024 года.

### Актуальность проекта

В современном мире ценность школьного образования сильно преуменьшена. В школах создается неблагоприятная среда для повышения привлекательности образования в глазах учеников. Проект IQKidBank – простое банковское приложение, помогающее конвертировать оценки учащихся в денежное поощрение от родителей на основании данных об оценках из Дневник.ру, вводимых родителем или иным меценатом вручную.

Наличие у ребенка собственных средств, которые он может «заработать», благодаря достижениям в учебе, и трата которых контролируется родителями, дает ему определенную свободу в выполнении своих желаний, а родителям – дополнительный инструмент поощрения, воспитания финансовой грамотности и контроля над юным членом семьи.

Так же приложение позволит детям вести безналичные банковские операции, изучать банковские инструменты (если дополнить проект обучающими материалами в игровой форме); а родителям – пополнять или уменьшать банковский счет и контролировать траты владельца счета.

Возможный (на данный момент не проработанный в коде, но продуманный) функционал приложения подразумевает:

* полную автоматизацию расчетов родителя с учеником на заданный период времени;
* возможность мецената/ родителя подключения блокировки средств на «оплату оценки» на родительской карте за определенный период времени, что является дополнительным гарантом получения средств для ученика;
* возможность экспортирования отметок из Exel-файла (для этого необходимо иметь договоренность с педагогом или руководством школы о регулярном экспорте и отправке родителям табеля успеваемости. Технически такая возможность в Дневник.ру существует, но доступна только из роли «Учитель».[[1]](#footnote-2)
* Масштабирование приложения с ролей ученик – родитель; отметка – денежное поощрение до стипендиат – меценат (в роли которого может выступать ВУЗ, организатор олимпиад, конкурсов и иных обучающих/развивающих мероприятий, за успешное прохождение которых учреждено денежное поощрение); достижение – поощрение (стипендия, денежный приз за победу в конкурсе, олимпиаде и тд);
* создание полноценных баз данных пользователей, расчетов и транзакций;
* создание полноценного десктопного и веб-приложения.

### Сложности разработки

В силу того, что реализация проекта подразумевала участие трех разработчиков, однако в процессе работы двое из трех членов команды отказались от участия, тестовая часть проекта реализована не полностью.

Зоны ответственности делились следующим образом:

1. разработка идеи приложения, use case, UML, создание кода для основного функционала (конвертирование оценок в денежный эквивалент), создание основных классов (банк, пользователь) и отношений между ними (на данный момент задачи реализованы);
2. создание баз данных (пользователи, транзакции) и взаимодействия баз данных с основным функционалом приложения, ERD (на данный момент база данных логинов и паролей пользователей реализована только в виде структуры данных Map с ключом – логином и значением – паролем пользователя;
3. написание тестов, GUI, создание графического окна для приложения, сборка проекта, API (не реализовано).

### 1.3 Архитектура проекта

Не обладая достаточной практикой в программировании, сложно сразу построить продуктивную архитектурную модель. К тому же в создании малых приложений в условиях отсутствия специализированного персонала под конкретные задачи и размытыми областями ответственностями, бытуют разные подходы:

* создавать проект, исходя из продуманной ранее архитектуры;
* написать рабочий проект, затем создать документацию.

В данной работе были объединены оба подхода: изначально разработанные диаграммы видоизменялись по мере написания основного кода.

Для того, чтобы разработать любое приложение, необходимо определить, кто или что и каким образом будет работать с этим приложением. Если представить моделирование любой программы, как процесс поуровневого спуска от абстрактной модели действующей бизнес-системы сперва к логической концепции, а после – к физической, точнее, программной составляющей (коду), то на первый план выходит диаграмма вариантов, use case diagram. В ней описываются конкретные задачи, выполняемые приложением, а так же обозначены акторы (субьекты, инициаторы) этих действий. Причем, в роли акторов могут выступать не только пользователи, но и другие механизмы и приложения.

#### 1.3.1 Use-Case

**Диаграмма вариантов использования (Use Case Diagram)** — диаграмма, на которой изображаются отношения между акторами и вариантами использования.

Диаграмма вариантов использования - это исходное концептуальное представление или концептуальная модель системы в процессе ее проектирования и разработки. Создание диаграммы вариантов использования имеет следующие цели:

* Определить общие границы и контекст моделируемой предметной области на начальных этапах проектирования системы;
* Сформулировать общие требования к функциональному поведению проектируемой системы;
* Разработать исходную концептуальную модель системы для ее последующей детализации в форме логических и физических моделей;
* Подготовить исходную документацию для взаимодействия разработчиков системы с ее заказчиками и пользователями;

Назначение данной диаграммы состоит в следующем: проектируемая программная система представляется в форме так называемых вариантов использования, с которыми взаимодействуют внешние сущности или акторы. При этом актором или действующим лицом называется любой объект, субъект или система, взаимодействующая с моделируемой бизнес-системой извне. Это может быть человек, техническое устройство, программа или любая другая система, которая служит источником воздействия на моделируемую систему так, как определит разработчик.

Вариант использования служит для описания сервисов, которые система предоставляет актору. Другими словами каждый вариант использования определяет набор действий, совершаемый системой при диалоге с актором. При этом ничего не говорится о том, каким образом будет реализовано взаимодействие акторов с системой и собственно выполнение вариантов использования.[1]

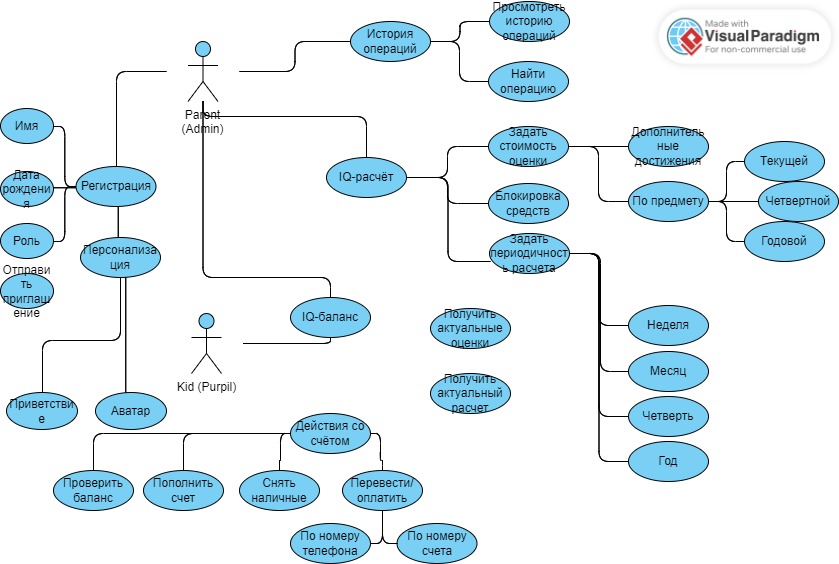


Рисунок 1. IQKidBank UseCase диаграмма.

В приложении IQKidBank акторами будут являться Parent и Kid, представляющие собой спонсора и получателя денежного эквивалента отметок. Данная диаграмма отображает функционал, продуманный для окончательного варианта приложения. На данном этапе функционал реализован частично.

Переходя от абстрактной идеи к реализации, необходимо понимать, что за сущности (классы) будут в программе, как они будут связаны между собой и какие функции (методы) будут реализовывать те или иные задачи use case. В этом поможет UML-диаграмма.

#### 1.3.2 UML

**UML-диаграмма** (далее UML) — это схема, нарисованная с применением символов, принятых в UML (англ. Unified Modeling Language) «унифицированном языке моделирования». Это графический язык, в котором каждый элемент (символ, стрелка, фигура) и сочетание элементов обладают конкретными значениями. UML позволяет визуализировать сложные явления и процессы в виде схем, понятных каждому, кто знаком с данной концепцией. Даже сложный и масштабный проект может быть описан несколькими связанными или отдельными UML-диаграммами.

Элементами диаграммы UML могут быть классы программного кода, страницы сайта, части механизма, зоны торгового зала — в зависимости от того, какой процесс или какую сущность описывает её создатель.[2]

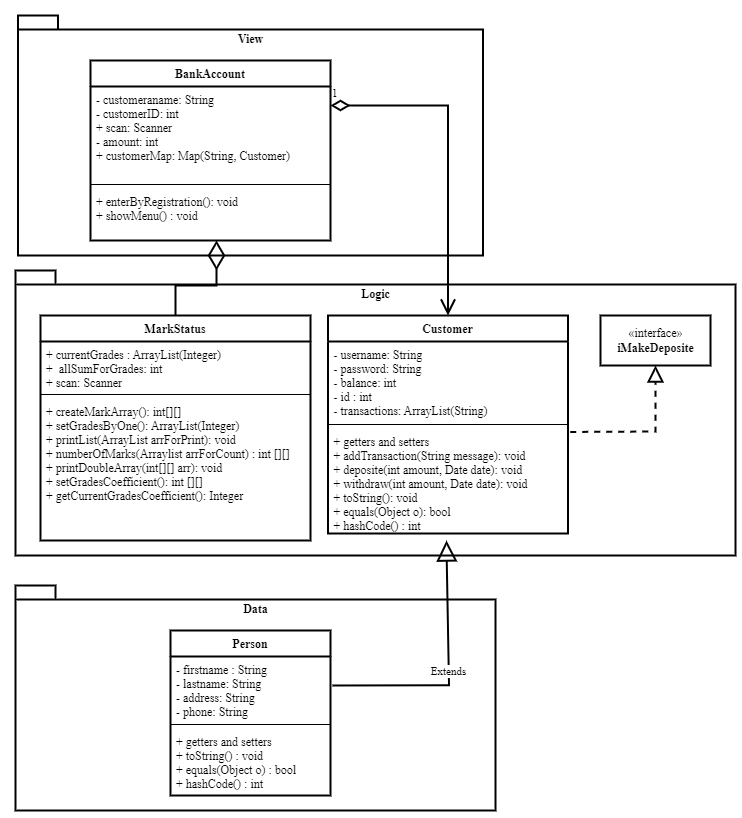
На рис. 2 представлена UML-диаграмма проекта IQKidBank. Здесь представлено три блока, Data, Logic и View, каждый из которых отвечает, соответственно, за данные, логику и представление приложения. Само приложение представлено четырьмя классами:

Рисунок 2. UML-диаграмма проекта IQKidBank

* **Person** содержит данные о персоне, некоем человеке, который взаимодействует с приложением.
* От класса Person наследуется класс **Customer** , представляющий непосредственно клиента банка, который может быть меценатом или получателем денежного эквивалента той или иной отметки.
* Интерфейс **iMakeDeposite**, реализованный в классе Customer, представляет возможность реализовать любой способ пополнения счета без вмешательства в класс Customer.
* Класс **MarkStatus** реализует логику перевода оценок успеваемости в денежный эквивалент.
* Класс **BankAccount** содержит основные меню вызова фунций, которыми может воспользоваться клиент приложения.

Даная UML не является полной и исчерпывающей, т.к. в проекте на текущий момент не представлено актуальное взаимодействие приложение с базой данных и не проработаны серверная и десктопная версии. Поэтому предлагается рассматривать имеющуюся диаграмму, как один из блоков проекта.

Так как текущая версия IQKidBank представляет собой скорее тестовую версию основных функций, на этом перечень архитектурной документации данного проекта исчерпан.

#### 1.3.3 Прочая документация

По стандартам архитектурной разработки программного обеспечения (далее ПО) для готового приложения понадобится следующая документация:

**GUI Графический интерфейс пользователя (ГИП)**, графический пользовательский интерфейс (ГПИ) (англ. graphical user interface, GUI) — система средств для взаимодействия пользователя с электронными устройствами, основанная на представлении всех доступных пользователю системных объектов и функций в виде графических компонентов экрана (окон, значков, меню, кнопок, списков и т. п.). <…>

Графический интерфейс пользователя является частью пользовательского интерфейса и определяет взаимодействие с пользователем на уровне визуализированной информации.[3]

Проще говоря, графический интерфейс – то, что видит пользователь, и с чем он работает (панели, кнопки, экраны приложения), а диаграмма GUI – визуализация этих элементов в виде набора изображений.

**ERD (структура базы данных)**. Диаграмма «Сущность-связь» (ERD, Entity-Relationship Diagram, ER-диаграмма) – это разновидность блок-схемы, где показано, как разные «сущности» (люди, объекты, концепции и так далее) связаны между собой внутри системы. ER-диаграммы чаще всего применяются для проектирования и отладки реляционных баз данных в сфере образования, исследования и разработки программного обеспечения и информационных систем для бизнеса. ER-диаграммы полагаются на стандартный набор символов, включая прямоугольники, ромбы, овалы и соединительные линии, для отображения сущностей, их атрибутов и связей. Эти диаграммы устроены по тому же принципу, что и грамматические структуры: сущности выполняют роль существительных, а связи – глаголов.[4]

**API (Application Programming Interface) - документация**. API – Application Programming Interface, что значит программный интерфейс приложения. В контексте API слово «приложение» относится к любому ПО с определенной функцией. Интерфейс можно рассматривать как сервисный контракт между двумя приложениями. Этот контракт определяет, как они взаимодействуют друг с другом, используя запросы и ответы. Документация API содержит информацию о том, как разработчики должны структурировать эти запросы и ответы.[5]

**Документация по тестированию (UAT, E2E, UNIT-тесты)**, принятая в компании или оговоренная с заказчиком при старте проекта или в ходе работы над ним.

### 1.4 Инструменты разработки

IQKidBank позиционируется, как простое банковское приложение или опция к существующему банковскому приложению. Для реализации любых операций с деньгами важны конфиденциальность данных, безопасность их передачи, безопасность взаимодействия с базами данных и отказоустойчивость.

Чуть ли не основная целевая аудитория данного приложения – дети и подростки, у которых в использовании могут быть различные гаджеты от компьютера до планшета и сотового телефона, соответственно, инструменты разработки должны обеспечивать его кросс-платформенность.

Если приложение будет востребованным – возникнет необходимость в его масштабировании, добавлении новых функций, разработке новых версий, соответственно инструменты разработки должны обеспечивать возможность создания новых функций и поддержания существующих с помощью разных библиотек[[2]](#footnote-3), и фреймворков[[3]](#footnote-4).

Для реализации проекта IQKidBank выбран высокоуровневый язык программирования Java.

#### 1.4.1 Язык программирования Java

Высокоуровневый язык программирования Java предлагает широкие возможности для создания простых банковских приложений, обеспечивая безопасность, платформенную независимость, простоту разработки и ряд других преимуществ.

Преимущества Java:

1. **Объектно-ориентированное программирование (далее ООП).**

Java предполагает полную поддержку принципов ООП, что позволяет даже начинающему программисту стремиться к разработке более чистого и структурированного кода. Этому способствует возможность разделения приложения на отдельные классы, каждый из которых выполняет определенную задачу.

