|  |  |
| --- | --- |
|  | **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. 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. 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)

[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 вывод 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)

[2.2.9. Модификаторы доступа Java 13](#_Toc17)

[2.2.10. Наследование в Java 13](#_Toc18)

[2.3. Создание программы для работы с фигурами. 14](#_Toc19)

[2.4. Создание программы для учета студентов 14](#_Toc20)

[2.5. Создание класса для работы с массивом строк 15](#_Toc21)

[Заключение 16](#_Toc22)

[Список литературы 17](#_Toc23)

[Приложение 1 18](#_Toc24)

[Приложение 2 19](#_Toc25)

[Приложение 3 20](#_Toc26)

# ВВЕДЕНИЕ

При разработке крупных проектов, особенно с участием больших команд, важно обеспечить совместное внесение изменений без потери времени и эффективности. Здесь на помощь приходят системы контроля версий — специальные программы для управления различными версиями проекта и упрощения совместной работы над кодом, что помогает минимизировать риски потери данных.

К числу популярных систем контроля версий относятся:

* Git
* Mercurial
* SVN (Subversion)

Git является наиболее распространенной системой и используется многими крупными компаниями, такими как Apple и Google. Его преимущества включают простоту освоения, гибкость и мощные инструменты для управления проектами. Git позволяет легко отслеживать изменения, возвращаться к предыдущим версиям кода и работать с разными ветками разработки, что делает процесс более организованным.

Разработка программного обеспечения также требует языков программирования, которые служат основным инструментом создания приложений. Java — один из самых популярных языков за последние два десятилетия, выделяющийся концепцией "Write Once, Run Anywhere" (WORA). Это позволяет запускать приложения на любой платформе с поддержкой Java Virtual Machine (JVM), что делает его идеальным для кроссплатформенной разработки.

Java обладает обширной экосистемой библиотек и фреймворков, таких как Spring и Hibernate, которые упрощают создание сложных приложений и помогают в управлении данными. Язык известен своей безопасностью и стабильностью, что делает его предпочтительным выбором для критически важных систем. Кроме того, наличие большого сообщества разработчиков и обучающих ресурсов облегчает освоение Java как для новичков, так и для опытных программистов.

Таким образом, Java продолжает оставаться одним из ведущих языков программирования благодаря своей универсальности и активному сообществу, что делает его незаменимым в современном мире разработки приложений.

# 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»**.

Эта команда создаёт в текущем каталоге новый подкаталог с именем .git (Рис. 1), содержащий все необходимые файлы – структуру Git.

