|  |  |
| --- | --- |
|  | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

|  |  |
| --- | --- |
| ФАКУЛЬТЕТ | ИУ «Информатика и системы управления» |

|  |  |
| --- | --- |
| КАФЕДРА | ИУ-1 «Системы автоматического управления» |

**ОТЧЕТ ПО УЧЕБНОЙ ПРАКТИКЕ**

|  |  |
| --- | --- |
| Студент | Дмитриев Максим |
|  | *фамилия, имя, отчество* |

|  |  |  |
| --- | --- | --- |
| Группа | ИУ1-11Б |  |

|  |  |
| --- | --- |
| Тип практики | Учебный практикум |

|  |  |
| --- | --- |
| Название предприятия | Кафедра «Системы автоматического управления» |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент |  | 03/12/2024 |  | М.О. Дмитриев |
|  |  | (Подпись, дата) |  | (И.О. Фамилия) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Руководитель практики |  | 03/12/2024 |  | А.А. Николаев |
|  |  | (Подпись, дата) |  | (И.О. Фамилия) |

|  |  |  |
| --- | --- | --- |
| Оценка |  |  |

*2024 г.*

**Министерство науки и высшего образования Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Московский государственный технический университет имени Н.Э. Баумана**

**(национальный исследовательский университет)»**

**(МГТУ им. Н.Э. Баумана)**

УТВЕРЖДАЮ

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Заведующий кафедрой | | | | | | | ИУ-1 | | |
|  | | | | | | | (Индекс) | | |
|  |  | | | |  | К.А. Неусыпин | | | |
|  | | | | | | (И.О. Фамилия) | | | |
|  | « | 24 | » | сентября | | | 20 | 24 | г. |

**ЗАДАНИЕ**

**на прохождение учебной практики**

|  |  |  |
| --- | --- | --- |
| Студент группы | ИУ1-11Б |  |
| Дмитриев Максим | | |
| (Фамилия, имя, отчество) | | |

|  |  |
| --- | --- |
| ***Задание*** | Ознакомление с системой контроля и доставки версий Git, удалённым |
| репозиторием проектов GitHub, основами программирования на Java. Реализация | |
| программного кода. | |
|  | |
|  | |
|  | |
|  | |
|  | |
|  | |
|  | |

***Оформление отчета по практике:***

|  |  |  |
| --- | --- | --- |
| Отчет на | 20 | листах формата А4. |
| Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.) | | |
| оформление графического материала в отчете по практике не предусмотрено | | |
|  | | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Дата выдачи задания | « | 24 | » | сентября | 20 | 24 | г. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Руководитель Практики** |  | 24/09/2024 |  | А.А. Николаев |
|  |  | (Подпись, дата) |  | (И.О. Фамилия) |
|  |  |  |  |  |
| **Студент** |  | 24/09/2024 |  | М.О. Дмитриев |
|  |  | (Подпись, дата) |  | (И.О. Фамилия) |

# СОДЕРЖАНИЕ

[СОДЕРЖАНИЕ 1](#_Toc1)

[ВВЕДЕНИЕ 2](#_Toc2)

[Тема 1. Git и Github 3](#_Toc3)

[1.1. Общие сведения о Git и Github. 3](#_Toc4)

[1.2. Работа с Github. 3](#_Toc5)

[1.2.1. Создание репозиториев 3](#_Toc6)

[1.2.2. Связка репозиториев 5](#_Toc7)

[1.2.3. Ветви в Git 6](#_Toc8)

[1.2.4. Файлы «.gitignore» и «readme.md» 7](#_Toc9)

[Тема 2. Java 7](#_Toc10)

[2.1. Общие сведения о языке программирования Java 7](#_Toc11)

[2.2. Устройство языка Java 8](#_Toc12)

[2.2.1. Типы данных в Java 8](#_Toc13)

[2.2.2. Ввод и вывод данных 8](#_Toc14)

[2.2.3. Массивы в Java 9](#_Toc15)

[2.2.4. Ветвление Java 10](#_Toc16)

[2.2.5. Циклы в Java 11](#_Toc17)

[2.2.6. Классы Java 12](#_Toc18)

[2.2.7. Модификаторы доступа Java 12](#_Toc19)

[2.2.8. Наследование в Java 13](#_Toc20)

[2.3. Создание программы для работы с фигурами. 13](#_Toc21)

[2.4. Создание программы для учета студентов 14](#_Toc22)

[2.5. Создание класса для работы с массивом строк 14](#_Toc23)

[Заключение 15](#_Toc24)

[Список литературы 16](#_Toc25)

[Приложение 1 17](#_Toc26)

[Приложение 2 18](#_Toc27)

[Приложение 3 19](#_Toc28)

# ВВЕДЕНИЕ

Во время работы над большими проектами, особенно когда в них задействованы большие команды, одним из ключевых аспектов становится использование систем контроля версий. Эти системы позволяют эффективно управлять изменениями в коде, обеспечивая возможность совместной работы программистов и минимизируя риски потери данных. Наиболее популярными системами контроля версий являются Git, Mercurial и SVN (Subversion). В частности, Git зарекомендовал себя как наиболее распространённая система, активно используемая многими крупными компаниями, такими как Apple и Google. Среди главных преимуществ Git можно выделить его простоту в освоении, гибкость и мощные инструменты для управления проектами. Он позволяет командам легко отслеживать изменения, создавать различные ветки разработки и при необходимости возвращаться к предыдущим версиям кода, что значительно упрощает процесс командной работы.