1. **Безопасность.**

Любые операции с денежными средствами подразумевают необходимость безопасного взаимодействия с интерфейсом и базами данных. Использование виртуальной машины Java (далее JVM) и системы контроля доступа (Security Manager) обеспечивает полный контроль над доступом к ресурсам системы и позволяет предотвратить действия злоумышленников.

1. **Кроссплатформенность.**

«Write once, run anywhere» - слоган 1995 года, созданный Sun Microsystems для иллюстрации кроссплатформенных преимуществ языка Java [6]. В вольном переводе на русский язык слоган можно перевести как «один раз написанный код, запускается где угодно». Приложение, написанное на Java, будет работать на любой платформе, поддерживающей JVM без дополнительных усилий по его ассимиляции.

1. **Обилие библиотек.**

Java имеет огромное сообщество разработчиков и, соответственно, множество фреймворков и библиотек, упрощающих разработку любых, в том числе и банковских, приложений. Например, Java EE (Java Enterprise Edition) предоставляет мощный набор инструментов для создания распределенных и масштабируемых приложений.

1. **Многопоточность.**

Java предлагает встроенную поддержку многопоточности, что позволяет более продуктивно и быстро обрабатывать одновременные запросы от нескольких клиентов.

1. **Дружелюбная среда разработки.**

Немаловажный фактор для начинающих программистов – возможность реализации своих замыслов в максимально подготовленной среде. Intelij Idea предлагает богатый функционал для разработки, тестирования и создания документации в процессе разработки ПО.

#### 1.4.2 Среда разработки IntelliJ Idea

Интегрированная среда разработки, ИСP (англ. integrated development environment — IDE), также единая среда разработки, ЕСР — комплекс программных средств, используемый программистами для разработки программного обеспечения (ПО).[7]

В разработке дипломного проекта использована среда IntelliJ Idea (далее IDE). Данная среда упрощает работу с кодом и подготавливает рабочее окружение. IDE содержит ряд преимуществ для начинающего программиста:

* Мультиязычность. Освоив функционал среды можно программировать не только на Java, но и на других языках (например, Kotlin, Phyton).
* IDE интегрирована с Github и Git. Созданный проект легко подгружается в систему контроля версий.
* Возможность настроить окружение. С помощью IDE можно не только писать код, но и задавать GUI, покрывать код тестами.
* Доступность. Бесплатная версия обладает всем доступным функционалом для разработки приложений.

Так же IDE подсвечивает возможные ошибки, следовательно, до этапа компиляции можно привести код в порядок; помогает быстро переименовать любой элемент программы, будь то переменная или класс, во всех местах использования; генерирует блоки кода (геттеры, сеттеры, конструкторы классов и т.д.) и выполняет множество других вспомогательных функций.

### 1.5 Среда окружения и вспомогательные ресурсы (допущения).

Среда окружения (англ. Environment) — в информатике совокупность значений системных переменных, путей, открытых файловых дескрипторов и других ресурсов операционной системы, передаваемые процессу (программе) при его запуске.

#### 1.5.1 Банковское ПО

Несмотря на то, что в данной разработке регистрация клиента осуществляется при введении логина и пароля, IQKidBank - может быть реализован, как дополнение к инфраструктурным возможностям банка и действовать на основании существующих реальных счетов/карт родителей и детей. Соответственно, средой окружения для приложения может быть уже существующее банковское ПО.

#### 1.5.2 Дневник.ру

На этапе написания работы было выяснено, что технически возможно выгрузить текущие оценки из Дневник.ру как педагогу, так и родителю. Однако родителю доступны только четвертные оценки, в то время, как педагог может выгружать все проставленные оценки как по всему классу, так и по каждому отдельному ученику за любой период времени [9]. На основании вышесказанного в ходе работы над дипломом автором по умолчанию принята возможность импортировать в приложение и обрабатывать подобные данные.

#### 1.5.3 Система контроля версий Git и онлайн-сервис Github

С помощью Git программисты и разработчики ориентируются в коде и отслеживают изменения. Git помогает вернуть файлы в исходное состояние и видеть изменения, внесённые в определённый период. Разработчик выполняет разные команды (например, commit, push), а все изменения синхронизируются с центральным репозиторием.

**Git** — это система контроля версий, а **GitHub** — онлайн-сервис, по сути, социальная сеть. Одна из основных целей GitHub — быть единым местом для проектов с исходным кодом. Предполагается, что пользователь делится чем-то полезным, а другие люди смогут участвовать в разработке.

Ещё один вариант — использовать GitHub как хранилище проектов для портфолио: легко дать на них ссылку.[10]

Использование этих ресурсов обеспечивает возможность работать над одним проектом одновременно нескольким программистам, контролировать вносимые изменения и, при необходимости, откатывать их до любого из предыдущих состояний.

## Создание банковского приложения

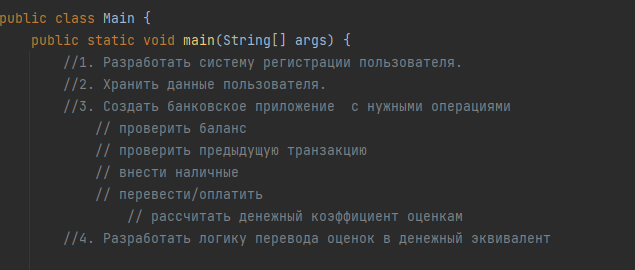
Для реализации приложения необходимо определиться с его структурой и функциональностью созданных классов. Задачи приложения IQKidBank в первом приближении будут выглядеть так:

Рисунок 3. Основные функции приложения

Для реализации этих задач необходимо определиться с основными сущностями ПО.

### 2.1 Классы Person и Customer

Одной из ключевых сущностей программы будет класс актора Customer, унаследованный от класса Person, экзепляр которого представляет собой пользователя приложения IQKidBank.

#### 2.1.1 Наследование классов

Одной из ключевых сущностей приложения является, конечно же, клиент банка. Без создания экземпляра этого класса невозможно авторизоваться и подключиться к пользовательскому интерфейсу.

Для реализации этого класса необходимо создать абстрактный родительский класс Person. С точки зрения объектно-ориентированного программирования создание отдельного родительского класса Person для класса Customer позволяет реализовать следующие принципы объектно-ориентированного программирования (далее ООП):

**Наследование:** между классами Person и Customer с помощью ключевого слова “extends” будут установлены отношения наследования. Это позволит классу Customer наследовать атрибуты и методы класса Person, что позволит использовать код повторно и уменьшит его дублирование.

**Полиморфизм**: когда Customer является подклассом Person, реализуется принцип полиморфизма, что позволяет работать с объектами типа Customer так же, как с объектами типа Person. Это упрощает обработку объектов различных типов и обеспечивает более гибкую работу с ними.

**Инкапсуляция:** создание отдельного класса Person позволяет лучше разделять ответственность между классами и улучшает инкапсуляцию данных. Класс Person содержит общие атрибуты и методы, такие, как имя, фамилия, адрес, номер телефона, а класс Customer - специфические для клиента банка: id, баланс, логин и пароль.

**Расширяемость:** при добавлении новых типов пользователей (например, класс Employee, Supplier) легко можно создать их как подклассы класса Person, что обеспечивает унифицированный подход к работе с различными типами пользователей.

Таким образом, создание отдельного родительского класса Person для класса Customer помогает повысить переиспользуемость, гибкость и структурированность кода в рамках объектно-ориентированного программирования.

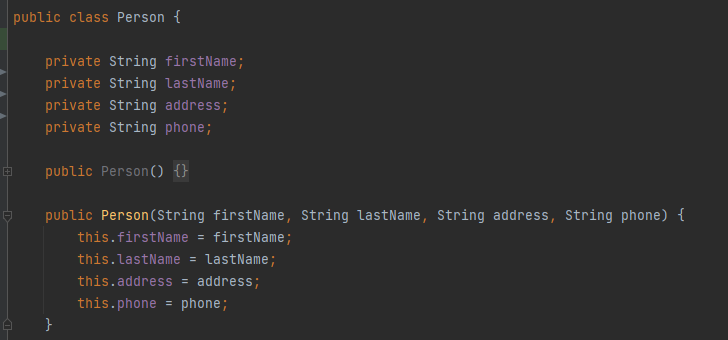
Согласно обозначенным выше принципам, полям класса Person будут присвоены общие для каждого человека данные, а методы родительского (или супер-класса) Person будут ограничены геттерами, сеттерами, конструктором и переопределением методов базового класса Object.

Рисунок 4. Поля и конструкторы класса Person

#### 2.1.2 Геттеры, сеттеры

Прежде, чем говорить о функциях геттеров и сеттеров в коде, необходимо сказать пару слов о модификаторах доступа в Java.

Модификаторы доступа в программировании определяют уровень доступа к классам, методам и переменным. Существует четыре основных модификатора доступа:

1. public - доступен из любой части программы;

2. private - доступен только внутри класса, в котором он определен;

3. protected - доступен внутри класса, в котором он определен и его подклассах;

4. default (package-private) - доступен только внутри пакета.

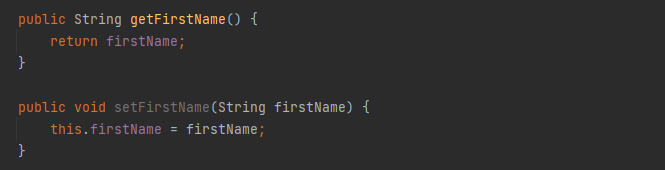
Геттеры и сеттеры — это специальные методы, которые используются для получения и установки значений приватных переменных класса (с модификатором доступа private). Они позволяют управлять доступом к данным и обеспечивают инкапсуляцию (скрытие данных от внешнего доступа), что позволяет предотвратить непосредственное изменение или чтение значений переменных объекта извне.

Рисунок 5. Пример использования геттера и сеттера переменной firstName с модификатором доступа private в коде IQKidBank

C помощью геттеров и сеттеров можно установить и, в дальнейшем, менять логику доступа к данным того или иного объекта, например, проверять значения перед присвоением или возвращать обработанные данные. Это позволяет обеспечить целостность данных и избежать несанкционированного доступа к ним.

Применение геттеров и сеттеров при модификаторе доступа public не является обязательным, так как данные объекта уже доступны извне и их можно менять напрямую. Однако, даже при использовании модификатора доступа public, рекомендуется использовать геттеры и сеттеры для обеспечения инкапсуляции данных и контроля доступа к ним, что поможет сделать код более надежным и избежать проблем при изменении логики доступа к данным в будущем.

#### 2.1.3 Конструкторы

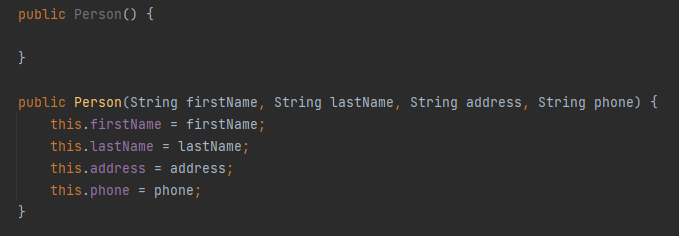
Возможность создания новых экземпляров любого класса IQKidBank будет реализована с помощью конструкторов.

Рисунок 6. Два примера конструкторов класса Person проекта IQKidBank

Как сказано в «Изучаем Java» 2-е издание, авторы: Кэти Сьерра и Берт Бейтс: «**Конструктор** выглядит и ведет себя как метод, но это все же не метод. Он отвечает за код, который выполняется, когда вы пишете ключевое слово “new”. Иными словами, это код, который запускается при создании экземпляра класса. <…> Вы можете создать конструктор для своего класса <…>, но даже в ином случае компилятор сам создаст его для вас. <…>

Ключевая особенность конструктора – он выполняется до того, как объект может быть присвоен ссылке. Это означает, что у вас есть шанс вмешаться в процесс и подготовить объект для дальнейшего применения.[11]».

Проще говоря, конструкторы в Java - это специальные методы, которые используются для создания и инициализации объектов при их создании. Конструкторы имеют тот же имя, что и класс, в котором они определены, и не возвращают значения. Основная задача конструкторов - задать начальные значения свойств объекта, подготовить его к использованию и обеспечить безопасное создание экземпляра класса[[4]](#footnote-5).

Ниже перечислены основные принципы создания конструкторов:

1. Конструктор имеет тот же имя, что и класс, и не возвращает значения.

2. Конструктор может быть перегружен - иметь различные параметры и служить для инициализации объекта в различных сценариях.

3. Если не определен никакой конструктор в классе, то компилятор автоматически создает конструктор без параметров (по умолчанию), который инициализирует объект.

4. Конструкторы могут вызывать другие конструкторы этого же класса с помощью ключевого слова "this".

С точки зрения объектно-ориентированного программирования (ООП) конструкторы являются специальными методами, которые обеспечивают создание и инициализацию объектов. Они позволяют устанавливать начальные значения полей объекта и гарантировать целостность и состояние объекта.

Касательно принципов SOLID, использование конструкторов в Java связано с принципом Single Responsibility Principle (Принцип единственной ответственности). Класс и его конструкторы должны иметь только одну причину для изменения. Конструкторы обеспечивают инициализацию объектов, что относится к их основной обязанности. Поэтому следует стремиться к тому, чтобы каждый конструктор выполнял только одну задачу, был понятен и прост в поддержке.

Общим правилом при разработке конструкторов в Java их является простота, ясность и эффективность.

#### 2.1.4 Переопределение методов базового класса Object

Переопределение методов toString(), equals() и hashCode() базового класса Object в Java означает создание своей собственной версии метода, который уже существует в базовом классе Object[[5]](#footnote-6). В основном это касается методов toString(), equals() и hashCode(), которые есть у всех объектов в Java.

#### 2.1.4.1 Переопределение метода toString() для класса Person



Рисунок 7. Переопределение метода toString() для класса Person

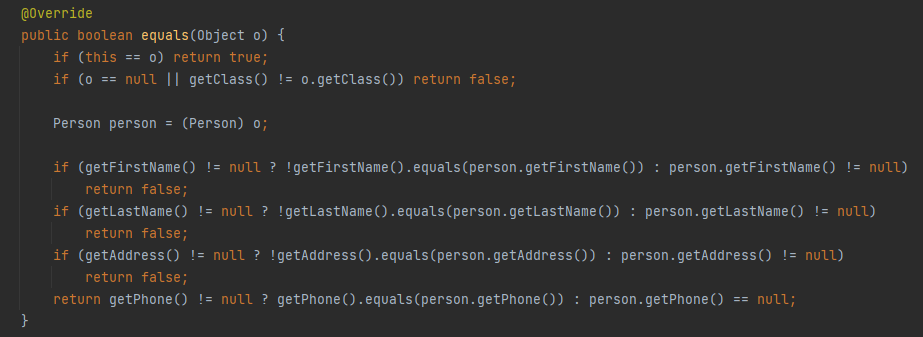
Метод toString() является методом класса Object и предназначен для возвращения строкового представления объекта. При вызове метода toString() для объекта, по умолчанию возвращается строка, содержащая имя класса объекта, знак '@' и хэш-код[[6]](#footnote-7) объекта.