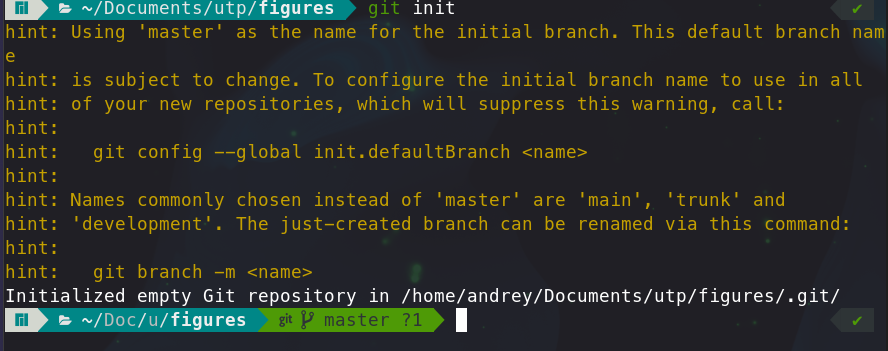


Рис. 1. Инициализация локального репозитория Git

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

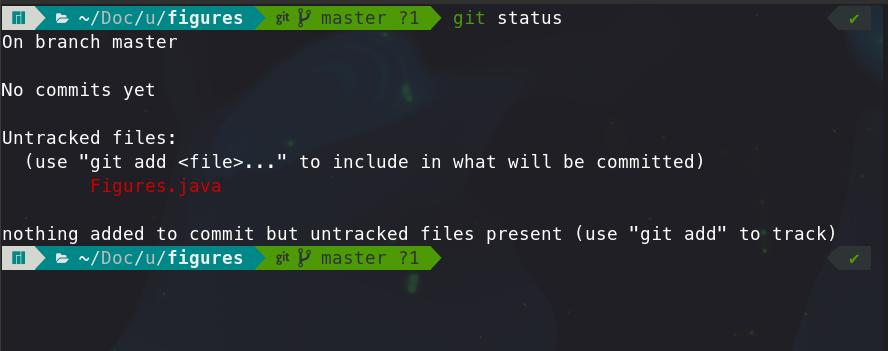


Рис. 2. Работа команды «git status»

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. 3 . Использование команды «git add .»

Команды «**git log**» и «**git log --oneline**» (более компактная версия первой) предоставляют возможность просмотра всех сделанных commit’ов (Рис. 5).



Рис. 4. Использование комманды git commit

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. 6).

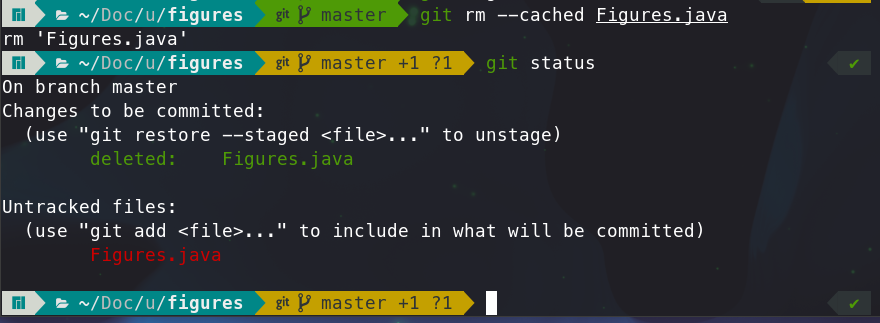


Рис. 5. Пример работы команды git rm --cached

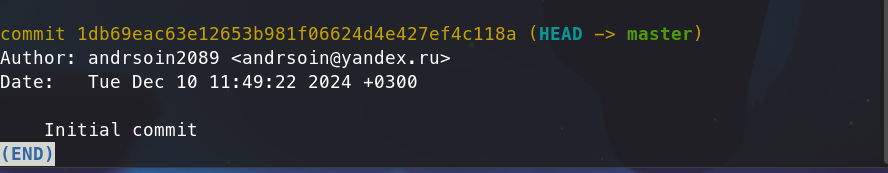


Рис. 6. Использование команды git log

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

Также существует команда «git reset» используемая для отката изменений сделанных с момента создания последнего коммита (Рис. 8).

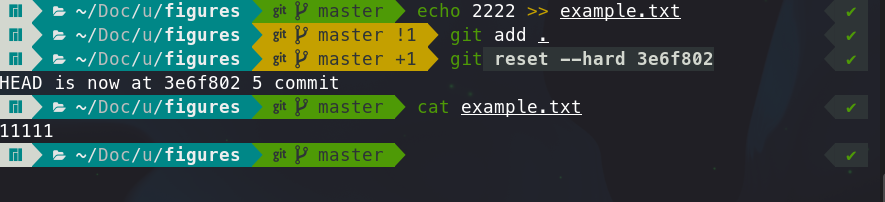


Рис. 8. Пример использования команды git reset

**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. 9).

Далее необходимо настроить конфигурации GitHub с помощью команд (Рис. 10):

