

Г. А. ЧИСТЯКОВ, М. Л. ДОЛЖЕНКОВА

ВВЕДЕНИЕ В JAVA-ТЕХНОЛОГИИ

Учебно-методическое пособие

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ
БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт математики и информационных систем
Факультет автоматики и вычислительной техники
Кафедра электронных вычислительных машин

Г. А. ЧИСТЯКОВ, М. Л. ДОЛЖЕНКОВА

ВВЕДЕНИЕ В JAVA-ТЕХНОЛОГИИ

Учебно-методическое пособие

Киров

2018

УДК 004.451(07)
Ч-689

Допущено к изданию методическим советом факультета автоматизации и вычислительной техники ВятГУ в качестве учебно-методического пособия для студентов направлений 09.03.01 «Информатика и вычислительная техника» и 09.03.03 «Прикладная информатика» всех профилей подготовки, всех форм обучения

Рецензент
канд. техн. наук,
доцент кафедры САУ ВятГУ,
В. И. Семеновых

Чистяков, Г. А.

Ч-689 Введение в Java-технологии : учебно-методическое пособие / Г. А. Чистяков, М. Л. Долженкова. – Киров : ВятГУ, 2018. – 52 с.

Учебно-методическое пособие предназначено для выполнения лабораторных работ по дисциплине «Разработка программных систем».

УДК 004.451(07)

© ВятГУ, 2018

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	4
1. ЛАБОРАТОРНАЯ РАБОТА №1. ЗНАКОМСТВО С ЯЗЫКОМ ПРОГРАММИРОВАНИЯ JAVA И СРЕДОЙ РАЗРАБОТКИ ECLIPSE.....	5
1.1 Принципы функционирования Java-программ	5
1.2 Задание на лабораторную работу	8
1.3 Примерные варианты заданий	9
1.4 Пример консольного приложения для демонстрации работы библиотеки классов	11
1.5 Указания к выполнению работы	13
2. ЛАБОРАТОРНАЯ РАБОТА №2. ИСПОЛЬЗОВАНИЕ СРЕДСТВ АВТОМАТИЧЕСКОЙ ГЕНЕРАЦИИ ДОКУМЕНТАЦИИ И РЕФАКТОРИНГА ПРОГРАММНОГО КОДА	14
2.1 Рефакторинг программного кода	14
2.2 Автоматическая генерация технической документации с использованием Javadoc	15
2.3 Задание на лабораторную работу	20
2.4 Указания к выполнению работы	21
3. ЛАБОРАТОРНАЯ РАБОТА №3. РАЗРАБОТКА ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА С ИСПОЛЬЗОВАНИЕМ ТЕХНОЛОГИИ SWING.....	22
3.1 Концепция MVC.....	22
3.2 Библиотека организации пользовательского интерфейса Swing	28
3.3 Задание на лабораторную работу	32
3.4 Указания к выполнению работы	33
4. ЛАБОРАТОРНАЯ РАБОТА №4. ЗНАКОМСТВО С ИНСТРУМЕНТОМ АВТОМАТИЗАЦИИ РЕШЕНИЯ ЗАДАЧ APACHE ANT	34
4.1 Автоматизация процесса сборки программных продуктов.....	34
4.2 Задание на лабораторную работу	39
4.3 Указания к выполнению работы	39
5. ЛАБОРАТОРНАЯ РАБОТА №5. ОРГАНИЗАЦИЯ ПРОЦЕССА РАЗРАБОТКИ ПО С ПРИМЕНЕНИЕМ СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ	40
5.1 Принципы функционирования систем контроля версий	40
5.2 Задание на лабораторную работу	46
5.3 Указания к выполнению работы	46
СПИСОК ЛИТЕРАТУРЫ.....	48

ВВЕДЕНИЕ

Представленное пособие содержит методические указания к выполнению первой части лабораторного практикума по дисциплине «Разработка программных систем». Данная дисциплина осваивается на третьем курсе подготовки бакалавров по направлениям 09.03.01 «Информатика и вычислительная техника», профили «Программное и аппаратное обеспечение вычислительной техники», «Программное обеспечение вычислительной техники и автоматизированных систем» и 09.03.03 «Прикладная информатика», профиль «Прикладная информатика в экономике».

Лабораторный практикум включает в себя пять работ, ориентированных на знакомство с встроенными в Java средствами и библиотеками: инструменты рефакторинга программного кода, Javadoc, Apache Ant, библиотека создания графического интерфейса пользователя Swing, – а также с применяемой на сегодняшний день повсеместно системой контроля версий Subversion.

ЛАБОРАТОРНАЯ РАБОТА №1.

ЗНАКОМСТВО С ЯЗЫКОМ ПРОГРАММИРОВАНИЯ JAVA И СРЕДОЙ РАЗРАБОТКИ ECLIPSE

Целью работы является знакомство с языком программирования Java и интегрированной средой разработки Eclipse.

1.1 Принципы функционирования Java-программ

В настоящее время язык Java является одним наиболее широко используемых стандартов в промышленном программировании. Причиной этого являются заложенные в него компанией Sun Microsystems при выпуске принципы.

Простота. Синтаксис языка Java изначально представлял собой упрощенный вариант синтаксиса языка C++. В языке отсутствуют указатели, адресная арифметика, деструкторы объектов. При этом «сборка мусора» осуществляется автоматически.

Объектная ориентированность. Java – строгий объектно-ориентированный язык. Программный код заключен внутри поименованных классов.

Строгая типизация. При выполнении программы не производится преобразований типов, не разрешено смешивание типов.

Надежность. Значительное внимание в Java уделяется раннему обнаружению возможных ошибок. В Java все объекты программы расположены в динамической памяти и доступны по объектным ссылкам, которые, в свою очередь, хранятся в стеке. Таким образом, исключается непосредственный доступ к памяти.

Безопасность. Язык Java изначально предназначался для создания программ, работающих в сети. Он имеет встроенную настраиваемую систему обеспечения безопасности при выполнении кода. Эта система предотвращает намеренное переполнение стека выполняемой программы, повреждение участков памяти, находящихся за пределами пространства, выделенного процессу, несанкционированное чтение файлов и их модификацию.

Кроссплатформенность. Компилятор генерирует объектный файл, формат которого не зависит от архитектуры компьютера. Скомпилированная программа может выполняться на любых процессорах, для ее работы необходима лишь исполняющая система Java.

Переносимость. Ни один из аспектов спецификации Java не зависит от реализации конкретной исполняющей системы.

Динамичность. Проверка и разрешение доступа к объектам осуществляется в ходе работы программы.

Java-программа состоит из одного или нескольких определений классов, размещенных в одном или нескольких файлах с расширением «.java». Для компиляции программ используется java-компилятор, в результате работы которого для каждого класса из исходного файла создается файл «.class», содержащий байт-код. В общем случае, один из классов программы должен быть открытым (public) и содержать метод main, с которого начинается выполнение программы. Процесс создания приложения представлен на рис. 1.

Байт-кодом называется машинно-независимый код низкого уровня, генерируемый транслятором и исполняемый интерпретатором. Большинство инструкций байт-кода эквивалентны одной или нескольким командам ассемблера.

Байт-код выполняется внутри JVM (Java Virtual Machine). JVM отрабатывает набор скомпилированных инструкций и манипулирует элементами памяти. В конечном счете, JVM интерпретирует инструкции байт-кода и преобразует их в машинный код (Native code) для выполнения. JVM реализована под различные аппаратные платформы и операционные системы, такие как Windows, Linux, Mac OS, IBM Mainframes и так далее. Несмотря на то, что детали реализации JVM различны, все они способны выполнять один и тот же байт-код.

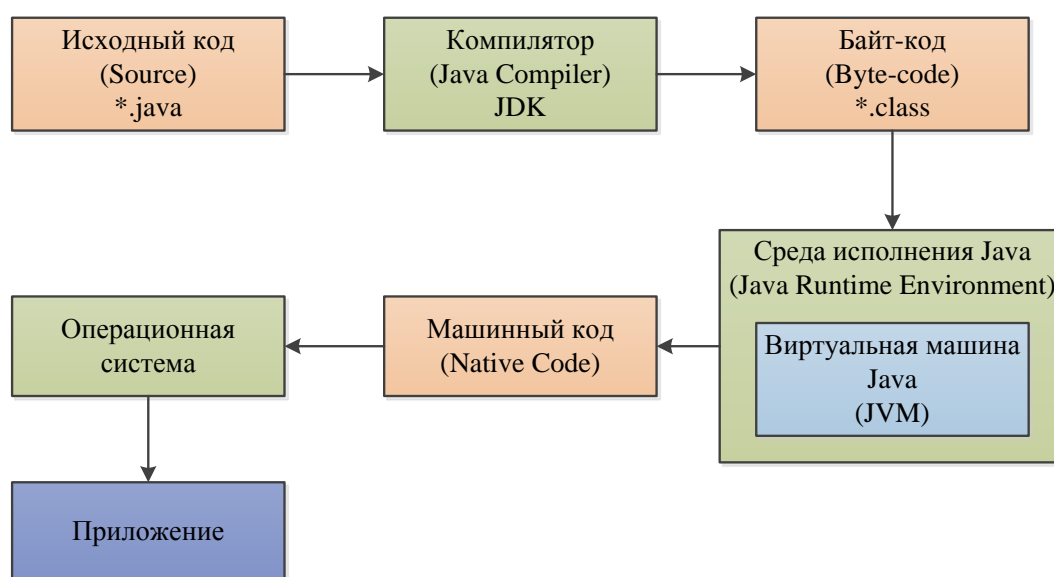


Рис. 1. Процесс создания и выполнения приложения на языке программирования Java

JVM является частью JRE (Java Runtime Environment) – среды для исполнения Java-приложений, которая, кроме JVM, включает в себя основные классы и набор библиотек, необходимых для обеспечения работоспособности любого приложения.

Таким образом, компонентами JRE являются:

- JVM, которая включает в себя Java-клиенты HotSpot и серверные виртуальные машины.
- Набор базовых библиотек.
- Компоненты поддержки технологий развертывания Java Web Start и Java Plug-in.
- Инструменты организации взаимодействия с пользователем: AWT, Java 2D, Swing, Sound, Image IO и так далее.
- Библиотеки интеграции: JDBC, IDL, RMI и так далее.
- Прочие дополнительные библиотеки: Beans, Networking, Security, Serialization, IO и так далее.

Однако, для разработки программ JRE недостаточно, поэтому существует ее расширенная версия – Java Development Kit (JDK). JDK – это средство разработки программного обеспечения, которое может использоваться и для выполнения приложений Java. Оно включает в себя JRE, компилятор Java, интерпретатор, отладчик, а так же средства просмотра документов.

