Министерство образования Республики Беларусь
Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет инновационного непрерывного образования

Кафедра проектирования информационных компьютерных систем

Дисциплина "Объектно-ориентированное программирование"

К защите допустить:
Руководитель курсовой работы
старший преподаватель
кафедры
_____ А.В.Михалькевич
13.05.2024

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему

Организация работы деканата

БГУИР КР 1-40 05 01-10 № 165 ПЗ

Студент (подпись студента)

Курсовая работа
представлена на проверку
13.05.2024
_____
(подпись студента)

2024

Реферат

БГУИР КР 1-40 05 01-10 № 165 ПЗ, гр. 894351

, Организация работы деканата, Минск: БГУИР - 2024.

Пояснительная записка 141540 с., 0 рис., 0 табл.

Ключевые слова: ОБЪЕКТНО ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ, НАСЛЕДОВАНИЕ, ИНКАПСУЛЯЦИЯ, ПОЛИМОРФИЗМ, АБСТРАКЦИЯ, РЕЛЯЦИОННАЯ БАЗА ДАННЫХ, ШАБЛОНЫ ПРОЕКТИРОВАНИЯ.

Предмет Объектно-ориентированное программирование, А.В.Михалькевич

Разработка пользовательского интерфейса для организации работы деканата заинтересованными пользователями с возможностью внесения изменений в данные, поиска данных и формирования отчетов по успеваемости

-

Содержание

Введение

ВВЕДЕНИЕ Эффективность работы учебного заведения зависит от ряда факторов, и самым важным из них является роль информации и информационных ресурсов в учебных процессах. Одним из самых важных условий успешного функционирования любого деканата является эффективный обмен информацией. На сегодняшний день, несмотря на достаточно полную укомплектованность компьютерами большинства деканатов, процессы информационного обмена являются очень трудоемким делом. Сотрудникам деканатов ежедневно приходится выполнять большой объем рутинной работы по учету контингента студентов и обеспечению учебного процесса на факультете. При этом возникает необходимость в предоставлении информации в различных форматах. Необходимость внедрения информационной системы, автоматизирующей основные функции образовательного процесса, в настоящее время является важным и приоритетным направлением. Цель разрабатываемого программного продукта — частично автоматизировать работу деканата, для чего создать: общее хранилище данных об учебном процессе, включающее в себя данные о кафедрах, группах и студентах, а так же о сдаче студентами экзаменов, исключив необходимость частого поиска журналов и ведомостей с нужными данными; удобный интерфейс, позволяющий даже не обученному пользователю легко работать с этим хранилищем, вносить данные, редактировать или удалять; хороший функционал, который позволит какому-либо работнику деканата вместо долгого поиска нужной информации просто ввести нужные критерии поиска и вывести всю

необходимую информацию; автоматически составлять необходимые отчеты об успеваемости, если раньше преподавателю приходилось часами сидеть, чтобы, к примеру, составить ведомость за семестр по успеваемости, то разрабатываемый продукт должен позволять вывести всю необходимую информацию быстро и просто; гибкость - в вузах может быть установлено самое разное программное обеспечение, однако специалистов, которые смогли бы настроить его под очень требовательную программную систему может не быть. Программа должна быть максимально платформено независимой и работать на любой операционной системе с минимальными требованиями к сопутствующему программному обеспечению.

1 Описание проекта

1. ОПИСАНИЕ ПРОЕКТА

1.1 Описание серверной части

В системе будет храниться множество различной информации, необходимой для работы различных сотрудников — это и преподаватели, и заведующие кафедр, заместители деканов и многие другие. Это означает, что вариант, когда хранилище доступно только с одного конкретного компьютера не подходит — оно должно быть доступно из любого места учебного заведения. Так же это позволит координировать работу, например, преподаватели групп вносят данные об успеваемости своих студентов, а заведующий кафедрой может сразу же увидеть общую картину успеваемости студентов.

В связи с этим программа обязательно должна иметь серверную часть, содержащую в себе общее хранилище данных.

Для организации хранилища данных лучше всего использовать современный системы управления базами данных. СУБД— комплекс программ, позволяющих создать базу данных (БД) и манипулировать данными (вставлять, обновлять, удалять и выбирать). Система обеспечивает безопасность, надёжность хранения и целостность данных, а также предоставляет средства для администрирования БД.

Среди множества СУБД существующих в настоящее время для организации хранилища данных была выбрана СУБД MySQL — это реляционная система управления базами данных с открытым исходным кодом. Из преимуществ СУБД MySQL стоит отметить простоту использования, гибкость, низкую стоимость владения (относительно платных СУБД), а также масштабируемость и производительность. Гибкость СУБД MySQL обеспечивается поддержкой большого количества типов таблиц: пользователи могут выбрать как таблицы типа MyISAM, поддерживающие полнотекстовый поиск, так и таблицы InnoDB, поддерживающие транзакции на уровне отдельных записей. Есть и другие типы таблиц, разработанные сообществом.

Далее был определен перечень требований к базе данных. Она должна обеспечивать хранение следующей информации:

Деканат (Уникальный идентификатор, Название);
Кафедра (Уникальный идентификатор, Название, Номер деканата);
Группа (Уникальный идентификатор, Название, Номер кафедры);
Студенты (Уникальный идентификатор, Название, Номер группы);
Успеваемость (Уникальный идентификатор, Номер студента, Номер экзамена, Оценка);
Экзамен (Уникальный идентификатор, Название, Номер семестра);
Семестр (Уникальный идентификатор, Название, Год)

Далее на языке Sql был написан скрипт базы данных (текст скрипта приведен в Приложении А) и выполнен в среде Sql Server Managment Studio. После этого в среде была сгенерирована получившееся схема базы данных, которая представлена в Приложении Б на рисунке Б.2.

1.2 Описание клиентской части

Рядовому пользователю не имеющих специальных знаний будет очень неудобно и трудно работать с СУБД напрямую, что приведет к затрате больших усилий на обучение вначале и составлении запросов к базе данных позднее. Поэтому было разработано программное приложение для работы с базой данных. Оно имеет удобный интерфейс для работы с таблицами базы данных, опции внесения, редактирования и удаления данных, возможности поиска данных, а так же составления отчетов на основе данных из нескольких таблиц.