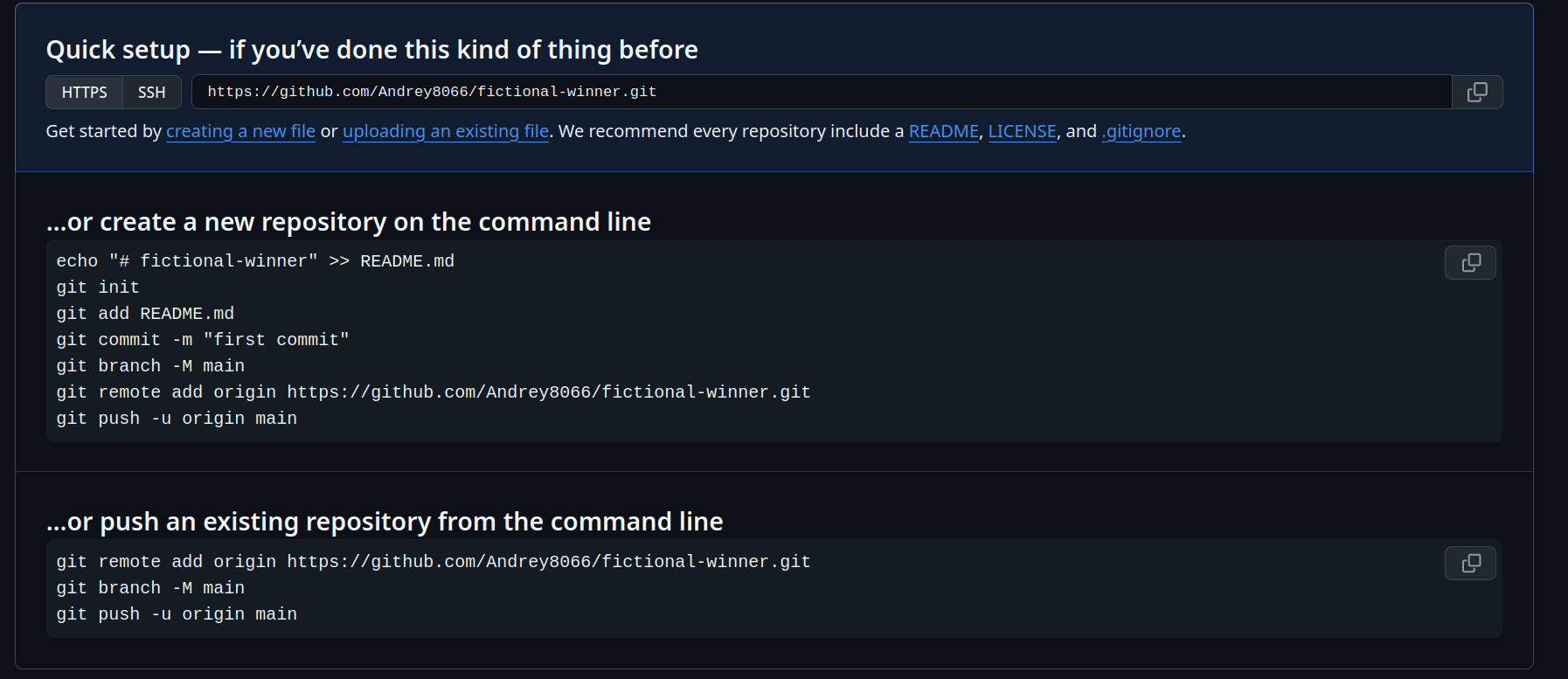


Рис. 9. Способы связки репозиториев

* «**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. 11).

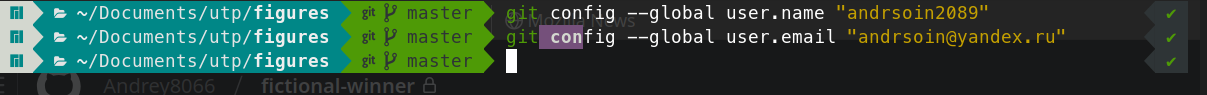


Рис. 10. Конфигурация GitHub

Для того чтобы передать последний commit в удалённый репозиторий испоьзуется команда «**git push –u origin <Название ветки>**» (Рис. 12).

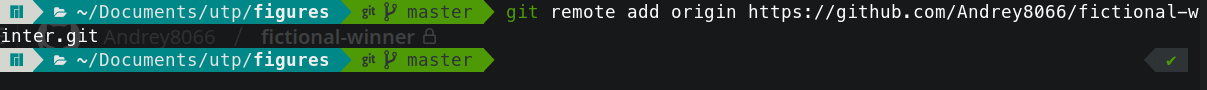


Рис. 11. Подключение удаленного репозитория к локальному

GitHub позволяет пользователям работать над проектом на разных компьютерах, для этого достаточно клонировать репозиторий с помощью команды «git clone <ссылка>» (Рис. 13).

