МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Юго-Западный государственный университет»
(ЮЗГУ)

Кафедра программной инженерии

УТВЕРЖДАЮ
Проректор по учебной работе
_____О.Г. Локтионова
«_4_»_09_____ 2025 г.

ПАТТЕРН ПРОЕКТИРОВАНИЯ «СТРАТЕГИЯ»

Методические указания по выполнению практической работы по дисциплине "Методология программной инженерии" для студентов направления подготовки магистров 09.04.04 "Программная инженерия"

Курск 2025

УДК 004.65

Составители: Т.М. Белова

Рецензент
Кандидат технических наук, доцент кафедры вычислительной техники ЮЗГУ Т.И. Лапина

Паттерн проектирования «Стратегия»: методические указания по выполнению практической работы по дисциплине "Методология программной инженерии" для студентов направления подготовки магистров 09.04.04 "Программная инженерия"/ Юго-Зап. гос. ун-т; сост.: Т.М. Белова – Курск, 2025. – 17 с.: ил. 15.

Изложена последовательность действий по разработке и применению паттерна проектирования «Стратегия».

Материал предназначен для студентов направления подготовки магистров 09.04.04 «Программная инженерия», а также будет полезен студентам всех направлений подготовки, изучающим технологии разработки программных продуктов с использованием паттернов.

Текст печатается в авторской редакции.

Подписано в печать . Формат 60x84 1/16.
Усл. печ. л. 1,0. Уч.-изд. л. 0,9. Тираж 50 экз. Заказ . Бесплатно.
Юго-Западный государственный университет
305040, Курск, ул.50 лет Октября, 94.

Содержание

1 Цель практической работы

Целью практической работы является приобретение знаний, умений и навыков использования возможностей паттерна проектирования «Стратегия» в разработке информационно-вычислительных систем.

2 Основные понятия

При реализации проектов по разработке программных систем и моделированию бизнес-процессов встречаются ситуации, когда решение проблем в различных проектах имеют сходные структурные черты. Попытки выявить похожие схемы или структуры в рамках объектно-ориентированного анализа и проектирования привели к появлению понятия паттерна, которое из абстрактной категории превратилось в непременный атрибут современных CASE-средств.

Паттерны различаются степенью детализации и уровнем абстракции. Предлагается следующая общая классификация паттернов по категориям их применения:

- архитектурные паттерны,
- паттерны проектирования,
- паттерны анализа,
- паттерны тестирования,
- паттерны реализации.

Архитектурные паттерны (Architectural patterns) - множество предварительно определенных подсистем со спецификацией их ответственности, правил и базовых принципов установления отношений между ними.

Архитектурные паттерны предназначены для спецификации фундаментальных схем структуризации программных систем. Наиболее известными паттернами этой категории являются паттерны GRASP (General Responsibility Assignment Software Pattern). Эти паттерны относятся к уровню системы и подсистем, но не к уровню классов. Как правило, формулируются в обобщенной форме, используют обычную терминологию и не зависят от области приложения. Паттерны этой категории систематизировал и описал К. Ларман.

Паттерны проектирования (Design patterns) - специальные схемы для уточнения структуры подсистем или компонентов программной системы и отношений между ними.

Паттерны проектирования описывают общую структуру взаимодействия элементов программной системы, которые реализуют исходную проблему проектирования в конкретном контексте. Наиболее известными паттернами этой категории являются паттерны GoF (Gang of Four), названные в честь Э. Гаммы, Р. Хелма, Р. Джонсона и Дж. Влиссидеса, которые систематизировали их и представили общее описание. Паттерны GoF включают в себя 23 паттерна. Эти паттерны не зависят от языка реализации, но их реализация зависит от области приложения.

Паттерны анализа (Analysis patterns) - специальные схемы для представления общей организации процесса моделирования.

Паттерны анализа относятся к одной или нескольким предметным областям и описываются в терминах предметной области. Наиболее известными паттернами этой группы являются паттерны бизнес-моделирования ARIS (Architecture of Integrated Information Systems), которые характеризуют абстрактный уровень представления бизнес-процессов. Паттерны анализа конкретизируются в типовых моделях с целью выполнения аналитических оценок или имитационного моделирования бизнес-процессов.

Паттерны тестирования (Test patterns) - специальные схемы для представления общей организации процесса тестирования программных систем.

К этой категории паттернов относятся такие паттерны, как тестирование черного ящика, белого ящика, отдельных классов, системы. Паттерны этой категории систематизировал и описал М. Гранд. Некоторые из них реализованы в инструментальных средствах, наиболее известными из которых является IBM Test Studio. В связи с этим паттерны тестирования иногда называют стратегиями или схемами тестирования.

Паттерны реализации (Implementation patterns) - совокупность компонентов и других элементов реализации, используемых в структуре модели при написании программного кода.

Эта категория паттернов делится на следующие подкатегории: паттерны организации программного кода, паттерны оптимизации программного кода, паттерны устойчивости кода, паттерны разработки графического интерфейса пользователя и др. Паттерны этой категории описаны в работах М. Гранда, К. Бека, Дж. Тидвелла и др. Некоторые из них реализованы в популярных интегрированных средах программирования в форме шаблонов создаваемых проектов. В этом

случае выбор шаблона программного приложения позволяет получить некоторую заготовку программного кода.

Шаблон проектирования или паттерн (англ. design pattern) в разработке программного обеспечения — повторяемая архитектурная конструкция, представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста.

Технически, паттерны (шаблоны) проектирования - это абстрактные примеры правильного использования небольшого числа комбинаций простейших техник объектно-ориентированного программирования. Паттерны проектирования - это простые примеры, показывающие правильные способы организации взаимодействий между классами или объектами.

