|  |  |
| --- | --- |
|  | **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 |  | 03/12/2024 |  | А.Д. Соин |
|  |  | (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 |  |
| Соин Андрей Дмитриевич | | |
| (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 | 21 | 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 |  | 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 вывод данных 9](#_Toc11)

[2.2.4. Массивы в Java 10](#_Toc12)

[2.2.5. Динамические массивы Java 11](#_Toc13)

[2.2.6. Ветвление Java 12](#_Toc14)

[2.2.7. Цикл while в Java 12](#_Toc15)

[2.2.8. Цикл for в Java 12](#_Toc16)

[2.2.9. Классы Java 12](#_Toc17)

[2.2.10. Модификаторы доступа Java 14](#_Toc18)

[2.2.11. Наследование в Java 14](#_Toc19)

[2.3. Создание программы для работы с фигурами. 15](#_Toc20)

[2.4. Создание программы для учета студентов 15](#_Toc21)

[2.5. Создание класса для работы с массивом строк 15](#_Toc22)

[Заключение 17](#_Toc23)

[Список литературы 18](#_Toc24)

[Приложение 1 19](#_Toc25)

[Приложение 2 20](#_Toc26)

[Приложение 3 21](#_Toc27)

# 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** — 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 репозитория**

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.

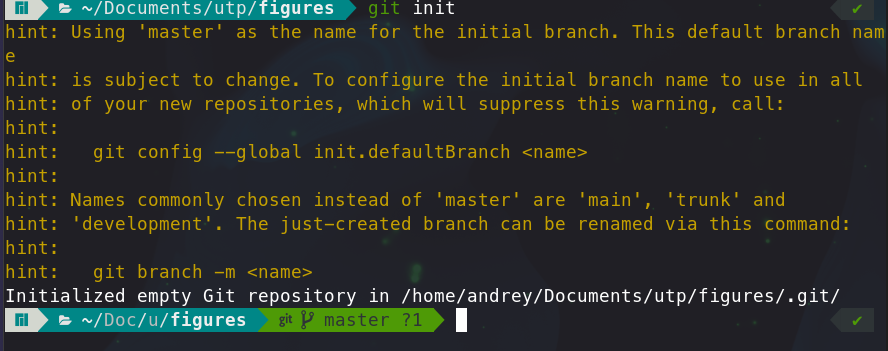


Рис. 1. Инициализация локального репозитория Git

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

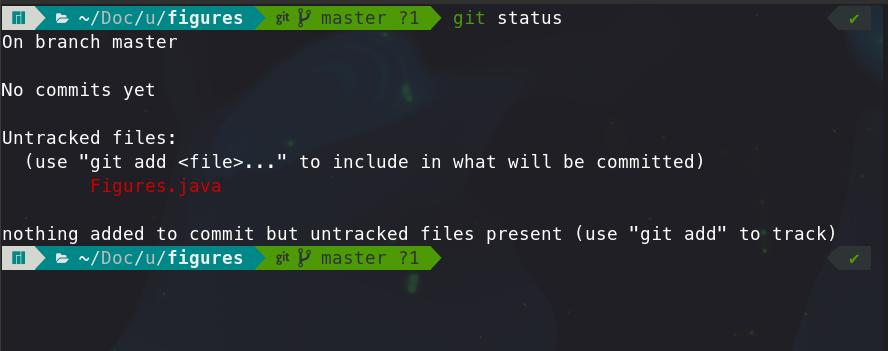


Рис. 2. Работа команды «git status»

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).

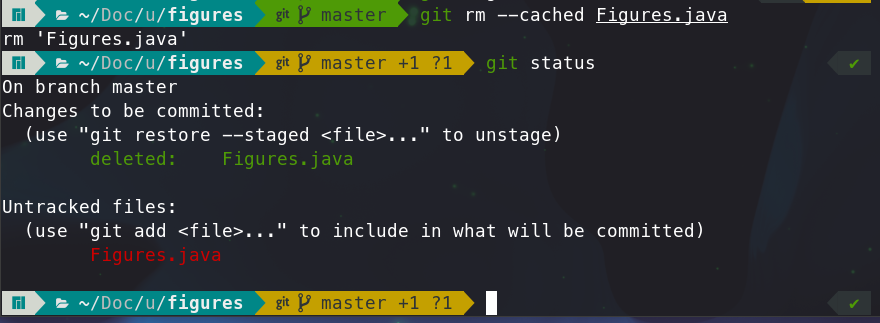


Рис. 5. Пример работы команды git rm --cached

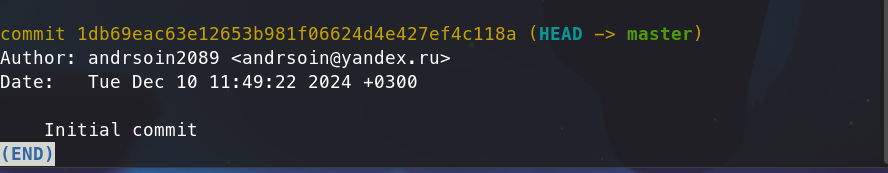


Рис. 6. Использование команды git log

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).