Для этого для каждой таблицы базы данных была создана отдельная страница в приложении, а так же форма для редактирования данных. Описание работы с таблицами базы данных приведено далее в пункте 1.2.1.

1.2.1 Описание работы с таблицами базы данных на примере таблицы «Успеваемость»

Таблицу «Успеваемость» можно считать основной таблицей системы. Она позволяет редактировать и просматривать оценки о сдаче каким либо студентом какого либо экзамена. Причем для удобства восприятия и представления полной картины выводится семестр и год в котором проходил экзамен, кафедра и деканат на которой учится студент.

Чтобы начать работу с таблицей «Успеваемость» необходимо в главном меню окна программы нажать пункт «Деканат», далее в выпадающем списке выбрать пункт «Успеваемость».

Рисунок 1.1 – Страница для работы с таблицей «Успеваемость»

Для добавления данных необходимо нажать кнопку «Добавить». Далее в форме редактирования ввести данные и нажать кнопку «Сохранить». Данные будут добавлены в таблицу

Рисунок 1.2 – Форма редактирования данных из таблицы «Успеваемость»

После нажатия кнопки «Сохранить» форма редактирования будет закрыта, а данные внесены в таблицу.

Рисунок 1.3 – Данные добавлены в таблицу

Для изменения данных необходимо выделить нужную строку таблицы и нажать кнопку «Изменить». Далее в форме редактирования изменить данные на необходимые и нажать кнопку «Сохранить». Данные будут изменены.

Рисунок 1.4 – Данные изменены

Для удаления данных необходимо выделить нужную строку таблицы и нажать кнопку «Удалить». Далее появится окно подтверждения удаления, где необходимо нажать «Да» для удаления записи или «Нет» для отмены удаления.

Рисунок 1.5 – Окно подтверждения удаления данных

1.2.2 Описание функционала поиска данных программы

Также, программа представляет возможность поиска данных о нужной студенческой группе или конкретном студенте.

Для поиска данных о группе необходимо выбрать пункт «Поиск» главного меню. В выпадающем списке выбрать пункт «Поиск группы». В результате будет открыта форма как на рисунке 1.6.

Рисунок 1.6 – Форма ввода критериев для поиска данных о группе

Поля, которые нужно учитывать при поиске нужно отметить галочкой «Учитывать». Если не выбрать ни одного поля будет выведено соответствующее сообщение.

Рисунок 1.7 – Сообщение об отсутствии критериев поиска

Для примера отметим поле «Кафедра» как ключевое для поля, и выберем в выпадающем списке кафедру «Кафедра физики». Далее необходимо нажать кнопку «Найти». Форма ввода закроется, а на главном окне будут в виде таблицы выведены найденные данные.

Рисунок 1.8 – Найденные данные при поиске группы

Поиск студента осуществляется аналогично, для поиска данных необходимо выбрать пункт «Поиск» главного меню. В выпадающем списке выбрать пункт «Поиск студента»

1.2.3 Описание работы с отчетами программы

Для облегчения анализа данных об успеваемости студентов в программе была предусмотрена возможность составления двух видов отчетов:

Отчет об успеваемости студента, за год или семестра;
Вывод среднего балла студента за год или семестр.

Для вывода отчета об успеваемости студента необходимо выбрать пункт «Отчет» главного меню. В выпадающем списке выбрать пункт «Отчет об успеваемости».

Рисунок 1.9 – Форма ввода данных для составления отчета об успеваемости

Аналогично поиску данных, при составлении отчетов можно так же выбирать поля, которые должны быть включены в критерии поиска информации. Это значит, что отчет можно составить нескольких видов:

успеваемость за какой либо год всех студентов или конкретного студента;
успеваемость за какой либо семестр всех студентов или конкретного студента;
успеваемость за какой либо год и семестр всех студентов или конкретного студента;

Для примера составим отчет об успеваемости всех студентов за 2020 год.

Для этого необходимо поставить галочку «Учитывать» напротив поля «Год» и ввести в это поле значение «2020», далее нажать кнопку «Составить».

Рисунок 1.10 – Отчет об успеваемости студентов за 2020 год

Для вывода среднего балла студента, необходимо выбрать пункт «Отчет» главного меню. В выпадающем списке выбрать пункт «Средний балл студента».

Рисунок 1.11 – Форма ввода данных для вывода среднего балла студента

Средний студента балл можно вывести трех видов:

за год;
за семестра;
за семестр определенного года;
за всё время.

Для примера выведем средний балл студента «Иванов И И» за 2020 год.

Для этого необходимо поставить галочку «Учитывать» напротив поля «Год» и ввести в это поле значение «2020», далее нажать кнопку «Составить».

Рисунок 1.12 – Средний балл студента «Иванов И И» за 2020 год

2 Обоснование выбора технологий

2. ОБОСНОВАНИЕ ВЫБОРА ТЕХНОЛОГИЙ

2.1 Описание основных средств разработки

Как было указано во введении, приложение должно быть максимально гибким и платформено независимым. Именно поэтому для реализации был выбран язык программирования **Java.**

Java — объектно-да. Изначально новый язык программирования назывался Oak (James Goslinориентированный язык программирования, разрабатываемый компанией Sun Microsystems с 1991 года и официально выпущенный 23 мая 1995 год) и java-машине (JVM) — программе, обрабатывающей байт-код и передающей инструкции оборудованию, как интерпретатор, но с тем отличи разрабатывался для бытовой электроники, но впоследствии был переименован в Java и стал использоваться для написания апплетов, приложений и серверного программного обеспечения.

Программы на Java могут быть транслированы в байт-код, выполняемый на виртуальнойем, что байт-код, в отличие от текста, обрабатывается значительно быстрее. Это значит, что программы **на Java могут работать на любой компьютерной архитектуре,** для которой существует реализация JVM.