Среди доступных в JDK компонентов можно выделить:

- Java. Загрузчик Java-программ, интерпретирующий байт-код Java и преобразующий их в машинный код.
 - Javac. Компилятор, преобразующий исходный код Java в байт-код.
 - Appletviewer. Инструмент запуска и отладки Java-апплетов.
 - Jar. Архиватор, объединяющий связанные библиотеки классов в один файл.
 - Javafxpackager. Инструмент упаковки и подписи приложений JavaFX.
 - Javadoc. Автоматический генератор технической документации.
- Существует четыре основных семейства Java-технологий.
- Java Card. Технология, которая позволяет небольшим Java-приложениям (апплетам) надежно работать на смарт-картах и других подобных устройствах с малым объемом памяти.

- Java ME. Включает несколько различных наборов библиотек для устройств с ограниченными ресурсами, небольшим размером дисплея и батареи. Часто используется для разработки приложений для мобильных устройств, КПК, ресиверов цифрового телевидения и принтеров.

- Java SE. Основное семейство технологий, предназначенное для использования на настольных ПК, серверах и другом подобном оборудовании.

- Java EE. Расширенный вариант Java SE, дополненный API для разработки клиент-серверных бизнес-приложений.

Почти сразу же после появления Java было создано большое количество интегрированных сред разработки программ для этого языка: Eclipse (Eclipse Foundation), NetBeans (Sun Microsystems), JBuilder (Inprise), Visual Age (IBM), VisualCafe (Symantec) и другие. Большинство из инструментальных сред написаны полностью на Java и имеют развитые средства визуального программирования.

Основные инструментальные средства Eclipse включают в себя современный редактор исходного кода, а также средства взаимодействия с Apache Ant, Javadoc, JUnit, CVS, WebDav, SWT, UI. Кроме этого в Eclipse доступны множество бесплатных и коммерческих дополнений (плагинов), таких, как инструментальные средства создания схем UML, разработка баз данных и прочих. Кроме того, Eclipse может использоваться и как среда для разработки на языках C/C++, Cobol, Perl, PHP, Ruby.

1.2 Задание на лабораторную работу

В соответствии с выбранным вариантом разработать набор классов на языке программирования Java. Для выполнения лабораторной работы необходимо решить следующие задачи.

- Установить на рабочую станцию виртуальную Java-машину (JDK).
- Установить на рабочую станцию среду разработки Eclipse.
- Настроить окружение.
- Создать новый проект.
- Реализовать группу классов.
- Реализовать консольное приложение, использующее в своей работе разработанную библиотеку классов.
- Продемонстрировать работу приложения на подготовленном сценарии.

1.3 Примерные варианты заданий

Вариант №1. Разработать класс для генерации простых чисел. Класс должен иметь два публичных метода:

- `int getRandomPrime()` – возвращает случайное простое число из диапазона $[2, 10^9]$;
- `int[] getRandomArray(int length)` – возвращает упорядоченный по возрастанию массив простых чисел размерности `length`;
- и два внутренних метода:
- `int getNext(int prime)` – возвращает следующее после `prime` простое число из диапазона $[2, 10^9]$;
- `boolean isPrime(int arg0)` – определяет является ли заданный аргумент простым числом.

Вариант №2. Разработать библиотеку статических методов для факторизации больших чисел. Должны быть реализованы следующие публичные методы:

- факторизация методом Полларда $p-1$;
- факторизация методом Полларда p ;
- факторизация методом Бента;
- факторизация методом Полларда-Монте-Карло;
- факторизация методом Ферма;
- разложение числа на простые множители.

Для внутреннего представления чисел необходимо использовать экземпляры класса `BigInteger`.

Вариант №3. Реализовать класс `SmallInteger` для работы с целыми числами, не превосходящими по абсолютному значению 10^4 . Класс должен содержать следующие публичные методы:

- сложения, вычитания, умножения;
- целочисленного деления и определения остатка от деления.

Сигнатура методов должна иметь вид «`public SmallInteger operation(SmallInteger arg)`». Кроме того, класс должен иметь не менее двух конструкторов. В случае выхода промежуточных или окончательного результата за границы диапазона представления должна инициироваться исключительная ситуация, расширяющая базовую иерархию.

Вариант №4. Разработать класс `BigFraction` для работы с дробной длинной арифметикой. Класс должен содержать следующие публичные методы:

- сложения, вычитания, умножения, деления;
- сокращения дроби.

Сигнатура методов должна иметь вид «`public BigFraction operation(BigFraction arg)`». Представление дроби должно инкапсулироваться посредством двух экземпляров классов `BigInteger`. Класс должен иметь не менее двух конструкторов. Для корректного представления экземпляров класса при их выводе на экран требуется переопределить метод `toString()`.

Вариант №5. Реализовать класс `Matrix` для работы с матрицами. Класс должен содержать публичные методы сложения, вычитания и умножения матриц.

Сигнатура методов должна иметь вид «`public Matrix operation(Matrix arg)`». В случае невозможности выполнения операции должна иницироваться исключительная ситуация, расширяющая базовую иерархию.

Вариант №6. Разработать библиотеку статических методов для работы со строками. Должны быть реализованы следующие публичные методы:

- `public static isPrefix(String str, String sub)` – определяет, является ли строка `sub` префиксом строки `str`;
- `public static isSuffix(String str, String sub)` – определяет, является ли строка `sub` суффиксом строки `str`;
- `public static isSubstring(String str, String sub)` – определяет, является ли строка `sub` подстрокой строки `str`;
- `public static isSubsequence(String str, String sub)` – определяет, является ли строка `sub` подпоследовательностью строки `str`.

Методы должны быть реализованы алгоритмически эффективно.

Вариант №7. Реализовать класс для представления графа. Требуется обеспечить поддержку трех форм хранения данных:

- матрица смежности;
- список ребер;
- список смежности вершин.

Требуется предусмотреть возможность работы как с взвешенными, так и с невзвешенными графами. Класс должен содержать набор методов для получения данных, связанных с конкретной вершиной, а также для

изменения структуры графа. В случае некорректного использования формы представления должна инициироваться исключительная ситуация, расширяющая базовую иерархию.

Вариант №8. Разработать класс, реализующий функционал по выполнению интервальных операций посредством sqrt-декомпозиции. Требуется реализовать методы для:

- изменения значения в заданной точке;
- изменения значений на интервале;
- определения суммы значений на интервале.

Класс должен корректно работать со всеми примитивными числовыми типами данных. В случае возникновения нештатной ситуации должно инициироваться исключение.

Вариант №9. Разработать класс, реализующий функционал хранения множества элементов как хэш-таблицы. Требуется реализовать методы для:

- добавления элемента в множество;
- удаления элемента из множества;
- проверки наличия элемента в множества.

Класс должен корректно работать со всеми примитивными числовыми типами данных. В случае возникновения нештатной ситуации должно инициироваться исключение.

Вариант №10. Разработать библиотеку статических методов для сортировки пар из данных ключ/значение. Должны быть реализованы следующие публичные методы:

- сортировка Шелла;
- интроспективная сортировка;
- плавная сортировка.

Представление пар данных должно быть организовано с помощью базового абстрактного класса, расширяемого в дальнейшем пользователями библиотеки.

1.4 Пример консольного приложения для демонстрации работы библиотеки классов

В качестве примера рассмотрим возможный вариант консольного приложения для варианта №5.

Требуется реализовать матричный калькулятор, обладающий следующим функционалом.

1. Ввод матрицы с клавиатуры посредством конструкции «*Name* = [*0*, *0*, ...]», где *Name* – имя переменной, в которой будут содержаться данные по окончании выполнения операции.

2. Ввод матрицы из файла посредством конструкции «*Name* = file(*filepath*)», где *Name* – имя переменной, в которой будут содержаться данные по окончании выполнения операции, а *filepath* – путь к файлу, из которого выполняется загрузка.

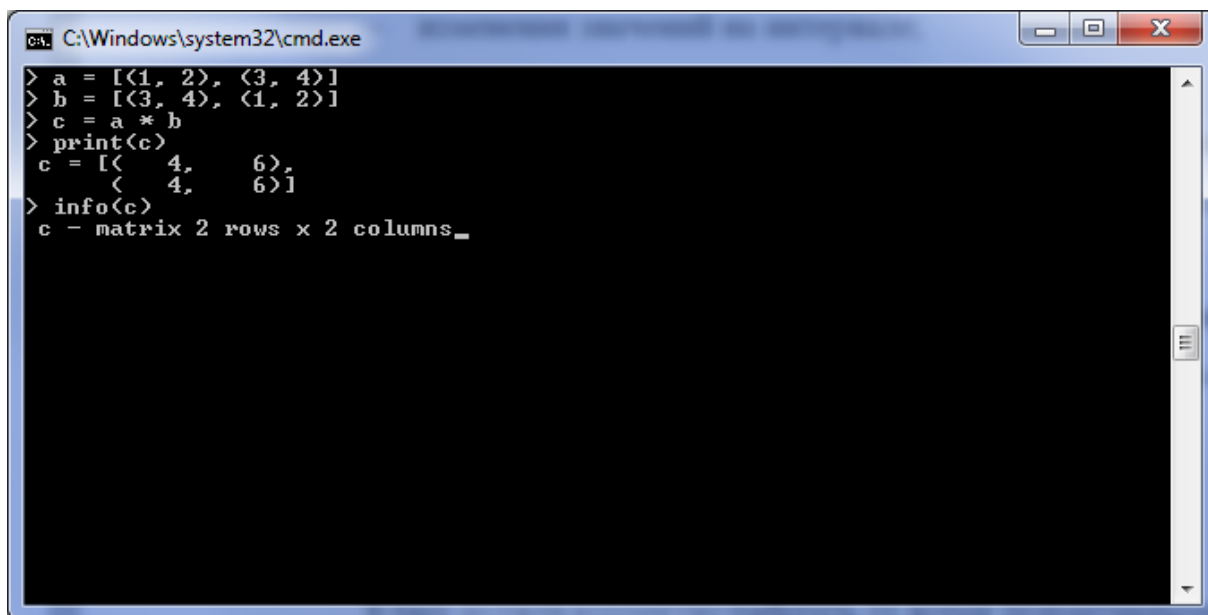
3. Выполнение операций сложения, вычитания и умножения матриц посредством конструкции «*M1* = *M2 operation M3*», где *M1* – имя переменной, в которой будут содержаться данные по окончании выполнения операции, *M2* и *M3* – имена переменных-аргументов, а *operation* – символ выполняемой операции (-, +, *).