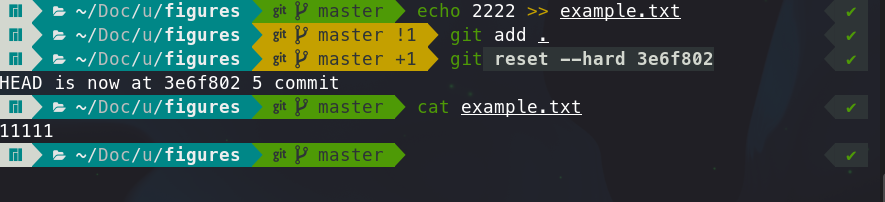


Рис. 8. Пример использования команды git reset

**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):

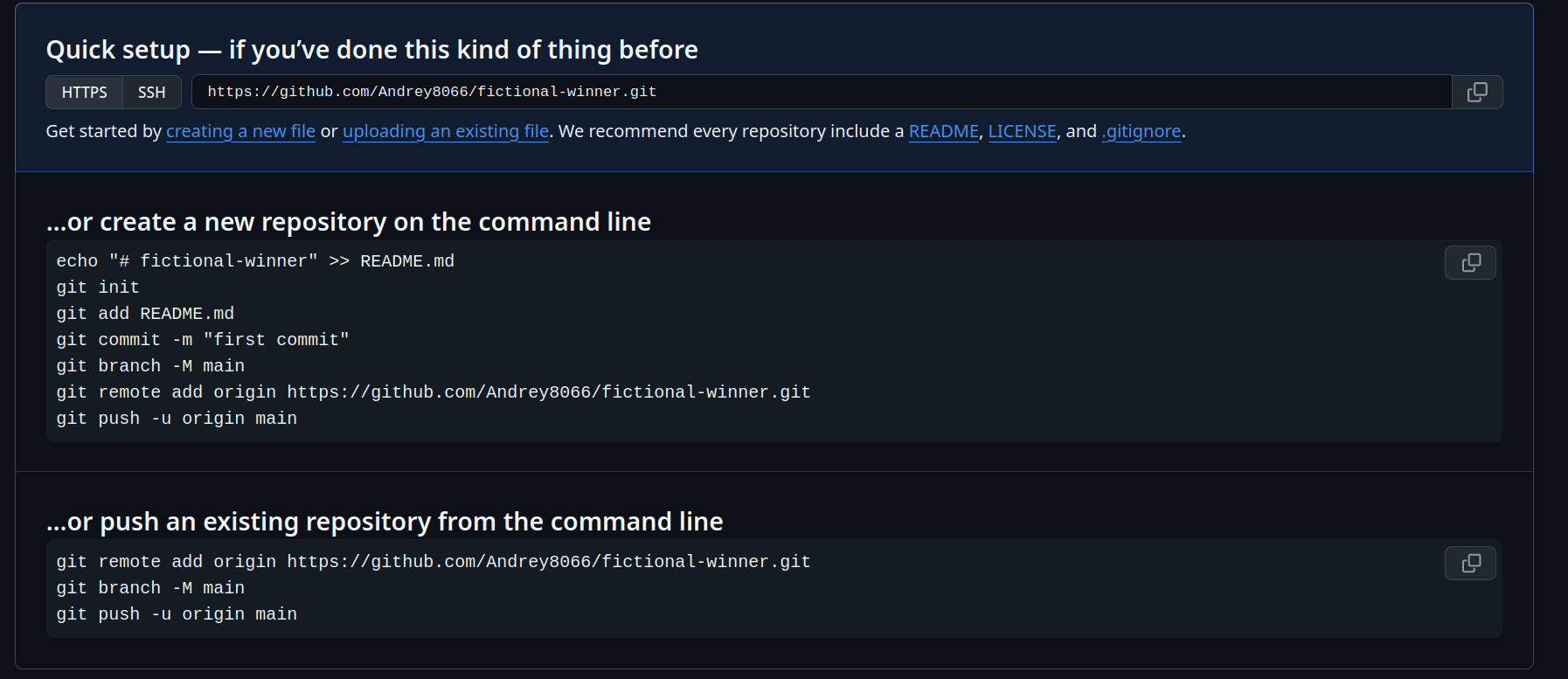


Рис. 9. Способы связки репозиториев

* «**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).

