



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.

З А Д А Н И Е

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
(Подпись, дата)

А.А. Николаев
(И.О. Фамилия)

Студент

24/09/2024
(Подпись, дата)

А.Д. Соин
(И.О. Фамилия)

СОДЕРЖАНИЕ

| | |
|---|----|
| СОДЕРЖАНИЕ | 1 |
| ВВЕДЕНИЕ | 2 |
| Тема 1. Git и Github | 3 |
| 1.1. Общие сведения о Git и Github | 3 |
| 1.2. Работа с Github | 3 |
| Тема 2. Java | 8 |
| 2.1. Общие сведения о языке программирования Java | 8 |
| 2.2. Устройство языка Java | 8 |
| 2.2.1. Типы данных в Java | 8 |
| 2.2.2. Операции с данными | 9 |
| 2.2.3. Ввод и вывод данных | 9 |
| 2.2.4. Массивы в Java | 10 |
| 2.2.5. Динамические массивы Java | 11 |
| 2.2.6. Ветвление Java | 12 |
| 2.2.7. Цикл while в Java | 12 |
| 2.2.8. Цикл for в Java | 12 |
| 2.2.9. Классы Java | 12 |
| 2.2.10. Модификаторы доступа Java | 14 |
| 2.2.11. Наследование в Java | 14 |
| 2.3. Создание программы для работы с фигурами | 15 |
| 2.4. Создание программы для учета студентов | 15 |
| 2.5. Создание класса для работы с массивом строк | 15 |
| Заключение | 17 |
| Список литературы | 18 |
| Приложение 1 | 19 |
| Приложение 2 | 20 |
| Приложение 3 | 21 |

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, минимизирует 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.

Тема 1. Git и Github

1.1. Общие сведения о Git и Github.

Git — распределённая система управления версиями. Проект был создан Линусом Торвалдсом 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.

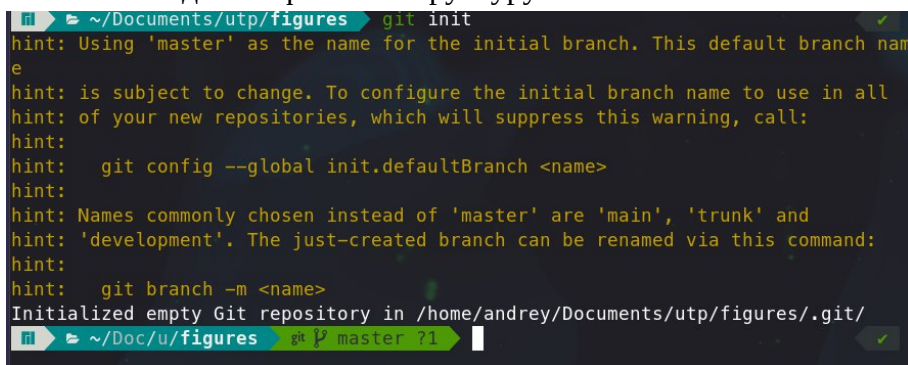
1.2. Работа с Github.

Создание репозитория

Перед началом работы с github необходимо создать репозиторий (репозиторий – каталог, хранящий файлы 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.