В случае переопределения метода toString() в классе, мы можем определить собственное строковое представление объекта, что часто полезно для удобного вывода информации об объекте.

В приведенном примере кода метод toString() был переопределен в классе Person. В данном случае он возвращает строку, содержащую значения полей объекта - firstName, lastName, address и phone. Таким образом, при вызове метода toString() для объекта класса Person будет возвращена строка, содержащая информацию об имени, фамилии, адресе и телефоне этого объекта.

Переопределение метода toString() является хорошей практикой программирования, так как позволяет удобно выводить информацию об объектах для отладки или для представления пользователю.

#### 2.1.4.2 Переопределение метода equals() для класса Person

Далее необходимо переопределить метод equals(Object o) для класса Person, чтобы определить критерии сравнения двух объектов этого типа.

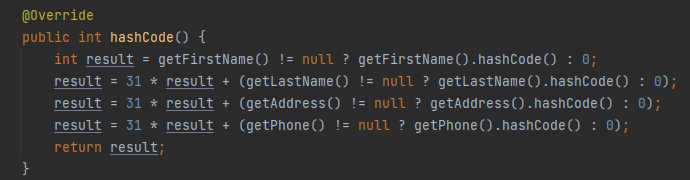
По умолчанию метод equals() будет сравнивать ссылки на объекты, а не их содержимое. Переопределение этого метода позволяет определить собственные условия для сравнения объектов, основанные на полях класса (в данном случае - firstName, lastName, address, phone).

Рисунок 8. Переопределение метода equals в родительском классе Person

В данном коде реализовано сравнение каждого поля объекта Person на равенство, чтобы определить, являются ли они эквивалентными. Если хотя бы одно поле отличается, то объекты будут считаться различными.

Переопределение метода equals() может быть полезно, например, при поиске объектов в коллекциях или при проверке на наличие дубликатов.

#### 2.1.4.3 Переопределение метода hashCode() для класса Person

Метод hashCode() в Java используется для вычисления хэш-кода объекта, который затем может быть использован в хеш-таблицах, например, для поиска объекта или оптимизации работы с коллекциями.

Переопределение метода hashCode() необходимо, всякий раз при переопределении метода equals() или использовании объектов в коллекциях, типа HashSet, HashMap и других, которые используют хэш-код для оптимизации работы.

Рисунок 9. Переопределение метода hashCode() класса Person

В данном коде метод hashCode() переопределен для объекта родительского класса Person. В методе с помощью значений полей класса вычисляется хэш-код для каждого из них (полей), далее значения объединяются в итоговый результат.

Переопределение метода hashCode() в данном случае позволяет правильно вычислить хэш-код объекта, что важно для корректной работы с коллекциями и поиска объектов в них по хэш-коду.

### 2.2 Класс Customer, наследованный от класса Person и имплементирующий интерфейс iMakeDeposite

Класс пользователя Customer создан наследованием от класса Person и имплементирует интерфейс iMakeDeposite.

#### 2.2.1 Оправданность использования интерфейса iMakeDeposite с точки зрения ООП

Имплементация метода deposite() из интерфейса iMakeDeposite в классе Customer позволяет гарантировать, что у всех объектов класса Customer будет реализован данный метод. Это упрощает работу с различными объектами, которые могут быть типизированы как Customer. К тому же это позволит при необходимости переопределить метод deposit().

Использование интерфейса в данном случае соответствует принципам ООП, так как позволяет использовать механизм полиморфизма. Это означает, что объекты разных классов, реализующих интерфейс iMakeDeposite, могут быть использованы одинаково, что делает код более гибким и упрощает его поддержку и расширение. Так же использование интерфейса повышает уровень абстракции и уменьшает связанность кода, что является одними из основных принципов ООП[[7]](#footnote-8).

### 2.2.2 Класс Customer

Класс Person представляет общие характеристики и свойства человека, такие как имя, фамилия, адрес и телефон. Класс Customer, в свою очередь, является дочерним классом класса Person и расширяет родительский класс, представляя дополнительные характеристики, специфичные для клиента, такие как имя пользователя, пароль, баланс, ID и список транзакций.

Рисунок 10. Поля класса Customer

#### 2.2.2.1 ArrayList<String> transactions для записи транзакций и использование дженериков

В данной версии проекта IQKidBank хранение информации о 10 последних транзакциях обеспечивается списком ArrayList<String> transactions, принимающим на вход только строковые данные (тип String), что подтверждается использованием дженерика[[8]](#footnote-9) <String> при создании ArrayList.

Дженерики в Java - это механизм обобщений, который позволяет создавать обобщенные классы, интерфейсы и методы, специализированные для работы с конкретными типами данных. Использование дженериков позволяет создавать более безопасный, конкретизированный и удобочитаемый код.

В данном случае, использование дженерика <String> в объявлении ArrayList<String> transactions означает, что список transactions будет хранить только объекты типа String. Это позволяет ограничить типы объектов, которые могут быть добавлены в список, только типом String, что упрощает работу с данными и страхует разработчика от ошибок типизации.

Использование дженериков соответствует одному из принципов SOLID[[9]](#footnote-10) - Принципу подстановки Барбары Лисков (Liskov Substitution Principle), который гласит, что объекты одного типа могут быть заменены объектами подтипов этого типа без изменения свойств программы. Использование дженериков с типом String в списке transactions обеспечивает строгую типизацию элементов списка, что позволяет сохранить принцип подстановки Барбары Лисков и обеспечивает безопасную работу с данными.

#### 2.2.2.2 Конструктор класса Customer

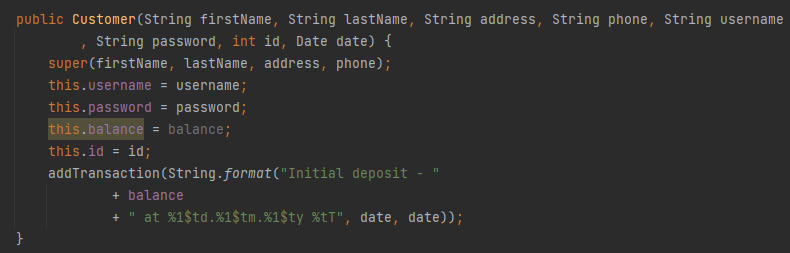
В теле конструктора класса Customer создан метод addTransaction(), позволяющий добавить информацию о начальном депозите клиента в список транзакций и указать актуальную дату данного изменения сразу при создании объекта класса Customer. Это позволяет инициализировать объект с определенным начальным балансом и добавить запись о начальном депозите в список транзакций, что может быть полезно для отслеживания истории финансовых операций клиента и дальнейшем учете и анализе счета.

Рисунок 11. Конструктор класса Customer с методом addTransaction()

#### 2.2.2.3 Метод addTransaction () класса Customer

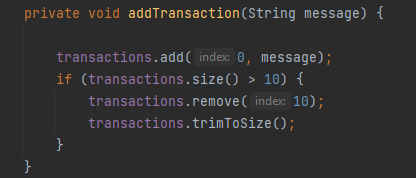


Рисунок 12. Метод addTransaction

Метод addTransaction() проверят список transactions на кол-во элементов. Если их больше 10, метод удаляет последний элемент с индексом 10 и вставляет новые данные в начало списка (индекс 0), реализуя тем самым способ организации структуры данных по принципу FIFO (с англ. «first in first out»). В FIFO первый элемент, добавляемый в структуру данных, — это тот же самый первый элемент, который должен быть удален первым. Таким образом, в тестовой версии IQKidBank остаются только актуальные (последние транзакции).

#### 2.2.2.4 Метод trimToSize() класса Customer

Метод trimToSize() используется для уменьшения емкости списка данных до его текущего размера, что позволяет избежать утечек памяти на обработку лишних данных. Утечка данных практически незаметна при столь малых объемах, однако, может сказаться не лучшим образом при работе с большими объемами данных.

Помимо стандартных конструкций: геттеров, сеттеров, переопределения методов базового класса Object toString(), equals() и hashCode(), подробно разобранных в предыдущей части «Класс Person», класс Customer содержит два важных метода:

* переопределенный метод интерфейса iMakeDeposite, deposite();
* метод withdraw().

#### 2.2.2.4.1 Формат даты в методе addTransaction()

Метод String.format в Java используется для форматирования строки с использованием спецификаторов формата. Он позволяет вставлять значения переменных в заданный шаблон строки, указывая типы данных и их порядок.

В данном коде следующая строка :

amount  
 + " withdrawed from your account. Balance - "  
 + balance  
 + " at %1$td.%1$tm.%1$ty %tT"

использует спецификаторы формата для вставки значений переменных в нужном формате:

- %1$td - этот спецификатор означает, что первое переданное значение (date) будет отформатировано как день месяца (число);

- %1$tm - указывает на месяц;

- %1$ty - указывает на двузначный год;

- %tT - указывает на время.

Таким образом, при использовании метода String.format() с этим шаблоном строки и переданным значением date, произойдет замена спецификаторов на соответствующие значения из объекта date, отформатированные в соответствии с заданными форматами. Это нужно для генерации строки с информацией о списании средств, балансе и времени проведения операции в наиболее удобном для пользователя формате.

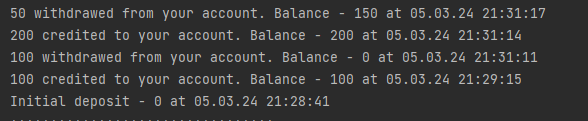


Рисунок 13. Вывод формата даты через метод .format()

Стандартный формат вывода даты и времени в Java без использования метода .format() выглядит примерно так:

Thu Sep 24 15:28:09 UTC 2021

Этот формат может быть непонятен пользователю из-за того, что он не представлен в удобочитаемом виде. Он содержит названия дней недели и месяцев на английском языке, а также часы, минуты, секунды и часовой пояс. Рекомендуется всегда использовать метод .format() для вывода даты в понятном и читаемом формате.

#### 2.2.2.5 Метод deposite () класса Customer

Метод deposite() в данном случае является переопределенным методом имплементированного интерфейса iMakeDeposite.

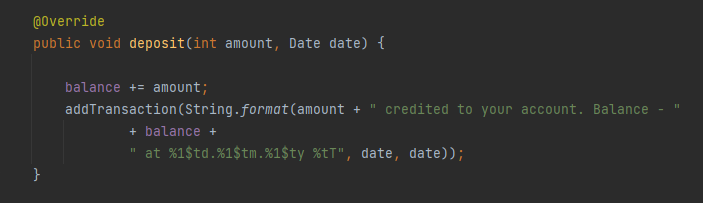


Рисунок 14. Метод deposite() класса Customer

Метод deposit() используется для зачисления денежных средств на счет. Как только метод вызывается, указанная сумма (amount) добавляется к переменной balance, которая представляет текущий баланс на счету.

Далее, метод addTransaction() вызывается для добавления информации о транзакции в список. Строка, представляющая информацию о транзакции, формируется с помощью метода String.format() в Java. В этой строке содержится информация о сумме зачисления, текущем балансе, а также дате и времени транзакции. Затем эта строка добавляется в список транзакций.

Например, если метод deposit вызывается с параметрами (100, new Date()), то сумма 100 будет зачислена на счет, текущий баланс увеличится на 100. Далее, информация о данной транзакции будет добавлена в список транзакций в виде строки "100 credited to your account. Balance - (новый баланс) at (дата и время транзакции)".

Этот метод полезен для отслеживания всех зачислений на счет и обеспечивает прозрачность и учет всех финансовых операций.

#### 2.2.2.6 Метод withdraw() класса Customer

Метод с сигнатурой[[10]](#footnote-11) withdraw(int amount, Date date) предназначен для снятия определенной суммы денег со счета.

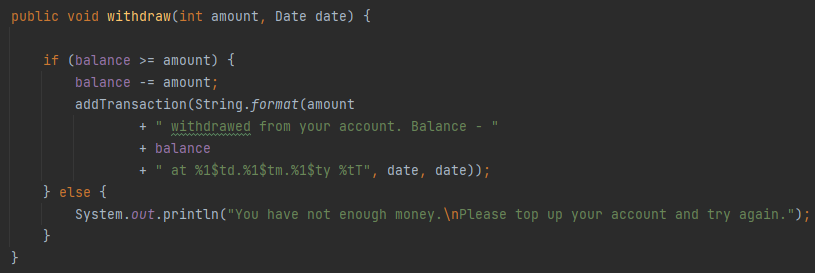


Рисунок 15. Метод withdraw класса Customer

На вход подаются следующие параметры:

- amount: целочисленное значение, которое указывает на сумму, которую необходимо снять со счета.

- date: объект типа Date, представляющий дату и время выполнения операции снятия денег.

В процессе реализации метода происходит проверка, достаточно ли средств на счете для выполнения операции. Если текущий баланс (переменная balance) больше или равен запрашиваемой сумме (amount), то из текущего баланса вычитается сумма (amount), средства снимаются со счета. Далее выполняется метод addTransaction() с аргументом, который формируется с помощью метода String.format(). Этот метод добавляет транзакцию в журнал операций, указывая сумму, снятую со счета, новый баланс после операции, дату и время выполнения операции.

Если на счете недостаточно средств для выполнения операции, то выводится сообщение с предупреждением и просьбой пополнить счет.

Итак, этот метод позволяет снимать деньги со счета, при этом проверяя наличие необходимой суммы на балансе и регистрируя операцию в журнале транзакций.

### 2.3 Класс MarkStatus

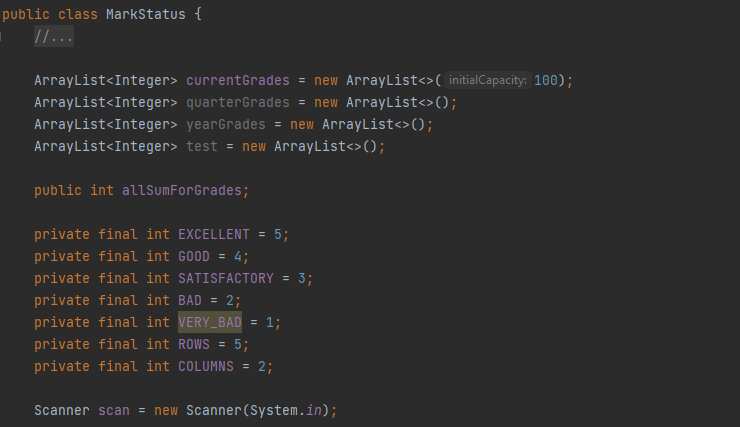
Основной интерес в рамках данной работы представляет расчет денежного коэффициента оценок. Для реализации этой задачи при каждом новом расчёте будет создаваться экземпляр класса MarkStatus.