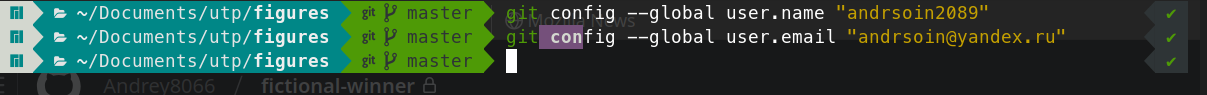


Рис. 10. Конфигурация GitHub

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

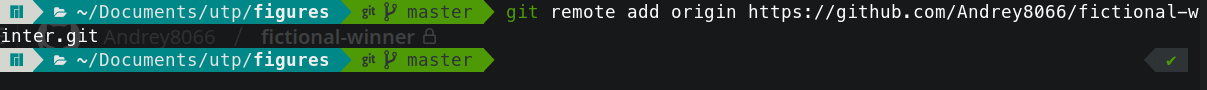


Рис. 11. Подключение удаленного репозитория к локальному

GitHub позволяет своим пользователям работать над проектом из разных мест и с разных, для переноса проекта с удаленного репозитория на свое устройство достаточно клонировать репозиторий с помощью команды «**git clone <ссылка>**» (Рис. 13).

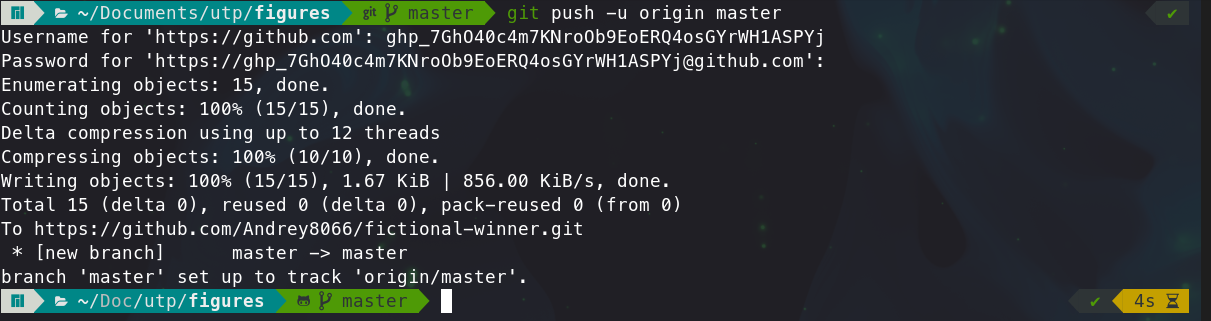


Рис. 12. Синхронизация коммитов

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

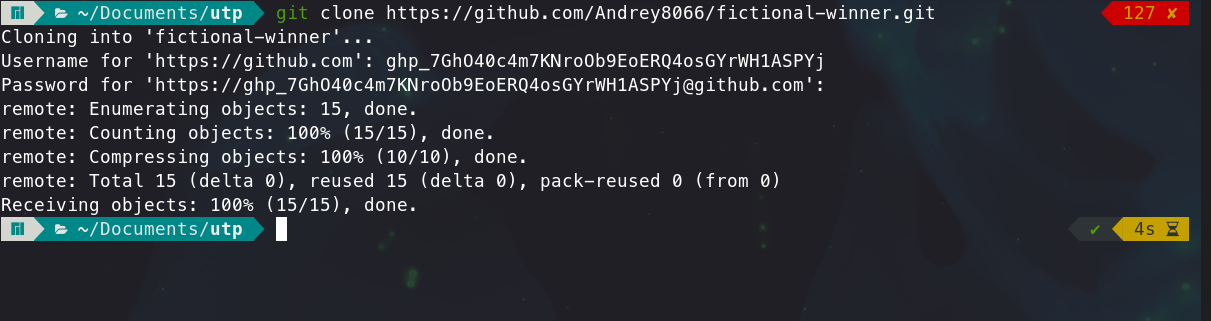


Рис. 13. Клонирование репозитория

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 используемый для предоставления информации о проекте или пакете. Его цель - помочь пользователям понять, как работать с данным проектом, а также передать другую информацию, которую разработчики считают важной. Он обычно располагается в корневом каталоге проекта.

