

Министерство образования Республики Беларусь  
Учреждение образования «Витебский государственный  
университет имени П.М. Машерова»  
Кафедра прикладного и системного программирования

**С.А. Ермоченко, Е.А. Корчевская**

# **ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

*Методические рекомендации*

*Витебск  
ВГУ имени П.М. Машерова  
2023*

УДК 004.432(075.8)  
ББК 32.972.1я73  
Е74

Печатается по решению научно-методического совета учреждения образования «Витебский государственный университет имени П.М. Машерова». Протокол № 6 от 10.03.2023.

Авторы: доцент кафедры прикладного и системного программирования ВГУ имени П.М. Машерова, кандидат физико-математических наук, доцент **С.А. Ермоченко**; заведующий кафедрой прикладного и системного программирования ВГУ имени П.М. Машерова, кандидат физико-математических наук, доцент **Е.А. Корчевская**

Р е ц е н з е н т ы :

заведующий кафедрой информационных систем и технологий УО «ВГТУ», кандидат технических наук, доцент *В.Е. Казаков*; доцент кафедры информационных технологий и управления бизнесом ВГУ имени П.М. Машерова, кандидат физико-математических наук *Е.А. Витько*

**Ермоченко, С.А.**

**Е74** Проектирование программного обеспечения : методические рекомендации / С.А. Ермоченко, Е.А. Корчевская. – Витебск : ВГУ имени П.М. Машерова, 2023. – 52 с.

В методических рекомендациях изложены основные принципы применения объектно-ориентированного программирования и проектирования, рассмотрены некоторые типовые решения наиболее часто встречающихся задач (шаблоны проектирования), а также уделено внимание такому аспекту проектирования программного обеспечения, как влияние на организацию командной работы над крупным проектом.

Предназначается для студентов первой ступени высшего образования специальностей: «Информационные системы и технологии» (дисциплины «Объектно-ориентированное проектирование и программирование», «Моделирование программного обеспечения» и «Средства и технологии анализа и разработки информационных систем»); «Программная инженерия» (дисциплины «Объектно-ориентированные технологии программирования и стандарты проектирования» и «Технологии взаимодействия с базами данных»); «Прикладная информатика» (дисциплина «Промышленное программирование»).

УДК 004.432(075.8)  
ББК 32.972.1я73

© Ермоченко С.А., Корчевская Е.А., 2023  
© ВГУ имени П.М. Машерова, 2023

## СОДЕРЖАНИЕ

<b>ВВЕДЕНИЕ</b> .....	4
<b>1. Язык моделирования UML</b> .....	5
1.1. Проектирование и язык моделирования UML .....	5
1.2. Диаграмма вариантов использования .....	6
1.3. Диаграмма классов .....	10
<b>2. Принципы проектирования SOLID</b> .....	14
2.1. Принцип единственной ответственности .....	14
2.2. Принцип открытости / закрытости .....	15
2.3. Принцип подстановки Лисков .....	15
2.4. Принцип инверсии зависимости .....	16
2.5. Принцип изоляции интерфейса .....	17
2.6. Задания на лабораторную работу №1 .....	17
<b>3. Шаблоны проектирования GRASP</b> .....	26
3.1. Задания на лабораторную работу №2 .....	29
<b>4. Порождающие шаблоны проектирования GoF</b> .....	30
4.1. Задание на лабораторную работу №3 .....	30
4.2. Задание на лабораторную работу №4 .....	38
<b>5. Структурные шаблоны проектирования GoF</b> .....	40
5.1. Задание на лабораторную работу №5 .....	40
5.2. Задание на лабораторную работу №6 .....	48
<b>СПИСОК РЕКОМЕНДОВАННЫХ ИСТОЧНИКОВ</b> .....	51

## ВВЕДЕНИЕ

Проектирование программного обеспечения – это важный этап жизненного цикла разработки программного обеспечения, на котором определяются архитектура системы в целом и структура отдельных её частей и модулей. То, каким будет разработанный проект системы, во многом обеспечивает успешность всей работы. Цена ошибки на данном этапе очень высока, а потому особенно актуально сформировать у студентов необходимые знания, умения и навыки работы, продемонстрировать специфику документации, изучить применяемые на этом этапе инструменты, языки, подходы и принципы.

Основополагающими принципами проектирования являются так называемые принципы объектно-ориентированного проектирования SOLID.

Для оформления документации как результата этапа проектирования часто используется унифицированный язык моделирования UML.

Для экономии времени как на поиск некоторого архитектурного или проектного решения, так и на его документирование (а в дальнейшем ещё и на поддержание разработанной документации в актуальном состоянии) в проектах зачастую применяются отдельные хорошо зарекомендовавшие себя типовые решения, называемые шаблонами проектирования. Наиболее популярны GRASP и GoF.

В предлагаемых методических рекомендациях рассматриваются описанные выше принципы, языки и подходы на примерах решения практических задач, а также предлагаются разноуровневые задания на лабораторные работы для закрепления полученных теоретических знаний на практике.

Материал соответствует темам учебных программ курсов: «Объектно-ориентированное проектирование и программирование», «Моделирование программного обеспечения» и «Средства и технологии анализа и разработки информационных систем» (специальность «Информационные системы и технологии»); «Объектно-ориентированные технологии программирования и стандарты проектирования» и «Технологии взаимодействия с базами данных» (специальность «Программная инженерия»); «Промышленное программирование» (специальность «Прикладная информатика»).

# 1. ЯЗЫК МОДЕЛИРОВАНИЯ UML

## 1.1. Проектирование и язык моделирования UML

На этапе проектирования решаются следующие основные задачи.

– Проектирование модели предметной области. При этом разрабатываются структуры данных, которые будут хранить информацию о предметной области, структурируя её понятным для разработчиков способом. Для объектно-ориентированного проектирования это, прежде все, разработка структуры классов, поля которых будут хранить необходимую информацию, и определение взаимосвязей между этими классами. Также при решении этой задачи проектируется структура постоянного хранилища данных. В подавляющем количестве случаев для этого используется одна из систем управления базами данных. В большинстве случаев – это некая серверная система управления реляционными базами данных, поддерживающая структурированный язык запросов SQL (Oracle, Microsoft SQL Server, PostgreSQL, MySQL и т.д.). В последнем случае под проектированием структуры постоянного хранилища понимается разработка ER-диаграммы базы данных и создание схемы базы данных (создание таблиц, ключей и других ограничений).

– Проектирование аппаратного обеспечения. При решении этой задачи осуществляется выбор компонент вычислительной системы, на базе которой будет разворачиваться информационная система. Прежде всего, это актуально для распределённых высоконагруженных информационных систем, которые способны одновременно обрабатывать большое число клиентских запросов. Для таких систем необходимо подобрать необходимое серверное оборудование, подходящее под планируемую нагрузку. А также спроектировать компьютерную сеть, объединяющую сервера в единую вычислительную систему.

– Проектирование программного обеспечения. При решении этой задачи проектируется набор модулей и способы взаимодействия между собой. При использовании объектно-ориентированного проектирования разрабатываются абстрактные классы и интерфейсы, обеспечивающие функционирование программного обеспечения на верхнем уровне абстракций. Эта часть проектирования называется проектированием архитектуры программного обеспечения. Также на данном этапе может проектироваться и внутреннее устройство компонентов программного обеспечения, но чаще всего это выполняется уже на этапе разработки. Также на данном этапе продолжается (или начинается) проектирование пользовательского интерфейса приложения.

За решение перечисленных задач на этапе проектирования отвечают:

– бизнес-архитектор (Business Architect), который отвечает за проектирование так называемых бизнес-процессов, то есть основной биз-

нес-логики функционирования информационной системы (данный специалист должен быть, в первую очередь, экспертом в области разработки программного обеспечения, но так же и иметь опыт работы с данной предметной областью, именно он отвечает за её моделирование);

– системный архитектор (System Architect), который отвечает за проектирование аппаратного и программного обеспечения (у таких специалистов может быть своя узкая специализация в области построения компьютерных сетей, проектирования распределённых приложений, в области объектно-ориентированного проектирования и т.д.);

– администратор баз данных (DataBase Administrator – DBA), специалист в области использования баз данных, в первую очередь реляционных баз данных с поддержкой SQL-запросов (отвечает за проектирование схемы базы данных, её оптимизации для специфики разрабатываемой информационной системы и т. п.);

– ведущий разработчик (Lead Developer), в основном работает на этапе разработки (программирования), но так как на этапе проектирования, как правило, программное обеспечение не проектируется в полном объёме, проектируются лишь общие модули и интерфейсы их взаимодействия, то детализируется проект на следующем этапе, где за такие работы отвечает, прежде всего, ведущий разработчик.

Основным инструментом на этапе проектирования программного обеспечения является унифицированный язык моделирования (Unified Modeling Language – UML). Этот язык состоит из различных графических диаграмм. Графическая форма языка позволяет быстрее создавать проектные решения и быстро и интуитивно их воспринимать. Рассмотрим наиболее популярные виды UML-диаграмм, применяющиеся при проектировании информационных систем.

## 1.2. Диаграмма вариантов использования

Диаграмма вариантов использования (Use Case Diagram, или диаграмма прецедентов) отражает отношения между *актёрами* и *вариантами использования* (прецедентами).

Прецедент – это некоторая возможность моделируемой системы, благодаря которой пользователь может получить конкретный, измеримый и нужный ему результат. Используется для спецификации пользовательского требования к приложению. Показывает, что можно сделать, но не как. Каждый вариант использования представляется на диаграмме в виде овала, в который вписывается, что может сделать пользователь. Примеры вариантов использования приведены на рисунке ниже.

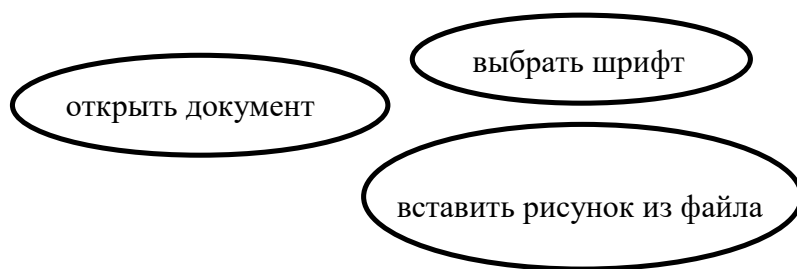


Рисунок 1.1. Примеры вариантов использования

Актёр – это роль, которую пользователи исполняют во время взаимодействия с вариантами использования. Однако в качестве актёров может выступать не только человек, но также аппаратное устройство, время или другая система. Примеры актёров представлены на рисунке ниже.



Рисунок 1.2. Примеры вариантов использования

Кроме самих актёров и вариантов использования на диаграмме используются различные виды связи между объектами диаграммы. Наиболее часто используемая связь, без которой сама диаграмма теряет смысл, это связь между актёром и вариантом использования. Такая связь называется ассоциацией (Association), и она показывает, что актёр может инициировать действие, указанное в варианте использования. Примеры таких ассоциаций показаны на рисунке ниже.

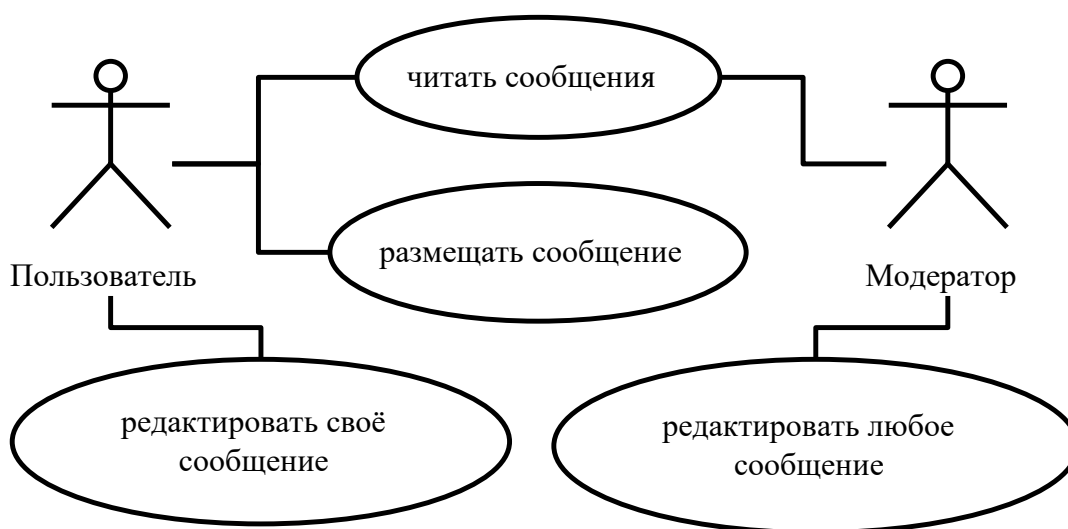


Рисунок 1.3. Пример ассоциации между актёрами и вариантами использования

Далее рассмотрим связи между актёрами. Такой вид связи называется обобщением (Generalization). Обобщение актёров (наследование) показывает, что одна из ролей является частным случаем другой роли. Фактически это обозначает, что все варианты использования, которые может инициировать более общая роль, доступны и для её наследников (но не наоборот). Пример такой связи приведён на рисунке 1.4. При этом некоторые актёры могут быть абстрактными (выделенные курсивом), что значит, что пользователя конкретно с такой ролью существовать не может.