Рисунок 16. Поля класса MarkStatus проекта IQKidBank

Из реализованных на данный момент функций этой версии приложения, ключевой является расчет денежного коэффициента за оценки, введенные в программу пользователем, который назначает коэффициент поощрения для каждого из класса оценок: за пятерки, четверки, тройки, двойки и единицы.

Соответственно, из полей класса, представленных структурами данных ArrayList, будет использован только массив currentGrades, в который будут записаны текущие оценки, введенные пользователем.

#### 2.3.1 Константы (private final переменные) и класс Enum

Отдельного внимания заслуживают private final переменные, которые будут соответствовать школьным оценкам от единицы (1) до пятерки(5). В Java ключевое слово final используется для объявления констант[[11]](#footnote-12). Когда примитивная переменная объявляется с использованием ключевого слова final, это означает, что значение этой переменной не может быть изменено после ее инициализации. Обычно в Java принято именовать переменные final в верхнем регистре с подчеркиваниями между словами, чтобы отличить их от обычных переменных. Этот стиль напоминает об именовании констант по соглашению об именовании в Java[[12]](#footnote-13).

При дальнейшей работе над кодом эти значения можно будет перенести в отдельный класс типа enum.

Перечисление (enum) в Java — это тип данных, который позволяет определить набор констант с заданными значениями.

Перечисления могут быть использованы для создания списка возможных значений, которые могут использоваться в <…> программе. Например, <…> в программе для управления автомобилем, <…> можно создать перечисление с возможными состояниями двигателя, такими, как «Запущен», «Выключен» и «Перегрев».[16] Или, как в данном случае, сформировать оценки EXCELLENT, GOOD, SATISFACTORY, BAD, VERY\_BAD.

Enum в Java помогает сделать код более читаемым, безопасным и поддерживаемым. Он также обеспечивает простой способ организации и перечисления констант.

#### 2.3.2 Scanner в Java

Класс Scanner(сканер) в Java представляет собой удобный инструмент для считывания данных из различных источников, таких как консоль, файлы или строки. Он предоставляет методы для считывания примитивных типов данных, строк и даже регулярных выражений. В приложении IQKidBank переменная scan типа Scanner используется для чтения информации, вводимой пользователем из консоли.

Использование сканера обусловлено рядом преимуществ, в частности для новичка в программировании:

* **Простота использования.** Класс Scanner предоставляет простые методы для считывания различных типов данных. А компактность кода делает его максимально удобным и простым в использовании.
* **Гибкость.** Scanner может считывать данные из различных источников, что делает его универсальным инструментом для ввода данных в программу.
* **Мощные функции.** Scanner поддерживает не только считывание примитивных типов данных, но и работу с регулярными выражениями, что означает гибкость при обработке ввода.

Однако, вместе с очевидными преимуществами, Scanner имеет и свои недостатки, к числу которых относятся:

* + **Низкая производительность.** Использование Scanner для считывания больших объемов данных может привести к замедлению работы программы из-за медленной обработки ввода.
  + **Ограниченные возможности.** В классе Scanner отсутствуют некоторые функции для более сложной обработки данных. Например, возможность работы с нестандартными форматами или специфическими типами данных.
  + **Уязвимость к ошибкам.** При неправильном использовании методов класса Scanner могут возникать критичные для работы программы ошибки, например, исключения при неправильном форматировании ввода или некорректных данных.

В целом, класс Scanner в Java является удобным инструментом для считывания данных из различных источников, однако перед его использованием необходимо учитывать как преимущества, так и недостатки объектов этого типа, чтобы избежать потенциальных проблем в работе программы.

#### 2.3.3 Методы класса MarkStatus

Перейдем к методам класса MarkStatus, назначение которых – обеспечить корректный и гибкий перевод оценок в денежный эквивалент. Проработка только четвертных оценок в тестовой версии не исключает дальнейшего расширения класса методами с такими функциями, как:

* установка дополнительного бонусного коэффициента по учету процента тех или иных оценок (например, бонус к итоговой сумме +5%, если количество пятерок превышает 50% от количества всех оценок);
* установка дополнительного бонусного коэффициента к четвертным и годовым оценкам;
* сохранение настроек расчета;
* сохранение и импорт введенных оценок и результатов расчёта;
* и так далее.

##### **2.3.3.1 Метод createMarkArray() класса MarkStatus**

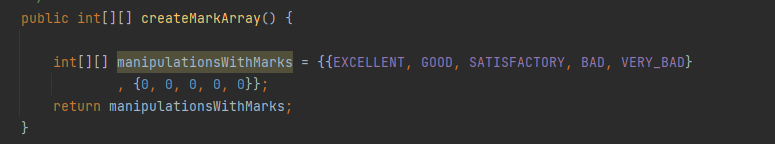
Данный метод создает двумерную матрицу для массива оценок, первая строка которой - школьные оценки от 1 до 5, вторая - денежные коэффициенты за каждую из них.

Рисунок 17. Метод createMarkArray класса MarkStatus

В целом двумерные массивы (матрицы) типа int[][] удобны и эффективны для работы с небольшим количеством данных в двумерном формате. К элементам массива можно быстро обратиться, зная их индекс.

Единственный минус использования этого типа в коде – необходимость использования вложенных циклов для доступа к элементам (например, для распечатки массива), что усложняет код для понимания.

##### **2.3.3.2 Метод setGradesByOne() класса MarkStatus и цикл do-while и оператор break**

Метод setGradesByOne() класса MarkStatus используется для ввода оценок в массив currentGrades типа ArrayList<Integer> по одной за раз. Метод запрашивает ввод оценок в консоль через enter, до тех пор, пока пользователь не введет -1 для завершения ввода.

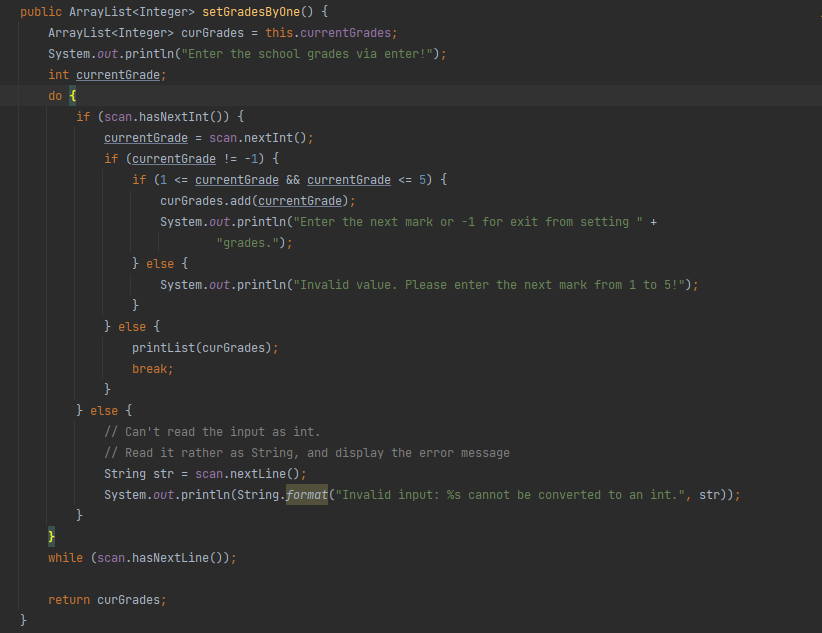


Рисунок 18.Метод setGradesByOne() класса MarkStatus

В цикле do-while выполняются проверки на неверный формат ввода данных и корректность введенных данных. Первая проверка проводится с помощью метода hasNextInt() на то, является ли следующий ввод целым числом. Вторая проверка контролирует, что введенное число не меньше 1 и не больше 5, т.к. именно таков диапазон школьных оценок в РФ на данный момент.

Если значение удовлетворяет всем условиям, оно добавляется в массив curGrades. Если же нет – пользователь получает уведомление об ошибке и имеет возможность продолжить ввод.

Цикл do-while является одним из базовых циклов в большинстве языков программирования. Он отличается от более простого цикла while тем, что блок кода выполняется хотя бы один раз, даже если основное условие не выполнено. Этот цикл прост в использовании и понятен в действии. Благодаря этому и гарантированному выполнению кода хотя бы один раз, является очевидным выбором для начинающих программистов.

Эта, простая, на первый взгляд, конструкция, однако же может весьма коварно вести себя на практике. Неправильное использование может привести к бесконечному циклу, блокировке программы или другим нежелательным последствиям. Также при использовании цикла do-while важно понимать, что большая вложенность может усложнить читаемость кода (что противоречит принципам SOLID), а неаккуратно выбранное условие выхода может повлечь за собой серьезные ошибки в выполнении программы.

В целом, цикл do-while является мощным инструментом в программировании, но его использование требует внимательности и осторожности во избежание ошибок и нежелательных последствий.

Оператор break применяется для прерывания текущей итерации (break (broke, broken) — ломать, разрывать). С его помощью происходит выход из блока фигурных скобок оператора цикла либо оператора switch с дальнейшей передачей управления следующему оператору. Если задействуются вложенные операторы, break обеспечивает выход из самого внутреннего оператора.[17]

##### **2.3.3.3 Методы printList() и printMatrix() класса MarkStatus, цикл for**

Для наглядности расчетов пользователю может потребоваться возможность убедиться в том, что введенные данные корректны, как и их обработка. В связи с этим в приложении необходим функционал, выводящий на экран или в консоль различные данные, структурированные в виде матрицы или массива.

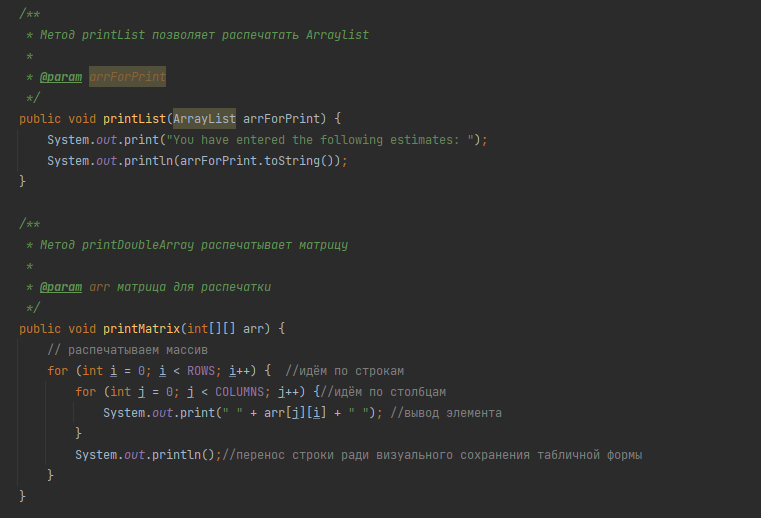
В первом методе, printList() использован метод Абстрактной коллекции(AbstractCollection.java[[13]](#footnote-14)) Java - .toString(). Из описания поведения метода: «Возвращает строковое представление этой коллекции. Строковое представление состоит из списка элементов коллекции в том порядке, в котором они возвращаются ее итератором, заключенного в квадратные скобки ("[]"). Соседние элементы разделены символами ", " (запятая и пробел). Элементы преобразуются в строки с помощью String.valueOf(Object). Возвращает: строковое представление этой коллекции.[19]

Рисунок 19.Методы printList() и printMatrix() класса MarkStatus

Метод printMatrix() предназначен для печати двумерного массива arr в виде матрицы. Он использует два вложенных цикла for: внешний цикл идет по строкам матрицы, а внутренний - по столбцам. На каждой итерации внутреннего цикла он выводит элемент массива arr[j][i] с использованием метода System.out.print(), добавляя пробелы для форматирования. Перевод строки System.out.println() добавляется после завершения каждой строки матрицы для разделения строк при выводе, что позволяет удобно отображать матрицу из двумерного массива на экране.

Цикл for в языке программирования Java является одним из самых распространенных и используется для повторения определенного блока кода определенное количество раз. Простой в использовании, с контролируемым количеством итераций и эффективный, цикл for предоставляет возможность управлять начальным значением, условием выполнения и шагом и является завсегдатаем любого кода.

Однако, он не удобен для работы с динамическими структурами данных, к тому же неправильно заданные условия могут привести к зацикленному коду[[14]](#footnote-15). Большая вложенность циклов for друг в друга и в другие циклы, как и в случае с do-while, делает код монотонным и усложняет его понимание и поддержку.

##### **2.3.3.4 Метод numberOfMarks() класса MarkStatus, конструкция ветвления**[[15]](#footnote-16) **if-else**

Принцип действия следующий: метод numberOfMarks() принимает список целочисленных значений arrForCount, представляющий оценки, введенные пользователем. Метод создает двумерный массив целых чисел countOfMarks с помощью метода createMarkArray(). Затем происходит итерация по каждому элементу списка arrForCount и, в зависимости от значения оценки, увеличивается соответствующий счетчик в массиве countOfMarks, который метод в итоге и возвращает. Данный массив содержит общее количество оценок EXCELLENT, GOOD, SATISFACTORY, BAD и VERY\_BAD.

Конструкция if-else является основным инструментом ветвления не только в Java, но и во многих других языках программирования C++, C#, Python, JavaScript, PHP, Ruby, Perl, Swift, и др.). Этот оператор позволяет задать определенные условия, при выполнении которых выполняются и определенные блоки кода.

Рисунок 20. Метод numberOfMarks() класса MarkStatus проекта IQKidBank

Среди основных преимуществ конструкции if-else можно назвать простоту использования и универсальность. Ведь эти операторы ветвления могут быть использованы для решения широкого круга задач. Правильно оформленные блоки ветвления делают код понятным и легко поддерживаемым.

С другой стороны, нарушения при оформлении блоков и большая вложенность могут, наоборот, усложнить читаемость и поддержку готового кода, а ошибки в управлении потоками могут привести к отказу приложения. Чрезмерное использование if-else может привести к нарушению принципа единственной ответственности, так как логика выбора будет распределена по разным частям кода.

Сложно вычленить в данном блоке ветвления соответствие принципам ООП, с точки же зрения SOLID операторы ветвления if-else могут отражаться на следующих принципах:

1. **Принцип единственной ответственности[[16]](#footnote-17) (SOLID**): хорошо структурированное использование if-else помогает соблюдать принцип единственной ответственности, разделяя логику выбора и действия, выполняемые в зависимости от выбора.