Файл README обычно содержит:

* 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 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, что позволяет 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.

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):

Чтобы переключиться на другую ветку, выполните команду «git checkout <имя\_ветки>». Например, чтобы переключиться на ветку ‘new\_function’, необходимо выполнить следующую команду (Рис. 17):



Рис. 16. Создание новой ветки

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

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):

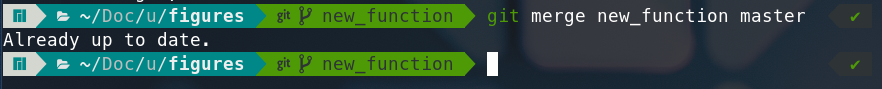


Рис. 18. Слияние веток

Преимущества использования веток:

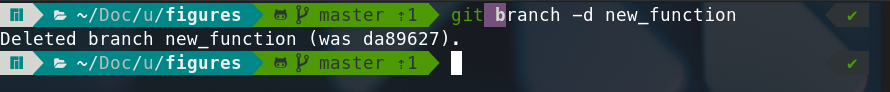


Рис. 19. Удаление ветки

* 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 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(Рис. 20).

Для доступа к элементу массива используйте индекс элемента в квадратных скобках(Рис. 21).

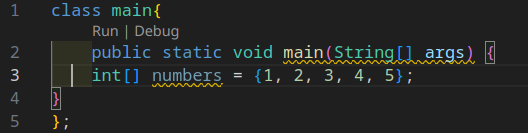


Рис. 20. Объявление массива

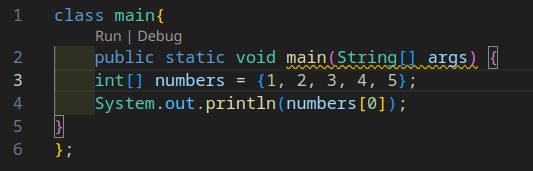


Рис. 21. Получение эллемента массива по индексу

Длина массива хранится в поле `length`(Рис. 22).

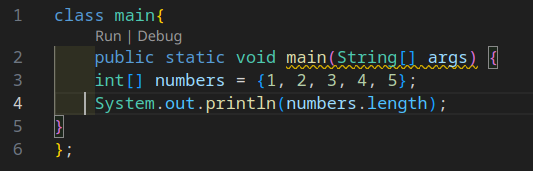


Рис. 22. Получение длинны массива

Java также поддерживает многомерные массивы (Рис. 23).

Особенности массивов в Java

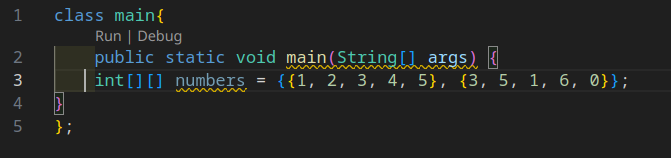


Рис. 23. Создание двумерного массива

* 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.

### Динамические массивы Java

Динамические массивы в Java – это структуры данных, которые позволяют хранить элементы в массиве, размер которого может изменяться во время выполнения программы. В Java для работы с динамическими массивами используют класс ArrayList.

Основные характеристики динамических массивов:

* Изменяемый размер: В отличие от обычных массивов в Java, динамические массивы могут увеличивать или уменьшать свой размер.
* Элементы: Динамические массивы могут хранить объекты любого типа,
* Производительность: Динамические массивы обеспечивают доступ к элементам по индексу за O(1) времени, но операции добавления или удаления могут занимать O(n) времени в худшем случае.

Основные методы класса ArrayList:

* add(E element): добавляет элемент в конец списка.
* add(int index, E element): вставляет элемент по указанному индексу.
* remove(int index): удаляет элемент по указанному индексу.
* get(int index): возвращает элемент по указанному индексу.
* size(): возвращает текущее количество элементов в списке.
* clear(): удаляет все элементы из списка.
* contains(Object o): проверяет, содержится ли элемент в списке.