Между вариантами использования также возможны связи, при этом различных типов:

- обобщение (Generalization);
- зависимость (Dependency), которая подразделяется на включение (Include) и расширение (Extend).

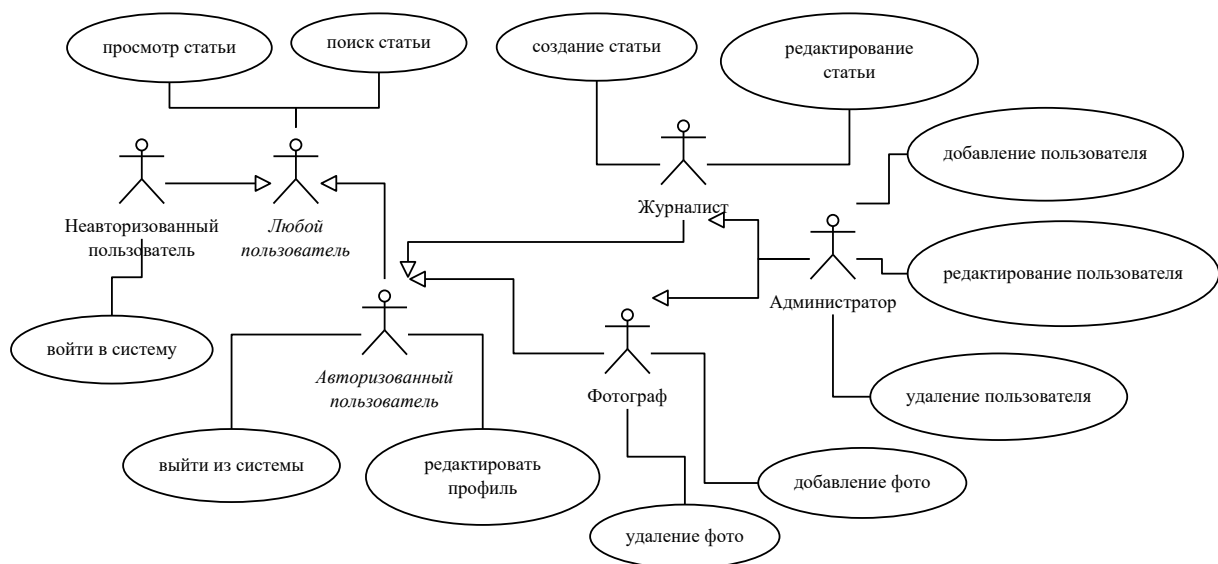


Рисунок 1.4. Пример обобщения между актёрами

Обобщение (наследование) используется, когда вариант использования, являющийся потомком, наследует поведение варианта использования, являющегося предком, дополняя его или заменяя, сохраняя общий интерфейс взаимодействия.

Обобщение между вариантами использования используется в том случае, когда наследник варианта использования может использоваться вместо предка. Пример таких вариантов использования смотри на рисунке ниже:



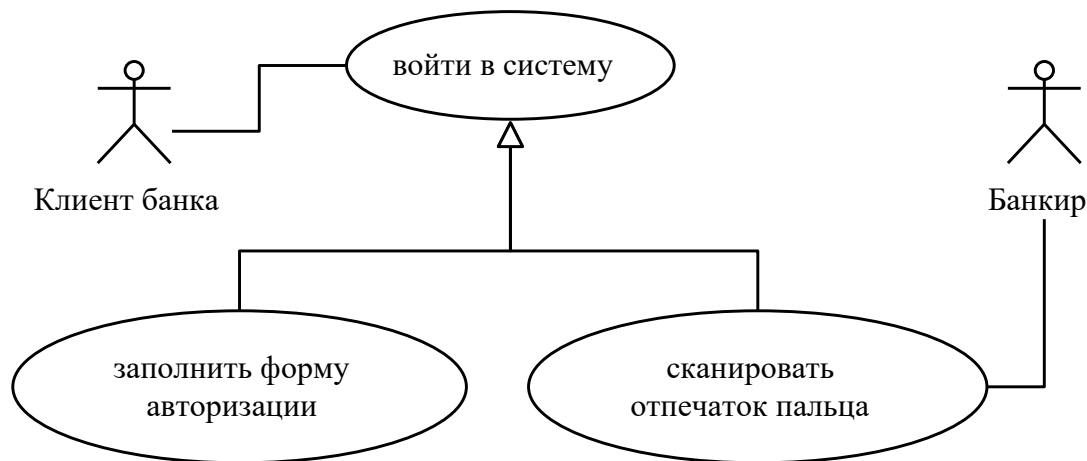


Рисунок 1.5. Пример обобщения между вариантами использования

Зависимости между вариантами использования, в общем случае, просто показывает некую зависимость. Но такая зависимость показывает непосредственную связь между выполняемыми действиями. Например, рассмотрим случай, когда для удаления некой записи, пользователю сначала необходимо открыть форму редактирования (просмотр записи), на которой он может либо исправить поля и нажать кнопку «сохранить» (редактировать запись), либо нажать кнопку «удалить» (удалить запись). В таком случае между вариантами использования «просмотр записи» и «удаление записи» зависимости не будет. То есть последовательность действий на диаграмме вариантов использования с помощью зависимостей не отображается, как, впрочем, и никакой другой связью.

Включение вариантов использования показывает, что в некоторой точке базового варианта использования может использоваться функционал включаемого варианта использования. Пример см. на рисунке. Включаемый вариант использования не может существовать (выполняться) отдельно от базового. То есть на представленном примере ассоциация актёра с вариантом использования «просмотр фото» будет некорректным. Как правило, включаемый вариант использования описывает общую для нескольких других вариантов использования функциональность.

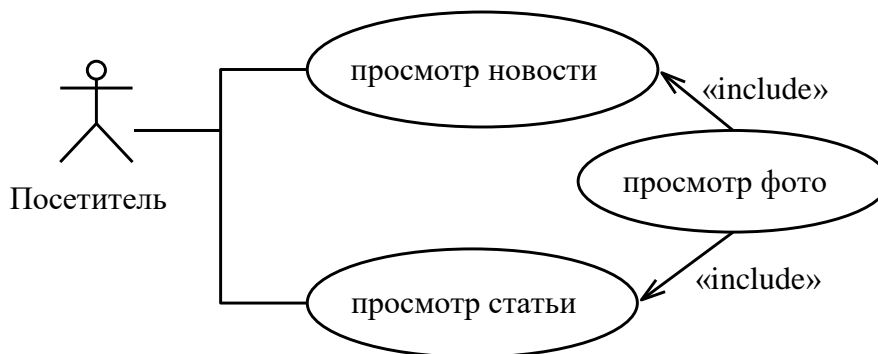


Рисунок 1.6. Пример включения вариантов использования

Расширение вариантов использования подразумевает, что базовый вариант использования неявно содержит в указанной точке функционал расширяющего. Расширяющий вариант использования передаёт своё поведение базовому (возможно при каком-то условии). В отличие от включения, расширяющий вариант использования имеет самостоятельный смысл и может отдельно выполняться актёром, этим он похож на обобщение. Но, в отличие от обобщения, выполнение актёров базового варианта использования не предполагает, что вместо него может выполняться расширяющий вариант использования (только если он явно ассоциирован с актёром).

### 1.3. Диаграмма классов


Диаграмма классов предназначена для описания структуры программного обеспечения информационной системы на уровне классов и взаимосвязей их друг с другом. На диаграмме отображаются сами классы, их атрибуты (поля, переменные класса), их операции (методы, функции класса) и отношения (связи) между классами.



Диаграмма классов для всей информационной системы, содержащая все разрабатываемые классы, будет иметь слишком большой размер, учитывая, что некоторые информационные системы могут содержать до нескольких тысяч классов. Поэтому диаграмму классов могут создавать для решения одной из следующих задач.

- Общее концептуальное моделирование системы (как правило, на уровне интерфейсов и абстрактных классов верхнего уровня иерархии).
- Подробное моделирование для трансляции модели в программный код (как правило, для реализации конкретного требования или модуля).
- Моделирование предметной области.

Диаграмма классов может иметь статический или аналитический вид. Вначале рассмотрим аналитический вид диаграммы классов. В этом случае все классы маркируются одной из иконок, которая показывает назначение создаваемого класса. Возможные виды классов приведены в таблице 1.1.

**Таблица 1.1. Виды классов на диаграмме классов**

Условное обозначение	Описание
 Ю граничные классы	Такие классы обеспечивают взаимодействие с внешними относительно разрабатываемой системы факторами. Например, действия пользователя, времени или внешней системы (всё, что может являться актёром в контексте диаграммы вариантов использования). При этом класс может отвечать как за обработку действия (нажатие на

	<p>кнопку, входящий запрос от внешней системы и т. д.), так и за представление (визуализацию) результата, например отображение формы с компонентом, визуализирующим данные, генерацию HTML-страницы в web-системах и т. д. Фактически, именно такие классы обеспечивают взаимодействие всей информационной системы со внешней средой, поэтому они и называются граничными.</p>
 <p>классы- обработчики</p>	<p>Классы-обработчики – это внутренние классы системы, которые отвечают за реализацию бизнес-логики приложения. К таким классам относятся все классы, отвечающие: за взаимодействие с источником данных; за обработку этих данных в соответствии с алгоритмами, которые необходимо реализовать в информационной системе; за проверку корректности этих данных и т.д. Фактически, все классы, не являющиеся граничными классами или классами-сущностями, являются классами-обработчиками.</p>
 <p>классы- сущности</p>	<p>Классы-сущности – это классы, которые хранят информацию о предметной области. Такие классы не выполняют никакой обработки информации. Хранимая информация содержится в полях класса, а все методы класса лишь предоставляют доступ к этой информации (так называемые методы get-теры и set-теры). При этом методы не должны выполнять даже проверку корректности данных, так как записывать информацию в объект с применением set-теров можно не только при вводе информации пользователем системы, но и при считывании информации из постоянного хранилища. При этом в большинстве информационных систем чтение данных из хранилища (которые уже должны быть корректны), происходит гораздо чаще, чем ввод данных пользователем (которые могут некорректными). Поэтому, с точки зрения производительности, выполнять даже элементарную проверку в классах-сущностях нецелесообразно. Такие классы используются для моделирования предметной области.</p>

Примером диаграммы классов в аналитическом виде может быть диаграмма для некоторого требования, согласно которому в некой банковской системе кассир может перевести деньги с одного банковского счёта на другой (см. рисунок 1.7).

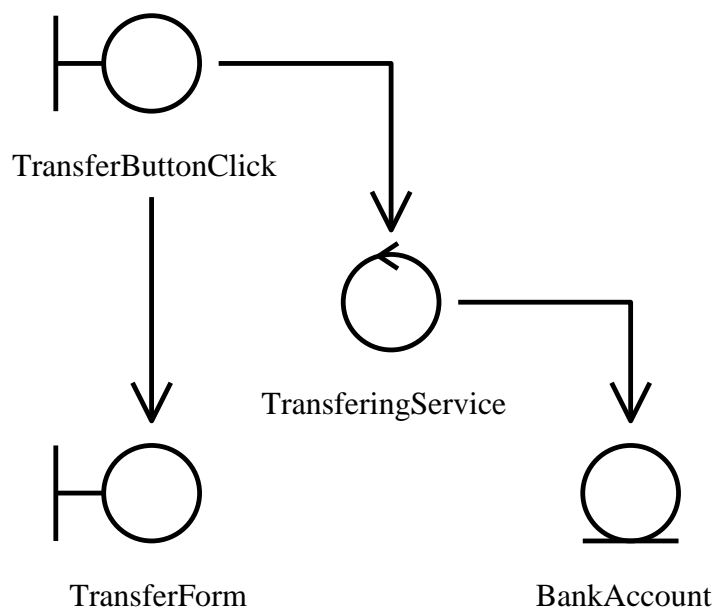


Рисунок 1.7. Аналитический вид диаграммы классов

В данном примере граничный класс `TransferButtonClick` обрабатывает событие нажатия на кнопку «Перевести», считывает с полей формы, которую создаёт граничный класс `TransferForm`, и обращается к классу-обработчику `TransferringService`, который изменяет состояние объектов класса-сущности `BankAccount` и сохраняет их в постоянное хранилище.