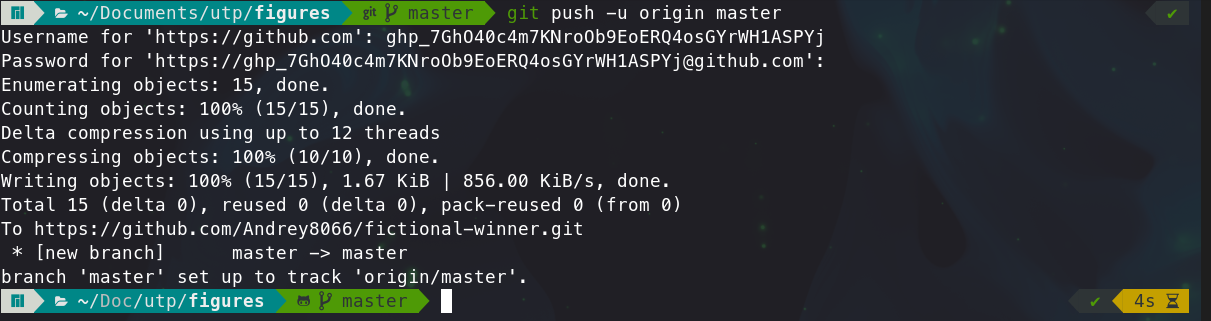


Рис. 12. Синхронизация коммитов

Для переноса изменений с глобального репозитория на локальный используется команда «**git pull**» (Рис. 14).

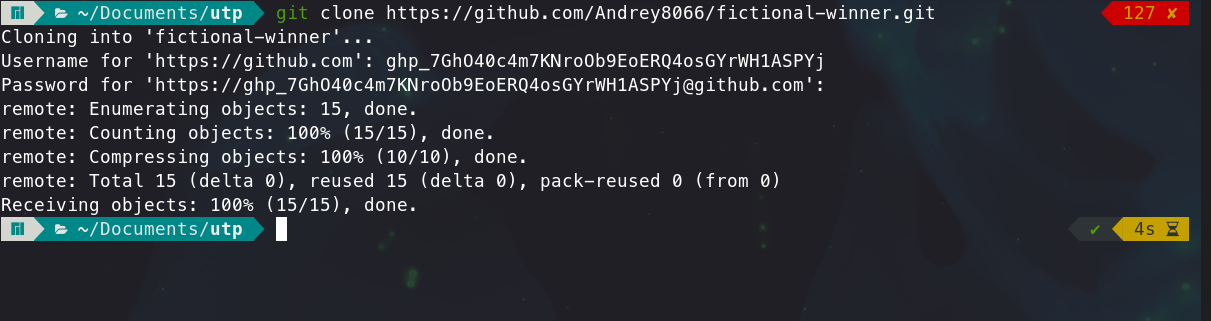


Рис. 13. Клонирование репозитория

Для переноса изменений с локального репозитория на глобальный используется команда «**git push**» (Рис. 15).



Рис. 14. Перенос изменений с глобального репозитория на локальный

Файлы «.gitignore» и «readme.md»



Рис. 15. Команда "git push"

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.

При создании глобального репозитория рекомендуется добавлять оба файла сразу. Для этого нужно отметить соответствующие поля (рис.19):

Ветви в Git

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. 16):



Рис. 16. 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. 17):

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. 18):



Рис. 17. Переключение на другую ветку

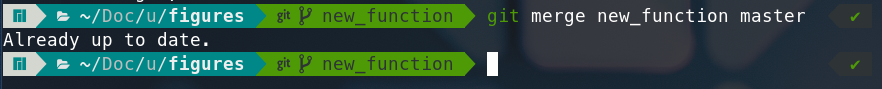


Рис. 18. Слияние веток

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. 19):

Преимущества использования веток:

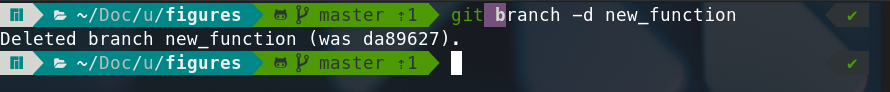


Рис. 19. Удаление ветки

* 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.