Паттерн "Стратегия" (Strategy) относится к группе, называемой поведенческой (Behavioral). Паттерны из этой группы определяют алгоритмы и взаимодействие между классами и объектами, то есть их поведение, увеличивая таким образом их гибкость.

В группу включены следующие паттерны:

- Цепочка обязанностей (Chain of responsibility),
- Команда (Command),
- Интерпретатор (Interpreter),
- Итератор (Iterator),
- Посредник (Mediator),
- Хранитель (Memento),
- Наблюдатель (Observer),
- Состояние (State),
- Стратегия (Strategy),
- Шаблонный метод (Template method),
- Посетитель (Visitor).

• Паттерн "Стратегия" (Strategy) - определяет семейство схожих алгоритмов и помещает каждый из них в собственный класс. После чего, алгоритмы можно взаимно заменять прямо во время исполнения программы.

3 Порядок выполнения лабораторной работы

Данный поведенческий шаблон предназначен, прежде всего, для использования в таких случаях, когда в решаемой задаче в каком-то определенном месте необходимо использовать различные алгоритмы в зависимости от конкретного состояния системы и/или ее окружения в конкретный момент времени. Рассматриваемый пример реализует алгоритмы, с помощью которых в зависимости от ситуации необходимо сложить, вычесть или перемножить два целых числа.

1. Открыть Eclipse и создать новый Java Project с именем Strategy (рисунок 1).

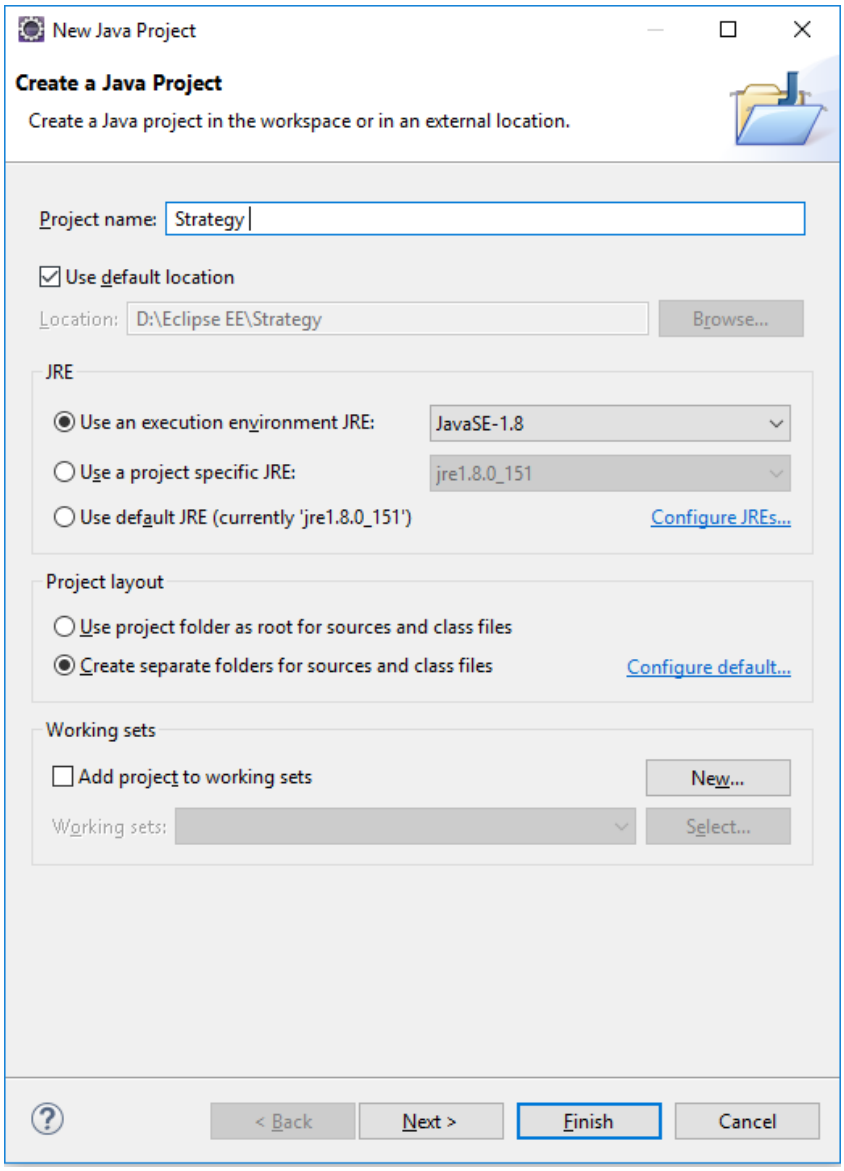

Рисунок 1

2. Схема задачи представлена ниже в виде UML-диаграммы, которая будет постепенно наполняться содержанием. Эта схема создается в этом проекте путем добавления PapyrusModel (рисунок 2).