Помимо систем контроля версий, разработка программного обеспечения требует использования языков программирования, которые служат основным инструментом для создания приложений. Одним из таких языков является Java, который занимает особое место в индустрии благодаря своей концепции "Write Once, Run Anywhere" (WORA), она подразумевает возможность запуска приложений на любой платформе с поддержкой Java Virtual Machine (JVM), что делает Java идеальным выбором для кроссплатформенной разработки.

Кроме того, Java известен своей высокой степенью безопасности и стабильности, что делает его предпочтительным выбором для разработки критически важных систем. Язык также поддерживается большим сообществом разработчиков, что облегчает изучение Java как для новичков, так и для опытных программистов. Наличие множества обучающих ресурсов и активных форумов способствует постоянному обмену знаниями и опыту между разработчиками. Таким образом, все перечисленные факторы делают Java одним из ведущих языков программирования в современном мире разработки приложений. Его универсальность и возможности делают его незаменимым инструментом для создания высококачественного программного обеспечения" Перепиши этот текст так, чтобы в результате он не был похож на себя

# Git и Github

## Общие сведения о Git и Github.

**Git** — распределённая система управления версиями. Проект был создан Линусом Торвальдсом для управления разработкой ядра Linux, первая версия выпущена 7 апреля 2005 года; координатор — Дзюн Хамано.

**GitHub** — крупнейший веб-сервис для хостинга IT-проектов и их совместной разработки.

Веб-сервис основан на системе контроля версий Git и разработан компанией GitHub, Inc. Сервис бесплатен для проектов с открытым исходным кодом и (с 2019 года) небольших частных проектов, предоставляя им все возможности, а для крупных корпоративных проектов предлагаются различные платные тарифные планы.

## Работа с Github.

### Создание репозиториев

Перед началом работы с github необходимо создать репозиторий (репозиторий – каталог, хранящий фалы проекта, это могут быть файлы кодов, документы, видео и аудиозаписи, а также изображения) на локальном компьютере.

Для этого в нужной директории следует выполнить команду «**git init»** (Рис. 1).

Эта команда создаёт в текущем каталоге новый подкаталог с именем .git, содержащий все необходимые файлы – структуру Git.

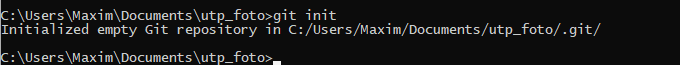


Рис. 1. Инициализация локального репозитория Git

Команда **«git status»** позволяет узнать состояние рабочего каталога, проверить индексацию изменений, а также увидеть какие файлы не отслеживаются git (Рис. 2).

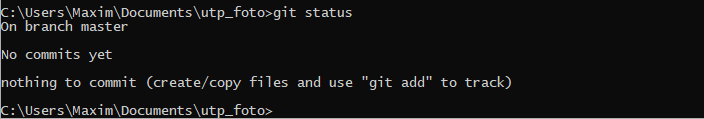


Рис. 2. Работа команды «git status»

Для того чтобы начать отслеживать изменения в файлах используется команда **«git add»** к которой в качестве аргумента можно добавить название необходимых файлов или «.» для добавления всех файлов в данной директории (Рис. 3).

Для того что бы сохранить состояние всех отслеживаемых файлов используется команда «**git commit – m ‘Название коммита’**» (Рис. 4).

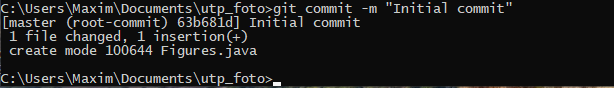


Рис. 4. Использование комманды git commit

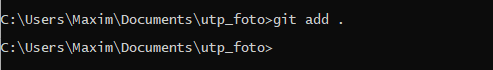


Рис. 3 . Использование команды «git add .»

Команды «**git log**» и «**git log --oneline**» (более компактная версия первой) предоставляют возможность просмотра всех сделанных commit’ов (Рис. 6).



Рис. 6. Использование команды git log

Для того чтобы перестать отслеживать файл необходимо использовать команду **«git rm —cached»**, она не удалит файл из каталога, но git перестанет отслеживать изменения в нем (Рис. 5).

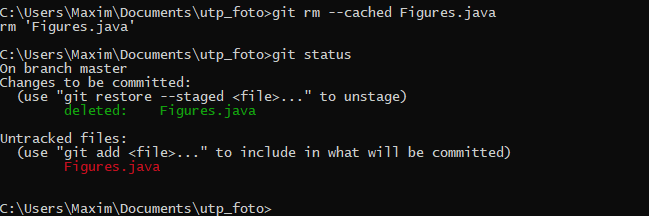


Рис. 5. Пример работы команды git rm --cached

Для того чтобы вернуться в одно из ранее сохраненных состояний репозитория (откатиться к коммиту) используется команда «**git revert <хэш\_коммита>**» (Рис. 7).