4. Вывод информации о размерности матрицы с помощью конструкции «Info(*Name*)», где *Name* – имя переменной, содержащей матрицу.

5. Форматированный вывод матрицы на экран с помощью конструкции «Print(*Name*)», где *Name* – имя переменной, содержащей матрицу. Выполнение конструкций из пунктов 1–3 без указания целевой переменной должно приводить к форматированному выводу результата.

В случае возникновения нештатной ситуации пользователь должен видеть на экране сообщение с точной причиной ошибки.

Пример работы калькулятора представлен на рис. 2.



```
C:\Windows\system32\cmd.exe
> a = [<1, 2>, <3, 4>]
> b = [<3, 4>, <1, 2>]
> c = a * b
> print(c)
c = [< 4, 6>,
    < 4, 6>]
> info(c)
c - matrix 2 rows x 2 columns_
```

Рис. 2. Пример работы матричного калькулятора

1.5 Указания к выполнению работы

Отчет по лабораторной работе должен содержать:

- титульный лист;
- задание на лабораторную работу;
- описание назначения и функциональных возможностей консольного приложения, использующего разработанную группу классов;
- исходный код реализованных классов и интерфейсов с комментариями;
- описание вспомогательных классов, реализованных в ходе разработки консольного приложения;
- экранные формы, подтверждающие работоспособность приложения;
- выводы по проделанной работе.

ЛАБОРАТОРНАЯ РАБОТА №2.

ИСПОЛЬЗОВАНИЕ СРЕДСТВ АВТОМАТИЧЕСКОЙ ГЕНЕРАЦИИ ДОКУМЕНТАЦИИ И РЕФАКТОРИНГА ПРОГРАММНОГО КОДА

Целью работы является знакомство с встроенными в среду разработки Eclipse средствами для автоматической генерации документации и рефакторинга кода.

2.1 Рефакторинг программного кода

Рефакторинг представляет собой процесс изменения программы, при котором не меняется внешнее поведение кода, но существенно улучшается его внутренняя структура, что положительным образом сказывается на трудоемкости процессов поддержки и сопровождения.

Рефакторинг улучшает композицию программного обеспечения, облегчает понимание программного кода, а также косвенно помогает в поиске ошибок и позволяет снижать временные затраты непосредственно на кодирование. В некоторых методологиях разработки ПО рефакторинг является неотъемлемой частью процесса цикла разработки.

При этом не следует путать рефакторинг с процессами оптимизации и реинжиниринга. Основная цель процесса оптимизации – получение более производительного кода, который чаще всего является менее удобным для дальнейшей поддержки. Задача реинжиниринга – добавление в проект дополнительной функциональности.

На сегодняшний день рефакторинг не производится путем непосредственных манипуляций с программным кодом. Вместо этого используются специализированные средства, позволяющие минимизировать шансы появления новых ошибок.

Процедуры рефакторинга, характерные для объектно-ориентированного программирования можно разбить на три группы:

- изменение именования и физической организации кода;
- изменение логической организации кода на уровне классов;
- изменение кода внутри классов.

Примерами методов первой группы являются:

- переименование элемента (пакета, класса, переменной);
- перемещение класса из одного пакета в другой.

Примерами изменения логической организации кода являются:

- преобразование анонимного класса во вложенный;
- создание класса из вложенного;
- перемещение методов и полей в иерархии классов;
- выделение группы методов в интерфейс.

К методам изменения кода внутри классов можно отнести:

- выделение группы операторов в метод;
- удаление метода и замена его вызова непосредственно кодом;
- преобразование выражения в константу и обратно;
- преобразование выражения во временную переменную и обратно;
- генерация методов доступа к полю;
- изменение сигнатуры метода.

Основанная проблема рефакторинга – отсутствие уверенности в правильном функционировании кода после проделанных изменений. Данная проблема, как правило, устраняется с помощью проведения регулярного автоматического unit-тестирования.

В настоящее время широкое распространение получили полуавтоматизированные средства рефакторинга, интегрированные практически во все среды разработки ПО. С их помощью осуществляется большая часть необходимых изменений. Разработчик лишь проверяет и корректирует сделанные изменения.

2.2 Автоматическая генерация технической документации с использованием Javadoc

В современном промышленном программировании под документированием принято понимать процесс создания документации, которая может быть разделена на четыре основных типа.

- проектная;
- техническая;
- пользовательская;
- маркетинговая.

Проектная документация представляет собой описание структурных и архитектурных особенностей ПО, являющегося предметом разработки. Данный вид документации, как правило, не предназначен для широкого

использования и представляет интерес исключительно для заказчика и исполнителя.

Техническая документация – текст, сопровождающий программный код и описывающий различные аспекты того, как именно он функционирует. Этот тип документации имеет строго технический характер и чаще всего используется для описания API и применяемых структур данных и алгоритмов.

Ориентированной на широкий круг потребителей является пользовательская и маркетинговая документация, предназначенная для ознакомления конечного клиента с особенностями работы продукта и продвижения разработки на рынке соответственно.

При этом следует заметить, что в некоторых специфичных ситуациях один тип документации может подменять собой другой. Например, в случае разработки библиотеки алгоритмов техническая документация будет одновременно являться и пользовательской.

Особое внимание следует обратить на техническую документацию, так как традиционно, в силу теоретической возможности существования проекта и без нее, ее созданию уделяется меньшее внимание.

Основными причинами для документирования кода являются:

- сложность обслуживания недокументированного кода (особенно при командной разработке);
- документированный код, как правило, имеет лучшую структуру;
- код, не прошедший документирование, неудобен в сопровождении.

Процесс создания технического документирования кода отличается достаточно высокой трудоемкостью, поэтому для составления технической документации чаще всего применяются специализированные генераторы. Примерами таких средств являются Javadoc, Doxygen, Epydoc, Sandcastle, XHelpGen и так далее.

Генерация документации выполняется с помощью анализа исходного кода программы и выделения в нем конструкций, называемых документирующими комментариями.

Синтаксис конструкций, применяемых в документирующих комментариях, зависит от того, какой генератор документации применяется. Как правило, документирующие комментарии позволяют указывать информацию об авторе кода, описывают назначение объектов программы, входных

и выходных параметров, возможные исключительные ситуации, особенности реализации и прочее.

Рассмотрим подробнее принципы работы с генераторами на примере Javadoc.

В данном случае документации формируется в HTML-формате из комментариев и, фактически, определяет стандарт для описания классов в Java. Документирующий комментарий имеет многострочный вид и начинается с последовательности символов «/**», обычно задаваемых в отдельной строке. Далее следует описательный текст, в котором для разметки различных секций применяются конструкции, называемые тегами. Заканчивается комментарий стандартной последовательностью «*/». Каждый из элементов программного кода: класс, интерфейс, метод, поле, – в случае необходимости его пояснения, сопровождается отдельным комментарием.

Пример Javadoc-комментария и соответствующей ему документации приведен ниже.

```
public class GCD {
    /**
     * Find the greatest common divisor of two variables, using the
    Euclidean algorithm.
     * @author Gennadiy Chistyakov
     * @param arg0 First variable
     * @param arg1 Second variable
     * @return The greatest common divisor of arg0 and arg1
     * @throws NegativeException If one or both of the argument is less
    than zero
     */
    public static long gcd(long arg0, long arg1) throws NegativeException
    {
        if(arg0 < 0 || arg1 < 0)
            throw new NegativeException();
        if(arg0 == 0)
            return arg1;
        if(arg1 == 0)
            return arg0;
        return gcd(arg1 % arg0, arg0);
    }
}
```

Форма автоматически сгенерированной документации по рассмотренному методу представлена на рис. 3.

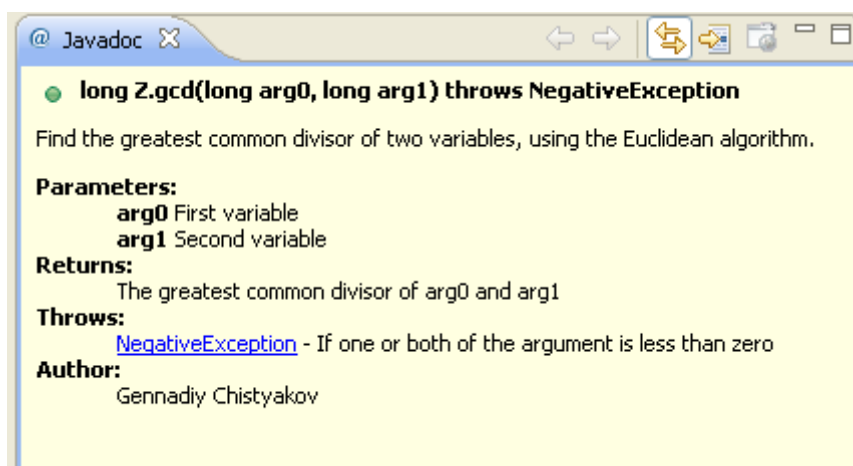


Рис. 3. Пример документации Javadoc в Eclipse

Перечень наиболее часто используемых тегов представлен в табл. 1.

Таблица 1

Перечень тегов Javadoc

Дескриптор	Описание	Область применения			
		Класс	Интерфейс	Метод	Поле
@author <i>author</i>	Дескриптор с указанием авторства.	+	+		
{ @docRoot }	Специализированный дескриптор, указывающий на корневой каталог пакета документации.	+	+	+	+
@version <i>version</i>	Дескриптор с указанием порядкового номера версии. Для каждого класса или интерфейса используется не более одного раза.	+	+		
@since <i>since-text</i>	Дескриптор с указанием порядкового номера версии, начиная с которой появился элемент.	+	+	+	+
@see <i>reference</i>	Ссылочный дескриптор,	+	+	+	+

Продолжение табл. 1

Дескриптор	Описание	Область применения			
		Класс	Интерфейс	Метод	Поле
	используемый для связи страниц документации друг с другом.				
<code>@param name description</code>	Дескриптор для описания параметров метода.			+	
<code>@return description</code>	Дескриптор для описания возвращаемого значения.			+	
<code>@exception classname description</code> <code>@throws classname description</code>	Дескрипторы для описания исключительных ситуаций, потенциально возможных при выполнении метода.			+	
<code>@deprecated description</code>	Дескриптор, указывающий на то, что сопровождаемый им элемент является устаревшим.	+	+	+	+
<code>{ @inheritDoc }</code>	Специализированный дескриптор, позволяющий унаследовать документацию от переопределяемого метода.			+	
<code>{ @link reference }</code>	Специализированный дескриптор для организации ссылочной связи.	+	+	+	+
<code>{ @linkplain reference }</code>	Специализированный дескриптор для организации ссылочной связи.	+	+	+	+
<code>{ @value #STATIC_FIELD }</code>	Дескриптор для указания значения				+

Дескриптор	Описание	Область применения			
		Класс	Интерфейс	Метод	Поле
	статического поля.				
{ @code <i>literal</i> }	Специализированный дескриптор для размещения в документации фрагментов кода.	+	+	+	+
{ @literal <i>literal</i> }	Специализированный дескриптор для размещения в документации фрагментов неформатируемого текста.	+	+	+	+
{ @serial <i>literal</i> }	Специализированный дескриптор для указания сериализуемого поля.				+
{ @serialData <i>literal</i> }	Специализированный дескриптор для сопровождения сериализуемых элементов.			+	+
{ @serialField <i>literal</i> }	Специализированный дескриптор для сопровождения сериализуемых элементов.				+

2.3 Задание на лабораторную работу

Подготовить комплект технической документации на разработанный ранее набор классов. Для выполнения лабораторной работы необходимо решить следующие задачи.

