

Оглавление

1. Использование Git-репозитория	3
1.1 Используемые термины.....	3
1.2 Установка и настройка Git.....	4
1.3 Фиксация изменений (коммиты)	5
1.4 Ветки проекта	6
1.5 Задание: Выполнение локального рабочего процесса Git через командную строку	6
2. Использование Maven для разработки Java приложений	15
2.1 Что такое Apache Maven?	15
2.2 Установка инструментов командной строки Maven.....	16
2.3 Запуск сборки Maven	17
2.5 Задание: Создать и собрать проект Java с использованием Maven.....	21
2.6 Задание: Запустить Java-программу с использованием Maven	24
2.7 Конфигурация и координаты проекта Maven	26
2.8 Плагины Maven, цели и жизненный цикл	27
2.9 Репозитории Maven и разрешение зависимостей	30
2.10 Многомодульные проекты (Агрегатор)	31
2.11 Использование профилей и свойств в Maven.....	31
2.12 Maven и система контроля версий	33
2.13 Настройки Maven	33
2.14 Полезные параметры Maven	35
2.15 Плагины Maven для анализа проекта	35
3. Использование Maven с Eclipse IDE	37
3.2 Установка и настройка Maven для Eclipse.....	38
3.3 Задание: Создать новый проект с поддержкой Maven в Eclipse	40
3.4 Задание: Добавить поддержку Maven в java-проект в Eclipse.....	44
3.5 Задание: Создать веб-проект на Java в Eclipse с использованием Maven	45
4. Тестирование программного обеспечения	47
4.1 Цель тестов ПО.....	47
4.2 Используемые термины.....	47
4.3 Использование JUnit.....	49

4.4 Использование JUnit 4.....	52
4.5 Поддержка Eclipse для JUnit 4	61
4.6 Установка JUnit.....	64
4.7 Настройка Eclipse для использования статического импорта	65
4.8 Задание: Создать проект и протестировать его с использованием JUnit.....	66
4.9 Объект Mocking	70
4.10 Обзор JUnit 5	70
5. Лабораторные работы	83
Лабораторная работа №1: Основы синтаксиса Java.....	83
Лабораторная работа №2: Основы объектно-ориентированного программирования	87
Лабораторная работа №3: Алгоритм A* («A star»)	91
Лабораторная работа № 4: Рисование фракталов.....	98
Лабораторная работа №5. Выбор и сохранения фракталов	108
Лабораторная работа №6. Многопоточный генератор фракталов	115
Лабораторная работа №7. Веб-сканер.....	121
Лабораторная работа №8: Модифицированный веб-сканер.....	131

1. Использование Git-репозитория

Система контроля версий записывает изменения в файл или набор файлов и позволяет вернуться к определенной версии. Например, вы можете вернуть набор файлов в состояние, в котором они были два дня назад.

Система контроля версий обычно используется для отслеживания изменений в текстовых файлах. Такими файлами могут быть, например, исходный код на языке программирования HTML или конфигурационные файлы. Система контроля версий не ограничивается только текстовыми файлами, она также может обрабатывать файлы других типов, например, она может отслеживать изменения в файлах формата png.

В настоящее время Git является наиболее популярной системой контроля версий. Она используется многими популярными проектами с открытым исходным кодом, командами разработчиков Android или Eclipse, а также многими коммерческими организациями.

1.1 Используемые термины

Репозиторий – виртуальное хранилище версий проекта. В репозитории Git хранятся файлы, ветки, тэги, версии и коммиты.

- Коммит – зафиксированное состояние проекта, “единица” версии. Все последующие коммиты должны наследоваться от предыдущего, тем самым образуя древовидную структуру. Репозиторий Git является распределенной системой управления версиями, то есть у каждого разработчика одного проекта есть своя копия репозитория. Эта копия находится в папке “.git”, которая расположена в корне проекта.

- Ветка – именованная ссылка на определенный коммит, указывающая на конец ответвления от главной ветки. То есть ветка начинается от последнего коммита. А заканчивается ветка merge коммитом, который обозначает, что необходимо объединить ветки.

Инициализированный репозиторий содержит только одну ветку, которая используется в качестве основной.

- Рабочая директория – директория в которой происходит сама разработка, а именно корень проекта;
- Локальный репозиторий – директория “.git”, где хранятся коммиты и другие объекты
- Удаленный репозиторий – общий репозиторий, куда передаются коммиты из локального репозитория для других разработчиков.
- Область подготовки (staging area) – коммит, в который включают выбранные изменения, то есть в данный коммит могут быть включены не все изменения.

1.2 Установка и настройка Git

Для установки git воспользуйтесь командой:

```
apt-get install git
```

Далее укажите ваше имя и адрес электронной почты, как это продемонстрировано ниже:

```
git config --global user.name "Ivan Petrov"
```

```
git config --global user.email ivanpetrov@example.com
```

Опция --global позволяет настраивать систему только один раз для всех ваших действий в ней. В случае если вам необходимо использовать для отдельных проектов другое имя и почту, то выполните эту команду без опции --global в каталоге проекта. Для проверки конфигураций используйте команду:

```
git config --list
```

1.3 Фиксация изменений (коммиты)

Создание пустого репозитория осуществляется следующей командой:

```
git init
```

В случае если вы хотите принять участие в разработке уже имеющегося проекта, то необходимо скопировать данный репозиторий в локальную папку из удаленного репозитория:

```
git clone <url_удаленного_репозитория>
```

После выше приведенной команды в текущей папке появляется папка .git, в которой будет содержаться копия удаленного репозитория.

Если вы произвели изменения в проекте и хотите их добавить в staging area, используйте следующую команду:

```
git add <имя_файла>
```

Далее вы производите коммит в локальный репозиторий:

```
git commit -m "комментарии_к_коммиту"
```

Для отправки коммитов в удаленный репозиторий выполните команду:

```
git push
```

Для получения изменений из удаленного репозитория воспользуйтесь командой:

```
git pull
```

Каждый коммит имеет свое имя, которое является результатом хеш функции sha-1 от содержимого коммита. Для просмотра коммитов используйте команду:

```
git log
```

Для более удобного просмотра выполните следующую команду:

```
git log --pretty=format:"%H[%cd]: %an - %s" --graph --date=format:%c
```

Для завершения просмотра нажмите на клавишу q.

Чтобы посмотреть, что находится в рабочей директории и staging area, выполните команду:

```
git status
```

Для переключения рабочей директории на предыдущее состояние используйте команду:

```
git checkout <hash_коммита>
```

Примечание: перед переключением выполните команду `git status` для проверки, нет ли у вас каких-либо локальных незафиксированных изменений.

1.4 Ветки проекта

Для создания новой ветки и переключения на нее выполните:

```
git pull
```

```
git checkout -b <имя_новой_ветки>
```

Создать ветку без переключения можно таким образом:

```
git branch <имя_новой_ветки>
```

Переключение на ветку:

```
git checkout <имя_ветки>
```

Для объединения двух веток необходимо выполнить `merge` коммит. При этом в основной ветке появится код, который находился в ветке задачи, но в ветке задачи не появится новый код из основной ветки.

Для объединения одной ветки с другой надо переключиться на ту ветку, которую вы хотите объединить командой `git checkout <имя_ветки>`, после получить последние изменения, сделанные в этой ветке, командой `git pull`, а затем выполнить команду:

```
git merge <имя_ветки>
```

Чтобы избежать конфликты при создании новой ветки, необходимо использовать команду `git pull`.

1.5 Задание: Выполнение локального рабочего процесса Git через командную строку

Откройте командную строку. Некоторые команды характерны Linux, например, добавление в файл или создание каталога. Замените эти команды при необходимости на команды для вашей операционной системы (ОС).

Создание каталога

Следующие команды создают пустой каталог, который позже будет использован для хранения рабочего дерева и репозитория Git.

```
# switch to the home directory
cd

# create a directory and switch into it
mkdir repo01
cd repo01

# create a new directory
mkdir datafiles
```

Рисунок 1.5.1. Команды для создания пустого каталога

Создание нового Git-репозитория

Теперь вы создадите новый Git-репозиторий с рабочим деревом.

Каждый Git-репозиторий хранится в папке `.git`. Этот каталог хранит полную историю хранилища. В файл `.git/config` находятся конфигурации хранилища.

Для того, чтобы создать Git-репозиторий в текущем каталоге, используйте команду `git init` (рис.1.5.2).

```
# you should still be in the repo01 directory
cd ~/repo01

# initialize the Git repository
# for the current directory
git init
```

Рисунок 1.5.2. Команды для создания Git-репозитория

Все файлы в папке репозитория, кроме папки `.git`, являются рабочим деревом для репозитория.

Создание нового контента

Используйте следующие команды для создания нескольких новых файлов.

```
# switch to your Git repository
cd ~/repo01

# create an empty file in a new directory
touch datafiles/data.txt

# create a few files with content
ls > test01
echo "bar" > test02
echo "foo" > test03
```

Рисунок 1.5.3. Команды для создание новых файлов

Просмотр текущего статуса хранилища

Команда `git status` демонстрирует состояние рабочего дерева, то есть какие файлы были изменены, какие файлы находятся на этапе подготовки и какие не находятся в области подготовки. Также эта команда показывает, у каких файлов существуют конфликты, и предоставляет индикацию, где предлагает произвести соответствующие изменения, например, добавить эти файлы в область подготовки или удалить их и т.д. Запустите эту команду.

Вывод должен выглядеть подобно следующему изображению.

```
On branch master

Initial commit

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

    datafiles/
    test01
    test02
    test03
```

Рисунок 1.5.4. Вывод команды `git status`

Добавление изменений в область подготовки

Перед тем как зафиксировать изменения в Git-репозитории, необходимо выбрать изменения, которые нужно зафиксировать, с помощью команды `git add`. Эта команда добавляет изменения в область подготовки. Вы можете добавить все изменения в область подготовки, используя опцию `."`.


```
# add all files to the index of the Git repository
git add .
```

Рисунок 1.5.5. Команда для добавления изменений в область подготовки

Для просмотра результата воспользуйтесь командой `git status`. Вывод должен соответствовать следующему изображению.

```
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:   datafiles/data.txt
    new file:   test01
    new file:   test02
    new file:   test03
```

Рисунок 1.5.6. Состояние после добавления изменений в область подготовки

Изменение файлов, которые находятся на этапе подготовки

Если вы измените один из файлов перед коммитом, вам необходимо снова добавить изменения в область подготовки. Это необходимо потому, что Git фиксирует состояние контента в области подготовки. Все новые изменения должны быть подготовлены.

```
# append a string to the test03 file
echo "foo2" >> test03

# see the result
git status
```

Рисунок 1.5.7. Изменение файла

Убедитесь, что новые изменения не подготовлены.

```
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:   datafiles/data.txt
    new file:   test01
    new file:   test02
    new file:   test03

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   test03
```

Рисунок 1.5.8. Состояние после изменений

Добавьте изменения в область подготовки с использованием команды “git add .”.

Проверьте, появились ли изменения в области подготовки. Результат должен соответствовать следующему изображению.

```
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:   datafiles/data.txt
    new file:   test01
    new file:   test02
    new file:   test03
```

Рисунок 1.5.9. Состояние после добавления изменений в область подготовки 2

Фиксирование области подготовки в репозитории

После добавления файлов в область подготовки Git вы можете зафиксировать их в Git-репозитории с помощью команды git commit. Она создает новый коммит изменениями в Git-репозитории с ссылками HEAD на новый коммит. Параметр -m (-message) позволяет указать сообщение коммита. Если вы пропустите этот параметр, то по умолчанию запустится редактор, где вы сможете ввести сообщение.

```
# commit your file to the local repository
git commit -m "Initial commit"
```

Рисунок 1.5.10. Фиксирование изменений

Git также предлагает интерактивный режим, в котором вы сможете выбрать, какие изменения нужно зафиксировать. Для использования данного режима введите команду: `git commit –interactive`.

Просмотр истории коммитов Git

Ранее вы создали локальный Git-репозиторий в папке `.git` и добавили туда файлы. Запустите команду `git log`, чтобы увидеть историю.

Удаление файлов

Если вы удаляете файл, используйте команду `git add .` для добавления файла на удаление в область подготовки.

```
# remove the "test03" file
rm test03
# add and commit the removal
git add .
# if you use Git version < 2.0 use: git add -A .
git commit -m "Removes the test03 file"
```

Рисунок 1.5.11. Удаление файлов

Также вы можете использовать команду `git rm` для удаления файла из рабочего дерева и записи удаления файла в область подготовки.

Отмена изменений в файлах в рабочем дереве

Используйте команду `git checkout` для сброса отслеживаемого файла (файл, который подготовлен или зафиксирован) в его последнее состояние подготовки или коммита. Команда удаляет изменения в рабочем дереве. Данная команда не может быть использована для файлов, которые еще не находятся в области подготовки или не зафиксированы.

```
echo "useless data" >> test02
echo "another unwanted file" >> unwantedfile.txt

# see the status
git status

# remove unwanted changes from the working tree
# CAREFUL this deletes the local changes in the tracked file
git checkout test02

# unwantedstaged.txt is not tracked by Git simply delete it
rm unwantedfile.txt
```

Рисунок 1.5.12. Отмена изменений

Изменение коммита с помощью параметра `amend`

Команда `git commit --amend` позволяет изменить последний коммит. Она создает новый коммит с откорректированными последними изменениями.

Предположим, что последний коммит был неверен, так как там была опечатка. Команда, приведенная ниже, исправляет опечатку с использованием параметра `--amend`.

```
# assuming you have something to commit
git commit -m "message with a tpyo here"
```

```
# amend the last commit
git commit --amend -m "More changes - now correct"
```

Рисунок 1.5.13. Изменение последнего коммита

Команду `git --amend` нельзя использовать для коммитов, которые были переданы в общедоступную ветку другого Git-репозитория. Она создает новый идентификатор коммита, и люди могут работать с проектом, основываясь на существующем коммите. В таком случае им необходимо будет перенести свои наработки на новый коммит.

Игнорирование файлов и директорий с помощью файла `.gitignore`

Создайте файл в `.gitignore` в корне каталога Git для того, чтобы игнорировать указанные в нем директории и файлы.

```
cd ~/repo01
touch .gitignore
echo ".metadata/" >> .gitignore
echo "doNotTrackFile.txt" >> .gitignore
```

Рисунок 1.5.14. Команды для игнорирования указанных файлов и директорий

В результате в файле должно быть следующее:

```
.metadata/
doNotTrackFile.txt
```

Рисунок 1.5.15. Файл `.gitignore`

Для файла `.gitignore` также, как и для других, необходимо фиксировать изменения.

```
# add the .gitignore file to the staging area
git add .gitignore
# commit the change
git commit -m "Adds .gitignore file"
```

Рисунок 1.5.16. Коммит файла .gitignore

2. Использование Maven для разработки Java приложений

В данном пособии описан инструмент Maven в рамках разработки Java-приложений.

2.1 Что такое Apache Maven?

Инструмент для сборки Apache Maven.

Apache Maven – это усовершенствованная программа для автоматизации сборки проектов.

Apache Maven решает следующие задачи: компиляция исходного кода, выполнение тестов и помещение результата в JAR_files. Кроме перечисленных стандартных возможностей, у Maven есть функционал по созданию веб-сайтов, загрузка результатов программы и формирование отчетов.

Maven предоставляет автоматизированный процесс создания начальной структуры папок для Java-приложений, выполняя компиляцию, тест, упаковывание и размещение конечного продукта. Он реализован на Java, что делает его платформонезависимым. Java также является наилучшей рабочей средой для Maven.

Ключевые особенности Maven.

Apache Maven может быть использован в средах, где используются такие инструменты сборки, как GNU Make или Apache Ant. Главными особенностями Maven являются:

- **Соглашение о конфигурации:** Maven старается избегать как можно больше конфигураций, выбирая реальные значения по умолчанию и предоставляя шаблоны проектов (архетипы)
- **Управление зависимостями:** Можно определить зависимости с другими проектами. Во время сборки, программа Maven разрешает зависимости и при необходимости строит зависимые проекты.
- **Репозиторий:** Зависимости проекта могут быть загружены из локальной файловой системы, из Интернета или из общедоступных

репозиториях. Компания, разрабатывающая Maven, также предоставляет центральное хранилище так называемое Maven Central.

- **Расширяемость с помощью плагинов:** Система сборки Maven расширяема с помощью плагинов, что позволяет сохранить оперативную память Maven небольшой.

Maven Central

Maven Central – это открытый репозиторий, предоставленный компанией Sonatype. Данный репозиторий содержит библиотеки, которые могут быть использованы в вашем проекте. Сборка Maven использует по умолчанию Maven Central для поиска требуемых библиотек.

2.2 Установка инструментов командной строки Maven

Загрузка и установка Maven

Данная установка не требуется при использовании Maven в Eclipse IDE.

В случае, если вы хотите использовать Maven из командной строки, то вам необходимо установить поддержку командной строки Maven.

Для ручной установки скачайте Maven на официальном сайте Maven. Далее распакуйте загруженный дистрибутив в выбранную директорию на вашем компьютере и добавьте переменную среды M2_HOME, указывающую на этот каталог. Обновите переменную PATH, добавив папку M2_HOME/bin .

Ubuntu

Большинство дистрибутивов включают Maven в свои основные репозитории. Для установки Maven в Ubuntu используйте следующую команду в командной строке:

```
sudo apt-get install mvn
# if that does not work, try
sudo apt-get install maven
```

Рисунок 2.2.1. Установки Maven в Ubuntu

Проверка установки.

Для проверки правильности установки откройте консоль и используйте следующую команду:


```
# command
mvn -version

# output should be similar to

Apache Maven 3.0.5
Maven home: /usr/share/maven
Java version: 1.7.0_55, vendor: Oracle Corporation
Java home: /usr/lib/jvm/java-7-openjdk-amd64/jre
Default locale: en_US, platform encoding: UTF-8
OS name: "linux", version: "3.13.0-33-generic", arch: "amd64", family:
"unix"
```

Рисунок 2.2.2. Проверка установки

В результате выполненной команды выводится отчет, где указана версия Maven.

2.3 Запуск сборки Maven

Запуск сборки из командной строки

Maven предоставляет инструмент командной строки.

Для того, чтобы создать проект из командной строки, введите команду `mvn`. Команда должна быть выполнена в директории, которая содержит `pom`-файл.

Инструментарий Maven читает `pom`-файл и разрешает зависимости проекта. При необходимости Maven проверяет доступны ли необходимые компоненты в локальном репозитории. Локальный репозиторий расположен в папке `m2/repository`.

Если зависимости не доступны в локальном репозитории, Maven загружает их из центрального репозитория.

Maven выполняет все этапы жизненного цикла до указанного.

Например, команда `mvn install` запускает упаковку `jar`, которая в себя включает компиляцию исходных кодов, выполнение тестов и упаковку скомпилированных файлов в `JAR`-файл. После перечисленных действий проект будет загружен в локальный репозиторий для использования его в качестве зависимости другими сборками Maven.

Результат сборки размещается в целевой папке.

```
mvn install
```

Рисунок 2.3.1. Сборка проекта Maven

Для автоматического удаления результата предыдущих сборок перед новой компоновкой используется следующая команда:

```
mvn clean install
```

Рисунок 2.3.2. Очистка предыдущих результатов и сборка проекта Maven

Maven по умолчанию обращается к центральному репозиторию для проверки на изменение зависимостей. Если вы хотите использовать локальный репозиторий, вы можете добавить к команде ключ `-o` для работы Maven в режиме офлайн.

```
mvn -o clean install
```

Рисунок 2.3.3. Работа Maven в режиме офлайн

Использование ошибки сборки.

Если у вас сложный многомодульный проект, у вас есть возможность протестировать конкретный модуль.

- `-fae, --fail-at-end` – вызывает сбой сборки после того, как все модули построены; позволяет всем незатронутым сборкам продолжиться;
- `-ff, --fail-fast` – останавливает при первом сбое сборки;
- `-fn, --fail-never` – НИКОГДА не завершает сборку независимо от ее результата.

Параметры `-fn` и `-fae` полезны для проверки сборок, которые выполняются внутри инструмента непрерывной интеграции, таких как Jenkins и для отображения всех ошибок сборки.

Скаффолдинг проекта с Maven

Maven поддерживает скаффолдинг, основанный на шаблонах проекта. Такой проект можно создать с помощью плагина архетипов. В Maven предусмотрены артефакты почти для всего, от простого Java-приложения до сложного веб-приложения.

Цель скаффолдинга — упрощение работы со средой Maven. Скаффолдинг предоставляет “стандартизированную” структуру проектов.

Вы можете создать проект, выполнив цель генерации в плагине архетипа. Для этого используйте следующую команду: `mvn archetype:generate`. В результате этой команды начинается создание проекта в интерактивном режиме, во время которого вам предлагают выбрать необходимые настройки.

2.4 Maven Wrapper

Присутствие воспроизводимых сборок на каждой машине очень востребована для того, чтобы локальные сборки также как сборки на CI сервере были одинаковыми. Во время сборки в Maven одинаковые входные параметры должны всегда сопровождаться одинаковыми выходными параметрами. Поэтому при работе на разных машинах, каждая из них должна иметь одну и ту же версию Maven.

Такое решение может быть затруднительным в случае, если разные проекты требуют разных версий Maven.

Чтобы не возникало подобных проблем, используют Maven Wrapper. Maven Wrapper — это скрипт-обертка, которая позволяет запускать проект Maven, без установки самой среды. Таким образом пользователям не нужно устанавливать Maven на свой компьютер. Но для создания файлов Maven Wrapper локальная установка среды Maven все же потребуется.

Такая концепция была позаимствована у среды Gradle, у которой также имеется Gradle Wrapper.

Создание Maven Wrapper

Создайте Maven Wrapper для проекта с последней доступной версией Maven.

```
cd {your-project}
mvn -N io.takari:maven:wrapper
```

CONSOLE

Рисунок 2.4.1. Создание Maven Wrapper

Создайте оболочку Maven для проекта с указанной версией Maven, используя следующую команду:

```
cd {your-project}
mvn -N io.takari:maven:wrapper -Dmaven=3.3.0
```

Рисунок 2.4.2. Создание оболочки Maven для проекта

После выполнения данных команд, в проекте maven будут созданы следующие файлы:

- mvnw (скрипт оболочки для Unix систем)
- mvnw.cmd (командный файл для Windows)
- .mvn/wrapper/maven-wrapper.jar (Maven Wrapper JAR)
- .mvn/wrapper/maven-wrapper.properties (свойства Maven Wrapper)

Эти файлы Maven Wrapper необходимо зарегистрировать в контроле версий (GIT или SVN), чтобы другие пользователи репозитория могли строить проекты без установки Maven. При использовании Maven Wrapper не нужно беспокоиться о том, правильная ли версия Maven установлена, так как Wrapper проекта определяет и автоматически загружает ее автоматически.

Параметр командной строки `-N`, `--non-recursive` позволяет строить родительский модуль без построения его подмодулей. В таком случае Maven Wrapper будет применяться только для основного проекта, а не для каждого подмодуля.

Выполнение Maven Wrapper

Для запуска Maven Wrapper вы можете использовать команды `mvnw` для unix систем или `mvnw.cmd` для Windows систем.

UNIX:

```
./mvnw clean package
```

WINDOWS:

```
mvnw.cmd clean package
```

Рисунок 2.4.3. Запуск Maven Wrapper

2.5 Задание: Создать и собрать проект Java с использованием Maven

В этом упражнении вы научитесь создавать Java проект из командной строки Maven, а также сможете его построить.

Создание проекта

В этом упражнении для создания Java-проекта будет использована функциональность Maven - скаффолдинг. Все требуемые свойства передаются непосредственно в командной строке для того, чтобы избежать интерактивного режима. В противном случае, Maven будет запрашивать все требуемые параметры. Введите следующую строку в командной строке:

```
mvn archetype:generate -DgroupId=com.vogella.build.maven.java \
-DartifactId=com.vogella.build.maven.java \
-DarchetypeArtifactId=maven-archetype-quickstart \
-DinteractiveMode=false
```

Рисунок 2.5.1. Создание проекта

С помощью данной команды Maven создает Java-проект.

Если эта цель выполняется впервые, тогда это может занять некоторое время. Задержка объясняется тем, что Maven в первую очередь загружает все необходимые плагины и артефакты для создания проекта из центрального репозитория.

Обзор созданного проекта

Убедитесь, что Maven создал проект в вашей файловой системе таким же образом, как изображено на следующей структуре.

