|  |  |
| --- | --- |
|  | **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 | Соин Андрей Дмитриевич |
|  | *фамилия, имя, отчество* |

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

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

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент |  | 24/12/2021 |  | А.Д. Соин |
|  |  | (Подпись, дата) |  | (И.О. Фамилия) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Руководитель практики |  | 24/12/2021 |  | А.А. Николаев |
|  |  | (Подпись, дата) |  | (И.О. Фамилия) |

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

*2021 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 | 21 | г. |

**ЗАДАНИЕ**

**на прохождение учебной практики**

|  |  |  |
| --- | --- | --- |
| Студент группы | ИУ1-11Б |  |
| Соин Андрей Дмитриевич | | |
| (Фамилия, имя, отчество) | | |

|  |  |
| --- | --- |
| ***Задание*** | Ознакомление с системой контроля и доставки версий Git, удалённым |
| репозитроием проектов GitHub, основами программирования на Python, а также | |
| библиотеками Pandas и Numpy. Реализация программного кода. | |
|  | |
|  | |
|  | |
|  | |
|  | |
|  | |
|  | |

***Оформление отчета по практике:***

|  |  |  |
| --- | --- | --- |
| Отчет на | 24 | 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 | 21 | г. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Руководитель Практики** |  | 24/09/2021 |  | А.А. Николаев |
|  |  | (Подпись, дата) |  | (И.О. Фамилия) |
|  |  |  |  |  |
| **Студент** |  | 24/09/2021 |  | А.Д. Соин |
|  |  | (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)

[Тема 2. Java 8](#_Toc6)

[2.1. Общие сведения о языке программирования Java 8](#_Toc7)

[2.2. Устройство языка Java 8](#_Toc8)

[2.2.1. Типы данных в Java 8](#_Toc9)

[2.2.2. Операции с данными 9](#_Toc10)

[2.2.3. Ввод и выод данных 9](#_Toc11)

[2.2.4. Массивы в Java 10](#_Toc12)

[2.2.5. Ветвление Java 11](#_Toc13)

[2.2.6. Цикл while в Java 11](#_Toc14)

[2.2.7. Цикл for в Java 12](#_Toc15)

[2.2.8. Классы Java 12](#_Toc16)

[2.3. Создание программы для работы с фигурами. 13](#_Toc17)

[2.4. Создание программы для учета студентов 14](#_Toc18)

[2.5. Создание класса для работы с массивом строк 14](#_Toc19)

[Заключение 15](#_Toc20)

[Список литературы 16](#_Toc21)

[Приложение 1 17](#_Toc22)

[Приложение 2. 18](#_Toc23)

[Приложение 3. 19](#_Toc24)

# ВВЕДЕНИЕ

При разработке больших проектов, особенно когда задействованы большие команды или даже несколько команд, крайне важна возможность совместного внесения изменений без потери времени и эффективности. В таких случаях на помощь приходят системы контроля версий — специализированное программное обеспечение, которое позволяет хранить и управлять несколькими версиями одного проекта, а также легко перемещаться между ними. Это значительно упрощает совместную работу над кодом, минимиз и потери данных.

Среди наиболее популярных систем контроля версий можно выделить следующие:

* Git
* Mercurial
* SVN (Subversion)
* Perforce

Git является самой распространенной из этих систем и используется многими крупными компаниями, такими как Apple, Google, Amazon и Facebook. Основные преимущества Git заключаются в его простоте освоения, высокой гибкости и мощных возможностях для управления проектами. Эта система позволяет разработчикам легко отслеживать изменения, возвращаться к предыдущим версиям кода и работать над различными ветками разработки, что делает процесс более организованным и эффективным. Системы контроля версий стали неотъемлемой частью современного процесса разработки программного обеспечения.

Разработка программного обеспечения невозможна без использования языков программирования, которые служат основным инструментом для создания приложений и систем. Одним из наиболее популярных языков в мире разработки на протяжении последних двух десятилетий является Java. Этот язык программирования был создан с акцентом на простоту, портативность и производительность, что сделало его предпочтительным выбором для многих разработчиков.

Java выделяется среди других языков благодаря своей концепции "Write Once, Run Anywhere" (WORA), что означает, что программа, написанная на Java, может выполняться на любой платформе, поддерживающей Java Virtual Machine (JVM). Это позволяет разработчикам создавать кроссплатформенные приложения без необходимости переписывать код для различных операционных систем. Благодаря этому принципу Java стала основным языком для разработки корпоративных приложений, мобильных приложений под Android, а также серверного программного обеспечения.

Еще одним значительным преимуществом Java является обширная экосистема библиотек и фреймворков. Существует множество мощных инструментов, таких как Spring и Hibernate, которые упрощают разработку сложных приложений и помогают разработчикам эффективно управлять данными и бизнес-логикой. Эти библиотеки позволяют значительно сократить время разработки и повысить качество создаваемого ПО.

Java также известна своей безопасностью и стабильностью, что делает ее идеальным выбором для критически важных систем, таких как банковское программное обеспечение или системы управления данными. Более того, язык имеет большое сообщество разработчиков и множество ресурсов для обучения, что облегчает процесс освоения языка как для начинающих программистов, так и для опытных специалистов.

Таким образом, Java продолжает оставаться одним из ведущих языков программирования благодаря своей универсальности, богатой экосистеме инструментов и активному сообществу. Эти качества делают его незаменимым в мире разработки современных приложений.

# Git и Github

## Общие сведения о Git и GitHub

Git представляет собой распределённую систему управления версиями, созданную Линусом Торвальдсом для упрощения разработки ядра Linux. Первая версия была выпущена 7 апреля 2005 года, а координатором проекта стал Дзюн Хамано.

GitHub — это крупнейший веб-сервис для хостинга IT-проектов и их совместной разработки, основанный на системе контроля версий Git и разработанный компанией GitHub, Inc. Сервис предоставляет бесплатный доступ для проектов с открытым исходным кодом и, начиная с 2019 года, для небольших частных проектов, предоставляя им все необходимые функции. Для крупных корпоративных проектов предлагаются различные платные тарифные планы.