Рассмотрим теперь диаграмму классов в статическом виде. Класс на диаграмме в таком виде представляется прямоугольником, разделённым на три секции: имя класса, поля класса, методы класса. В качестве примера рассмотрим класс `BankAccount` (см. рисунок 1.8).

<b>BankAccount</b>
-clientName: String -balance: Integer
+getClientName(): String +setClientName(name:String) +getBalance(): Integer +setBalance(balance:Integer)

Рисунок 1.8. Класс на диаграмме классов статического вида

При отображении класса в статическом виде используются различные соглашения для отображения различных тонкостей, касающихся класса. Так, например, области видимости помечаются значками возле описания поля и метода (знак «+» для `public`-членов, знак «#» для `protected`-членов, знак «-» для `private`-членов). Курсивом набираются имена абстрактных

классов и интерфейсов, а также абстрактных методов. С помощью подчёркивания отображаются статические члены класса.

На диаграмме классов используются различные отношения между классами. Рассмотрим эти отношения.

**Обобщение** (наследование или расширение, Extend) – показывает, что один класс наследуется от другого класса. Отображается сплошной линией, заканчивающейся треугольной стрелкой с белой заливкой.

**Реализация** (Implement) – показывает, что класс реализует некоторый интерфейс. Отображается штриховой линией, заканчивающейся треугольной стрелкой с белой заливкой.

**Ассоциация** (Association) – признаком ассоциации является наличие в классе ссылки на объект другого класса. Отображается сплошной линией с обычной стрелкой.

**Агрегация** (Aggregation) – подвид ассоциации, моделирует связь «часть-целое». Отображается, как и ассоциация, но в начале линии ставится ромб с белой заливкой.

**Композиция** (Composition) – подвид агрегации, имеет привязку к времени жизни объектов. Дочерние объекты не могут существовать после уничтожения контейнера. Отображается, как и агрегация, но ромб в начале линии отображается с чёрной заливкой.

**Зависимость** (Dependency) – связь (всегда направленная), которая возникает в случае, если в классе есть метод, принимающий параметр-ссылку на объект другого класса, или если в классе есть метод, возвращающий ссылку на объект другого класса. Однако может существовать и неявно при создании и использовании объекта внутри некоторого метода класса. Отображается штриховой линией с обычной стрелкой.

## 2. ПРИНЦИПЫ ПРОЕКТИРОВАНИЯ SOLID

В данном разделе предполагается изучение основ объектно-ориентированного проектирования, что в первую очередь предполагает использование объектно-ориентированного программирования как парадигмы, а не как возможности языка программирования. В первом случае речь идет о применении на практике принципов инкапсуляции, наследования, абстракции и полиморфизма. Во втором случае это лишь возможность объявления классов и описания в них полей и методов.

Различие объектно-ориентированного программирования и объектно-ориентированного проектирования заключается в том, что программирование – это инструмент, а проектирование – это метод его применения. То есть объектно-ориентированное программирование (инкапсуляция, наследование, абстракция, полиморфизм) – это лишь способ декомпозиции программного кода на классы (их поля и методы). В то время как объектно-ориентированное проектирование – это принципы, согласно которым программный код разбивается на классы определенным образом (в общем случае, такую декомпозицию можно произвести множеством способов), который обеспечивает:

- удобство командной разработки приложений;
- гибкость расширения функциональных возможностей приложения в будущем;
- легкость изменения приложения в случае изменившихся требований пользователя;
- простоту понимания структуры приложения разработчиками;
- и т.д.

По разным источникам выделяются разное количество принципов объектно-ориентированного проектирования. Тем не менее, существует основные пять принципов, которые встречаются у всех авторов. Остальные принципы дополняют базовые пять принципов применительно к особенностям разработки приложений (существуют специализированные принципы для разработки web-приложений, для разработки настольных приложений и т.д.), к особенностям языков программирования (существуют принципы проектирования специально для Java или Small Talk).

Рассмотрим основные пять принципов.

### 2.1. Принцип единственной ответственности

*Single Responsibility Principle* (принцип единственной ответственности) – принцип, согласно которому должна быть только одна причина изменения класса. Этой причиной может быть только изменение требований,

напрямую касающихся цели, с которой создавался этот класс. Отсюда следует, что при создании класса должна быть только одна цель, которую преследует существование класса.

Более простыми словами у класса должна быть только одна предельно четкая обязанность. Наличие размытого круга обязанностей класса свидетельствует о нарушении данного принципа.

## 2.2. Принцип открытости / закрытости

*Open Closed Principle* (принцип открытости / закрытости) – принцип, согласно которому класс должен быть закрыт для изменения, но открыт для расширения. То есть изменение набора полей и методов класса может производиться только в случае изменения требований, которые касаются этого класса (смотри предыдущий принцип). В иных случаях (например, при расширении функций приложения) поля и методы класса меняться не должны, так как для новых функций должны создаваться новые классы. Такая неизменяемость ранее созданных классов и называется закрытостью для изменений.

Если новый функционал похож на ранее реализованный, то весьма вероятно, что класс для нового функционала будет похож на уже существующий. Часто в таком случае новый класс создается как копия существующего класса с незначительными изменениями, но на практике такой подход приводит к тому, что обе копии должны модифицироваться зачастую одновременно, что усложняет поддержку такого приложения. В таком случае одним из способов избавления от дублирующегося кода является применение наследования, т.е. новый класс наследуется от существующего, добавляя нужный функционал.

Проектирование класса таким образом, чтобы от него можно было наследоваться при добавлении нового функционала, и называется открытостью класса для расширения.

## 2.3. Принцип подстановки Лисков

*Liskov Substitution Principle* (принцип подстановки Лисков) – принцип, сформулированный Барбарой Лисков, гласит, что экземпляр подкласса может использоваться всюду, где может быть использован экземпляр суперкласса.

На самом деле, это утверждение – одна из особенностей полиморфизма, то есть работать должно и так всегда. Однако на практике часто приходится сталкиваться с ситуациями, когда спроектированный класс используется в некотором методе только за счет того, что в этом методе известны некоторые особенности реализации используемого класса. Но когда созда-

ется новый подкласс данного класса, в котором реализация изменяется и подкласс теряет указанную особенность, то формально (согласно принципу полиморфизма) экземпляр этого подкласса можно передать в метод вместо экземпляра суперкласса, но работать корректно этот метод уже не сможет.

Такое поведение и называют нарушением принципа подстановки Лисков. Таким образом, можно рекомендовать всегда при проектировании класса предполагать, как будет работать приложение, если появиться новый подкласс спроектированного класса.

## 2.4. Принцип инверсии зависимости

*Dependency Inversion Principle* (принцип инверсии зависимости) – класс не должен зависеть от другого класса напрямую, он должен зависеть от него через интерфейс.

Рассмотрим пример: в приложении необходимо считать из файла список некоторых записей по некоторому критерию (например, все книги одного автора). Далее этот список сортируется (например, по названию книги) и в нем производится двоичный поиск по запросу пользователя. Для удобства реализации этих операций создается отдельный класс для хранения списка в виде массива. Тогда классы: читающий список из файла; отсеивающий записи по нужному критерию; сортирующий список; выполняющий поиск в списке – все эти классы зависят от этого списка. В случае, если для увеличения производительности встанет задача использования списка на основе компонентов связности, придется все классы, зависящие от массива, незначительно модифицировать. Такая ситуация, особенно в крупных проектах, может привести к большим временным затратам при модификации или сопровождении приложения.

Наличие упомянутых прямых зависимостей и является нарушением принципа инверсии зависимости. Для соблюдения данного принципа в упомянутом примере необходимо, чтобы классы чтения, фильтрации, сортировки и поиска зависели не от списка в виде массива напрямую, а от некоторого общего для всех списков интерфейса. В таком интерфейсе могут быть объявлены методы добавления и доступа к элементам списка. В классах, зависящих от списков, может храниться ссылка интерфейсного типа, который может принимать значения ссылки на экземпляр любого класса, реализующего этот интерфейс. Тогда все изменения для перевода приложения с использования одного вида списка на другой, будут касаться только создания экземпляра класса списка, все остальные действия, которые выполняются через интерфейс, изменены не будут.



## 2.5. Принцип изоляции интерфейса

*Interface Segregation Principle* (принцип изоляции интерфейса) – принцип гласит, что класс не должен зависеть от тех методов интерфейса, которые ему не нужно реализовывать. Данный принцип используется при проектировании интерфейсов (или абстрактных классов).

Если в интерфейсе объявлено несколько методов, то любой класс, реализующий этот интерфейс, должен реализовать все эти методы, хотя не всегда такая возможность может представляться.

Например, если говорить о тех же списках, разработчик может применить в приложении список на основе массивов, однако, предполагая, что в будущем может понадобиться изменить этот список на двунаправленный список на основе компонентов связности, можно заранее объявить интерфейс, в котором будут присутствовать методы итерирования списка в прямом и обратном направлении. Но если в будущем понадобится создать класс для однонаправленного списка на основе компонентов связности, реализовать такой интерфейс будет затруднительно, так как итерирование однонаправленного списка в обратном направлении реализовать очень сложно, да и работать такой перебор будет чрезвычайно медленно. А это значит, что даже если эти методы и будут реализованы (с применением больших трудовых и временных затрат), то все равно их будут стараться не использовать.

С другой стороны, можно «ненужные» методы не реализовывать (то есть оставлять некоторую пустую их реализацию, например, с генерацией некоторой исключительной ситуации). Но такой подход тоже является не совсем корректным, так как лишает приложение определенной гибкости.

Создание «раздутых» интерфейсов является нарушением принципа изоляции интерфейса. В таких случаях рекомендуют разбивать интерфейс на несколько интерфейсов. Для приведенного примера один интерфейс может содержать методы прямого итерирования, второй – методы обратного итерирования.

## 2.6. Задания на лабораторную работу №1

Для приложения своего варианта постройте диаграмму классов, указав на ней все поля и методы и их области видимости, а также связи между классами.

На основе построенной диаграммы классов реализуйте приложение на языке программирования Java.

### **Варианты:**

1. Приложение, которое читает данные из файла о координатах и размерах различных графических примитивов (отрезок, прямоугольник, эллипс, полигон и т.д.), их цвете (контур и заливки) и номере слоя. Далее

приложение должно строить изображение, которое состоит из этих примитивов. При этом следует помнить, что хранение информации о графическом объекте и его рисование на форме – две отдельные задачи. Для разделения задач хранения необходимой информации и рисование примитивов можно использовать механизм обобщения (Generic).

Структура приложения должна соответствовать принципу единственной ответственности.

*Примечание:*

Для изображений, состоящих из нескольких графических объектов, которые могут перекрываться друг с другом, используется понятие слоя. Слой содержит один объект – это аналог прозрачной кальки, на которой нарисован этот объект. Такие кальки накладываются друг на друга. При этом слои нумеруются, минимальный номер слоя соответствует либо нижнему слою, либо верхнему, по выбору разработчика. При выводе изображения на экран сначала рисуется объект, находящийся на самом нижнем слое, затем поверх него – объект на следующем слое, и т.д. до последнего слоя.

2. Приложение, которое для одной из трех функций (по выбору пользователя):

$$\begin{aligned} & \_ f_1(x) = e^x \cos x, \\ & \_ f_2(x) = (x^2 + 3x - 4)^{-1}, \\ & \_ f_3(x) = \ln \sqrt{x}, \end{aligned}$$

вычисляет определенный интеграл на выбранном пользователем отрезке, при этом вычисление должно осуществляться либо по формуле левых прямоугольников, либо по формуле правых прямоугольников, либо по формуле трапеций (также по выбору пользователя). При этом вычисления необходимо производить с точностью до четырех знаков после запятой. Точность контролировать пересчетом с увеличением вдвое количества частей разбиения отрезка интегрирования.

Структура приложения должна соответствовать принципу открытости/закрытости.