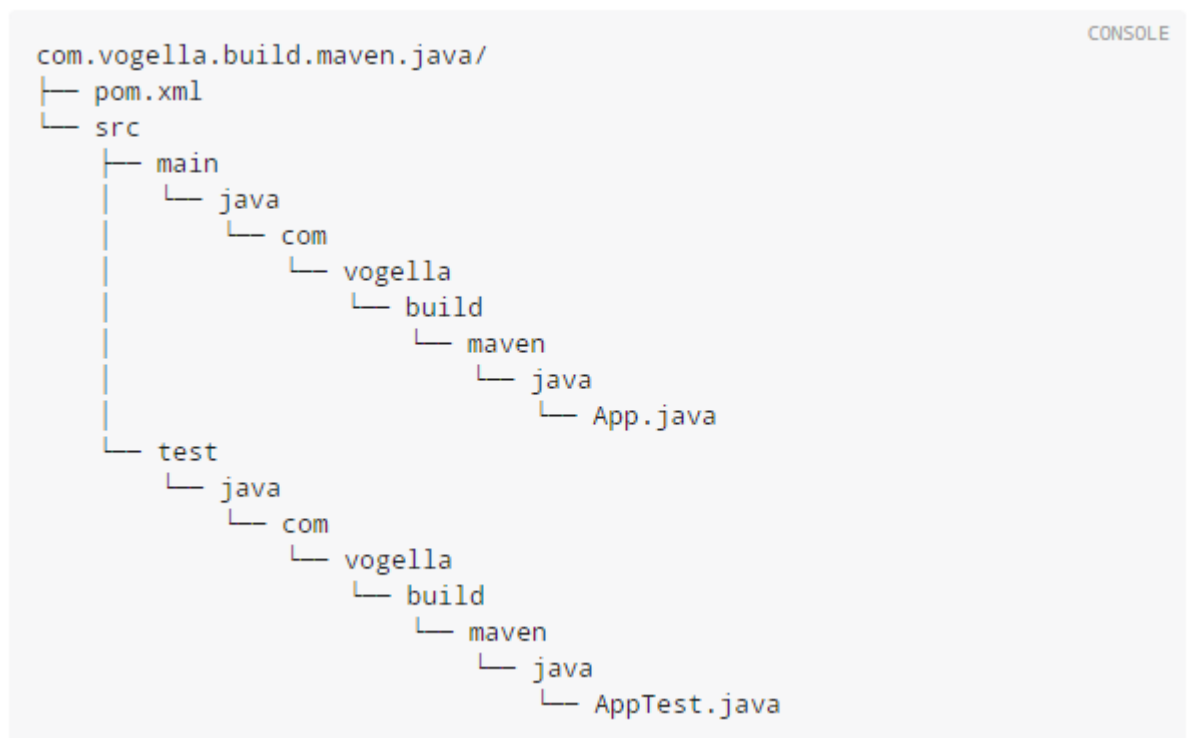


Рисунок 2.5.2. Структура проекта Maven

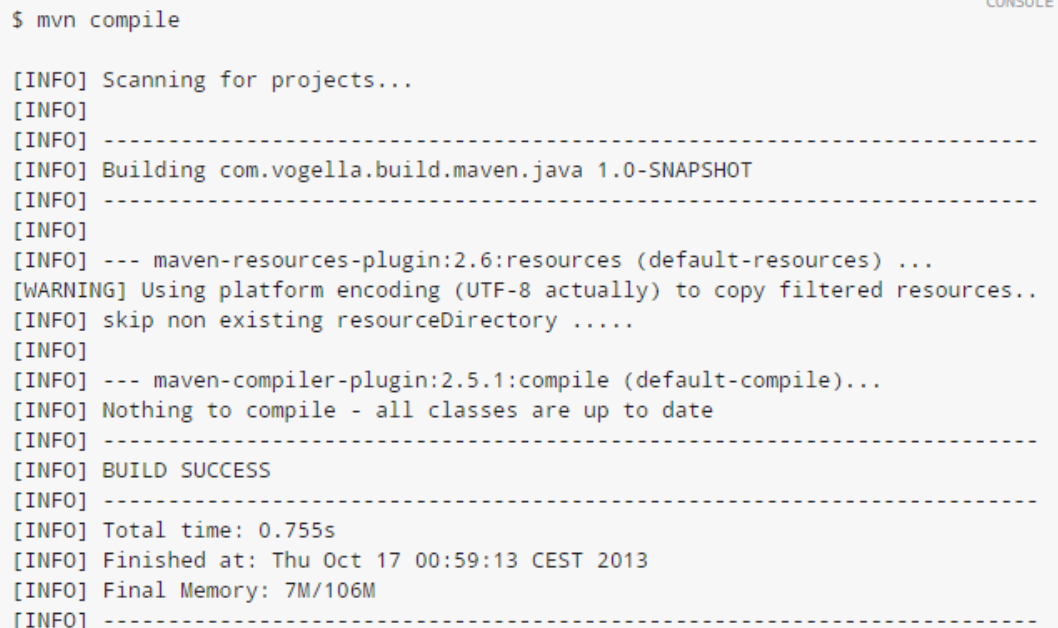
Вы создали структуру проекта Maven с исходным кодом java. В папке `./src/main` должен был появиться файл `app.java`, при открытии которого выводится строка «Hello World». Также в папке `./src/test/` должен был быть создан тест класса. В корневой же папке должен находиться файл `pom.xml`.

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.vogella.build.maven.java</groupId>
  <artifactId>com.vogella.build.maven.java</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>com.vogella.build.maven.java</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

Рисунок 2.5.3. Созданный проект

Компиляция исходного кода

Для компиляции вашего исходного Java-кода откройте командную строку в расположении вашей корневой папки проектов и запустите следующую команду.



```
$ mvn compile

[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building com.vogella.build.maven.java 1.0-SNAPSHOT
[INFO] -----
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) ...
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources..
[INFO] skip non existing resourceDirectory .....
[INFO]
[INFO] --- maven-compiler-plugin:2.5.1:compile (default-compile)...
[INFO] Nothing to compile - all classes are up to date
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 0.755s
[INFO] Finished at: Thu Oct 17 00:59:13 CEST 2013
[INFO] Final Memory: 7M/106M
[INFO] -----
```

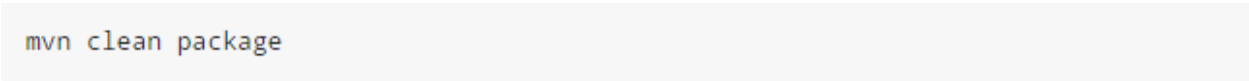
Рисунок 2.5.4. Компиляция проекта

Код проходит через все этапы компиляции Maven, которая включает в себя: разрешение зависимостей, загрузку артефактов JUnit, сборку исходного кода и даже выполнение теста JUnit

Создание JAR-файла

Теперь вам необходимо создать исполняемый JAR-файл из вашего проекта. Команда `package` создает развертываемый JAR-файл.

Чтобы удостовериться, что предыдущая сборка артефактов была удалена, вы можете использовать `clean`.



```
mvn clean package
```

Рисунок 2.5.5. Создание JAR-файла

После чего вы можете запустить полученный файл.

Запуск теста

Вместо выполнения полной сборки, также можно запустить фазы тестирования жизненного цикла Maven.

```
$ mvn test
```

.....

TESTS

Running com.vogella.build.maven.java.AppTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.003 sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
.....

Рисунок 2.5.6. Запуск теста

Удаление всех результатов сборки / Очистка проекта

Для очистки проекта и удаления сгенерированных файлов в папке `./target/`, выполните следующую команду.

```
$ mvn clean
```

Рисунок 2.5.7. Очистка проекта

2.6 Задание: Запустить Java-программу с использованием Maven

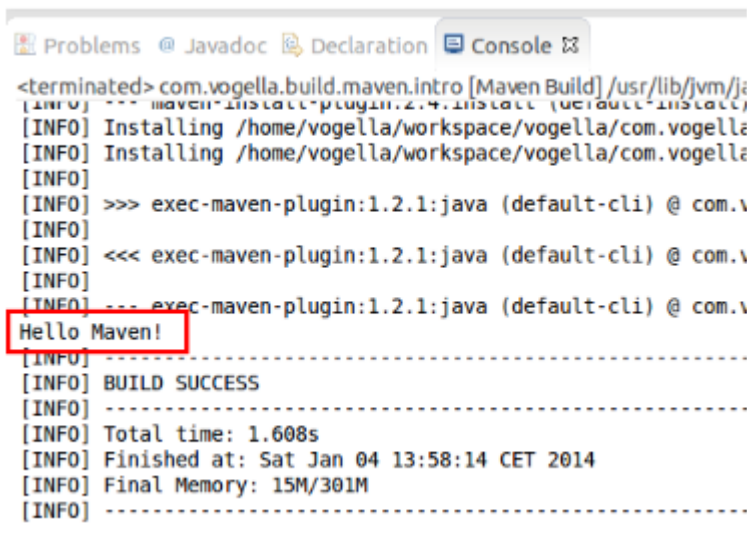
Если вы хотите запустить на выполнение программу, вы можете использовать `exec-maven-plugin`. Чтобы запустить программу, используйте цель `exec:java` в Maven. Далее продемонстрирован файл проекта `pom.xml`.


```

<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.vogella.build.maven.intro</groupId>
  <artifactId>com.vogella.build.maven.intro</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>mavenintroduction</name>
  <build>
    <sourceDirectory>src</sourceDirectory>
    <plugins>
      <plugin>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.3</version>
        <configuration>
          <source>1.8</source>
          <target>1.8</target>
        </configuration>
      </plugin>
      <plugin>
        <groupId>org.codehaus.mojo</groupId>
        <artifactId>exec-maven-plugin</artifactId>
        <version>1.2.1</version>
        <configuration>
          <mainClass>com.vogella.build.maven.intro.Main</mainClass>
        </configuration>
      </plugin>
    </plugins>
  </build>
</project>

```

Рисунок 2.6.1. Рот-файл проекта



```

<terminated> com.vogella.build.maven.intro [Maven Build] /usr/lib/jvm/ji
[INFO] --- maven-install-plugin:2.4.1:install (default-install)
[INFO] Installing /home/vogella/workspace/vogella/com.vogella
[INFO] Installing /home/vogella/workspace/vogella/com.vogella
[INFO] >>> exec-maven-plugin:1.2.1:java (default-cli) @ com.v
[INFO] <<< exec-maven-plugin:1.2.1:java (default-cli) @ com.v
[INFO] --- exec-maven-plugin:1.2.1:java (default-cli) @ com.v
Hello Maven!
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 1.608s
[INFO] Finished at: Sat Jan 04 13:58:14 CET 2014
[INFO] Final Memory: 15M/301M
[INFO] -----

```

Рисунок 2.6.2. Результат работы приложения

2.7 Конфигурация и координаты проекта Maven

Project Object Model (POM)

Конфигурация проектов Maven выполняется с помощью Project Object Model (POM). Эта модель обычно представлена pom.xml файлом. Этот файл конфигурации Maven называется pom-файлом.

Pom-файл описывает проект, конфигурирует плагины и объявляет зависимости. Он присваивает проекту имя и предоставляет набор уникальных идентификаторов (называемых координатами) для проекта. Также он определяет отношение между этим и другими проектами с помощью зависимостей, родителей и предпосылок.

Pom-файл мультипроекта включает в себя раздел модулей. Этот раздел несет в себе информацию о том, какие проекты являются частью сборки.

В разделе сборки pom-файла вы можете определить плагины, которые необходимы для сборки.

Уникальный идентификатор проекта (GAV) – координаты проекта

Maven определяет набор идентификаторов, которые могут быть использованы для уникальной идентификации компонентов Maven. К примеру, это может быть использовано для определения точной версии тестовой библиотеки JUnit, которая должна использоваться для проекта. Их определяют через свойства groupId, artifactId, version и packaging.

Таблица 2.7.1. Координатные атрибуты

Имя	Описание
groupId	Определяет уникальное базовое имя организации или группы, которая создала проект. Это обычно обратное доменное имя. Для создания groupId также определяют пакет основного класса

Имя	Описание
artifactId	Определяет уникальное имя проекта. Если вы создаете новый проект через Maven, оно также используется в качестве корневой папки для проекта.
packaging	Определяет метод упаковки. Это может быть, к примеру, jar-, war- или ear-файл. Если тип упаковки pom, то Maven ничего не создает для этого проекта, то это просто метаданные.
version	Определяет версию проекта.

Проект groupId:artifactId:version (т.е. GAV) инициализирует проект.

Полные координаты Maven записываются в следующем формате:

groupId:artifactId:packaging:version.

По умолчанию, это единственный конфигурационный файл необходимый для сборки процесса.

Результат сборки называется артефактом. Артефакт может быть, например, исполняемым файлом или архивом документов.

2.8 Плагины Maven, цели и жизненный цикл

Плагины и задачи Maven

Плагин Maven – это набор из одной или нескольких целей. Целью является «рабочая единица» в Maven.

Цели могут определять параметры, которые могут иметь значения по умолчанию. Они могут быть привязаны к фазам жизненного цикла. Цели выполняются на основе найденной информации в pom-файле проекта, например, компилятор: цель компиляции проверяет POM на соответствующие параметры.

Жизненный цикл Maven

Каждая сборка соответствует указанному жизненному циклу. Жизненный цикл по умолчанию в Maven включает в себя часто используемые этапы сборки, такие как компиляция, тестирование и упаковка.

Этапы жизненного цикла Maven.

- `validate` - проверка правильности проекта и доступность всей необходимой информации;
- `compile` - компиляция исходного кода проекта;
- `test` – выполнение теста;
- `package` – упаковка проекта в такие форматы, как `jar`, `war` или `ear`;
- `integration-test` – выполнение дополнительных тестов над упакованным результатом;
- `verify` – проверка действителен ли пакет;
- `install` - установка пакета в локальный репозиторий;
- `deploy` - копирование финального пакета в удалённый репозиторий.

Все фазы жизненного цикла можно посмотреть на <http://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html>.

Если вы укажете Maven выполнить определенный этап, Maven выполнит все этапы, предшествующие указанному этапу в предварительно определенной последовательности. Все соответствующие цели выполняются в течение этого процесса. Цель соответствует этапу только тогда, когда плагин Maven или `pom` связывают эту цель с соответствующим этапом жизненного цикла.

Пакеты и связанные с ними цели

Каждый упаковка имеет набор целей, связанных с определенными этапами жизненного цикла. Например, у пакета `jar` есть следующие цели, связанные с этапами жизненного цикла:

Таблица 2.8.1. Цели упаковки

Этапы жизненного цикла	Цели
process-resources	resources:resources
compile	compiler:compile
process-test-resources	resources:testResources
test-compile	compiler:testCompile
test	surefire:test
package	jar:jar
install	install:install
deploy	deploy:deploy

Добавление задач к этапам жизненного цикла

Вы можете добавить цели к этапам жизненного цикла, настроив плагины и добавив их к жизненному циклу в pom файле. Необходимо указать, какая цель должна быть выполнена. Если в плагине не указан жизненный цикл по умолчанию, который он должен запустить, необходимо указать этап жизненного цикла.

```

<plugin>
  <groupId>com.vogella.example</groupId>
  <artifactId>vogella-some-maven-plugin</artifactId>
  <version>1.0</version>
  <executions>
    <execution>
      <phase>verify</phase>
      <goals>
        <goal>checklinks</goal>
      </goals>
    </execution>
  </executions>
</plugin>

```

XML

Рисунок 2.8.1. Плагин проекта

2.9 Репозитории Maven и разрешение зависимостей

Репозитории Maven

На начальном этапе сборки Maven проверяет, есть ли определенная версия всех требуемых зависимостей артефакта и плагинов Maven. Если они отсутствуют, то он извлекает их из репозитория Maven. Репозиторий – директория, где хранятся библиотеки, плагины и артефакты необходимые Maven.

При возникновении необходимости Maven загружает артефакты и плагины в локальный репозиторий. Локальный репозиторий по умолчанию находится в домашней директории пользователя в папке `.m2/repository`. Если артефакт или плагин доступен в локальном репозитории, то Maven использует его.

По умолчанию Maven использует удаленный репозиторий (), откуда он загружает плагины Maven и зависимости. Вы можете задать такую конфигурацию Maven, чтобы использовались другие репозитории, заменив репозитории по умолчанию.

После выполнения команды `mvn install` созданные артефакты будут установлены в локальный репозиторий Maven.

Разрешение зависимостей Maven и Maven Reactor

Каждый проект может определять зависимости используя уникальный идентификатор (GAV) компонента, который необходим для компиляции.

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>3.8.1</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

Рисунок 2.9.1. Зависимости проекта

Во время сборки система Maven пытается разрешить зависимости строящихся модулей. Для разрешения зависимостей, Maven использует следующие источники в указанном порядке:

- Модули, которые включены в проект Maven (Maven Reactor)
- Локальный репозиторий
- Центральный репозиторий Maven

2.10 Многомодульные проекты (Агрегатор)

Maven поддерживает сборку сразу нескольких проектов. Многомодульный проект (агрегатор) определяется родительским POM, ссылающимся на один или несколько проектов. Агрегатор может также содержать конфигурацию сборки или включать другой родительский POM для получения его конфигурации.



```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.vogella.tychoexample</groupId>
  <artifactId>com.vogella.tycho.aggregator</artifactId>
  <version>1.0.0-SNAPSHOT</version>
  <packaging>pom</packaging>

  <parent>
    <groupId>com.vogella.tychoexample</groupId>
    <artifactId>com.vogella.tycho.parent</artifactId>
    <version>1.0.0-SNAPSHOT</version>
    <relativePath>../com.vogella.tycho.parent</relativePath>
  </parent>

  <modules>
    <module>../com.vogella.build.targetdefinition</module>
    <module>../com.vogella.tycho.plugin1</module>
    <module>../com.vogella.tycho.plugin2</module>
    <module>../com.vogella.tycho.rcp</module>
    <module>../com.vogella.tycho.product</module>
    <module>../com.vogella.tycho.feature</module>
    <module>../com.vogella.tycho.update</module>
    <module>../com.vogella.tycho.unittests</module>
  </modules>
</project>
```

Рисунок 2.10.1. Многомодульный проект

2.11 Использование профилей и свойств в Maven

Использование профилей.

Maven поддерживает использование профилей для определения различных конфигураций. После запуска Maven, вы можете указать определенный профиль для дальнейшего его использования. Для этого укажите параметр `-P`, после которого (без пробела) необходимо указать выбранный профиль, например, `-PyourProfile`.

```

<profiles>
  <profile>
    <id>dev</id>
    <activation>
      <activeByDefault>true</activeByDefault>
    </activation>
    <properties>
      <db.location>URL_to_dev_system</db.location>
      <logo.image>companylogo.png</logo.image>
    </properties>
  </profile>
  <profile>
    <id>production</id>
    <properties>
      <db.location>URL_to_prod_system</db.location>
      <logo.image>companylogo2.png</logo.image>
    </properties>
  </profile>
</profiles>

```

Рисунок 2.11.1. Профили и свойства в Maven

Использование свойств.

В файлах сборки вы можете указать свойства. Свойства можно переопределить через командную строку с использованием параметра `-D`, после которого следует (без пробела) само значение. Пример продемонстрирован в следующей главе.

Пример изменения свойств проекта: пропуск тестов в сборке Maven

С использованием свойств можно осуществить пропуск теста во время сборки. Данный шаг продемонстрирован в следующем фрагменте.

```

<properties>
  <skipTests>true</skipTests>
  <maven.build.timestamp.format>yyyyMMdd-HH:mm</maven.build.timestamp.format>
  <buildTimestamp>${maven.build.timestamp}</buildTimestamp>
  <buildId>${buildType}${buildTimestamp}</buildId>
</properties>

```

Рисунок 2.11.2 Изменения свойств проекта

Переопределение рассматриваемых параметров через командную строку осуществляется следующим образом:

```

mvn clean install -DskipTests=false

```

Рисунок 2.11.3. Переопределение параметров

Полезные свойства

`skipTests` – свойство, которое определяет должны ли быть выполнены тесты или нет.

`showWarnings` – свойство, которое, определяет осуществлять ли отображение результатов сборки Maven (предупреждения компилятора).

2.12 Maven и система контроля версий

Результаты сборки Maven размещает в целевой папке проекта. Эти результаты не нужно включать в систему управления версиями.

Для этого добавьте этот каталог в список ресурсов игнорирования. Например, если вы используете Git в качестве системы контроля версии, добавьте “target/” в файл `.gitignore` в корне проекта.

2.13 Настройки Maven

Maven позволяет определить настройки на глобальном, пользовательском и проектном уровнях. Распространенным случаем является определение на пользовательском уровне таких параметров, как прокси-сервер или пароли для загрузки артефактов сборки на сервер.

Вы можете посмотреть расположение файлов в Eclipse IDE через настройки `Windows Preferences Maven User`.

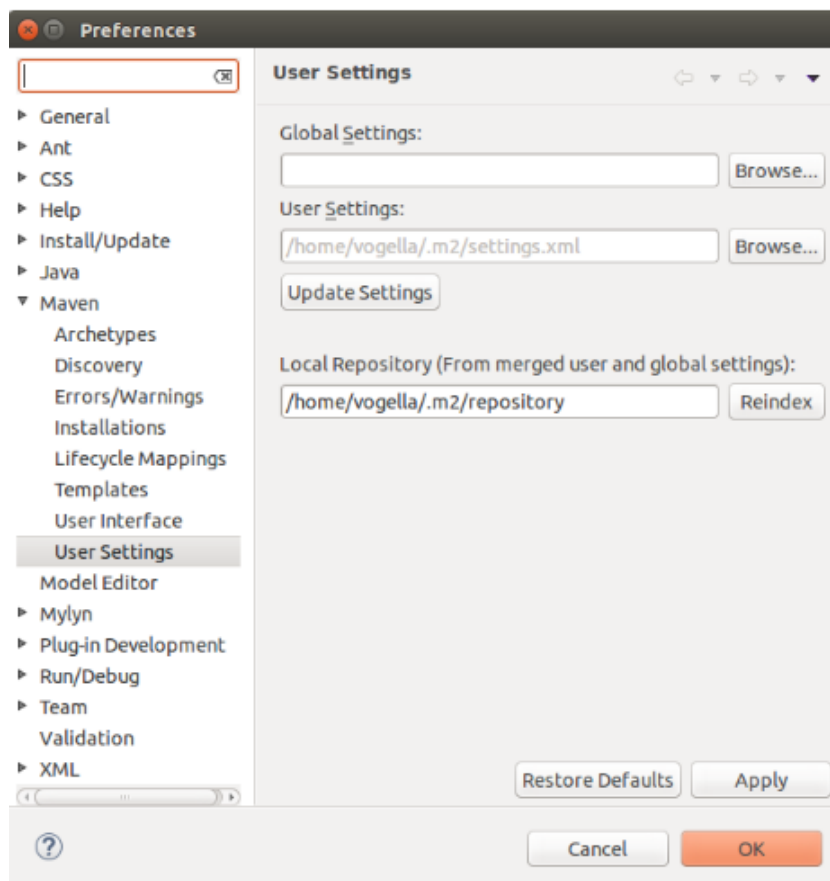


Рисунок 2.13.1. Настройки Maven

В файл settings.xml определен прокси-сервер. Если этот файл размещен в папке .m2, Maven использует указанный в нем прокси-сервер.

```
<settings>
<proxies>
<proxy>
<active>true</active>
<protocol>http</protocol>
<host>proxy</host>
<port>8080</port>
<username>your_user</username>
<password>your_password</password>
<nonProxyHosts>www.google.com|*.test.com</nonProxyHosts>
</proxy>
</proxies>
</settings>
```

Рисунок 2.13.2. Настройка прокси-сервера

2.14 Полезные параметры Maven

В следующей таблице приведены полезные параметры Maven.

Таблица 2.14.1.Параметры сборки Maven

Параметр	Описание
--log-filelog.txt	Результат сборки записывается в указанный файл
--debug	Выводит подробную информации во время процесса сборки

2.15 Плагины Maven для анализа проекта

Отображение дерева зависимостей

Maven предоставляет плагин, который можно использовать для визуализации дерева зависимостей в консоли или в выходном файле.

```
# Show dependencies inside the console
mvn dependency:tree -Dverbose
# Write dependency to a file
mvn dependency:tree -Dverbose -DoutputFile=/home/simon/maven-dependencies
```

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.vogella</groupId>
  <artifactId>com.vogella.maven.dependencies</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>pom</packaging>

  <dependencies>
    <dependency>
      <groupId>org.slf4j</groupId>
      <artifactId>slf4j-log4j12</artifactId>
      <version>1.7.10</version>
    </dependency>
  </dependencies>

</project>
```

Рисунок 2.15.1. Дерево зависимостей

Результат команды `mvn dependency:tree -Dverbose` имеет следующий вид:

```
com.vogella:com.vogella.maven.dependencies:pom:0.0.1-SNAPSHOT
\-- org.slf4j:slf4j-log4j12:jar:1.7.10:compile
    +- org.slf4j:slf4j-api:jar:1.7.10:compile
    \-- log4j:log4j:jar:1.2.17:compile
```

Рисунок 2.15.2. Результат команды `mvn dependency:tree -Dverbose`

Плагин версий.

Задача плагина версий Maven (*the Versions Maven Plugin*) – `display-dependency-updates`. Его используют для определения, доступны ли более новые версии зависимости.

Результат выполнения команды `mvn versions:display-dependency-updates` имеет следующий вид:

```
--- versions-maven-plugin:2.1:display-dependency-updates (default-cli) @ manifest-maven-plugin ---
artifact biz.aQute:bnd: checking for updates from central
artifact biz.aQute:bnd: checking for updates from release
artifact commons-io:commons-io: checking for updates from central
artifact commons-io:commons-io: checking for updates from release
artifact junit:junit: checking for updates from central
artifact junit:junit: checking for updates from release
artifact org.apache.maven:maven-plugin-api: checking for updates from central
artifact org.apache.maven:maven-plugin-api: checking for updates from release
artifact org.apache.maven.plugin-tools:maven-plugin-annotations: checking for updates from central
artifact org.apache.maven.plugin-tools:maven-plugin-annotations: checking for updates from release
The following dependencies in Dependencies have newer versions:
  junit:junit ..... 3.8.1 -> 4.12
  org.apache.maven:maven-plugin-api ..... 3.2.3 -> 3.2.5

-----
BUILD SUCCESS
-----
Total time: 4.214s
Finished at: Tue Feb 10 16:00:31 CET 2015
Final Memory: 13M/297M
-----
```

Рисунок 2.15.3. Результат команды `mvn versions:display-dependency-updates`

3. Использование Maven с Eclipse IDE

Данный раздел описывает использование Maven с Eclipse IDE для разработки Java приложений.

Eclipse IDE предоставляет отличную поддержку для Maven. Этот инструмент разработан в проекте M2Eclipse.

Данный инструмент управляет зависимостями проекта и обновляет путь к классам зависимостей проекта в Eclipse IDE. Он обеспечивает максимально простое и удобное использование Maven в Eclipse. Также данный инструмент предоставляет различные виды импорта мастеров для создания новых проектов на основе Maven.

Он также предоставляет редактор для файла конфигурации `pom.xml` через структурированный интерфейс. Вы можете выбрать вкладку `pom.xml` для непосредственного редактирования данных.

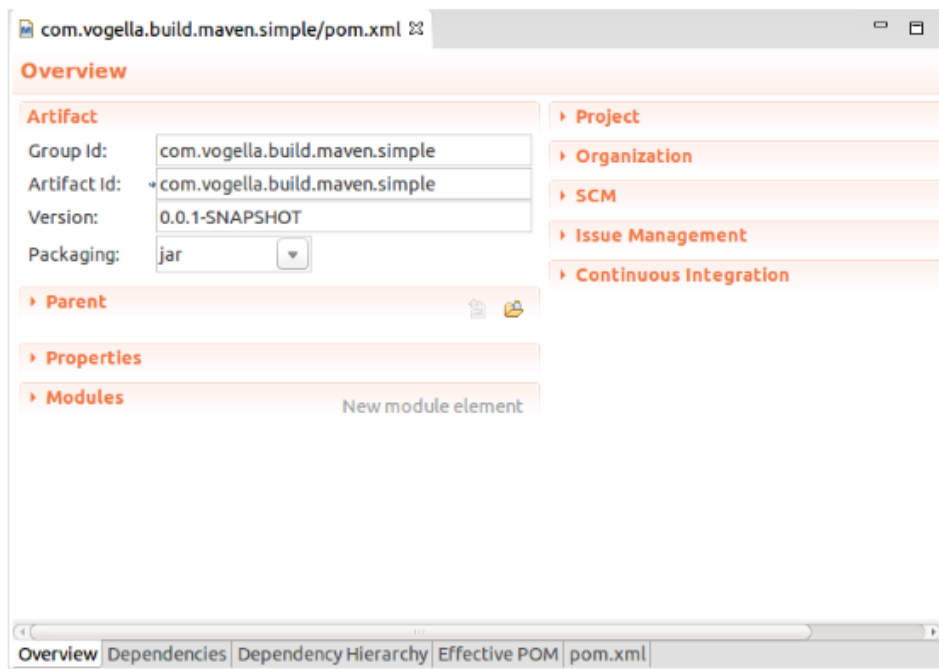


Рисунок 3.1. Редактор для файла конфигурации `pom.xml`

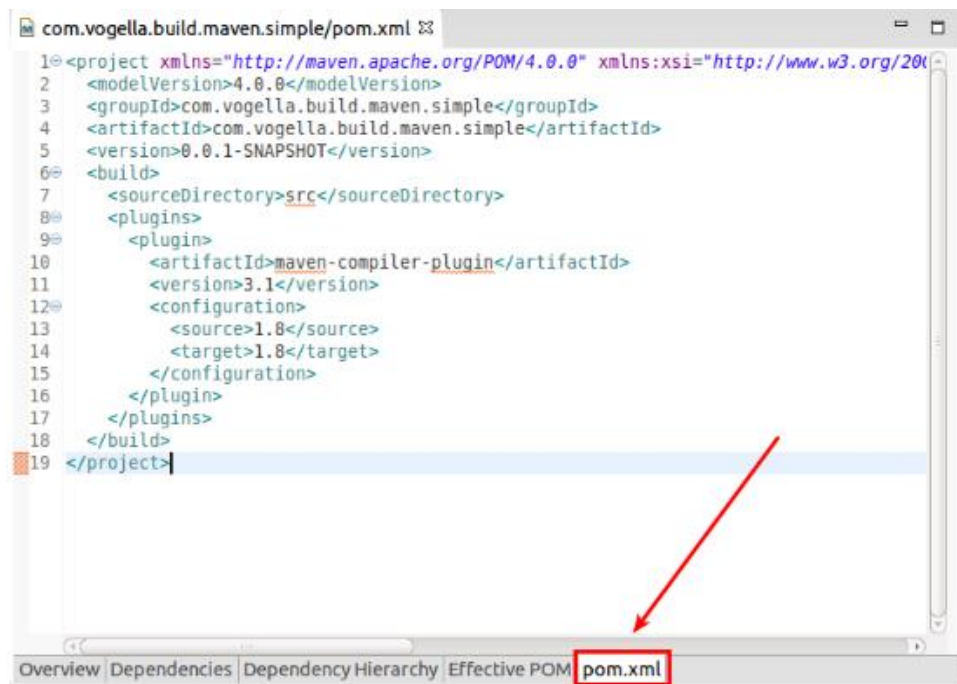


Рисунок 3.2. Вкладка pom.xml в редакторе

3.2 Установка и настройка Maven для Eclipse

Установка поддержки Maven для Eclipse (m2e)

Большинство установочников Eclipse уже включают в себя инструмент Maven. При его отсутствии вы можете установить этот инструмент через основное обновление своего релиза (Help→Install new Software). На следующем фрагменте указаны сайт обновления для релиза Neon и сайт обновления, поддерживаемый проектом m2e.