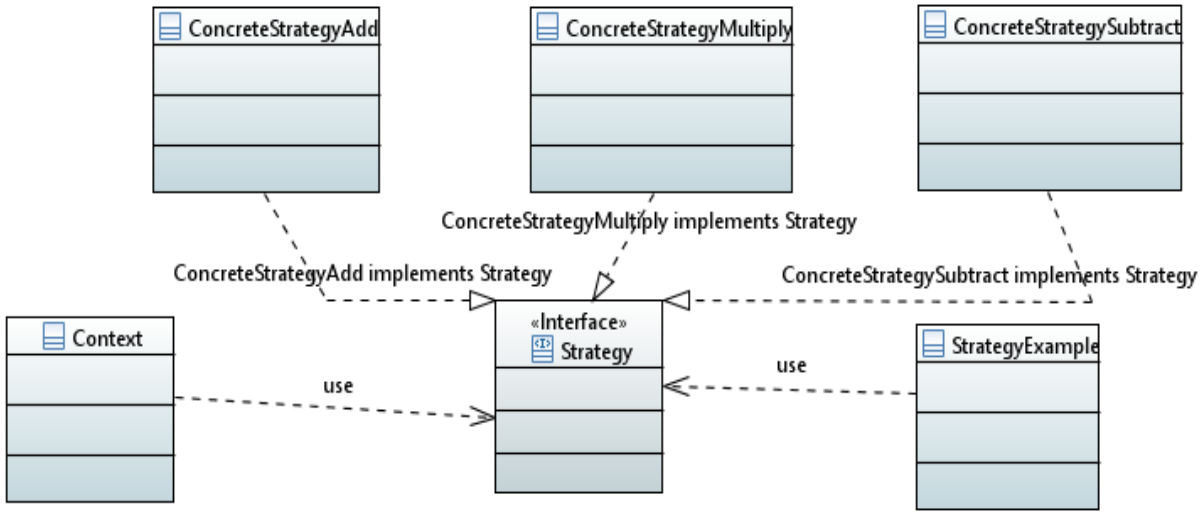

Рисунок 2

3. По этой схеме сгенерировать Java-код. Автоматически создаются заготовки для будущих классов и интерфейса.
4. Добавить в интерфейс Strategy метод execute, принимающий на вход два целых числа a и b (рисунок 3).

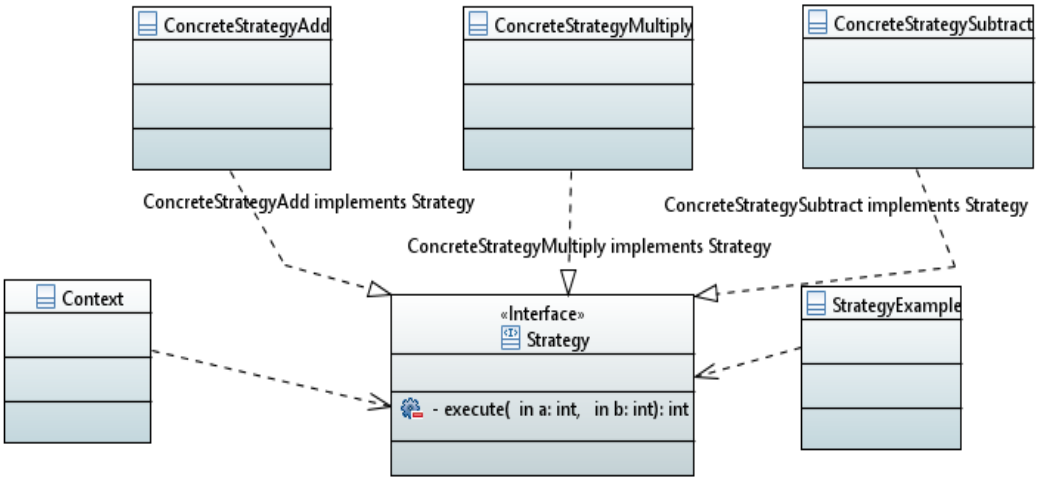

Рисунок 3

Интерфейс будет иметь следующий вид (рисунок 4):

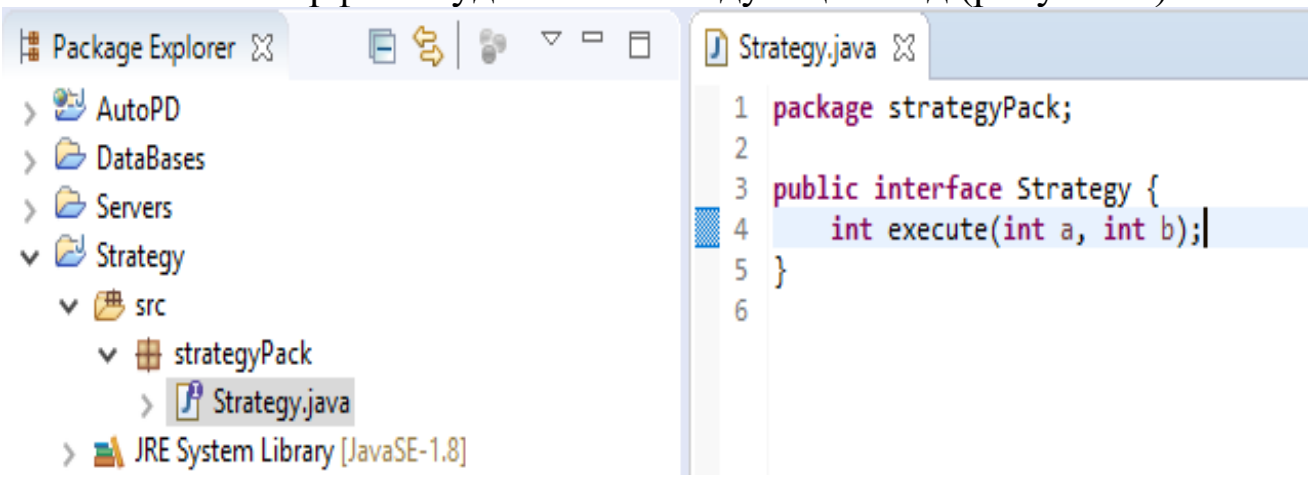

Рисунок 4

5. В классе ConcreteStrategyAdd нужно реализовать метод execute, который будет выводить результат сложения двух целых чисел (рисунок 5).