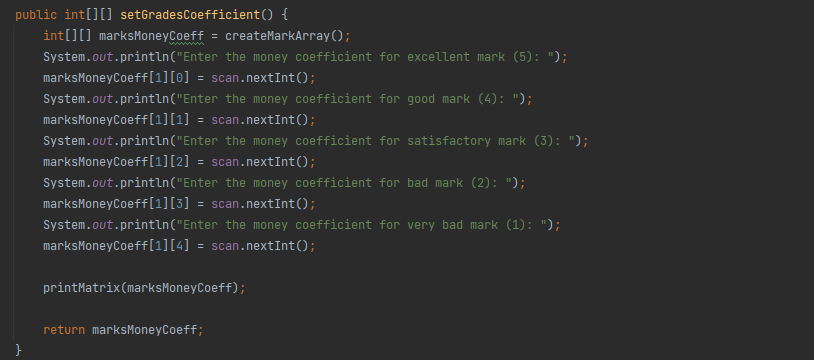
2. **Принцип открытости/закрытости[[17]](#footnote-18) (SOLID)**: if-else может нарушить принцип открытости/закрытости, если логика выбора изменяется часто и приводит к частым изменениям в коде.

3. **Принцип подстановки Барбары Лисков[[18]](#footnote-19) (SOLID)**: корректное использование if-else позволяет следовать принципу подстановки, если условия ветвления не изменяют поведение при замене объектов на их подклассы.

Таким образом, if-else - мощный инструмент в программировании, который, при правильном применении, может обеспечить логичное ветвление кода. Однако необходимо быть внимательным к его использованию, чтобы избежать негативных последствий для приложения и других разработчиков.

##### **2.3.3.5 Метод setGradesCoefficient() класса MarkStatus**

Метод setGradesCoefficient позволяет задать с консоли денежный коэффициент за каждую из оценок.

Рисунок 21.Метод setGradesCoefficient() класса MarkStatus

Метод setGradesCoefficient() используется для установки денежных коэффициентов оценок. Для этого метод создает двумерный массив marksMoneyCoeff, который представляет собой матрицу хранения коэффициентов для каждой отметки. Первая строка массива содержит номера оценок (от 1 до 5), а вторая - соответствующие коэффициенты, так что после их установки пользователь сможет в удобном формате увидеть назначенные коэффициенты для каждой отметки и тем самым проконтролировать величину итоговой суммы.

Затем метод запрашивает у пользователя ввод коэффициентов для каждого балла, начиная с отличной отметки (5) и заканчивая очень плохой (1), и сохраняет введенные значения в соответствующие ячейки массива.

После ввода всех коэффициентов метод выводит матрицу оценок и их коэффициентов на экран с помощью метода printMatrix() и возвращает эту матрицу.

##### **2.3.3.6 Метод getCurrentGradesCoefficient() класса MarkStatus**

Метод getCurrentGradesCoefficient() класса MarkStatus реализует итоговые вычисления денежного коэффициента за оценки, введенные пользователем, и возвращает сумму к переводу в виде целочисленной переменной типа int[[19]](#footnote-20).

Принцип действия метода следующий: сначала getCurrentCoefficient() вызывает метод setGradesByOne(), благодаря которому пользователь устанавливает случайные оценки для студента. Затем вызывается метод setGradesCoefficient(), который устанавливает коэффициенты для каждой оценки, и создает двумерный массив b.

Далее метод создает новый двумерный массив getSumMoneyForGrades, который будет хранить произведение оценок на их коэффициенты. Переменная allSumForGrades, куда будет впоследствии передана итоговая сумма, начисленная за оценки, инициализируется нулем.

Затем, с помощью вложенного цикла for, совершается проход по всем элементам массива a, который содержит оценки, и производится вычисление произведения оценки на заданный пользователем коэффициент. Если j не равно 0 (то есть не первый элемент), произведение добавляется к переменной allSumForGrades.

После этого метод выводит в консоль матрицу getSumMoneyForGrades и общую сумму allSumForGrades. Также метод вызывает метод setAllSumForGrades() для установки итогового значения allSumForGrades и возвращает его.

Таким образом, данный метод позволяет вычислить коэффициент текущих оценок студента на основе установленных оценок и их коэффициентов, а также вывести полученные значения в консоль.

### 2.4 Класс BankAccount

Итак, после того, как созданы сущности Person и Customer, описывающие клиента банка, MarkStatus, позволяющий хранить данные, введенные пользователем и реализующий логику расчета денежного эквивалента достижений (оценок) учащегося, необходимо создать класс, отвечающий за интерфейс, который увидит пользователь, и реализовать удобную систему управления опциями приложения IQKidBank.

#### 2.4.1 Пользовательский интерфейс и особенности его реализации в коммерческих приложениях

С точки зрения создания интерфейса для коммерческого приложения важно учесть множество нюансов, как то:

- **особенности целевой аудитории:** нужно понимать, кто будет пользоваться приложением и, исходя из этого, формировать общий макет ПО с разбивкой функциональных элементов по экранам (UI-UX).

- **цели приложения:** какие функциональные возможности ПО должны быть доступны пользователям и администраторам, какие элементы будут использованы для реализации этих возможностей.

- **дизайн интерфейса:** важно создать не только функциональное приложение, но и обеспечить привлекательный для целевой аудитории дизайн, который поможет пользователям легко ориентироваться в возможностях приложения и корректно взаимодействовать с ним. Нужно понимать, что дизайн мессенджера для подростковой аудитории будет разительно отличаться от, скажем, приложения с заметками для пожилой аудитории. Применительно к приложению IQKidBank, дизайн будет отличаться для взрослой аудитории, младших школьников, представителей средней и старшей школьной ступени.

- **разработка под разные платформы:** приложение должно корректно отображаться не только на экране смартфона, нужен адаптивный веб-дизайн[[20]](#footnote-21) для корректного отображения на других устройствах: компьютере, планшете.

- **эффективность:** для пользователя важно быстродействие и эффективность системы. Соответственно, приложение должно быстро откликаться на действия пользователя и не должно загружать систему.

- **тестировани**е: выпускать коммерческое приложение без предварительного тестирования – значит ставить на карту репутацию компании. Тестирование рабочих функций, отказоустойчивости приложения, а, так же, его интерфейса на различных устройствах и операционных системах для подтверждения корректности его работы и удобства использования – тот шаг перед релизом, который нельзя пропустить. Правда, некоторые компании обходят его, предлагаю пользователю бета-версию продукта и исправляя ошибки на основе отзывов, но такой подход может позволить себе только компания, которая не радеет за лояльность пользователей.

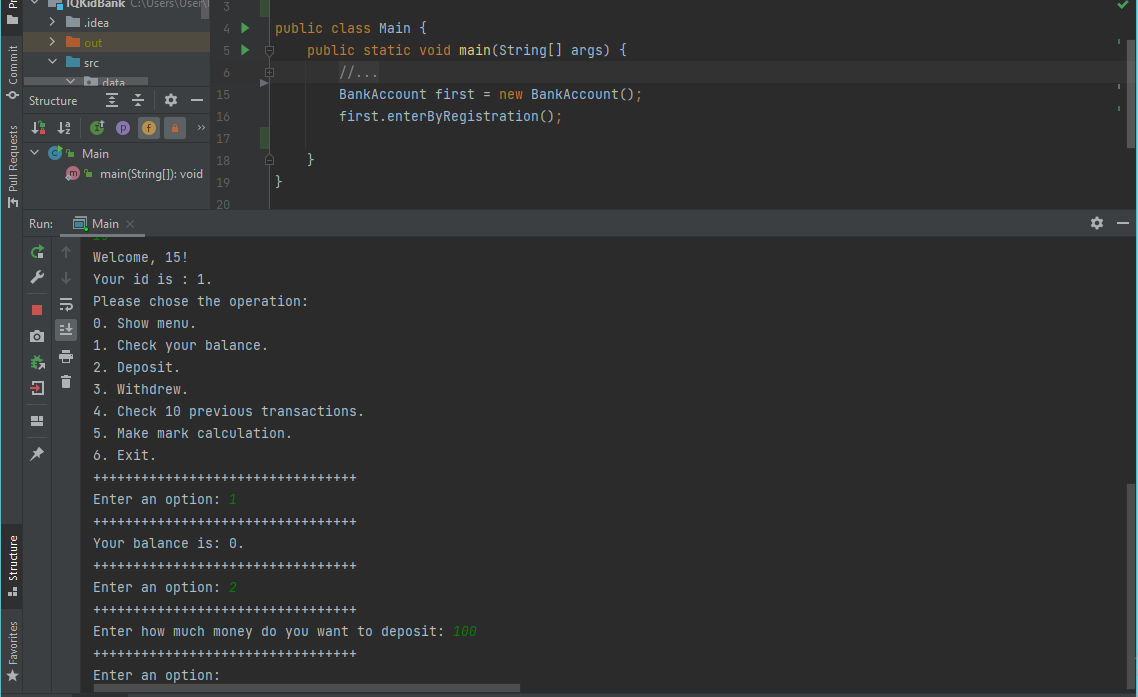
В данной работе все вышеописанные нюансы не находятся в фокусе внимания, т.к., в силу разных обстоятельств, проработан только необходимый код для базовой функциональности приложения IQKidBank. Да и в реальных проектах front-end разработкой занимается, как правило, специализированный сотрудник. Поэтому для тестовой реализации пользовательского интерфейса избран вариант консольного приложения[[21]](#footnote-22), вариант, где программа «общается» с пользователем (выдает информационные сообщения и запросы и принимает на вход сообщения пользователя) через консоль.

Рисунок 22. Пример консольного приложения на Java. Дипломная работа GeekBrains, проект IQKidBank, реализация автора.

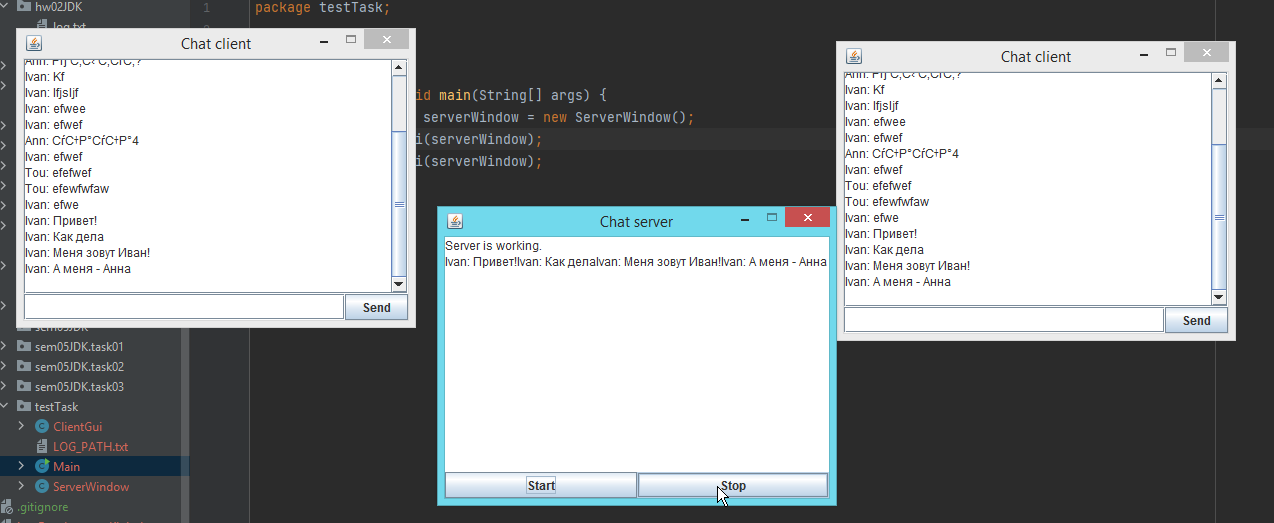
Преимущества консольного интерфейса для приложения, являющегося дипломной работой начинающего программиста очевидны:

1. **Простота написания кода и дальнейшего тестирования**: консольный интерфейс прост в реализации и предоставляет возможность прямого взаимодействия с программой, что упрощает проведение тестов.
2. **Быстрая отладка:** консольный интерфейс оптимален для быстрой проверки работоспособности отдельных компонентов программы, к тому же, благодаря подобной организации приложения, можно легко отслеживать работу компонентов приложения в реальном времени.
3. **Уменьшение количества зависимостей:** консольный интерфейс обычно не требует применения дополнительных библиотек и фреймворков для работы, что делает процесс работы с кодом в процессе написания, отладки и использования, более независимым и простым.

Конечно, нельзя забывать и о недостатках данного подхода, среди которых можно выявить:

1. **Ограниченность функциональности:** консольный интерфейс может быть неудобным в использовании для пользователей, привыкших к графическим интерфейсам. Здесь не идет речь о привычном для пользователи действии: «нажми на кнопку – получи результат», так что приложение может показаться аудитории неудобным и устаревшим.
2. **Отсутствие визуальной информации:** консольный интерфейс не позволяет в полной мере визуализировать данные или результаты работы программы, что может затруднить понимание работы программы.
3. **Техническое неудобство взаимодействия с командной строкой:** среди компаний-производителей ПО высока конкуренция за лояльность пользователей, поэтому удобству пользователя отводится немаловажная роль. С момента появления компьютеров работа с командной строкой считалась прерогативой «продвинутых пользователей», поэтому сейчас пользователь, не знакомый с командной строкой вряд ли будет разбираться с ней, скорее он отдаст предпочтение приложению с привычным оконно-кнопочным интерфейсом.

#### 2.4.2 Ещё немного о реализации оконных приложений



Исходя из вышесказанного, не вызовет удивления тот факт, что в последующих версиях приложения IQKidBank консольный подход будет изменен, и приложение будет выполнено, как полноценное оконное[[22]](#footnote-23), доступное для использования на разных устройствах.

Рисунок 23. Пример оконного приложения. Источник: домашнее задание GeekBrains, курс Java Development Kit, реализация автора.

В чем же преимущество оконного приложения перед консольным?

1. **Интерактивность:** оконные приложения обеспечивают более удобный и интерактивный пользовательский интерфейс, благодаря чему пользователи могут взаимодействовать с приложением непосредственно через графический интерфейс.

2. **Визуализация:** оконные приложения обладают возможностью использовать графику, анимацию и другие визуальные элементы для представления информации более привлекательным образом.

3. **Удобство использования:** оконные приложения обеспечивают более интуитивный и удобный способ работы с приложением для конечного пользователя, что может улучшить общий опыт использования.

4. **Мультитаскинг (мультизадачность):** оконные приложения позволяют пользователям одновременно запускать несколько программ и переключаться между ними, что облегчает одновременную работу с разными задачами.

5. **Легкость расширения:** оконные приложения поддерживают более простое и эффективное добавление новых функций и возможностей через встроенные плагины и расширения.

6. **Более привлекательный и современный вид:** оконные приложения часто непосредственно соответствуют современным трендам в дизайне и предоставляют более эстетически приятный внешний вид.