```
// Neon update site
http://download.eclipse.org/releases/neon

// Update site provided by m2e project
http://download.eclipse.org/technology/m2e/releases
```

Рисунок 3.2.1. Вкладка pom.xml в редакторе

При использовании Maven для Java проектов, вам будет необходим компонент m2e. Для веб-разработки на Java потребуется m2e-wtp.

Установка индексного файла Maven

По умолчанию, инструмент Maven не загружает индексный файл Maven для Eclipse IDE. Через индексный файл Maven вы сможете найти зависимости, выбрать их и добавить в ваш pom файл. Чтобы скачать индексный файл, выберете Windows->Preferences->Maven и щелкните на Download repository index updates on startup.

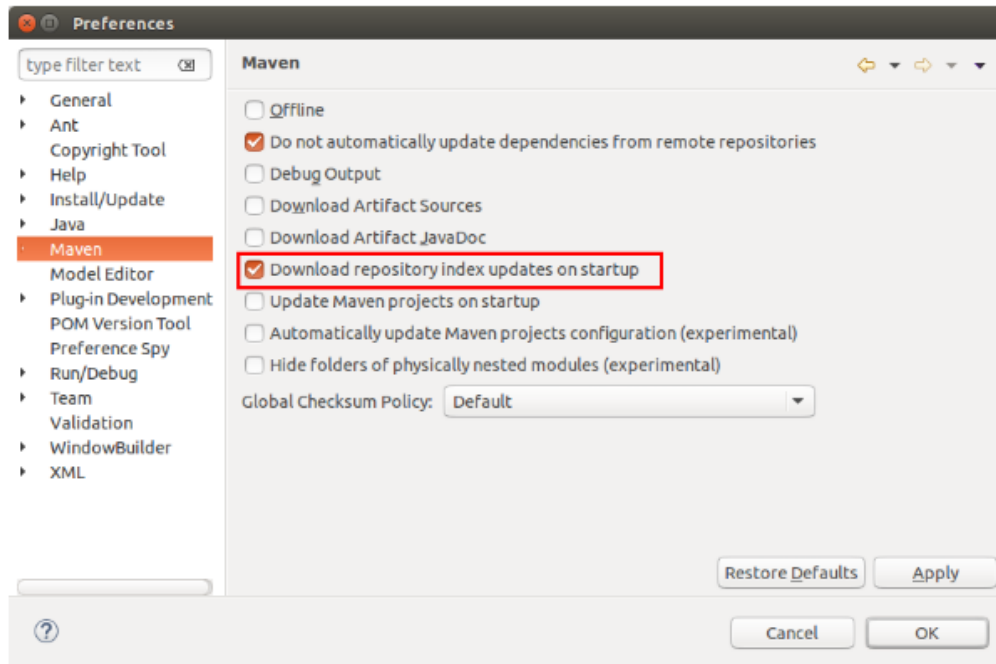


Рисунок 3.2.2. Загрузка индексного файла

После изменений в настройках, перезапустите Eclipse. При запуске Eclipse начнется загрузка индексного файла Maven. Вы можете убрать этот флаг, если хотите избежать использование сетевого трафика при каждом запуске Eclipse.

3.3 Задание: Создать новый проект с поддержкой Maven в Eclipse

В этом упражнении показан процесс создания нового проекта Maven в Eclipse.

Создание проекта Maven

Создайте новый проект Maven через File -> New -> Other... -> Maven -> Maven Project

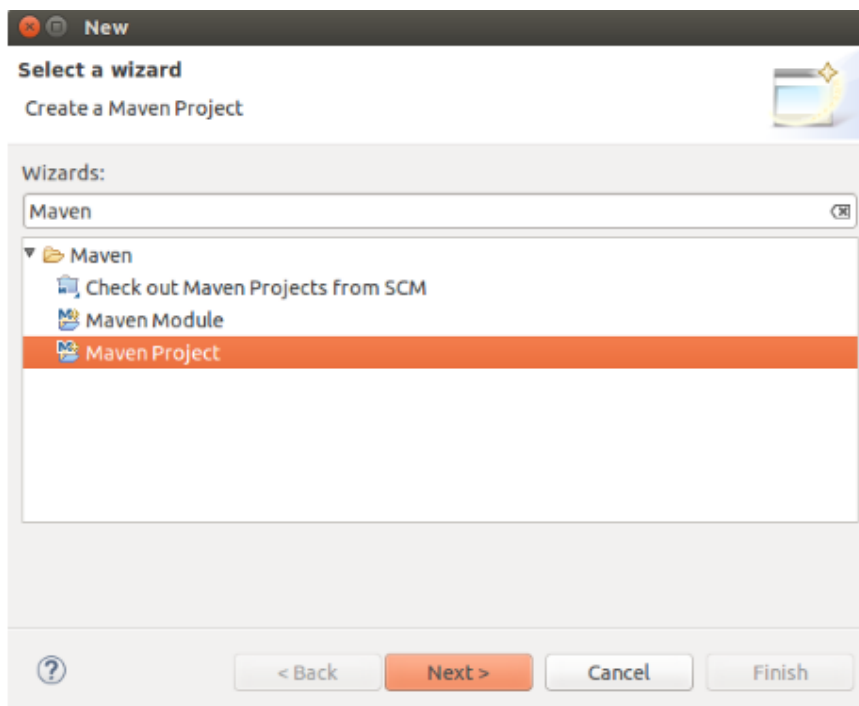


Рисунок 3.3.1. Создание проекта Maven

На первой странице мастера можно выбрать создание простого проекта (Create simple project), тогда вы пропускаете выбор артефакта. В данном случае будет использован артефакт из шаблона для создания проекта.

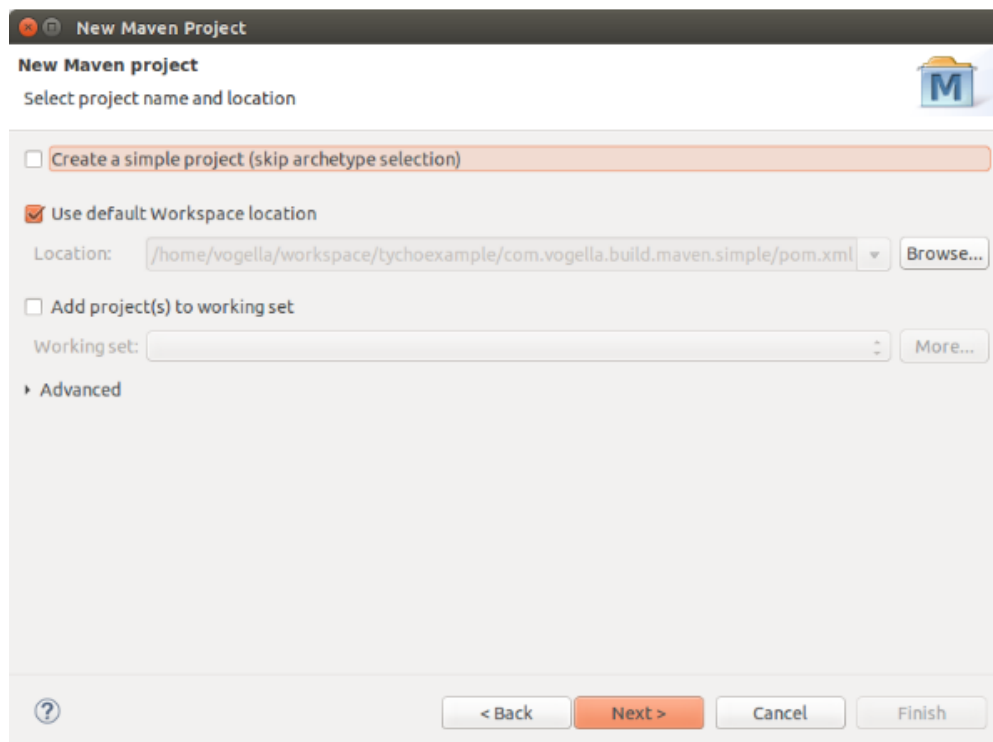


Рисунок 3.3.2. Создание простого проекта

Нажмите next, задайте фильтр «quickstart» и выберите запись the maven-archetype-quickstart. Это классический пример артефакта для создания проекта.

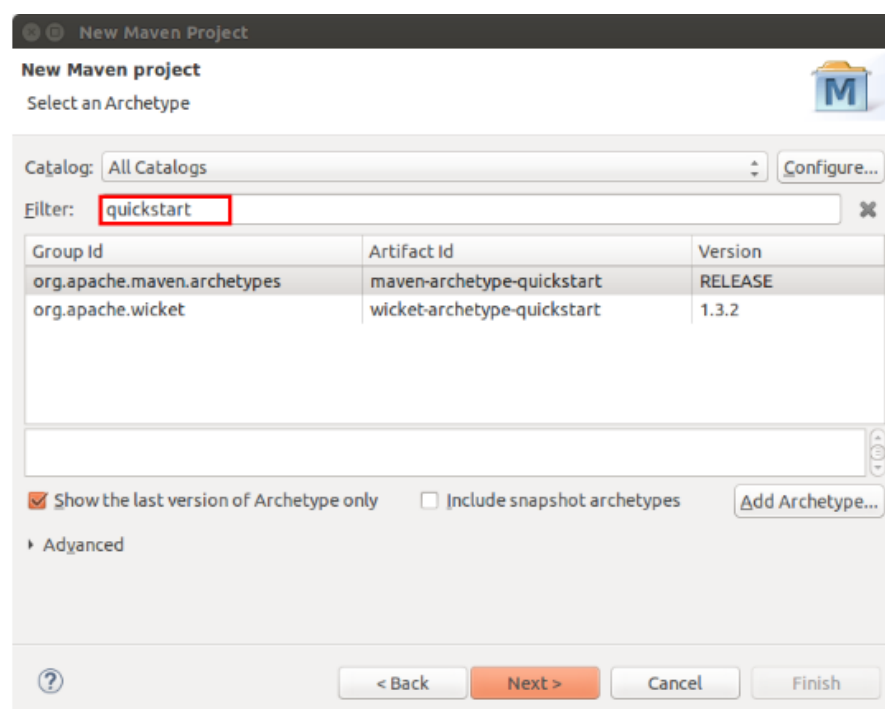


Рисунок 3.3.3. Выбор фильтра

На последней вкладке введите GAV проекта, как это показано на следующем изображении.

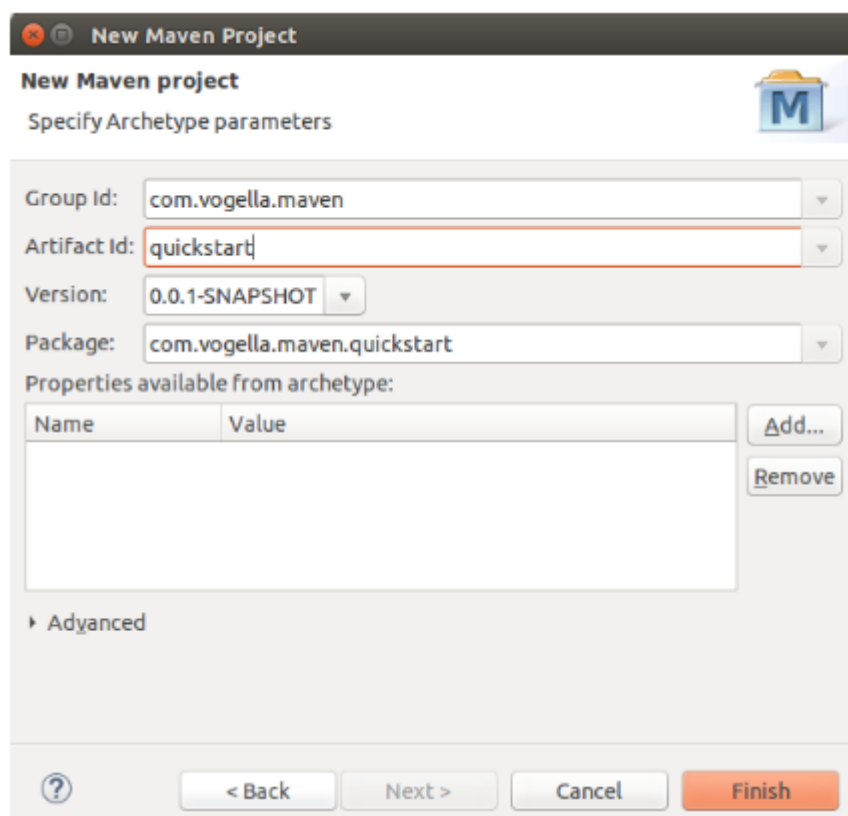


Рисунок 3.3.4. Выбор GAV проекта

Запуск сборки

Удостоверьтесь, что установка генератора работает правильно, запустив сборку. Для этого нажмите правой кнопкой мыши по файлу `pom.xml` и выберите `Run As → Maven build`.

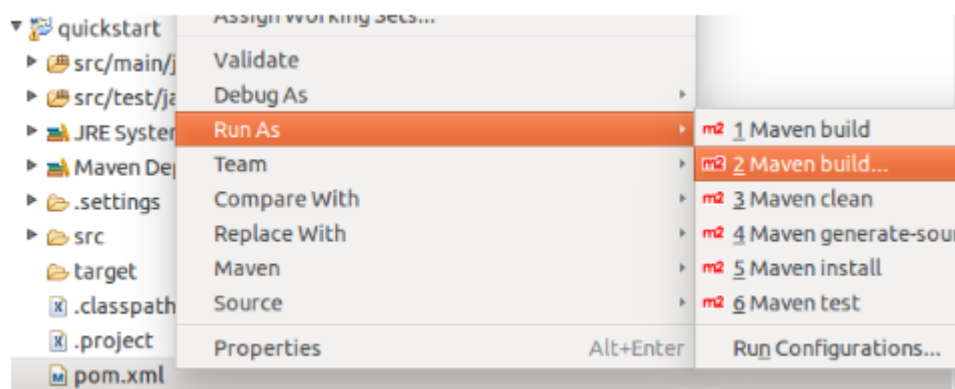


Рисунок 3.3.5. Запуск сборки

После чего откроется диалоговое окно, которое позволяет определить параметры для запуска. Введите `clean verify` в `Goals` и нажмите на кнопку `Run`.

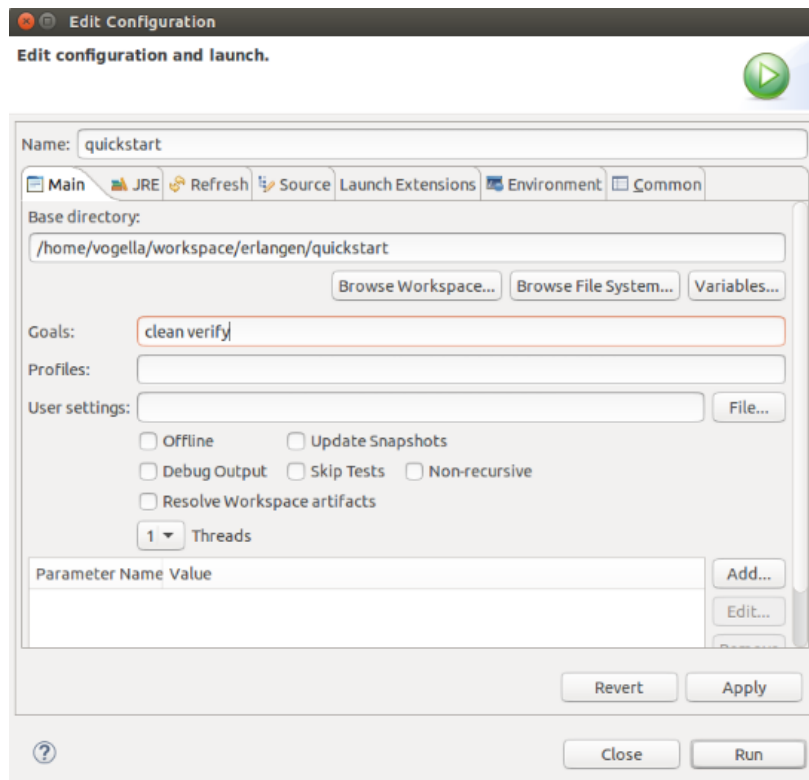


Рисунок 3.3.6. Определение параметров для запуска сборки

Добавление зависимостей в проект

Инструмент Eclipse Maven упрощает добавление зависимостей в путь к классам вашего проекта. Вы можете непосредственно добавляет его в файл pom или же воспользоваться вкладкой Dependencies (Зависимости) редактора pom.

Переключитесь на вкладку Dependencies и нажмите на кнопку Add.

В этом примере мы добавим Gson как зависимость. Для этого необходимо использовать GAV, который можно найти на сайте <http://search.maven.org>. Если индексный файл Maven загружен (См. [\[maven eclipseinstallation index\]](#)), вы также можете найти эту зависимость через диалоговое окно.

Использование библиотеки

Измените или создайте класс App.java в папке src/main/java. Этот класс будет использовать Gson. После того как Maven добавит его в путь к классу, он должен быть скомпилирован, и можно будет запустить класс через Eclipse.

```
package com.vogella.maven.lars;

import com.google.gson.Gson;

public class App
{
    public static void main( String[] args )
    {
        Gson gson = new Gson();
        System.out.println(gson.toJson("Hello World!") );
    }
}
```

Рисунок 3.3.7. Класс App

3.4 Задание: Добавить поддержку Maven в java-проект в Eclipse

В этом упражнении продемонстрирован способ преобразования Java-проекта в Maven-проект.

Создание Java-проекта

Создайте новый Java-проект с именем *com.vogella.build.maven.simple* в Eclipse.

Добавьте класс с именем Main. Этот класс должен содержать в себе метод main, который выводит «*Hello Maven!*» в командной строке.

```
package com.vogella.build.maven.simple;

public class Main {

    public static void main(String[] args) {
        System.out.println("Hello Maven!");
    }
}
```

Рисунок 3.4.1. Класс Main

Преобразование в Maven-проект

Выберете созданный вами проект, щелкните на нем правой кнопкой мышки и выберите Configure → Convert to Maven project... .

После этого будет создан файл pom.xml.

Сборка Maven

Нажмите правой кнопкой мышки на файле pom.xml и выберите Run As →Maven build

В поле Goal введите “clean install”.

Цели необходимо вводить вручную. Кнопка select не работает, при нажатии на нее открывается пустое диалоговое окно.

Нажмите на кнопку Finish. После чего будет запущена сборка, за которой можно проследить в консоли.

После окончания сборки нажмите F5 в активном окне с проектом для его обновления. В целевой папке вы можете увидеть артефакты сборки, например, JAR-файл.

3.5 Задание: Создать веб-проект на Java в Eclipse с использованием Maven

В этом упражнении показан способ создания веб-приложения в Eclipse с использованием Maven. Дальнейшие действия предполагают, что вы уже настроили Eclipse для создания веб-приложений.

Создание веб-проекта Maven.

Создайте новый Maven-проект с названием com.vogella.javaweb.maven.first через File ->New->Other->Maven->MavenProject. При выборе артефакта выберете the maven-archetype-webapp entry и кликните на кнопку Next.

Введите группу, артефакт и версию вашего нового компонента Maven.

В результате может возникнуть следующая ошибка: «The superclass "javax.servlet.http.HttpServlet" was not found on the Java Build Path». Чтоб исправить это, нажмите правой кнопкой мыши на вашем проекте и выберите Properties (свойства). На вкладке Targeted Runtimes выберите ваш веб-сервер, типа Tomcat

Построение проекта

Запустите сборку командой `mvn clean verify`. Убедитесь, что при сборке не возникло проблем.

Выполнение на сервере

Нажмите правой кнопкой мыши на вашем проекте и выберите Run as →Run on server menu.

Если вы откроете браузер, то вы увидите результат работы вашего веб-приложения.

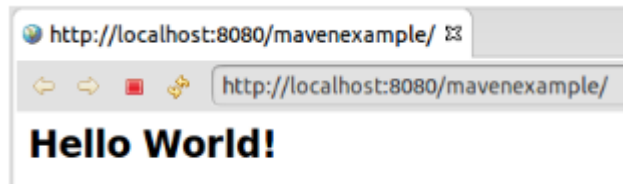


Рисунок 3.5.1. Результат работы веб-приложения

4. Тестирование программного обеспечения

В данном разделе объясняется модульное тестирование с JUnit 4.x и JUnit5. Объясняется создание тестов JUnit. Также охватывается использование Eclipse IDE для разработки тестов программных обеспечений (ПО).

4.1 Цель тестов ПО

Что такое тест для ПО?

Тест для ПО является необходимой частью при разработке ПО. Он проверяет, приводит ли разработанный код в ожидаемое состояние (тестирование состояния) или выполняется ли ожидаемая последовательность событий (тестирование поведения).

Зачем необходимо тестирование ПО?

Тестирование ПО помогает разработчику проверить логическую часть программы.

Выполнение тестов автоматически помогает выявить регрессию ПО, вызванные изменениями в исходном коде. Использование тестов позволяет вам продолжить разработку программы без необходимости выполнения долгой и сложной отладки.

4.2 Используемые термины

Модульные тесты и проверка блоков.

Модульный тест - часть кода, реализованная разработчиком, которая проверяет какие-либо функциональные возможности исходного кода.

Процентное соотношение кода, которое тестируется модульным тестом, является тестовым покрытием.

Модульный тест предназначен для небольшого фрагмента кода, например, метода или класса. Внешние зависимости не должны использоваться в модульных тестах. Их заменяют на тестовую реализацию или (макет) объект, созданные инфраструктурой тестирования.

Модульные тесты не подходят для тестирования сложного пользовательского интерфейса или взаимодействия компонентов. Для этого обычно используют интеграционные тесты.

Интеграционные тесты

Цель интеграционного теста - тестирование поведения компонента или взаимодействия между набором компонентов. Интеграционные тесты проверяют систему на то, работает ли она так, как это предполагается, поэтому при их использовании снижается необходимость в ручном тестировании.

Тестирование производительности

Тесты производительности используются для многократного сравнения компонентов программного обеспечения. Их цель - убедиться, что тестируемый код работает быстро, даже если он находится под высокой нагрузкой.

Тестирование состояния и поведения.

Тестирование поведения – это тест, который проверяет были ли вызваны определенные методы с правильными входными параметрами. Тест поведения не проверяет результат вызова метода.

Тестирование состояния – это проверка результата. Тестирование поведения – это проверка поведения тестируемого приложения.

Если вам необходимо протестировать алгоритмы или функциональность системы, то вам стоит использовать тест состояния, а не поведения. Обычная установка теста использует макеты и заглушки связанных классов, чтобы абстрагировать взаимодействия с этими классами. После чего вы проверяете состояние или поведение в зависимости от необходимости.

Фреймворки для тестирования

Для Java существуют несколько фреймворков тестирования. Самыми популярными считаются JUnit и TestNG.

Далее более подробно будет рассмотрен фреймворк Junit, а именно JUnit 4.x и JUnit 5.

Где разместить тест?

Обычно, модульные тесты создаются в отдельном проекте или в отдельной исходной папке для того, чтобы тестовый код находился отдельно от реального. В таких инструментах, как Maven и Gradle для этого используются следующие директории:

- Src/main/java – для классов java
- Src/test/java – для тестовых классов

Что необходимо тестировать?

Задача выбора объекта тестирования является наиболее сложной. Некоторые разработчики предлагают тестировать каждое утверждение в коде.

В любом случае программные тесты необходимо писать для критических и трудных мест вашего приложения. При внедрении новых функций надежный тестовый набор защитит ваш код от регрессии.

Не стоит тестировать тривиальный код. Например, бессмысленно писать тесты для методов `getter` и `setter`, которые присваивают значения соответствующим полям. Создание тестов для таких методов занимает много времени и не имеет смысла. JVM включает в себя тестовый случай для таких методов. Если вы разрабатываете приложения для конечных пользователей, то можете быть уверены в том, что данные методы работают правильно.

Если вы начинаете разработку тестов для уже созданного кода, который ранее не тестировался, то рекомендуется начать писать тесты для кода, в котором ранее возникало больше всего ошибок. Таким образом вы сфокусируетесь на критических участках вашего приложения.

4.3 Использование JUnit

Фреймворк JUnit

JUnit – это фреймворк, который использует аннотации для определения тестовых методов. JUnit является проектом с открытыми исходными кодами, размещенным на Github.

Как определить является ли метод тестовым в JUnit?

Тест JUnit – это метод, содержащийся в классе, который используется только для тестирования. Его называют тестовым классом. Чтобы определить, что определенный метод является тестовым можно по аннотации `@Test`.

Для сравнения ожидаемого результата с фактическим используется класс `Assert` с набором методов утверждений. При его использовании вы передаете утверждениям класса значения, которые нужно проверить.

Пример JUnit теста.

Следующий код иллюстрирует тест, реализованный с JUnit 5-ой версии. Данный тест подразумевает, что в вашем проекте есть класс `MyClass` с методом `multiply(int, int)`.