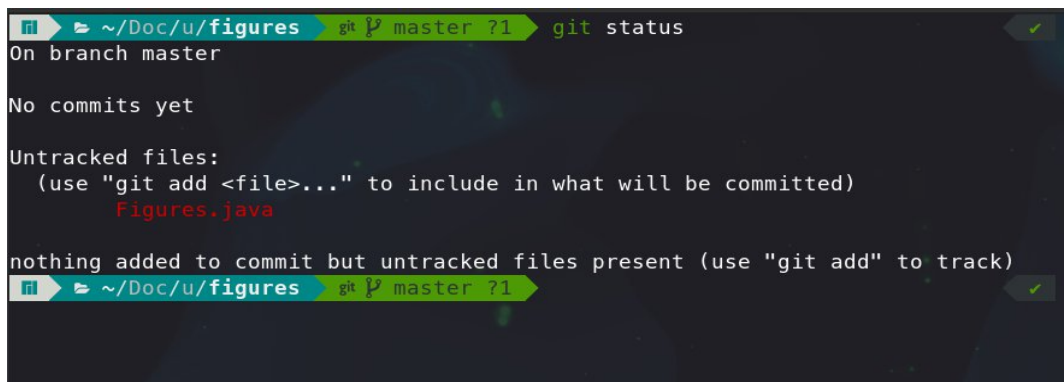
```

~/Documents/utp/figures$ git init
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:     git config --global init.defaultBranch <name>
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:     git branch -m <name>
Initialized empty Git repository in /home/andrey/Documents/utp/figures/.git/
~/Documents/utp/figures$ git status

```

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).



```

~/Documents/utp/figures$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    Figures.java

nothing added to commit but untracked files present (use "git add" to track)
~/Documents/utp/figures$

```

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).

```
~/Doc/u/figures git master ?1 git add .
```

Рис. 3. Использование команды «git add .»

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

```
~/Doc/u/figures git master +1 git commit -m "Initial commit"
[master (root-commit) 1db69ea] Initial commit
1 file changed, 108 insertions(+)
create mode 100644 Figures.java
```

Рис. 4. Использование команды git commit

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

```
commit 1db69eac63e12653b981f06624d4e427ef4c118a (HEAD -> master)
Author: andrsoin2089 <andrsoin@yandex.ru>
Date: Tue Dec 10 11:49:22 2024 +0300

    Initial commit
(END)
```

Рис. 6. Использование команды git log

Для того чтобы перестать отслеживать файл необходимо использовать команду «**git rm --cached**», она не удалит файл из каталога, но git перестанет отслеживать изменения в нем (Рис. 6).

```
~/Doc/u/figures git master git rm --cached Figures.java
rm 'Figures.java'
~/Doc/u/figures git master +1 ?1 git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        deleted:    Figures.java

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        Figures.java
```

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

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

```
~/Doc/u/figures git master ?1 git revert 652a18f
```

Рис. 7. Пример использования команды git revert

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

```

~/Doc/u/figures git P master echo 2222 >> example.txt
~/Doc/u/figures git P master !1 git add .
~/Doc/u/figures git P master +1 git reset --hard 3e6f802
HEAD is now at 3e6f802 5 commit
~/Doc/u/figures git P master cat example.txt
11111
~/Doc/u/figures git P master

```

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 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).

```

Quick setup — if you've done this kind of thing before
HTTPS SSH https://github.com/Andrey8066/fictional-winner.git
Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.

...or create a new repository on the command line
echo "# fictional-winner" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/Andrey8066/fictional-winner.git
git push -u origin main

...or push an existing repository from the command line
git remote add origin https://github.com/Andrey8066/fictional-winner.git
git branch -M main
git push -u origin main

```

Рис. 9. Способы связки репозиториев

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

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

```

~/Documents/utp/figures git P master git config --global user.name "andrsoin2089"
~/Documents/utp/figures git P master git config --global user.email "andrsoin@yandex.ru"
~/Documents/utp/figures git P master

```

Рис. 10. Конфигурация GitHub

Чтобы связать локальный репозиторий с только что созданным репозиторием на GitHub используется команда «git remote add origin <link>» (Рис. 11).