## Работа с GitHub

### Создание репозиториев

Перед тем как начать работу с GitHub, необходимо создать репозиторий — каталог, который будет хранить файлы проекта (например, код, документы, изображения и аудиофайлы) на локальном компьютере.

Для создания репозитория в нужной директории следует выполнить команду git init. Эта команда создаст новый подкаталог с именем .git, в котором будут содержаться все необходимые файлы структуры Git.

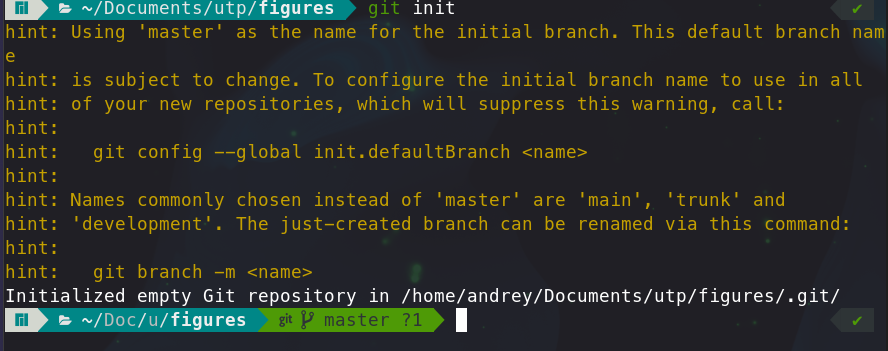


Рис. 1. Инициализация локального репозитория Git

Команда git status позволяет проверить текущее состояние рабочего каталога, увидеть индексацию изменений и узнать, какие файлы не отслеживаются системой Git.

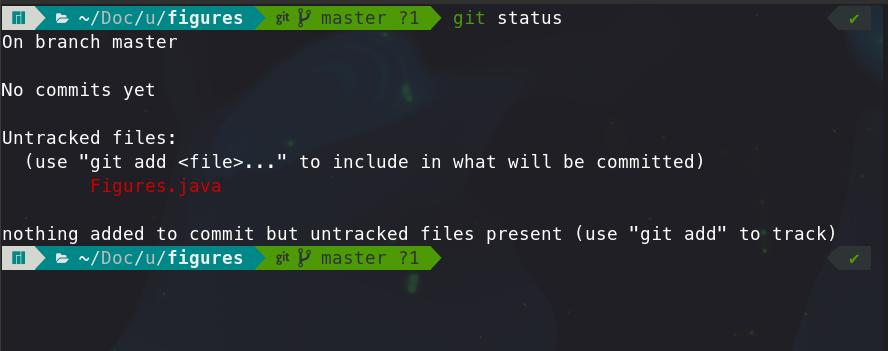


Рис. 2. Работа команды «git status»

Чтобы начать отслеживание изменений в конкретных файлах, используйте команду git add, добавив название файлов или указав . для добавления всех файлов в текущей директории.



Рис. 3 . Использование команды «git add .»

Чтобы сохранить текущее состояние всех отслеживаемых файлов, применяется команда git commit -m 'Название коммита'.



Рис. 4. Использование команды git commit

Команды git log и git log --oneline (более компактная версия) позволяют просмотреть все выполненные коммиты.

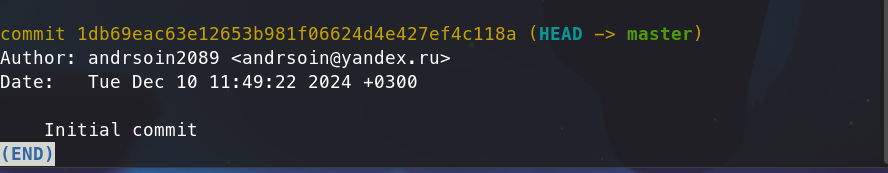


Рис. 6 . Вывод git log

Если необходимо прекратить отслеживание файла, используется команда git rm --cached. Эта команда не удалит файл из каталога, но Git перестанет следить за его изменениями.

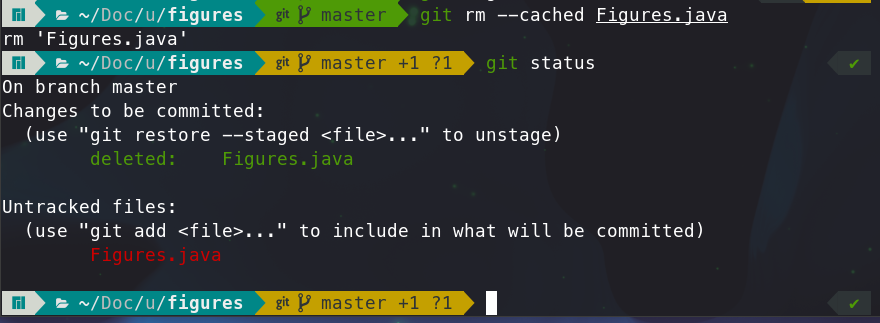


Рис. 5. Результат работы команды git rm

Для возврата к ранее сохранённому состоянию репозитория (отката к коммиту) применяется команда git revert <хэш\_коммита>.



Рис. 7. Пример использования команды git revert

Также существует команда git reset, которая используется для отката изменений, сделанных после последнего коммита.

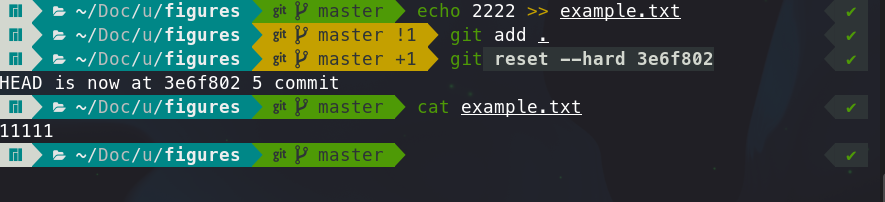


Рис. 8. Пример использования команды git reset

### Связка репозиториев

Локальный и удалённый репозитории можно связать для внесения изменений в проект и их сохранения на сервере, что помогает избежать потери данных. Для этого на сайте github.com создаётся новый репозиторий с заданным названием и выбором типа (публичный или приватный). После создания репозитория GitHub предложит инструкции по его связыванию.