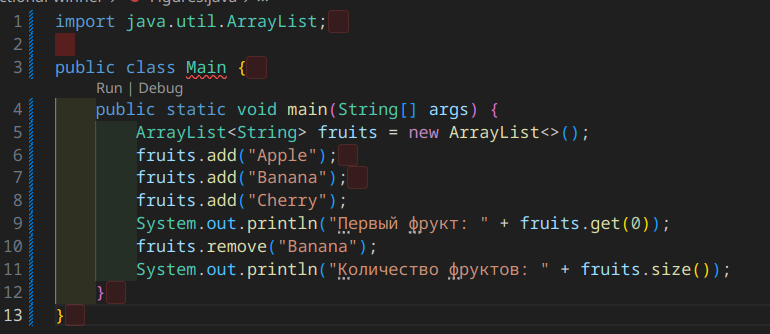


Рис. 24. Работа с динамическими массивами

### 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. 25).

### Цикл while в Java

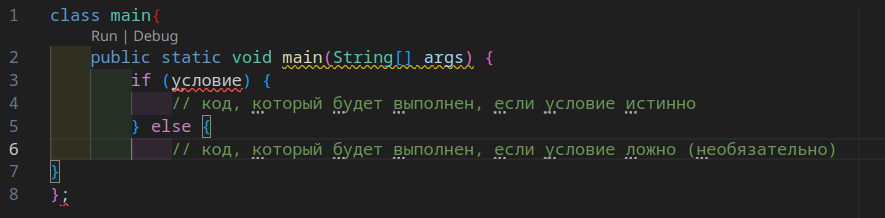


Рис. 25. Ветвление в java

Цикл `while` выполняет блок кода до тех пор, пока выполняется заданное условие (Рис. 26).

### Цикл for в Java

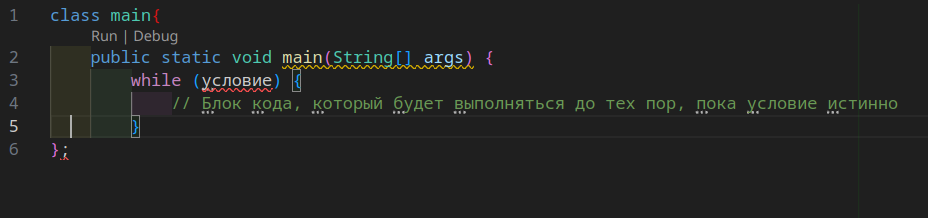


Рис. 26. Цикл while

Цикл `for` используется для итерации по последовательности значений. Он состоит из трех частей: инициализации, выражения и обновления (Рис. 27).

### Классы Java

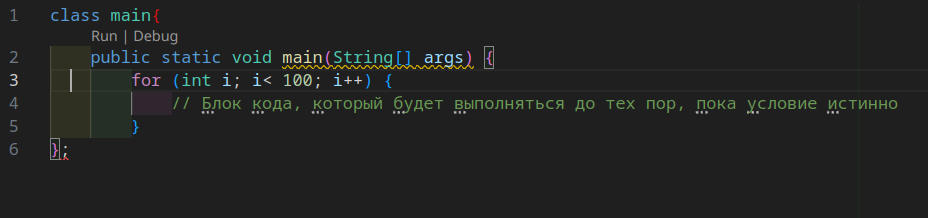


Рис. 27. Цикл for

Класс Java - это шаблон или план, используемый для создания объектов. Он содержит данные (переменные) и методы (функции), которые определяют поведение и характеристики объектов (Рис. 28).

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. 29).

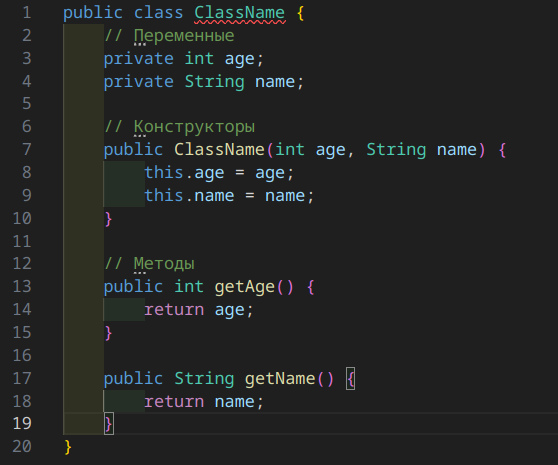


Рис. 28. Пример класса в Java

Чтобы вызвать метод класса нужно указать объект и метод класса к которому принадлежит данный объект с необходимыми аргументами (Рис. 30).

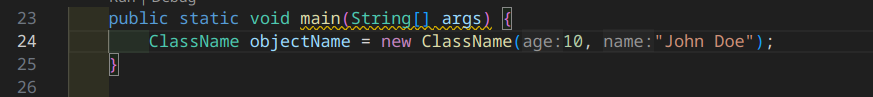


Рис. 29. Создание объекта класса в Java.