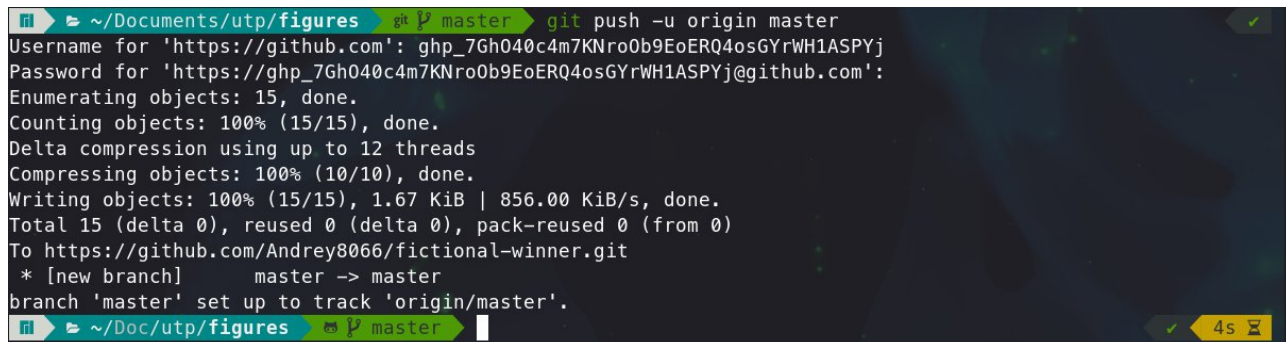
```

~/Documents/utp/figures git P master git remote add origin https://github.com/Andrey8066/fictional-w
inter.git
~/Documents/utp/figures git P master

```

Рис. 11. Подключение удаленного репозитория к локальному

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



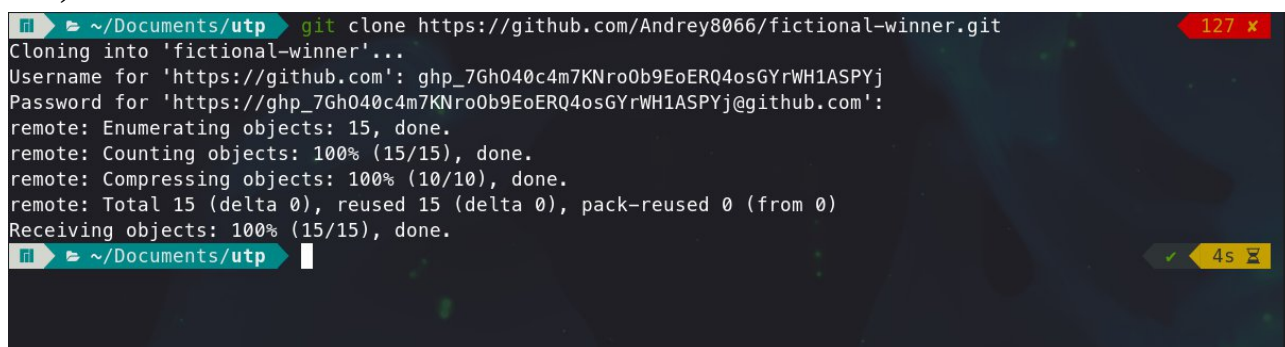
```

~/Documents/utp/figures git P master git push -u origin master
Username for 'https://github.com': ghp_7Gh040c4m7KNro0b9EoERQ4osGYrWH1ASPYj
Password for 'https://ghp_7Gh040c4m7KNro0b9EoERQ4osGYrWH1ASPYj@github.com':
Enumerating objects: 15, done.
Counting objects: 100% (15/15), done.
Delta compression using up to 12 threads
Compressing objects: 100% (10/10), done.
Writing objects: 100% (15/15), 1.67 KiB | 856.00 KiB/s, done.
Total 15 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/Andrey8066/fictional-winner.git
 * [new branch]      master -> master
branch 'master' set up to track 'origin/master'.
~/Doc/utp/figures git P master

```

Рис. 12. Синхронизация коммитов

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



```

~/Documents/utp git clone https://github.com/Andrey8066/fictional-winner.git
Cloning into 'fictional-winner'...
Username for 'https://github.com': ghp_7Gh040c4m7KNro0b9EoERQ4osGYrWH1ASPYj
Password for 'https://ghp_7Gh040c4m7KNro0b9EoERQ4osGYrWH1ASPYj@github.com':
remote: Enumerating objects: 15, done.
remote: Counting objects: 100% (15/15), done.
remote: Compressing objects: 100% (10/10), done.
remote: Total 15 (delta 0), reused 15 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (15/15), done.
~/Documents/utp