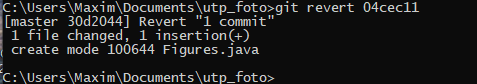


Рис. 7. Пример использования команды git revert

### Связка репозиториев

Локальный и глобальный репозитории могут быть связаны для внесения изменений в проект и их сохранения на удаленном сервере, что помогает избежать потери данных. Для этого необходимо на сайте [github.com](https://github.com/) создать репозиторий, задать ему необходимое название, и выбрать будет ли репозиторий публичным или приватным. После создания репозитория github предложит инструкция по связыванию репозиториев (Рис. 8).

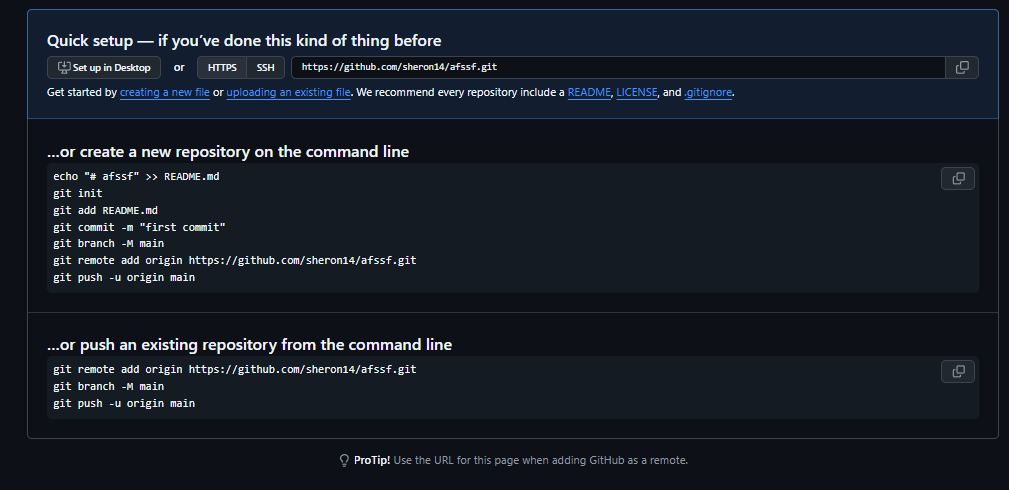


Рис. 8. Способы связки репозиториев

Далее необходимо настроить конфигурации GitHub с помощью команд:

* «**git config --global <user.name>»**
* «**git config --global <user.email>»**

Чтобы связать локальный репозиторий с только что созданным репозиторием на GitHub используется команда «**git remote add origin <link>**» (Рис. 9).

Для того чтобы синхронизировать локальный и удаленный репозитории используется команда «**git push –u origin <Название ветки>**» (Рис. 10).



Рис. 9. Связка локального репозитория с удаленным

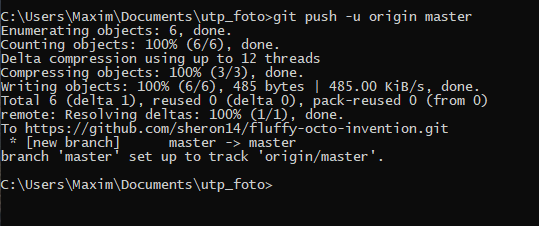


Рис. 10. Синхронизация локального и удаленного репозиториев

GitHub позволяет пользователям работать над проектом на разных компьютерах, для этого достаточно клонировать репозиторий с помощью команды «**git clone <ссылка>**» (Рис. 11).

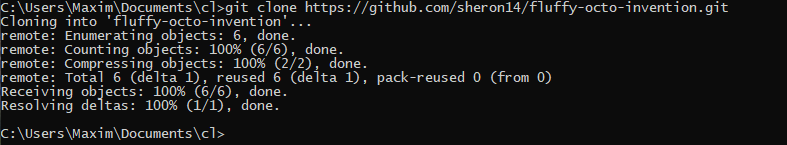


Рис. 11. Клонирование удаленного репозитория на локальный компьютер

Для переноса изменений с глобального репозитория на локальный используется команда «**git pull**» (Рис. 12).

### Ветви в Git

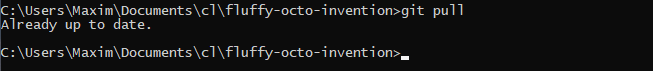


Рис. 12. Перенос изменений в локальный репозиторий

В Git ветка представляет собой параллельный поток развития кодовой базы. Это позволяет работать над разными версиями или функциями независимо друг от друга.