- Провести ряд преобразований программного кода, полученного в ходе выполнения предыдущей работы, с использованием встроенных средств рефакторинга.
- Сопроводить код комментариями с использованием Javadoc.
- Сгенерировать документацию к разработанным классам.

2.4 Указания к выполнению работы

Дополнительные сведения о Javadoc могут быть найдены по ссылке <https://www.oracle.com/technetwork/articles/java/index-137868.html>.

Отчет по лабораторной работе должен содержать:

- титульный лист;
- задание на лабораторную работу;
- описание проделанных преобразований над исходным кодом с указанием применяемого инструментария;
- исходный код реализованных классов и интерфейсов с комментариями Javadoc;
- сгенерированную техническую документацию;
- выводы по проделанной работе.

ЛАБОРАТОРНАЯ РАБОТА №3.

РАЗРАБОТКА ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА С ИСПОЛЬЗОВАНИЕМ ТЕХНОЛОГИИ SWING

Целью работы является получение навыков разработки графического пользовательского интерфейса с применением технологии Swing.

3.1 Концепция MVC

Концепция MVC была впервые предложена в 1979 году. Основная цель концепции заключается в разделении логики и визуализации. MVC предполагает разделение обрабатываемых данных, визуального интерфейса и управляющих действий на три компонента: модель, представление и контроллер.

Модель определяет данные предметной области и методы работы с ними. Представление отвечает за визуализацию информации. Контроллер обеспечивает связь между пользователем и системой: осуществляет контроль за вводом данных, отвечает за интерпретацию пользовательских действий и оповещает модель о необходимости изменений.

На рис. 4 представлено схематическое изображение концепции MVC. Сплошными линиями показаны прямые связи (вызов функций, передача данных), пунктирными – косвенные (через сообщения).

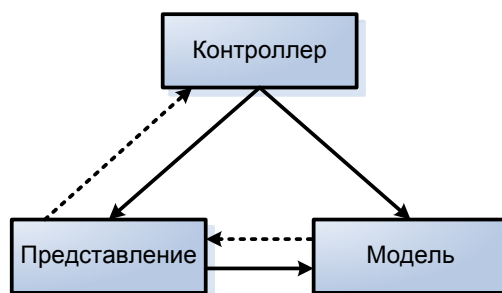


Рис. 4. Концепция MVC

Представление и контроллер зависят от модели, однако модель от них никоим образом не зависит. Тем самым достигается возможность работать с моделью независимо от ее визуального представления.

Принято выделять две модификации MVC:

- Пассивная модель. Модель не влияет на представление и контроллер, а пользуется ими только в качестве источника данных для

отображения. Все изменения отслеживаются контроллером, который, если это необходимо, отвечает за перерисовку, если это необходимо.

- Активная модель. Модель оповещает контроллер и представления о своем изменении. Заинтересованные в конкретных изменениях представления подписываются на оповещение.

Классической принято считать активную модель.

Разделяют MVC компонентного уровня (framework), и уровня приложения. Swing, с одной стороны, является реализацией идеи MVC на компонентном уровне, но при проектировании приложения можно также воспользоваться преимуществами MVC, выделив логику приложения в модель, и построив представление и контроллер на основе соответствующих классов Swing.

К достоинствам использования MVC следует, прежде всего, отнести.

- Возможность присоединения к одной модели нескольких представлений без ее модификации (одни и те же данные могут быть представлены и таблицей, и диаграммой).

- Возможность присоединения к одному представлению разных контроллеров, что позволяет получить разную реакцию на действия пользователя.

- Более высокий уровень разделения труда программистов. Разработчики, занимающиеся логикой, не связаны с разработкой представления.

- Повышение степени повторного использования кода.

Среди основных недостатков концепции следует отметить.

- В ряде случаев чрезмерное усложнения архитектуры приложения.

- Относительная сложность концепции, вызывающая ее непонимание и неправильное использование.

В основе реализации MVC лежит большое число шаблонов проектирования, основными из которых являются «наблюдатель», «стратегия» и «компоновщик».

«Наблюдатель» – это шаблон поведения объектов, определяющий связь «один ко многим» между объектами таким образом, что при изменении состояния одного объекта все объекты, зависящие от него оповещаются об этом изменении и автоматически обновляются.

Разбиение системы на множество взаимосвязанных классов приводит к необходимости поддерживать их согласованное состояние. Данный

шаблон позволяет обеспечить это без жесткой связи классов, что повышает повторную используемость кода.

Структура шаблона приведена на рис. 5.

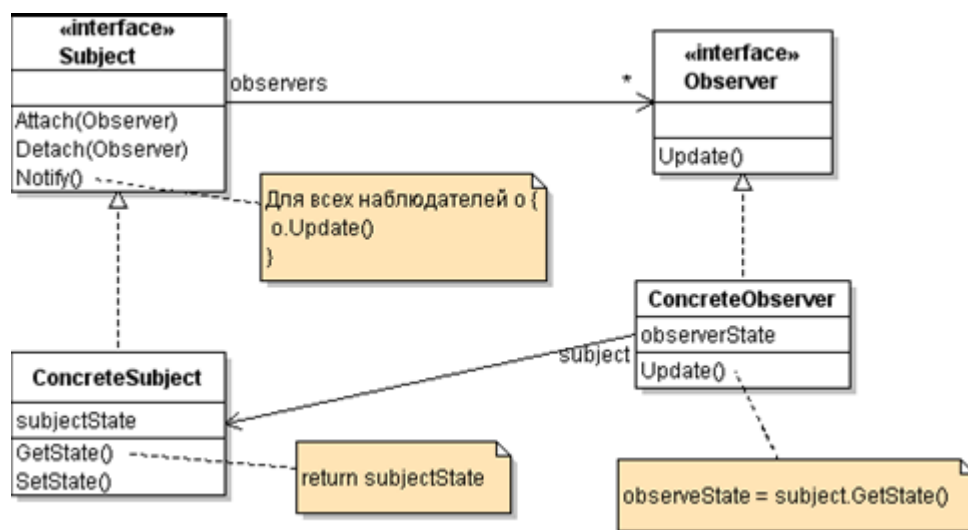


Рис. 5. Структура шаблона «наблюдатель»

Участниками являются субъект (Subject), наблюдатель (Observer), конкретный субъект (ConcreteSubject), конкретный наблюдатель (ConcreteObserver).

Субъект представляет интерфейс для присоединения и отделения наблюдателей. Также субъект обладает информацией обо всех своих наблюдателях.

Конкретный субъект сохраняет состояние, представляющее интерес для конкретного наблюдателя, а также оповещает всех наблюдателей об изменении своего состояния.

Наблюдатель определяет интерфейс для обновления состояния объектов-наблюдателей.

Конкретный наблюдатель хранит ссылку на субъект и значение состояния субъекта, за которым осуществляется наблюдение. Также конкретный наблюдатель реализует интерфейс обновления, определенный в наблюдателе.

Существенными достоинствами рассмотренного шаблона является.

- Простота модификации классов. В некоторых случаях небольшие изменения в классе субъекта могут потребовать огромных изменений в классах-наблюдателях. Данный шаблон предотвращает подобные «лавинные» изменения.

– Абстрактная связанность субъекта и наблюдателя. Субъект знает только о том, что у него есть ряд наблюдателей. Он не знает ни их типов, ни положения в иерархии классов. Таким образом, связь между субъектом и наблюдателями сведена к минимуму и носит абстрактный характер.

– Повышение повторной используемости кода. За счет высокой степени абстракции шаблон обладает высокой степенью повторной используемости.

– Обеспечение широковещательной коммуникации. Уведомление об изменении автоматически поступают всем наблюдателям, подписавшимся на него. При этом субъекту не важно, проигнорирует ли конкретный наблюдатель это сообщение или обработает.

«Стратегия» – поведенческий шаблон проектирования, определяющий семейство алгоритмов, инкапсуляции каждого из них и их взаимозаменяемость.

В некоторых ситуациях программа должна обеспечивать различные варианты поведения разных экземпляров одного класса. Иногда поведение требуется менять на стадии выполнения. Решение данной задачи возможно с помощью отделения процедуры выбора алгоритма от его реализации.

Структура шаблона приведена на рис. 6.

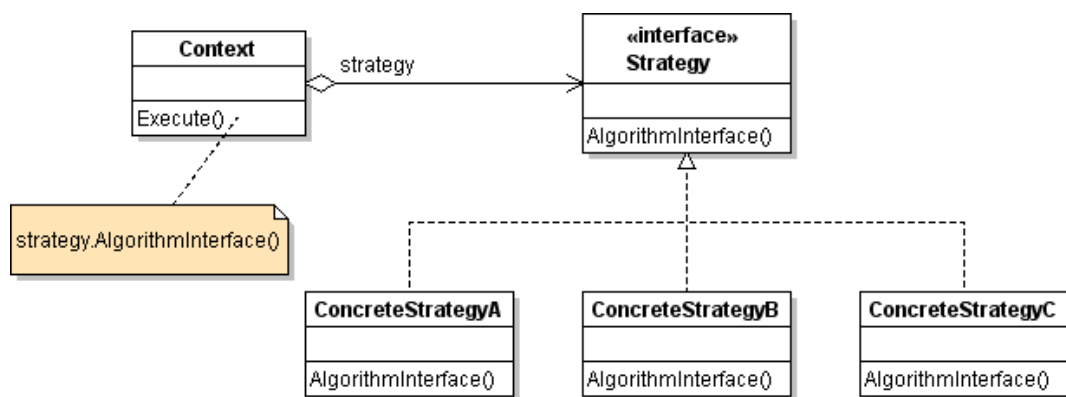


Рис. 6. Структура шаблона «стратегия»

Strategy представляет интерфейс реализации алгоритма.