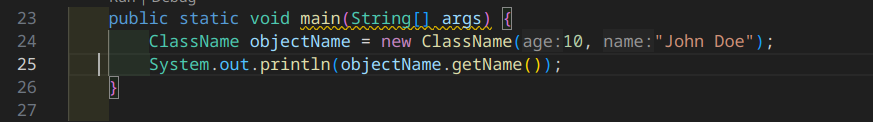


Рис. 30. Вызов метода класса

### 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), которые наследуют свойства и поведение от своих предшествующих классов (называемых родительскими или базовыми).

Преимущества наследования:

* Повторное использование кода
* Расширяемость
* Инкапсуляция
* Полиморфизм

Ключевые слова наследования:

* `extends` - используется для указания, что дочерний класс наследует от 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 наследует от нескольких родительских классов (не поддерживается в Java).
* Многоуровневое наследование: Дочерний класс наследует от другого дочернего класса, который, в свою очередь, наследует от родительского класса.

Доступ к членам классов:

* Private: Члены не доступны в дочерних классах.
* Protected: Члены доступны в дочерних классах и классах, находящихся в том же пакете.
* Default: Члены доступны в дочерних классах, находящихся в том же пакете.
* Public: Члены доступны везде.

Дочерние классы могут переопределять методы, наследуемые от родительских классов, предоставляя им собственную реализацию.

Дочерние классы должны вызывать суперконструктор родительского класса с помощью ключевого слова `super` в своем конструкторе.

## 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 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](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 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](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).