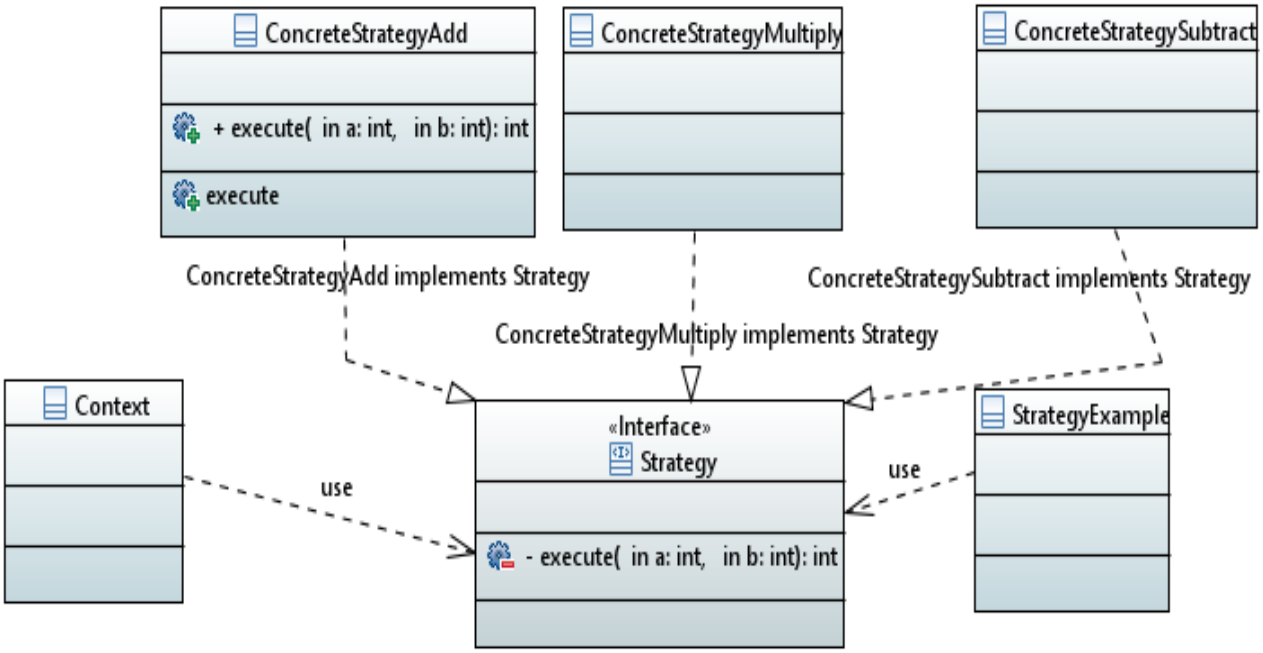

Рисунок 5

Класс будет иметь следующий вид (рисунок 6):

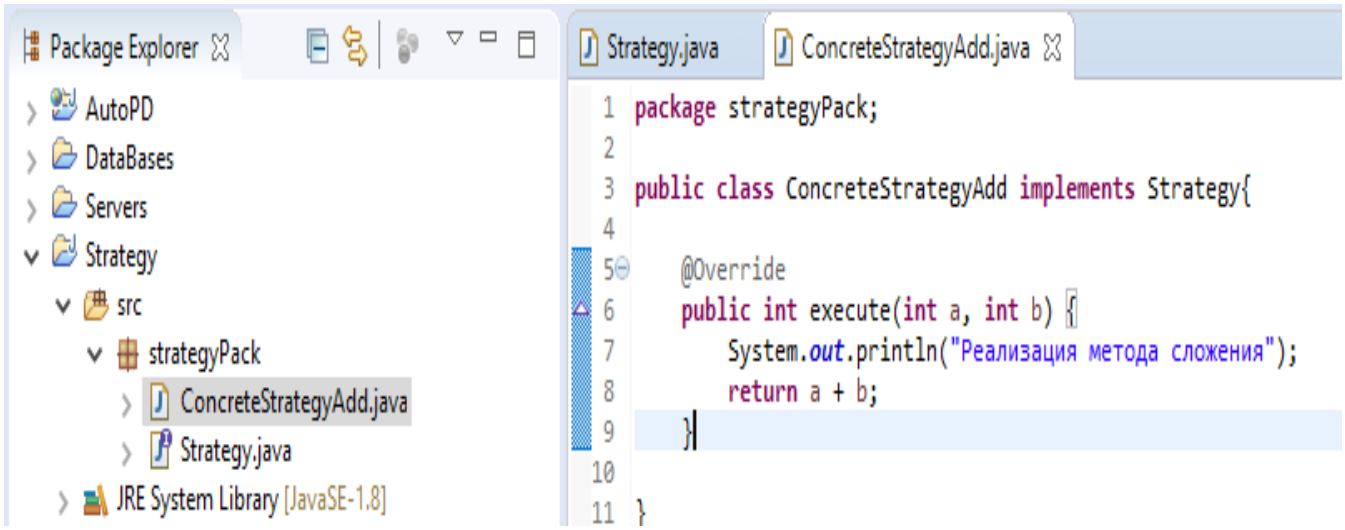

Рисунок 6

6. В классе ConcreteStrategySubtract нужно реализовать метод execute, который будет выводить результат вычитания двух целых чисел (рисунок 7).