ConcreteStrategy реализует алгоритм, использующий интерфейс, объявленный в Strategy.

Класс Context конфигурируется объектом класса ConcreteStrategy и хранит ссылку на объект класса Strategy. Иногда Context определяет

интерфейс, позволяющий получить объектам класса Strategy доступ к своим данным.

Шаблон обладает следующими достоинствами.

- Позволяет использовать семейство алгоритмов.
- Предотвращает порождение большого числа подклассов.
- Позволяет избавиться от условных операторов, определяющих стратегию поведения.
- Позволяет динамически выбирать реализацию.
- Позволяет определять несколько реализаций одного и того же алгоритма. Выбирать конкретную реализацию можно в зависимости от ситуации или быстроедействия.

В качестве основных недостатков можно отметить:

Увеличение числа объектов в приложении.

– Клиент должен знать о различных стратегиях. Для выбора подходящей стратегии клиент должен знать некоторые их особенности, поэтому, наверняка, клиенту придется раскрыть некоторые особенности реализаций стратегий.

– Стратегии могут различаться по сложности. Простой стратегии могут не требоваться данные от клиента, а сложной понадобится много дополнительной информации. В этом случае простой стратегии будет предана избыточная информация, что может значительно повысить накладные расходы.

«Компоновщик» – структурный шаблон проектирования, объединяющий объекты в древовидную структуру для представления иерархии от частного к целому.

Основной задачей шаблона является предоставление одинакового доступа как к объектам, так и к группам объектов.

Структура шаблона приведена на рис. 7.

Component предоставляет интерфейс для компонуемых объектов и определяет общую для всех классов операцию по умолчанию.

Leaf является листовым узлом композиции и не имеет потомков. Определяет поведение примитивных объектов композиции.

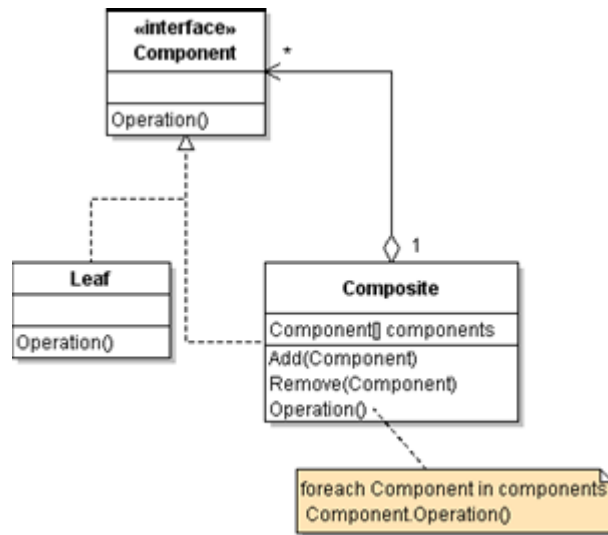


Рис. 7. Структура шаблона «компоновщик»

Composite является составным объектом. Хранит объекты-потомки и реализует относящиеся к управлению потомками (добавление или удаление) операции. Перенаправляет запросы на обработку своим потомкам.

Важнейшими достоинствами шаблона является.

- Определение иерархии классов из примитивных и составных объектов.

- Упрощение архитектуры. С помощью данного шаблона приложение может однообразно работать и с простыми, и с составными объектами. Это упрощает код, так как нет необходимости писать ветвящиеся функции.

- В случае добавления новых компонентов нет необходимости переписывать код приложения.

Самым существенным недостатком является невозможность ограничить типы используемых примитивных объектов. В некоторых ситуациях бывает необходимо, чтобы в состав контейнера входили только определенные типы примитивов. Рассмотренный шаблон не позволяет пользоваться для реализации таких ограничений статической системой типов.

Наиболее типичная реализация концепции MVC изолирует модель от вида с помощью установления между ними протокола взаимодействия на основе аппарата оповещений.

Когда происходит изменение внутренних данных в модели, она оповещает все представления, связанные с ней, с помощью шаблона «наблюдатель».

При обработке реакции пользователя представление выбирает нужный контроллер для обеспечения той или иной реакции. Выбор осуществляется с помощью шаблона «стратегия».

Однотипная работа с объектами сложно-составного вида обеспечивается благодаря использованию шаблона проектирования «компонент».

Общая схема взаимодействия компонентов приведена на рис. 8.

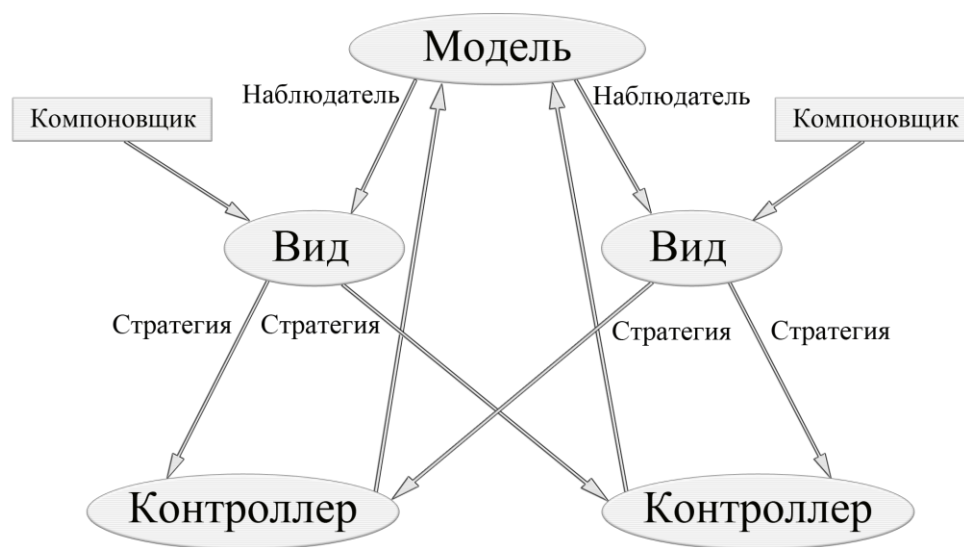


Рис. 8. Общая схема взаимодействия компонентов

3.2 Библиотека организации пользовательского интерфейса Swing

Библиотека Swing содержит более пятисот классов и интерфейсов, обеспечивающих оформления графического интерфейса. Классы для стандартного интерфейса сгруппированы в пакет `javax.swing`.

Для работы с таблицами используют пакет `javax.swing.table`.

Работу с объектами, расположенными в виде дерева организуют с помощью классов пакета `javax.swing.tree`.

При создании текстового редактора можно использовать классы из пакета `javax.swing.text`. Они обеспечивают нужную форму курсора, отслеживают и изменяют его позицию, выделяют фрагмент текста, задают формат дат и чисел и прочее.

Пакеты `javax.swing.text.html` и `javax.swing.text.rtf` позволяют текстовому редактору работать с форматами HTML и RTF, а классы пакета

`javax.swing.text.html.parser` включают средства синтаксического разбора HTML-файлов.

Для оформления рамок, ограничивающих группы компонентов, используют классы пакета `javax.swing.border`.

Классы пакета `javax.swing.plaf.*` задают внешний вид и поведение приложения (Look and Feel, L&F) в различных графических средах. Одна из особенностей библиотеки Swing – возможность изменять внешний вид и поведение графических элементов приложения. Вид (look) каждого графического компонента задают его форма, тип и цвет рамки, цвет фона, цвет, тип и размер шрифта, форма курсора мыши. Поведение (feel) определяет реакцию на действия мыши, ввод с клавиатуры, способ перемещения окон и так далее. Существует несколько сотен свойств, определяющих L&F приложения. Если сделать вид и поведение независимым от графической оболочки операционной системы, то приложение всегда будет выглядеть одинаково и в равной степени реагировать на события от мыши и клавиатуры. Возможно, наоборот привязать приложение графической среде: MS Windows, CDE/Motif, Macintosh, и приложение будет выглядеть для нее как «родное» и реагировать на внешние воздействия по правилам данной среды. Кроме того, изменение внешнего вида и поведения можно заложить в настройки приложения, сделав его изменяемым по желанию пользователя (Pluggable Look and Feel, PL&F, PLAF или `plaf`).

Большинство Swing компонентов построено по модифицированной версии Модель-Вид-Контроллер (MVC), представленной на рис. 9. Для простоты реализации и дальнейшего использования вид и контроллер объединяются в единое целое. Появляется новый элемент, называемый представителем (delegate) пользовательского интерфейса (User Interface, UI).

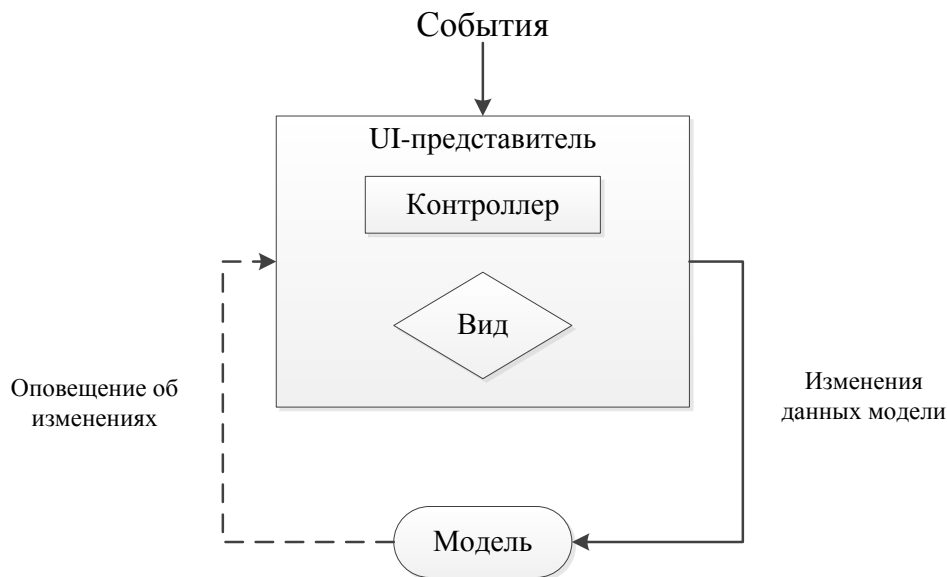


Рис. 9. Модифицированная схема MVC

Все действия пользователя поступают не в контроллер, определяющий реакцию на них, а представителю, в котором происходит значительная часть работы. Делегат определяет, нужно ли реагировать на те или иные действия и, если нужно, сразу же без генерации событий и модификации данных меняет вид, после чего представитель сообщает модели о необходимости изменения данных. Модель обновляет хранящиеся в ней данные и оповещает заинтересованных субъектов (чаще всего того же представителя). В ответ внешний вид компонента окончательно перестраивается, чтобы соответствовать новым данным.