```

Рис. 13. Клонирование репозитория

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



```

~/Documents/utp git pull

```

Рис. 14. Перенос изменений с глобального репозитория на локальный

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



```

~/Documents/utp git push -u origin master

```

Рис. 15. Команда "git push"

Файлы «.gitignore» и «readme.md»

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

Содержание файла README обычно включает:

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

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

- Бинарные файлы (например, изображения, видео)

- Скомпилированные файлы (например, исполняемые файлы)
- Конфиденциальные данные (например, пароли, ключи API)
- Кэшированные файлы
- Установочные файлы

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

Общие правила в файле .gitignore:

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

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):



Рис. 17. Переключение на другую ветку

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):

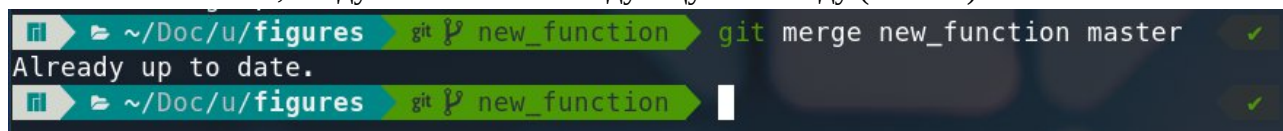


Рис. 18. Слияние веток

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):

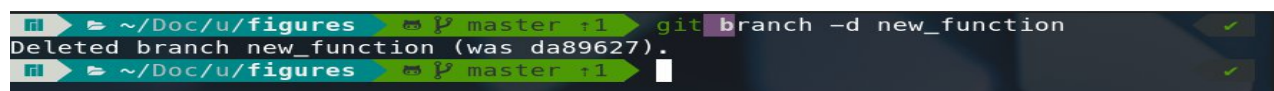


Рис. 19. Удаление ветки

Преимущества использования веток:

- 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.

Тема 2. Java

2.1. 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.

2.2. Устройство языка Java

2.2.1. 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.

2.2.2. 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

2.2.3. 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:

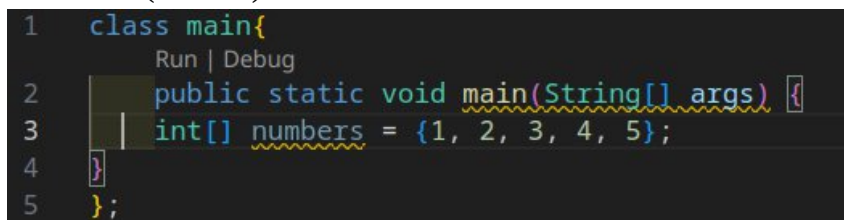
- `System.out.println()` Выводит данные с новой строкой
- `System.out.print()` Выводит данные без новой строки
- `PrintWriter.println()` Выводит данные с новой строкой
- `PrintWriter.print()` Выводит данные без новой строки

2.2.4. 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).

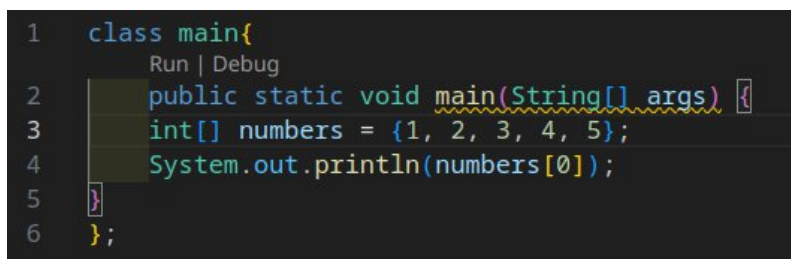


```

1  class main{
    Run | Debug
2  public static void main(String[] args) {
3  |   int[] numbers = {1, 2, 3, 4, 5};
4  |
5  | }
   
```