# 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 в формате одинарной точности IEEE 754, занимающее 32 бита (4 байта).
* double: Число с плавающей точкой в формате двойной точности IEEE 754, 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.

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

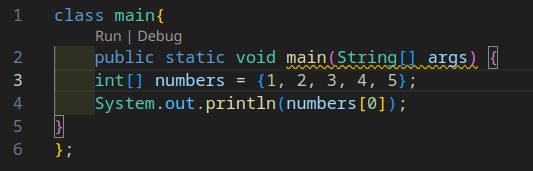


Рис. 21. Получение эллемента массива по индексу

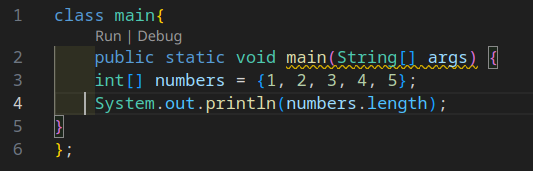


Рис. 22. Получение длинны массива

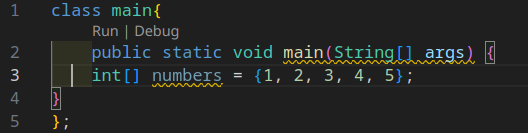


Рис. 20. Объявление массива

Особенности массивов в Java

* 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 проверяет истинность условия и выполняет блок кода, если условие истинно.**

**Оператор if-else**

С помощью конструкции if-else можно задать альтернативные действия в зависимости от истинности условия.

**Оператор switch**

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

### Циклы в Java

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.

Цикл while выполняет блок кода, пока условие истинно. Условие проверяется перед каждой итерацией.

На цикл while, но условие проверяется после выполнения блока кода. Это гарантирует, что блок выполнится хотя бы один раз.

Цикл for-each (или улучшенный цикл for) позволяет удобно перебрать элементы коллекций или массивов без необходимости использования индексов.

### 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. 27).

Компоненты класса

* 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. 28).

Чтобы вызвать метод класса нужно указать объект и метод класса к которому принадлежит данный объект с необходимыми аргументами (Рис. 29).

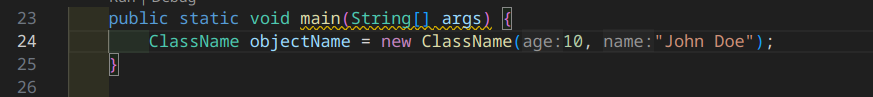


Рис. 28. Создание объекта класса в Java.

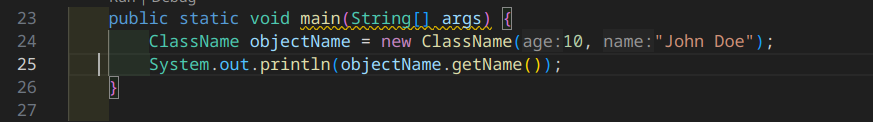


Рис. 29. Вызов метода класса

### 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).

Преимущества наследования:

Повторное использование кода: Можно использовать уже написанный код без необходимости его дублирования.

Расширяемость: Легче добавлять новые функции в дочерние классы.

Инкапсуляция: Позволяет скрыть детали реализации, обеспечивая доступ только к необходимым членам класса.

Полиморфизм: Возможность использования объектов различных классов через один интерфейс.

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, который также унаследовал что-то от своего родителя.

Доступ к членам классов:

Private: Члены недоступны в дочерних классах.

Protected: Члены доступны в дочерних классах и в других классах того же пакета.

Default (package-private): Члены доступны только в рамках одного пакета.

Public: Члены доступны из любого места.

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/Andrey8066/figures) 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.

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

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

Разработанные классы и их методы были протестированы.