Модели Swing, делятся на две категории:

- управляющие GUI-модели;
- модели данных приложения.

Модели GUI определяют визуальный статус элемента управления, например, нажата ли кнопка или какие элементы выбраны в списке.

Модели данных приложения определяют информацию существенную в контексте приложения, например, значение ячейки в таблице или элементы, отображаемые в списке.

В библиотеке Swing модели описаны интерфейсами, перечисляющими все необходимые методы. У каждого интерфейса есть хотя бы одна стандартная реализация, которая принимается компонентами Swing по умолчанию. Некоторые классы могут реализовать сразу несколько интерфейсов, а одни и те же интерфейсы могут быть реализованы несколькими классами.

Для реализации модели MVC библиотека Swing в качестве представителя назначает экземпляр класса с именем `xxxModel`. Класс, описывающий компонент, содержит защищенное или даже закрытое поле `model` и метод `getModel()`, для доступа к этому полю. Сложные компоненты могут иметь несколько моделей.

Делегирование полномочий используется, в том числе и для обеспечения PL&F. Класс `JComponent` содержит защищенное поле `ui`, связанное с экземпляром класса-представителя `ComponentUI` из пакета `javax.swing.plaf`, который отвечает за вывод изображения на экран в нужном виде. Класс-представитель содержит методы `paint()` и `update()`, формирующие и обновляющие графические примитивы. Представители образуют иерархию с общим суперклассом `ComponentUI`.

Класс, описывающий компонент, дублирует большинство методов модели, например, используя множество методов доступа к информации `getXxx()`, большинство из них просто обращаются к соответствующим методам модели. При построении графического интерфейса пользователя редко приходится обращаться к моделям и представителям компонента. В большинстве случаев достаточно обращаться к методам самого класса компонента. Если модель, принятая по умолчанию, не устраивает разработчика, ее можно заменить, реализовав требуемый интерфейс или расширив существующий класс `xxxModel`. Новая модель данных устанавливается методом `setModel(xxxModel)`.

При разработке экранных форм обычно необходимо указывать координаты и размеры каждого компонента, размещенного в окне. Данный метод подходит, когда известны характеристики предполагаемого устройства отображения информации. В Java имеются специальные схемы размещения (`Layout Managers`), которые гарантируют, что интерфейс будет выглядеть одинаково, не зависимо от размеров окна и разрешения экрана. Чтобы использовать любую из схем, достаточно создать ее экземпляр, а затем связать его с контейнером.

Swing предоставляет следующие схемы:

- `FlowLayout`. Построчная схема. При изменении размеров окна компоненты изменяют свое взаимное расположение.
- `GridLayout`. Организует компоненты как строки и столбцы таблицы, все компоненты имеют одинаковый размер. При этом если

размер окна приложения изменяется, то меняются размеры компонентов, а их взаимное расположение сохраняется.

- **VoxLayout.** Располагает несколько компонентов окна горизонтально или вертикально. При изменении размеров окна элементы управления не смещаются со своих позиций.

- **BorderLayout.** Разделяет пользовательское окно на пять неравных зон (южную, северную, восточную, западную и центральную), при этом каждая область заполняется полностью отдельным компонентом.

- **CardLayout.** Представляет собой набор элементов, причем в каждый момент времени активен только один из них. Используется если необходимо создать, например, компонент с вкладками.

- **GridBagLayout.** Табличный менеджер, позволяющей задавать размеры компонентов несколькими ячейками таблицы.

Таким образом, следует выделить преимущества библиотеки Swing:

- содержит богатый и удобный набор элементов пользовательского интерфейса;

- имеет отдельную архитектуру модель-представитель;

- минимально зависит от той платформы, на которой должна выполняться программа;

- обеспечивает одинаковое восприятие конечными пользователями приложений с GUI на разных платформах, поддерживает различные стили (Look and feel);

- обладает встроенным редактором форм почти во всех средах разработки и имеет встроенную поддержку HTML.

Swing делает создание GUI более легким за счет применения набора настраиваемых границ (Borders) и менеджеров размещения (LayoutManagers).

3.3 Задание на лабораторную работу

Разработать графическое приложение с использованием библиотеки Swing. Для выполнения лабораторной работы необходимо решить следующие задачи.

Выбрать и согласовать с преподавателем задачу, для решения которой может быть использована программа, разработанная в ходе предыдущей лабораторной работы.

Разработать программу для решения выбранной задачи (взаимодействие с пользователем должно осуществляться с применением графического интерфейса).

3.4 Указания к выполнению работы

Дополнительные сведения по менеджерам компоновки могут быть найдены по ссылке <https://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html>.

Информация по работе со Swing-компонентами находится по адресу <https://docs.oracle.com/javase/10/docs/api/javax/swing/package-summary.html>.

В ходе разработки интерфейса следует использовать не менее двух различных менеджеров компоновки.

Отчет по лабораторной работе должен содержать:

- титульный лист;
- задание на лабораторную работу;
- диаграммы, описывающий разработанный интерфейс;
- эскизные макеты интерфейса;
- программный код, обеспечивающий взаимодействия пользователя с компонентами Swing;
- экранные формы;
- выводы по проделанной работе.

ЛАБОРАТОРНАЯ РАБОТА №4.

ЗНАКОМСТВО С ИНСТРУМЕНТОМ АВТОМАТИЗАЦИИ РЕШЕНИЯ ЗАДАЧ APACHE ANT

Целью работы является знакомство с базовыми возможностями средства автоматизации решения повседневных задач Apache Ant.

4.1 Автоматизация процесса сборки программных продуктов

В настоящее время под сборкой проектов понимают этап написания скриптов для решения повседневных задач создания ПО: работа с файловой системой, компиляция исходного кода в бинарные файлы, выполнение тестов, отправка отчетов о проделанных действиях по электронной почте и так далее.

Все описанные задачи могут быть решены с помощью командной строки, однако с ростом их количества и частоты выполнения, такое решение представляется крайне неудобным.

Для автоматизации рассматриваемых процессов активно применяются как скрипты, содержащие в себе только последовательности необходимых операций, так и специальные программные средства, позволяющие описывать целые сценарии выполнения требуемых задач.

Одной из первых утилит, автоматизирующих процесс преобразования файлов, является утилита make. Make была разработана в 1977 году Стюартом Фельдманом в Bell Labs. В настоящее время она включена практически во все дистрибутивы Unix.

Программа make выполняет команды согласно правилам, указанным в специальном make-файле. Как правило, в нем описывается каким образом нужно компилировать и компоновать программу.

Еще одной распространенной утилитой для автоматизации сборки является Apache Ant. Ant является кроссплатформенным аналогом Unix-утилиты make. Для работы Ant требуется, чтобы в системе была установлена лишь JRE.

Ant является теговым языком, который обрабатывает организованные особым образом XML-файлы. Каждый тег, по сути, является Java-классом. Причем возможно создание собственных тегов.

Сценарий представляет собой детальный план сборки, включающий ряд операндов для выполнения команд копирования, удаления и пере-

мещения файлов, компиляции java-файлов, формирование документации к коду и исполняемого jar-файла.

Корневой элемент сценария проект (project) содержит три необязательных атрибута:

- name – имя проекта;
- default – цель проекта, определяемая по умолчанию;
- basedir – базовая директория, относительно которой вычисляются все пути.

Проект содержит хотя бы одну цель, которая представляет собой набор необходимых для выполнения задач. Цель выбирается при запуске Ant, либо выполняется цель, заданная по умолчанию. Элемент, описывающий цель проекта (target) содержит следующие атрибуты:

- name – обязательный атрибут имя цели;
- depends – промежуточные цели, от которых зависит данная цель; (имена перечисляются через запятую);
- if – условие запуска цели, определяет какие свойства должны быть установлены;
- unless – условие запуска цели, определяет какие свойства должны сброшены;
- description – краткое описание цели.

Цель, по сути, является функцией, в которой размещаются задачи, что обеспечивает возможность их повторного использования без дублирования.

Примерами целей можно считать clean (удаление промежуточных файлов), compile (компиляция всех классов), deploy (развертывание приложения на сервере) и прочие. Конкретный набор целей и их взаимосвязи зависят только от специфики проекта.

Все target-элементы являются составными и содержат действия, необходимые для успешного выполнения той или иной операции в рамках сборки проекта. Выполнение каждого последующего target'a невозможно без завершения всех предыдущих. Поэтому между целями определяется отношение зависимости. Например, имеются две цели: для компиляции и для изъятия файлов из базы данных. Скомпилировать файлы можно только после того, как они будут загружены. Цели, указанные как зависящие от каких-то других с помощью установки атрибута depends, будут запускаться только при условии выполнения всех целей, указанных в этом

атрибуте. Цель исполняется только один раз, даже если от нее зависит более чем одна цель.

Каждая цель включает задачи (tasks). Каждой задаче в рамках сценария присваивается уникальный идентификатор (id), по которому она затем может вызываться. Задача соответствует инструкциям командной строки. Группа задач может выполняться последовательно с помощью указания целей.

В поставку Ant входит множество библиотек, затрагивающих широкий набор задач. Стандартная версия содержит более 150 типов задач. В табл. 2 приведено краткое описание основных из них.