Чтобы создать новую ветку, выполните команду «**git branch <имя\_ветки>**». Например, чтобы создать ветку для новой функции, нужно выполнить следующую команду (Рис. 13):



Рис. 13. Создание новой ветки

Чтобы переключиться на другую ветку, выполните команду «**git checkout <имя\_ветки>**». Например, чтобы переключиться на ветку ‘new\_function’, необходимо выполнить следующую команду (Рис. 14):

После завершения работы над веткой ее можно объединить с основной веткой. Это делается с помощью команды «**git merge**». Например, чтобы объединить ветку `new\_function` с основной веткой, следует выполнить следующую команду (Рис. 15):



Рис. 14. Переключение на другую ветку

Когда ветка больше не нужна, ее можно удалить с помощью команды **«git branch -d <имя\_ветки>**». Например, чтобы удалить ветку `новая\_функция`, выполните следующую команду (Рис. 16):

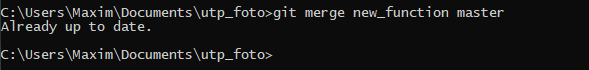


Рис. 15. Слияние веток

Преимущества использования веток:

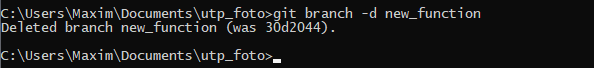


Рис. 16. Удаление ветки

* Параллельная разработка: Позволяет нескольким разработчикам работать над разными функциями или исправлениями одновременно.
* Изоляция кода: Обновления кода в ветке развития не влияют на основную ветку, что обеспечивает стабильность.
* Экспериментирование: Позволяет разработчикам экспериментировать с новыми идеями или изменениями без риска испортить основную ветку.
* Управление версиями: Ветки позволяют отслеживать различные версии кодовой базы и возвращаться к предыдущим состояниям, если это необходимо.

### Файлы «.gitignore» и «readme.md»

Файл README — это текстовый файл, который используется для предоставления информации о программном проекте или пакете. Его цель - помочь пользователям понять, как использовать и настраивать проект, а также любую другую информацию, которую разработчики считают важной. Он обычно располагается в корневом каталоге проекта.

Содержание файла README обычно включает:

* Описание проекта
* Инструкции по установке и настройке
* Примеры использования
* Список авторов и участников
* Ссылки на документацию и другие ресурсы

Файл .gitignore — это текстовый файл, который используется системой управления версиями Git для исключения определенных файлов и папок из отслеживания. Это полезно для таких файлов, как:

* Бинарные файлы (например, изображения, видео)
* Скомпилированные файлы (например, исполняемые файлы)
* Конфиденциальные данные (например, пароли, ключи API)
* Кэшированные файлы
* Установочные файлы

Правила в файле .gitignore используют шаблонный синтаксис, позволяющий использовать подстановочные знаки для сопоставления файлов и папок.

Общие правила в файле .gitignore:

* Шаблоны чувствительны к регистру.
* Пустая строка игнорирует все файлы.
* "#" указывает начало комментария.
* Шаблон "" соответствует нулю или более символов в пути.
* Шаблон "" соответствует любому одному символу в пути.
* Шаблон "\/" соответствует косой черте (/) в пути.

# Java

## Общие сведения о языке программирования Java

**Java**— строго типизированный объектно-ориентированный язык программирования общего назначения, разработанный компанией Sun Microsystems (в последующем приобретённой компанией Oracle). Разработка ведётся сообществом, организованным через Java Community Process; язык и основные реализующие его технологии распространяются по лицензии GPL.

## Устройство языка Java

### Типы данных в Java

Примитивные типы данных

Примитивные типы данных в Java предназначены для хранения одного значения и не относятся к объектам. В языке Java существуют следующие примитивные типы:

* byte: Целое число со знаком, занимающее 8 бит (1 байт), с диапазоном от -128 до 127.
* short: Целое число со знаком, занимающее 16 бит (2 байта), с диапазоном от -32,768 до 32,767.
* int: Целое число со знаком, занимающее 32 бита (4 байта), с диапазоном от -2,147,483,648 до 2,147,483,647.
* long: Целое число со знаком, занимающее 64 бита (8 байт), с диапазоном от -9,223,372,036,854,775,808 до 9,223,372,036,854,775,807.
* float: Число с плавающей точкой занимающее 32 бита (4 байта).
* double: Число с плавающей точкой в формате двойной точности занимающее 64 бита (8 байт).
* boolean: Логический тип данных, принимающий значения true или false.
* char: Символ в кодировке Юникод, занимающий 16 бит (2 байта).

Ссылочные типы данных

Ссылочные типы данных представляют собой указатели на объекты в памяти. В отличие от примитивных типов данных, которые хранят значение непосредственно, ссылки на объекты хранят адрес расположения объекта. В Java имеется только один тип данных для ссылок на объекты.

Специальное значение null

Значение null является специальным индикатором и используется для обозначения того, что ссылка не указывает на какой-либо существующий объект.

### Ввод и вывод данных

Ввод данных в Java

В языке программирования Java для ввода данных используются специальные классы, которые предлагают разнообразные методы для считывания информации из различных источников, таких как консоль, файлы и другие.

Среди основных классов, предназначенных для ввода данных, выделяются:

* Scanner: Это наиболее универсальный класс для получения ввода данных. Он предоставляет возможность считывать данные разных типов — от строк до чисел и не только.
* BufferedReader: Данный класс позволяет более точно управлять процессом ввода. Он имеет возможность читать строки как посимвольно, так и построчно, что дает разработчикам гибкость в обработке данных.
* InputStream: Это базовый класс для работы с вводом байтов. Он часто используется для выполнения более сложных операций ввода, таких как работа с бинарными данными и потоками информации.