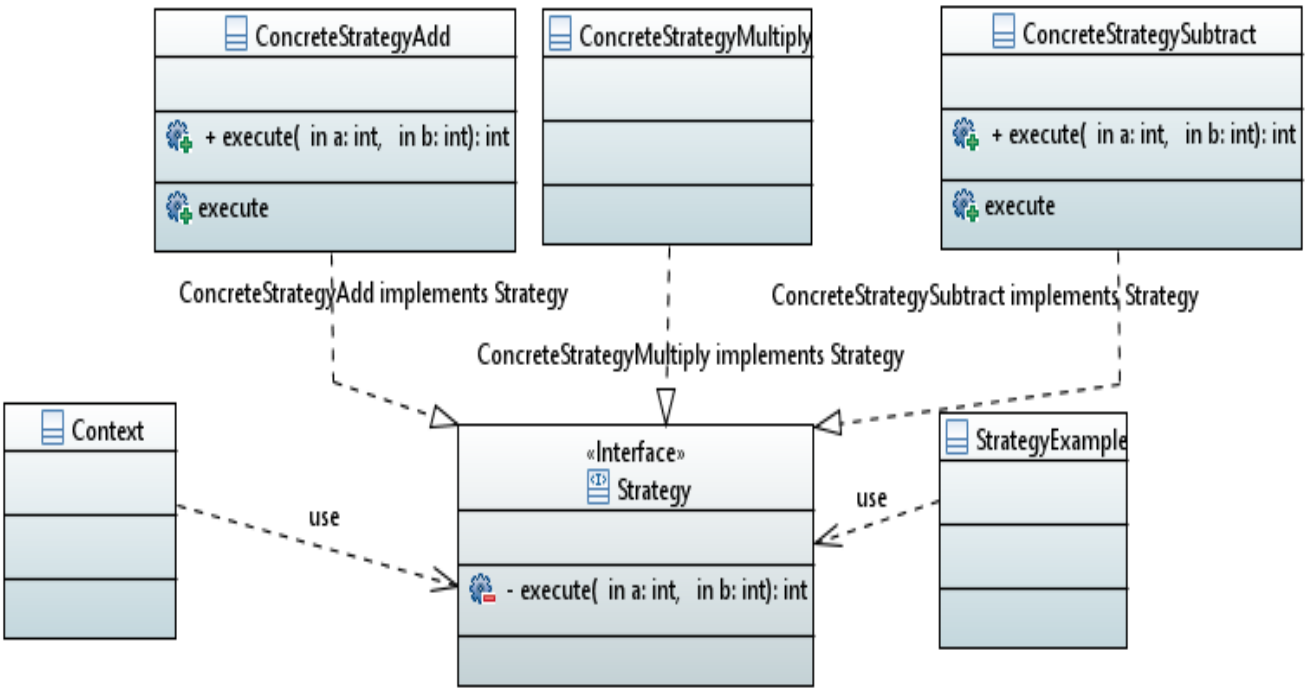

Рисунок 7

Класс будет иметь следующий вид (рисунок 8):

Strategy.java | ConcreteStrategyAdd.java | ConcreteStrategySubtract.java

```
1  package strategyPack;
2
3  public class ConcreteStrategySubtract implements Strategy {
4
5      @Override
6      public int execute(int a, int b) {
7          System.out.println("Реализация метода вычитания");
8          return a - b;
9      }
10
11 }
12
```

Рисунок 8

7. В классе ConcreteStrategyMultiply нужно реализовать метод execute, который будет выводить результат умножения двух целых чисел (рисунок 9).

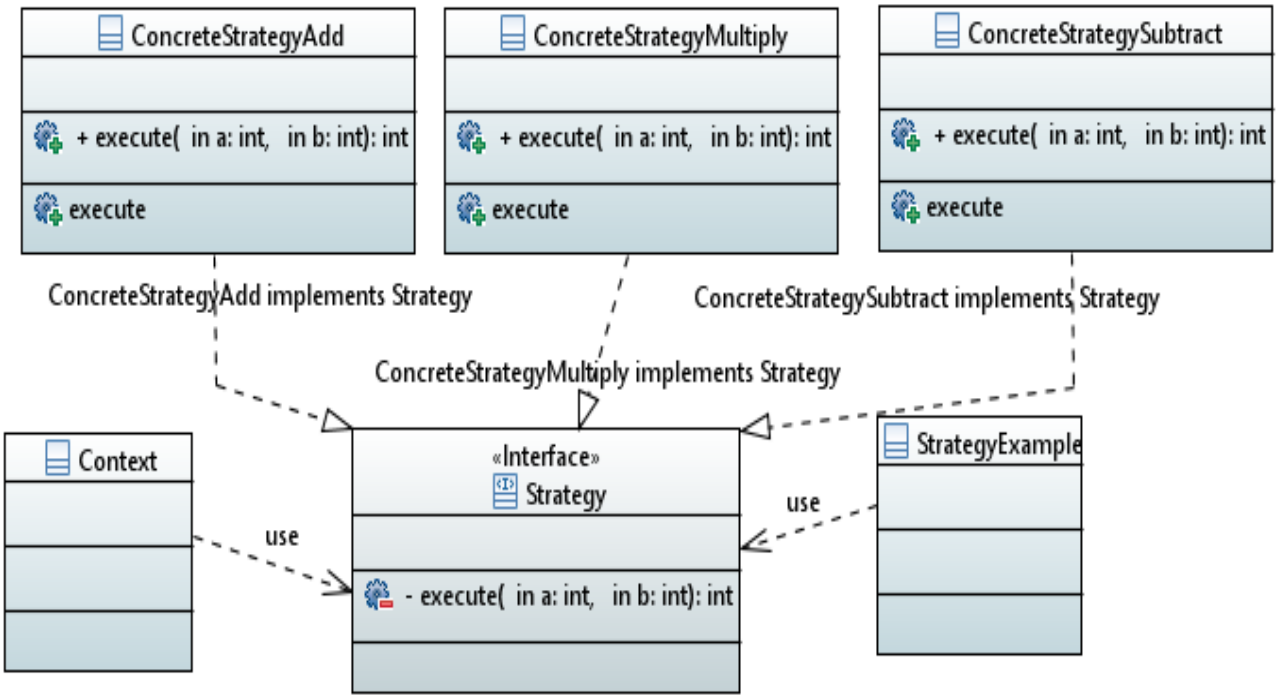

Рисунок 9

Класс будет иметь следующий вид (рисунок 10):

Strategy.java | ConcreteStrategyAdd.java | ConcreteStrategySubtract.java | ConcreteStrategyMultiply.java

```
1  package strategyPack;
2
3  public class ConcreteStrategyMultiply implements Strategy {
4
5      @Override
6      public int execute(int a, int b) {
7          System.out.println("Реализация метода умножения");
8          return a * b;
9      }
10
11 }
12
```

Рисунок 10

8. В класс Context (рисунок 11), использующий интерфейс Strategy, добавить методы setStrategy (устанавливает конкретную стратегию) и executeStrategy (реализует установленную стратегию).