Исходный код можно увидеть в репозитории [github](https://github.com/Andrey8066/students) 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. Для соблюдения упорядоченности массива реализован выбор места для вставки новой строки основанный на поиске последней строки длина которой больше необходимой и вставке новой строки перед ней.
* 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](https://github.com/Andrey8066/string_container) или в [Приложении 3](#_Приложение_4).

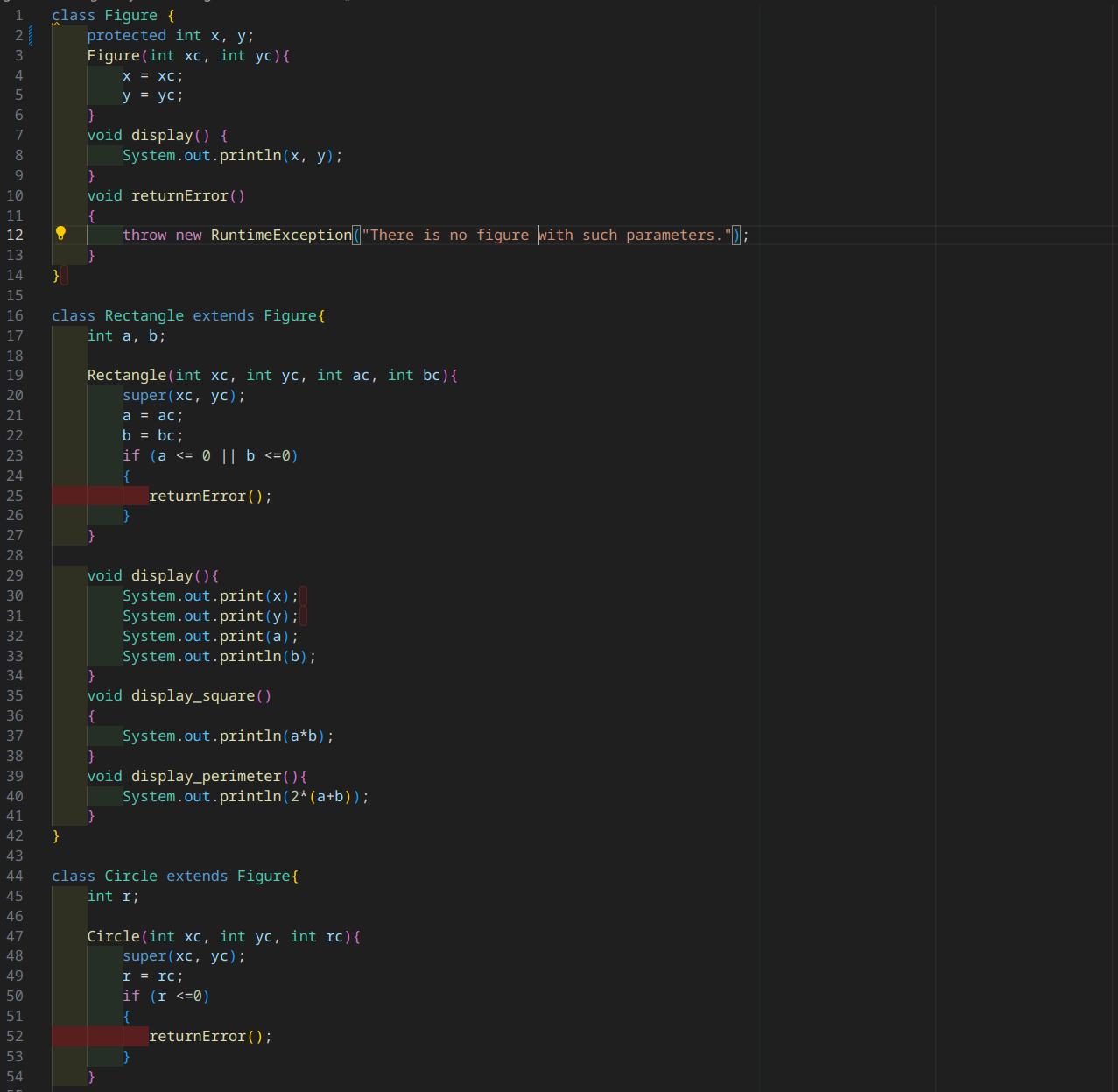
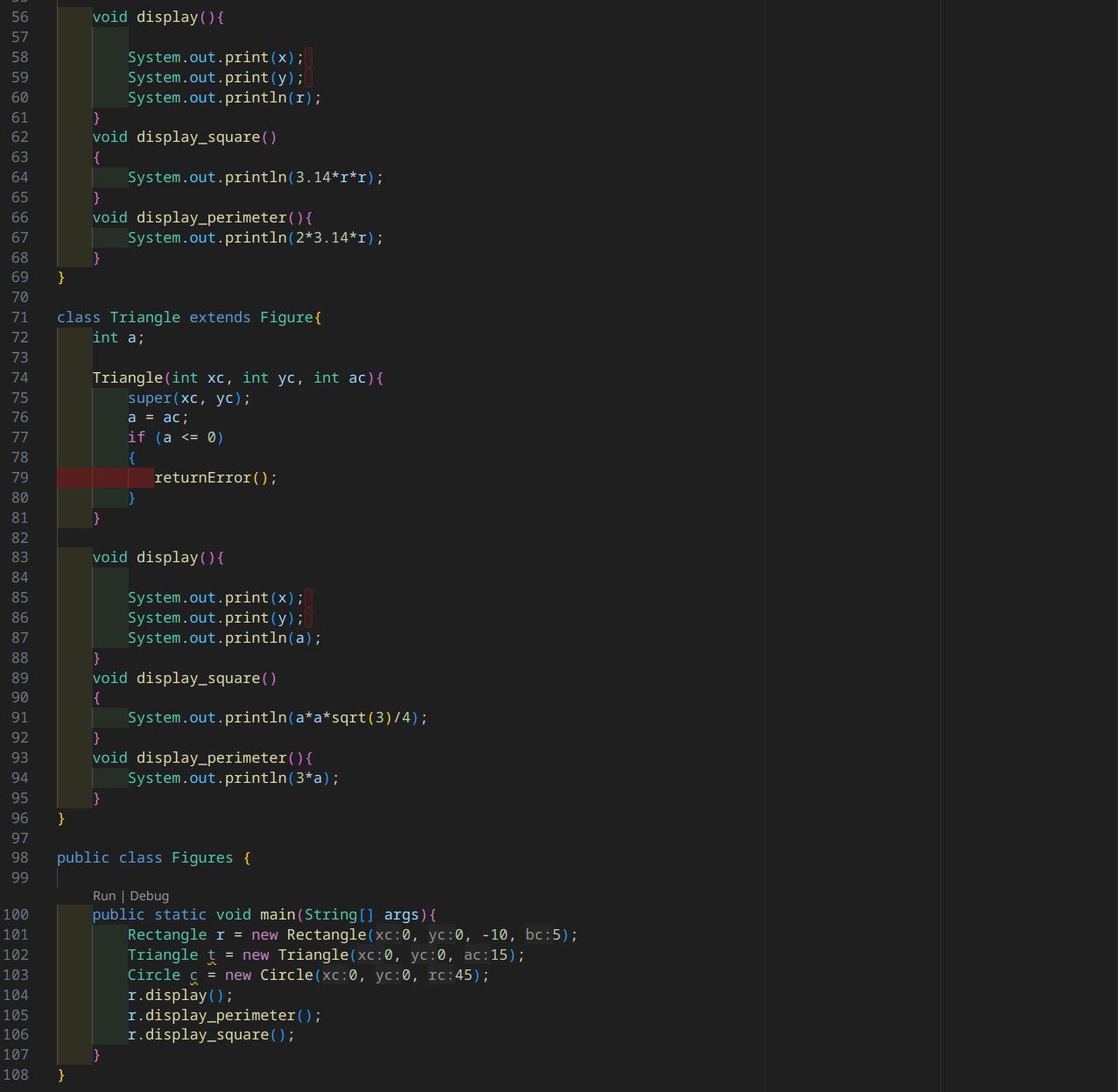
# Заключение

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.