Что касается методов ввода данных, то их можно перечислить следующим образом:

* nextLine(): Этот метод считывает строку целиком, включая символ новой строки на конце.
* nextInt(): Метод предназначен для считывания целого числа, позволяя легко получать числовые значения.
* nextDouble(): Используется для считывания числа с плавающей запятой, что удобно при работе с дробными значениями.
* nextBoolean(): Этот метод считывает логическое значение (true или false), что полезно при получении булевых данных.
* read(): Метод читает один байт данных, что может быть необходимо в случае обработки низкоуровневой информации.

Вывод данных в Java

Для вывода информации в Java также существуют определенные классы и методы, которые позволяют записывать данные на консоль, в файлы и другие места хранения.

К основным классам для вывода данных можно отнести:

* System.out: Это стандартный класс, используемый для вывода информации на консоль. Он обеспечивает удобный способ отображения данных пользователю.
* PrintWriter: Этот класс предоставляет более продвинутые функции управления выводом. Он позволяет записывать данные в файлы или потоки, что делает его особенно полезным при работе с текстовой информацией.
* OutputStream: Базовый класс для вывода байтовых данных. Часто применяется для выполнения более сложных операций записи, например, при работе с бинарной информацией.

Среди методов вывода данных выделяются:

* System.out.println(): Этот метод выводит данные на экран и автоматически переходит на новую строку после завершения вывода.
* System.out.print(): Метод выводит информацию без перехода на новую строку, что позволяет продолжать вывод на той же строке.
* PrintWriter.println(): Печатает данные с добавлением символа новой строки в конце.
* PrintWriter.print(): Выводит данные без перехода на новую строку, обеспечивая контроль над форматом отображаемой информации.

### Массивы в Java

Массив в Java представляет собой контейнер, который способен хранить набор элементов одного типа.

Чтобы объявить массив, необходимо указать тип данных элементов, затем имя массива и добавить квадратные скобки (Рис. 18).

Для доступа к конкретному элементу массива используется его индекс, заключенный в квадратные скобки (например, numbers[0] для обращения к первому элементу) (Рис. 17).  
 Длина массива может быть получена через специальное поле length (например, numbers.length), которое возвращает количество элементов в массиве (Рис. 19).

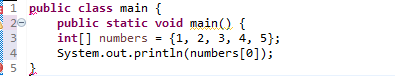


Рис. 17. Получение эллемента массива по индексу

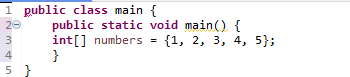


Рис. 18. Объявление массива

Особенности массивов в Java

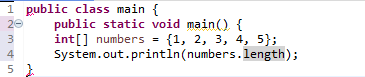


Рис. 19. Получение длинны массива

* Массивы в Java имеют фиксированный размер, который определяется во время их инициализации и не может быть изменен после этого.
* Все элементы в массиве должны быть одного и того же типа данных, что обеспечивает однородность коллекции.
* Индексация массива начинается с 0, что означает, что первый элемент имеет индекс 0, второй — 1 и так далее.
* Хотя массивы сами по себе не могут принимать значение null, они могут содержать элементы, которые равны null.
* В Java массивы являются объектами. Это значит, что к ним можно применять методы и производить сравнения с другими массивами.

### Ветвление Java

Ветвление в Java — это механизм, позволяющий выполнять разные блоки кода в зависимости от выполнения определенных условий. Этот подход помогает управлять потоком выполнения программы и принимать решения на основе логических условий. В Java для ветвления используются несколько ключевых конструкций:

**Оператор if-else**

С помощью конструкции if-else можно задать различные действия в зависимости от истинности условия (Рис. 20).

**Оператор switch**

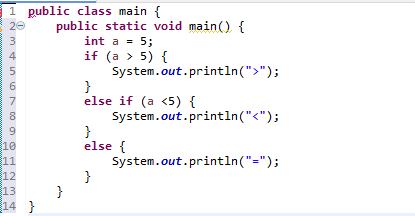


Рис. 20. Оператор if-else

Конструкция switch позволяет выбрать один из множества вариантов выполнения кода в зависимости от значения выражения (Рис. 21).

### Циклы в Java

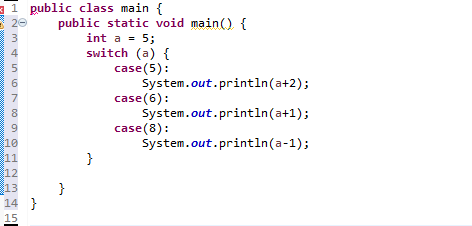


Рис. 21. Оперератор switch

Циклы в Java — это конструкции, которые позволяют многократно выполнять блок кода до тех пор, пока выполняется заданное условие. В Java есть несколько видов циклов, каждый из которых предназначен для различных сценариев. Рассмотрим основные из них:

Цикл for используется, когда заранее известно количество итераций. Он состоит из трех частей: инициализации, условия и итерации (Рис. 23).