Рис. 20. Объявление массива

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

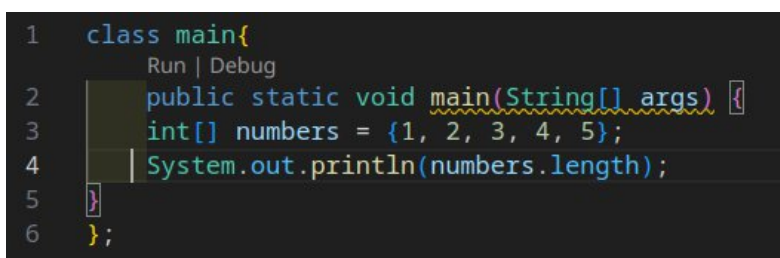


```

1  class main{
    Run | Debug
2  public static void main(String[] args) {
3  |   int[] numbers = {1, 2, 3, 4, 5};
4  |   System.out.println(numbers[0]);
5  |
6  | }
   
```

Рис. 21. Получение элемента массива по индексу

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

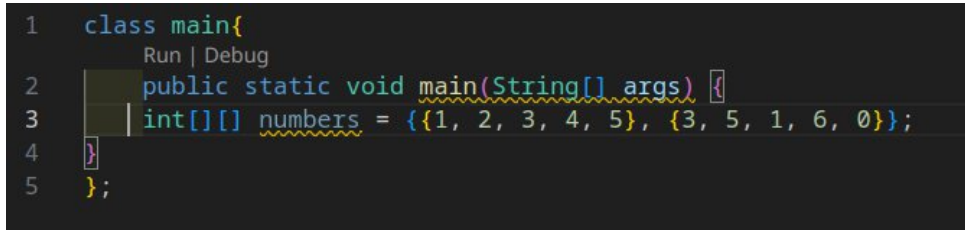


```

1  class main{
    Run | Debug
2  public static void main(String[] args) {
3  |   int[] numbers = {1, 2, 3, 4, 5};
4  |   System.out.println(numbers.length);
5  |
6  | }
   
```

Рис. 22. Получение длины массива

Java также поддерживает многомерные массивы (Рис. 23).



```

1  class main{
    Run | Debug
2  public static void main(String[] args) {
3  int[][] numbers = {{1, 2, 3, 4, 5}, {3, 5, 1, 6, 0}};
4  }
5  };

```

Рис. 23. Создание двумерного массива

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.

2.2.5. Динамические массивы Java

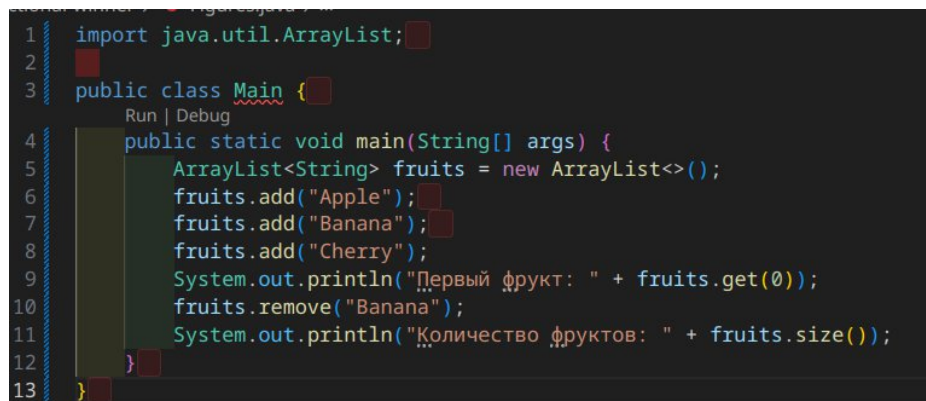
Динамические массивы в Java – это структуры данных, которые позволяют хранить элементы в массиве, размер которого может изменяться во время выполнения программы. В Java для работы с динамическими массивами используют класс ArrayList.