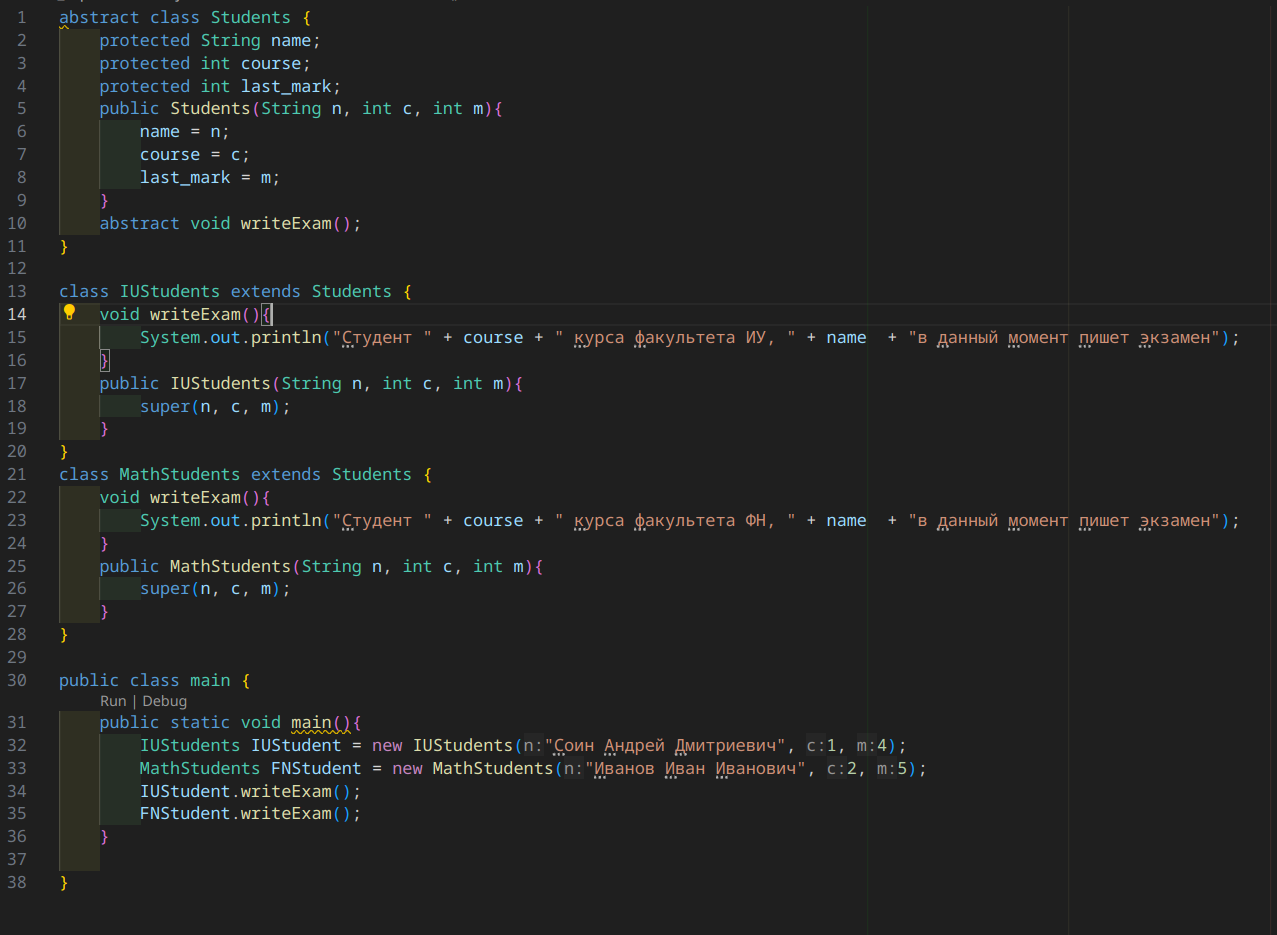
# Список литературы

* 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