Цикл while выполняет блок кода, пока условие истинно. Условие проверяется перед каждой итерацией (Рис. 22).

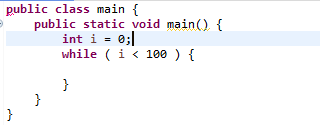


Рис. 22. Цикл while

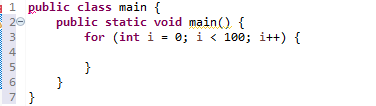


Рис. 23. Цикл for

### Классы Java

Класс Java - это шаблон или план, используемый для создания объектов. Он содержит данные (переменные) и методы (функции), которые определяют поведение и характеристики объектов (Рис. 24).

Компоненты класса

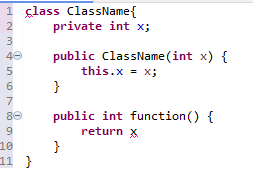


Рис. 24. Пример класса Java

* Переменные: Представляют данные, связанные с объектами.
* Конструкторы: Специальные методы, вызываемые при создании объекта для инициализации его переменных.
* Методы: Функции, определяющие поведение объектов.

Для того чтобы работать с классом необходимо содать объект данного класса.Для этого в Java используется метод ‘new’ (Рис. 25).

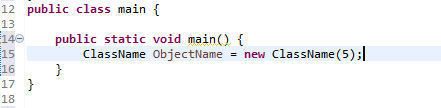


Рис. 25. Создание объекта класса

Чтобы вызвать метод класса нужно указать объект и метод класса к которому принадлежит данный объект с необходимыми аргументами (Рис. 26).



Рис. 26. Вызов метода класса

### Модификаторы доступа Java

В Java используются следующие модификаторы доступа:

* **public**: публичный, общедоступный класс или член класса. Поля и методы, объявленные с модификатором public, видны другим классам из текущего пакета и из внешних пакетов.
* **private**: закрытый класс или член класса, противоположность модификатору public. Закрытый класс или член класса доступен только из кода в том же классе.
* **protected**: такой класс или член класса доступен из любого места в текущем классе или пакете или в производных классах, даже если они находятся в других пакетах
* **Модификатор по умолчанию**. Отсутствие модификатора у поля или метода класса предполагает применение к нему модификатора по умолчанию. Такие поля или методы видны всем классам в текущем пакете.
* **static**: поля общие для всех объектов одного класса и классов наследников.
* **abstract**: классы объекты которых не могут быть созданы, но которые используются для наследования. Методы которые должны быть переопределены в производных классах.

### Наследование в Java

Наследование — это механизм, который позволяет создавать новые классы (называемые дочерними), унаследующие свойства и поведение от существующих классов (родительских или базовых).

Ключевые слова для наследования:

* extends: Указывает на то, что дочерний класс наследует функциональность родительского класса.
* super: Используется для вызова конструктора, методов и переменных родительского класса.

Типы наследования:

* Единичное наследование: Дочерний класс наследует от одного родительского класса.
* Множественное наследование: Дочерний класс может наследовать от нескольких родительских классов (не поддерживается в Java).
* Многоуровневое наследование: Дочерний класс наследует от другого дочернего класса,

Дочерние классы могут переопределять методы, унаследованные от родительских классов, предоставляя собственные реализации. Кроме того, дочерние классы должны вызывать суперконструктор родительского класса с помощью ключевого слова super в своих конструкторах.

## Создание программы для работы с фигурами.

**Задание**:

Реализовать классы, которые будет описывать прямоугольник, круг, треугольник и методы, которые будут вычислять их площади и периметры. Создать репозиторий на гите, загрузить туда и отправить ссылку.

Если фигуры с такими параметрами не существует, то выбрасываем ошибку, следующей командой: throw new RuntimeException("There is no figure with such parameters.")

Залить на удаленный репозиторий и отправить ссылку.

При выполнении задания реализованы следующие классы:

* Figures – базовый класс содержащий метод returnError для возвращения ошибки и конструктор класса.
* Rectangle — производный класс для прямоугольника с методами для вычисления площади и периметра, а также конструктором возвращающим ошибки если такая фигура не может существовать (стороны меньше или равны 0).
* Circle — производный класс для прямоугольника с методами для вычисления площади и периметра, а также конструктором возвращающим ошибки если такая фигура не может существовать (радиус меньше или равен 0).
* Triangle — производный класс для прямоугольника с методами для вычисления площади и периметра, а также конструктором возвращающим ошибки если такая фигура не может существовать(стороны меньше или равны 0, или две стороны в сумме меньше или равны 3).
* Main — демонстрация использования всех классов.

Разаработанные классы и их методы были протестированы.

Исходный код можно увидеть в репозитории [github](https://github.com/sheron14/Figures.git) или в [Приложении 1](#_Приложение) .