Таблица 2

Перечень типовых задач Apache Ant

Имя задачи	Описание
Archive Tasks	
Jar	Упаковывает набор файлов в Jar
Unzip	Распаковывает zip
Zip	Создает архивы zip
Compile Tasks	
Javac	Компилирует определенные исходные файлы внутри запущенной виртуальной машины или с помощью новой VM, если определен атрибут fork
JspC	Запускает JSP-компилятор. Используется для предварительной компиляции JSP-страниц с целью более быстрого запуска с сервера, или при отсутствии JDK на нем, для проверки синтаксиса, без установки на сервер
Wljspc	Компилирует JSP-страницы, используя Weblogic JSP компилятор
Execution Tasks	
Ant	Запускает Ant для выбранного файла сценария, с передачей параметров (или их новых значений). Эта задача может быть использована для запуска подпроектов
AntCall	Запускает другую цель внутри того же сценария
Exec	Исполняет системную команду. Причем, если атрибут os определен, команда исполняется, только если Ant запущен под определенную

Имя задачи	Описание
	систему
Java	Исполняет Java класс внутри запущенной виртуальной машины, если определен атрибут fork
File Tasks	
Copy	Копирует файл или директорию
Delete	Удаляет один файл или файлы, а так же и поддиректории в определенном каталоге.
Mkdir	Создает директорию при необходимости.
Move	Переносит в новую директорию файл, каталог, или набор(ы) файлов
Miscellaneous Tasks	
Echo	Выводит текст в System.out или в файл
Fail	Выходит из текущей сборки, генерируя исключение
Input	Позволяет пользователю интерактивно вмешиваться в процесс сборки с помощью вывода сообщений и считывания строки с консоли
Taskdef	Добавляет задачу в текущий проект
Property Tasks	
Available	Устанавливается, если определенный файл, каталог, или JVM системный ресурс доступен во время выполнения
Condition	Устанавливается, если определенное условие выполняется
LoadFile	Загрузка файла в параметр
Property	Устанавливает параметр по имени и значению, в проект

Параметр свойства (**property**) определяет пару имя/значение, многократно используемую в сценарии подобно переменным. Свойства можно определять как внутри файла сценария, так и в отдельных файлах. Свойства могут включать следующие атрибуты:

- name – имя свойства;
- value – значение свойства;
- location – устанавливает абсолютный путь. Если относительный путь, то подставляется базовая директория. Символы / и \ меняются автоматически в зависимости от платформы;

- resource – имя ресурса содержащего настройки в специальном формате;
- file – путь к файлу настройки;
- url – адрес настройки;
- environment – префикс используемый для организации доступа к переменным окружения. Например, если определено `myenv`, то к переменным обращаются как «`myenv.PATH`»;
- classpath – путь к ресурсам;
- prefix – префикс добавляется к свойствам, загруженным из файла, ресурса, или url. По умолчанию – префикс «.».

Свойства используются для определения переменных в файлах сборки. Однако их значения не могут быть изменены в ходе сценария. С помощью параметра `property` можно определить условия выполнения цели. Это позволяет контролировать процесс сборки, в зависимости, например, от операционной системы и версии Java. Ant только проверяет, установлено ли некоторое свойство, значение его не учитывается.

Ant позволяет определять собственные типы заданий путем создания Java-классов, реализующих определенные интерфейсы. Если одни и те же операции необходимо выполнить с группой файлов, удобно применять шаблоны. Пример Ant-скрипта представлен ниже.

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="exampleProject" default="finalize">
  <description>
    This is a simple example using Ant
  </description>

  <target name="makeJAR" depends="compile, init">
    <jar destfile="package-${DSTAMP}.jar" basedir="tmp">
      <manifest>
        <attribute name="Main-class" value="Z"/>
      </manifest>
    </jar>
  </target>
  <target name="compile" depends="init">
    <javac srcdir="src" destdir="tmp" debug="true" includes="Z.java"/>
  </target>