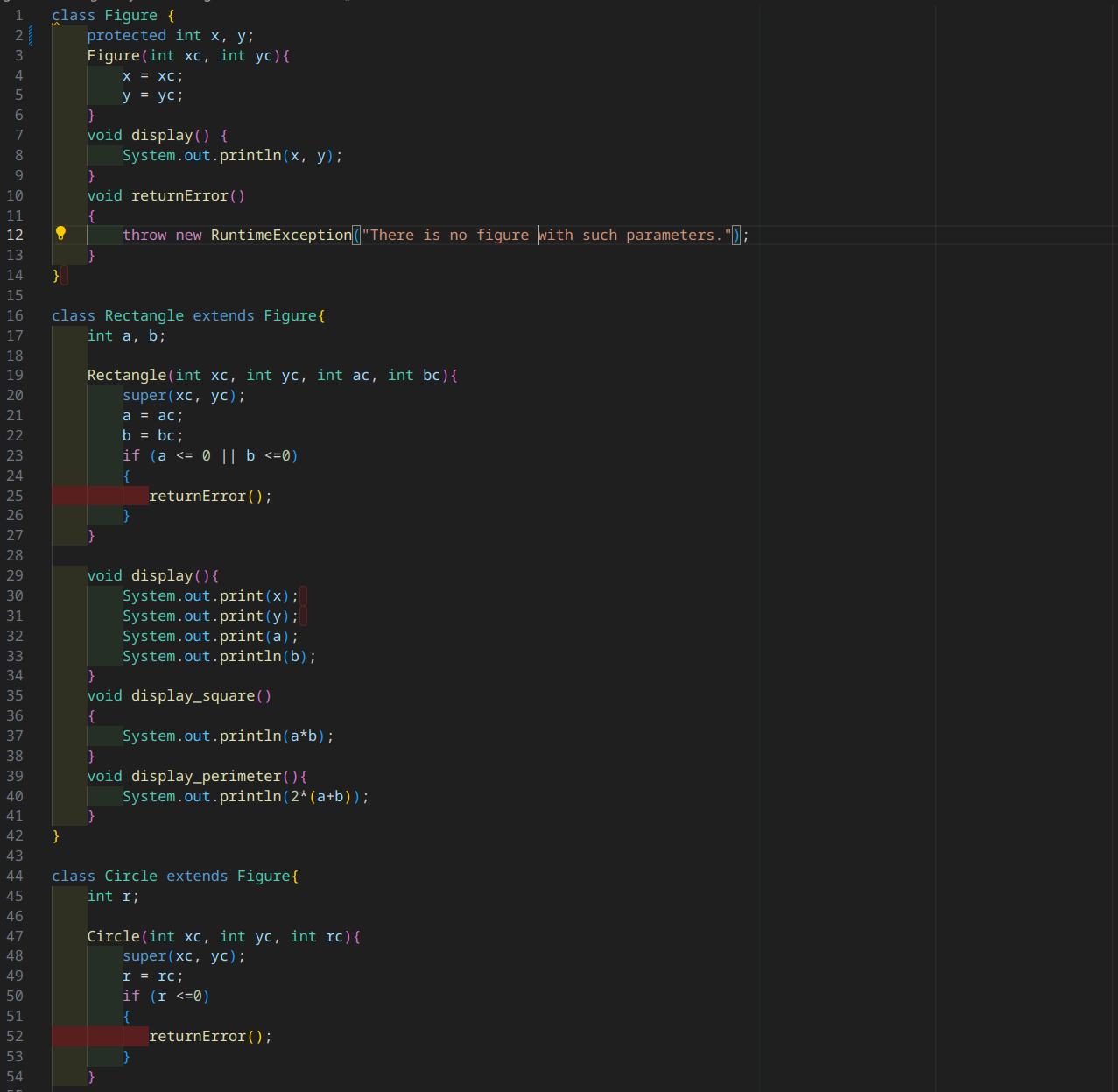
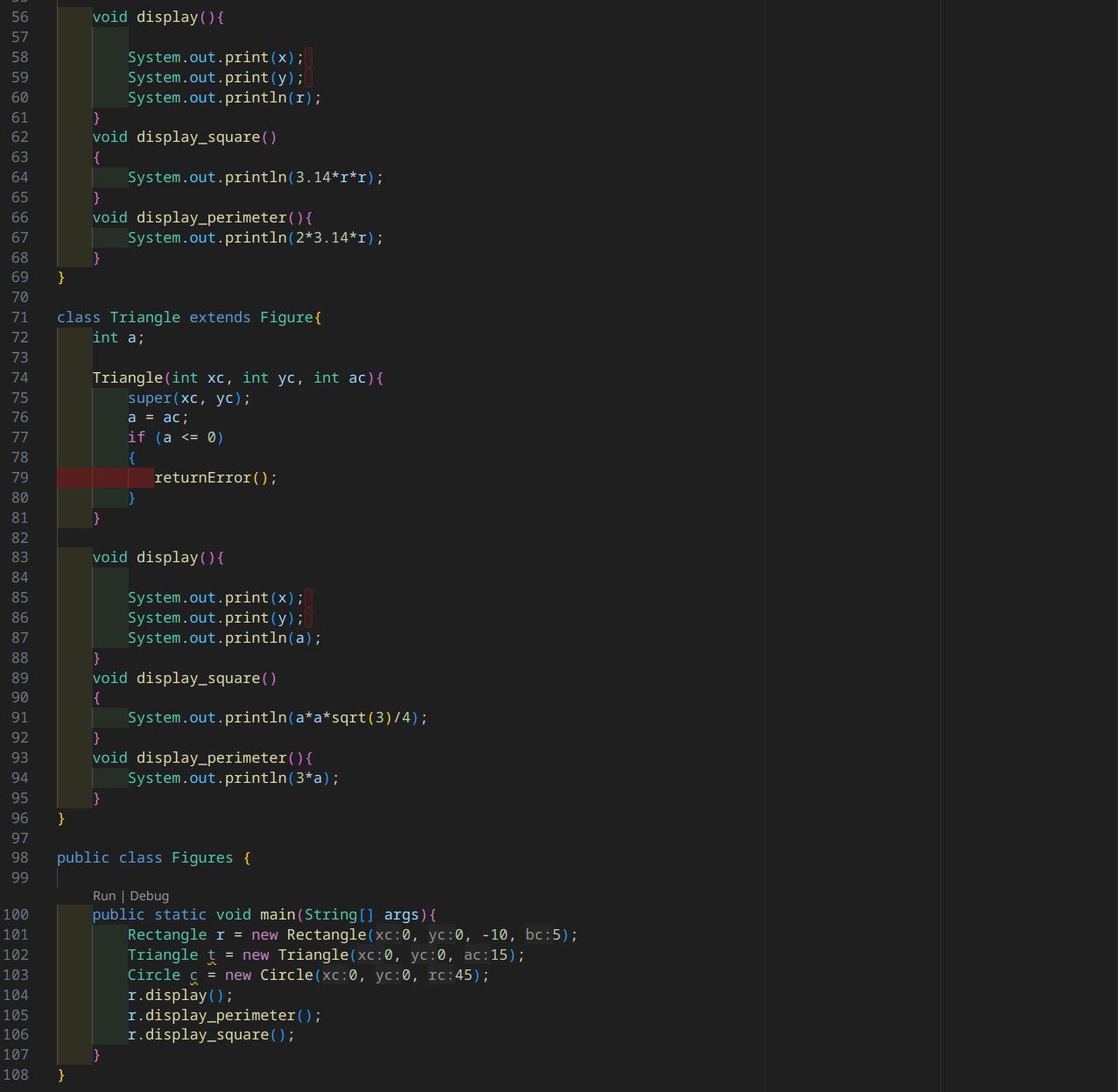
# Заключение

В отчете описывается базовый функционал Git и GitHub для работы с локальными и удаленными репозиториями , а также язык 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.