## Создание программы для учета студентов

**Задание :**

Реализовать абстрактный класс Student с полями ФИО, курс, оценка за последний экзамен, определить конструктор и абстрактный метод writeExam. Создать два класса наследника: IUStudent, MathStudent и определить у них абстрактный метод так чтобы он выводил информацию о том, что студент такого-то направления пишет экзамен. Создать экземпляры данных классов и протестировать работу методов.

Разработан требуемый абстрактный класс Student с защищенными полями ФИО(fullName), курс(course), балл(lastExamScore). В классе Student реализованы конструктор класса и абстракатный метод writeExam. В дочерних классах метод writeExam переопределен. Он возвращает строку ‘<ФИО> пишет экзамен по <предмет>’. ФИО задается при создании объекта класса, а предмет определяется тем к какому из классов принадлежит объект. У объектов класса IUStudent это экзамен по информатике, а у объектов класса MathStudent это экзамен по математике.

Разработанные классы и их методы были протестированы.

Исходный код можно увидеть в репозитории [github](https://github.com/sheron14/Students.git) или в [Приложении 2](#_Приложение_6).

## Создание класса для работы с массивом строк

**Задание:**

Реализовать класс, который будет хранить массив строк в порядке возрастания их длины. Класс должен содержать методы добавления элемента, возвращения максимального (по длине) элемента, возвращение средней длины элементов. Протестировать.

В хоже выполнения задания был разработан класс stringArray c полем arrayList представляющим собой динамический массив строк. У класса определен конструктор, метод add принимающий в качестве аргумента строку которую надо добавить в массив, затем в зависимости от длинны этой строки выбирается необходимая позиция в массиве, такая что массив я вляется отсортированным по возрастанию. Также реализован метод getMaxLengthEllement, проверяющий, что массив не пустой, и в случае если это верно возвращающий самую длинную строку в массиве ( последнюю строку в данном массиве). Метод getAverageLEngth, реализованный в этом классе, путем перебора всех строк в массиве ищет суммарное количество всех символов в них, дели его на количество строк и возвращает полученное значение – среднюю длину строки. Для удобства тестирования разработанного класса реализован метод display, выводящий на экран все строки из массива.

Разработанные классы и их методы были протестированы.

Исходный код можно увидеть в репозитории [github](https://github.com/sheron14/string_container.git) или в [Приложении 3](#_Приложение_4).

# Заключение

В данной работе представлено детальное описание базового функционала, необходимого для эффективной работы с локальными и удаленными репозиториями, используя системы Git и GitHub, а также язык программирования Java. В процессе изучения рассмотрены основные команды, операторы и функции, которые предоставляют Git и GitHub, а также изучен базовый синтаксис языка программирования Java.

Git — это мощная система контроля версий, которая позволяет разработчикам отслеживать изменения в коде, сотрудничать над проектами и управлять различными версиями своих файлов. GitHub, в свою очередь, является популярной платформой для хостинга проектов, использующих систему Git, и предоставляет множество дополнительных инструментов для совместной работы, таких как возможности для управления задачами и ведения обсуждений.

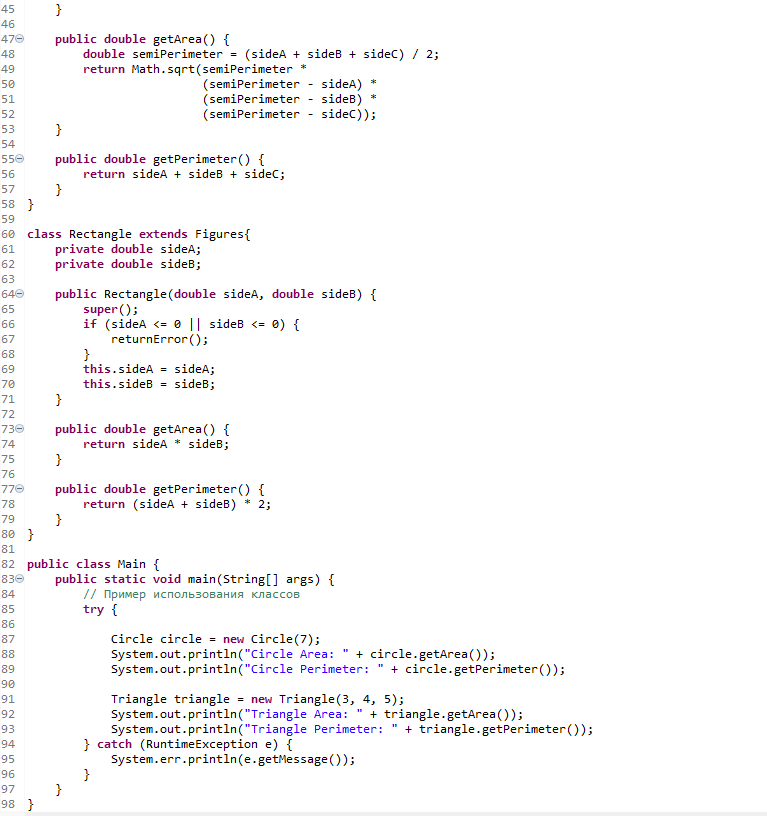
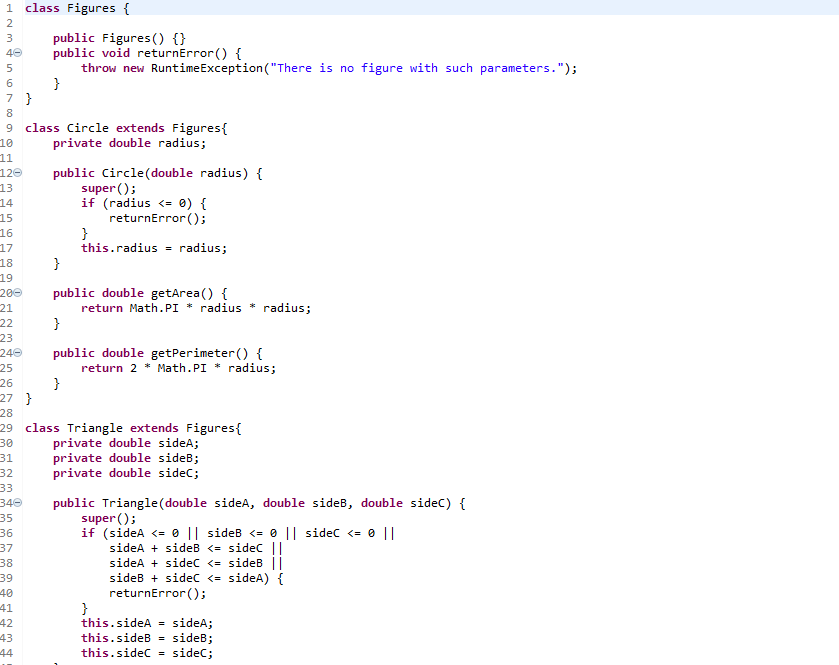
Язык программирования Java также занимает важное место в мире разработки программного обеспечения благодаря своей платформенной независимости и объектно-ориентированному подходу. В работе мы будем изучены базовые концепции Java 8, включая его синтаксис, типы данных, а также основные операторы и конструкции. Уделено особое внимание таким аспектам, как ввод и вывод данных, работа с массивами, а также управление потоками выполнения программы через ветвление и циклы.