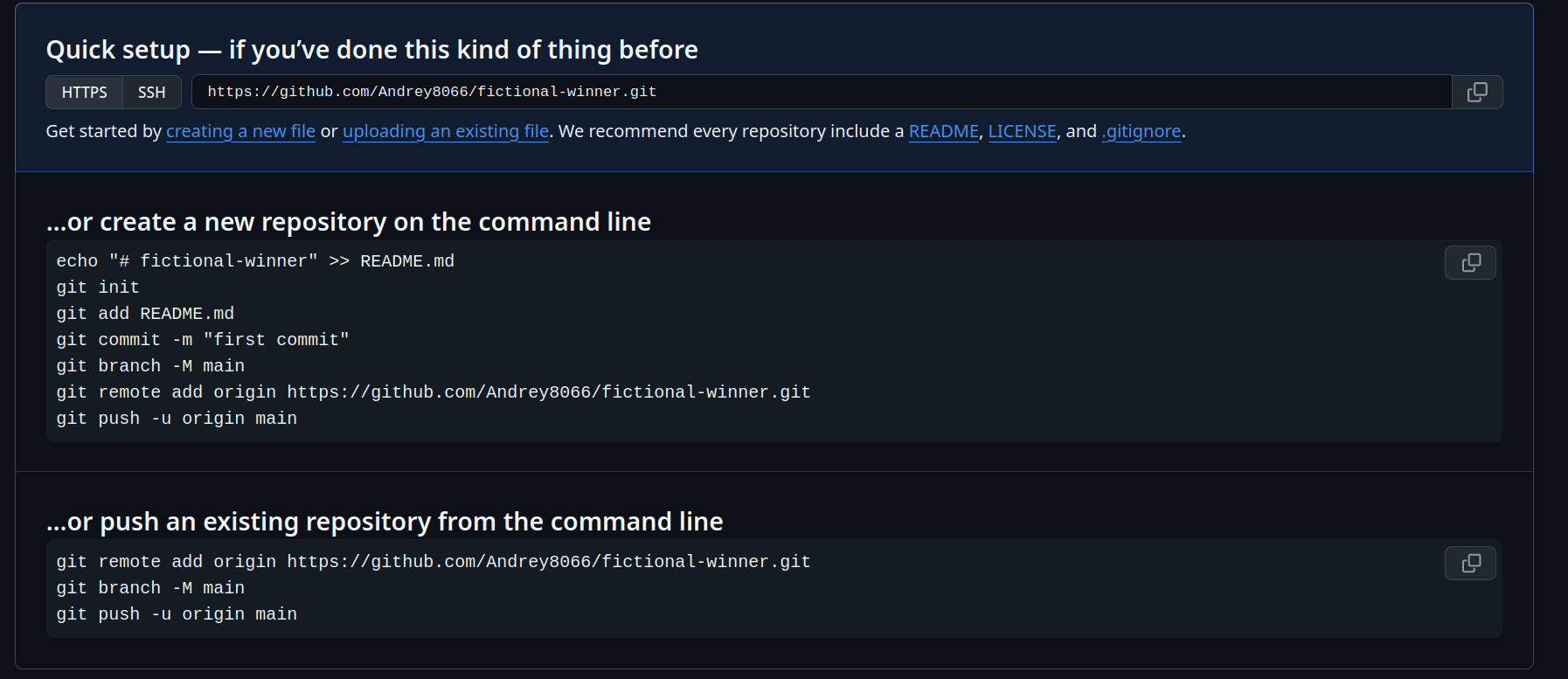


Рис. 9. Способы связки репозиториев

Далее нужно настроить конфигурации GitHub с помощью следующих команд:

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

Чтобы связать локальный репозиторий с только что созданным на GitHub, используется команда git remote add origin <link>.

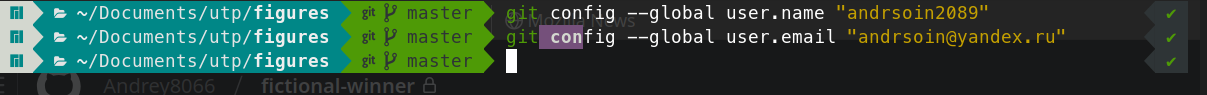


Рис. 10. Конфигурация GitHub

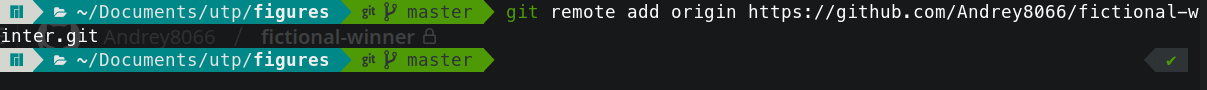


Рис. 11. Подключение удаленного репозитория к локальному

Для передачи последнего коммита в удалённый репозиторий применяется команда git push -u origin <Название ветки>.

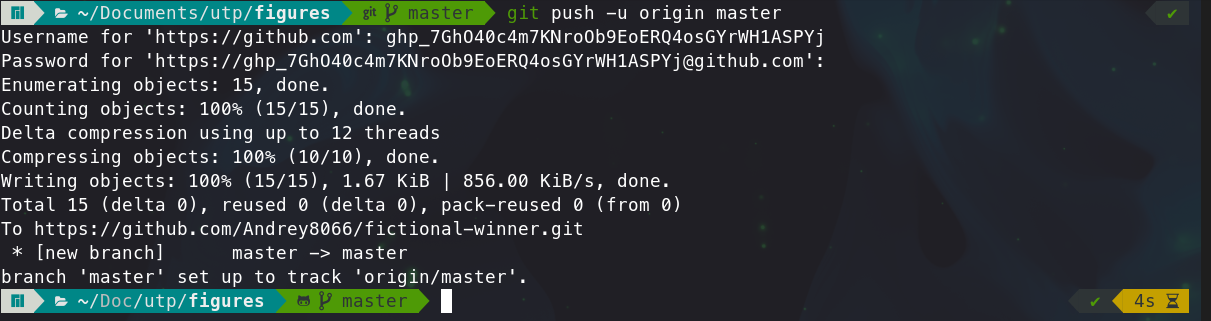


Рис. 12. Синхронизация коммитов

GitHub также позволяет пользователям работать над проектами на разных компьютерах: достаточно клонировать репозиторий при помощи команды git clone <ссылка>.