```
import static org.junit.jupiter.api.Assertions.assertEquals;

import org.junit.jupiter.api.Test;

public class MyTests {

    @Test
    public void multiplicationOfZeroIntegersShouldReturnZero() {
        MyClass tester = new MyClass(); // MyClass is tested

        // assert statements
        assertEquals("10 x 0 must be 0", 0, tester.multiply(10, 0));
        assertEquals("0 x 10 must be 0", 0, tester.multiply(0, 10));
        assertEquals("0 x 0 must be 0", 0, tester.multiply(0, 0));
    }
}
```

Рисунок 4.3.1. Пример JUnit теста

JUnit соглашение об именах.

Существуют несколько потенциальных соглашений об именах для тестов JUnit. Широко-используемое решение для классов – использовать суффикс `Test` в конце названия тестовых классов.

Как правило, тест называют так, чтобы его название давало некоторое понимание о том, что он делает. Это позволяет избежать чтения фактической реализации.

Еще одним из соглашений является использование “should” в имени метода тестирования. Например, “ordersShouldBeCreated”

(очереди Должны Быть Созданы) или “menuShouldGetActive” (меню Должно Стать Активным). Данное действие поясняет, что должно произойти в случае, если метод выполнен.

Существует и другой подход — использование “Given[ExplainYourInput]When[WhatIsDone]Then[ExpectedResult]” в названии метода тестирования.

Соглашение об именах JUnit для Maven

Если вы тестируете систему сборки Maven, то необходимо использовать суффикс “Test” для тестовых классов. Система сборки Maven автоматически включает такие классы в её области тестирования.

Запуск теста через командную строку

Вы также можете запустить тест JUnit вне вашего IDE через стандартный код Java. Системы сборки такие, как Apache Maven или Gradle в сочетании с Continuous Integration Server (как Jenkins) могут использоваться для выполнения автоматических тестов.

Класс `org.junit.runner.JUnitCore` предоставляет метод `runClasses()`. Этот метод позволяет выполнить один или несколько тестовых классов. В качестве возвращаемого параметра вы получите объект типа `org.junit.runner.Result`. Этот объект может быть использован для восстановления информации о тестах.

Класс, продемонстрированный в следующем фрагменте, использует тестовый класс `MyClassTest`. Полученные сбои этот класс выводит в консоль.

```
package de.vogella.junit.first;

import org.junit.runner.JUnitCore;
import org.junit.runner.Result;
import org.junit.runner.notification.Failure;

public class MyTestRunner {
    public static void main(String[] args) {
        Result result = JUnitCore.runClasses(MyClassTest.class);
        for (Failure failure : result.getFailures()) {
            System.out.println(failure.toString());
        }
    }
}
```

Рисунок 4.3.2. Использование `org.junit.runner.JUnitCore`

Этот класс может быть выполнен как любая другая программа на Java из командной строки. Для этого нужно добавить JAR-файл библиотеки JUnit в путь к классу.

Порядок выполнения тестов

JUnit предоставляет возможность, чтобы все тестовые методы можно было выполнить в произвольном порядке. Хорошо написанный тестовый код не должен принимать любой порядок, то есть тесты не должны зависеть от других тестов.

Начиная с JUnit 4.11, по умолчанию используется детерминированный, но не предсказуемый, порядок выполнения тестов.

Вы можете использовать аннотацию для того, чтобы тестовые методы были отсортированы по имени метода в лексикографическом порядке. Чтобы активировать эту возможность, аннотируйте ваш тестовый класс аннотацией `@FixMethodOrder(MethodSorters.Name_ASCENDING)`. Вы также можете явно задать значения по умолчанию, используя параметр `MethodSorters.DEFAULT` в этой аннотации. Также можно использовать `MethodSorters.JVM`, который использует значения по умолчанию JVM, которые в свою очередь могут варьироваться от запуска к запуску.

4.4 Использование JUnit 4

Определение тестовых методов

JUnit использует аннотации для маркировки тестовых методов и конфигурирования их. Следующая таблица дает общее представление о наиболее важных аннотациях в JUnit для версий 4.x и 5.x.

Таблица 4.4.1. Аннотации JUnit 4

JUnit 4	Значение
<code>import org.junit.*</code>	Импортирует операторы для использования следующих аннотаций.

JUnit 4	Значение
@Test	Идентифицирует (определяет) метод как тестовый.
@Before	Выполняется перед каждым тестом. Используется для подготовки среды к тесту.
@After	Выполняется после каждого теста. Используется для очистки среды после теста (удаляет временные данные, восстанавливает значения по умолчанию).
@BeforeClass	Выполняется один раз перед началом всех тестов. Метод используется для выполнения инициализации, которую нужно выполнить только один раз, например, создание соединения с базой данных. Такие методы должны быть определены как static для работы с JUnit
@AfterClass	Выполняется один раз, после выполнения всех тестов. Используется для деинициализации, например, закрытие соединения с базой данных. Такие методы должны быть определены как static для работы с JUnit
@Ignore or @Ignore("Why disabled")	Игнорирование тестового метода. Используется при изменении основного кода, а соответствующие изменения теста не

JUnit 4	Значение
	были сделаны. Или в случае, если время выполнения теста занимает много времени. Рекомендуется использовать данный метод с описанием, почему тест не используется.
@Test (expected = Exception.class)	Выдает ошибку, если метод не выбрасывает исключение, принадлежащее классу.
@Test(timeout=100)	Выдает ошибку, если время выполнения тестового метода превышает указанный параметр timeout.

Операторы контроля

JUnit предоставляет статические методы для теста каких-либо состояний через класс Assert. Операторы данного класса обычно начинаются с ключевого слова assert. Эти операторы выдают сообщение об ошибке при обнаружении некорректных данных. Метод assert сравнивает фактическое значение, возвращаемое тестом с ожидаемым значением. Обычно приводит к завершению программы при отрицательном результате теста.

В следующей таблице приведены методы и их описания. Переменные в квадратных скобках не обязательны и имеют тип string.

Таблица 4.4.2. Операторы контроля

Оператор	Описание
fail(message)	Генерирует исключение. Используется для проверки на то, достигнута ли определенная часть кода.

Оператор	Описание
	Передаваемый объект необязателен.
<code>assertTrue([message, boolean condition])</code>	Проверяет истинность утверждения.
<code>assertFalse([message, boolean condition])</code>	Проверяет ложность утверждения.
<code>assertEquals([message, expected, actual])</code>	<p>Проверяет, совпадают ли ожидаемый и полученный результаты.</p> <p>Примечание: при проверке массивов сравниваются их ссылки.</p>
<code>assertEquals([message, expected, actual, tolerance])</code>	<p>Проверяет, совпадают ли ожидаемый и полученный результаты.</p> <p>Параметр <code>tolerance</code> - это число десятичных знаков, которые должны совпадать.</p>
<code>assertNull([message, object])</code>	Проверяет, является ли объект пустым.
<code>assertNotNull([message, object])</code>	Проверяет, является ли объект не нулевым.
<code>assertSame([message, expected, actual])</code>	Проверяет, ссылаются ли оба объекта на один и тот же объект.
<code>assertNotSame([message, expected, actual])</code>	Проверяет, не ссылаются ли оба объекта на один и тот же объект.

Набор тестов JUnit

Если имеется несколько тестовых классов, то можно объединить их в набор тестов. При выполнении набора тестов выполняются все тестовые классы в определенном порядке. Набор тестов также может включать в себя другие наборы тестов.

В следующем примере продемонстрирован вариант использования набора тестов. Имеется два набора тестов (MyClassTest и MySecondClassTest). В случае если вы хотите добавить дополнительный тестовый класс, тогда добавьте его к оператору `@Suite.SuiteClasses`

```
package com.vogella.junit.first;

import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;

@RunWith(Suite.class)
@SuiteClasses({
    MyClassTest.class,
    MySecondClassTest.class })

public class AllTests {

}
```

JAVA

Рисунок 4.4.1. Использование набора тестов

Отключение тестов

Аннотация `@Ignore` позволяет статически игнорировать тест. В качестве альтернативы, вы можете использовать `Assume.assumeFalse` или `Assume.assumeTrue` для определения результата для теста. `Assume.assumeFalse` отмечает тест недействительным, если утверждение истинно. `Assume.assumeTrue` оценивает тест недействительным, если утверждение ложно. Например, следующая операция отключает тест в ОС Linux:

```
Assume.assumeFalse(System.getProperty("os.name").contains("Linux"));
```

JAVA

Рисунок 4.4.2. Игнорирование теста

Параметризованный тест

JUnit позволяет вам использовать параметры в тестовом классе. Этот класс может содержать один тестовый метод, который выполняется с разными параметрами.

Вы помечаете тестовый класс как параметризованный тест с аннотацией `@RunWith(Parameterized.class)`

Такой тестовый класс должен содержать статический метод с аннотацией `@Parameters`. Этот метод создает и возвращает коллекцию массивов. Каждый элемент в этой коллекции используется как аргумент для тестового метода.

Вы также можете использовать аннотацию для общедоступных полей, чтобы получить значения теста, введенные в тест.

В качестве альтернативы аннотации `@Parameter` вы можете использовать конструктор, в котором вы сохраняете значения для каждого теста. Количество элементов в каждом массиве, предоставляемых методом с аннотацией `@Parameters`, должно совпадать с количеством параметров в конструкторе класса. Класс создается для каждого параметра, и тестовые значения передаются через конструктор к классу.

Если вы запустите на выполнение этот тестовый класс, то тестовый метод будет выполняться с каждым указанным параметром.

Правила Junit

Правила представляют собой класс, который следит за выполнением теста и реагирует на изменение состояний. Для каждого состояния в классе имеется соответствующий метод. Для правил используется аннотация `@Rule`. Данная возможность придает вашим тестам гибкость. Вы, например, можете указать какое сообщение об исключении вы ожидаете во время выполнения тестового кода.

```

package de.vogella.junit.first;

import org.junit.Rule;
import org.junit.Test;
import org.junit.rules.ExpectedException;

public class RuleExceptionTesterExample {

    @Rule
    public ExpectedException exception = ExpectedException.none();

    @Test
    public void throwsIllegalArgumentExceptionIfIconIsNull() {
        exception.expect(IllegalArgumentException.class);
        exception.expectMessage("Negative value not allowed");
        ClassToBeTested t = new ClassToBeTested();
        t.methodToBeTest(-1);
    }
}

```

Рисунок 4.4.3. Использование правил

В JUnit уже заложены несколько полезных реализаций правил. Например, класс `TemporaryFolder` создает временные файлы и папки, которые автоматически будут удалены после выполнения теста.

В следующем фрагменте продемонстрирован пример использования `TemporaryFolder`.

```

package de.vogella.junit.first;

import static org.junit.Assert.assertTrue;

import java.io.File;
import java.io.IOException;

import org.junit.Rule;
import org.junit.Test;
import org.junit.rules.TemporaryFolder;

public class RuleTester {

    @Rule
    public TemporaryFolder folder = new TemporaryFolder();

    @Test
    public void testUsingTempFolder() throws IOException {
        File createdFolder = folder.newFolder("newfolder");
        File createdFile = folder.newFile("myfilefile.txt");
        assertTrue(createdFile.exists());
    }
}

```

Рисунок 4.4.4. Использование класса `TemporaryFolder`

Реализация пользовательских правил JUnit

Чтобы создать правило, необходимо реализовать интерфейс `TestRule`. В этом интерфейсе должен быть определен метод `apply(Statement, Description)`, который возвращает объект типа `org.junit.rules.TestRule`. `Statement` представляет собой тесты со временем выполнения JUnit и `Statement#evaluate()`. Параметр `Description` описывает индивидуальный тест. Это позволяет узнать информацию о тесте.

Следующий код представляет собой простой пример добавления оператора журнала в приложение для Android до и после выполнения теста.

```
package testing.android.vogella.com.asyncTask;

import android.util.Log;

import org.junit.rules.TestRule;
import org.junit.runner.Description;
import org.junit.runners.model.Statement;

public class MyCustomRule implements TestRule {
    private Statement base;
    private Description description;

    @Override
    public Statement apply(Statement base, Description description) {
        this.base = base;
        this.description = description;
        return new MyStatement(base);
    }

    public class MyStatement extends Statement {
        private final Statement base;

        public MyStatement(Statement base) {
            this.base = base;
        }

        @Override
        public void evaluate() throws Throwable {
            System.
            Log.w("MyCustomRule",description.getMethodName() + "Started" );
            try {
                base.evaluate();
            } finally {
                Log.w("MyCustomRule",description.getMethodName() + "Finished");
            }
        }
    }
}
```

Рисунок 4.4.5. Создание пользовательского правила

Для того, чтобы использовать это правило, добавьте поле с аннотацией `@Rule` в ваш тестовый класс.

```
@Rule
public MyCustomRule myRule = new MyCustomRule();
```

Рисунок 4.4.6. Использование пользовательского правила

Категории

Можно разбить тесты по категориям с использованием аннотации `@Category`. В следующем примере используется JUnit версии 4.8.

```
public interface FastTests { /* category marker */
}

public interface SlowTests { /* category marker */
}

public class A {
    @Test
    public void a() {
        fail();
    }

    @Category(SlowTests.class)
    @Test
    public void b() {
    }
}

@Category({ SlowTests.class, FastTests.class })
public class B {
    @Test
    public void c() {
    }
}

@RunWith(Categories.class)
@IncludeCategory(SlowTests.class)
@SuiteClasses({ A.class, B.class })
// Note that Categories is a kind of Suite
public class SlowTestSuite {
    // Will run A.b and B.c, but not A.a
}

@RunWith(Categories.class)
@IncludeCategory(SlowTests.class)
@ExcludeCategory(FastTests.class)
@SuiteClasses({ A.class, B.class })
// Note that Categories is a kind of Suite
public class SlowTestSuite {
    // Will run A.b, but not A.a or B.c
}
```

Рисунок 4.4.7. Использование категорий

4.5 Поддержка Eclipse для JUnit 4

Создание тестов JUnit

Вы можете написать тесты JUnit вручную, но Eclipse предоставляет возможность создания тестов JUnit с использованием мастеров.

Например, для создания теста JUnit или тестового класса необходимо сделать следующие действия (дальнейшая инструкция подразумевает, что вы уже добавили библиотеку junit.jar к маршруту построения в Eclipse):

1. откройте Мастер New (File→New→JUnit Test Case);
2. введите имя вашего тестового класса в поле Name;
3. нажмите на Finish.

Запуск тестов JUnit

Eclipse IDE также предоставляет поддержку для выполнения ваших тестов в интерактивном режиме.

Чтобы выполнить тест, нажмите правой кнопкой мыши на тестовом классе и выберите Run-as→JUnit Test.

Эта команда запустит JUnit, после чего будут выполнены все тестовые методы в классе.

В Eclipse есть горячие клавиши alt+shift+x для запуска тестов выбранном классе.

Для запуска определенного теста, наведите курсор на имя тестового класса и нажмите на горячие клавиши.

Для отображения результата теста JUnit Eclipse использует представление JUnit, которое выводит результаты теста. В этом представлении вы также можете выбрать конкретные модульные тесты, для этого нажмите на них правой кнопкой мыши и выберите Run для повторного выполнения.

По умолчанию это представление показывает все тесты. Но вы можете настроить его так, чтобы в нем отображались только неудавшиеся тесты.

Вы также можете настроить представление так, чтобы оно было активным только в случае возникновения ошибки в тесте.

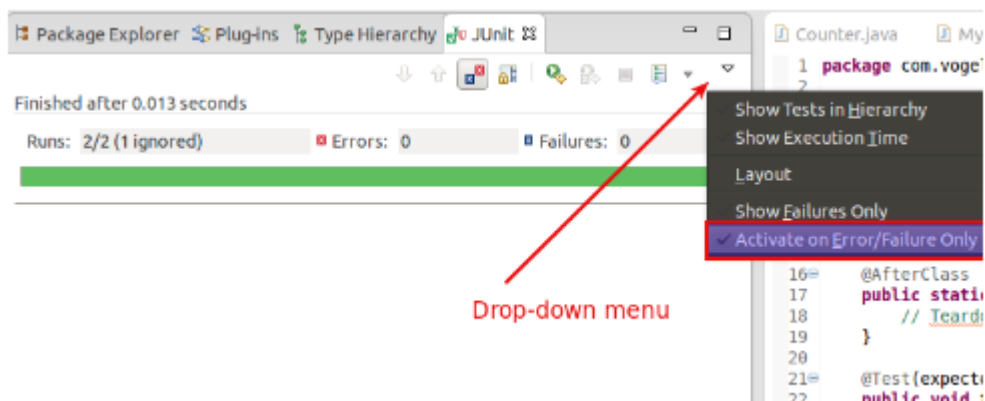


Рисунок 4.5.1. Запуск теста в интерактивном режиме

ПРИМЕЧАНИЕ: Eclipse создает конфигурации выполнения для тестов. Вы можете увидеть и настраивать их через Run → Run Configurations...

Получение списка неудачных тестов

Чтобы получить список неудачных тестов, нажмите правой кнопкой мыши на результате теста и выберите Copy Failure List. Неудавшиеся тесты будут скопированы в буфер обмена.

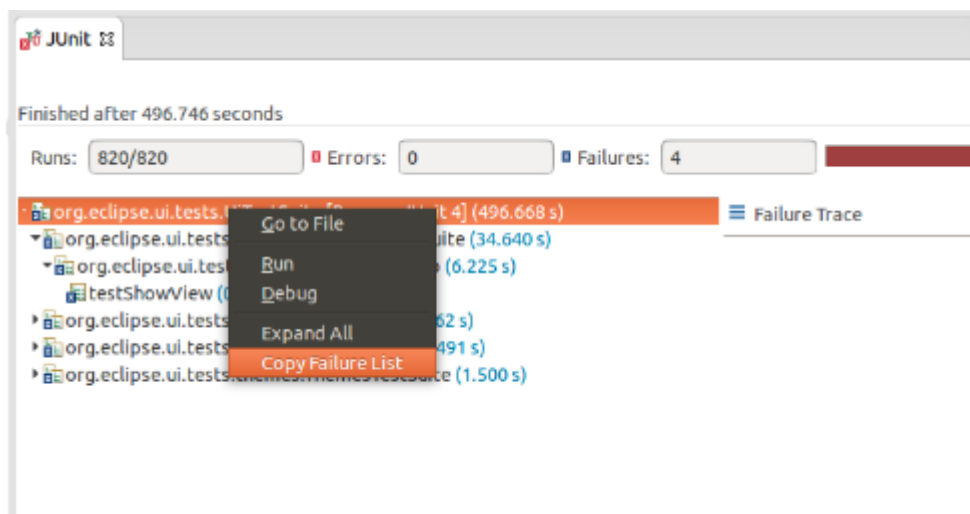


Рисунок 4.5.2. Получение списка неудачных тестов

Статический импорт JUnit

Статический импорт – это свойство, которое позволяет использовать поля и методы, определенные в классе как `public static`, без указания принадлежности к классу.

Операторы `assert` JUnit обычно определяется как `public static` для того, чтобы разработчик мог писать короткие тестовые утверждения. В следующем фрагменте продемонстрировано использование метода `assertEquals` с использованием и без использования статического импорта.

```
// without static imports you have to write the following statement
Assert.assertEquals("10 x 5 must be 50", 50, tester.multiply(10, 5));

// alternatively define assertEquals as static import
import static org.junit.Assert.assertEquals;

// more code

// use assertEquals directly because of the static import
assertEquals("10 x 5 must be 50", 50, tester.multiply(10, 5));
```

Рисунок 4.5.3. Статический импорт

Мастер создания набора тестов

Вы можете создать набор тестов в Eclipse. Для этого выберите необходимые пакеты, нажмите правой кнопкой мыши на них и выберите `New→Other...→JUnit→JUnit Test Suite`.

Тестирование исключений

Аннотация `@Test (expected = Exception.Class)` позволяет тестировать только одно исключение. Для тестирования нескольких исключений, вы можете поступить следующим образом:

```
try {
    mustThrowException();
    fail();
} catch (Exception e) {
    // expected
    // could also check for message of exception, etc.
}
```

Рисунок 4.5.4. Тестирование исключений

Тесты плагинов JUnit

Тесты плагинов JUnit используются для написания модульных тестов для ваших плагинов. Эти тесты выполняются специальными исполнителем тестов, который отдельно запускает другой экземпляр Eclipse в виртуальной машине (VM). Тестовые методы выполняются в этом экземпляре.

4.6 Установка JUnit

Использование JUnit с Gradle

Чтоб использовать JUnit для сборки Gradle, добавьте зависимость testCompile в файл сборки.

```
apply plugin: 'java'

dependencies {
    testCompile 'junit:junit:4.12'
}
```

Рисунок 4.6.1. Добавление зависимости в файл сборки Gradle

Использование JUnit с Maven

Чтобы использовать JUnit для сборки Maven, добавьте следующую зависимость в pom-файл.

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.12</version>
</dependency>
```

Рисунок 4.6.2. Добавление зависимости в файл сборки Maven

Использование JUnit в составе Eclipse

Eclipse IDE поставляется с версией JUnit. Если Вы используете Eclipse, никаких дополнительных загрузок не требуется.

Загрузка библиотеки JUnit

Если вы собираетесь использовать библиотеку Junit более детально, необходимо загрузить JUnit4.x.jar с веб-сайта JUnit. Загружаемый файл содержит junit-4.*.jar. Добавьте эту библиотеку в Ваш Java проект и пропишите путь к классу: «<http://junit.org/>».

4.7 Настройка Eclipse для использования статического импорта

Eclipse IDE не может создать соответствующий оператор `static import` автоматически.

Вы можете настроить Eclipse IDE так, чтобы использовать во время завершения кода вставку обычного метода JUnit, который вызовет или добавит статический импорт автоматически. Для этого откройте Preferences через Window ->Preferences и выберите Java ->Editor->Content Assist -> Favorites.

Нажмите на кнопку New Type, чтобы добавить следующие классы:

- `org.junit.Assert;`
- `org.hamcrest.Core;`
- `org.hamcrest.Matchers.`

Благодаря этому методы `assertTrue`, `assertFalse` и `assertEquals` будут доступны напрямую в Content Assists.

Теперь Вы можете использовать Content Assists (клавиши быстрого доступа CTRL+ПРОБЕЛ), чтобы добавить метод и импорт.

4.8 Задание: Создать проект и протестировать его с использованием JUnit

Подготовка проекта

Создайте новый проект под названием `com.vogella.junit.first`. Создайте новую папку источника `test`. Для этого нажмите правой кнопкой мыши на вашем проекте, выберите `Properties` и выберите `Java` → `Build Path`. Откройте вкладку `Source`.

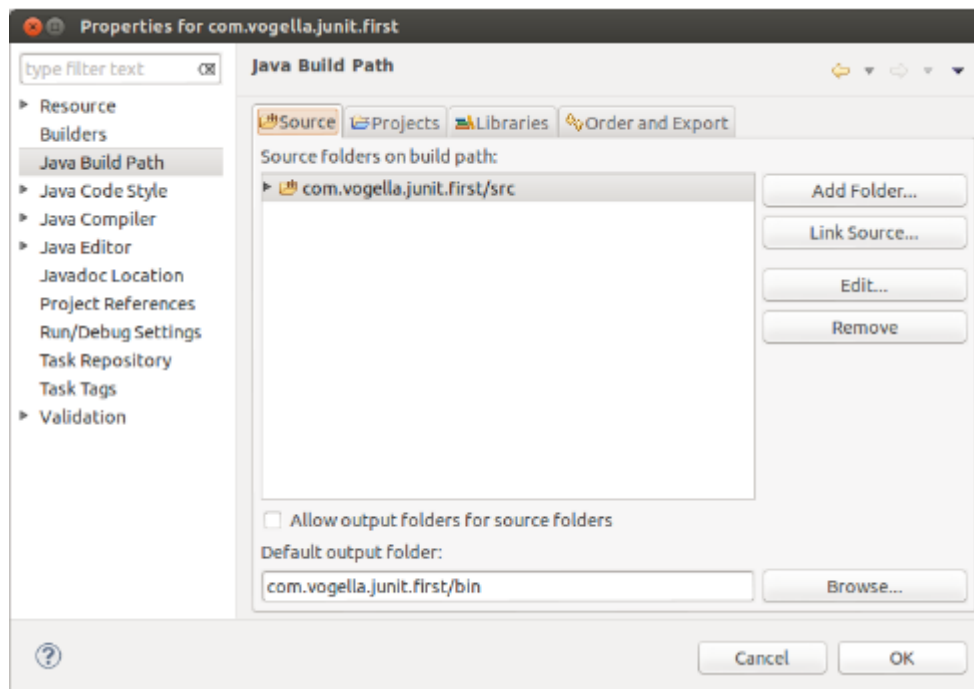


Рисунок 4.8.1. Вкладку Source

Нажмите на кнопку `Add Folder`. Далее выберите `Create New Folder`. Назовите папку `test`.

Результат продемонстрирован на следующем скриншоте.