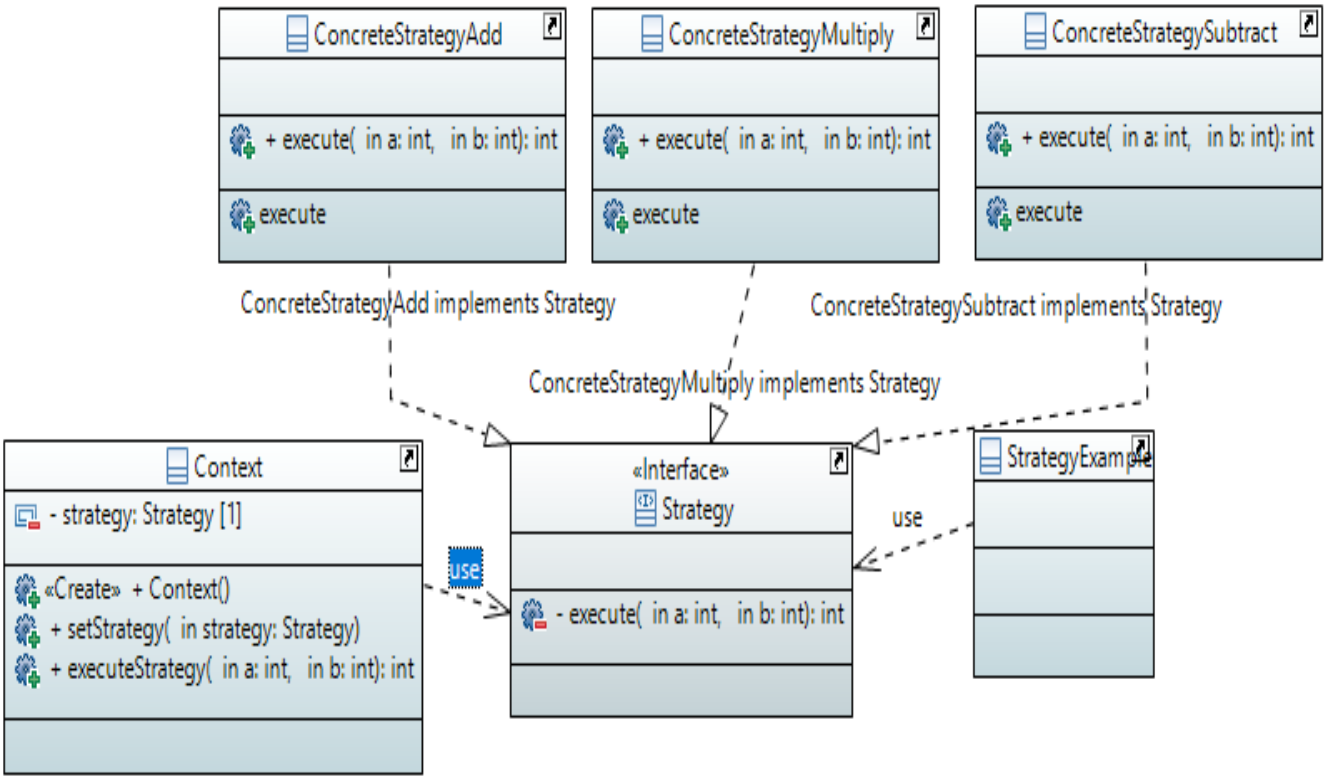

Рисунок 11

Класс будет иметь следующий вид (рисунок 12):

Context.java | Strategy.java | ConcreteStrategyAdd.java

```
1  package strategyPack;
2
3  class Context {
4      private Strategy strategy;
5
6      // конструктор
7      public Context() {
8      }
9
10     // выбор стратегии
11     public void setStrategy(Strategy strategy) {
12         this.strategy = strategy;
13     }
14
15     public int executeStrategy(int a, int b) {
16         return strategy.execute(a, b);
17     }
18 }
19
```

Рисунок 12

9. В класс StrategyExample необходимо добавить метод main, в котором следует написать тестовое приложение для паттерна «Стратегия» (рисунок 13).