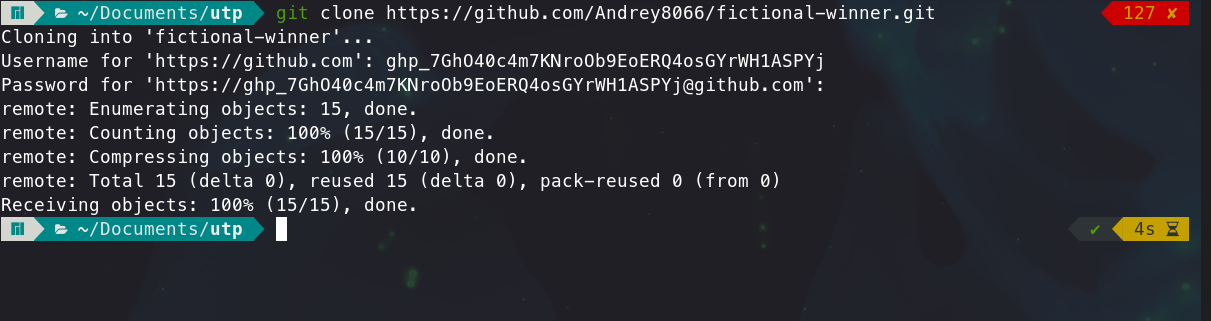


Рис. 13. Клонирование репозитория

Чтобы перенести изменения с глобального репозитория на локальный, используется команда git pull, а для переноса изменений с локального на глобальный — команда git push.



Рис. 15. Команда "git push"



Рис. 14. Перенос изменений с глобального репозитория на локальный

### Файлы .gitignore и readme.md

Файл README — это текстовый документ, который предоставляет информацию о проекте или пакете программного обеспечения. Его цель — помочь пользователям понять, как использовать и настраивать проект, а также предоставить другую важную информацию от разработчиков. Обычно файл README располагается в корневом каталоге проекта и может включать:

Описание проекта

Инструкции по установке и настройке

Примеры использования

Список авторов и участников

Ссылки на документацию и дополнительные ресурсы

Файл .gitignore является текстовым документом, который используется Git для исключения определённых файлов и папок из отслеживания. Это полезно для таких файлов, как:

Бинарные файлы (например, изображения или видео)

Скомпилированные файлы (например, исполняемые файлы)

Конфиденциальные данные (например, пароли или ключи API)

Кэшированные файлы

Установочные файлы

Правила в .gitignore используют шаблонный синтаксис с подстановочными знаками для сопоставления файлов и папок:

Шаблоны чувствительны к регистру.

Пустая строка игнорирует все файлы.

"#" обозначает начало комментария.

Шаблон "\*" соответствует нулю или более символам в пути.

Шаблон "?" соответствует одному символу в пути.

Шаблон "/" соответствует косой черте (/) в пути.

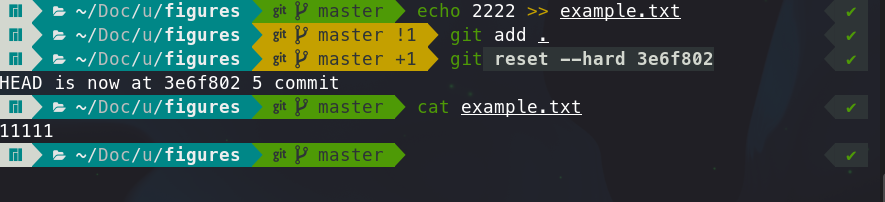


Рис. 8. Пример использования команды git revert

При создании глобального репозитория рекомендуется добавлять оба файла одновременно.

### Ветви в Git

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

Для создания новой ветки выполните команду git branch <имя\_ветки>. Например, чтобы создать ветку для новой функции, выполните следующую команду:



Рис. 16. Просмотр списка веток

Чтобы переключиться на другую ветку, используйте команду git checkout <имя\_ветки>. Например, чтобы перейти на ветку ‘new\_function’, выполните команду:



Рис. 17. Переключение на другую ветку

После завершения работы над веткой её можно объединить с основной веткой с помощью команды git merge. Например, чтобы объединить ветку new\_function с основной веткой:

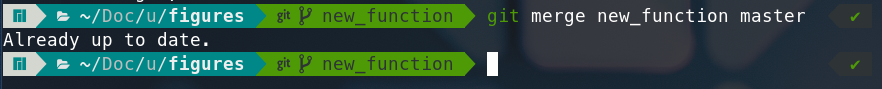


Рис. 18. Слияние веток

Когда ветка больше не нужна, её можно удалить командой git branch -d <имя\_ветки>. Например, чтобы удалить ветку ‘новая\_функция’, выполните команду:

**Преимущества использования веток:**

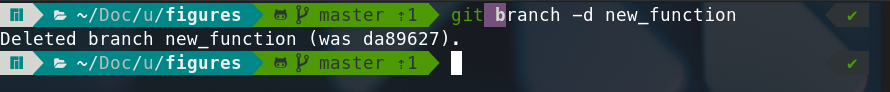


Рис. 19. Удаление ветки

Параллельная разработка: Позволяет нескольким разработчикам одновременно работать над разными функциями или исправлениями.

Изоляция кода: Обновления кода в отдельной ветке не влияют на основную ветку, что обеспечивает стабильность.

Экспериментирование: Разработчики могут пробовать новые идеи или изменения без риска повредить основную ветку.

Управление версиями: Ветки позволяют отслеживать различные версии кодовой базы и при необходимости возвращаться к предыдущим состояниям.