*Примечание:*

И для формул прямоугольников, и для формул трапеций отрезок интегрирования разбивается на  $n$  частей (количество частей разбиения выбирается произвольным образом, например равным 10), далее рассчитываются частичные суммы:

$$L_n = \sum_{i=0}^{n-1} f(x_{i+1}) \Delta x$$

– формула левых прямоугольников,

$$P_n = \sum_{i=0}^{n-1} f(x_i) \Delta x$$

– формула правых прямоугольников,

$$T_n = \sum_{i=0}^{n-1} \frac{f(x_i) + f(x_{i+1})}{2} \Delta x$$

– формула трапеций,

где  $f(x)$  – интегрируемая функция,

$\Delta x = \frac{b-a}{n}$ ,  $x_0 = a$ ,  $x_n = b$ ,  $[a, b]$  – отрезок интегрирования.

Далее вычисляются суммы  $L_{2n}$ ,  $P_{2n}$  или  $T_{2n}$ , затем  $L_{4n}$ ,  $P_{4n}$  или  $T_{4n}$ , и так далее, пока модуль разности сумм при двух последних способах разбиения меньше 0,0001. Как видно, эти формулы имеют много общего, что приводит к одинаковым циклам для каждой формулы. Необходимо разработать такую структуру приложения, которая позволит избежать дублирования кода по организации таких циклов.

3. Консольное приложение, которое считывает из файла (или файлов, по решению разработчика) список платежей разных типов некоторой организации:

- по счетам юридическим лицам (назначение платежа, сумма платежа, название банка, номер счета, название юридического лица, номер государственной регистрации, фамилия руководителя, имя руководителя, отчество руководителя);

- по счетам физическим лицам (назначение платежа, сумма платежа, название банка, номер счета, фамилия, имя, отчество, серия паспорта, номер паспорта);

- наличными физическим лицам (назначение платежа, сумма платежа, номер чека, фамилия, имя, отчество, серия паспорта, номер паспорта).

Затем выводит в виде таблицы платежи определенного типа (при этом набор столбцов таблицы должен зависеть от типа платежа). Выводимый список платежей должен сортироваться по выбранному пользователем столбцу. Тип выводимых платежей так же определяется пользователем.

Структура приложения должна соответствовать принципу подстановки Лисков.

*Примечание:*

Для построения таблицы в консоли можно использовать псевдографику (специальные символы для имитации границ таблицы). Данный функционал реализуется достаточно сложно, поэтому алгоритм визуализации необходимо отделить от хранения данных. Это можно сделать, реализовав

в каждом классе, хранящем необходимые данные, возможность перебора всех его полей, например, используя карты отображений. В качестве ключей карты можно хранить название столбца, в качестве значений – значение, возвращаемое соответствующим методом, преобразованное в строку. Кстати, в один столбец можно свести данные нескольких полей, например, три поля: фамилия, имя, отчество – можно свести в один столбец с фамилией и инициалами.

4. Приложение, позволяющее пользователю вводить две последовательности чисел, одна из которых размещается в бинарное дерево, вторая в стек. Далее пользователю выводится содержимое дерева и стека, после чего из сформированного стека (из вершины) по одному извлекаются элементы, и те из них, которые присутствуют в дереве, добавляются в новый стек. После чего пользователю выводится содержимое нового стека. Дерево и стек должны быть реализованы самостоятельно, ввод и вывод элементов в/из дерева/стека должны осуществляться единообразно.

Структура приложения должна соответствовать принципу изоляции интерфейсов.

*Примечание:*

Для того чтобы ввод или вывод значений из стека и дерева осуществлялся единообразно, необходимо реализовать единообразие перебора элементов стека и дерева и единообразие добавление элемента в стек и дерево. Сама реализация стека и дерева может быть любой (по выбору разработчика), однако следует реализовать такую структуру в приложении, чтобы минимальными изменениями можно было добавить новую реализацию стека или дерева.

5. Приложение, управляющее балансом абонентов мобильной телефонной сети. В отдельном файле должна храниться информация об абонентах: телефонный номер абонента, баланс счета, тарифный план. В отдельном файле находятся данные о совершенных абонентом звонках, отсланных SMS-сообщениях и установленных соединениях с сетью Интернет. Необходимо считать данные о совершенных операциях и рассчитать новый баланс абонента. Все номера записываются в формате XXXXX-YY-ZZZZZZ, где XXXXX – код страны, YY – код оператора мобильной связи или код города в стационарной телефонной сети, ZZZZZZ – сам телефонный номер абонента. Все телефонные звонки на номера, у которых первые 5 цифр отличны от 00375 тарифицируются как роуминг (считаем, что звонки доступны в любую страну по одинаковым тарифам). Все звонки на номера, начинающиеся с 00375-55, тарифицируются как внутрисетевые. Все звонки на номера, начинающиеся с 00375-25, 00375-29, 00375-33, 00375-44, тарифицируются как звонки в другие мобильные сети. Остальные номера тарифицируются как звонки на стационарную сеть. В приложении реализовать поддержку следующих тарифных планов (все тарифные планы не имеют абонентской платы):

– «план А», звонки на номера из этой же сети 25 руб./мин., звонки другим мобильным операторам 125 руб./мин., звонки на стационарную сеть 95 руб./мин., роуминг 2500 руб./мин., тарификация по 10 сек., SMS 150 руб., SMS в роуминге 1500 руб., Интернет 3000 руб./час.

– «план Б», звонки на номера из этой же сети 5 руб./мин. (первая минута) и 50 руб./мин. (последующие), звонки другим мобильным операторам 75 руб./мин. (первая минута) и 500 руб./мин. (последующие), звонки на стационарную сеть 50 руб./мин. (первая минута) и 250 руб./мин. (последующие), роуминг 5000 руб./мин., тарификация по 10 сек., тарификация при роуминге по 1 мин., SMS 150 руб., SMS в роуминге 1000 руб., Интернет 1000 руб./Мб за первых 50 Мб и 1250 руб./Мб за все последующие.

– «план В», звонки на номера из этой же сети 100 руб./мин. (первая минута) и 5 руб./мин. (последующие), звонки другим мобильным операторам 100 руб./мин. (первая минута) и 25 руб./мин. (последующие), звонки на стационарную сеть 100 руб./мин. (первая минута) и 25 руб./мин. (последующие), роуминг 1000 руб./мин., тарификация по 1 мин., SMS 150 руб., SMS в роуминге 150 руб., Интернет 1000 руб./Мб за первых 50 Мб и 1250 руб./Мб за все последующие.

– «план Г», звонки на номера из этой же сети 500 руб./мин. (первая минута) и 100 руб./мин. (последующие), звонки другим мобильным операторам 500 руб./мин. (первая минута) и 100 руб./мин. (последующие), звонки на стационарную сеть 500 руб./мин. (первая минута) и 100 руб./мин. (последующие), роуминг 500 руб./мин., тарификация по 10 сек., SMS 150 руб., SMS в роуминге 5000 руб., Интернет 500 руб./Мб за первых 100 Мб, 1000 руб./Мб при трафике от 100 Мб до 500 Мб, 5000 руб./Мб за все последующие.

Структура приложения должна соответствовать принципу инверсии зависимостей.

*Примечание:*

В файле со звонками, SMS и подключению к сети Интернет должны быть только сведения о том, кто и куда позвонил (отправил SMS), дата и время начала звонка (подключения к сети Интернет), дата и время окончания звонка (подключения к сети Интернет), объем переданных данных. Тарификация (учет стоимости операции) выполняет приложение в зависимости от тарифного плана вызывающего абонента. При этом структура приложения должна позволять легко добавлять новый тарифный план. Также должна быть возможность относительно легко добавить новую услугу (отправка MMS, выполнение платного USSD запроса и т.д.).

б. Приложение, читающее из входного файла в формате CSV, данные о продажах товаров (название товара, стоимость одной единицы; количество проданных штук), и записывающее в выходной файл (также в форма-

те CSV) данные о выручке по каждому товару (название товара; объем выручки), отсортированные по убыванию выручки.

Структура приложения должна соответствовать принципу единственной ответственности.

*Примечание:*

Формат CSV (Comma Separated Values) – формат текстовых файлов для хранения таблиц. Каждая строка файла (последовательность символов, разделенная символами перехода строки и возврата каретки) соответствует одной строке таблицы. Ячейки в пределах одной строки таблицы разделяются символом ‘;’. Если ячейка содержит символ ‘;’, то вся строка берется в двойные кавычки, если же внутри ячейки, заключенной в кавычки, содержится символ двойной кавычки, то он удваивается. Но следует помнить, что построчное чтение и запись файла, а также обработка одной ее строки – разные задачи. Кроме того, при некорректном формате входного файла необходимо проинформировать пользователя о номере строки и столбца, в котором содержатся некорректные данные.

7. Приложение, вычисляющее корни системы линейных алгебраических уравнений методом Гаусса или Жордана-Гаусса (по выбору пользователя) с выбором главного элемента по матрице.

Структура приложения должна соответствовать принципу открытости/закрытости.

*Примечание:*

По методу Гаусса матрица системы приводится к верхнетреугольному виду, затем последовательно выражаются все неизвестные. По методу Жордано-Гаусса матрица системы приводится к единичному виду, при этом в столбце свободных членов остается столбец решений. Но для получения нулевых значений в нужных элементах матрицы, применяются преобразования (умножение строки на ненулевое число, сложение строк), с помощью которых, используя элемент на главной диагонали, получаются нули в том же столбце. Однако, если диагональный элемент равен нулю, то выполнить такие преобразования становится невозможным. И даже если элемент на главной диагонали не равен нулю, но очень близок к нему, из-за ограниченности разрядной сетки такая ситуация приводит к значительной вычислительной погрешности. Для минимизации этой погрешности используют выбор максимального по модулю элемента в той части матрицы, которая правее и ниже элемента на главной диагонали, а затем обменом строк и столбцов этот элемент ставится на место диагонального. При этом обмен строк при решении систем линейных алгебраических уравнений не влияет на результат, а обмен столбцов приводит к изменению порядка значений в векторе решений. Поэтому при обмене столбцов необходимо запоминать индексы меняемых местами строк, чтобы затем в обратном порядке поменять местами значения в векторе решений. Поскольку

идея выбора главного элемента по матрице одинакова для обоих методов решения системы, реализовывать этот выбор необходимо только один раз.

8. Приложение – матричный калькулятор. Приложение должно позволять работать с квадратными матрицами размера  $m \times m$  (при старте приложения пользователь выбирает размер  $m$ ). Калькулятор должен поддерживать четыре операции: сложение, вычитание, умножение, деление ( $A/B = A \cdot B^{-1}$ , где  $B^{-1}$  – обратная матрица). При вводе пользователем матрицы-операнда, он выбирает тип матрицы (диагональная; верхне- или нижнетреугольная; симметричная; обычная; константная – все элементы которой равны одному и тому же значению; константная диагональная – все диагональные элементы которой равны одному и тому же значению, а остальные – нулевые). При этом хранение элементов матриц необходимо оптимизировать по объему используемой памяти.

Структура приложения должна соответствовать принципу подстановки Лисков.

9. Приложение, осуществляющее выполнение различных операций со счетами клиентов в некотором банке. В некотором файле хранится таблица с информацией о клиентских счетах: номер счета, баланс счета, тип клиента (физическое или юридическое лицо). Также в некотором файле хранится баланс счета самого банка, на который поступает комиссия с проводимых операций. Еще в одном файле хранится баланс счета налоговой инспекции, на который поступает налог государству с совершаемых операций. Доступно несколько способов перечисления денег с одного счета на другой:

- перевод между двумя физическими лицами средства в объеме до 1 млн. обычным переводом без налога и комиссии;
- перевод между двумя физическими лицами средства в объеме до 10 млн. перевод с комиссией банку 2.5% от суммы перевода;
- перевод между двумя физическими лицами средства в объеме свыше 10 млн. перевод с комиссией банку 5% и налогом государству 15% от суммы перевода;
- перевод от физического лица юридическому лицу с комиссией банку 10% от суммы перевода;
- перевод от юридического лица физическому лицу с комиссией банку 2% и налогом государству 5% от суммы перевода;
- перевод между двумя юридическими лицами с налогом государству 20% от суммы перевода.

Пользователь приложения (банковский работник) выбирает номер счета отправителя платежа и получателя платежа. Приложение, в зависимости от типа счетов (физическое или юридическое лицо) и (если необходимо) от суммы платежа, выбирает нужный способ перечисления платежа, выводит информацию о проводимом платеже и выдает пользователю запрос на подтверждение действия. Если пользователь подтверждает опера-

цию, в файл с состоянием счетов записываются соответствующие изменения, в файлы с состоянием счетов самого банка и налоговой инспекции также вносятся при необходимости соответствующие изменения, в файл истории платежей вносится информация о проведенном платеже.