  <target name="init">
    <mkdir dir="tmp"/>
    <tstamp/>
  </target>
```



```
<target name="finalize" depends="makeJAR">
  <delete dir="tmp"/>
</target>
</project>
```

4.2 Задание на лабораторную работу

Разработать скрипт Apache Ant для автоматической сборки проекта, реализованного в ходе выполнения предыдущей лабораторной работы.

4.3 Указания к выполнению работы

Дополнительные сведения по Apache Ant могут быть найдены по ссылке <http://ant.apache.org/manual/index.html>.

Отчет по лабораторной работе должен содержать:

- титульный лист;
- задание на лабораторную работу;
- табличное описание использованных в ходе разработке скрипта целей;
- текст разработанного скрипта;
- выводы по проделанной работе.

ЛАБОРАТОРНАЯ РАБОТА №5.

ОРГАНИЗАЦИЯ ПРОЦЕССА РАЗРАБОТКИ ПО С ПРИМЕНЕНИЕМ СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ

Целью работы является освоение метода установки расширений для IDE Eclipse, а также получение практических навыков взаимодействия с системой контроля версий Subversion.

5.1 Принципы функционирования систем контроля версий

Система контроля версий (или система управления версиями) – это программное обеспечение, предназначенное для работы с часто изменяющейся информацией.

Основные функции, возлагаемые на систему контроля версий (СКВ):

- поддержка возможности хранения нескольких версий одного и того же документа;
- возможность возврата к предыдущим версиям документа.

Наибольшую популярность СКВ получили в следующих областях:

- разработка программного обеспечения;
- САПР;
- конфигурационное управление (комплекс методов для систематизации изменений, вносимых в ПО).

В процессе работы типичной бывает ситуация, когда документ претерпевает ряд изменений. Иногда бывает полезно иметь не только последнюю версию документа, но и несколько таких версий (зачастую – все версии документа за все время его существования). Именно для этой задачи и разработан инструмент СКВ.

СКВ позволяют выбирать и получать нужную копию документа по некоторым параметрам (например, по дате создания).

В простейшем случае можно организовать хранение всех созданных версий документа за все время его существования, однако такой способ организации СКВ избыточен.

Большая часть современных СКВ основана на алгоритмах сжатия данных, наибольшей популярностью из которых пользуется алгоритм дельта-компрессии. Его идея заключается в том, что, помимо первой версии документа, имеет смысл хранить только последовательность изменений, примененных к документу. При этом для повышения

надежности системы иногда выполняется сохранение полноценных копий документа, что позволяет восстановить практически все данные в случае повреждения репозитория.

Традиционным способом организации СКВ считается централизованная модель. Данная модель подразумевает наличие некоторого сервера, который осуществляет управление единым хранилищем документов. Сервер также обрабатывает запросы, поступающие от пользователей, работающих с документами.

Основные операции, поддерживаемые современными СКВ:

- Извлечение версии. Позволяет разработчику получить заданную версию проекта. После поступления запроса на извлечение заданной версии устанавливается соединение с сервером и проект копируется на локальный компьютер пользователя. Чаще всего создается две локальных копии проекта: в первой пользователь осуществляет изменения, а вторая используется в качестве эталона, позволяя в любой момент, без обращения к серверу, определить какие изменения были произведены пользователем.

- Обновление рабочей копии. Рабочая копия со временем устаревает, поэтому имеет смысл выполнять ее обновление.

- Фиксация изменений. После того, как разработчик завершил работу, необходимо зафиксировать изменения. СКВ устанавливает связь с сервером и выполняет слияние рабочей копии пользователя с копией на сервере.

- Создание ветви проекта. Взаимодействовать напрямую с рабочей копией проекта не всегда бывает удобно. Для внесения в проект крупных изменений разумным решением является создание второй копии. В таком случае такая копия называется «ответвлением». Чаще всего работу с веткой выполняет группа разработчиков, задачей которой является внесение крупных модификаций для достижения определенных целей (например, внесение в проект нового функционала). После того, как модификации ветви будут завершены, необходимо выполнить слияние с текущей копией рабочей версии.

- Слияние версий. Выполняется как после небольших изменений в рабочей версии проекта (фиксация изменений), так и при слиянии ветвей проекта в единую копию. В случае, когда изменения были внесены только в рабочую копию проекта или только в ветвь (или локальную копию разработчика), слияние представляет собой легко решаемую задачу, если

же изменения были внесены и в рабочую копию проекта и в ветвь, то слияние становится нетривиальной операцией, чаще всего требующей вмешательства разработчика.

- Блокировка доступа. В некоторых случаях позволяет решить проблему возникновения конфликтов на этапе слияния версий. Разработчик, обладающий достаточными привилегиями, может временно ограничить доступ других пользователей к некоторым файлам рабочей копии (на запись). Таким образом, он становится «монополистом», а слияние копий гарантированно пройдет без конфликтов.

Принято выделять четыре основных типа ветвей:

- Стволовая ветвь. Основная ветвь, создается один раз в самом начале проекта. Все остальные ветви берут свое начало от нее. Хорошим тоном разработки считается поддерживать в стволовой ветви рабочую копию проекта (то есть ту ветвь, которая на данный момент является наиболее стабильной и представляет собой законченную часть разрабатываемого программного продукта).

- Релизная (release) ветвь. Создается перед выпуском очередной версии программного продукта. Не допускает внесения дестабилизирующих изменений.

- Функциональная ветвь. Создается как ответвление от стволовой ветви для внесения в проект дополнительного функционала, который не может быть интегрирован в стволовую ветвь за одну итерацию. Может длительное время пребывать в нестабильном состоянии. После реализации необходимого функционала выполняется слияние со стволовой ветвью. Таким образом, для стволовой ветви внесение функционала выполняется за одну итерацию.

- Форк (fork). Разделение стволовой ветви на два независимых проекта. В некоторых случаях проекты в дальнейшем вновь объединяются.

С технической точки зрения, определенные сложности вызывает возникновение конфликтной ситуации, когда при слиянии нескольких версий проекта изменения, сделанные в них, пересекаются между собой.

При возникновении конфликта СКВ попытается разрешить его автоматически. В этом случае большая часть современных СКВ руководствуется следующими правилами.

- Если изменения относятся к несвязанным файлам или каталогам, то они объединяются автоматически.

- Операции со связанными файлами и каталогами объединяются автоматически, если они не конфликтуют. Конфликтующими являются случаи удаления и изменения одного и того же объекта файловой системы, удаление и переименование одного и того же объекта, создание объекта с одинаковым именем и разным содержимым.

- Изменения в пределах текстового файла объединяются автоматически, если они не пересекаются.

- Изменения в пределах бинарного файла всегда конфликтуют.

Если при слиянии копий возникает конфликт, то СКВ предлагает разработчику решить его вручную. При этом СКВ предоставляет специальный инструмент для объединения изменений в единую версию.

Важным моментом с точки зрения организации процесса взаимодействия программиста с СКВ является понятие номера версии. В настоящее время применяется два подхода к нумерации:

- версионность файлов;
- версионность проекта.

В первом случае каждая версия каждого файла проекта имеет свой номер, уникальный в рамках репозитория. Часто для идентификации версии проекта используется система тегов. Набор файлов и каталогов помечается определенным тегом (символьной меткой), что свидетельствует о принадлежности данного набора к определенной версии проекта.

Во втором случае уникальный номер сопоставляется всему проекту в целом. При изменении одного файла создаются новые копии всех остальных объектов файловой системы, и всему проекту присваивается следующий номер.

Как было отмечено выше, в настоящий момент наибольшей популярностью пользуются СКВ с централизованной архитектурой.

Схема централизованной архитектуры СКВ представлена на рис. 10.

Цикл работы с централизованной СКВ прост: загрузка разработчиком рабочей версии проекта, изменение, фиксация изменений. Созданные ветви проекта хранятся в центральном репозитории.

Вторым существующим вариантом организации СКВ является распределенная архитектура. Схема СКВ с распределенной архитектурой представлена на рис. 11.

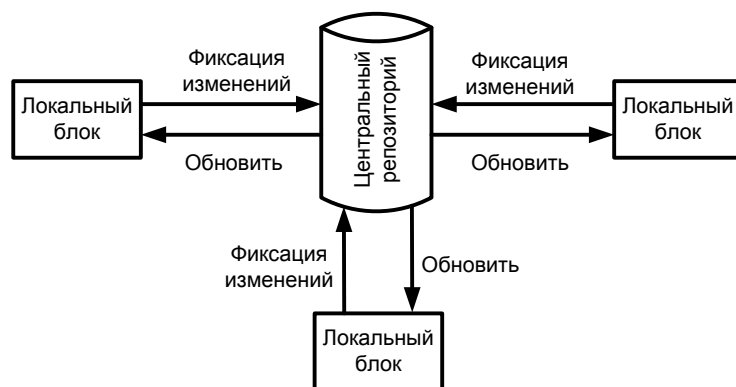


Рис. 10. Централизованная СКВ

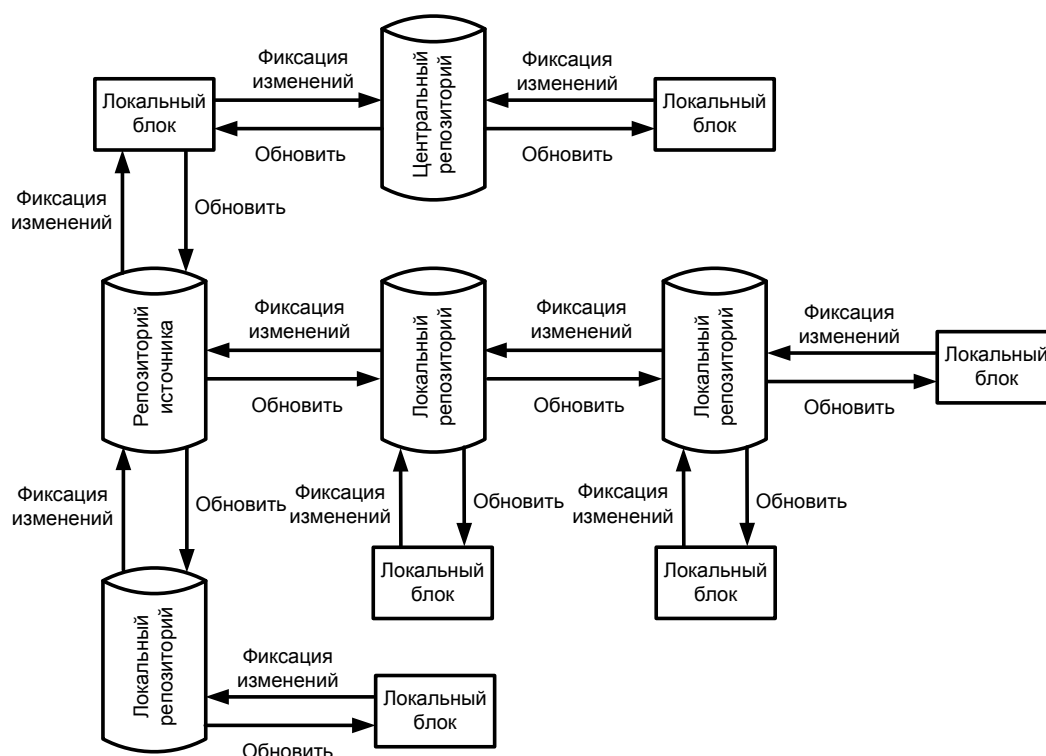


Рис. 11. Распределенная СКВ

При использовании распределенной архитектуры допускается создание локальных репозиториях, используемых группами связанных между собой разработчиков. После того, как работа над частью проекта завершена и локальное хранилище становится не нужным, изменения, сделанные группой сливаются с вышестоящим репозиторием, а вышестоящие программисты получают доступ к проделанным изменениям. В распределенной модели взаимодействия центральный репозиторий может и вовсе отсутствовать. Тогда синхронизация между двумя

независимыми хранилищами осуществляется с помощью обмена патчами (наборами изменений).

Таким образом, вдобавок к вышеописанным основным функциям, распределенные СКВ должны обеспечивать выполнение двух дополнительных действий.

- Получение репозитория от удаленного компьютера. Выполняется слияние имеющейся копии проекта с копией проекта в удаленном репозитории. Возможно возникновение конфликтных ситуаций, требующих для разрешения вмешательства разработчика.

- Передача своего репозитория на удаленный компьютер. Копия, полученная после слияния, сохраняется в удаленном хранилище. Возможно возникновение конфликтных ситуаций, требующих вмешательства разработчика.

Основным достоинством распределенной архитектуры СКВ является большая гибкость и автономность рабочего места. К недостаткам относится увеличение требуемой внешней памяти и невозможность совершать некоторые действия, характерные для клиент-серверной СКВ (например, блокировка файлов).

Одной из наиболее популярных централизованных СКВ является Subversion (SVN), изначально разрабатываемая как замена широко распространенной, но устаревшей к 2004 году морально CVS.

SVN распространяется свободно, под лицензией Apache 2.0; официальной документацией к системе является справочное пособие издательства O'Reilly Media, размещенное на сайте <http://svnbook.red-bean.com>.

К наиболее важным особенностям SVN следует отнести.

- Полноценную поддержку ветвления.
- Прозрачную поддержку многопользовательской работы с репозиторием.
- Использование атомарных транзакций при фиксации изменений, положительным образом сказывающихся на обеспечении целостности данных.
- Эффективно реализованную работу с текстовыми файлами.
- Обеспечение доступа к репозиторию по нескольким протоколам.
- Повышенную надежность за счет возможности зеркалирования хранилища.

При сохранении очередных версий документов применяется алгоритм дельта-компрессии: система находит отличия новой версии от предыдущей и записывает только их, избегая дублирования данных.

В Subversion используется нестандартная концепция меток: метка, как это обычно принято, не является символическим именем, адресующим набор файлов, а представляет собой копию набора файлов и их состояния.

Структура проекта Subversion может быть любой, однако на практике рекомендуется придерживаться рекомендаций, сформулированных в справочном руководстве.

5.2 Задание на лабораторную работу

Необходимо сконфигурировать рабочее окружение с системой контроля версий Subversion, а также провести ряд преобразований над проектом, полученным ранее. Для выполнения лабораторной работы необходимо решить следующие задачи.

- Установить расширение Subversive для Eclipse.
- Зарегистрировать проект, выполненный в ходе предыдущей лабораторной работы, на ресурсе, предоставляющем пространство для проектов Subversion.
- Добавить ссылку на созданный репозиторий в Eclipse.
- Синхронизировать репозиторий с проектом.
- Изменить функционал проекта.
- Обновить файлы в репозитории.
- Откатить сделанные изменения с помощью информации из репозитория.
- Создать побочную ветвь проекта и произвести конфликтующие изменения в стволовой и побочной ветвях.
- Разрешить возникший конфликт посредством встроенных инструментов Subversion.

5.3 Указания к выполнению работы

Пространство для проектов Subversion предоставляется практически всеми крупными веб-сервисами для хостинга IT-проектов (например, <https://sourceforge.net>).

Информация по работе с Subversion может быть найдена на ресурсе <http://svnbook.red-bean.com>.

Информация по работе с Subversive может быть найдена на ресурсе <http://www.eclipse.org/subversive>.

Установка расширений в Eclipse выполняется с помощью пункта Help – Install New Software.

Отчет по лабораторной работе должен содержать:

- титульный лист;
- задание на лабораторную работу;
- детализированное описание хода выполнения лабораторной работы текст разработанного скрипта;
- выводы по проделанной работе.

СПИСОК ЛИТЕРАТУРЫ

1. Шилдт, Г. Java. Полное руководство [Текст] / Г. Шилдт. – 10-е изд. – Москва : Диалектика, 2018. – 1488 с.
2. Приемы объектно-ориентированного проектирования. Паттерны проектирования [Текст] / Э. Гамма [и др.]. – Санкт-Петербург : Питер, 2016. – 366 с.
3. Рефакторинг. Улучшение проекта существующего кода [Текст] / М. Фаулер [и др.] ; пер. И. Красикова. – Москва : Вильямс, 2017. – 448 с.
4. Кериевски, Д. Java. Рефакторинг с использованием шаблонов [Текст] / Д. Кериевски. – Москва : Вильямс, 2016. – 400 с.
5. Лафоре, Р. Java. Структуры данных и алгоритмы в Java [Текст] / Р. Лафоре. – Санкт-Петербург : Питер, 2018. – 704 с.
6. Pilato, M. Version Control with Subversion. Next Generation Open Source Version Control [Text] / M. Pilato, B. Collins-Sussman, B. Fitzpatrick. – 2-nd Edition. – New York : O'Reilly Media, 2018. – 432 p.
7. Коллинз-Сассман, Б. Управление версиями в Subversion [Электронный ресурс] / Б. Коллинз-Сассман, Б. Фитцпатрик, М. Пилато. – Режим доступа: <http://svnbook.red-bean.com/nightly/ru/svn-book.html>. – 18.10.2018.
8. Eclipse Subversive – Subversion (SVN) Team Provider [Electronic resource]. – Access mode : <http://www.eclipse.org/subversive/>. – 18.10.2018.
9. Apache Ant 1.10.5 Manual [Electronic resource]. – Access mode : <http://ant.apache.org/manual/index.html>. – 18.10.2018.
10. A Visual Guide to Layout Managers [Electronic resource]. – Access mode : <https://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html>. – 18.10.2018.
11. Package javax.swing [Electronic resource]. – Access mode : <https://docs.oracle.com/javase/10/docs/api/javax/swing/package-summary.html>. – 18.10.2018.
12. How to Write Doc Comments for the Javadoc Tool [Electronic resource]. – Access mode : <https://www.oracle.com/technetwork/articles/java/index-137868.html>. – 18.10.2018.

Учебное издание

Чистяков Геннадий Андреевич
Долженкова Мария Львовна

ВВЕДЕНИЕ В JAVA-ТЕХНОЛОГИИ

Учебно-методическое пособие

Авторская редакция
Технический редактор А. Е. Свинина

Подписано в печать 18.10.2018. Печать цифровая. Бумага для офисной техники.
Усл. печ. л. 3,22. Тираж 50 экз. Заказ № 5456.

Федеральное государственное бюджетное образовательное учреждение высшего
образования «Вятский государственный университет».

610000, г. Киров, ул. Московская, 36, тел.: (8332) 74-25-63, <http://vyatsu.ru>