# 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 хранят единственное значение и не являются объектами. В Java есть следующие примитивные типы данных:

* byte: Целое число со знаком, состоящее из 8 бит (1 байт), со значениями от -128 до 127
* short: Целое число со знаком, состоящее из 16 бит (2 байта), со значениями от -32768 до 32767
* int: Целое число со знаком, состоящее из 32 бит (4 байта), со значениями от -2147483648 до 2147483647
* long: Целое число со знаком, состоящее из 64 бит (8 байт), со значениями от -9223372036854775808 до 9223372036854775807
* float: Число с плавающей запятой, состоящее из 32 бит (4 байта), с арифметикой одинарной точности IEEE 754
* double: Число с плавающей запятой, состоящее из 64 бит (8 байт), с арифметикой двойной точности IEEE 754
* boolean: Логический тип данных, который может принимать значения true или false
* char: Символ Юникода, состоящий из 16 бит (2 байта)

Ссылки на объекты

Ссылки на объекты представляют ссылки на объекты в куче. В отличие от примитивных типов данных, ссылки на объекты хранят адрес объекта, а не его значение. В Java есть один объект-ссылка.

Специальное значение null

Значение null - это специальное значение, которое указывает на то, что ссылка не указывает на действительный объект.

### Операции с данными

Арифметические операции

* + Сложение
* - Вычитание
* \ Умножение
* / Деление
* % Остаток от деления

Операции присваивания

* = Присваивание
* += Прибавление с присваиванием
* -= Вычитание с присваиванием
* \= Умножение с присваиванием
* /= Деление с присваиванием
* %= Остаток от деления с присваиванием

Операции сравнения

* == Равно
* != Не равно
* < Меньше
* <= Меньше или равно
* > Больше
* >= Больше или равно

Логические операции

* && И
* || Или
* ! Не

Битовые операции

* & Побитовое И
* Побитовое ИЛИ
* ^ Побитовое исключающее ИЛИ
* ~ Побитовое НЕ
* << Побитовый сдвиг влево
* >> Побитовый сдвиг вправо

### Ввод и вывод данных

Ввод данных в Java

В Java для ввода данных используются классы, которые предоставляют различные методы для чтения с консоли, файлов и других источников.

Основные классы для ввода данных:

* Scanner: Самый универсальный класс для ввода данных. Позволяет считывать данные разных типов, такие как строки, числа и другие.
* BufferedReader: Класс для более детального управления вводом. Позволяет читать строки посимвольно или построчно.
* InputStream: Базовый класс для ввода байтов. Используется для более сложных операций ввода, таких как работа с бинарными данными.

Методы ввода данных:

* `nextLine()` Считывает строку, включая знак новой строки
* `nextInt()` Считывает целое число
* `nextDouble()` Считывает число с плавающей запятой
* `nextBoolean()` Считывает булево значение
* `read()` Считывает один байт данных

Вывод данных в Java

Для вывода данных в Java используются классы и методы, которые позволяют записывать данные в консоль, файлы и другие источники.

Основные классы для вывода данных:

* System.out: Стандартный класс для вывода данных в консоль.
* PrintWriter: Класс для более детального управления выводом. Позволяет записывать данные в файлы или потоки.
* OutputStream: Базовый класс для вывода байтов. Используется для более сложных операций вывода, таких как запись бинарных данных.

Методы вывода данных:

* `System.out.println()` Выводит данные с новой строкой
* `System.out.print()` Выводит данные без новой строки
* `PrintWriter.println()` Выводит данные с новой строкой
* `PrintWriter.print()` Выводит данные без новой строки

### Массивы в Java

Массив в Java — это контейнер, который может хранить группу однотипных элементов.

Объявление массива

Чтобы объявить массив, укажите тип элемента, за которым следует имя массива и квадратные скобки:

Для доступа к элементу массива используйте индекс элемента в квадратных скобках:

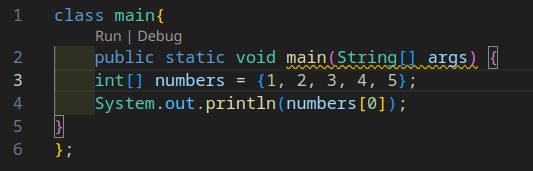


Рис. 20. Получение элемента массива по индексу

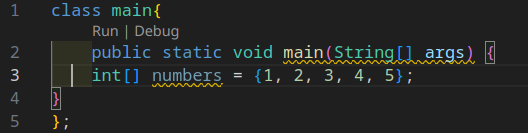


Рис. 21. Создание массива на Java

Длина массива хранится в поле `length`:

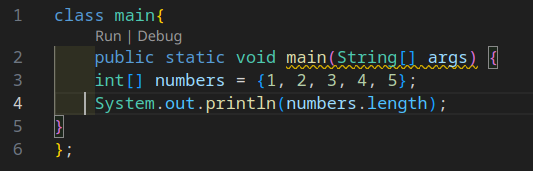


Рис. 22. Получение длинны массива

Java также поддерживает многомерные массивы. Например, двумерный массив может представлять таблицу:

Особенности массивов в Java

* Массивы в Java имеют фиксированный размер, определенный при инициализации.
* Элементы массива должны иметь один и тот же тип данных.
* Индексы массива начинаются с 0.
* Массивы не могут быть null, но могут содержать элементы null.
* Массивы в Java являются объектами, что означает, что к ним можно применять методы и сравнивать их с другими массивами.

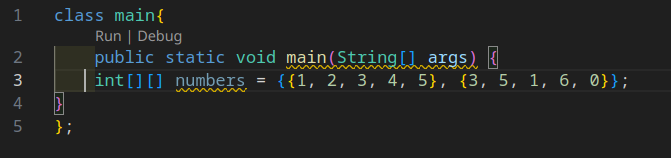


Рис. 23. Создание двумерного массива

### Ветвление Java

Ветвление - это конструкция управления потоком, позволяющая программе принимать разные пути выполнения в зависимости от результатов условия. В Java ветвление реализуется с помощью оператора `if`.

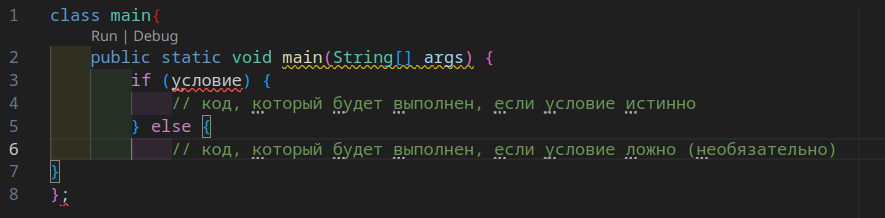


Рис. 24. Ветвление в java

### Цикл while в Java

Цикл `while` выполняет блок кода до тех пор, пока выполняется заданное условие.