Информация о проводимом платеже содержит: номера счетов отправителя и получателя, сумма платежа, комиссия банка, налог государству, дата и время операции.

Структура приложения должна соответствовать принципу изоляции интерфейсов.

*Примечание:*

Для каждого типа платежа, подразумевающего банковскую комиссию, комиссия должна списываться единообразно. Также и для каждого типа платежа, подразумевающего государственный налог, он должен списываться единообразно. При списывании комиссии или налога не нужно знать тип проведенного платежа, нужно уметь получить сумму комиссии или налога, и сделать соответствующие изменения в нужном файле.

10. Приложение — эмулятор простейшей вычислительной системы. Приложение должно считывать программу из текстового файла и выполнять её по шагам. Программа должна иметь следующий синтаксис.

Каждая команда программы располагается на отдельной строке файла. Формат команды: КОП <операнды>. Здесь КОП - название команды, состоящее из английских букв. В зависимости от типа команды у неё может быть один или два операнда, отделённые от названия команды пробелом. Два операнда разделяются запятой. Операнды могут быть именем переменной (строка, состоящая из английских букв) или числом (вещественное число с точкой в качестве разделителя целой и дробной части). В эмуляторе должны быть реализованы следующие команды:

**MOV операнд1, операнд2**

копирование содержимого операнд2 в операнд1, операнд1 может быть только названием переменной, при этом, если переменная не существовала, она создаётся, иначе изменяется значение существующей переменной;

**ADD операнд1, операнд2**

сложение операндов с присвоением результата в операнд1, который может быть только названием переменной;

**SUB операнд1, операнд2**

вычитание из операнд1 значения операнд2 с присвоением результата в операнд1, который может быть только названием переменной;

**MUL операнд1, операнд2**

умножение операндов с присвоением результата в операнд1, который может быть только названием переменной;

**DIV операнд1, операнд2**

деление операнд1 на операнд2 с присвоением результата в операнд1, который может быть только названием переменной;



**IN операнд**

считывает с клавиатуры число и заносит его в операнд, который должен быть названием переменной;

**OUT операнд**

выводит значение операнд в консоль;

**CMP операнд1, операнд2**

сравнивает два операнда и запоминает результат в специальную переменную, результатом может быть -1, если операнд1 меньше операнд2; 0, если операнды равны; 1, если операнд1 больше операнд2;

**JMP операнд**

переходит к выполнению команды в строке с номером, равным операнду;

**JE операнд**

переходит к выполнению команды в строке с номером, равным операнду, если результат сравнения равен 0;

**JNE операнд**

переходит к выполнению команды в строке с номером, равным операнду, если результат сравнения не равен 0;

**JG операнд**

переходит к выполнению команды в строке с номером, равным операнду, если результат сравнения равен 1;

**JNG операнд**

переходит к выполнению команды в строке с номером, равным операнду, если результат сравнения не равен 1;

**JL операнд**

переходит к выполнению команды в строке с номером, равным операнду, если результат сравнения равен -1;

**JNL операнд**

переходит к выполнению команды в строке с номером, равным операнду, если результат сравнения не равен -1;

Приложение должно обрабатывать команды, в случае же, если команду выполнить невозможно, выполнение программы должно прерываться с выводом сообщения об ошибке.

Добавление новых команд в приложение должно происходить с минимальными изменениями в существующих классах.

Структура приложения должна удовлетворять принципу инверсии

### 3. ШАБЛОНЫ ПРОЕКТИРОВАНИЯ GRASP

Шаблоны распределения ответственности, или General Responsibility Assignment Software Patterns (GRASP) – это набор рекомендаций по декомпозиции системы на отдельные классы с чётким разграничением обязанностей между классами. Шаблоны GRASP тесно переплетаются с принципами SOLID, дополняя и уточняя их.

Рассмотрим пять основных шаблонов.

Шаблон **«информационный эксперт»** принято первым применять при проектировании сложных систем. Основная цель этого шаблона выделить классы, которые будут хранить информацию, и классы, которые будут ее обрабатывать. Под обработкой понимается не только работа алгоритмов, реализующих основную логику приложения, но и взаимодействие (добавление, чтение, изменение, удаление данных) с системой хранения данных, а также визуализация данных, проверка их корректности и т.д. Таким образом, согласно данному шаблону необходимо выделить классы, являющиеся информационными экспертами. Поля таких классов хранят нужную информацию, а методы лишь предоставляют к ней доступ (зачастую это просто методы чтения и записи значений данных полей). После того, как информационные эксперты выделены, эти классы не реализуют никакой логики обработки хранимой информации (даже проверку на корректность данных). Для реализации логики используются другие классы, которые будут обрабатывать данные, хранимые информационными экспертами. По большому счету, все остальные шаблоны регламентируют распределение ответственности между этими оставшимися классами.

Шаблон **«сильное зацепление»** используется при разделении обязанностей между всеми классами и регламентирует количество задач, возлагаемых на каждый класс. Данный шаблон практически полностью повторяет принцип единственной ответственности. Единственное различие принципа единственной ответственности и шаблона проектирования «сильное зацепление» в том, что принцип единственной ответственности формулируется исходя из цели проектирования (обеспечение гибкой расширяемости приложения), а шаблон проектирования «сильное зацепление» способ достижения этой цели – максимальная конкретизация поставленной перед классом задачи. Один из признаков нарушения этого шаблона – название класса, не вполне адекватно отражающее то, что данный класс делает.

Шаблон **«слабая связанность»** также используется при разделении обязанностей между всеми классами, но регламентирует он количество связей класса с другими классами. Согласно данному шаблону проектирования при проектировании классов нужно стараться минимизировать его связи с другими классами. Практически, этот шаблон декларирует

еще один способ соблюдения принципа единственной ответственности. Также этот шаблон связан с шаблоном «сильное зацепление». Действительно, если класс имеет четко обозначенный круг обязанностей (то есть «сильно зацеплен» за свои обязанности), то для решения этих задач ему достаточно минимального количества связей с другими классами. Явными признаками связи между класса являются:

- наследование;
- реализация интерфейса;
- наличие в классе поля-ссылки на другой класс;
- наличие в классе метода, принимающего в качестве параметра ссылку на другой класс;
- наличие в классе метода, возвращающего в качестве результата ссылку на другой класс;
- наличие в методе класса локальной переменной-ссылки на другой класс.

Для ослабления некоторых зависимостей может использоваться другой шаблон – «создатель». Следует, однако, заметить, что зависимости между информационными экспертами и другими классами, как правило, не учитываются. Это происходит от того, что информационные эксперты используются для транспорта информации от одних классов, реализующих обработку данных, другим. Точнее было бы сказать, что не учитывается степень связанности информационного эксперта с другими классами, но для этих классов связи с информационными экспертами могут учитываться. Хотя в такой ситуации проще использовать шаблон «сильное зацепление». А вот степень связанности нескольких информационных экспертов – важный показатель, так как от этого зависит простота расширения предметной области. Но зачастую основным фактором при моделировании предметной области все же является не шаблоны «сильное зацепление» или «слабая связанность», а адекватность создаваемой модели.

Шаблон «создатель» и «контроллер» используется при распределении определенного рода обязанностей.

Шаблон «создатель», в частности, – это класс, который берет на себя ответственность за создание (инстанцирование) экземпляра некоторого класса (как правило не информационного эксперта). Чаще всего методы класса-создателя возвращают ссылку на созданный объект, инкапсулируя процесс создания и инициализации этого объекта. В общем случае классом-создателем можно назвать любой класс, который неким образом создает любой объект. Так, например, класс, который читает некоторую информацию из файла, а затем создает на основе этой информации объект информационного эксперта (или список таких объектов) и возвращает ссылку на этот объект, уже можно назвать создателем.

Шаблон «контроллер» – это класс, который берет на себя ответственность взаимодействия с внешними по отношению к системе факто-

рами. Чаще всего это взаимодействие с пользователем, но также это может быть обработка некоторых событий, возникающих в определенное время, или обработка сигналов, приходящих с различных датчиков. Мы будем рассматривать только контроллеры, обрабатывающие пользовательские запросы. Такие контроллеры условно можно разделить на две группы:

- единая точка входа (Front Controller);
- контроллер прецедента (Page Controller).

К первой группе относятся контроллеры, которые обрабатывают абсолютно все запросы пользователя, извлекают из такого запроса все необходимые данные, принимают решения, кому перенаправить этот запрос для обработки, после чего либо обработка запроса заканчивается (то есть часть системы, которой передали запрос на обработку, выполнила все необходимые действия и сама сообщила пользователю о результатах), либо контроллер может принимать результат обработки запроса и сообщать его пользователю. Такие контроллеры удобно использовать при создании сетевых серверов, получающих запросы от клиента по некоторому прикладному протоколу (HTTP, FTP и т.д.); при создании настольных приложений с использованием непосредственно Windows API (при наличии единой оконной функции для приложения); при разработке консольных приложений, работающих в командном режиме (когда пользователь сначала полностью набирает текст команды, которая содержит всю необходимую информацию, а затем ее обрабатывает, как, например, консольный клиент для MS SQL Server).

Ко второй группе относятся контроллеры, которые обрабатывают специализированные запросы (например, запрос на создание новой записи в базе данных обрабатывает один контроллер, а запрос на удаление записи – другой). Такие контроллеры удобно использовать при разработке web-приложений, работающих под управлением стандартного web-сервера (Apache, Small HTTP Server, и т.д.). В таком случае общую обработку запроса осуществляет сам web-сервер, а затем вызывается определенная часть нашего приложения (например, определенный PHP-скрипт). Также такие контроллеры удобно использовать в настольных приложениях с использованием высокоуровневых библиотек (VCL, MFC или QT для C++; AWT, Swing или SWT для Java и т.д.). В таком случае один обработчик некоторого события (нажатия на кнопку, выбора элемента из списка и т.д.) является отдельным контроллером. Еще подобные контроллеры используются при разработке консольных приложений, в которых пользователь сначала выбирает некоторое общее действие в меню, а затем в диалоговом режиме приложение уточняет нужные данные. Тогда каждому действию в меню ставится в соответствие свой контроллер, который затем начинает свой диалог с пользователем.

### 3.1. Задания на лабораторную работу №2

Разработайте приложение-игру пасьянс, позволяющее раскладывать один из карточных пасьянсов (см. свой вариант). Приложение должно выполнять первоначальный расклад, давать пользователю возможность совершать ход (проверяя, является ли он корректным), проверять, есть ли ещё возможные ходы для пользователя, а также выдавать сообщение о том, что пасьянс сошёлся или не сошёлся. При проектировании структуры приложения используйте следующие шаблоны распределения ответственности:

- информационный эксперт;
- создатель;
- контроллер;
- слабая связанность;
- сильное зацепление.

Процесс проектирования проиллюстрируйте UML-диаграммами классов и состояний.

#### ***Варианты:***

1. Баян.
2. Дважды два.
3. Карлтон.
4. Косынка.
5. Монте-Карло.
6. Пары.
7. Пирамида.
8. Перекресток.
9. Солитер.
10. Тузы.

## 4. ПОРОЖДАЮЩИЕ ШАБЛОНЫ ПРОЕКТИРОВАНИЯ GOF

### 4.1. Задание на лабораторную работу №3

Рассмотрим исходный приложения, которое считывает из XML-документа данные о днях рождения некоторых людей и выводит их в виде таблицы на форме, позволяя отсортировать по различным критериям.