Для реализации оконного графического интерфейса приложения можно выбрать один из следующих путей:

1. **Использование библиотеки Swing.** Swing является стандартным набором библиотек для создания графического пользовательского интерфейса на платформе Java. Swing предоставляет большой набор компонентов, таких как кнопки, текстовые поля, окна и диалоговые окна, которые можно использовать для создания пользовательского интерфейса.

2. **Использование JavaFX.** JavaFX - это современная технология для создания интерактивных пользовательских интерфейсов в Java. Он предоставляет более широкие возможности для создания красивого и функционального пользовательского интерфейса, чем Swing.

3. **Использование библиотеки AWT (Abstract Window Toolkit).** AWT - это старый набор библиотек для создания графического пользовательского интерфейса на платформе Java. Он предоставляет более простой и менее гибкий способ создания интерфейса, чем Swing и JavaFX.

4. **Использование библиотеки Java GUI Builder.** Это инструмент, который позволяет создавать пользовательский интерфейс приложения визуально без необходимости писать код.

Это – наиболее распространенные варианты для реализации оконного пользовательского интерфейса. Конкретный способ реализации зависит от специфики приложения, требований к интерфейсу и опыта программиста.

#### 2.4.3 Поля класса BankAccount

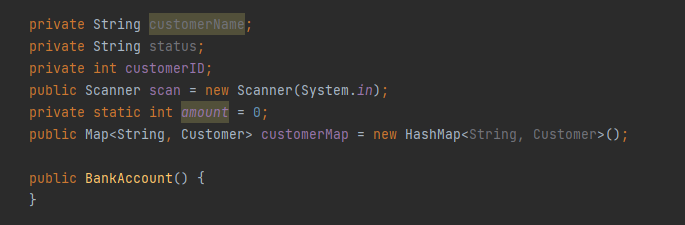
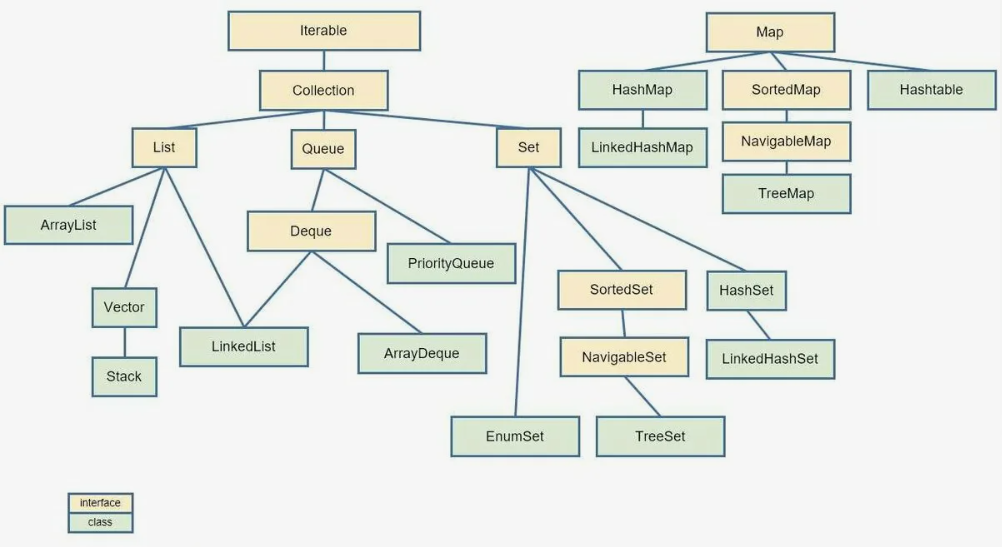
Экземпляр класса BankAccount позволит создавать отдельные банковские аккаунты для конкретного пользователя.

Рисунок 24.Поля класса BankAccount приложения IQKidBank

В полях класса будет находиться информация о логине пользователя (он же customerName), статусе пользователя (родитель/адресант и ученик/адресат перевода; status), id пользователя (customerId), так же здесь будет создан экземпляр класса Scanner для чтения информации, вводимой пользователем в процессе работы.

Отдельного внимания заслуживает структура данных Map, где будут храниться данные о пользователях, причем, ключом будет выступать логин пользователя, а значением – экземпляр класса Customer.

#### 2.4.4 Структура данных Map

Map - это абстрактный тип данных, который представляет собой отображение ключей на значения. В отличие от коллекций, которые хранят только объекты и позволяют получать к ним доступ по индексам или итерироваться по ним, Map хранит пары "ключ-значение", где каждый ключ связан с соответствующим ему значением.

Таким образом, Map не является коллекцией, потому что хранит не просто набор элементов, а структуру данных, которая предназначена для быстрого поиска элемента по ключу. В то время, как коллекции управляются интерфейсами Collection и Iterable, Map управляется интерфейсами Map и SortedMap (если требуется сортировка).[23]

Рисунок 25. Структура данных Map по отношению к коллекциям в Java

В данной версии приложения IQKidBank ключом, т.е. уникальным идентификатором пользователя будет выступать именно логин, т.к., независимо от возможности совпадения паролей, имен и других характеристик, одинаковые логины программа не пропустит. Второй вариант – сделать ключом id пользователя, будет реализован в последующих версиях приложения.

#### 2.4.5 Методы класса BankAccount

У класса BankAccount будет только два метода, каждый из которых будет отвечать за отображение того или иного запроса на взаимодействие с пользователем.

##### **2.4.4.1 Метод showMenu(), строковые данные и пул строк.**

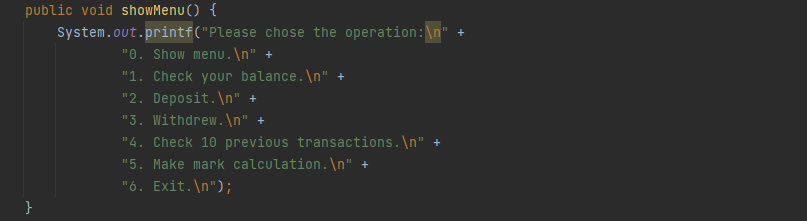
Метод showMenu() отвечает за отображение меню операций, доступных зарегистрированному пользователю приложения.

Рисунок 26. Метод ShowMenu() приложения IQKidBank

Как видно из рисунка 26, метод состоит из вывода строковых значений.

В Java строки представлены классом String, который представляет последовательность символов Unicode. Строковые значения могут быть сохранены в различных форматах и структурах данных для оптимальной работы с ними.

1. **Пул строк (String Pool):** В Java существует механизм для управления объектами типа String. Он представляет собой специальную область в памяти, где хранятся все уникальные строки, созданные в приложении.

При создании новой строки Java автоматически проверяет наличие уже созданной строки с таким же содержимым в пуле строк. Если она там уже есть, то возвращается ссылка на существующий объект String, а не создается новый[25]. Это позволяет экономить память и улучшает производительность.

2. **StringBuilder и StringBuffer:** Классы StringBuilder и StringBuffer используются для изменения содержимого строки. Они позволяют выполнять различные операции над строками, такие как добавление, удаление, замена символов. Основное отличие между ними в том, что StringBuilder потокобезопасен, в отличие от StringBuffer.

3. **Класс String:** Класс String представляет неизменяемую строку в Java. Это означает, что после создания объекта String его значение нельзя изменить. Вместо этого для аннулирования строки создается новый объект. Неизменяемость строк обеспечивает их безопасность и защиту от неожиданных изменений.

4. **Классы** **Pattern и Matcher**: Классы Pattern и Matcher предоставляют возможность работы с регулярными выражениями в Java. С их помощью можно искать и обрабатывать строки в соответствии с заданным шаблоном.

В целом, строки в Java могут быть хранены в различных структурах данных и форматах, в зависимости от требуемых операций и характеристик задачи. Класс String и его методы обеспечивают удобство работы с текстовыми данными, а оптимизация пула строк помогает улучшить производительность и экономию памяти.

##### **2.4.4.2 Метод enterByRegistration() проекта IQKidBank, конструкции while(true) и swich case**

Последний метод тестовой версии проекта IQKidBank. Метод, относящийся к уровню View, представления, отвечающий за «общение» приложения с пользователем.

Из заслуживающих внимания конструкций здесь можно назвать цикл while(true) и конструкцию switch … case.

##### **2.4.4.2.1 «Вечный» цикл while(true)**

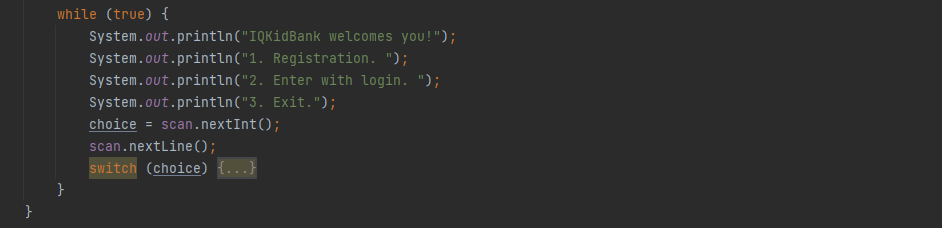
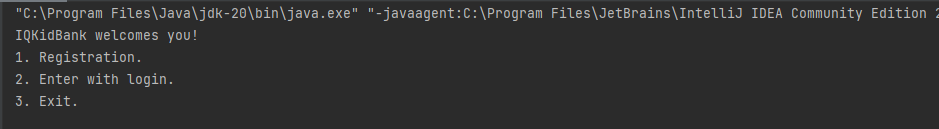
Конструкция while(true) представляет собой разновидность while – do и обеспечивает бесконечный цикл, который будет выполняться до тех пор, пока условие while будет истинным (true). Эта конструкция используется, когда необходимо продолжать выполнение определенного блока кода без остановки до тех пор, пока какое-то условие не будет выполнено. 

Рисунок 27. Конструкция while(true) метода enterByRegistration() проекта IQKidBank

Частота использования while(true) зависит от конкретной задачи и логики программы. Эта конструкция легка в реализации для новичка в программировании, однако обычно она не рекомендуется, так как может привести к зависанию программы из-за бесконечного цикла.

С точки зрения принципов ООП, использование while(true) может не соответствовать принципу единственной ответственности (SRP) и принципу открытости/закрытости (OCP), так как может привести к сложности в поддержке и расширении кода.

Условия выхода из цикла while(true) могут быть установлены внутри самого цикла, например, с помощью оператора break при выполнении определенного условия. Важно не забыть учитывать условия выхода из бесконечного цикла, чтобы избежать зависания программы.

В случае проекта IQKidBank, конструкция while(true) обеспечивает вывод основного меню, отвечающего за регистрацию пользователя, вход зарегистрированного пользователя и выход из приложения.

Рисунок 28. Основное меню, обеспеченное циклом while(true)

###### **2.4.4.2.2 Оператор выбора switch… case**

Оператор switch в Java проверяет переменную на равенство в отношении списка значений. Каждое значение называется case, и переменная проверяется для каждого case.

В Java оператор switch может иметь дополнительный default case, который должен находиться в конце switch. Default case может быть использован для выполнения задачи, когда ни один из case не является правильным.

Следующие правила применяются для оператора switch:

* Переменные, которые используются в операторе switch, могут быть только целые числа, конвертированные в целые числа (byte, short, char), строки и перечисления.
* Можно иметь любое количество операторов case в рамках одного switch. За каждым case следует сравниваемое значение, а затем идёт двоеточие.
* Значение case должно быть того же типа данных как и переменная в switch, а она должна быть константой или литералом.
* Когда переменная switch на равна оператору case, операторы следующие за case, будут выполняться до тех пор, пока не будет достигнут оператор break.
* При достижении оператора break, switch завершается, и поток управления переходит к следующей строке после оператора switch.
* Не каждый case должен содержать break. Если отсутствует break, поток управления попадет на следующий case и так далее, до тех пор, пока break не будет достигнут.
* В Java оператор switch может иметь дополнительный default case, который должен находиться в конце switch. Default case может быть использован для выполнения задачи, когда ни один из case является правильным. Break не требуется в default case.[25]

В проекте три основных кейса, отвечающие, соответственно, за регистрацию, вход в приложение и выход из приложения.

###### **2.4.4.3 Case 1. Регистрация**

Этот метод предназначен для регистрации нового клиента в банковской системе.

При выборе в базовом меню пункта 1, пользователя попросят внести данные о себе. В следующей версии приложения на данном этапе будут добавлены проверки корректного введения, например, кол-во цифр в телефонном номере, отсутствие нулевых значений для каждого из полей и др.

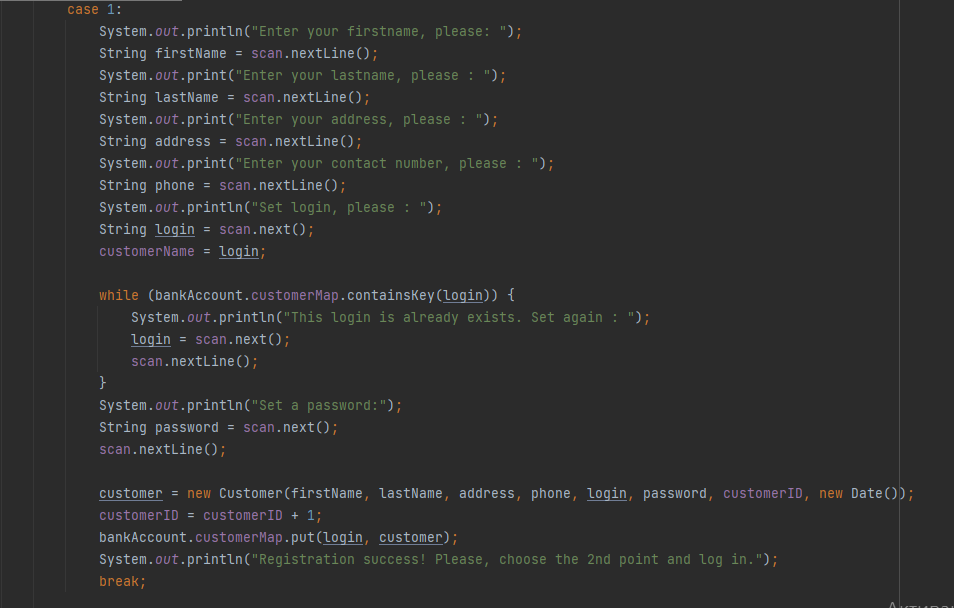
Ключевыми полями являются логин (login) и пароль(password), т.к. логин должен быть уникален, а без пароля пользователь не сможет зайти в систему.