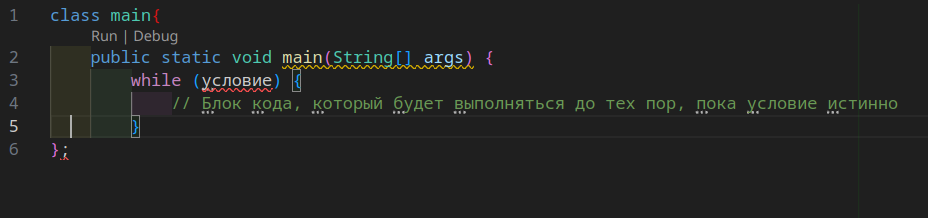


Рис. 25. Цикл while

### Цикл for в Java

Цикл `for` используется для итерации по последовательности значений. Он состоит из трех частей: инициализации, выражения и обновления.

### Классы Java

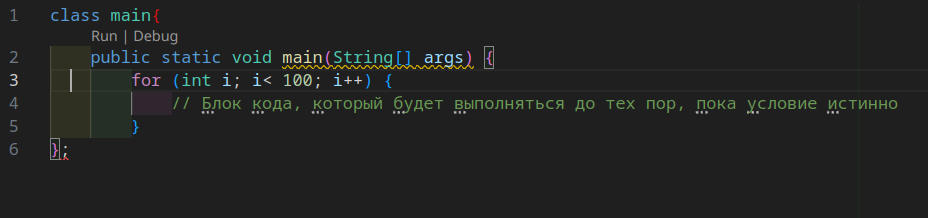


Рис. 26. Цикл for

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

Компоненты класса

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

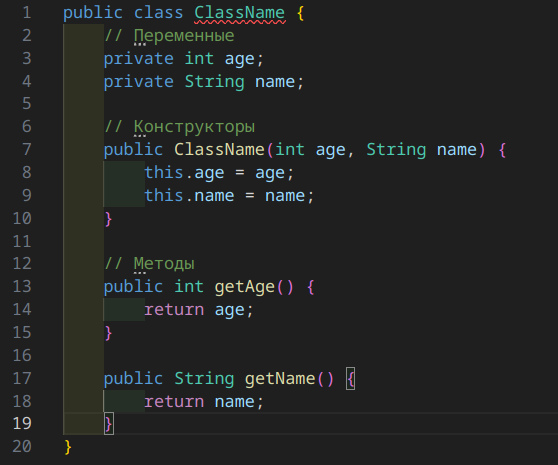


Рис. 27. Привер класса в Java

У компонентов класса могут быть разные модификаторы указывающие как на то кто может к ним обращаться, так и на то как их следует обрабатывать:

* Public: метод доступен из любого места программы.
* Protected: метод доступен в том же классе и подклассах.
* Default (неявный): метод доступен в том же пакете, что и сам класс.
* Private: метод доступен только в том же классе.
* Static: метод принадлежит классу, а не объекту.
* Final: метод не может быть переопределен в подклассах.
* Synchronized: метод может выполняться только одним потоком одновременно.
* Native: метод реализован на низкоуровневом языке (например, C++).
* Abstract: метод объявлен без реализации и должен быть переопределен в подклассах.

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

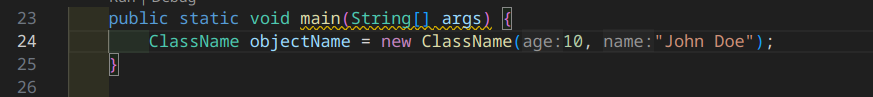


Рис. 28. Создание объекта класса в Java.

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

## Создание программы для работы с фигурами.

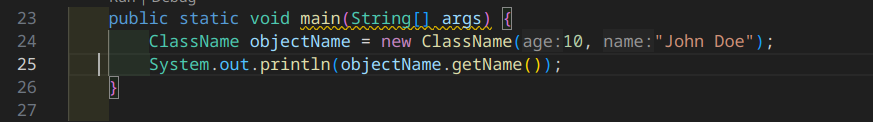


Рис. 29. Вызов метода класса

**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 реализован класс Figures содержащий параметры необходимой фигуры, метод для вывода данных о фигуре и метод возвращающий сообщающий о невозможности существования такой фигуры. Были созданы три дочерних класса Rectangle, Triangle, Circle, также содержащие необходимые данные о фигурах, а также переопределенный метод display, метод displaqy\_square, выводящий площадь фигуры, и метод display\_perimeter, выводящий периметр фигуры.

Разаработанные классы и их методы были протестированы.

Исходный код можно увидеть в репозитории github 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 абстрактны метод так чтобы он выводил информацию о том, что студент такого-то направления пишет экзамен. Создать экземпляры данных классов и протестировать работу методов.

При выполнении программы реализован абстрактный класс Students, с требуемыми защищенными полями ФИО(name), курс(course), последняя оценка(last\_mark). Также определен требуемый конструктор класса Students, объявлен абстрактный метод writeExam.

В наследуемых классах IUStudnts и MathStudents создан конструктор класса. Метод writeExam переопределен, в результате его вызова на экран выводится следующая строка: «Студент <номер курса> курса факультета <Факультет>, <ФИО> в данный момент пишет экзамен». Поле факультет зависит от класса объекта, ФИО и номер курса задаются при вызове конструктора.

Разработанные классы и их методы были протестированы.

Исходный код можно увидеть в репозитории github 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 программе реализован требуемый класс (String\_container), в нем создан защищенный динамический массив строк(container). Реализованы следующие методы:

* addString – метод для добавления элемента в конец массива.
* getLongest – метод возвращающий самую длиную строку в массиве основанный на переборе всех элементов массива и выбора самого длинного.
* getAverageLength-метод возвращающий среднюю длину строк в массиве в формате десятичной дроби. Основан на переборе всех элементов массива

Разработанные классы и их методы были протестированы.

Исходный код можно увидеть в репозитории github или в [Приложении 3](#_Приложение_4).