Содержимое файла Person.java:

```
import java.util.Date;

public class Person {
    private String firstName;
    private String middleName;
    private String lastName;
    private Date birthday;

    public String getFirstName() { return firstName; }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getMiddleName() { return middleName; }

    public void setMiddleName(String middleName) {
        this.middleName = middleName;
    }

    public String getLastName() { return lastName; }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public Date getBirthday() { return birthday; }

    public void setBirthday(Date birthday) {
        this.birthday = birthday;
    }
}
```

Содержимое файла PersonXmlReader.java:

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.text.ParseException;
import java.text.SimpleDateFormat;
```

```

import java.util.ArrayList;
import java.util.List;

import javax.xml.stream.XMLInputFactory;
import javax.xml.stream.XMLStreamException;
import javax.xml.stream.XMLStreamReader;

public class PersonXmlReader {
    public static List<Person> readFromFile(String fileName) {
        List<Person> persons = new ArrayList<>();
        SimpleDateFormat format =
            new SimpleDateFormat("yyyy-MM-dd");

        try {
            InputStream stream = new FileInputStream(fileName);
            XMLInputFactory factory = XMLInputFactory.newInstance();
            XMLStreamReader reader =
                factory.createXMLStreamReader(stream);

            Person person = null;
            int elementType;
            String tagName;
            while(reader.hasNext()) {
                elementType = reader.next();
                switch(elementType) {
                    case XMLStreamReader.START_ELEMENT: {
                        tagName = reader.getLocalName();
                        switch(tagName) {
                            case "person":
                                person = new Person();
                                break;
                            case "first-name":
                                person.setFirstName(
                                    reader.getElementText()
                                );
                                break;
                            case "middle-name":
                                person.setMiddleName(
                                    reader.getElementText()
                                );
                                break;
                            case "last-name":
                                person.setLastName(
                                    reader.getElementText()
                                );
                                break;
                            case "birthday":
                                String birthday =
                                    reader.getElementText();
                                person.setBirthday(
                                    format.parse(birthday)
                                );
                                break;
                        }
                    }
                }
            }
        } catch (XMLStreamException e) {
            e.printStackTrace();
        }
    }
}

```

```

        );
        break;
    }
    break;
}
case XMLStreamReader.END_ELEMENT: {
    tagName = reader.getLocalName();
    switch(tagName) {
        case "person":
            persons.add(person);
            break;
    }
    break;
}
}
}
}
reader.close();
} catch(FileNotFoundException | XMLStreamException |
        ParseException ignored) {}
return persons;
}
}
}

```

Содержимое файла Runner.java:

```

import java.awt.BorderLayout;
import java.util.List;

import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.UIManager;
import javax.swing.UnsupportedLookAndFeelException;

public class Runner {
    public static void main(String[] args) {
        try {
            UIManager.setLookAndFeel(
                UIManager.getSystemLookAndFeelClassName()
            );
        } catch (ClassNotFoundException
            | InstantiationException
            | IllegalAccessException
            | UnsupportedLookAndFeelException e) {}
        JFrame window = new JFrame("Главное окно");
        window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        window.setSize(500, 400);
    }
}

```



```

List<Person> persons =
    PersonXmlReader.readFromFile("persons.xml");
PersonTableModel model = new PersonTableModel(persons);
JTable table = new JTable(model);
JScrollPane scrollPane = new JScrollPane(table);
window.add(scrollPane, BorderLayout.CENTER);
JPanel chooser = new JPanel(new BorderLayout());
JComboBox<String> comboBox = new JComboBox<>();
comboBox.addItem("по дате");
comboBox.addItem("по дню недели");
comboBox.addItem("по месяцу");
chooser.add(comboBox, BorderLayout.CENTER);
JButton sortButton = new JButton("сортировать");
sortButton.addActionListener(
    new SortButtonListener(comboBox, model)
);
chooser.add(sortButton, BorderLayout.EAST);
window.add(chooser, BorderLayout.NORTH);
window.setVisible(true);
window.setLocationRelativeTo(null);
}
}

```

Содержимое файла PersonTableModel.java:

```

import java.text.DateFormat;
import java.util.ArrayList;
import java.util.List;

import javax.swing.event.TableModelEvent;
import javax.swing.event.TableModelListener;
import javax.swing.table.TableModel;

public class PersonTableModel implements TableModel {
    private final List<Person> persons;
    private final DateFormat format;

    public PersonTableModel(List<Person> persons) {
        this.persons = persons;
        format = DateFormat.getDateInstance(DateFormat.MEDIUM);
    }

    @Override
    public int getColumnCount() { return 4; }

    @Override
    public String getColumnName(int i) {
        switch(i) {
            case 0: return "Фамилия";
            case 1: return "Имя";
        }
    }
}

```

```

        case 2: return "Отчество";
        case 3: return "Дата рождения";
    }
    return null;
}

@Override
public Class<?> getColumnClass(int i) { return String.class; }

@Override
public int getRowCount() { return persons.size(); }

@Override
public Object getValueAt(int i, int j) {
    switch(j) {
        case 0: return persons.get(i).getLastName();
        case 1: return persons.get(i).getFirstName();
        case 2: return persons.get(i).getMiddleName();
        case 3:
            if(persons.get(i).getBirthday() != null) {
                return format.format(
                    persons.get(i).getBirthday()
                );
            } else {
                return "неизвестна";
            }
    }
    return null;
}

@Override
public void setValueAt(Object value, int i, int j) {}

@Override
public boolean isCellEditable(int i, int j) { return false; }

private final List<TableModelListener> listeners =
    new ArrayList<>();

@Override
public void addTableModelListener(TableModelListener listener) {
    listeners.add(listener);
}

@Override
public void removeTableModelListener(TableModelListener
    listener) {
    listeners.remove(listener);
}
}

```

```

public List<Person> getPersons() { return persons; }

public void update() {
    for(TableModelListener listener : listeners) {
        listener.tableChanged(new TableModelEvent(this));
    }
}
}

```

Содержимое файла SortButtonListener.java:

```

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Collections;

import javax.swing.JComboBox;

public class SortButtonListener implements ActionListener {
    private final JComboBox<String> comboBox;
    private final PersonTableModel model;

    public SortButtonListener(JComboBox<String> comboBox,
                              PersonTableModel model) {
        this.comboBox = comboBox;
        this.model = model;
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        Collections.sort(
            model.getPersons(),
            new PersonComparator((String)comboBox.getSelectedItem())
        );
        model.update();
    }
}

```

Содержимое файла PersonComparator.java:

```

import java.util.Calendar;
import java.util.Comparator;

public class PersonComparator implements Comparator<Person> {
    private final String type;

    public PersonComparator(String type) {
        this.type = type;
    }
}

```

```

@Override
public int compare(Person p1, Person p2) {
    Calendar c1 = Calendar.getInstance();
    c1.setTime(p1.getBirthday());
    Calendar c2 = Calendar.getInstance();
    c2.setTime(p2.getBirthday());
    switch(type) {
        case "по дате": return c1.compareTo(c2);
        case "по дню недели": return Integer.compare(
            c1.get(Calendar.DAY_OF_WEEK),
            c2.get(Calendar.DAY_OF_WEEK)
        );
        case "по месяцу": return Integer.compare(
            c1.get(Calendar.MONTH),
            c2.get(Calendar.MONTH)
        );
    }
    throw new IllegalArgumentException();
}
}
}

```

Содержимое файла persons.dtd:

```

<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT persons (person*)>
<!ELEMENT person (first-name,middle-name,last-name,birthday)>
<!ELEMENT first-name (#PCDATA)>
<!ELEMENT middle-name (#PCDATA)>
<!ELEMENT last-name (#PCDATA)>
<!ELEMENT birthday (#PCDATA)>

```

Содержимое файла persons.xml:

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE persons SYSTEM "persons.dtd" >
<persons>
  <person>
    <first-name>Иванов</first-name>
    <middle-name>Иван</middle-name>
    <last-name>Иванович</last-name>
    <birthday>1982-01-01</birthday>
  </person>
  <person>
    <first-name>Петров</first-name>
    <middle-name>Пётр</middle-name>
    <last-name>Петрович</last-name>
    <birthday>1965-02-25</birthday>
  </person>
  <person>
    <first-name>Сидоров</first-name>
    <middle-name>Сидор</middle-name>

```

```

    <last-name>Сидорович</last-name>
    <birthday>1978-05-03</birthday>
</person>
<person>
    <first-name>Васильев</first-name>
    <middle-name>Василий</middle-name>
    <last-name>Васильевич</last-name>
    <birthday>1991-02-13</birthday>
</person>
<person>
    <first-name>Кюльбрехт</first-name>
    <middle-name>Сигизмунд</middle-name>
    <last-name>Эдуардович</last-name>
    <birthday>2000-01-31</birthday>
</person>
</persons>

```

Необходимо выполнить оптимизацию структуры классов приложения в соответствии с шаблоном проектирования «Factory Method», с помощью которого необходимо создавать объекты классов, реализующих интерфейс Comparator. При этом создание таких объектов нужно реализовать через объект класса JComboBox, который может быть параметризован любым классом. При отображении строк в выпадающем списке, используется метод toString() класса, которым параметризован компонент JComboBox. Также можно получить выбранный пользователем объект методом getSelectedItemAt() класса JComboBox, выполнив явное приведение типов к классу, которым параметризован JComboBox.

После оптимизации структуры классов необходимо добавить в таблицу ещё один строковый столбец и способ сортировки по этому столбцу в соответствии с вариантом.

**Варианты:**

1. почтовый адрес (строка, содержащая улицу, номер дома, номер квартиры, почтовый индекс) с сортировкой по индексу;
2. номер телефона (код телефона в круглых скобках и номер телефона после скобок) с сортировкой по части номера телефона после скобок;
3. адрес электронной почты с сортировкой по доменному имени почтового сервера;
4. номер паспорта (серия и собственно номер, разделённые пробелом) с сортировкой по собственно номеру;
5. номер кредитной карты (четыре группы цифр, разделённые пробелом) с сортировкой по последней группе цифр;
6. почтовый адрес (строка, содержащая улицу, номер дома, номер квартиры, почтовый индекс) с сортировкой количеству символов в названии улицы;

7. номер телефона (код телефона в круглых скобках и номер телефона после скобок) с сортировкой по количеству цифр в коде телефона;
8. адрес электронной почты с сортировкой по длине имени пользователя;
9. доменное имя с сортировкой по уровню домена;
10. номер кредитной карты (четыре группы цифр, разделённые пробелом) с сортировкой по количеству цифр в первой группе.

## 4.2. Задание на лабораторную работу №4

Даны списки некоторых объектов, хранящихся в некотором внешнем хранилище (см. задание своего варианта). Необходимо написать приложение, считывающее из аргументов командной строки тип и имена файлов, в которых хранятся объекты. Первый аргумент определяет тип используемых файлов (csv или xml формат). Далее приложение выполняет требуемую обработку данных и выводит их на экран.

*Указание:*

Создание списков объектов на основе файлов различных типов реализовать через шаблон проектирования «Builder». Анализ расширения файла и выбор нужного «строителя» реализовать через шаблон проектирования «Factory Method». Сам класс, реализующий «Factory Method», создавать через шаблон проектирования «Singleton».

*Варианты:*

1. Даны: список грузовых автомобилей (гос. номер, марка, водитель) и список рейсов (дата, масса груза, длина маршрута, гос. номер автомобиля). Определить суммарный пробег каждого автомобиля и среднюю массу перевозимого им груза за последний месяц.
2. Даны: список счетов банка (номер счёта, владелец) и список платежей (дата, счёт плательщика, счёт получателя, дата платежа, сумма платежа). Вывести для каждого счёта баланс по дням (в порядке возрастания даты).
3. Даны: список квартир (номер квартиры, подъезда, этаж, имя владельца) и список заявок на ремонт (квартира, дата выполнения работ, описание работы, планируемое время работы в часах). Вывести для каждой квартиры суммарное время выполнения всех работ (в порядке убывания этого времени).
4. Даны: список организаций (название, адрес) и список направлений на распределение (организация, фамилия и инициалы студента, специальность, дата выхода на работу). Вывести для каждой организации список всех направлений на распределение с сортировкой по дате выхода на работу, сам список организаций отсортировать по убыванию количества заявок.

5. Даны: список адвокатов (фамилия, имя, отчество, специализация) и список судебных процессов (адвокат, начало и окончание процесса, описание, результативность: выиграл или проиграл). Вывести для каждого адвоката список его дел (в порядке возрастания даты начала процесса) с подсчётом результативности адвоката (отношение количества часов, затраченных на успешные дела, к общему количеству часов по всем делам). Сам список адвокатов отсортировать по специализации с сортировкой по убыванию результативности адвоката по каждой специализации.

6. Даны: список абонентов телефонного оператора (номер телефона, баланс счёта в рублях, имя владельца) и исходящих звонков (абонент, дата и время начала и окончания звонка, стоимость минуты звонка в рублях). Вывести список абонентов, отсортированных по имени, и для каждого абонента вывести список звонков по возрастанию времени начала звонка. Для каждого абонента вывести актуальный баланс (за вычетом стоимости каждого звонка).

7. Даны: список планет (название, радиус, масса) и список спутников (родительская планета, название, масса, дата открытия). Вывести для каждой планеты список её спутников в порядке их открытия, сам список планет отсортировать по радиусу планеты, для каждой планетарной системы подсчитать общую массу.