Язык Java зародился как часть проекта создания передового программного обеспечения для различных бытовых приборов. Реализация проекта была начата на языке C++, но вскоре возник ряд проблем, наилучшим средством борьбы с которыми было изменение самого инструмента — языка программирования. Стало очевидным, что необходим платформено-независимый язык программирования, позволяющий создавать программы, которые не приходилось бы компилировать отдельно для каждой архитектуры и можно было бы использовать на различных процессорах под различными операционными системами.

Язык Java потребовался для создания интерактивных продуктов для сети Internet. Фактически, большинство архитектурных решений, принятых при создании Java, было продиктовано желанием предоставить синтаксис, сходный с C и C++. В Java используются практически идентичные соглашения для объявления переменных, передачи параметров, операторов и для управления потоком выполнением кода. В Java добавлены все хорошие черты C++.

Три ключевых элемента объединились в технологии языка Java:

Java предоставляет для широкого использования свои апплеты (applets) — небольшие, надежные, динамичные, не зависящие от платформы активные сетевые приложения, встраиваемые в страницы Web. Апплеты Java могут настраиваться и распространяться потребителям с такой же легкостью, как любые документы HTML;

Java высвобождает мощь объектно-ориентированной разработки приложений, сочетая простой и знакомый синтаксис с надежной и удобной в работе средой разработки. Это позволяет широкому кругу программистов быстро создавать новые программы и новые апплеты;

Java предоставляет программисту богатый набор классов объектов для ясного абстрагирования многих системных функций, используемых при работе с окнами, сетью и для ввода-вывода. Ключевая черта этих классов заключается в том, что они обеспечивают создание независимых от используемой платформы абстракций для широкого спектра

системных интерфейсов.

2.2 Реализация объектно-ориентированных технологий программирования в современных программно-математических средах

Далее будет рассмотрен объектный подход на примере выбранного для разработки программы языка программирования Java.

Концепции ООП являются основополагающими элементами и составляют основу языка программирования Java. В рамках данного подхода выделяют следующие термины: **абстракция**, **инкапсуляция**, **наследование** и **полиморфизм**. Понимание данных принципов служит ключом к построению целостной картины того, как работают программы, написанные на Java. По большому счету, объектно-ориентированный подход позволяет нам описывать классы, определять методы и переменные таким образом, чтобы затем использовать их вновь, частично либо полностью, без нарушения безопасности.

Принципы ООП в Java:

Абстракция. Абстракция означает использование простых вещей для описания чего-то сложного. Например, мы все знаем как пользоваться телевизором, но в тоже время нам не нужно обладать знаниями о том, как он работает чтобы смотреть его. В Java под абстракцией подразумеваются такие вещи, как **объекты**, **классы** и **переменные**, которые в свою очередь лежат в основе более сложного кода. Использование данного принципа позволяет избежать сложности при разработке ПО.

Инкапсуляция. Под инкапсуляцией подразумевается сокрытие полей внутри объекта с целью защиты данных от внешнего, бесконтрольного изменения со стороны других объектов. Доступ к данным (полям) предоставляется посредством публичных методов (геттеров/сеттеров). Это защитный барьер позволяет хранить информацию в безопасности внутри объекта.

Наследование. Это особая функциональность в объектно-ориентированных языках программирования, которая позволяет описывать новые классы на основе уже существующих. При этом поля и методы класса-предка становятся доступны и классам-наследникам. Данная фича делает классы более чистыми и понятным за счет устранения дублирования программного кода.

Полиморфизм. Данный принцип позволяет программистам использовать одни и те же термины для описания различного поведения, зависящего от контекста. Одной из форм полиморфизма в Java является переопределение метода, когда различные формы поведения определяются объектом из которого данный метод был вызван. Другой формой полиморфизма является перегрузка метода, когда его поведение определяется набором передаваемых в метод аргументов.

Принцип работы абстракции. Основная цель использования данной концепции — это уменьшение сложности компонентов программы за счет скрытия от программиста, использующего эти компоненты, ненужных ему подробностей. Это позволяет реализовать более сложную логику поверх предоставленной абстракции, не вдаваясь в подробности ее реализации.

Приготовление кофе с помощью кофемашины является хорошим примером абстракции. Все, что нам надо знать, что бы ей пользоваться: как налить воды, засыпать кофейные зерна, включить и выбрать вид кофе, который хотим получить. А, как машина будет варить кофе — нам знать не нужно.
В данном примере кофемашина представляет собой абстракцию, которая от нас скрывает все подробности варки кофе. Нам лишь остается просто взаимодействовать с простым интерфейсом, который не требует от нас каких-либо знаний о внутренней реализации машины.
Этот же подход можно использовать и в объектно-ориентированных языках программирования, таких как Java.

Принцип работы инкапсуляции. Инкапсуляция позволяет нам пользоваться возможностями класса без создания угрозы безопасности данных за счет ограничения прямого доступа к его полям. Также она позволяет изменять код классов не создавая проблем их пользователям (другим классам). В Java данный принцип достигается за счет использования ключевого слова **private**.

Принцип работы наследования. Наследование — еще одна важная концепция ООП, которая позволяет сэкономить время на написании кода. Возможности наследования раскрываются в том, что новому классу передаются свойства и методы уже описанного ранее класса. Класс, который наследуется называется дочерним (или подклассом). Класс, от которого наследуется новый класс — называется родительским, предком и т.д. В языке программирования Java используется ключевое слово **extends** для того, чтобы указать на класс-предок.

Принцип работы полиморфизма. Полиморфизм предоставляет возможность единообразно обрабатывать объекты с различной реализацией при условии наличия у них общего интерфейса или класса. По-простому: способность вызывать нужные методы у объектов, имеющие разные типы (но находящиеся в одной иерархии). При этом происходит автоматический выбор нужного метода в зависимости от типа объекта.
Рассмотрим примеры полиморфизма в Java: переопределение и перегрузка методов.
В случае с переопределением метода, дочерний класс, используя концепцию полиморфизма, может изменить (переопределить) поведение метода родительского класса. Это позволяет программисту по разному использовать один и тот же метод, определяя поведение из контекста вызова (вызывается метод из класса предка или класса наследника).
В случае же с перегрузкой, метод может проявлять различное поведение в зависимости от того, какие аргументы он принимает. В данном случае контекст вызова определяется набором параметров метода.