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.



```

1  import java.util.ArrayList;
2
3  public class Main {
    Run | Debug
4  public static void main(String[] args) {
5      ArrayList<String> fruits = new ArrayList<>();
6      fruits.add("Apple");
7      fruits.add("Banana");
8      fruits.add("Cherry");
9      System.out.println("Первый фрукт: " + fruits.get(0));
10     fruits.remove("Banana");
11     System.out.println("Количество фруктов: " + fruits.size());
12 }
13 }

```

Рис. 24. Работа с динамическими массивами

2.2.6. 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).

```

1  class main{
    Run | Debug
2      public static void main(String[] args) {
3          if (условие) {
4              // код, который будет выполнен, если условие истинно
5          } else {
6              // код, который будет выполнен, если условие ложно (необязательно)
7          }
8      };

```

Рис. 25. Ветвление в java

2.2.7. Цикл while в Java

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

```

1  class main{
    Run | Debug
2      public static void main(String[] args) {
3          while (условие) {
4              // Блок кода, который будет выполняться до тех пор, пока условие истинно
5          }
6      };

```

Рис. 26. Цикл while

2.2.8. Цикл for в Java

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

```

1  class main{
    Run | Debug
2      public static void main(String[] args) {
3          for (int i; i < 100; i++) {
4              // Блок кода, который будет выполняться до тех пор, пока условие истинно
5          }
6      };

```

Рис. 27. Цикл for

2.2.9. 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. 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.

```

1  public class ClassName {
2      // Переменные
3      private int age;
4      private String name;
5
6      // Конструкторы
7      public ClassName(int age, String name) {
8          this.age = age;
9          this.name = name;
10     }
11
12     // Методы
13     public int getAge() {
14         return age;
15     }
16
17     public String getName() {
18         return name;
19     }
20 }

```

Рис. 28. Пример класса в Java

Для того чтобы работать с классом необходимо создать объект данного класса. Для этого в Java используется метод 'new' (Рис. 29).

```

23  public static void main(String[] args) {
24      ClassName objectName = new ClassName(age:10, name:"John Doe");
25  }
26

```

Рис. 29. Создание объекта класса в Java.

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

```

23  public static void main(String[] args) {
24      ClassName objectName = new ClassName(age:10, name:"John Doe");
25      System.out.println(objectName.getName());
26  }
27

```

Рис. 30. Вызов метода класса

2.2.10. 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.

2.2.11. 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.

2.3. Создание программы для работы с фигурами.

Задание:

Реализовать классы, которые будут 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 сообщаемый 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 `display_square`, выводящий площадь фигуры, и метод `display_perimeter`, выводящий периметр фигуры.

Разработанные классы и их методы были протестированы.

Исходный код можно увидеть в репозитории [github](#) или в [Приложении 1](#).

2.4. 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 `IUStudents` 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](#).

2.5. 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, возвращения 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 длинную 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](#).

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

- 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

```

1 class Figure {
2     protected int x, y;
3     Figure(int xc, int yc){
4         x = xc;
5         y = yc;
6     }
7     void display() {
8         System.out.println(x, y);
9     }
10    void returnError()
11    {
12        throw new RuntimeException("There is no figure with such parameters.");
13    }
14 }
15
16 class Rectangle extends Figure{
17     int a, b;
18
19     Rectangle(int xc, int yc, int ac, int bc){
20         super(xc, yc);
21         a = ac;
22         b = bc;
23         if (a <= 0 || b <= 0)
24         {
25             returnError();
26         }
27     }
28
29     void display(){
30         System.out.print(x);
31         System.out.print(y);
32         System.out.print(a);
33         System.out.println(b);
34     }
35     void display_square()
36     {
37         System.out.println(a*b);
38     }
39     void display_perimeter(){
40         System.out.println(2*(a+b));
41     }
42 }
43
44 class Circle extends Figure{
45     int r;
46
47     Circle(int xc, int yc, int rc){
48         super(xc, yc);
49         r = rc;
50         if (r <= 0)
51         {
52             returnError();
53         }
54 }