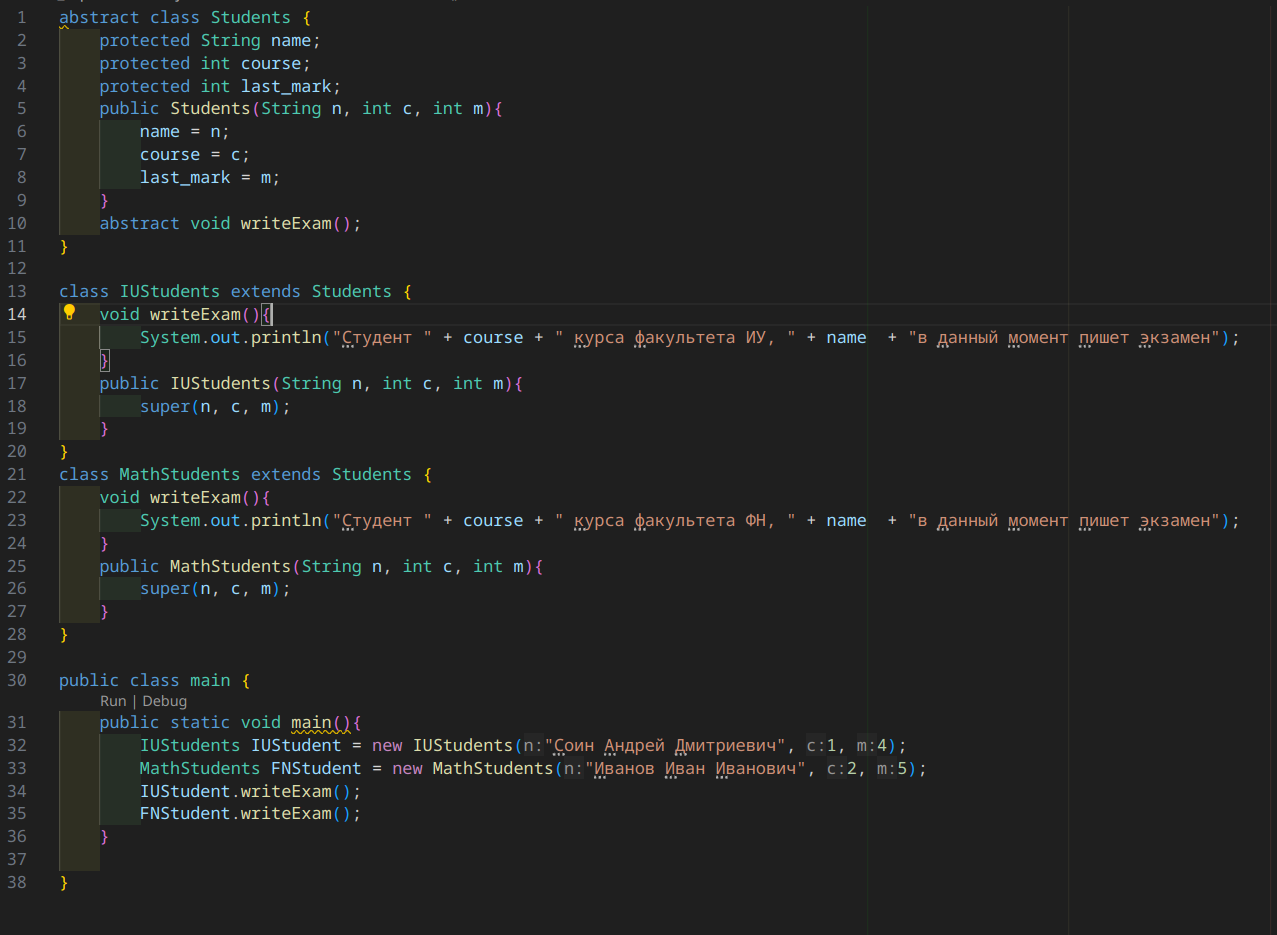
# Список литературы

* 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