3 Инструментарий

3. ИНСТРУМЕНТАРИЙ

3.1 Описание выбранной среды разработки

В качестве среды разработки под Java была выбрана **IntelliJ IDEA.**

IntelliJ – одна из самых мощных и популярных интегрированных сред разработки (IDE) для Java. Он разработан и поддерживается JetBrains и доступен как окончательная версия для сообщества. Эта многофункциональная IDE обеспечивает быструю разработку и помогает улучшить качество кода.

IDE расшифровывается как интегрированная среда разработки. Это комбинация нескольких инструментов, которые делают процесс разработки программного обеспечения более простым, надежным и менее подверженным ошибкам. Он имеет следующие преимущества по сравнению с текстовым редактором:

интеграция с полезными инструментами, такими как компилятор, отладчик, система контроля версий, инструменты сборки, различные платформы, профилировщики приложений и так далее;

поддерживает функции навигации по коду, автозавершения кода, рефакторинга и генерации кода, что ускоряет процесс разработки;

поддерживает модульное тестирование, интеграционное тестирование и покрытие кода с помощью плагинов;

предоставляет богатый набор плагинов для дальнейшего расширения функциональности IDE .

IntelliJ IDEA обладает некоторыми наиболее эффективными функциями завершения кода Java. Его алгоритм прогнозирования может точно предполагать, что программист пытается набрать, и завершает его для него, даже если он не знает точного имени определенного класса, члена или любого другого ресурса.

IntelliJ IDEA действительно понимает и глубоко понимает код, а также контекст программиста, что делает его таким уникальным среди других Java IDE.

интеллектуальное завершение кода – поддерживает контекстное завершение кода. Он дает список наиболее значимых символов, применимых в текущем контексте;

цепное завершение кода – это расширенная функция завершения кода, которая перечисляет соответствующие символы, доступные через методы или методы получения в текущем контексте;

завершение статического члена – позволяет использовать статические методы или константы и автоматически добавляет необходимые операторы импорта, чтобы избежать ошибки компиляции;

обнаружение дубликатов – он обнаруживает фрагменты дублированного кода на лету и дает уведомление / предложение об этом пользователю;

инспекции и быстрые исправления. Всякий раз, когда IntelliJ обнаруживает, что программист собирается совершить ошибку, в одной строке появляется небольшое ламповое уведомление. Нажав на нее, можно увидиеть список предложений.

Чтобы помочь разработчикам организовать рабочий процесс, IntelliJ IDEA предлагает им

удивительный набор инструментов, который включает в себя декомпилятор, поддержку Docker, средство просмотра байт-кода, FTP и многие другие инструменты:

контроль версий – IntelliJ поддерживает большинство популярных систем контроля версий, таких как Git, Subversion, Mercurial, CVS, Perforce и TFS;

инструменты сборки – IntelliJ поддерживает Java и другие инструменты сборки, такие как Maven, Gradle, Ant, Gant, SBT, NPM, Webpack, Grunt и Gulp;

тестовый прогон и покрытие кода – IntelliJ IDEA позволяет с легкостью выполнять модульное тестирование. Среда IDE включает в себя тестовые прогоны и инструменты покрытия для основных сред тестирования, включая JUnit, TestNG, Spock, Cucumber, ScalaTest, spec2 и Karma;

декомпилятор – IntelliJ поставляется со встроенным декомпилятором для классов Java. Если необходимо заглянуть внутрь библиотеки, для которой нет исходного кода, вможно сделать это без использования сторонних плагинов;

терминал – IntelliJ предоставляет встроенный терминал. В зависимости от платформы вможно работать с командной строкой, например PowerShell или Bash;

инструменты базы данных – IntelliJ предоставляет инструменты базы данных, которые позволяют подключаться к действующим базам данных, выполнять запросы, просматривать и обновлять данные и даже управлять своими схемами в визуальном интерфейсе из самой IDE;

сервер приложений – IntelliJ поддерживает основные серверы приложений: Tomcat, JBoss, WebSphere, WebLogic, Glassfish и многие другие. Можно развернуть свои артефакты на серверах приложений и отладить развернутые приложения в самой IDE;

поддержка Docker. Через отдельный плагин IntelliJ предоставляет специальное окно инструментов, которое позволяет подключаться к локально работающим компьютерам Docker.

3.2 Описание системы контроля версий

Для более удобной и эффективной разработки и сопровождения программного продукта будет хорошим решением использовать систему контроля версий. Это позволит при необходимости откатить программу на более ранний вариант, например если будет внесена правка, повлекшая за собой критические ошибки в работе программы, или если решение той или иной задачи оказалось более подходящим в какой либо момент. Так же система контроля версий в случае серьезного расширения проекта сделает легкодоступным и удобным командную разработку.

В качестве системы контроля версий был выбран Git.

Git — это набор консольных утилит, которые отслеживают и фиксируют изменения в файлах (чаще всего речь идет об исходном коде программ, но можно использовать его для любых файлов). С его помощью можно откатиться на более старую версию проекта, сравнивать, анализировать, сливать изменения и многое другое. Этот процесс называется контролем версий. Существуют различные системы для контроля версий: SVN, Mercurial, Perforce, CVS, Bitkeeper и другие.

Git является распределенным, то есть не зависит от одного центрального сервера, на

котором хранятся файлы. Вместо этого он работает полностью локально, сохраняя данные в папках на жестком диске, которые называются репозиторием. Тем не менее, можно хранить копию репозитория онлайн, это сильно облегчает работу над одним проектом для нескольких людей. Для этого используются сайты вроде github и bitbucket.

В целях создания удаленного доступа к git-репозиторию проекта, репозиторий был создан на сайте github. **GitHub** — крупнейший веб-сервис для хостинга IT-проектов и их совместной разработки. Веб-сервис основан на системе контроля версий Git и разработан на Ruby on Rails и Erlang компанией GitHub, Inc (ранее Logical Awesome). Сервис бесплатен для проектов с открытым исходным кодом и (с 2019 года) небольших частных проектов, предоставляя им все возможности (включая SSL), а для крупных корпоративных проектов предлагаются различные платные тарифные планы.