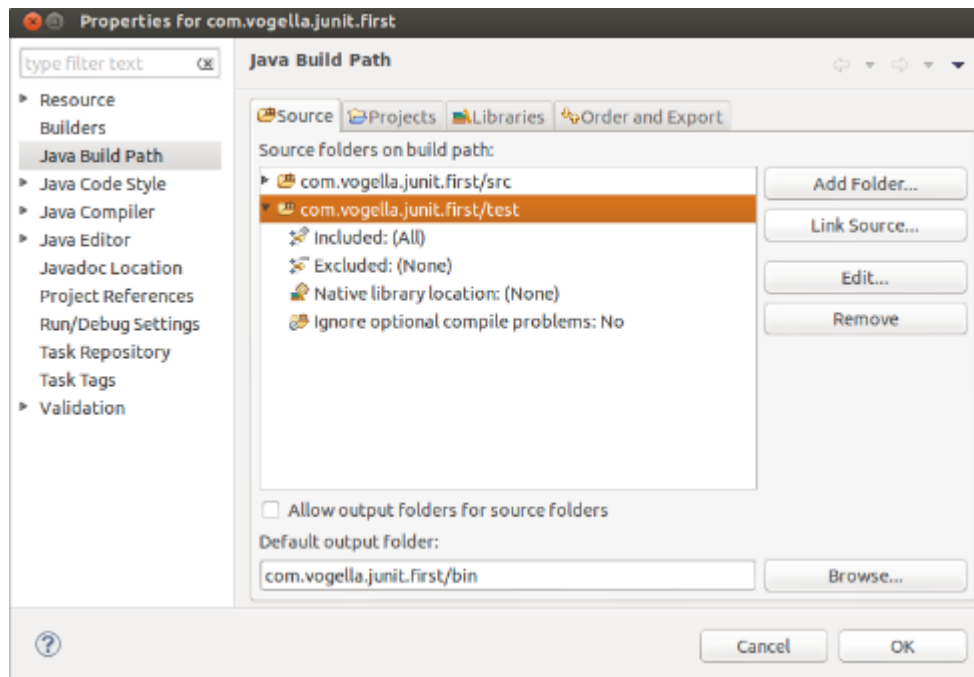


Рисунок 4.8.2. Создание новой папки на вкладку Source

Примечание: Вы можете добавить новую папку источника, нажав правой кнопкой мыши на проекте и выбрав New → Source Folder.

Создание Java-класса

В папке src, создайте модуль com.vogella.junit.first и следующий класс.

```
package com.vogella.junit.first;

public class MyClass {
    public int multiply(int x, int y) {
        // the following is just an example
        if (x > 999) {
            throw new IllegalArgumentException("X should be less than 1000");
        }
        return x / y;
    }
}
```

Рисунок 4.8.3. Исходный код модуля com.vogella.junit.first

Создание JUnit-теста

Нажмите правой кнопкой мыши на вашем новом классе в Package Explorer и выберите New→JUnit Test Case.

Убедитесь, что выбран флаг New JUnit 4 test, и задайте папке имя test.

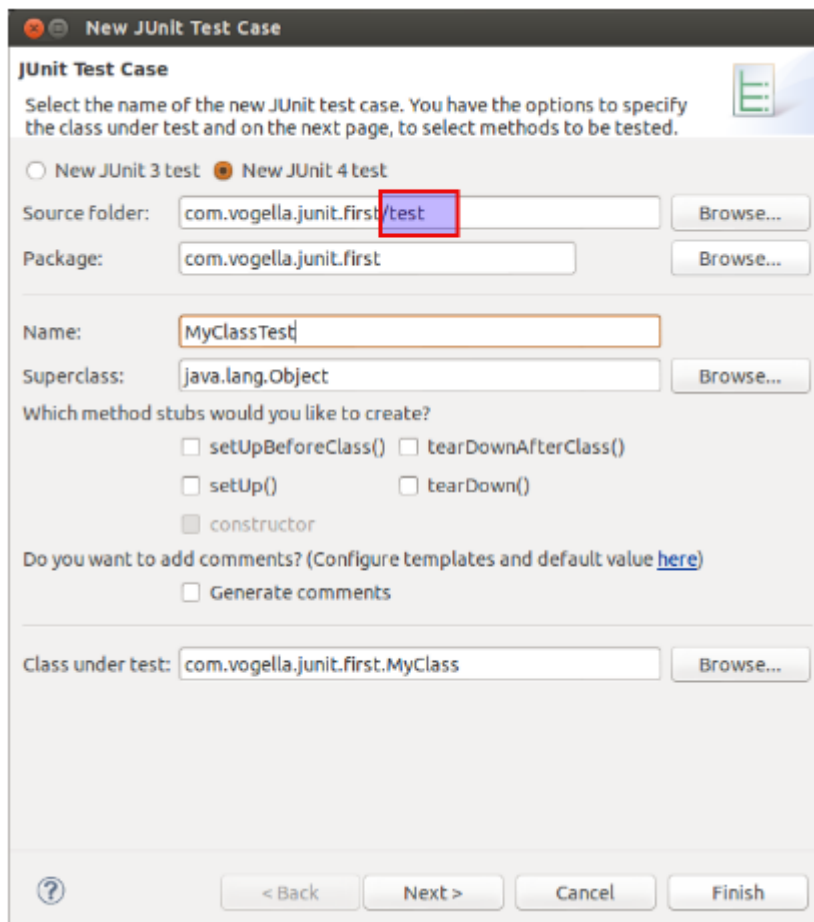


Рисунок 4.8.4. Создание JUnit-теста

Нажмите на кнопку Next и выберите методы, которые хотите протестировать.

Если путь к библиотеке JUnit не прописан, Eclipse подскажет вам добавить его. Добавьте JUnit к вашему проекту, как это показано ниже.

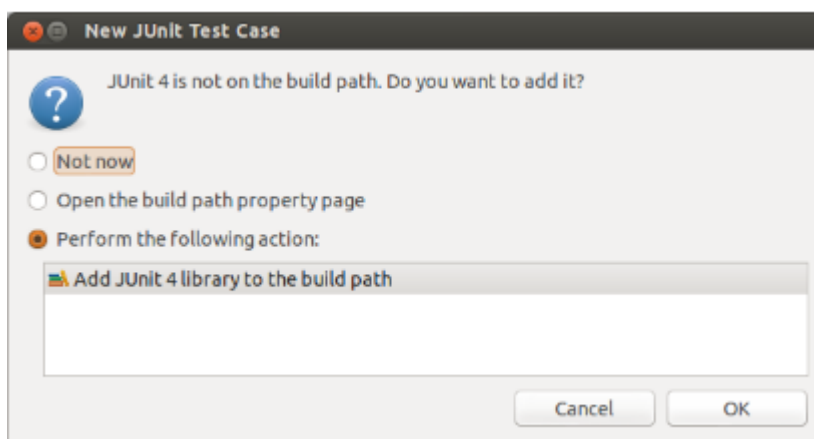


Рисунок 4.8.5. Добавление пути к проекту

Создайте тест со следующим кодом.

```

package com.vogella.junit.first;

import static org.junit.Assert.assertEquals;

import org.junit.AfterClass;
import org.junit.BeforeClass;
import org.junit.Test;

public class MyClassTest {

    @Test(expected = IllegalArgumentException.class)
    public void testExceptionIsThrown() {
        MyClass tester = new MyClass();
        tester.multiply(1000, 5);
    }

    @Test
    public void testMultiply() {
        MyClass tester = new MyClass();
        assertEquals("10 x 5 must be 50", 50, tester.multiply(10, 5));
    }
}

```

Рисунок 4.8.6. Исходный код теста

Выполнение теста в Eclipse

Нажмите правой кнопкой мыши на вашем новом тестовом классе и выберите Run-As->JUnitTest.



Рисунок 4.8.7. Запуск теста на выполнение

Результат тестов отображен в представлении JUnit. В данном примере один тест должен быть успешным и один тест - ошибочным. Полученная ошибка выделена красной полосой.

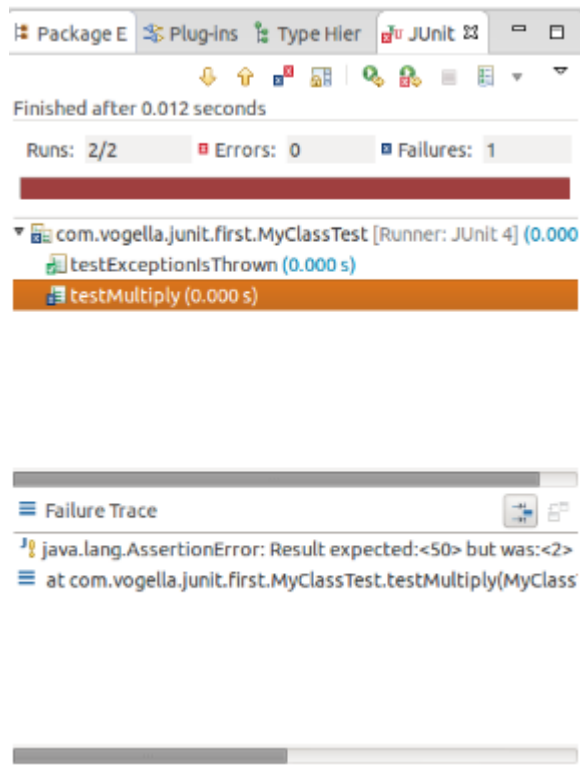


Рисунок 4.8.8. Результат тестирования

Тест не удался, потому что класс `multiplier` работает некорректно. Он выполняет операцию деления вместо умножения. Исправьте ошибку и перезапустите тест, чтобы в результате полоса имела зеленый цвет.

4.9 Объект Mocking

Модульный тест может использовать объект `mocking`. В этом случае реальный объект заменяется объектом, у которого поведение для теста предопределено.

Существует несколько фреймворков доступных для `mocking`. Чтоб узнать больше о фреймворках `mock`, посмотрите [Mockito Tutorial](#).

4.10 Обзор JUnit 5

JUnit 5 — последний на сегодняшний день выпуск JUnit, и он до сих находится на этапе разработки. JUnit 5 состоит из нескольких компонентов.

- JUnit platform — основа платформы, которая позволяет запускать различные тестовые фреймворки на JVM;
- JUnit Jupiter — тестовый фреймворк JUnit 5, который запускается платформой JUnit;

-JUnit Vintage — устаревший тестовый движок, который предоставляет поддержку старых тестов.

Использование JUnit 5 с Gradle

```
buildscript {
    repositories {
        mavenCentral()
        // The following is only necessary if you want to use SNAPSHOT releases.
        // maven { url 'https://oss.sonatype.org/content/repositories/snapshots' }
    }
}

dependencies {
    classpath 'org.junit.platform:junit-platform-gradle-plugin:1.0.0-M4'
}

repositories {
    mavenCentral()
}

apply plugin: 'java'
apply plugin: 'eclipse'
apply plugin: 'org.junit.platform.gradle.plugin'

dependencies {
    testCompile("org.junit.jupiter:junit-jupiter-api:5.0.0-M4")
    testRuntime("org.junit.jupiter:junit-jupiter-engine:5.0.0-M4")
    // to run JUnit 3/4 tests:
    testCompile("junit:junit:4.12")
    testRuntime("org.junit.vintage:junit-vintage-engine:4.12.0-M4")
}
```

Рисунок 4.10.1. Использование JUnit 5 с Gradle

Вы можете найти официальный gradle.build здесь: <https://github.com/junit-team/junit5-samples/blob/master/junit5-gradle-consumer/build.gradle>

После того, как вы установите Gradle, созданный вами проект может выполнить ваш тест JUnit 5 через терминал:

gradle junitPlatformTest.

Если вы используете Eclipse, лучше всего установить Build ShipTooling. Тогда вы сможете запустить тесты через Run As→Gradle Test. Результат выполнения теста будет показан в консоли.

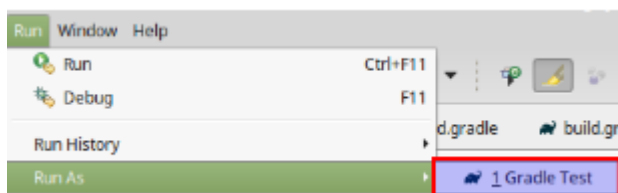


Рисунок 4.10.2. Запуск теста с использованием Build ShipTooling

Использование JUnit 5 с Maven

Данный пример показывает, как импортировать компоненты JUnit 5 в ваш проект.

Зарегистрируйте отдельные компоненты:

```
<build>
  <plugins>
    <plugin>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.1</version>
      <configuration>
        <source>${java.version}</source>
        <target>${java.version}</target>
      </configuration>
    </plugin>
    <plugin>
      <artifactId>maven-surefire-plugin</artifactId>
      <version>2.19.1</version>
      <configuration>
        <includes>
          <include>**/Test*.java</include>
          <include>**/*Test.java</include>
          <include>**/*Tests.java</include>
          <include>**/*TestCase.java</include>
        </includes>
        <properties>
          <!-- <includeTags>fast</includeTags> -->
          <excludeTags>slow</excludeTags>
        </properties>
      </configuration>
      <dependencies>
        <dependency>
          <groupId>org.junit.platform</groupId>
          <artifactId>junit-platform-surefire-
provider</artifactId>
          <version>${junit.platform.version}</version>
        </dependency>
        <dependency>
          <groupId>org.junit.jupiter</groupId>
          <artifactId>junit-jupiter-engine</artifactId>
          <version>${junit.jupiter.version}</version>
        </dependency>
        <dependency>
          <groupId>org.junit.vintage</groupId>
          <artifactId>junit-vintage-engine</artifactId>
          <version>${junit.vintage.version}</version>
        </dependency>
      </dependencies>
    </plugin>
  </plugins>
</build>
```

Рисунок 4.10.3. Регистрация компонентов

И добавьте зависимости

```
<dependencies>
  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-api</artifactId>
    <version>${junit.jupiter.version}</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>${junit.version}</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```


Рисунок 4.10.4. Добавление зависимостей

Более подробный пример настройки maven вы можете найти здесь:

<https://github.com/junit-team/junit5-samples/blob/r5.0.0-M4/junit5-maven-consumer/pom.xml>

Данная настройка подходит для Java проектов, но не подходит для Android-проектов

Определение тестовых методов

JUnit использует аннотации для того, что бы маркировать методы как тестовые и конфигурировать их. Следующая таблица дает обзор самых важных аннотаций в JUnit для 5.x версии. Все эти аннотации могут быть использованы на методах.

Таблица 4.10.1. Аннотации JUnit 5

JUnit 5	Значение
<code>import org.junit.jupiter.api.*</code>	Импорт состояний для использования аннотаций.
<code>@Test</code>	Идентифицирует метод как тестовый
<code>@RepeatedTest(<Number>)</code>	Повторяет тест N-ое количество раз
<code>@TestFactory</code>	Используется для динамического выполнения тестов
<code>@BeforeEach</code>	Выполняется перед каждым тестом. Он используется для подготовки тестовой среды (например, чтение входных данных, инициализация класса).
<code>@AfterEach</code>	Выполняется после каждого теста. Это

JUnit 5	Значение
<code>import org.junit.jupiter.api.*</code>	Импорт состояний для использования аннотаций.
	используется для того, чтобы очистить тестовую среду (удаляет временные данные, восстанавливает значения по умолчанию). Он также может сохранять память путем очистки засоренной памяти.
<code>@BeforeAll</code>	Выполняется один раз, перед выполнением всех тестов. Выполняет действия, которые занимают длительное время, например, подключение к базам данных. Методы, маркированные этой аннотацией, должны быть определены как <code>static</code> для работы с JUnit
<code>@AfterAll</code>	Выполняется один раз, после завершения всех тестов. Используется для очистки активностей, например, завершение соединения с базой данных. Методы, маркированные этой аннотацией, должны быть определены как <code>static</code> для работы с JUnit
<code>@Nested</code>	Позволяет вам встраивать внутренние тестовые классы для принудительного выполнения в определенном порядке.
<code>@Tag("<TagName>")</code>	Используется для фильтрации тестов.

JUnit 5	Значение
<code>import org.junit.jupiter.api.*</code>	Импорт состояний для использования аннотаций.
	Например, запуск тестов только с тегом «fast».
<code>@ExtendWith</code>	Используется для регистрации пользовательских расширений.
<code>@Disabled</code> or <code>@Disabled("Why disabled")</code>	Отключает тест. Обычно используется, когда базовый код был изменен, а тестовый пример еще не адаптирован, или если время выполнения этого теста слишком велико. Рекомендуется использовать описание, в котором необходимо указать, почему тест отключен.
<code>@DisplayName("<Name>")</code>	<code><Name></code> - имя, которое будет отображаться. В отличие от имен методов <code>DisplayName</code> может содержать пробелы.

Отключение тестов

Тесты в JUnit 5 отключаются также, как и в 4 версии.

Наборы тестов

JUnit 5 предоставляет 2 аннотации для объединения тестов в наборы:

-`@SelectPackages` – используется для указания названия модулей, которые будут выбраны при запуске тестов.

-`@SelectedClasses` – используется для указания имен классов, которые будут выбраны при запуске тестов. Они могут находиться в разных модулях.

```
@RunWith(JUnitPlatform.class)
@SelectPackages("com.vogella.junit5.examples")
public class AllTests {}
```

```
@RunWith(JUnitPlatform.class)
@SelectClasses({AssertionTest.class, AssumptionTest.class, ExceptionTest.class})
public class AllTests {}
```

Рисунок 4.10.5. Объединение тестов в набор

Ожидание исключений

За исключением отвечает функция `org.junit.jupiter.api.assertions.expectThrows()`. Вы создаете объект класса `Exception` и кладете туда те параметры, которые должны вызвать исключение.

```
import static org.junit.jupiter.api.Assertions.expectThrows;

@Test
void exceptionTesting() {
    // set up user
    Throwable exception = expectThrows(IllegalArgumentException.class, () ->
    user.setAge("23"));
    assertEquals("Age must be an Integer.", exception.getMessage());
}
```

Рисунок 4.10.6. Ожидание исключений

Благодаря этому вы сможете определить, какая часть теста должна бросить исключение.

Сгруппированные утверждения

На следующем фрагменте продемонстрировано создание сгруппированных исключений.

```
@Test
void groupedAssertions() {
    Address address = new Address();
    // In a grouped assertion all assertions are executed, even after a failure.
    // The error messages get grouped together.
    assertAll("address name",
        () -> assertEquals("John", address.getFirstName()),
        () -> assertEquals("User", address.getLastName())
    );
}

=> org.opentest4j.MultipleFailuresError: address name (2 failures)
expected: <John> but was: <null>
expected: <User> but was: <null>
```

Рисунок 4.10.7. Сгруппированные утверждения

Тайм-аут тестов

Если вы хотите убедиться, что тест не был пройден в течение определенного времени, вы можете использовать метод `AssertTimeout()`.

```
import static org.junit.jupiter.api.Assertions.assertTimeout;
import static java.time.Duration.ofSeconds;
import static java.time.Duration.ofMinutes;

@Test
void timeoutNotExceeded() {
    assertTimeout(ofMinutes(1), () -> service.doBackup());
}

// if you have to check a return value
@Test
void timeoutNotExceededWithResult() {
    String actualResult = assertTimeout(ofSeconds(1), () -> {
        return restService.request(request);
    });
    assertEquals(200, request.getStatus());
}
=> org.opentest4j.AssertionFailedError: execution exceeded timeout of 1000 ms by
212 ms
```

Рисунок 4.10.8. Использование тайм-аута для тестов

Если вы хотите, чтобы ваши тесты были отменены после окончания времени ожидания, можете использовать метод `assertTimeoutPreemptively()`.

```
@Test
void timeoutNotExceededWithResult() {
    String actualResult = assertTimeoutPreemptively(ofSeconds(1), () -> {
        return restService.request(request);
    });
    assertEquals(200, request.getStatus());
}
=> org.opentest4j.AssertionFailedError: execution timed out after 1000 ms
```

Рисунок 4.10.9. Отмена теста после окончания времени ожидания

Повторное выполнение теста на наборе данных

Иногда появляется необходимость в повторном выполнении теста на наборе данных. Выполнение теста на наборе данных в цикле ведет в тому, что ошибка первого утверждения остановит выполнение теста. В JUnit 5 это решается также, как и в JUnit 4 через параметризованные тесты:

```

package testing;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.junit.runners.Parameterized;
import org.junit.runners.Parameterized.Parameters;

import java.util.Arrays;
import java.util.Collection;

import static org.junit.Assert.assertEquals;
import static org.junit.runners.Parameterized.*;

@RunWith(Parameterized.class)
public class ParameterizedTestFields {

    // fields used together with @Parameter must be public
    @Parameter(0)
    public int m1;
    @Parameter(1)
    public int m2;
    @Parameter(2)
    public int result;

    // creates the test data
    @Parameters
    public static Collection<Object[]> data() {
        Object[][] data = new Object[][] { { 1, 2, 2 }, { 5, 3, 15 }, {
121, 4, 484 } };
        return Arrays.asList(data);
    }

    @Test
    public void testMultiplyException() {
        MyClass tester = new MyClass();
        assertEquals("Result", result, tester.multiply(m1, m2));
    }

    // class to be tested
    class MyClass {
        public int multiply(int i, int j) {
            return i * j;
        }
    }
}

```

Рисунок 4.10.10. Повторное выполнение теста

Использование динамических тестов

JUnit предлагает возможность использования динамических тестов. Для динамических тестов используется аннотация `@TestFactory`, такие методы могут возвращать такие объекты, как итератор (`Iterable`), коллекция (`Collection`) или поток (`Stream`) `DynamicTests`. После чего JUnit запускает каждый `DynamicTest` при выполнении теста. Методы `@BeforeEach` и `@AfterEach` не используются для динамических тестов. То есть вы не сможете использовать их, чтобы сбросить тестовый объект, если вы измените его состояние в лямбда-выражении для динамического теста.

В этом примере возвращается поток (`Stream`):

```

import static org.junit.jupiter.api.Assertions.*;
import static org.junit.jupiter.api.DynamicTest.dynamicTest;

import java.util.Arrays;
import java.util.stream.Stream;

import org.junit.jupiter.api.DynamicTest;
import org.junit.jupiter.api.TestFactory;

public class DynamicTestCreationTest {

    @TestFactory
    public Stream<DynamicTest> testMultiplyException() {
        MyClass tester = new MyClass();
        int[][] data = new int[][] { { 1, 2, 2 }, { 5, 3, 15 }, { 121, 4, 484 } };
    };

    return Arrays.stream(data).map(entry -> {
        int m1 = entry[0];
        int m2 = entry[1];
        int expected = entry[2];
        return dynamicTest(m1 + " * " + m2 + " = " + expected, () -> {
            assertEquals(expected, tester.multiply(m1, m2));
        });
    });
}

// class to be tested
class MyClass {
    public int multiply(int i, int j) {
        return i * j;
    }
}
}

```

Рисунок 4.10.11. Использование динамических тестов

Использование параметризованных тестов

JUnit 5 также поддерживает параметризованные тесты. Для их использования необходимо добавить модуль `junit-jupiter-params` в тестовых зависимостях. При использовании `gradle` это выглядит так:

```

dependencies {
    // ..
    testCompile group: 'org.junit.jupiter', name: 'junit-jupiter-params', version:
    '5.0.0-M4'
}

```

Рисунок 4.10.12. Добавление модуля `junit-jupiter-params` в зависимости (Gradle)

Для этого примера использовалась аннотация `@MethodSource`. Мы передаем ей имя функции, которую собираемся вызвать, чтобы получить тестовые данные. Функция должна быть статической и должна возвращать либо `Collection`, `Iterator`, `Stream` или `Array`. При выполнении тестовый метод вызывается один раз для каждого входа в источник данных. Для параметризованных тестов методы `@BeforeEach` и `@AfterEach` будут вызваны.