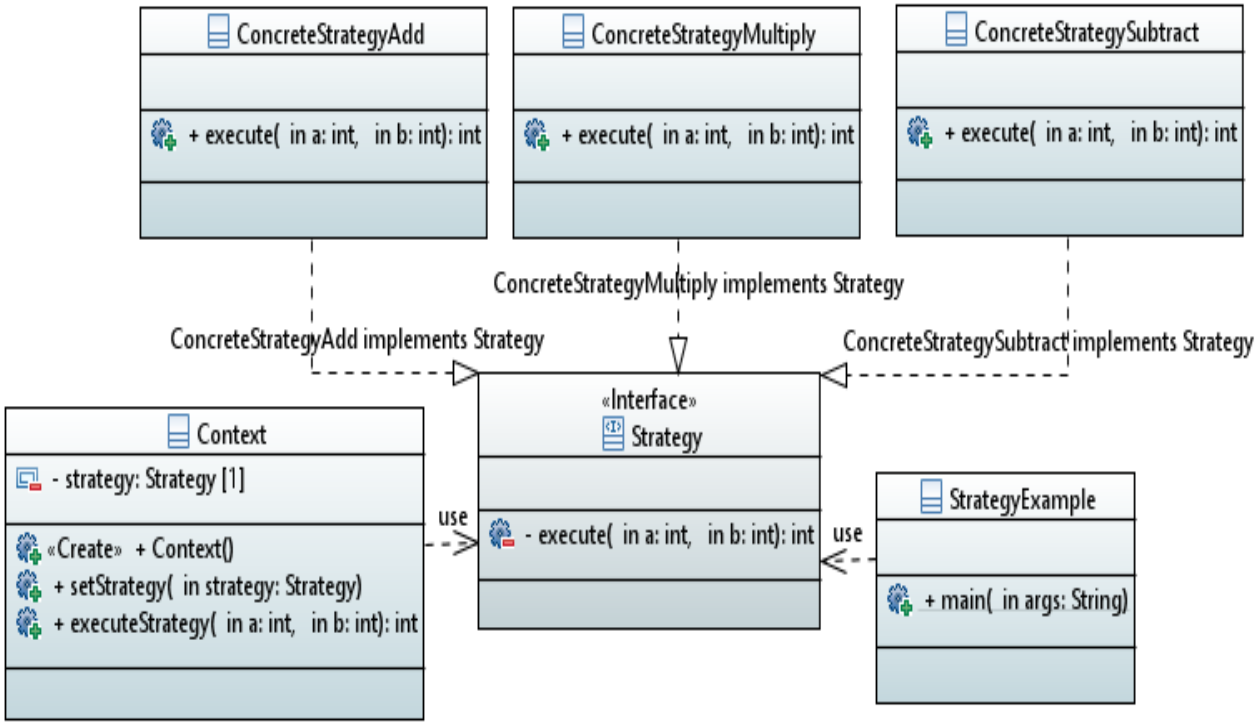

Рисунок 13

Класс тестового приложения будет иметь следующий вид (рисунок 14):

StrategyExample.java | Context.java | Strategy.java | ConcreteS

```
1  package strategyPack;
2
3  class StrategyExample {
4
5      public static void main(String[] args) {
6          Context context = new Context();
7
8          context.setStrategy(new ConcreteStrategyAdd());
9          int resultA = context.executeStrategy(3,4);
10         System.out.println("Result A : " + resultA );
11
12         context.setStrategy(new ConcreteStrategySubtract());
13         int resultB = context.executeStrategy(3,4);
14         System.out.println("Result B : " + resultB );
15
16         context.setStrategy(new ConcreteStrategyMultiply());
17         int resultC = context.executeStrategy(3,4);
18         System.out.println("Result C : " + resultC );
19     }
20
21 }
22
```

Рисунок 14

10. Результат выполнения тестового приложения (рисунок 15).

StrategyExample.java | Context.java | Strategy.java

```
1  package strategyPack;
2
3  class StrategyExample {
4
5      public static void main(String[] args) {
6          Context context = new Context();
7
8          context.setStrategy(new ConcreteStrategyAd
9          int resultA = context.executeStrategy(3,4)
10         System.out.println("Result A : " + resultA
11
12         context.setStrategy(new ConcreteStrategySu
13         int resultB = context.executeStrategy(3,4)
14         System.out.println("Result B : " + resultB
15
16         context.setStrategy(new ConcreteStrategyMu
17         int resultC = context.executeStrategy(3,4)
18         System.out.println("Result C : " + resultC
19     }
20
21 }
22
```

Problems | Javadoc | Declarat

```
<terminated> StrategyExample [Java App
Реализация метода сложения
Result A : 7
Реализация метода вычитания
Result B : -1
Реализация метода умножения
Result C : 12
```

Рисунок 15

4 Содержание отчета по практической работе

Отчет по практической работе включает:

- титульный лист;
- условие задания;
- диаграммы классов для решения задачи;
- диаграммы последовательности для решения задачи;
- текст программы реализации паттерна проектирования «Стратегия» для индивидуального задания;
- результаты тестирования программы.

5 Вопросы к защите индивидуальной работы

1. Что такое паттерн проектирования?
2. Для чего предназначен паттерн проектирования «Стратегия»?
3. К какому типу паттернов проектирования относится «Стратегия»?
4. Какие классы/интерфейсы являются участниками «Стратегии»?
5. Для чего необходим метод setStrategy в классе Context?
6. Назовите родственные для «Стратегии» паттерны проектирования.
7. Выделите достоинства и недостатки этого паттерна проектирования.

6 Индивидуальные задания

При разработке использовать поведенческий паттерн проектирования «Стратегия».

1. Реализовать программный продукт, выполняющий проверку введенной пользователем строки – что введено: число, строка в нижнем регистре или строка в верхнем регистре?
2. Реализовать программный продукт, выполняющий проверку введенного пользователем числа – что введено: однозначное число, двузначное число или число с большим количеством знаков?
3. Реализовать программный продукт, выполняющий подсчет заработной платы сотрудника за месяц в зависимости от типа его занятости – полная рабочая неделя (40 часов), сокращенная рабочая неделя для лиц от 16 до 18 лет (36 часов) и сокращенная рабочая неделя для лиц младше 16 лет (24 часа). Стоимость часа работы выбрать самостоятельно.
4. На сайте есть группа пользователей, каждый из них принадлежит к определённой группе, например, это администратор,

редактор и гость. Администраторы могут добавлять материал, удалять материал, редактировать и читать. Редакторы могут только редактировать и читать. А гости только читать. Реализовать программный продукт, демонстрирующий возможности каждой из этих групп пользователей.

5. Есть три типа героев – король, воин и маг. У каждого своя атака, тип оружия и наносимый урон. Реализовать программный продукт, демонстрирующий возможности каждого из этих типов героя.