```

```

56     void display(){
57
58         System.out.print(x);
59         System.out.print(y);
60         System.out.println(r);
61     }
62     void display_square()
63     {
64         System.out.println(3.14*r*r);
65     }
66     void display_perimeter(){
67         System.out.println(2*3.14*r);
68     }
69 }
70
71 class Triangle extends Figure{
72     int a;
73
74     Triangle(int xc, int yc, int ac){
75         super(xc, yc);
76         a = ac;
77         if (a <= 0)
78         {
79             returnError();
80         }
81     }
82
83     void display(){
84
85         System.out.print(x);
86         System.out.print(y);
87         System.out.println(a);
88     }
89     void display_square()
90     {
91         System.out.println(a*a*sqrt(3)/4);
92     }
93     void display_perimeter(){
94         System.out.println(3*a);
95     }
96 }
97
98 public class Figures {
99
100     Run | Debug
101     public static void main(String[] args){
102         Rectangle r = new Rectangle(xc:0, yc:0, -10, bc:5);
103         Triangle t = new Triangle(xc:0, yc:0, ac:15);
104         Circle c = new Circle(xc:0, yc:0, rc:45);
105         r.display();
106         r.display_perimeter();
107         r.display_square();
108     }

```

Приложение 2

```

1  abstract class Students {
2      protected String name;
3      protected int course;
4      protected int last_mark;
5      public Students(String n, int c, int m){
6          name = n;
7          course = c;
8          last_mark = m;
9      }
10     abstract void writeExam();
11 }
12
13 class IUStudents extends Students {
14     void writeExam(){
15         System.out.println("Студент " + course + " курса факультета ИУ, " + name + " в данный момент пишет экзамен");
16     }
17     public IUStudents(String n, int c, int m){
18         super(n, c, m);
19     }
20 }
21 class MathStudents extends Students {
22     void writeExam(){
23         System.out.println("Студент " + course + " курса факультета ФН, " + name + " в данный момент пишет экзамен");
24     }
25     public MathStudents(String n, int c, int m){
26         super(n, c, m);
27     }
28 }
29
30 public class main {
31     Run | Debug
32     public static void main(){
33         IUStudents IUStudent = new IUStudents("Соин Андрей Дмитриевич", c:1, m:4);
34         MathStudents FNStudent = new MathStudents("Иванов Иван Иванович", c:2, m:5);
35         IUStudent.writeExam();
36         FNStudent.writeExam();
37     }
38 }

```

Приложение 3

```

1  import java.util.ArrayList;
2
3  class String_container {
4      public ArrayList<String> container = new ArrayList<String>();
5
6      public void addString (String s){
7          if (container.size() < 1) {
8              container.add(s);
9          }
10         else if (s.length() >= container.getLast().length())
11         {
12             container.addLast(s);
13         }
14         else {
15             for (int i = container.size(); i>0; i--)
16             {
17                 if (container.get(i).length() >= s.length())
18                 {
19                     container.add(i, s);
20                 }
21             }
22         }
23     }
24     public String getLongest(){
25         String longest = container.get(0);
26         for (int i = 0; i < container.size(); i++){
27             if (longest.length() < container.get(i).length()){
28                 longest = container.get(i);
29             }
30         }
31         return longest;
32     }
33     public float getAvaerageLength(){
34         float length = 0;
35         for (int i = 0; i < container.size(); i++){
36             length += container.get(i).length();
37         }
38         return length/container.size();
39     }
40 }
41
42 public class main {
43
44     Run | Debug
45     public static void main(String[] args) {
46         String_container sc = new String_container();
47         sc.addString(s:"Арбуз");
48         sc.addString(s:"Лимон");
49         sc.addString(s:"Яблоко");
50         System.out.println(sc.getAvaerageLength());
51         System.err.println(sc.getLongest());
52         System.out.println(sc.container.get(2));
53     }

```