```

import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.MethodSource;

public class DynamicTestCreationTest {

    public static int[][] data() {
        return new int[][] { { 1, 2, 2 }, { 5, 3, 15 }, { 121, 4, 484 } };
    }

    @ParameterizedTest
    @MethodSource(names = "data")
    void testWithStringParameter(int[] data) {
        MyClass tester = new MyClass();
        int m1 = data[0];
        int m2 = data[1];
        int expected = data[2];
        assertEquals(expected, tester.multiply(m1, m2));
    }

    // class to be tested
    class MyClass {
        public int multiply(int i, int j) {
            return i * j;
        }
    }
}

```

Рисунок 4.10.13. Использование параметризованных тестов

Источник данных

Следующая таблица дает общее представление о всех возможных тестовых источниках данных для параметризованных тестов.

Таблица 4.10.2. Аннотации для параметризованных тестов

<code>@ValueSource(ints = { 1, 2, 3 })</code>	Позволяет определить массив тестовых значений. Допустимые типы: String, int, long или double.
<code>@EnumSource(value = Months.class, names = { "JANUARY", "FEBRUARY" })</code>	Позволяет передавать константы Enum в качестве тестового класса. Благодаря необязательным именам атрибутов вы можете выбрать, какие константы

	<p>следует использовать.</p> <p>В противном случае используются все атрибуты.</p>
<pre>@MethodSource(names = "genTestData")</pre>	<p>Результат указанного в параметрах метода передается в качестве аргумента тесту.</p>
<pre>@CsvSource({ "foo, 1", "'baz, qux', 3" }) void testMethod(String first, int second) {</pre>	<p>Строки будут обрабатываться как Csv. Разделителем является - ','.</p>
<pre>@ArgumentsSource(MyArgumentsProvider.class)</pre>	<p>Указывает класс, который предоставляет тестовые данные.</p>

Преобразование аргументов

JUnit автоматически преобразовывает исходные строки, чтобы они соответствовали ожидаемым аргументам тестируемого метода.

Если необходимо явное преобразование, можно определить конвертор с аннотацией `@ConvertWith`. Чтобы определить свой собственный конвертор, необходимо реализовать интерфейс `ArgumentConverter`. В следующем примере используется абстрактный базовый класс `SimpleArgumentConverter`

```

@ParameterizedTest
@ValueSource(ints = {1, 12, 42})
void
testWithExplicitArgumentConversion(@ConvertWith(ToOctalStringArgumentConverter.class) String argument) {
    System.err.println(argument);
    assertNotNull(argument);
}

static class ToOctalStringArgumentConverter extends SimpleArgumentConverter {
    @Override
    protected Object convert(Object source, Class<?> targetType) {
        assertEquals(Integer.class, source.getClass(), "Can only convert from
Integers.");
        assertEquals(String.class, targetType, "Can only convert to String");
        return Integer.toOctalString((Integer) source);
    }
}

```

Рисунок 4.10.14. Преобразование аргументов

5. Лабораторные работы

Лабораторная работа №1: Основы синтаксиса Java

В данной лабораторной работе вы изучите основы синтаксиса Java с помощью нескольких простых задач программирования. Далее вы узнаете, как использовать компилятор Java и виртуальную машину Java для запуска программы. От вас потребуется решить следующие задачи:

Простые числа

Создайте программу, которая находит и выводит все простые числа меньше 100.

1. Создайте файл с именем Primes.java, в этом файле опишите следующий класс:

```
public class Primes {  
    public static void main(String[] args) {  
    }  
}
```

Воспользовавшись данным классом, соберите и запустите программу. Так как в данной программе нет конкретной реализации, результата выполнения ее вы не увидите.

2. Внутри созданного класса, после метода main(), опишите функцию IsPrime (Int n), которая определяет, является ли аргумент простым числом или нет. Можно предположить, что входное значение n всегда будет больше 2. Полное описание функции будет выглядеть так:

```
public static boolean isPrime(int n)  
{  
}
```

Данный метод вы можете реализовать по вашему усмотрению, однако простой подход заключается в использовании цикла for. Данный цикл должен перебирать числа, начиная с 2 до (но не включая) n, проверяя существует ли

какое-либо значение, делящееся на *n* без остатка. Для этого можно использовать оператора остатка “%”. Например, `17%7` равняется 3, и `16%4` равно 0. Если какая-либо переменная полностью делится на аргумент, сработает оператор `return false`. Если же значение не делится на аргумент без остатка, то это простое число, и оператор покажет `return true`. (Оператор `return` в Java используется для возврата данных из функции, таким способом закрывается метод.)

3. После того, как этот участок будет реализован, приступайте к заполнению основного метода `main()` другим циклом, который перебирает числа в диапазоне от 2 до 100 включительно. Необходимо вывести на печать те значения, которые ваш помощник `IsPrime ()` посчитал простыми.

4. После завершения вашей программы скомпилируйте и протестируйте её. Убедитесь, что результаты правильные. В интернете вы сможете найти списки простых чисел.

Кроме того, как видно из примера, не следует забывать об использовании комментариев: перед классом с его назначением и перед методом с его целью. Когда вы пишете программы, крайне важно писать подобные комментарии.

Палиндромы

Вторая программа, которую вам необходимо будет написать, показывает, является ли строка палиндромом.

1. Для этой программы, создайте класс с именем `Palindrome` в файле под названием `Palindrome.java`. На этот раз вы можете воспользоваться следующим кодом:

```
public class Palindrome {  
    public static void main(String[] args) {  
        for (int i = 0; i < args.length; i++) {  
            String s = args[i];  
        }  
    }  
}
```

Скомпилируйте и запустите эту программу в таком виде, результат работы не будет отображен.

2. Ваша первая задача состоит в том, чтобы создать метод, позволяющий полностью изменить символы в строке. Сигнатура (последовательность) метода должна быть следующей:

```
public static String reverseString(Strings)
```

Вы можете реализовать этот метод путем создания локальной переменной, которая начинается со строки "", а затем добавлять символы из входной строки в выходные данные, в обратном порядке. Используйте метод `length()`, который возвращает длину строки, и метод `charAt(int index)`, который возвращает символ по указанному индексу. Индексы начинаются с 0 и увеличиваются на 1. Например:

```
String s = "pizzeria";  
System.out.println(s.length()); //Выводим 8  
System.out.println(s.charAt(5)); //Выводим r
```

Вы можете использовать оператор конкатенации (соединения) строк `+` или оператор `+=`, на ваше усмотрение.

3. После того, как вы применили метод `reverseString ()`, создайте еще один метод `public static boolean isPalindrome(String s)`. Этот метод должен перевернуть слово `s`, а затем сравнить с первоначальными данными. Используйте метод `Equals (Object)` для проверки значения равенства. Например:

```
String s1 = "hello";  
String s2 = "Hello";  
String s3 = "hello";  
s1.equals(s2); // Истина  
s1.equals(s3); // Ложь
```

Не используйте `==` для проверки равенства строк. Этим занимается другой тест в Java, который будет рассмотрен далее.

4. Скомпилируйте и протестируйте программу! На этот раз входными данными будут аргументы командной строки, например:

```
java Palindrome madam racecar apple kayak song noon
```

Ваша программа должна вывести ответ, является ли каждое слово палиндром.

5. Убедитесь в наличии комментариев, где указаны назначения вашей программы и используемых методов.

Лабораторная работа №2: Основы объектно-ориентированного программирования

Java позволяет использовать объекты. В данной лабораторной работе необходимо использовать классы по одному на файл, чтобы описать, как эти объекты работают. Вот код для простого класса, который представляет двумерную точку:

```
/**
 * двумерный класс точки.
 */

public class Point2d {
    /** координата X */
    private double xCoord;
    /** координата Y */
    private double yCoord;
    /** Конструктор инициализации */
    public Point2d ( double x, double y) {
        xCoord = x;
        yCoord = y;
    }
    /** Конструктор по умолчанию. */
    public Point2d () {
        //Вызовите конструктор с двумя параметрами и определите источник.
        this(0, 0);
    }
    /** Возвращение координаты X */
    public double getX () {
        return xCoord;
    }
    /** Возвращение координаты Y */
```

```
public double getY () {  
    return yCoord;  
}  
/** Установка значения координаты X. */  
public void setX ( double val) {  
    xCoord = val;  
}  
/** Установка значения координаты Y. */  
public void setY ( double val) {  
    yCoord = val;  
}  
}
```

Сохраните данный код в файле с именем Point2d.java, согласно требованиям Java к именам классов и именам файлов.

Экземпляр класса можно также создать, вызвав любой из реализованных конструкторов, например:

```
Point2d myPoint = new Point2d (); //создает точку (0,0)  
Point2d myOtherPoint = new Point2d (5,3); //создает точку (5,3)  
Point2d aThirdPoint = new Point2d ();
```

Примечание: `myPoint != aThirdPoint`, несмотря на то, что их значения равны. Объясняется это тем, что оператор равенства `==` (и его инверсия, оператор неравенства `!=`) сравнивает ссылки на объекты. Другими словами, `==` оператор вернет `true`, если две ссылки указывают на один и тот же объект. В данном случае `myPoint` и `aThirdPoint` ссылаются на разные объекты класса `Point2d`, поэтому операция сравнения `myPoint == aThirdPoint` вернет `false`, несмотря на то, что их значения те же!

Для того, чтобы проверить равны ли сами значения, а не ссылки, необходимо создать метод в классе `Point2d`, который будет сравнивать значения соответствующих полей объектов класса `Point2d`.

Рекомендации при программирования

Стиль программирования является неотъемлемой частью при создании программного обеспечения. При создании приложений большая часть времени уходит на отладку программы. Читаемый код и использование комментариев в нем позволяют сэкономить время при отладке программы.

Ваши задачи:

1. Создайте новый класс `Point3d` для представления точек в трехмерном Евклидовом пространстве. Необходимо реализовать:

- создание нового объекта `Point3d` с тремя значениями с плавающей точкой (`double`);
- создание нового объекта `Point3d` со значениями (0.0, 0.0, 0.0) по умолчанию,
- возможность получения и изменения всех трех значений по отдельности;
- метод для сравнения значений двух объектов `Point3d`.

Нельзя предоставлять непосредственный доступ к внутренним элементам объекта класса `Point3d`.

2. Добавьте новый метод `distanceTo`, который в качестве параметра принимает другой объект `Point3d`, вычисляет расстояние между двумя точками с точность двух знаков после запятой и возвращает полученное значение.

3. Создайте другой класс под названием `Lab1`, который будет содержать статический метод `main`. Помните, что метод `main` должен быть общедоступным (`public`) с возвращаемым значением `void`, а в качестве аргумента должен принимать строку (`String`). Этот класс должен иметь следующую функциональность:

- Ввод координат трех точек, находящихся в трехмерном пространстве. Создание трех объектов типа `Point3d` на основании полученных данных. (Предполагается, что пользователь вводит корректные данные.)

- Создайте второй статический метод `computeArea`, который принимает три объекта типа `Point3d` и вычисляет площадь треугольника, образованного этими точками. (Вы можете использовать формулу Герона.) Верните получившееся значение площади в формате типа `double`.

- На основе полученных данных и с использованием реализованного алгоритма посчитайте площадь и выведите полученное значение пользователю.

Перед вызовом метода `computeArea` проверьте на равенство значений всех трех объектов `Point3d`. Если одна из точек равна другой, то выведите соответствующее сообщение пользователю и не вычисляйте площадь.

4. Скомпилируйте оба исходных файла вместе:

```
javac Point3d.java Lab1.java
```

и затем запустите программу `Lab1`, тестируя ее с несколькими образцами треугольников.

Лабораторная работа №3: Алгоритм A* («A star»)

Если вы когда-нибудь играли в какую-либо игру на компьютере на основе карт, то вы, вероятно, сталкивались с органами компьютерного управления, которые умеют самостоятельно рассчитывать путь из пункта А в пункт Б. На самом деле это обычная распространенная проблема как в играх, так и в других видах программного обеспечения - поиск пути от начального местоположения до пункта назначения с успешным преодолением препятствий.

Один очень широко используемый алгоритм для такого рода проблемы называют A* (произносится "A-star"). Он является наиболее эффективным алгоритмом для поиска пути в компьютерной программе. Концепция алгоритма довольно проста, начиная с исходного местоположения, алгоритм постепенно строит путь от исходной точки до места назначения, используя наикратчайший путь, чтобы сделать следующий шаг. Это гарантирует, что полный путь будет также оптимальным.

Вам не придется реализовывать алгоритм A*; это было уже сделано за вас. На самом деле существует даже пользовательский интерфейс для того, чтобы экспериментировать с этим алгоритмом:

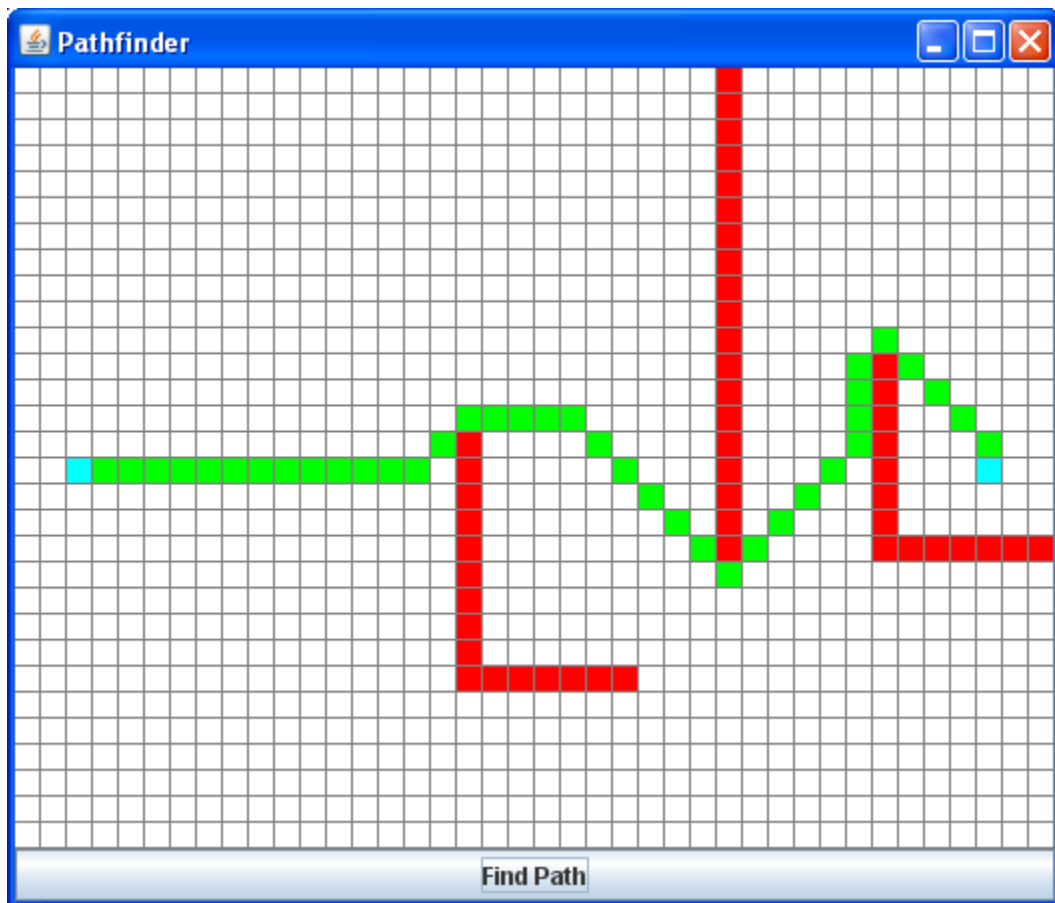


Рисунок 5.1. Пользовательский интерфейс для работы с алгоритмом A*.

Вы можете щелкнуть по различным квадратам, чтобы сделать из них в барьеры (красные) или проходимые клетки (белые). Синие клетки обозначают начало и конец пути. Нажав на кнопку "Find Path", программа вычислит путь, используя алгоритм A*, и затем отобразит его зеленым цветом. В случае если путь не будет найден, то программа просто не отобразит путь.

Алгоритм A* содержит много информации для отслеживания, и классы Java идеально подходят для такого рода задач. Существует два основных вида информации для организации управления алгоритмом A*:

- **Локации** – это наборы координат конкретных клеток на двумерной карте. Алгоритм A* должен иметь возможность ссылаться на определенные места на карте.
- **Вершины** - это отдельные шаги на пути, которые генерирует алгоритм A*. Например, продемонстрированные выше зеленые ячейки - это последовательность вершин на карте.

Каждая путевая точка владеет следующей информацией:

- Расположение ячейки для вершины.
- Ссылка на предыдущую вершину маршрута. Конечный путь - это последовательность вершин от пункта назначения до исходной точки.
- Фактическая стоимость пути от начального местоположения до текущей вершины по определенному пути.
- Эвристическая оценка (приблизительная оценка) остаточной стоимости пути от текущей вершины до конечной цели.

Так как алгоритм A^* строит свой путь, он должен иметь два основных набора вершин:

- Первый набор хранит "открытые вершины" или вершины, которые все еще должны учитываться алгоритмом A^* .
- Второй набор хранит "закрытые вершины" или вершины, которые уже были учтены алгоритмом A^* и их не нужно будет больше рассматривать.

Каждая итерация алгоритма A^* довольно проста: найти вершину с наименьшей стоимостью пути из набора открытых вершин, сделать шаг в каждом направлении от этой вершины для создания новых открытых вершин, а затем переместить вершины из открытого набора в закрытый. Это повторяется до тех пор, пока не будет достигнута конечная вершина! Если во время этого процесса заканчиваются открытые вершины, то пути от начальной вершины до конечной вершины нет.

Данная обработка в первую очередь зависит от расположения вершин, поэтому очень полезно сохранять путевые точки как отображение местоположений до соответствующих вершин. Таким образом, вы будете использовать хранилище `java.util.HashMap` для каждого из этих наборов с объектами `Location` в качестве ключей, и объектами `Waypoint` в качестве значений.

Прежде чем начать

Прежде чем начать, вам необходимо скачать исходные файлы для данной лабораторной работы:

- Map2D.java - представляет карту, по которой перемещается алгоритм A*, включая в себя информацию о проходимости ячеек
- Location.java - этот класс представляет координаты конкретной ячейки на карте
- Waypoint.java - представляет отдельные вершины в сгенерированном пути
- AStarPathfinder.java - этот класс реализует алгоритм поиска пути A* в виде статического метода.
- AStarState.java - этот класс хранит набор открытых и закрытых вершин, и предоставляет основные операции, необходимые для функционирования алгоритма поиска A*.
- AStarApp.java - простое Swing-приложение, которое обеспечивает редактируемый вид 2D-карты, и запускает поиск пути по запросу
- JMapCell.java - это Swing -компонент, который используется для отображения состояния ячеек на карте

Обратите внимание, что приложение будет успешно компилироваться в том виде, какое оно есть, но функция поиска пути не будет работать, пока вы не завершите задание. Единственные классы, которые вы должны изменить это Location и AStarState. Все остальное - это код платформы, которая позволяет редактировать карту и показывать путь, который генерирует алгоритм. (Не рекомендуется редактировать исходный код других файлов)

Локации

Для начала необходимо подготовить класс Location для совместного использования с классами коллекции Java. Поскольку вы будете использовать контейнеры для хеширования для выполнения данного задания, то для этого необходимо:

- Обеспечить реализацию метода equals ().
- Обеспечить реализацию метода hashCode().

Добавьте реализацию каждого из этих методов в класс Location, следуя шаблонам в классе. После этого вы можете использовать класс Location в качестве ключевого типа в контейнерах хеширования, таких как HashSet и HashMap.

Состояния A*

После того, как класс Location готов к использованию, вы можете завершить реализацию класса AStarState. Это класс, который поддерживает наборы открытых и закрытых вершин, поэтому он действительно обеспечивает основную функциональность для реализации алгоритма A*.

Как упоминалось ранее, состояние A* состоит из двух наборов вершин, один из открытых вершин и другой из закрытых. Чтобы упростить алгоритм, вершины будут храниться в хэш-карте, где местоположение вершин является ключом, а сами вершины являются значениями. Таким образом, у вас будет такой тип:

HashMap<Location, Waypoint>

(Очевидный вывод из всего этого заключается в том, что с каждым местоположением на карте может быть связана только одна вершина.)

Добавьте два (нестатических) поля в класс AStarState с таким типом, одно для "открытых вершин" и другой для "закрытых вершин". Кроме того, не забудьте инициализировать каждое из этих полей для ссылки на новую пустую коллекцию.

После создания и инициализации полей, вы должны реализовать следующие методы в классе AStarState:

1) public int numOpenWaypoints()

Этот метод возвращает количество точек в наборе открытых вершин.

2) public Waypoint getMinOpenWaypoint()

Эта функция должна проверить все вершины в наборе открытых вершин, и после этого она должна вернуть ссылку на вершину с наименьшей общей стоимостью. Если в "открытом" наборе нет вершин, функция возвращает NULL.

Не удаляйте вершину из набора после того, как вы вернули ее; просто верните ссылку на точку с наименьшей общей стоимостью.

3) `public boolean addOpenWaypoint(Waypoint newWP)`

Это самый сложный метод в классе состояний A*. Данный метод усложняет то, что он должен добавлять указанную вершину только в том случае, если существующая вершина хуже новой. Вот что должен делать этот метод:

- Если в наборе «открытых вершин» в настоящее время нет вершины для данного местоположения, то необходимо просто добавить новую вершину.
- Если в наборе «открытых вершин» уже есть вершина для этой локации, добавьте новую вершину только в том случае, если стоимость пути до новой вершины меньше стоимости пути до текущей. (Убедитесь, что используете не общую стоимость.) Другими словами, если путь через новую вершину короче, чем путь через текущую вершину, замените текущую вершину на новую

Как вы могли заметить, что в таком случае вам потребуется извлечь существующую вершину из «открытого набора», если таковая имеется. Данный шаг довольно прост - замените предыдущую точку на новую, используя метод `HashMap.put()`, который заменит старое значение на новое. Пусть данный метод вернет значение `true`, если новая вершина была успешно добавлена в набор, и `false` в противном случае.

4) `public boolean isLocationClosed(Location loc)`

Эта функция должна возвращать значение `true`, если указанное местоположение встречается в наборе закрытых вершин, и `false` в противном

случае. Так как закрытые вершины хранятся в хэш-карте с расположениями в качестве ключевых значений, данный метод достаточно просто в реализации.

5) `public void closeWaypoint(Location loc)`

Эта функция перемещает вершину из набора «открытых вершин» в набор «закрытых вершин». Так как вершины обозначены местоположением, метод принимает местоположение вершины.

Процесс должен быть простым:

- Удалите вершину, соответствующую указанному местоположению из набора «открытых вершин».

- Добавьте вершину, которую вы удалили, в набор закрытых вершин.

Ключом должно являться местоположение точки.

Компиляция и тестирование

Как только вы реализуете вышеуказанную функциональность, запустите программу поиска пути, чтобы проверить правильность ее выполнения. Если вы реализовали все правильно, то у вас не должно возникнуть проблем при создании препятствий и последующим поиском путей вокруг них.

Скомпилируйте и запустите программу также, как и всегда:

```
javac *.java
```

```
java AStarApp
```

Лабораторная работа № 4: Рисование фракталов

В следующих нескольких лабораторных работ вы создадите небольшое JAVA-приложение, которое сможет рисовать фракталы. Если вы никогда не играли с фракталами раньше, вы будете удивлены тем, как просто можно создать красивые изображения. Это будет сделано с помощью фреймворка Swing и Java API, который позволяет создавать графические пользовательские интерфейсы.

Начальная версия приложения будет довольно проста, но в следующих лабораторных работах будут добавлены некоторые полезные функции такие как, как сохранение сгенерированных изображений, возможность переключения между различными видами фракталов. И графический интерфейс (GUI), и механизм для поддержки различных фракталов будут зависеть от иерархий классов.

Вот простой пример графического интерфейса в его начальном состоянии:

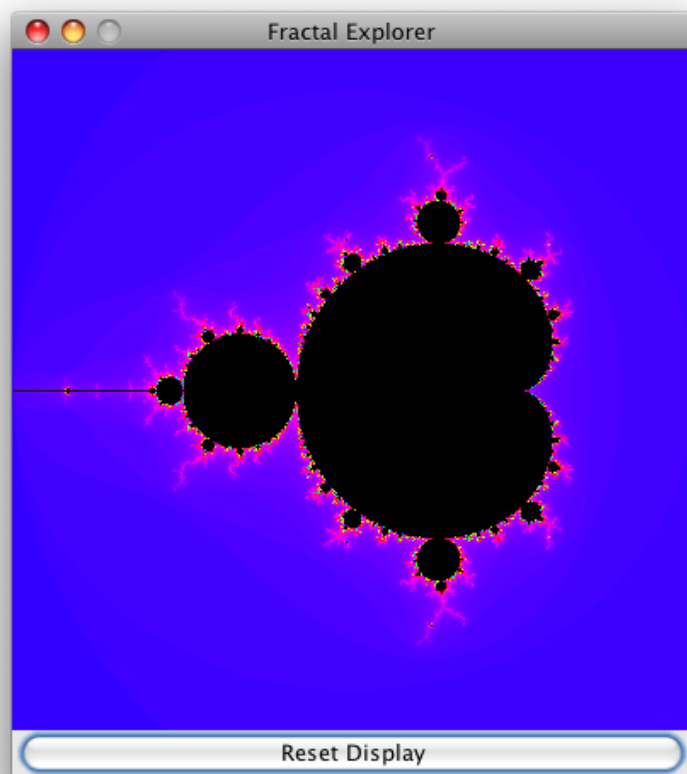


Рисунок 5.2. Пример графического интерфейса

И, вот некоторые интересные области фрактала: слоны и морские коньки!

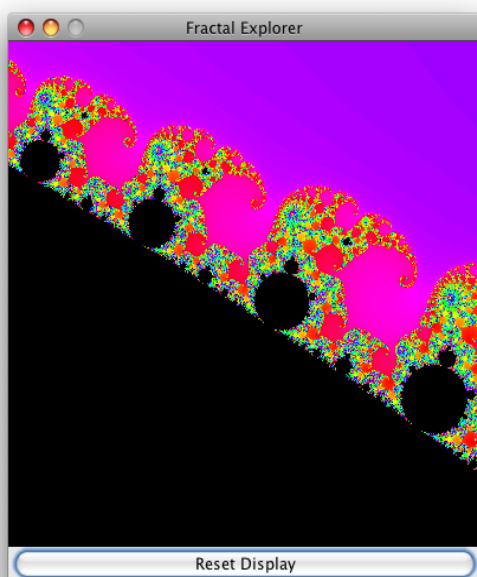


Рисунок 5.3. Фрактал «Слоны»

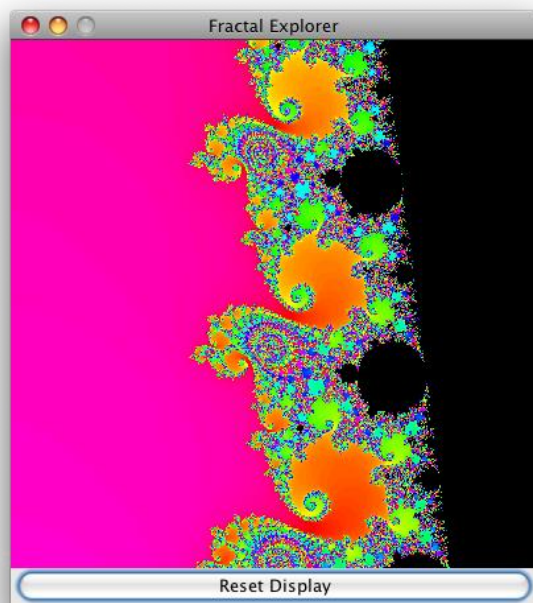


Рисунок 5.4. Фрактал «Морские коньки»

Создание пользовательского интерфейса

Прежде чем рисовать фракталы, необходимо создать графический виджет, который будет их отображать. Swing не предоставляет такой компонент, но его очень просто создать самостоятельно. Обратите внимание на то, что в этой лабораторной работе мы будем использовать широкий спектр классов Java AWT и Swing, детали которых здесь будут не раскрыты. Для более подробной информации вы можете воспользоваться онлайн-документами по API Java. Просто перейдите к пакету данного класса Java, выберите сам класс, а затем прочтите подробную информацию о том, как использовать класс.

- Создайте класс `JImageDisplay`, производный от `javax.swing.JComponent`. Класс должен иметь одно поле с типом доступа `private`, экземпляр `java.awt.image.BufferedImage`. Класс `BufferedImage` управляет изображением, содержимое которого можно записать.

- Конструктор `JImageDisplay` должен принимать целочисленные значения ширины и высоты, и инициализировать объект `BufferedImage` новым изображением с этой шириной и высотой, и типом изображения `TYPE_INT_RGB`. Тип определяет, как цвета каждого пикселя будут

представлены в изображении; значение `TYPE_INT_RGB` обозначает, что красные, зеленые и синие компоненты имеют по 8 битов, представленные в формате `int` в указанном порядке.

Конструктор также должен вызвать метод `setPreferredSize()` родительского класса метод с указанной шириной и высотой. (Вы должны будете передать эти значения в объект `java.awt.Dimension`) Таким образом, когда ваш компонент будет включен в пользовательский интерфейс, он отобразит на экране все изображение.

- Пользовательские компоненты Swing должны предоставлять свой собственный код для отрисовки, переопределяя защищенный метод `JComponent paintComponent (Graphics g)`. Так как наш компонент просто выводит на экран данные изображения, реализация будет очень проста! Во-первых, нужно всегда вызывать метод суперкласса `paintComponent (g)` так, чтобы объекты отображались правильно. После вызова версии суперкласса, вы можете нарисовать изображение в компоненте, используя следующую операцию:

```
g.drawImage (image, 0, 0, image.getWidth(), image.getHeight(), null);
```

(Мы передаем значение `null` для `ImageObserver`, поскольку данная функциональность не требуется.)

- Вы также должны создать два метода с доступом `public` для записи данных в изображение: метод `clearImage ()`, который устанавливает все пиксели изображения в черный цвет (значение RGB 0), и метод `drawPixel (int x, int y, int rgbColor)`, который устанавливает пиксель в определенный цвет. Оба метода будут необходимы для использования в методе `setRGB ()` класса `BufferedImage`.

Не забывайте про комментарии!

Вычисления фрактала Мандельброта

Следующая ваша задача: написать код для вычисления фрактала Мандельброта.

Для создания фракталов используйте следующий исходный файл [FractalGenerator.java](#), от которого будут унаследованы все ваши фрактальные

генераторы. Как вы могли заметить данный файл предоставляет также некоторые полезные операции для перевода из экранных координат в систему координат вычисляемого фрактала.

Виды фракталов, с которыми нужно будет работать, вычисляются в комплексном виде и включают в себя простые математические функции, которые выполняются многократно, пока не выполнится определенное условие. Функция для фрактала Мандельброта имеет вид: $z_n = z_{n-1}^2 + c$, где все значения — это комплексные числа, $z_0 = 0$, и c - определенная точка фрактала, которую мы отображаем на экране. Вычисления повторяются до тех пор, пока $|z| > 2$ (в данной ситуации точка находится не во множестве Мандельброта), или пока число итераций не достигнет максимального значения, например, 2000 (в этом случае делается предположение, что точка находится в наборе).

Процесс построения фрактала Мандельброта прост: необходимо перебрать все пиксели изображения, рассчитать количество итераций для соответствующей координаты, и затем установить пиксель в цвет, основанный на количестве рассчитанных итераций. Все это будет сделано позже, на данном этапе необходимо реализовать приведенные выше вычисления.

- Создайте подкласс `FractalGenerator` с именем `Mandelbrot`. в нем вам необходимо будет обеспечить только два метода: `getInitialRange()` и `numIterations()`.

- `getInitialRange (Rectangle2D.Double)` - метод позволяет генератору фракталов определить наиболее «интересную» область комплексной плоскости для конкретного фрактала. Обратите внимание на то, что методу в качестве аргумента передается прямоугольный объект, и метод должен изменить поля прямоугольника для отображения правильного начального диапазона для фрактала. (Пример можно увидеть в методе `FractalGenerator.recenterAndZoomRange()`.) В классе `Mandelbrot` этот метод должен установить начальный диапазон в $(-2 - 1.5i) - (1 + 1.5i)$. Т.е. значения x и y будут равны -2 и -1.5 соответственно, а ширина и высота будут равны 3 .

- Метод `numIterations(double, double)` реализует итеративную функцию для фрактала Мандельброта. Константу с максимальным количеством итераций можно определить следующим образом:

```
public static final int MAX_ITERATIONS = 2000;
```

Затем вы сможете ссылаться на эту переменную в вашей реализации.

Обратите внимание на то, что у Java нет подходящего типа данных для комплексных чисел, поэтому необходимо будет реализовать итеративную функцию, используя отдельные переменные для действительной и мнимой частей. (Вы можете реализовать отдельный класс для комплексных чисел.) Ваш алгоритм должен обладать быстродействием, например, не стоит сравнивать $|z|$ с 2; сравните $|z|^2$ с 2^2 для того, чтобы избежать сложных и медленных вычислений квадратного корня. Также не стоит использовать метод `Math.pow()` для вычисления небольших степеней, лучше перемножьте значение, иначе ваш быстродействие вашего кода сильно упадет.

В случае, если алгоритм дошел до значения `MAX_ITERATIONS` нужно вернуть -1, чтобы показать, что точка не выходит за границы.

Ваши задачи

Создайте класс `FractalExplorer`, который позволит вам исследовать различные области фрактала, путем его создания, отображения через графический интерфейс `Swing` и обработки событий, вызванных взаимодействием приложения с пользователем.

Как видно из приведенных выше изображений пользовательского интерфейса, `FractalExplorer` очень прост, он состоит из `JFrame`, который в свою очередь содержит объект `JImageDisplay`, который отображает фрактал, и объект `JButton` для сброса изображения, необходимый для отображения целого фрактала. Данный макет можно создать, установив для фрейма `BorderLayout`, затем поместив отображение в центр макета и кнопку сброса в "южной" части макета.

- Класс `FractalExplorer` должен отслеживать несколько важных полей для состояния программы:

- 1) Целое число «размер экрана», которое является шириной и высотой отображения в пикселях. (Отображение фрактала будет квадратным.)

- 2) Ссылка `JImageDisplay`, для обновления отображения в разных методах в процессе вычисления фрактала.

- 3) Объект `FractalGenerator`. Будет использоваться ссылка на базовый класс для отображения других видов фракталов в будущем.

- 4) Объект `Rectangle2D.Double`, указывающий диапозона комплексной плоскости, которая выводится на экран.

Все вышеприведенные поля будут иметь тип доступа `private`.

- У класса должен быть конструктор, который принимает значение размера отображения в качестве аргумента, затем сохраняет это значение в соответствующем поле, а также инициализирует объекты диапозона и фрактального генератора. Данный конструктор не должен устанавливать какие-либо компоненты `Swing`; они будут установлены в следующем методе.

- Создайте метод `createAndShowGUI ()`, который инициализирует графический интерфейс `Swing`: `JFrame`, содержащий объект `JImageDisplay`, и кнопку для сброса отображения. Используйте `java.awt.BorderLayout` для содержимого окна; добавьте объект отображения изображения в позицию `BorderLayout.CENTER` и кнопку в позицию `BorderLayout.SOUTH`.

Вам необходимо дать окну подходящий заголовок и обеспечить операцию закрытия окна по умолчанию (см. метод `JFrame.setDefaultCloseOperation ()`).

После того, как компоненты пользовательского интерфейса инициализированы и размещены, добавьте следующую последовательность операций:

```
frame.pack ();
```

```
frame.setVisible (true);
```



```
frame.setResizable (false);
```

Данные операции правильно разметят содержимое окна, сделают его видимым (окна первоначально не отображаются при их создании для того, чтобы можно было сконфигурировать их прежде, чем выводить на экран), и затем запретят изменение размеров окна.

- Реализуйте вспомогательный метод с типом доступа `private` для вывода на экран фрактала, можете дать ему имя `drawFractal ()`. Этот метод должен циклически проходить через каждый пиксель в отображении (т.е. значения `x` и `y` будут меняться от 0 до размера отображения), и сделайте следующее:

- Вычислите количество итераций для соответствующих координат в области отображения фрактала. Вы можете определить координаты с плавающей точкой для определенного набора координат пикселей, используя вспомогательный метод `FractalGenerator.getCoord ()`; например, чтобы получить координату `x`, соответствующую координате пикселя `X`, сделайте следующее:

```
//x - пиксельная координата; xCoord - координата в пространстве фрактала
```

```
double xCoord = FractalGenerator.getCoord (range.x, range.x + range.width, displaySize, x);
```

- Если число итераций равно -1 (т.е. точка не выходит за границы, установите пиксель в черный цвет (для `rgb` значение 0). Иначе выберите значение цвета, основанное на количестве итераций. Можно также для этого использовать цветовое пространство `HSV`: поскольку значение цвета варьируется от 0 до 1, получается плавная последовательность цветов от красного к желтому, зеленому, синему, фиолетовому и затем обратно к красному! Для этого вы можете использовать следующий фрагмент:

```
float hue = 0.7f + (float) numIters / 200f;
```

```
int rgbColor = Color.HSBtoRGB(hue, 1f, 1f);
```

Если вы придумали другой способ отображения пикселей в зависимости от количества итераций, попробуйте реализовать его!

- Отображение необходимо обновлять в соответствии с цветом для каждого пикселя.

- После того, как вы закончили отрисовывать все пиксели, вам необходимо обновить `JImageDisplay` в соответствии с текущим изображением. Для этого вызовите функцию `repaint()` для компонента. В случае, если вы не воспользуетесь данным методом, изображение на экране не будет обновляться!

- Создайте внутренний класс для обработки событий `java.awt.event.ActionListener` от кнопки сброса. Обработчик должен сбросить диапазон к начальному, определенному генератором, а затем перерисовать фрактал.

После того, как вы создали этот класс, обновите метод `createAndShowGUI()`.

- Создайте другой внутренний класс для обработки событий `java.awt.event.MouseListener` с дисплея. Вам необходимо обработать события от мыши, поэтому вы должны унаследовать этот внутренний класс от класса `MouseAdapterAWT`. При получении события о щелчке мышью, класс должен отобразить пиксельные координаты щелчка в область фрактала, а затем вызвать метод генератора `recenterAndZoomRange()` с координатами, по которым щелкнули, и масштабом 0.5. Таким образом, нажимая на какое-либо место на фрактальном отображении, вы увеличиваете его!

Не забывайте перерисовывать фрактал после того, как вы меняете область фрактала.

Далее обновите метод `createAndShowGUI()`, чтобы зарегистрировать экземпляр этого обработчика в компоненте фрактального отображения.

- В заключении, вам необходимо создать статический метод `main()` для `FractalExplorer` так, чтобы можно было его запустить. В `main` необходимо будет сделать:

- Инициализировать новый экземпляр класса `FractalExplorer` с размером отображения 800.
- Вызовите метод `createAndShowGUI()` класса `FractalExplorer`.
- Вызовите метод `drawFractal()` класса `FractalExplorer` для отображения начального представления.

После выполнения приведенных выше действий, вы сможете детально рассмотреть фрактал Мандельброта. Если вы увеличите масштаб, то вы можете столкнуться с двумя проблемами:

- Во-первых, вы сможете заметить, что в конечном итоге уровень детализации заканчивается; это вызвано тем, что в таком случае необходимо более 2000 итераций для поиска точки во множестве Мандельброта! Можно увеличить максимальное количество итераций, но это приведет к замедлению работы алгоритма.
- Во-вторых, при сильном увеличении масштаба, вы столкнетесь с пиксельным выводом отображения! Это вызвано тем, что вы работаете в пределе того, что могут предоставить значения с плавающей запятой с двойной точностью.

При рисовании фрактала экран ненадолго зависает. Следующая лабораторная работа будет направлена на решение данной проблемы.

Лабораторная работа №5. Выбор и сохранение фракталов

В данной лабораторной работе генератор фракталов будет расширен двумя новыми функциями. Во-первых, вы добавите поддержку нескольких фракталов и реализуете возможность выбирать нужный фрактал из выпадающего списка. Во-вторых, вы добавите поддержку сохранения текущего изображения в файл. Ниже приведен скриншот, где продемонстрировано, как будет выглядеть новая программа

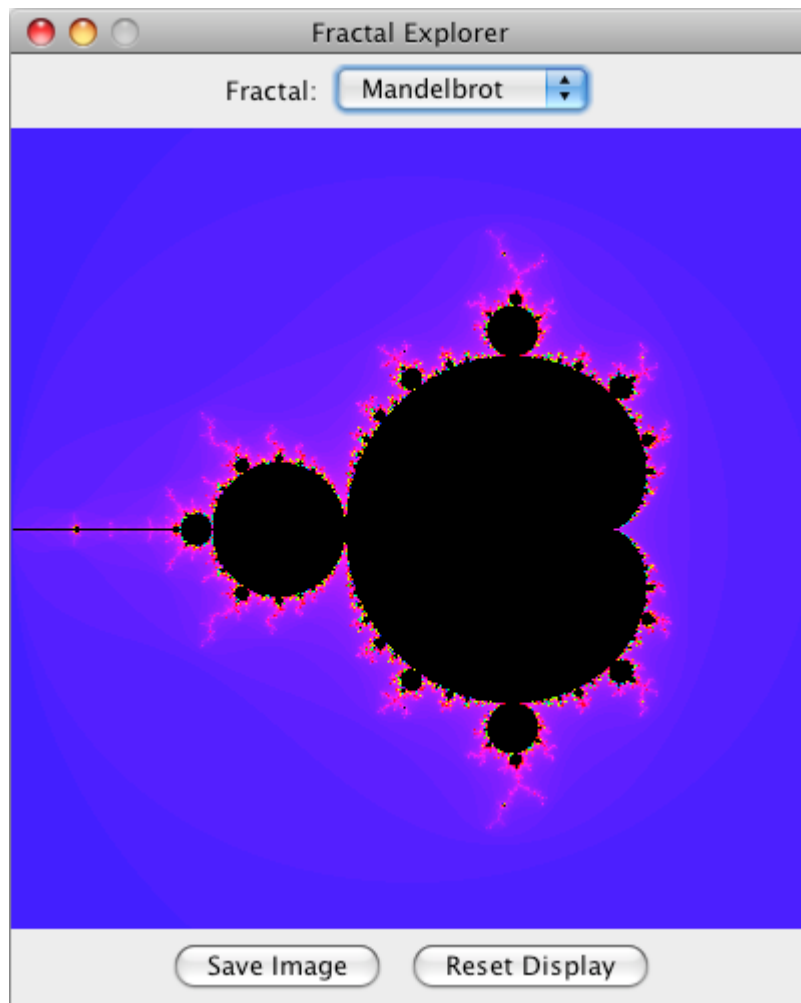


Рисунок 5.5. Графический интерфейс нового приложения

Верхняя панель генератора фракталов включает в себя 2 виджета, позволяющих пользователю выбирать фрактал, а нижняя панель включает в себя кнопку "Save Image", которая сохраняет текущее изображения фрактала.

Так как теперь будет несколько источников событий (action-event sources), вы сможете попрактиковаться в обработке всех источников с использованием одного метода ActionListener в вашем классе.

Поддержка нескольких фракталов

Так как в реализацию была введена абстракция `FractalGenerator`, добавление нескольких фракталов не будет проблемой. В данной лабораторной работе вы добавите поддержку нескольких фракталов, и пользователь сможет выбирать между ними, используя *combo-box*. Программный интерфейс Swing (Swing API) предоставляет *combo-box* через класс `javax.swing.JComboBox`, а также запускает `ActionEvents` при выборе нового элемента. Необходимо сделать:

- Создать 2 новые реализации `FractalGenerator`

Первым будет фрактал `tricorn`, который должен находиться в файле `Tricorn.java`. Для этого нужно создать подкласс `FractalGenerator` и реализация будет почти идентична фракталу Мандельброта, кроме двух изменений. Вы даже можете скопировать исходный код фрактала Мандельберта и просто внести следующие изменения:

- Уравнение имеет вид $z_n = z_{n-1}^2 + c$. Единственное отличие только в том, что используется комплексное сопряжение z_{n-1} на каждой итерации.
- Начальный диапазон для трехцветного фрактала должен быть от $(-2, -2)$ до $(2, 2)$.

Второй фрактал, который необходимо реализовать - это фрактал «Burning Ship», который в реальности не похож на пылающий корабль. Данный фрактал имеет следующие свойства:

- Уравнение имеет вид $z_n = (|\operatorname{Re}(z_{n-1})| + i |\operatorname{Im}(z_{n-1})|)^2 + c$. Другими словами, вы берете абсолютное значение каждого компонента z_{n-1} на каждой итерации.
- Начальный диапазон для данного фрактала должен быть от $(-2, -2.5)$ до $(2, 1.5)$.

- Combo-бокс в Swing может управлять коллекцией объектов, но объекты должны предоставлять метод `toString()`. Убедитесь, что в каждой реализации фракталов `tcnm` метод `toString()`, который возвращает имя, например «Mandelbrot», «Tricorn» и «Burning Ship».

- Настроить JComboBox в вашем пользовательском интерфейсе можно с использованием конструктора без параметров, а затем использовать метод addItem(Object) для того, чтобы добавить реализации вашего генератора фракталов. Как указывалось в предыдущем шаге, выпадающий список будет использовать метод toString () в ваших реализациях для отображения генераторов в выпадающем списке.

Необходимо будет также добавить объект label в разрабатываемый пользовательский интерфейс перед выпадающим списком, в качестве пояснения к выпадающему списку. Это можно сделать, создав новый объект JPanel и добавив в него объекты JLabel и JComboBox, а затем разместить панель на позиции NORTH на вашем макете окна.

И наконец, необходимо добавить поддержку выпадающего списка в реализацию ActionListener. В случае, если событие поступило от выпадающего списка, вы можете извлечь выбранный элемент из виджета и установить его в качестве текущего генератора фракталов. (Используйте метод getSelectedItem()) При этом не забудьте сбросить начальный диапазон и перерисовать фрактал!

Ниже приведены изображения фракталов «Tricorn» и «Burning Ship» для проверки правильности работы алгоритма

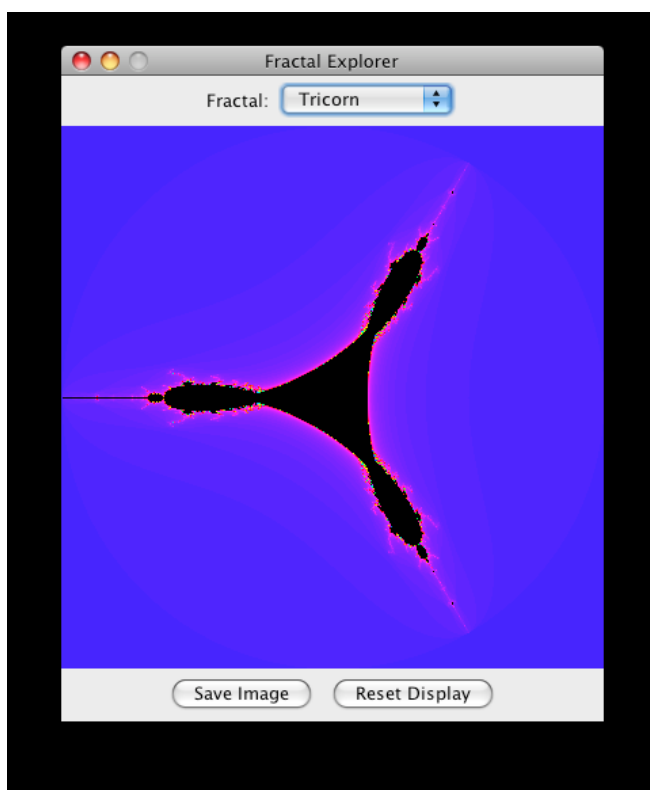


Рисунок 5.6. Фрактал «Tricorn»

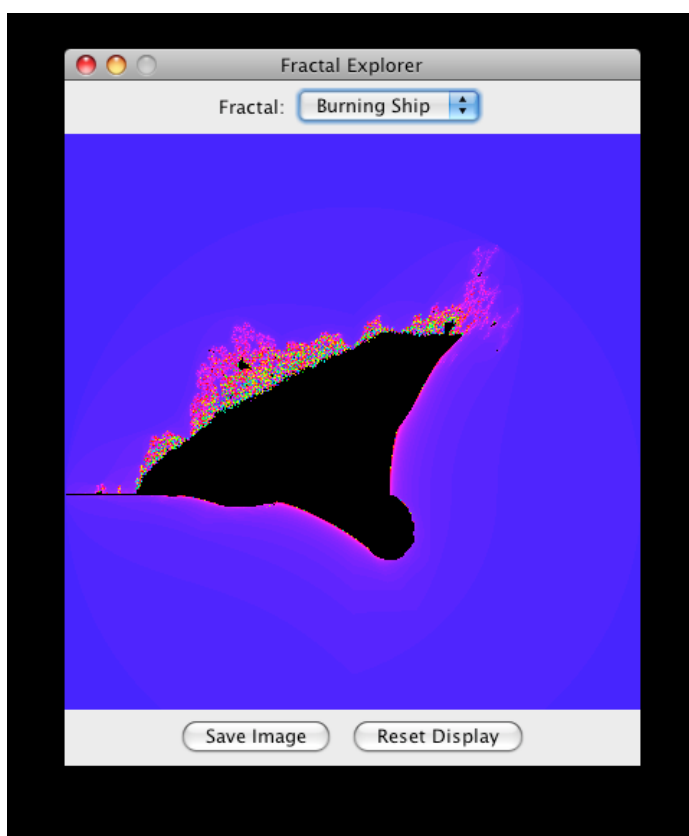


Рисунок 5.7. Фрактал «Burning Ship»

Сохранения изображения Фракталя

Следующая ваша задача - сохранение текущего изображения фрактала на диск. Java API предоставляет несколько инструментов для реализации данной задачи.

- Во-первых, вам нужно добавить кнопку «Save Image» в ваше окно. Для этого вы можете добавить обе кнопки «Save Image» и «Reset» в новую JPanel, а затем разместить эту панель в SOUTH части окна.

События от кнопки «Save Image» также должны обрабатываться реализацией ActionListener. Назначьте кнопкам «Save Image» и «Reset» свои значения команд (например, «save» и «reset») для того, чтобы обработчик событий мог отличить события от этих двух разных кнопок.

- В обработчике кнопки «Save Image» вам необходимо реализовать возможность указания пользователем, в какой файл он будет сохранять изображение. Это можно сделать с помощью класса javax.swing.JFileChooser. Указанный класс предоставляет метод showSaveDialog(), который открывает диалоговое окно «Save file», позволяя тем самым пользователю выбрать директорию для сохранения. Метод принимает графический компонент, который является родительским элементом для диалогового окна с выбором файла, что позволяет центрированию окна с выбором относительно его родителя. В качестве родителя используйте окно приложения.

Как вы могли заметить, данный метод возвращает значение типа int, которое указывает результат операции выбора файла. Если метод возвращает значение JFileChooser.APPROVE_OPTION, тогда можно продолжить операцию сохранения файлов, в противном случае, пользователь отменил операцию, поэтому закончите данную обработку события без сохранения. Если пользователь выбрал директорию для сохранения файла, вы можете ее узнать, используя метод getSelectedFile(), который возвращает объект типа File.

- Также необходимо настроить средство выбора файлов, чтобы сохранять изображения только в формате PNG, на данном этапе вы будете работать только с данным форматом. вы сможете это настроить с помощью

`javax.swing.filechooser.FileNameExtensionFilter`, как это продемонстрировано ниже:

```
JFileChooser chooser = new JFileChooser();
FileFilter filter = new FileNameExtensionFilter("PNG Images", "png");
chooser.setFileFilter(filter);
chooser.setAcceptAllFileFilterUsed(false);
```

Последняя строка гарантирует, что средство выбора не разрешит пользователю использование отличных от png форматов.

- Если пользователь успешно выбрал файл, следующим шагом является сохранения изображения фрактала на диск! Для данного рода задач Java включает в себя необходимую функциональность. Класс `javax.imageio.ImageIO` обеспечивает простые операции загрузки и сохранения изображения. Вы можете использовать метод `write(RenderedImage im, String formatName, File output)`. Параметр `formatName` будет содержать значение «png». Тип «`RenderedImage`» - это просто экземпляр `BufferedImage` из вашего компонента `JImageDisplay`. (Используйте для него тип доступа `public`)

Метод `write()` может вызвать исключение, поэтому вам необходимо заключить этот вызов в блок `try/catch` и обработать возможную ошибку. Блок `catch` должен проинформировать пользователя об ошибке через диалоговое окно. Swing предоставляет класс `javax.swing.JOptionPane` для того, чтобы упростить процесс создания информационных диалоговых окон или окон, где нужно выбрать да/нет. Для этого вы можете использовать статический метод `JOptionPane.showMessageDialog(Component parent, Object message, String title, int messageType)`, где `messageType` у вас будет `JOptionPane.ERROR_MESSAGE`. В сообщении об ошибке вы можете использовать возвращаемое значение метода `getMessage()`, а заголовком окна может быть, например, «Cannot Save Image». Родительским компонентом будет окно для того, чтобы диалоговое окно с сообщением об ошибке выводилось относительно центра окна.

После того, как вы закончите реализацию этих функций, запустите. Теперь вы сможете исследовать различные фракталы, а также вы сможете

сохранять их на диск. Вы также можете проверить приложение на вывод сообщений об ошибках, попробуйте сохранить изображение в файл, который уже существует, но доступен только для чтения. Или вы можете попробовать сохранить файл с именем, которое является каталогом в целевой папке.

Лабораторная работа №6. Многопоточный генератор фракталов

В данной лабораторной работе необходимо будет реализовать возможность рисования фрактала с несколькими фоновыми потоками. Два преимущества данного подхода: первое - пользовательский интерфейс не будет зависеть в процессе рисования нового фрактала, а второе - если у вас компьютер многоядерный, то процесс рисования будет намного быстрее. Несмотря на то, что многопоточное программирование может быть очень сложным, процесс изменения приложения будет прост, благодаря встроенной поддержке Swing фоновых потоков.

На данный момент приложение генератора фракталов выполнялось в одном потоке (Event-Dispatch Thread). Это поток, который обрабатывает все события Swing, такие как нажатие кнопок, перерисовка и т.д. Поэтому разработанный пользовательский интерфейс зависит во время вычисления фрактала; так как вычисление выполняется в потоке обработки событий, возникающие события не могут быть обработаны до завершения вычисления.

В этой лабораторной работе нужно изменить программу так, чтобы она использовала один или несколько фоновых потоков для вычисления фрактала. В частности, поток обработки событий не будет использоваться для вычисления фрактала. Теперь, если вычисление будет выполняться несколькими потоками, необходимо будет разбить его на несколько независимых частей. Например, при рисовании фракталов можно дать каждому потоку по одной строке фрактала для вычисления. Сложность заключается в том, что необходимо соблюдать важное ограничение Swing, а именно, что в потоке обработки событий происходит взаимодействие только с компонентами Swing. Для этого Swing предоставляет инструменты, что упрощает поставленную задачу.

В программировании часто возникают подобные проблемы, что через пользовательский интерфейс запускают длительную операцию, и данная операция должна выполняться в фоновом режиме для сохранения

работоспособности пользовательского интерфейса. Наиболее яркий пример — веб-браузеры; пока страница загружается и отображается, у пользователя должна быть возможность отменить операцию, щелкнуть ссылку или выполнить любое другое действие. Для такого рода проблем Swing предоставляет класс `javax.swing.SwingWorker`, который облегчает процесс организации фонового потока. `SwingWorker` - абстрактный класс, включающий в себя следующие важные методы:

- `doInBackground()` - метод, который фактически выполняет фоновые операции. Swing вызывает этот метод в фоновом потоке, а не в потоке обработки событий.
- `done()` - этот метод вызывается, когда фоновая задача завершена. Он вызывается в потоке обработки событий, поэтому данному методу разрешено взаимодействовать с пользовательским интерфейсом.

Класс `SwingWorker` имеет запутанную спецификацию, на самом деле это `SwingWorker <T, V>`. Тип `T` - это тип значения, возвращаемого функцией `doInBackground()`, когда задача полностью выполнена. Тип `V` используется, когда фоновая задача возвращает промежуточные значения во время выполнения; эти промежуточные значения будут доступны при использовании методов `publish ()` и `process ()`. Оба типа могут не использоваться, в таких случаях необходимо указать `Object` для неиспользуемого типа.

Рисование в фоновом режиме

В данной лабораторной работе в основном необходимо будет работать в классе `FractalExplorer`. Часть кода будет новой, но некоторые части будут представлять из себя модифицированный код, который вы уже написали.

1) Создайте подкласс `SwingWorker` с именем `FractalWorker`, который будет внутренним классом `FractalExplorer`. Это необходимо для того, чтобы у него был доступ к нескольким внутренним членам `FractalExplorer`. Помните, что класс `SwingWorker` является универсальным, поэтому нужно указать параметры - можно просто указать `Object` для двух параметров, потому что в

данной реализации эти параметры не будут использоваться. В результате у вас должна получиться следующая строчка кода:

```
private class FractalWorker extends SwingWorker<Object, Object>
```

2) Класс `FractalWorker` будет отвечать за вычисление значений цвета для одной строки фрактала, поэтому ему потребуются два поля: целочисленная у-координата вычисляемой строки, и массив чисел типа `int` для хранения вычисленных значений RGB для каждого пикселя в этой строке. Конструктор должен будет получать у-координату в качестве параметра и сохранять это. (На данном этапе не надо выделять память под целочисленный массив, так как он не потребуется, пока строка не будет вычислена.)

3) Метод `doInBackground()` вызывается в фоновом потоке и отвечает за выполнение длительной задачи. Поэтому в вашей реализации вам нужно будет взять часть кода из вашей предыдущей функции «draw fractal» и поместить ее в этот метод. Вместо того, чтобы рисовать изображение в окне, цикл должен будет сохранить каждое значение RGB в соответствующем элементе целочисленного массива. Вы не сможете изменять отображение из этого потока, потому что вы нарушите ограничения потоков `Swing`.

4) Вместо этого выделите память для массив целых чисел в начале реализации этого метода (массив должен быть достаточно большим для хранения целой строки значений цвета), а затем сохраните цвет каждого пикселя в этом массиве. Единственные различия между настоящим и предыдущим кодом в том, что вам нужно будет вычислить фрактал для указанной строки, и что вы на данном этапе не обновляете отображение.

Метод `doInBackground()` должен возвращать объект типа `Object`, так как это указано в объявлении `SwingWorker <T, V>`. Просто верните `null`.

5) Метод `done()` вызывается, когда фоновая задача завершена, и этот метод вызывается из потока обработки событий `Swing`. Это означает, что вы можете модифицировать компоненты `Swing` на ваш вкус. Поэтому в этом

методе вы можете перебирать массив строк данных, рисуя пиксели, которые были вычислены в `doInBackground ()`.

После того, как строка будет вычислена, вам нужно будет сообщить `Swing`, перерисовать часть изображения, которая была изменена. Поскольку вы изменили только одну строку, перерисовывать изображение целиком будет затратно, поэтому вы можете использовать метод `JComponent.repaint()`, который позволит вам указать область для перерисовки. У данного метода есть неиспользуемый параметр типа `long`, вы можете просто указать 0 для этого аргумента. В качестве остальных параметров укажите вычисленную строку, значения начала фрагмента для перерисовки (0, y) и конечные значения фрагмента (`displaySize, 1`).

После того, как вы завершили класс для фоновой задачи, следующим шагом нужно будет привязать его к процессу рисования фракталов. Так как часть кода из функции «draw fractal» уже задействована в разрабатываемом классе, на данном этапе можно изменить функцию «draw fractal», а именно, для каждой строки в отображении создать отдельный рабочий объект, а затем вызвать для него метод `execute ()`. Это действие запустит фоновый поток и запустит задачу в фоновом режиме. Помните, что класс `FractalWorker` отвечает за генерацию данных строки и за рисование этой строки, поэтому функция «draw fractal» должна быть простой.

После завершения данной функции, вы сможете заметить отображения стало более быстрым, а пользовательский интерфейс стал более отзывчивым.

Также вы сможете заметить одну проблему в вашем пользовательском интерфейсе - если вы нажмете на экран или на кнопку во время перерисовки, программа обработает это событие, хотя оно должно быть проигнорировано до завершения операции.

Игнорирование событий во время перерисовки

Самый простой способ решить проблему игнорирования событий во время перерисовки - отслеживать количество оставшихся строк, которые

должны быть завершены, и игнорировать или отключать взаимодействие с пользователем до тех пор, пока не будут нарисованы все строки. Для этого нужно добавить поле «rows remaining» в класс Fractal Explorer и использовать его, чтобы узнать, когда будет завершена перерисовка. Чтение и запись этого значения будет происходить в потоке обработки событий, чтобы не было параллельного доступа к этому элементу. Если взаимодействие с ресурсом будет происходить только из одного потока, то не возникнет ошибок параллелизма. Для этого:

- Создайте функцию `void enableUI(boolean val)`, которая будет включать или отключать кнопки с выпадающим списком в пользовательском интерфейсе на основе указанного параметра. Для включения или отключения этих компонентов можно использовать метод `Swing.setEnabled(boolean)`. Убедитесь, что ваш метод обновляет состояние кнопки сохранения, кнопки сброса и выпадающего списка.

- Функция «draw fractal» должна сделать еще две вещи. Первая - она должна вызвать метод `enableUI (false)`, чтобы отключить все элементы пользовательского интерфейса во время рисования. Вторая - она должна установить значение «rows remaining» равным общему количеству строк, которые нужно нарисовать. Эти действия должны быть сделаны перед выполнением каких-либо рабочих задач, иначе это может привести к некорректной работе алгоритма.

- В методе `done()`, уменьшите значение «rows remaining» на 1, как последний шаг данной операции. Затем, если после уменьшения значение «rows remaining» равно 0, вызовите метод `enableUI (true)`.

- Наконец, измените реализацию `mouse-listener` для того, чтобы она сразу возвращалась в предыдущее состояние, если значение «rows remaining» не равно нулю. Другими словами, приложение будет реагировать на щелчки мышью, только в том случае, если больше нет строк, которые должны быть нарисованы. (Обратите внимание, что также не нужно вносить аналогичные

изменения в обработчике событий, потому что все эти компоненты будут отключены с помощью метода `enableUI ()`.)

После выполнения данных шагов, должна получиться программа для рисования фракталов с несколькими потоками и, которая запретит действия пользователя, пока процесс рендеринга происходит в фоновом режиме.

Лабораторная работа №7. Веб-сканер

В этой лабораторной работе вам необходимо будет реализовать элементарный веб-сканер. Сканер будет автоматически загружать веб-страницы из Интернета, искать новые ссылки на этих страницах и повторять их. Он будет просто искать новые URL-адреса (местоположения веб-страниц) на каждой странице, собирать их и выводит в конце работы программы. Более сложные веб-сканеры используются для индексации содержимого Интернета или для очистки адресов электронной почты от спама. Если вы когда-нибудь использовали поисковую систему, то вы в ответ на запрос получали данные, генерируемые поисковым роботом.

Терминология

- **URL:** унифицированный указатель ресурса. Это адрес веб-страницы. Он имеет следующую структуру:

- 1) метод доступа к ресурсу;
- 2) доменное имя
- 3) путь к файлу
- 4) данные о файле

В данной лабораторной работе будет рассмотрен метод доступа «http://».

- **HTTP:** Hyper Text Transfer Protocol (Протокол передачи гипертекста). Это стандартный текстовый протокол, используемый для передачи данных веб-страницы через Интернет. Последней спецификацией HTTP является версия 1.1, которую будет использована в данной лабораторной работе.

- **Сокет:** Сокет(разъем) - это ресурс, предоставляемый операционной системой, который позволяет вам обмениваться данными с другими компьютерами по сети. Вы можете использовать сокет для установки соединения с веб-сервером, но вы должны использовать TCP-сокет и использовать протокол HTTP для того, чтобы сервер мог ответить.

- **Порт:** несколько разных программ на одном сервере могут слушать соединения через разные порты. Каждый порт обозначается номером в диапазоне 1..65535. Номера от 1 до 1024 зарезервированы для операционной системы. У большинства серверов есть порт по умолчанию. Для HTTP-соединений обычно используется порт 80.

Программа для записи

Ниже указаны требования к программе, которую вы должны написать.

1. Программа должна принимать в командной строке два параметра:
 - 1) Строку, которая представляет собой URL-адрес, с которого можно начать просмотр страницы.
 - 2) Положительное целое число, которое является максимальной глубиной поиска (см. ниже)

Если указаны некорректные аргументы, программа должна немедленно остановиться и выдать сообщение об используемых аргументах, например:

```
usage: java Crawler <URL><depth>
```

2. Программа должна хранить URL-адрес в виде строки вместе с его глубиной (которая для начала будет равна 0). Вам будет необходимо создать класс для представления пар [URL, depth].

3. Программа должна подключиться к указанному сайту в URL-адресе на порт 80 с использованием сокета (см. ниже) и запросить указанную веб-страницу.

4. Программа должна проанализировать возвращаемый текст, построчно для любых подстрок, имеющих формат:

```
<a href="[любой_URL-адрес_начинающийся_с_http://]">
```

Найденные URL-адреса должны быть сохранены в паре с новым значением глубины в LinkedList (URL, depth) (подробнее о LinkedLists см. ниже). Новое значение глубины должно быть больше, чем значение глубины URL-адреса, соответствующего анализируемой странице.

5. Далее программа должна закрыть соединение сокета с хостом.

6. Программа должна повторять шаги с 3 по 6 для каждого нового URL-адреса, если глубина, соответствующая URL-адресу, меньше максимальной. Обратите внимание, что при извлечении и поиске определенного URL-адреса глубина поиска увеличивается на 1. Если глубина URL-адреса достигает максимальной глубины (или больше), не извлекайте и не ищите эту веб-страницу.

7. Наконец, программа должна вывести все посещенные URL-страницы вместе с их глубиной поиска.

Предположения

- Сложно разобрать, а тем более подключиться ко всем правильно и неправильно сформированным гиперссылкам в Интернете. Предположим, что каждая ссылка сформирована правильно, с полным именем хоста, путем и ресурсу. Вы можете создать небольшой сайт для тестирования, или вы можете выбрать любой другой сайт в интернете с уровнем доступа `http://`. (Вы можете попробовать `http://slashdot.org/` или `http://www.nytimes.com.`)

В случае, если вы найдете URL-адрес с отличным от `http://` методом доступа, вы должны его игнорировать.

- Предположим, если ваш `BufferedReader` возвращает значение `null`, то сервер завершил отправку веб-страницы. На самом деле это не всегда верно для медленных веб-серверов, но для текущих целей это приемлемо.

Полезные классы и методы

Указанные ниже классы и методы должны помочь вам начать работу. Обратите внимание, что большинство этих методов выбрасывают различные виды исключений, с которыми вам придется работать. Более подробную информацию вы можете найти на сайте Java API.

Socket

Для использования сокетов (`Socket`) вам необходимо включить эту строку в вашу программу:

```
import java.net. *;
```

Конструктор

`Socket (String host, int port)` создает новый сокет из полученной строки с именем хоста и из целого числа с номером порта, и устанавливает соединение.

Методы

- `void setSoTimeout(int timeout)` устанавливает время ожидания сокета (`Socket`) в миллисекундах. Данный метод необходимо вызвать после создания сокета, чтобы он знал, сколько нужно ждать передачи данных с другой стороны. В противном случае у него будет бесконечное время ожидания, что приведет к неэффективности разрабатываемого сканера.
- `InputStream getInputStream()` возвращает `InputStream`, связанный с `Socket`. Этот метод позволяет сокету получать данные с другой стороны соединения.
- `OutputStream getOutputStream()` возвращает `OutputStream`, связанный с `Socket`. Этот метод позволяет сокету отправлять данные на другую сторону соединения.
- `void close()` закрывает сокет (`Socket`).

Потоки (Streams)

Для использования потоков в разрабатываемой программе необходимо включить в код следующую строку:

```
import java.io.*;
```

Для эффективного использования сокетов, вам необходимо преобразовать `InputStream` и `OutputStream`, связанные с `Socket`, во что-то более удобное в использовании. Объекты `InputStream` и `OutputStream` являются примитивными, так как они могут читать только байты или массивы байтов. Поскольку в данной работе необходимо читать и писать символы, вы должны использовать объекты, которые преобразуют байты в символы и печатают целые строки. Java API делает это несколькими разными способами для ввода и вывода.

Потоки ввода (Input Streams)

Для потоков ввода вы можете использовать классы `InputStreamReader` следующим образом:

```
InputStreamReader in = new InputStreamReader  
(my_socket.getInputStream());
```

Теперь `in` имеет тип `InputStreamReader`, который может читать символы из сокета (`Socket`). Но данный подход не очень удобен, потому что по-прежнему приходится работать с отдельными символами или массивами символов. Для чтения целых строк вы можете использовать класс `BufferedReader`. Вы можете создать `BufferedReader` с объектами типа `InputStreamReader`, а затем вызвать метод `readLine` предусмотренный в `BufferedReader`. Данный метод будет читать целую строку с другого конца соединения.

Потоки вывода (Output Streams)

Потоки вывода организованы проще. Вы можете создать экземпляр `PrintWriter` непосредственно из объекта `OutputStream`, а затем вызвать его метод `println` для отправки строки текста на другой конец соединения. Для этого используйте следующий конструктор:

```
PrintWriter (OutputStream out, boolean autoFlush)
```

Параметр `autoFlush` установите в значение `true`. Это приведет к очищению буфера вывода после каждого вызова метода `println`.

Строковые методы

Приведенные ниже методы `String` будут полезны в работе. Смотрите документацию по API.

- `boolean equals (Object anObject)`
- `String substring (int beginIndex)`
- `String substring (int beginIndex, int endIndex)`
- `boolean startsWith (String prefix)`

ПРИМЕЧАНИЕ. Не используйте оператор `==` для сравнения строк! Оператор будет возвращать `true`, только если две строки являются одним и тем

же объектом. Если вы хотите сравнить содержимое двух строк, используйте метод equals.

Списки (Lists)

Списки похожи на массивы объектов, за исключением того, что они могут легко менять размерность при необходимости, и в них не используются скобки для поиска отдельных элементов. Чтобы использовать списки, вы должны включить следующую строку в код программы:

```
import java.util. *;
```

Для хранения пар (URL, depth) используйте LinkedList, который является реализацией List. Создайте его следующим образом:

```
LinkedList <URLDepthPair> myList = новый LinkedList <URLDepthPair>  
();
```

Посмотрите в API используемые методы в списках и различные реализации списков. (В частности, вы можете заметить, что разные реализации List предоставляют разные функции. Именно поэтому рекомендуется LinkedList, некоторые из его функций больше подходят для этой работы.)

Специальный синтаксис для создания LinkedList продемонстрированный выше использует поддержку Java 1.5 generics. Этот синтаксис означает, что вам не нужно использовать приведение типов для объектов, которые вы храните или извлекаете из списка.

Исключения

В случае, если вы найдете URL-адрес, который не начинается с «http: //», вы должны выдать исключение MalformedURLException, которое является частью Java API.

Советы по проектированию

Ниже приведены рекомендации по разработке вашего поискового сканера.

Пары URL-Depth

Как упоминалось выше, вы должны создать специальный класс `URLDepthPair`, каждый экземпляр которого включает в себя поле типа `String`, представляющее URL-адрес, и поле типа `int`, представляющее глубину поиска. У вас также должен быть метод `toString`, который выводит содержимое пары. Этот метод упрощает вывод результатов веб-сканирования.

Отдельные URL-адреса необходимо разбить на части. Этот анализ URL-адреса и манипуляция им должны быть частью созданного вами класса `URLDepthPair`. Правила хорошего объектно-ориентированного программирования гласят, что если какой-либо класс хранит в себе определенный тип данных, тогда любые виды манипуляций с этими данными также должны быть реализованы в данном классе. Итак, если вы пишете какие-либо функции для того, чтобы разбить URL-адрес, или для проверки на то, является ли URL-адрес допустимым, поместите их в этот класс.

Сканеры (Crawlers)

Как уже упоминалось выше, вы должны спроектировать класс `Crawler`, который будет реализовывать основные функциональные возможности приложения. Этот класс должен иметь метод `getSites`, который будет возвращать список всех пар URL-глубины, которые были посещены. Вы можете вызвать его в основном методе после завершения сканирования; получить список, затем выполнить итерацию по нему и распечатать все URL-адреса.

Самый простой способ отслеживания посещенных сайтов состоит в том, чтобы хранить два списка, один для всех сайтов, рассмотренных до текущего момента, и один, который включает только необработанные сайты. Вам следует перебирать все сайты, которые не были обработаны, удаляя каждый сайт из списка перед загрузкой его содержимого, и каждый раз, когда вы находите новый URL-адрес, необходимо поместить его в необработанный список. Когда необработанный список пуст, сканер завершает работу.

Несмотря на то, что возникает предположение, что открытие сокета по URL-адресу - операция, связанная с URL-адресом, и, следовательно, должна быть реализована в классе пар URL-Depth, это было бы слишком специализированным для целей класса. Класс пар URL-Depth является только местом для хранения URL-адресов и значений глубины, а также включает в себя несколько дополнительных утилит. Сканер (Crawler) - это класс, который перемещается по веб-страницам и ищет URL-адреса, поэтому класс сканера должен включать в себя код, который фактически открывает и закрывает сокеты.

Вам нужно создать новый экземпляр Socket для каждого URL-адреса, с которого вы загружаете текст. Обязательно закройте сокет, когда вы закончите сканирование этой веб-страницы, чтобы операционная система не исчерпала сетевые ресурсы. (Компьютер может держать открытыми очень много сокетов одновременно). Кроме того, не используйте рекурсию для поиска глубоко вложенных веб-страниц; реализуйте эту функцию через цикл. Это также сделает ваш веб-сканер более эффективным с точки зрения использования ресурсов.

Константы

Разрабатываемая программа будет содержать строки типа «http: //» и «a href = \», и у вас может возникнуть желание использовать эти строки везде, где они вам могут понадобиться. Кроме того, вам понадобятся длины для разных строковых операций, поэтому у вас также может возникнуть желание жестко закодировать длины этих строк в коде. Не стоит это делать. Если вы сделаете опечатку или позже поменяете способ поиска, вам придется соответственно менять достаточное количество строк кода.

Вместо этого создайте строковые константы в классах. Например:

```
public static final string URL_PREFIX = "http: //";
```


Теперь, если вам будет нужна эта строка, используйте константу `URL_PREFIX`. Если вам нужна ее длина, используйте строковый метод для приведенной выше константы: `URL_PREFIX.length()`.

Продумайте расположение констант. Вам необходимо, чтобы каждая константа появлялась один раз во всем проекте, и вы должны разместить константу там, где это наиболее целесообразно. Например, поскольку префикс URL-адреса нужен для того, чтобы определить, действителен ли URL-адрес, вы должны поместить эту константу в класс `URL-Depth`. Если у вас есть еще одна константа для ссылок HTML, поместите ее в класс сканера. Если у сканера появится необходимость в префиксе URL, он может ссылаться на константу пары `URL-depth`, вместо дублирования этой константы.

Дополнительное задание

- Добавьте код для добавления сайтов в необработанный список, если они не были просмотрены ранее.
- Расширьте возможности по поиску гиперссылок сканера, используя поиск по регулярным выражениям в собранных данных. Вам также понадобится больше логики, чтобы решить, к какой машине подключаться в следующей итерации. Сканер должен иметь возможность перемещаться по ссылкам на различных популярных сайтах.
- Создайте пул из пяти (или более) сканеров. Каждый должен работать в своем потоке, каждый может получить URL-адрес для просмотра и каждый из них должен вернуть список ссылок по завершению работы. Посылайте новые URL-адреса в этот пул, как только они станут доступными.
- Расширьте многопоточный сканер так, чтобы можно было выполнить поиск на глубину до 1 000 000. Сохраняйте результаты каждого обхода в базе данных через JDBC и запишите, сколько раз на каждую конкретную уникальную страницу ссылались другие. Включите интеллектуальный алгоритм для «поиска смысла» на каждой странице путем взвешивания слов и фраз на основе повторяемости, близости к началу абзацев и

разделов, размера шрифта или стиля заголовка и, по крайней мере, метаключевых слов.

Лабораторная работа №8: Модифицированный веб-сканер

Сканер в прошлой лабораторной работе был не особенно эффективным. В данной лабораторной работе вы расширите сканер для использования поточной обработки Java так, чтобы несколько веб-страниц можно было сканировать параллельно. Это приведет к значительному повышению производительности, так как время, которое каждый поток сканера тратит на ожидание завершения сетевых операций, может прерываться другими операциями обработки в других потоках.

Подробное введение в многопоточное программирование на Java вы можете прочитать в данном учебном руководстве. Самое главное прочитать этот подраздел.

Расширение веб-сканера

В данной лабораторной работе вы расширите и измените разработанную ранее программу:

1. Реализуйте класс с именем `URLPool`, который будет хранить список всех URL-адресов для поиска, а также относительный "уровень" каждого из этих URL-адресов (также известный как "глубина поиска"). Первый URL-адрес, который нужно будет найти, будет иметь глубину поиска равную 0, URL-адреса, найденные на этой странице, будут иметь глубину поиска равную 1 и т.д. Необходимо сохранить URL-адреса и их глубину поиска вместе, как экземпляры класса с именем `URLDepthPair`, как это было сделано в прошлой лабораторной работе. `LinkedList` рекомендуется использовать для хранения элементов, так как это поможет эффективно выполнить необходимые операции.

У пользователя класса `URLPool` должен быть способ получения пары URL-глубина из пула и удаления этой пары из списка одновременно. Должен также быть способ добавления пары URL-глубина к пулу. Обе эти операции должны быть поточно-ориентированы, так как несколько потоков будут взаимодействовать с `URLPool` одновременно.

У пула URL не должно быть максимального размера. Для этого нужен список необработанных URL-адресов, список уже отсканированных URL-адресов и еще одно поле, о котором будет написано ниже.

2. Чтобы выполнить веб-сканирование в нескольких потоках, необходимо создать класс `CrawlerTask`, который реализует интерфейс `Runnable`. Каждый экземпляр `CrawlerTask` должна иметь ссылку на один экземпляр класса `URLPool`, который был описан выше. (Обратите внимание на то, что все экземпляры класса `CrawlerTask` используют единственный пул!) Принцип работы веб-сканера заключается в следующем:

- 1). Получение пары URL-Depth из пула, ожидая в случае, если пара не будет сразу доступна.

- 2). Получение веб-страницы по URL-адресу.

- 3). Поиск на странице других URL-адресов. Для каждого найденного URL-адреса, необходимо добавить новую пару URL-Depth к пулу URL-адресов. Новая пара должна иметь глубину на единицу больше, чем глубина текущего URL-адреса, по которому происходит сканирование.

- 4). Переход к шагу 1.

Данный цикл должен продолжаться до тех пор, пока в пуле не останется пар URL-Depth.

3. Так как веб-сканер будет порождать некоторое количество потоков, измените программу так, чтобы она принимала третий параметр через командную строку, который будет определять количество порождаемых потоков веб-сканера. Функция `main` должна выполнять следующие задачи:

- 1). Обработать аргументы командной строки. Сообщить пользователю о любых ошибках ввода.

- 2). Создать экземпляр пула URL-адресов и поместить указанный пользователем URL-адрес в пул с глубиной 0.

3). Создать указанное пользователем количество задач (и потоков для их выполнения) для веб-сканера. Каждой задаче поискового робота нужно дать ссылку на созданный пул URL-адресов.

4). Ожидать завершения веб-сканирования.

5) Вывести получившийся список URL-адресов, которые были найдены.

4. Синхронизируйте объект пула URL-адресов во всех критических точках, так как теперь код должен быть ориентирован на многопоточность.

5. Веб-сканер не должен постоянно опрашивать пул URL-адресов в случае, если он пуст. Вместо этого пусть они ожидают в случае, когда нет доступных URL-адресов. Реализуйте это, используя метод `wait()` внутри «`get URL`» в случае, если ни один URL-адрес в настоящее время недоступен. Соответственно, метод "add URL" пула URL-адресов должен использовать функцию `notify()` в случае, когда новый URL-адрес добавлен к пулу.

Обратите внимание на то, что потоки веб-сканера не сами будут выполнять какие-либо из этих операций синхронизации/ожидания/уведомления. По той же причине, что и пул URL-адресов скрывает детали того, как URL-адреса хранятся и извлекаются: инкапсуляция! Точно так же, как и в вашей реализации пользователи пула URL-адресов не должны знать о деталях реализации, также они не должны знать о деталях организации потоков.

Советы по проектированию

Вот некоторые советы для успешного выполнения лабораторной работы №8:

- Вы можете использовать часть кода из лабораторной работы №7 с небольшим изменением. Класс `URLDepthPair` изменять не нужно. Основные отличия в том, что код загрузки URL-адреса и сканирование страницы находится теперь в классе, который реализует `Runnable` и код будет получать и добавлять URL-адреса в экземпляре `URLPool`.

- Вы должны синхронизировать доступ к внутренним полям URLPool, поскольку к ним будут обращаться сразу несколько потоков. Самый простой подход заключается в использовании методов синхронизации. Не нужно синхронизировать конструктор URLPool! Подумайте о том, какие методы должны быть синхронизированы.

- Напишите методы URLPool для использования методов wait() и notify() так, чтобы потоки сканера могли ожидать появления новых URL-адресов.

- Пусть URLPool определяет, какие URL-адреса попадают в список необработанных URL-адресов, исходя из глубины каждого URL-адреса, добавляемого в пул. Если глубина URL-адреса меньше максимальной, добавьте пару в очередь ожидания. Иначе добавьте URL-адрес в список обработанных, не сканируя страницу.

- Самая сложная часть данной лабораторной работы заключается в поиске момента для выхода из программы, когда больше нет URL-адресов для сканирования. В таком случае все потоки будут в режиме ожидания нового URL-адреса в URLPool. Для этого рекомендуется, чтобы URLPool отслеживал, сколько потоков ожидает новый URL-адрес. Поэтому необходимо добавить поле типа int, которое будет увеличиваться непосредственно перед вызовом wait() и уменьшаться сразу после выхода из режима ожидания. Создав счетчик, нужно реализовать метод, который возвращает количество ожидающих потоков. Отслеживать количество потоков вы можете в функции main() и в случае, если общее количество потоков равно количеству потоков, которое вернул соответствующий метод, необходимо вызвать System.exit() для завершения работы. Проверку можно выполнять по таймеру (с интервалом 1 сек), что приведет к более эффективной работе программы.

Дополнительное задание

- Обновите пару URL-Depth для использования класса `java.net.URL` и произведите соответствующие изменения в веб-сканере для того, чтобы он соответствовал и относительным URL-адресам, и абсолютным.
- Выход из программы с использованием вызова `System.exit()` - грубая операция. Найдите способ более корректного выхода из программы.
- Реализуйте список URL-адресов, которые были просмотрены, и избегайте возврата к ним. Используйте один из классов коллекций `java`. Какой-то набор, который поддерживает постоянное время поиска и вставку, будет наиболее подходящим.
- Добавьте другой дополнительный параметр командной строки для того, чтобы определить, сколько времени поток веб-сканера должен ждать сервера для возврата требуемой веб-страницы.