Рисунок 29. Case 1: регистрация пользователя в приложения IQKidBank

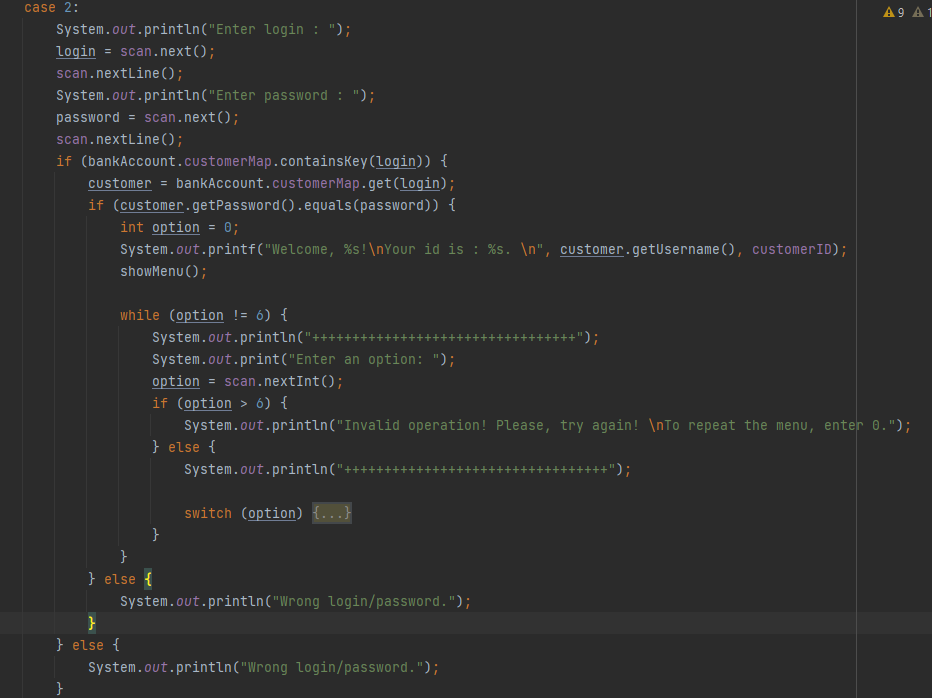
Итак, после ввода логина с клавиатуры, происходит проверка на наличие такого же логина в системе. Если логин существует, приложение запросит повторный ввод другого логина, если нет – у пользователя будет запрошен ввод пароля.

Далее создается новый объект Customer с информацией о клиенте банка. Объект добавляется в коллекцию customerMap объекта BankAccount.

По окончанию этого этапа пользователь видит в консоли сообщение об успешной регистрации в системе, и происходит выход из цикла.

Этот метод выполняет важную функцию в банковской системе - он позволяет новым клиентам зарегистрироваться и начать пользоваться услугами банка.

###### **2.4.4.4 Case 2. Вход в приложение, реализация банковских опций.**

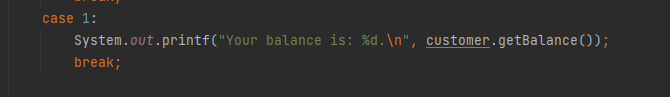
Метод представляет собой код-обработчик для входа в систему банковского аккаунта.

Метод начинается с запроса ввода логина и пароля от пользователя. Затем происходит проверка, существует ли введенный логин в Map, хранящей информацию о клиентах. Если логин существует, метод извлекает объект клиента с помощью метода .get() из Map. Далее он проверяет соответствие введенного пароля с паролем клиента. Если пароль совпадает, метод выводит приветственное сообщение, показывает имя пользователя и его id, вызывает метод showMenu() для отображения основного меню.

Рисунок 30 Case 2: Вход в приложение, реализация банковских опций IQKidBank

Так же внутри case 2 метода enterByRegistration() находится блок swich..case, отвечающий за выбор и реализаций функций, заявленных в методе showMenu() с проверкой на то, что введенное пользователем число находится в диапазоне от 0 до 6.

Рисунок 31. Меню выбора опций Case 0: вызов меню выбора опций

Case 0 вызывает метод showMenu() класса BankAccount.

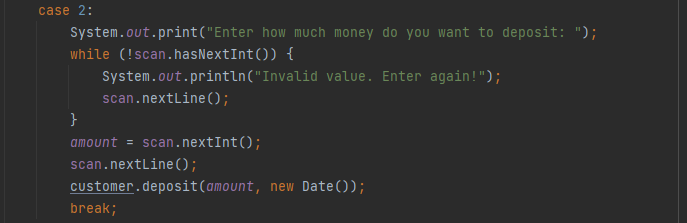
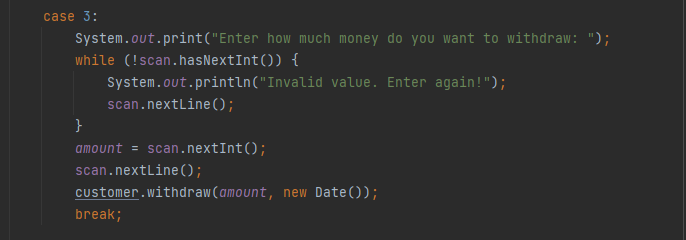
Case 1 создает текстовое сообщение, которое выводит на экран текущий баланс клиента. Для этого используется метод getBalance(), вызываемый у объекта customer класса Customer. После того, как сообщение о текущем балансе выведено на экран, выполнение кода заканчивается с помощью ключевого слова break;, что приводит к выходу из switch…case конструкции или цикла, в котором данный кейс был вызван.

Рисунок 33. Меню выбора опций Case 2: пополнение счета

Рисунок 32. Меню выбора опций Case 1: проверка баланса

В участке кода, за который отвечает case 2, происходит запрос у пользователя на ввод суммы денег, которую он хочет внести на счет. Пользователь должен ввести целочисленное значение, иначе будет выводиться сообщение об ошибке и пользователю будет предложено ввести данные снова.

После успешного ввода целочисленного значения, оно сохраняется в переменную amount. Затем вызывается метод deposit() объекта customer с передачей в него введенной суммы и объекта Date для того, чтобы зафиксировать дату и время операции.

Данный кейс завершается инструкцией break, что позволяет перейти к выполнению следующих инструкций после блока switch в данном методе или потоке выполнения программы.

В кейсе 3 находится метод withdraw(). Метод withdraw() принимает два параметра: сумму денег для снятия и текущую дату. Внутри метода происходит проверка наличия достаточной суммы на счете клиента для снятия указанной суммы. Если сумма достаточна, она вычитается из баланса счета, и добавляется запись о снятии денег на текущую дату. Если сумма недостаточна, выводится сообщение об ошибке. Метод withdraw() является частью класса Customer, который содержит информацию о клиенте, его счете и операциях с деньгами.

Рисунок 34. Меню выбора опций Case 3: списание средств со счета

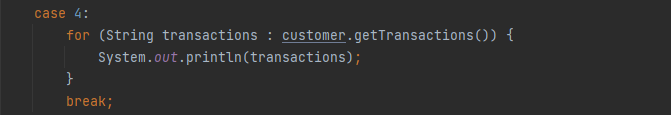
В кейсе 4 метод getTransactions() возвращает список последних 10 (по размеру ArrayList transactions в классе Customer) транзакций (операций) клиента. В данном случае цикл for используется для перебора всех транзакций, полученных из вызова метода .getTransactions() класса Customer, предоставляющего доступ к истории транзакций клиента. Каждая транзакция содержит информацию о дате операции, сумме и типе транзакции (например, пополнение счета или снятие денег).

Рисунок 35. Меню выбора опций Case 4: последние 10 транзакций

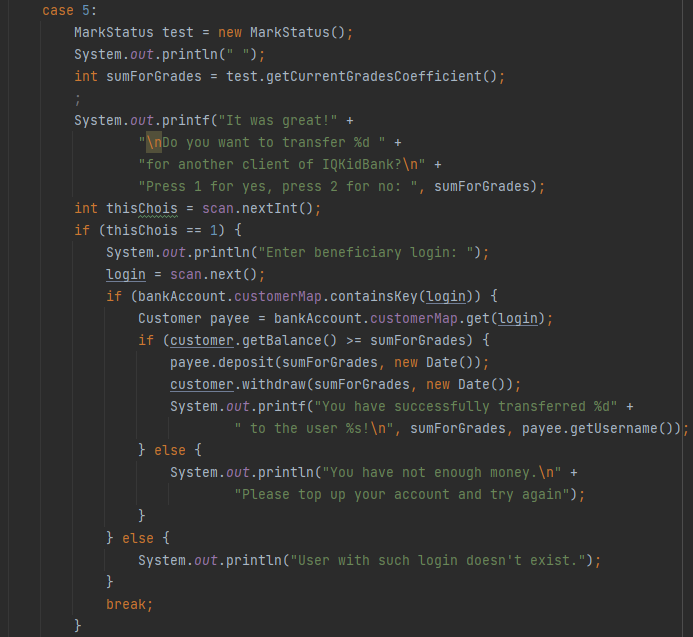
Пятый кейс представляет собой инициацию расчета денежного коэффициента за оценки и перевод средств между клиентами банка при условии наличия достаточного количества оных на счету отправителя.

Рисунок 36. Меню выбора опций Case 5: расчет денежного коэффициента оценок и перевод средств зарегистрированному клиенту IQKidBank

1. Создается новый экземпляр класса MarkStatus.

2. Высчитывается текущий коэффициент оценок с помощью метода getCurrentGradesCoefficient().

3. Выводится сообщение с предложением перевести сумму для другого клиента банка.

4. Пользователь вводит выбор (1 для согласия или 2 для отказа).

5. Если пользователь выбирает перевод средств (thisChois == 1), запрашивается логин получателя.

6. Проверяется, существует ли пользователь с таким логином в списке клиентов банка.

7. Если получатель найден, и у отправителя достаточно средств на счету - происходит перевод указанной суммы.

8. Если у отправителя недостаточно средств, выводится сообщение о нехватке средств на счету.

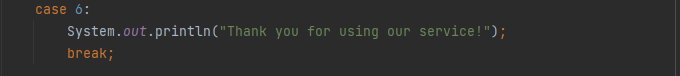
9. Завершается выполнение метода с помощью оператора break.

Рисунок 37. Меню выбора опций Case 6: выход из внутреннего меню IQKidBank

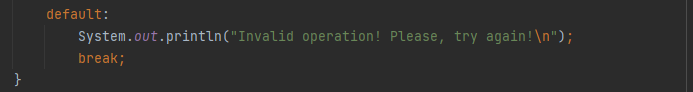
 Case 6 позволяет выйти из внутреннего меню в основное чтобы, например, зайти под другим логином и паролем или покинуть приложение.

Рисунок 38. Меню выбора опций оператор default приложения IQKidBank

Данный фрагмент кода представляет собой кейс по умолчанию (default case) в конструкции switch-case Java. Он выполняется, когда ни один из case-блоков не соответствует выражению в switch.

Если ни одна из заданных операций не соответствует введенной пользователем операции, то будет выведено сообщение "Invalid operation! Please, try again!".

Ключевое слово break указывает на завершение выполнения блока switch…case после выполнения default.

###### **2.4.4.5 Case 3. Выход из приложения**

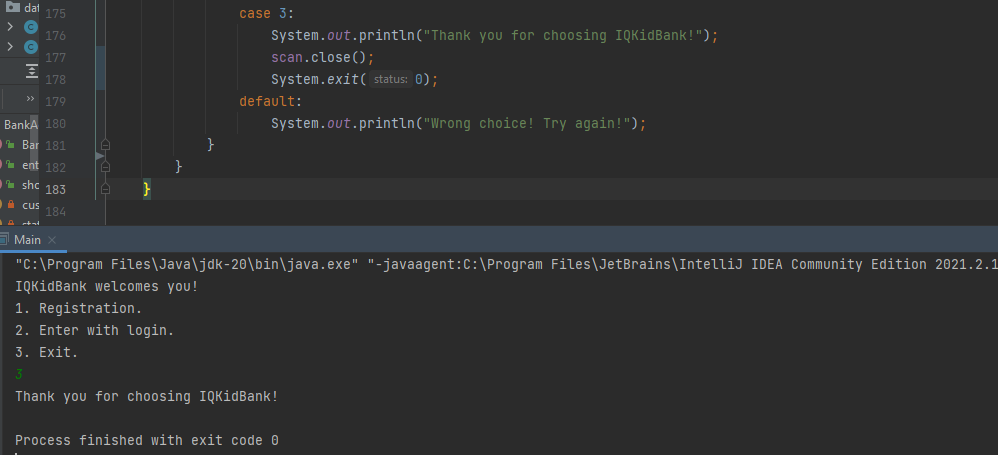


Рисунок 39. Case 3: выход из приложения IQKidBank

Ранее было упомянуто о том, что цикл while(true) необходимо использовать с осторожностью, чтобы исключить зацикливание программы. В данном случае точкой выхода из приложения будет ввод пользователем в основном меню значения «3». Как видно из кода, в этом меню любое значение, выходящие за рамки значений от 1 до 3 включительно вызовет сообщение об ошибке выбора и запрос на ввод корректного значения.

В случае выбора значения «3», приложение:

* поблагодарит пользователя за использование ресурса;
* закроет сканнер. Закрытие Scanner в Java важно для эффективного управления ресурсами. Когда Scanner используется для чтения данных из ввода (например, из консоли или файла), он создает поток ввода данных, который должен быть закрыт после завершения работы с ним. Закрытие Scanner также позволяет избежать утечек памяти и других проблем, связанных с ресурсами, так как закрытие освобождает все ресурсы, которые могут быть заняты Scanner. Команда scan.close() позволяет правильно закрыть поток ввода данных и избежать потенциальных проблем в программе;
* завершит выполнение программы с помощью команды System.exit(0): Команда System.exit(0) завершает выполнение программы с кодом завершения 0, что обычно означает успешное и безошибочное завершение программы.

После вызова System.exit(0) в коде нет необходимости использовать оператор break, позволяющий выйти из цикла while(true).

#### 2.4.5 Слабые места реализации класса BankAccount

Класс BankAccount в том виде, в котором он существует сейчас, и, в частности, метод enterByRegistration(), является слишком громоздким и неудобным для чтения, что необходимо учесть в разработке следующей версии.

Если сохранять консольный интерфейс, то, согласно принципу единой ответственности, необходимо выделить в отдельный метод внутреннее меню (варианты транзакций), и уже для этого меню создать свои методы для перевода просчитанной суммы денежного коэффициента клиенту банка, что будет обязательно реализовано в будущем.

Однако, учитывая тот факт, что в дальнейшем view-часть приложения будет модернизирована под оконный интерфейс, эта проблема будет решена.

### 2.5 Класс Main

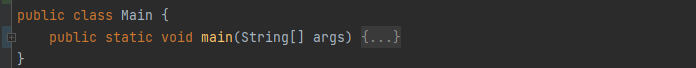
Класс Main в Java - это основной класс программы, который содержит метод main. Сигнатура класса Main всегда выглядит следующим образом:

Рисунок 40. Сигнатура класса Main

- public: ключевое слово, указывающее на доступ к классу из других пакетов.

- class: ключевое слово, обозначающее определение класса.

- Main: имя класса.