8. Даны: список станков (название, производительность: количество деталей в час, прибыль с одной детали в рублях) и список ремонтных работ за месяц (станок, дата ремонта, описание, длительность в часах, стоимость в рублях). Вывести для каждого станка список всех ремонтных работ (с сортированием по дате) и доходность каждого станка (прибыль от всех произведенных за месяц деталей, за вычетом времени, проведённого на ремонте, и стоимости самого ремонта, при этом считать, что станок в штатном режиме работает 10 часов в день, 25 дней в месяц).

9. Даны: список вопросов теста (содержимое, максимальная оценка в баллах) и список вариантов ответов (вопрос, содержание ответа, правильный ли это вариант). Вывести для каждого вопроса список вариантов ответов в случайном порядке, подсчитать вероятность угадывания ответа для каждого вопроса. Сам список вопросов вывести в порядке убывания максимальной оценки.

10. Даны: список экскурсий туристического оператора (описание экскурсии, стоимость экскурсии) и список туристов (экскурсия, на которую записан турист; имя; возраст; является ли студентом). Вывести для каждой экскурсии список её участников. Список участников отсортировать по имени, список экскурсий - по убыванию выручки за экскурсию (стоимость экскурсии суммируется по всем участникам, при этом студенты оплачивают лишь половину стоимости, подростки до 15 лет - треть стоимости, дети до 7 лет участвуют в экскурсии бесплатно).

## 5. СТРУКТУРНЫЕ ШАБЛОНЫ ПРОЕКТИРОВАНИЯ GOF

### 5.1. Задание на лабораторную работу №5

Рассмотрим исходный приложения, которое в окне строит некоторое графическое изображение.

Содержимое файла Point.java:

```
import java.awt.Graphics2D;
public class Point {
    private final int x;
    private final int y;
    public Point(int x, int y) { this.x = x; this.y = y; }
    public int getX() { return x; }
    public int getY() { return y; }
    public void paint(Graphics2D g) {
        g.fillOval(x - 1, y - 1, 2, 2);
    }
}
```

Содержимое файла Line.java:

```
import java.awt.Graphics2D;
public class Line {
    private final Point begin;
    private final Point end;
    public Line(Point begin, Point end) {
        this.begin = begin;
        this.end = end;
    }
    public Point getBegin() { return begin; }
    public Point getEnd() { return end; }
    public void paint(Graphics2D g) {
        g.drawLine(begin.getX(), begin.getY(), end.getX(),
                    end.getY());
    }
}
```

Содержимое файла Circle.java:

```
import java.awt.Graphics2D;
public class Circle {
    private final Point center;
    private final int radius;
    private final int x;
    private final int y;
    private final int diameter;
    public Circle(Point center, int radius) {
        this.center = center;
        this.radius = radius;
    }
}
```



```

    x = center.getX() - radius;
    y = center.getY() - radius;
    diameter = 2 * radius;
}
public Point getCenter() { return center; }
public int getRadius() { return radius; }
public void paint(Graphics2D g) {
    g.fillOval(x, y, diameter, diameter);
}
}

```

Содержимое файла CircleBorder.java:

```

import java.awt.Graphics2D;
public class CircleBorder {
    private final int x;
    private final int y;
    private final int diameter;
    public CircleBorder(Point center, int radius) {
        x = center.getX() - radius;
        y = center.getY() - radius;
        diameter = 2 * radius;
    }
    public void paint(Graphics2D g) {
        g.drawOval(x, y, diameter, diameter);
    }
}

```

Содержимое файла Rectangle.java:

```

import java.awt.Graphics2D;
public class Rectangle {
    private final int x;
    private final int y;
    private final int width;
    private final int height;
    public Rectangle(Point center, int width, int height) {
        x = center.getX() - width / 2;
        y = center.getY() - height / 2;
        this.width = width;
        this.height = height;
    }
    public void paint(Graphics2D g) {
        g.fillRect(x, y, width, height);
    }
}

```

Содержимое файла RectangleBorder.java:

```

import java.awt.Graphics2D;
public class RectangleBorder {

```

```

private final int x;
private final int y;
private final int width;
private final int height;
public RectangleBorder(Point center, int width, int height) {
    x = center.getX() - width / 2;
    y = center.getY() - height / 2;
    this.width = width;
    this.height = height;
}
public void paint(Graphics2D g) {
    g.drawRect(x, y, width, height);
}
}

```

Содержимое файла Polyline.java:

```

import java.awt.Graphics2D;
public class Polyline {
    private final int[] x;
    private final int[] y;
    private final int n;
    public Polyline(Point ... points) {
        n = points.length;
        x = new int[n];
        y = new int[n];
        for(int i = 0; i < n; i++) {
            x[i] = points[i].getX();
            y[i] = points[i].getY();
        }
    }
    public void paint(Graphics2D g) {
        g.drawPolyline(x, y, n);
    }
}

```

Содержимое файла Polygon.java:

```

import java.awt.Graphics2D;
public class Polygon {
    private final int[] x;
    private final int[] y;
    private final int n;
    public Polygon(Point ... points) {
        n = points.length;
        x = new int[n];
        y = new int[n];
        for(int i = 0; i < n; i++) {
            x[i] = points[i].getX();
            y[i] = points[i].getY();
        }
    }
}

```

```

    }
    public void paint(Graphics2D g) {
        g.fillPolygon(x, y, n);
    }
}

```

Содержимое файла Colour.java:

```

import java.awt.Color;
import java.awt.Graphics2D;
public class Colour {
    private final Color color;
    public Colour(Color color) { this.color = color; }
    public void paint(Graphics2D g) {
        g.setColor(color);
    }
}

```

Содержимое файла Stroke.java:

```

import java.awt.BasicStroke;
import java.awt.Graphics2D;
public class Stroke {
    private final java.awt.Stroke stroke;
    public Stroke(int width) {
        stroke = new BasicStroke((float)width);
    }
    public void paint(Graphics2D g) {
        g.setStroke(stroke);
    }
}

```

Содержимое файла LinearGradient.java:

```

import java.awt.Color;
import java.awt.Graphics2D;
import java.awt.LinearGradientPaint;
import java.awt.MultipleGradientPaint;
public class LinearGradient {
    private final MultipleGradientPaint gradient;
    public LinearGradient(Line line, Color beginColor,
        Color endColor) {
        gradient = new LinearGradientPaint(
            line.getBegin().getX(),
            line.getBegin().getY(),
            line.getEnd().getX(),
            line.getEnd().getY(),
            new float[] {0, 1},
            new Color[] {beginColor, endColor}
        );
    }
    public void paint(Graphics2D g) {
        g.setPaint(gradient);
    }
}

```

Содержимое файла RadialGradient.java:

```
import java.awt.Color;
import java.awt.Graphics2D;
import java.awt.MultipleGradientPaint;
import java.awt.RadialGradientPaint;
public class RadialGradient {
    private final MultipleGradientPaint gradient;
    public RadialGradient(Circle circle, Color centerColor,
        Color borderColor) {
        gradient = new RadialGradientPaint(
            circle.getCenter().getX(),
            circle.getCenter().getY(),
            circle.getRadius(),
            new float[] {0, 1},
            new Color[] {centerColor, borderColor}
        );
    }
    public void paint(Graphics2D g) {
        g.setPaint(gradient);
    }
}
```

Содержимое файла Picture.java:

```
import java.awt.Canvas;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import java.util.Collection;
public class Picture extends Canvas {
    private final Collection<Object> figures;
    public Picture(Collection<Object> figures) {
        this.figures = figures;
    }
    @Override
    public void paint(Graphics g) {
        super.paint(g);
        Graphics2D g2 = (Graphics2D)g;
        g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
            RenderingHints.VALUE_ANTIALIAS_ON);
        g2.setColor(Color.WHITE);
        java.awt.Rectangle r = getBounds();
        g2.fillRect(r.x, r.y, r.width, r.height);
        for(Object figure : figures) {
            if(figure instanceof Point) {
                ((Point)figure).paint(g2);
            } else if(figure instanceof Line) {
                ((Line)figure).paint(g2);
            } else if(figure instanceof Rectangle) {

```

```

        ((Rectangle)figure).paint(g2);
    } else if (figure instanceof RectangleBorder) {
        ((RectangleBorder)figure).paint(g2);
    } else if (figure instanceof Circle) {
        ((Circle)figure).paint(g2);
    } else if (figure instanceof CircleBorder) {
        ((CircleBorder)figure).paint(g2);
    } else if (figure instanceof Polyline) {
        ((Polyline)figure).paint(g2);
    } else if (figure instanceof Polygon) {
        ((Polygon)figure).paint(g2);
    } else if (figure instanceof Colour) {
        ((Colour)figure).paint(g2);
    } else if (figure instanceof LinearGradient) {
        ((LinearGradient)figure).paint(g2);
    } else if (figure instanceof RadialGradient) {
        ((RadialGradient)figure).paint(g2);
    } else if (figure instanceof Stroke) {
        ((Stroke)figure).paint(g2);
    }
}
}
}
}
}

```

Содержимое файла MainFrame.java:

```

import java.awt.Insets;
import javax.swing.JFrame;
public class MainFrame extends JFrame {
    public MainFrame(String title, int width, int height,
        Picture picture) {
        super("Рисунок");
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        add(picture);
        setVisible(true);
        setResizable(false);
        Insets insets = getInsets();
        setSize(width + insets.left + insets.right,
            height + insets.top + insets.bottom);
        setLocationRelativeTo(null);
    }
}

```

Содержимое файла Runner.java:

```

import java.awt.Color;
import java.util.ArrayList;
import java.util.Collection;
import javax.swing.UIManager;
import javax.swing.UnsupportedLookAndFeelException;
public class Runner {

```

```

public static void main(String[] args) throws
    ClassNotFoundException, InstantiationException,
    IllegalAccessException, UnsupportedLookAndFeelException {
    Collection<Object> figures = new ArrayList<>();
    figures.add(new LinearGradient(new Line(new Point(0, 250),
        new Point(0, 350)), new Color(0, 128, 255),
        new Color(0, 0, 128)));
    figures.add(new Rectangle(new Point(275, 300), 550, 100));
    figures.add(new Colour(Color.YELLOW));
    figures.add(new Circle(new Point(450, 50), 25));
    figures.add(new Colour(Color.ORANGE));
    figures.add(new Stroke(2));
    figures.add(new Line(new Point(200, 25),
        new Point(200, 125)));
    figures.add(new Colour(Color.ORANGE));
    figures.add(new Stroke(2));
    figures.add(new Line(new Point(175, 50),
        new Point(225, 50)));
    figures.add(new Colour(Color.BLACK));
    figures.add(new Stroke(2));
    figures.add(new Polyline(new Point(50, 200),
        new Point(200, 50), new Point(500, 200)));
    figures.add(new Colour(new Color(255, 200, 200)));
    figures.add(new Rectangle(new Point(225, 175), 150, 50));
    figures.add(new Colour(Color.RED));
    figures.add(new Stroke(1));
    figures.add(new RectangleBorder(new Point(225, 175),
        150, 50));
    figures.add(new Colour(new Color(255, 200, 200)));
    figures.add(new Rectangle(new Point(212, 137), 75, 25));
    figures.add(new Colour(Color.RED));
    figures.add(new Stroke(1));
    figures.add(new RectangleBorder(new Point(212, 137),
        75, 25));
    figures.add(new Colour(new Color(0, 128, 0)));
    figures.add(new Point(175, 175));
    figures.add(new Colour(new Color(0, 128, 0)));
    figures.add(new Point(200, 175));
    figures.add(new Colour(new Color(0, 128, 0)));
    figures.add(new Point(225, 175));
    figures.add(new Colour(new Color(0, 128, 0)));
    figures.add(new Point(250, 175));
    figures.add(new Colour(new Color(0, 128, 0)));
    figures.add(new Point(275, 175));
    figures.add(new Colour(Color.LIGHT_GRAY));
    figures.add(new Polygon(new Point(50, 200),
        new Point(500, 200), new Point(400, 300),
        new Point(150, 300)));
    figures.add(new Colour(Color.GRAY));
}

```

```

figures.add(new Stroke(2));
figures.add(new Polyline(new Point(50, 200),
    new Point(500, 200), new Point(400, 300),
    new Point(150, 300), new Point(50, 200)));
figures.add(new RadialGradient(new Circle(
    new Point(175, 250), 25), Color.CYAN, Color.WHITE));
figures.add(new Circle(new Point(175, 250), 25));
figures.add(new Colour(Color.MAGENTA));
figures.add(new Stroke(3));
figures.add(new CircleBorder(new Point(175, 250), 25));
figures.add(new RadialGradient(new Circle(
    new Point(275, 250), 25), Color.CYAN, Color.WHITE));
figures.add(new Circle(new Point(275, 250), 25));
figures.add(new Colour(Color.MAGENTA));
figures.add(new Stroke(3));
figures.add(new CircleBorder(new Point(275, 250), 25));
figures.add(new RadialGradient(new Circle(
    new Point(375, 250), 25), Color.CYAN, Color.WHITE));
figures.add(new Circle(new Point(375, 250), 25));
figures.add(new Colour(Color.MAGENTA));
figures.add(new Stroke(3));
figures.add(new CircleBorder(new Point(375, 250), 25));
UIManager.setLookAndFeel(
    UIManager.getSystemLookAndFeelClassName());
new MainFrame("Рисунок", 550, 350, new Picture(figures));
}
}

```

Необходимо выполнить оптимизацию структуры классов приложения в соответствии с шаблоном проектирования «Decorator», с помощью которого необходимо графическим примитивам задавать цвет фона и линий, ширину линий. После чего необходимо, с использованием переработанных классов, разработать ещё одно приложения, которое будет строить в окне рисунок в соответствии с вариантом. При этом в рисунке необходимо использовать все графические примитивы, цвета и способы заливки.