Ссылка на репозиторий проекта - https://github.com/mihkek/DecanWork/

4 Архитектурный шаблон проектирования MVC

4. АРХИТЕКТУРНЫЙ ШАБЛОН ПРОЕКТИРОВАНИЯ MVC

4.1 Общее описание шаблона проектирования MVC

Model-View-Controller (**MVC**, «Модель-Представление-Контроллер», «Модель-Вид-Контроллер») — схема разделения данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: модель, представление и контроллер — таким образом, что модификация каждого компонента может осуществляться независимо:

Модель (*Model*) предоставляет данные и реагирует на команды контроллера, изменяя своё состояние;

Представление (*View*) отвечает за отображение данных модели пользователю, реагируя на изменения модели;

Контроллер (*Controller*) интерпретирует действия пользователя, оповещая модель о необходимости изменений.

Основная цель применения этой концепции состоит в отделении бизнес-логики (*модели*) от её визуализации (*представления*, *вида*). За счёт такого разделения повышается возможность повторного использования кода. Наиболее полезно применение данной концепции в тех случаях, когда пользователь должен видеть те же самые данные одновременно в различных контекстах и/или с различных точек зрения. В частности, выполняются следующие задачи:

1. К одной *модели* можно присоединить несколько *видов*, при этом не затрагивая реализацию *модели*. Например, некоторые данные могут быть одновременно представлены в виде электронной таблицы, гистограммы и круговой диаграммы;
2. Не затрагивая реализацию *видов*, можно изменить реакции на действия пользователя (нажатие мышью на кнопке, ввод данных) — для этого достаточно использовать другой *контроллер*;
3. Ряд разработчиков специализируется только в одной из областей: либо разрабатывают графический интерфейс, либо разрабатывают бизнес-логику. Поэтому возможно добиться того, что программисты, занимающиеся разработкой бизнес-логики (*модели*), вообще не будут осведомлены о том, какое *представление* будет использоваться.

Концепция MVC позволяет разделить модель, представление и контроллер на три отдельных компонента:

Рисунок 4.1 – Общая схема взаимодействия компонентов шаблона MVC

Модель предоставляет данные и методы работы с ними: запросы в базу данных, проверка на корректность. Модель не зависит от представления (не знает как данные визуализировать) и контроллера (не имеет точек взаимодействия с пользователем) , просто предоставляя доступ к данным и управлению ими.

Модель строится таким образом, чтобы отвечать на запросы, изменяя своё состояние, при этом может быть встроено уведомление «наблюдателей».

Модель, за счёт независимости от визуального представления, может иметь несколько различных представлений для одной «модели».

Представление отвечает за получение необходимых данных из модели и отправляет их пользователю. Представление не обрабатывает введённые данные пользователя.

Контроллер обеспечивает «связь» между пользователем и системой. Контролирует и направляет данные от пользователя к системе и наоборот. Использует модель и представление для реализации необходимого действия.

Функциональные возможности и расхождения

Поскольку MVC не имеет строгой реализации, то реализован он может быть по-разному. Нет общепринятого определения, где должна располагаться бизнес-логика. Она может находиться как в контроллере, так и в модели. В последнем случае, модель будет содержать все бизнес-объекты со всеми данными и функциями.

Некоторые фреймворки жестко задают где должна располагаться бизнес-логика, другие не имеют таких правил.

Также не указано, где должна находиться проверка введённых пользователем данных. Простая валидация может встречаться даже в представлении, но чаще они встречаются в контроллере или модели.

Интернационализация и форматирование данных также не имеет четких указаний по расположению.

4.2 Применение шаблона проектирования MVC в программе

В программе данный шаблон реализован как набор классов для работы с таблицами базы данных.

Компонент «View» шаблона проектирования MVC в программе представляет встроенный Java-класс Jtable.

Jtable - класс библиотеки Java Swing, предназначенной для создания графического интерфейса. Он является одним из основных компонентов, обеспечивающих работу шаблона MVC в программе. Jtable создает таблицу исходя из того, какой объект находится в поле Model этого класса. Это поле может быть объектом любого класса, наследующего класс TableModel.

Компонент «Model» шаблона проектирования MVC в программе представляет встроенный Java-класс TableModel.

TableModel – базовый класс для создания модели для представления. Класс, наследующий класс TableModel должен обязательно переопределять следующие методы:

1. void add(Object value) — метод добавления данных в модель;
2. int getColumnCount() – метод необходимый классу представления для получения количества столбцов в модели;
3. boolean isCellEditable – позволяет получить, является ли столбец с заданным номером редактируемым. Если метод возвращает true – то в графическом представлении таблицы возможно напрямую изменять данные строк Jtable в этом столбце, если нет то прямое редактирование недоступно. В настоящей программе прямое редактирование запрещено для всех столбцов, для этого созданы специальные формы;
4. getRowCount() – позволяет получить количество строк в модели;
5. getColumnClass(int columnIndex) – позволяет получить класс столбца модели;
6. set<TableModelListener> listeners – «прослушки», именно эти объекты уведомляют модель из представления о том, что какие-то данные изменились, таким образом связывая модель и представление;
7. getValueAt(int rowIndex, int columnIndex) – позволяет представлению получить данные по заданным строке и столбцу.

Все эти методы используются классом представления (Jtable) для того, чтобы получать данные из модели, и уже исходя из них рисовать графическое представление таблицы.

Отдельного и ярко выраженного класса контроллера не создается, как правило, в классе модели определяются все необходимые методы для манипуляции с данными (если речь идёт о работе с таблицами), и помимо этого в классе Jtable имеется множество встроенных и невидимых программисту функций, таких как отправка сигналов об изменении данных в модель, или сбор информации из модели и графическая отрисовка таблицы.

В целом, класс модели может быть абсолютно любым, а как будут данные добавляться, выводится, удалятся и изменяться — дело программиста.