- public static void main(String[] args): это метод main, который является точкой входа программы. Он должен быть объявлен как public, static и возвращать void. Метод принимает один аргумент - массив строк args, который содержит аргументы командной строки.

Класс Main в Java является основной точкой запуска программы. То есть, JVM ищет метод main в классе Main и выполняет код, который находится внутри этого метода. Без него Java Virtual Machine могла запустить программу.

Отличие класса Main от других классов заключается в том, что класс Main содержит метод main, который является точкой входа в программу. В других классах может отсутствовать метод main или он может быть использован для других целей, не связанных с запуском программы. Класс Main обычно служит для организации и запуска остальных частей программы.

Рисунок 41. Класс Main с методом main() приложения IQKidBank

В дипломном приложении в методе main создается экземпляр first класса BankAccount, который инициирует вызов меню и реализует основной интерфейс пользователя.

Затем у созданного экземпляра класса вызывается метод enterByRegistration() для инициации регистрации пользователя и возможности взаимодействовать с основным функционалом программы.

## 3. Заключение

Результатом дипломной работы по курсу «Программист Java Цифровые профессии» стала тестовая консольная версия приложения IQKidBank. Это простое банковское приложение, обеспечивающее базовые функции отслеживания, пополнения баланса, снятие средств, расчет конвертации оценок студента, введенных вручную пользователем, в денежный эквивалент с возможностью перевода полученной суммы клиенту банка.

В рамках проекта реализован так же функционал регистрации пользователей и их входа в личный кабинет с соответствующими проверками существования этих пользователей по логину и паролю.

Реализованы следующие знания:

* разработка консольного приложения Java;
* написание рабочего кода с применением теоретических и практических знаний о синтаксисе Java;
* применение паттернов проектирования;
* знание принципов объектно-ориентированного программирования;
* умение описать свой код;
* применение знаний об архитектуре ПО при решении прикладных задач;
* применение алгоритмов для решения задач;
* работа с системой контроля версий Git;
* работа с потоками на примере класса Scanner;
* применение знаний JDK;
* и другие.

# Список литературы и ссылки

1. <https://intuit.ru/studies/courses/32/32/lecture/1004> («Нотация и семантика языка UML Автор: Александр Леоненков | Школа IT-менеджмента АНХ при Правительстве РФ)
2. <https://practicum.yandex.ru/blog/uml-diagrammy/> (Internet – статья «На каком языке рисуют схемы: что такое UML и почему его понимают во всём мире» «Яндекс практикум» Маргарита Нижельская автор курса «Системный аналитик», Лена Шпрингер редактор)
3. <https://ru.wikipedia.org/wiki/Графический_интерфейс_пользователя> (Материалы из свободной интернет-энциклопедии Википедия, статья «Графический интерфейс пользователя»)
4. [https://infostart.ru/pm/1451560/#:~:text=Диаграмма%20«Сущность-связь»%20(ERD%2C%20Entity-Relationship%20Diagram%2C,и%20информационных%20систем%20для%20бизнеса](https://infostart.ru/pm/1451560/#:~:text=Диаграмма%20) (Internet-статья «Краткий путеводитель по методологиям и нотациям описания и моделирования бизнес-процессов. Часть 4. Глава «Нотации ER-Диаграмм» Автор: Анастасия Штей Публикация № 1451560 Создание 01.06.21 13:00 Обновление 01.06.21 13:00)
5. https://aws.amazon.com/ru/what-is/api/ (Internet- статья «Что такое интерфейс прикладного программирования (API)? Глава «Что значит API?»)
6. <https://en.wikipedia.org/wiki/Write_once,_run_anywhere> (Материалы из свободной интернет-энциклопедии Википедия).
7. <https://ru.wikipedia.org/wiki/Интегрированная_среда_разработки> (Материалы из свободной интернет-энциклопедии Википедия, статья «Интегрированная среда разработки»)
8. <https://ru.wikipedia.org/wiki/Среда_окружения> (Материалы из свободной интернет-энциклопедии Википедия, статья «Среда окружения»)
9. <https://support.dnevnik.ru/94-160-216--pechatnaya-versiya-klassnogo-zhurnala-obsheobrazovatelnoj-organizacii/> (Официальный сайт поддержки Дневник.ру).
10. <https://practicum.yandex.ru/blog/chto-takoe-github-kak-on-rabotaet/#raznica-mezhdu-git-i-github> («Яндекс практикум» статья «Что такое GitHub и как он работает», раздел «В чём разница между Git и GitHub» авторы: Сергей Савельев наставник курса "Java-разработчик", старший разработчик в Яндексе Анна Коротеева, редактор 23 июня 2022г.)
11. «Изучаем Java» 2-е издание Кэти Сьерра и Берт Бейтс
12. Sams Teach Yourself Java in 21 Days (Covers Java 11/12), 8th Edition. Rogers Cadenhead 2020г.
13. <https://blog.skillfactory.ru/glossary/generics/> (Блог «SkillFactory Media» статья Generics от 26.03.2023 (автор неизвестен)
14. <https://ru.wikipedia.org/wiki/SOLID_(программирование)> (Материалы из свободной интернет-энциклопедии Википедия, статья «SOLID (программирование)»)
15. <https://ru.m.wikipedia.org/wiki/Константа> (Материалы из свободной интернет-энциклопедии Википедия, статья «Константа»)
16. <https://pershin.io/java-enum/#enum-java> (Материалы с сайта pershin.io, статья «Java enum – перечисления в Java», глава «Что такое Enum в Java», автор Денис Першин, 2023 г.)
17. <https://otus.ru/nest/post/666/> (Материалы с сайта otus.ru, статья «Операторы break и continue — что это? Оператор break в Java», автор Сергей, 10.04.19 в 11:22)
18. Java. Collection, author: Josh Bloch, Neal Gafter.
19. <https://ru.wikipedia.org/wiki/Ветвление_(программирование)> (Материалы из свободной интернет-энциклопедии Википедия, статья «Ветвление (программирование)»)
20. <https://habr.com/ru/articles/208442/> (Материалы с сайта habr.com, статья «Шпаргалка по SOLID-принципам с примерами на PHP», главы «Принцип единственной ответственности (Single responsibility)», «Принцип открытости/закрытости (Open-closed)», автор andrewnester, 8 янв 2014 в 13:31)
21. Marcotte, Ethan Responsive web design (англ.). A List Apart (25 мая 2010). Архивировано 24 мая 2013 года
22. <https://info-master.su/programming/kurs/types-of-programs.php> (Материалы с сайта info-master.su, статья «Виды программ», автор и дата публикации неизвестны)
23. <https://dzen.ru/a/ZMeUQ34YMBBUtSMq> (Материалы с сайта dzen.ru, статья «Java 827. Почему Map — это не Collection, в то время как List и Set являются Collection?», автор DEBAGanov, 31 июля 2023)
24. <https://dzen.ru/a/ZIAc8FxstWJkpady> (Материалы с сайта dzen.ru, статья «Java 75. Что такое пул строк и для чего он нужен?», автор DEBAGanov, 7 июня 2023)
25. <https://proglang.su/java> (Материалы с сайта prolang.su, «Самоучитель по программированию с нуля», глава «12.4. Java – Оператор switch..case», источник: Java Tutorial https://www.tutorialspoint.com/java/switch\_statement\_in\_java.htm )

1. Подробные инструкции по выгрузке оценок конкретного ученика и всего класса в Exel можно найти [здесь](https://udalenking.ru/kak-raspechatat-ocenki-iz-elektronnogo-dnevnika/). [↑](#footnote-ref-2)
2. Библиоте́ка (от англ. library) в программировании — сборник подпрограмм или объектов, используемых для разработки программного обеспечения (ПО). [↑](#footnote-ref-3)
3. Фреймворк — это программная платформа, которая упрощает разработку программного продукта, определяет структуру проекта и помогает удобно объединять в нём разные компоненты. [↑](#footnote-ref-4)
4. Объекты, которые также называются экземплярами, представляют собой автономные элементы программы со связанными функциями и данными. По большей части вы используете класс просто для создания экземпляров, а затем работаете с этими экземплярами.[12] [↑](#footnote-ref-5)
5. Класс Object в Java является базовым классом для всех других классов в языке. Это означает, что любой класс, который мы создаем в Java, автоматически является подклассом класса Object. [↑](#footnote-ref-6)
6. Хэш-код объекта - это числовое значение, которое используется для идентификации объекта в программировании. Он генерируется по определенному алгоритму, который уникален для каждого объекта. Хэш-код позволяет быстро и эффективно искать и сравнивать объекты в коллекциях данных, таких как хеш-таблицы или множества. [↑](#footnote-ref-7)
7. Связанность кода в программировании относится к степени, в которой различные части программы зависят друг от друга. Чем выше связанность кода, тем сложнее его изменять, тестировать и поддерживать. Уменьшение связанности кода улучшает читаемость и тестируемость кода, уменьшает риск возникновения ошибок, позволяет использовать уже написанный код повторно. [↑](#footnote-ref-8)
8. Дженерики (generics) в языке программирования Java — это сущности, которые могут хранить в себе данные только определенного типа. Например, список элементов, в котором могут быть одни числа. Но не только: дженерик — обобщенный термин для разных структур.[13] [↑](#footnote-ref-9)
9. SOLID (сокр. от англ. single responsibility, open–closed, Liskov substitution, interface segregation и dependency inversion) в программировании — мнемонический акроним, введённый Майклом Фэзерсом (Michael Feathers) для первых пяти принципов, названных Робертом Мартином[1][2] в начале 2000-х[3], которые означали 5 основных принципов объектно-ориентированных проектирования и программирования.<…> При создании программных систем использование принципов SOLID способствует созданию такой системы, которую будет легко поддерживать и расширять в течение долгого времени[3].

   Принципы SOLID — это руководства, которые также могут применяться во время работы над существующим программным обеспечением для его улучшения, например, для удаления «дурно пахнущего кода».[14] [↑](#footnote-ref-10)
10. Сигнатура метода - это уникальная характеристика метода, которая включает его имя, список параметров и типы параметров. Сигнатура метода используется для определения перегрузки методов в Java, то есть возможности создания методов с одинаковыми именами, но разной сигнатурой. Например, сигнатура метода withdraw() будет: withdraw(int, Date). [↑](#footnote-ref-11)
11. Константа (в программировании) - способ адресации данных, изменение которых рассматриваемой программой не предполагается или запрещается[15]. [↑](#footnote-ref-12)
12. Соглашение об именовании в Java - это набор правил и рекомендаций по выбору имен для классов, методов, переменных, констант и других элементов программного кода на языке Java. Соблюдение соглашения об именовании помогает сделать код более читаемым, понятным и однозначным для других разработчиков. [↑](#footnote-ref-13)
13. AbstractCollection - это абстрактный класс в Java, который представляет базовый класс для коллекций объектов. Он расширяет класс AbstractObject и реализует часть методов интерфейса Collection.

    AbstractCollection предоставляет базовую реализацию общих методов коллекций, таких как добавление, удаление, содержит и т. д. Он содержит абстрактные методы, которые должны быть переопределены в конкретных классах-наследниках, таких как size(), iterator(), add() и другие. Классы-наследники AbstractCollection, такие как ArrayList и HashSet, используются для реализации конкретных коллекций объектов в Java.[18] [↑](#footnote-ref-14)
14. В компьютерном программировании бесконечный цикл (или endless loop) - это последовательность инструкций, которая будет продолжаться бесконечно, если не произойдет внешнего вмешательства, такого как полное отключение питания или принудительное завершение задачи. [↑](#footnote-ref-15)
15. Ветвление в программировании — операция, применяющаяся в случаях, когда выполнение или невыполнение некоторого набора команд должно зависеть от выполнения или невыполнения некоторого условия. Ветвление — одна из трёх (наряду с последовательным исполнением команд и циклом) базовых конструкций структурного программирования[19]. [↑](#footnote-ref-16)
16. Принцип единственной ответственности гласит — «На каждый объект должна быть возложена одна единственная обязанность». Т.е. другими словами — конкретный класс должен решать конкретную задачу — ни больше, ни меньше.[21] [↑](#footnote-ref-17)
17. Данный принцип гласит — "программные сущности должны быть открыты для расширения, но закрыты для модификации". На более простых словах это можно описать так — все классы, функции и т.д. должны проектироваться так, чтобы для изменения их поведения, нам не нужно было изменять их исходный код.[20, 21] [↑](#footnote-ref-18)
18. См. стр. 20 [↑](#footnote-ref-19)
19. В Java существуют два типа для работы с целыми числами: int и Integer. Вот их основные отличия:

    int (примитивный тип данных):

    - int является примитивным типом данных в Java.

    - int хранит целочисленное значение без десятичной части.

    - Переменная типа int занимает фиксированный объем памяти (обычно 4 байта) и не может быть null.

    - Нельзя использовать методы и операции, доступные для объектов, например, сравнение через equals().

    - Преимущества int: более эффективно использует память и быстрее вычисляется.

    2. Integer (объектный тип данных):

    - Integer является классом-оболочкой (wrapper class) для типа int.

    - Integer позволяет использовать объектную обертку для примитивного типа int, включая возможность присвоить null.

    - Можно использовать методы и операции, доступные для объектов, например, сравнение через equals().

    - Иногда требуется Integer вместо int, например, при работе с коллекциями, которые не могут хранить примитивы. [↑](#footnote-ref-20)
20. Адаптивный веб-дизайн (англ. responsive web design) — дизайн веб-страниц, обеспечивающий правильное отображение сайта на различных устройствах, подключённых к интернету, и динамически подстраивающийся под заданные размеры окна браузера.

    Целью адаптивного веб-дизайна является универсальность отображения содержимого веб-сайта для различных устройств. Для того, чтобы веб-сайт был удобно просматриваемым с устройств форматов и с экранами различных разрешений, по технологии адаптивного веб-дизайна не нужно создавать отдельные версии веб-сайта для отдельных видов устройств. Один сайт может работать на смартфоне, планшете, ноутбуке и телевизоре с выходом в интернет, то есть на всем спектре устройств[22]. [↑](#footnote-ref-21)
21. Консольное приложение - это программа, которая работает с командной строкой. То есть это обычное окно, где пользователь может ввести какую-то команду и получить результат. Здесь нет никаких кнопочек и прочих прелестей Windows.

    Пример консольного приложения - это командный интерпретатор, который есть в любой операционной системе. В Windows 95/98/ME - это программа command.com (впрочем, он есть и в более поздних версиях Windows). В Windows 2000 и выше - это программа cmd.exe. [↑](#footnote-ref-22)
22. Оконное приложение - это привычная всем программа Windows. То есть это окошко с разными кнопочками и полями для ввода-вывода данных. На сегодняшний день это, пожалуй, самый распространённый вид программ. Именно оконные приложения создают большинство программистов. [↑](#footnote-ref-23)