**Варианты:**

1. Домик в деревне.
2. Новогодняя елка.
3. Космический корабль.
4. Автомобиль.
5. Яхта.
6. Подводная лодка.
7. Паровоз.
8. Снеговик.
9. Клоун.
10. Ваза с цветами.

## 5.2. Задание на лабораторную работу №6

Разработать приложение в соответствии с вариантом.

### *Варианты:*

1. Разработать приложение, генерирующее несколько последовательностей чисел (по заранее заданному алгоритму, например, последовательность чисел Фибоначчи, последовательность простых чисел, последовательность факториалов целых неотрицательных чисел). Генерирование типа последовательности и количество генерируемых элементов должно определяться пользователем. Для каждой последовательности после генерации указать время работы соответствующего алгоритма. Определение этого времени реализовать, используя шаблон проектирования «Decorator».

2. Разработать приложение, создающее список плоских геометрических фигур (круг, квадрат, треугольник и т.д., но не менее 5 фигур). Создание списка должно быть реализовано несколькими способами (ввод с клавиатуры, чтение из файла, генерирование случайным образом). Далее необходимо вывести данные о тех фигурах, площадь которых меньше средней площади всех фигур. При этом вычисление площади фигур может производиться обычным способом, а может округляться по избытку, по недостатку или до ближайшего значения до указанного пользователем количества знаков после запятой (вид округления тоже указывается пользователем). Округление реализовать самостоятельно (через соответствующие методы класса Math), используя шаблон проектирования «Decorator».

3. Разработать приложение, читающее список неких товаров (название, вес одной единицы, стоимость одной единицы, количество единиц товара) из некоторого файла (формат файла можно выбрать произвольно). Далее приложение должно выводить этот список на экран, отсортировав его по суммарному весу всех единиц каждого товара. Затем приложение должно выводить этот же список на экран ещё раз, отсортировав его по суммарной стоимости всех единиц каждого товара. Для сортировки списка товаров использовать возможности класса Collections. После каждого вывода отсортированного списка необходимо вывести количество операций сравнения, использовавшееся при сортировке списка. Подсчёт количества операций сравнения реализовать, используя шаблон проектирования «Decorator».

4. Разработать приложение с графическим пользовательским интерфейсом, представляющее собой простейший калькулятор целых чисел, содержащий одно текстовое поле (заблокированное для непосредственного редактирования) и 16 кнопок: 10 цифр, 4 операции (сложение, вычитание, умножение, деление), кнопку «равно» и кнопку «+/-». Приложение должно



работать в демонстрационном режиме, то есть после запуска оно должно считать нажатия пользователя на клавиши, и после 10 нажатий, остальные действия игнорировать. Реализацию ограничения демонстрационной версии организовать через шаблон проектирования «Decorator».

5. Разработать приложение, формирующее прайс-лист товаров некоторого магазина. Список товаров (категория, название, закупочная стоимость) считывается из некоторого файла (формат можно выбрать произвольным образом). Также для каждой категории товаров в отдельном файле задаётся процент наценки, если для некоторой категории процент наценки не указан, то принимается наценка по умолчанию (заданная в том же файле). Ещё в одном файле определяются скидки на некоторые конкретные товары, при чём на некоторые товары скидка определяется в процентах, для некоторых - в рублях. Приложение должно вывести все товары с окончательной ценой. Расчет наценок и скидок реализовать через шаблон проектирования «Decorator».

6. Разработать приложение, формирующее отчёты по расписанию занятий. Расписание храниться в отдельном файле, для каждого пункта расписания определены: название предмета, тип занятия (лекция, практика, лабораторная), дата и время начала, длительность в академических часах, номер аудитории. Приложение должно уметь на основе этих данных генерировать три вида отчетов: расписание по конкретному предмету с сортировкой по типам занятий, расписание на конкретный день с сортировкой по времени, расписание для конкретной аудитории с указанием с сортировкой по дню проведения занятия. При этом отчёт должен выводиться в файл на диске, но при ошибке записи отчёта (при возникновении IOException или другой ошибки), в отдельный файл ошибок необходимо вывести сообщение об ошибке, а генерацию отчёта прервать (с удалением ошибочного файла). Обработку ошибок в приложении реализовать, используя шаблон проектирования «Decorator».

7. Разработать приложение, выводящее список товаров мебельного магазина, отсортированного по цене. У каждого товара должны быть заданы: категория, название, артикул (уникальный код) и цена. Для мебели в категории «шкафы» необходимо дополнительно указывать тип дверей (обычные, купе, стеклянные). Для мебели категории «стол» необходимо указывать количество выдвижных ящиков. Для мебели категории «стул» необходимо указывать наличие спинки. Кроме продажи отдельных предметов мебели в магазине могут продаваться наборы со скидкой 5% от общей стоимости всех предметов мебели этого набора. Подсчёт стоимости набора мебели организовать через шаблон проектирования «Composite».

8. Разработать приложение, выводящее список товаров компьютерного магазина, отсортированного по цене. У каждого товара должны быть заданы: категория, название, артикул (уникальный код) и цена. Для комплектующих в категории «процессор» необходимо дополнительно указы-

вать тактовую частоту, количество ядер и тип процессорного разъёма. Для комплектующих в категории «оперативная память» необходимо указывать объём в мегабайтах. Для комплектующих категории «видеокарты» необходимо указывать наличие активного охлаждения и объём памяти. Для комплектующих категории «жёсткие диски» необходимо указывать объём в гигабайтах и скорость вращения шпинделя. Для комплектующих категории «материнская плата» необходимо указывать наличие встроенной видеокарты, максимальный поддерживаемый объём оперативной памяти, тип процессорного разъёма. Для комплектующих категории «корпус» необходимо указывать мощность блока питания и тип корпуса (FullTower, Tower, MiniTower, Desktop). Для комплектующих категории «монитор» необходимо указывать размер диагонали экрана и соотношение сторон экрана. Для комплектующих категории «устройства ввода» необходимо указывать тип клавиатуры (обычная, мультимедиа) и количество кнопок мыши. Кроме продажи отдельных комплектующих магазин может продавать системные блоки (корпус, материнская плата, процессор, оперативная память, жёсткий диск, опционально: видеокарта) с надбавкой 15% за сборку или компьютеры целиком (системный блок, монитор, устройства ввода) с надбавкой 5% за сборку. Подсчёт стоимости товаров организовать через шаблон проектирования «Composite».

9. Разработать приложение, вычисляющее длительность технологических процессов для различных видов деталей. Есть набор элементарных технологических операций, для каждой из которых задаются свои параметры, влияющие на скорость ее выполнения. Операция «Подготовка заготовки»: диаметр стального цилиндра, диаметр заготовки, длина заготовки - время линейно зависит от объёма считываемого металла (по 1 мм<sup>3</sup>/с). Операция «Нарезка резьбы»: диаметр заготовки, длина резьбы, шаг резьбы - время линейно зависит от общей длины резьбы (длина окружности заготовки умноженная на длину резьбы, поделенную на шаг резьбы даёт общую длину резьбы, время операции равно 1 см/с). Операция «Термообработка»: температура, время выдерживания, скорость охлаждения - время операции складывается из времени разогревания (линейно зависит от температуры: 10°С/с), времени выдерживания и времени охлаждения. Элементарные технические операции используются для производится нескольких видов деталей: винтов (термообработка заготовки при температуре 1000°С с выдерживанием в течение 10 минут и охлаждением со скоростью 0,25°С/с, подготовка заготовки, нарезка резьбы); анкерных болтов (термообработка заготовки при температуре 1250°С с выдерживанием в течение 30 минут и охлаждением со скоростью 0,5°С/с, подготовка заготовки, нарезка резьбы, термообработка заготовки при температуре 1500°С с выдерживанием в течение 30 минут и охлаждением со скоростью 300°С/с); гусеничные траки (подготовка заготовки, термообработка заготовки при температуре 1000°С с выдерживанием в течение 60 минут и охлаждением со скоростью 300°С/с, термообработка заготовки при темпера-

туре 1000°C с выдерживанием в течение 30 минут и охлаждением со скоростью 0,1°C/с). Расчёт времени технологического процесса, состоящего из нескольких технологических операций, реализовать с использованием шаблона проектирования «Composite».

10. Разработать приложение, вычисляющее значение матричного выражения, записанного в Польской инверсной записи. Выражение состоит из операндов (обозначающих квадратные матрицы) и операций, операнды и операции отделяются друг от друга пробелом. В Польской инверсной записи сначала следуют операнды, а затем знак операции, что позволяет избежать использование скобок в выражении. Например, выражение «(a + b) \* (c + d) - e / f» в Польской инверсной записи примет вид: «a b + c d + \* e f / -». После ввода пользователем выражения программа должна запрашивать у него значение матриц, обозначенных операндами (при этом один и тот же операнд не должен запрашиваться дважды). При вводе значений матриц сначала запрашивается размер матриц (общий для всех операндов - квадратных матриц), затем тип матрицы (обычная, симметричная, верхнетреугольная, нижнетреугольная, диагональная, диагональная с постоянным элементом на главной диагонали), затем вводятся элементы матрицы. После чего вычисляется и выводится значение введённого выражения, но промежуточные результаты в памяти храниться не должны, вычисление всех операций (сложение, вычитание, умножение, деление: умножение на обратную матрицу) должно быть реализовано через шаблон проектирования «Composite».

*Указания:*

Для шаблона проектирования «Decorator» реализовать сначала базовую функциональность приложения, и только после этого, не меняя классы, реализующие эту функциональность, добавить класс (классы), реализующие с помощью декоратора расширенную функциональность. Аналогично для шаблона проектирования «Composite» сначала необходимо реализовать обработку базовых объектов, и только после этого с помощью компоновщика объединять их в составные объекты (в наборы объектов).

## СПИСОК РЕКОМЕНДОВАННЫХ ИСТОЧНИКОВ

1. Гамма, Э. Приёмы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес. – Санкт-Петербург: Питер, 2014. – 368 с.
2. Фаулер, М. Архитектура корпоративных программных приложений. – Москва: Вильямс, 2007. – 544 с.
3. Фаулер, М. UML. Основы. Краткое руководство по стандартному языку объектного моделирования / М. Фаулер. – Москва: Символ-Плюс, 2011.

Учебное издание

**ЕРМОЧЕНКО** Сергей Александрович

**КОРЧЕВСКАЯ** Елена Алексеевна

## **ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

Методические рекомендации

Технический редактор

*Г.В. Разбоева*

Компьютерный дизайн

*А.В. Табанюхова*

Подписано в печать 06.07.2023. Формат 60x84<sup>1</sup>/<sub>16</sub>. Бумага офсетная.

Усл. печ. л. 3,02. Уч.-изд. л. 2,45. Тираж 45 экз. Заказ 67.

Издатель и полиграфическое исполнение – учреждение образования  
«Витебский государственный университет имени П.М. Машерова».

Свидетельство о государственной регистрации в качестве издателя,  
изготовителя, распространителя печатных изданий

№ 1/255 от 31.03.2014.

Отпечатано на ризографе учреждения образования  
«Витебский государственный университет имени П.М. Машерова».

210038, г. Витебск, Московский проспект, 33.