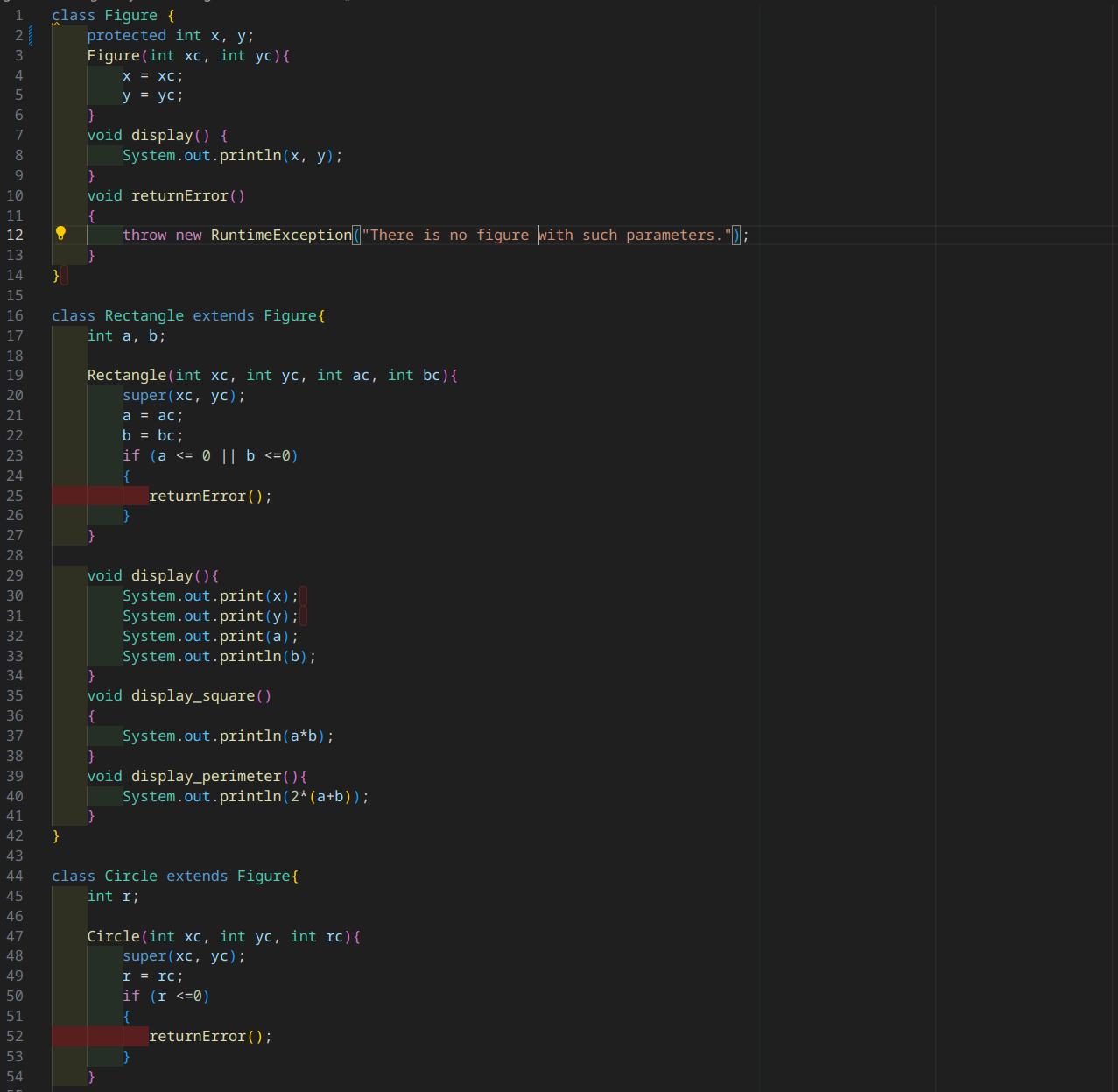
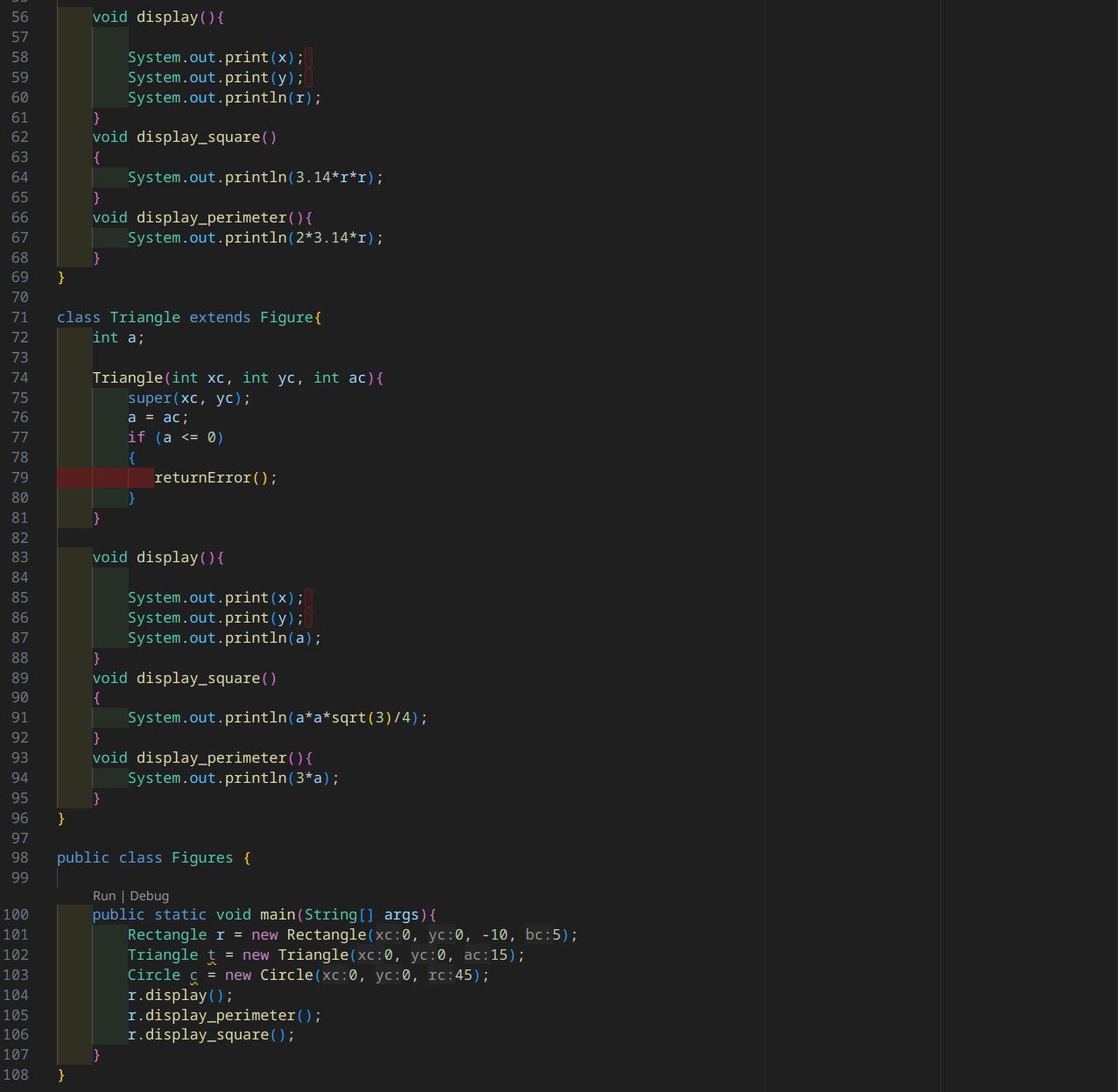
# Заключение