Далее была создана общая диаграмма классов для хорошего видения всей картины. Рисунок 4.2 – Диаграмма классов

Основными и базовыми классами для моделей таблиц являются классы BaseDbTable и BaseDbRow. BaseDbRow — абстрактный класс, хранящий набор базовых методов, которые должна реализовывать строка модели таблицы. В классе BaseDbTable определены все основные методы и поля для работы с таблицами, переопределены методы класса TableModel. Сам класс является абстрактным, т.е создать объект этого класса нельзя, он представляет собой базовый набор общих функций для таблиц данных. Некоторые из них являются полностью универсальными, и будут одинаковые при работе с любой таблицей, некоторые нужно будет переопределить. Также он содержит коллекцию типа List<BaseDbRow> records, в ней будут храниться все строки модели, с этой коллекцией будут взаимодействовать методы редактирования и получения данных.

Классы DecanModel, EkzamTable, EkzstTable, GroupModel, KafedraTable, SemestrTable, StudentTable являются наследниками класса BaseDbTable. И уже содержат весь необходимый функционал для взаимодействия с представлением, но помимо этого, в нем определено количество столбцов и конкретный тип данных, хранящийся в коллекции records. Стоит отметить, создание базового абстрактного класса и создание нескольких конкретных его реализаций — пример шаблона проектирования «Абстрактная фабрика», описание применения

этого шаблона в программе будет приведено далее в 5 главе.

В интерфейсе одновременно существуют несколько страниц представлений. Программа представляет собой 1 (одно) основное окно с главным меню вверху, и кнопками «Добавить», «Изменить», «Удалить» внизу. Остальное место окна занимает объект графической библиотеки Swing – Jframe. Именно здесь будет находиться страница с данными — currentPage класса framePage.

Класс framePage нужен для более удобной работы с таблицей и объединяет в себе таблицу, полосу прокрутки и панель в 1 (одном) объекте.

В главном классе интерфейса — modFrame – поле currentPage содержит активный объект framePage, также содержится ряд дополнительных объектов этого класса – страницы приложения, по командам пользователя активная страница может сменяться.

За счет использования шаблона MVC, Jtable может содержать в себе любую модель, никакой логики кроме графической отрисовки таблицы данный класс не выполняет. Все необходимые операции, в том числе обеспечение представления информацией о том, что необходимо выводить в интерфейс, выполняет класс модели — один из наследников класса BaseDbTable.

Далее было разработано несколько диаграмм последовательности для лучшего представления работы шаблона MVC в программе.

Диаграмма последовательности при работе MVC шаблона на просмотр данных модели на примере таблицы «Деканат»:

Диаграмма последовательности при смене модели для Jtable:

Диаграмма последовательности при добавлении данных на примере таблицы «Деканат»:

5 Шаблон проектирования практических задач

5. ШАБЛЛОН ПРОЕКТИРОВАНИЯ ПРАКТИЧЕСКИХ ЗАДАЧ

Помимо шаблона проектирования MVC в программе используется ещё один не менее популярный шаблон - «Абстрактная фабрика».

Паттерн (шаблон) проектирования — это продуманный способ построения исходного кода программы для решения часто возникающих в повседневном программировании проблем проектирования. Иными словами, это уже придуманное решение, для типичной задачи. При этом, паттерн не готовое решение, а просто алгоритм действий, который должен привести к желаемому результату. Один из наиболее часто используемых порождающих паттернов — Абстрактная фабрика (Abstract Factory).

Существует три вида паттернов проектирования:

Порождающие паттерны позволяют выполнять инициализацию объектов наиболее удобным и оптимальным способом;

Структурные паттерны описывают взаимоотношения между различными классами или объектами, позволяя им совместно реализовывать поставленную задачу;

Поведенческие паттерны позволяют грамотно организовать связь между сущностями для оптимизации и упрощения их взаимодействия.

Абстрактная фабрика (Abstract Factory) – это порождающий паттерн, предоставляющий возможность создания семейства взаимосвязанных или родственных объектов, не специфицируя их классов. То есть, определяется интерфейс для создания взаимосвязанных объектов, без необходимости реализации конкретных классов.

Паттерн абстрактная фабрика позволяет создавать группы взаимосвязанных объектов. То есть, для начала мы должны определить группу свойств и методов, которые будут характерны для всех представителей группы. Например, возможность передвигаться у живых существ. Потом мы можем выделить конкретные фабрики для производства разных групп существ, например, фабрику по производству птиц, которые будут летать, фабрику для млекопитающих, которые будут бегать, и фабрику пресмыкающихся, которые будут ползать. Каждая из фабрик будет создавать экземпляры своего типа, но при этом все они будут иметь возможность передвигаться, но по своему.

Аналогично примеру с группами существ, в программе паттерн работает с классами таблиц базы данных. Базовый абстрактный класс — BaseDbTable, и в нем определяются все методы, характерные для класса модели таблицы данных – методы отправки данных в базу данных (writeData()), методы чтения (readData()), удаление (deleteRow()) и т.д. Общие признаки таблицы данных, такие как название таблицы в базе данных, с которой будет взаимодействовать какой-либо наследник класса (dbTableName()), количество столбцов (columnCount). Помимо этого, будут определены все заголовки методов, которые уже в каждой разновидности модели таблицы данных будут свои – методы получения значения какого-либо столбца записи по номеру строки и номеру столбца, или генерация данных для формы ввода, сколько и каких полей ввода нужно будет создавать.

Вторая абстрактная фабрика — семейство классов BaseDbRow. Аналогично фабрике моделей таблиц данных, данное семейство классов представляет конкретную строку в таблице данных. Содержатся общие для всех строк черты — количество столбцов (roleCount), а так же объявление методов - сборка данных строки для вставки в базу данных (buildForInsert()), сборка

данных строки для обновления строки в базе данных (buildForUpdate]()), метод получения значения поля, что является первичным ключём (getPrimaryKeyValue()) и ряд других методов.

Эти две фабрики взаимодействуют между собой. В классе BaseDbTable есть поле — records – это коллекция строк таблицы, она состоит из объектов BaseDbRow, это значит, что данную коллекцию каждый наследник класса BaseDbTable может заполнить нужными ему строками — наследниками класса BaseDbRow.