В результате были решены три задачи на языке Java, исходный код решений загружен на GitHub. При разработке использованы полученные знания о git, а также об основах Java: типах данных, массивах, ветвлении и циклах , а также о классах и их методах.

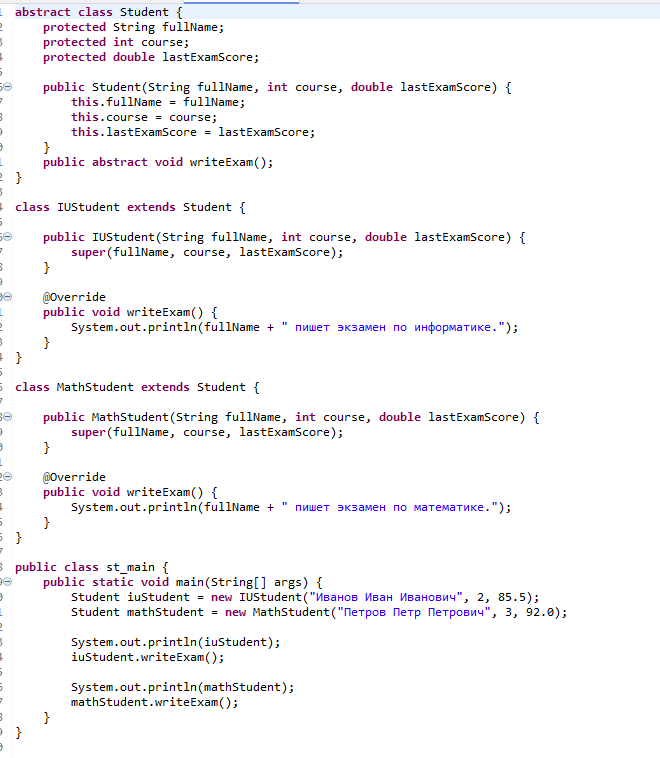
# Список литературы

* Чакон С., Штрауб Б.  Git для профессионального программиста - Санкт-Петербург : Питер, 2016. - 496 с. - ISBN 978-5-496-01763-3.
* Профессиональная работа с Git. - Москва : АСТ, 2024. - 160 с. - ISBN 978-5-17-160274-1
* Васильев А. Java для всех. - СПб.: Питер, 2020. - 512 с. - ISBN 978-5-4461-1382-8
* Эванс, Бенджамин Дж., Флэнаган, Дэвид. Java. Справочник разработчика, 7-е изд. : Пер. с англ. - СПб. : ООО "Диалектика", 2019. - 592 с. - ISBN 978-5-907144-61-3 (рус. )
* Флэнаган Д. Java в примерах. Справочник, 2-е издание - Пер. с англ. - СПб: Символ-Плюс, 2003. - 664 с. - ISBN 5-93286-042-1
* Документация по Java от Oracle. URl: https://docs.oracle.com/en/java/
* Документация по Git. URL: https://git-scm.com/book/en/v2

# Приложение 1



# Приложение 2



# Приложение 3