В этой работе описывается базовый функционал для работы с локальными и удаленными репозиториями Git и GitHub и языком программирования Java; представлены основные команды, операторы и функции Git и GitHub, а также базовый синтаксис языка программирования Java. Кроме того, приводятся примеры, демонстрирующие использование функциональных возможностей Git и Java, чтобы дать первое представление об основах системы Git и языке программирования Java и помочь студентам приобрести навыки работы с системой контроля версий Git и языком программирования Java.

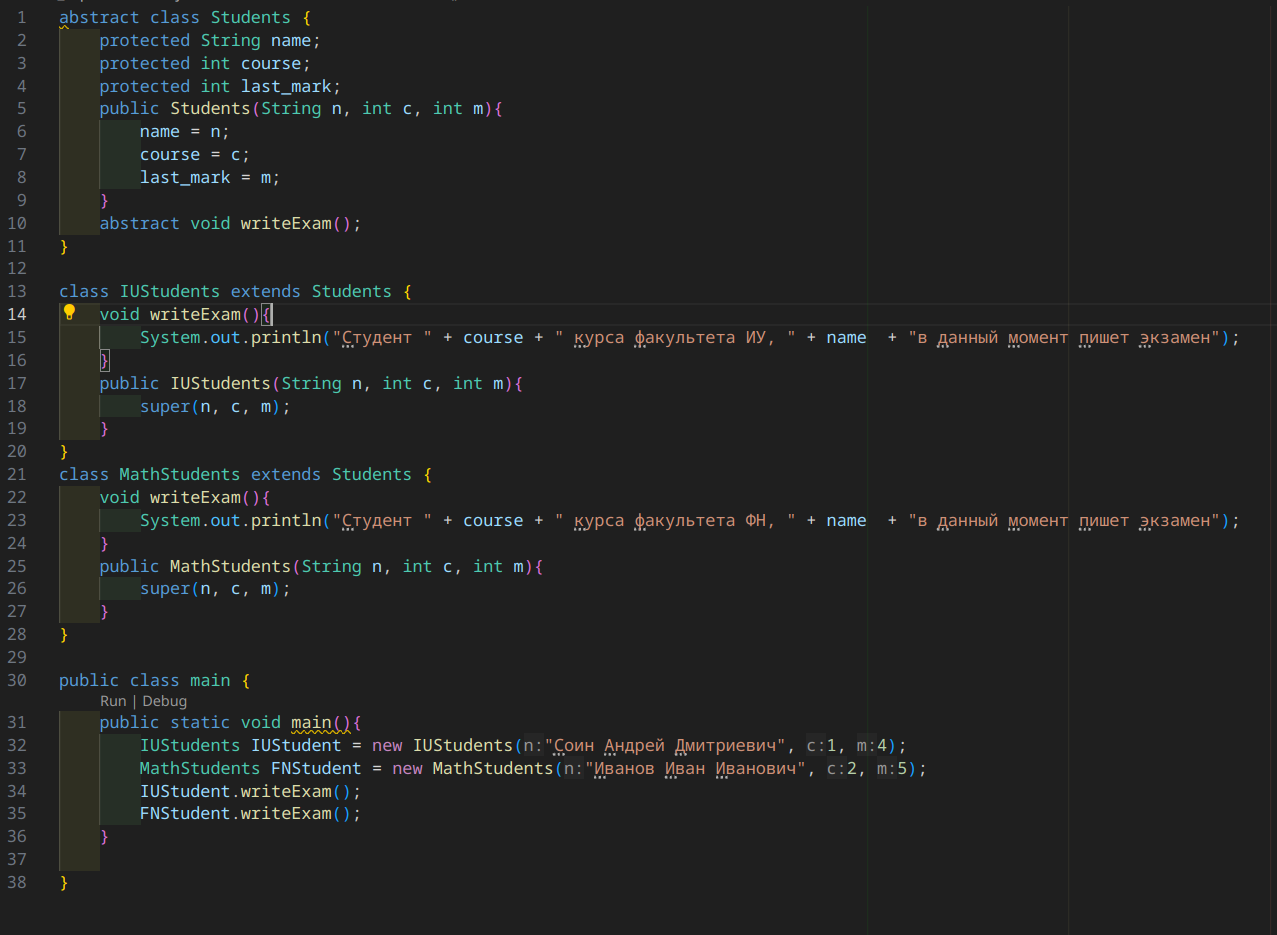
# Список литературы

* Герберт Шилдт Java. Полное руководство - 752, c ил стр., с ил.; ISBN 978-5-907515-36-9, 978-1-260-46355-2
* 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