Часть диаграммы классов, на которой можно увидеть, как наглядно выглядит шаблон «Абстрактная фабрика» в иерархии классов программы представлена на рисунке 5.1. На диаграмме видно, что основное поведение наследников классов BaseDbTable и BaseDbRow определено в базовых классах. А так же, что каждому классу модели таблицы соответствует класс представляющий строку этой таблицы и они связаны композицией — то есть класс строки является атрибутом класса модели таблицы.

Рисунок 5.1 – Часть диаграммы классов

Использование данного шаблона дает множество преимуществ. Во первых — расширяемость, если будет нужно добавить возможность вывода какой-либо ещё таблицы базы данных, то нужно только создать класс-наследник класса BaseDbTable и соответствующую ему реализацию класса BaseDbRow и определить основные признаки таблицы в них.

Во вторых — удобство расширения функциональности классов и исправления ошибок. Поскольку поведение классов определенно в базовом, если вдруг потребуется, например, изменить базовый алгоритм сортировки записей в модели, то не нужно будет менять код во всех моделях — достаточно будет изменить метод sort () класса BaseDbTable, аналогичная ситуация с исправлением неполадок.

В третьих, использование шаблона серьезно расширяет возможности программирования. Например, в программе есть класс, реализующий форму ввода — InputForm. Данный класс предназначен для автоматической генерации формы ввода данных, исходя из того, какой объект был задан в конструкторе. Если бы не использовалась абстракция, то пришлось бы создавать для каждой таблицы отдельную форму ввода, и у каждой в качестве поля класса ставить объект, представляющий строку нужного вида. Однако, используя шаблон, появляется возможность создать одну форму ввода с полем абстрактного типа (поле result). А затем, при создании объекта формы генерировать её содержимое с помощью данных из конкретной реализации класса BaseDbRow.

6 Приложения

ПРИЛОЖЕНИЕ А
Текст sql-скрипта базы данных

```
-- phpMyAdmin SQL Dump
```

```
-- version 5.0.1
-- https://www.phpmyadmin.net/
--
-- Хост: 127.0.0.1
-- Время создания: Май 22 2020 г., 19:10
-- Версия сервера: 10.4.11-MariaDB
-- Версия PHP: 7.2.28

SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
SET AUTOCOMMIT = 0;
START TRANSACTION;
SET time_zone = "+00:00";

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8mb4 */;
--
-- База данных: `decan`
--
-- --------------------------------------------------------
--
-- Структура таблицы `decanat`
--
CREATE TABLE `decanat` (
`id` int(11) NOT NULL,
`name` varchar(100) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
--
-- Дамп данных таблицы `decanat`
--
INSERT INTO `decanat` (`id`, `name`) VALUES
(1, 'ФПИТЕ'),
(2, 'ФВТ'),
(8, 'Мед'),
(9, 'ФМТ');
-- --------------------------------------------------------
--
-- Структура таблицы `ekzam`
--
CREATE TABLE `ekzam` (
`id` int(11) NOT NULL,
`name` varchar(200) NOT NULL,
`year` int(20) DEFAULT NULL,
`idSem` int(11) NOT NULL
```

```
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
--
-- Дамп данных таблицы `ekzam`
--
INSERT INTO `ekzam` (`id`, `name`, `year`, `idSem`) VALUES
(1, 'Математика', 1991, 2),
(2, 'Информатика', 2020, 3),
(3, 'Русский', 2020, 2),
(4, 'Химия', 2020, 2),
(5, 'Физика', 2020, 3);
-- --------------------------------------------------------
--
-- Структура таблицы `ekzst`
--
CREATE TABLE `ekzst` (
`id` int(11) NOT NULL,
`idSt` int(11) NOT NULL,
`idEkz` int(11) NOT NULL,
`score` varchar(100) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
--
-- Дамп данных таблицы `ekzst`
--
INSERT INTO `ekzst` (`id`, `idSt`, `idEkz`, `score`) VALUES
(4, 1, 1, '4'),
(5, 2, 4, '5'),
(7, 1, 3, '3'),
(8, 5, 2, '5'),
(9, 2, 3, '4'),
(12, 4, 2, '2'),
(13, 2, 1, '2'),
(14, 1, 5, '2'),
(15, 3, 3, '4');
-- --------------------------------------------------------
--
-- Структура таблицы `groupa`
--
CREATE TABLE `groupa` (
`id` int(11) NOT NULL,
`name` varchar(100) NOT NULL,
`idKaf` int(11) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
--
-- Дамп данных таблицы `groupa`
--
```

```
INSERT INTO `groupa` (`id`, `name`, `idKaf`) VALUES
(1, '16во1', 6),
(2, '20во12', 3),
(4, '14вв2', 1),
(5, '17дд3', 7),
(6, '18ППВ', 1);
-- --------------------------------------------------------
--
-- Структура таблицы `kafedra`
--
CREATE TABLE `kafedra` (
`id` int(11) NOT NULL,
`name` varchar(100) NOT NULL,
`idDec` int(11) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
--
-- Дамп данных таблицы `kafedra`
--
INSERT INTO `kafedra` (`id`, `name`, `idDec`) VALUES
(1, 'Кафедра физики', 1),
(3, 'Кафедра химии', 8),
(6, 'ИВС', 2),
(7, 'ТТС', 9);
-- --------------------------------------------------------
--
-- Структура таблицы `semestr`
--
CREATE TABLE `semestr` (
`id` int(11) NOT NULL,
`name` varchar(100) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
--
-- Дамп данных таблицы `semestr`
--
INSERT INTO `semestr` (`id`, `name`) VALUES
(2, 'Зимний'),
(3, 'Летний'),
(4, 'Итоговый');
-- --------------------------------------------------------
--
-- Структура таблицы `student`
--
CREATE TABLE `student` (
`id` int(11) NOT NULL,
`name` varchar(150) NOT NULL,
```

```
`idGroup` int(11) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
--
-- Дамп данных таблицы `student`
--
INSERT INTO `student` (`id`, `name`, `idGroup`) VALUES
(1, 'Иванов И И', 2),
(2, 'Петров П П', 4),
(3, 'Сидоров В В', 1),
(4, 'Ворошилов М А', 5),
(5, 'Березов И И', 1);
--
-- Индексы сохранённых таблиц
--
--
-- Индексы таблицы `decanat`
--
ALTER TABLE `decanat`
ADD PRIMARY KEY (`id`);
--
-- Индексы таблицы `ekzam`
--
ALTER TABLE `ekzam`
ADD PRIMARY KEY (`id`),
ADD KEY `idSem` (`idSem`);
--
-- Индексы таблицы `ekzst`
--
ALTER TABLE `ekzst`
ADD PRIMARY KEY (`id`),
ADD KEY `idSt` (`idSt`,`idEkz`),
ADD KEY `idEkz` (`idEkz`);
--
-- Индексы таблицы `groupa`
--
ALTER TABLE `groupa`
ADD PRIMARY KEY (`id`),
ADD KEY `idKaf` (`idKaf`);
--
-- Индексы таблицы `kafedra`
--
ALTER TABLE `kafedra`
ADD PRIMARY KEY (`id`),
ADD KEY `idDec` (`idDec`);
--
```

```
-- Индексы таблицы `semestr`
--
ALTER TABLE `semestr`
ADD PRIMARY KEY (`id`);
--
-- Индексы таблицы `student`
--
ALTER TABLE `student`
ADD PRIMARY KEY (`id`),
ADD KEY `idGroup` (`idGroup`);
--
-- AUTO_INCREMENT для сохранённых таблиц
--
--
-- AUTO_INCREMENT для таблицы `decanat`
--
ALTER TABLE `decanat`
MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=10;
--
-- AUTO_INCREMENT для таблицы `ekzam`
--
ALTER TABLE `ekzam`
MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=6;
--
-- AUTO_INCREMENT для таблицы `ekzst`
--
ALTER TABLE `ekzst`
MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=16;
--
-- AUTO_INCREMENT для таблицы `groupa`
--
ALTER TABLE `groupa`
MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=7;
--
-- AUTO_INCREMENT для таблицы `kafedra`
--
ALTER TABLE `kafedra`
MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=8;
--
-- AUTO_INCREMENT для таблицы `semestr`
--
ALTER TABLE `semestr`
MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=5;
--
-- AUTO_INCREMENT для таблицы `student`
```

```
--
ALTER TABLE `student`
MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=6;
--
-- Ограничения внешнего ключа сохраненных таблиц
--
--
-- Ограничения внешнего ключа таблицы `ekzam`
--
ALTER TABLE `ekzam`
ADD CONSTRAINT `ekzam_ibfk_1` FOREIGN KEY (`idSem`) REFERENCES `semestr` (`id`) ON
DELETE CASCADE ON UPDATE CASCADE;
--
-- Ограничения внешнего ключа таблицы `ekzst`
--
ALTER TABLE `ekzst`
ADD CONSTRAINT `ekzst_ibfk_1` FOREIGN KEY (`idSt`) REFERENCES `student` (`id`) ON
DELETE CASCADE ON UPDATE CASCADE,
ADD CONSTRAINT `ekzst_ibfk_2` FOREIGN KEY (`idEkz`) REFERENCES `ekzam` (`id`) ON
DELETE CASCADE ON UPDATE CASCADE;
--
-- Ограничения внешнего ключа таблицы `groupa`
--
ALTER TABLE `groupa`
ADD CONSTRAINT `groupa_ibfk_1` FOREIGN KEY (`idKaf`) REFERENCES `kafedra` (`id`) ON
DELETE CASCADE ON UPDATE CASCADE;
--
-- Ограничения внешнего ключа таблицы `kafedra`
--
ALTER TABLE `kafedra`
ADD CONSTRAINT `kafedra_ibfk_1` FOREIGN KEY (`idDec`) REFERENCES `decanat` (`id`) ON
DELETE CASCADE ON UPDATE CASCADE;
--
-- Ограничения внешнего ключа таблицы `student`
--
ALTER TABLE `student`
ADD CONSTRAINT `student_ibfk_1` FOREIGN KEY (`idGroup`) REFERENCES `groupa` (`id`) ON
DELETE CASCADE ON UPDATE CASCADE;
COMMIT;

/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
```

ПРИЛОЖЕНИЕ Б

Созданные схемы и диаграммы

Рисунок Б.1 – Общая схема взаимодействия компонентов шаблона MVC

Рисунок Б.2 – Схема базы данных

Рисунок Б.3 – Диаграмма классов приложения

Заключение

Таким образом в результате проделанной работы : Создана база данных на языке Sql и с помощью средств среды работы с СУБД MySql PhpMyAdmin; Составлены запросы к базе данных на языке Sql; Изучен шаблон проектирования MVC — один из самых популярных шаблонов проектирования, использующийся как в настольных программах, так и в мобильных и веб-приложениях; Изучен шаблон проектирования «Абстрактная фабрика»; Изучен язык программирования Java и среды разработки для него; Написан код на языке Java в среде разработки Inteliij Idea Community с использованием изученных шаблонов проектирования; Оформлена пояснительная записка по проделанной работе. Разработанное программное обеспечение удовлетворяет всем поставленным требованиям, оно платформено независимо

благодаря языку Java и кросс-платформенной СУБД MySql, гибкое и расширяемое за счет использования шаблонов проектирования, и выполняет все необходимые функции вывода данных, записи, редактирования, удаления и поиска.

Список использованных источников

1. [url] **Репозиторий GitHub** https://github.com/mihkek/DecanWork

Приложения

1. [электронный документ] 5ecf7a9017ffc_Курсовая работа _Каллаур А.П.-89435012_ИСиТ (БМ)_final.odt
2. [электронный документ] 5ecf7a96c2427_Курсовая работа _Каллаур А.П.-89435012_ИСиТ (БМ)_final.pdf
3. [электронный документ] 5ecf7a9cb7372_Репозиторий GitHub.txt