6. Существуют различные легковые машины, которые используют разные источники энергии: электричество, бензин, газ. Есть гибридные автомобили. Каждый из типов имеет свой расход топлива. Реализовать программный продукт, демонстрирующий возможности каждого из типов автомобиля.

7. Реализовать программный продукт, демонстрирующий возможности компрессии файлов, для одного из доступных алгоритмов: zip, arj или rar.

8. Существуют различные грузовые машины, которые используют разные источники энергии: электричество, бензин, газ. Есть гибридные автомобили. Каждый из типов имеет свой расход топлива. Реализовать программный продукт, демонстрирующий возможности каждого из типов автомобиля.

9. На городской автобазе имеется автотранспорт для перевозки пассажиров. Автотранспорт использует разные источники энергии: электричество, бензин, газ. Есть гибридные автобусы. Каждый из типов имеет свой расход топлива. Реализовать программный продукт, демонстрирующий возможности каждого из типов автомобиля.

10. Есть три типа героев – король, командир и солдат. Герой ходит, либо бегает, но, возможно, в будущем он также сможет рыть под землей. Реализовать программный продукт, демонстрирующий возможности каждого из этих типов героя.

11. Есть три типа героев – король, воин и маг. Герой ходит, либо бегает, но, возможно, в будущем он также сможет летать. Реализовать программный продукт, демонстрирующий возможности каждого из этих типов героя.

12. Есть три типа героев – король, воин и маг. Герой ходит, либо бегает, но, возможно, в будущем он также сможет плавать. Реализовать программный продукт, демонстрирующий возможности каждого из этих типов героя.

13. Есть три типа героев – король, воин и маг. Герой ходит, либо бегает, но, возможно, в будущем он также сможет телепортироваться.

Реализовать программный продукт, демонстрирующий возможности каждого из этих типов героя.

14. Прокси-система накапливает объекты от разных источников данных и формирует очередь из разнородных объектов (queue processing and saving data). Тогда после извлечения из очереди объекта и последующее его сохранение определяется стратегией выбора, на основе свойств этого объекта. Реализовать программный продукт, демонстрирующий алгоритмы обработки объектов.

15. Шифрование (encrypting): для небольших файлов может использовать стратегию «в памяти», когда весь файл читается и хранится в памяти (скажем, для файлов <1 ГБ). Для больших файлов можно использовать другую стратегию, где части файла читаются в памяти, а частично зашифрованные результаты хранятся в файлах tmp. Это могут быть две разные стратегии для одной и той же задачи. Реализовать программный продукт, демонстрирующий использование алгоритмов шифрования файлов.

Список использованных источников

1. Ларман, К. Применение UML 2.0 и шаблонов проектирования/ К. Ларман. - М.: Издательский дом «Вильямс», 2013. -736 с. - Текст: непосредственный

2. Османи, Э. Паттерны для масштабируемых JavaScript-приложений/ Э. Османи. - М.: Техносфера, 2015. -188 с. - Текст: непосредственный

3. Фримен Э. Паттерны проектирования/ Э. Фримен, Э. Фримен, К. Сьерра, Б. Бейтс. – СПб.: Питер, 2016. - 653 с. - Текст: непосредственный

4. Перл, И. А. Введение в методологию программной инженерии : учебное пособие: [16+]/ И. А. Перл, О. В. Калёнова. – Санкт-Петербург : Университет ИТМО, 2019. – 53 с. : ил., схем. – Режим доступа: по подписке. – URL: https://biblioclub.ru/index.php?page=book&id=566776 (дата обращения: 28.08.2025). – Бибиогр. в кн. – Текст: электронный.

5. Шуваев, А. В. Программная инженерия: учебное пособие для магистрантов направления подготовки 09.04.02 – Информационные системы и технологии: [16+]/ А. В. Шуваев; Ставропольский государственный аграрный университет, Кафедра информационных систем. – Ставрополь: Ветеран, 2020. – 84 с. : ил., табл. – Режим доступа: по подписке. – URL: https://biblioclub.ru/index.php?page=book&id=700960 (дата обращения: 28.08.2025). – Библиогр. в кн. – Текст: электронный.

МИНОБРНАУКИРОССИИ

Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Юго-Западный государственный университет»
(ЮЗГУ)

Кафедра программной инженерии

УТВЕРЖДАЮ
Проректор по учебной работе
_____О.Г. Локтионова
«_4_»_09_____ 2025 г.

ПАТТЕРН ПРОЕКТИРОВАНИЯ «КОМПОНОВЩИК»

Методические указания по выполнению практической работы по дисциплине "Методология программной инженерии" для студентов направления подготовки магистров 09.04.04 "Программная инженерия"

Курск 2025

УДК 004.65

Составители: Т.М. Белова

Рецензент
Кандидат технических наук, доцент кафедры вычислительной техники ЮЗГУ Т.И. Лапина

Паттерн проектирования «Абстрактная фабрика»: методические указания по выполнению практической работы по дисциплине "Методология программной инженерии" для студентов направления подготовки магистров 09.04.04 "Программная инженерия"/ Юго-Зап. гос. ун-т; сост.: Т.М. Белова – Курск, 2025. – 28 с.: ил. 34.

Изложена последовательность действий по разработке и применению паттерна проектирования «**Абстрактная фабрика**».

Материал предназначен для студентов направления подготовки магистров 09.04.04 «Программная инженерия», а также будет полезен студентам всех направлений подготовки, изучающим технологии разработки программных продуктов с использованием паттернов.

Текст печатается в авторской редакции.

Подписано в печать . Формат 60x84 1/16.
Усл. печ. л. 1,6. Уч.-изд. л. 1,5. Тираж 50 экз. Заказ . Бесплатно.
Юго-Западный государственный университет
305040, Курск, ул.50 лет Октября, 94.

Содержание

1 Цель практической работы

Целью практической работы является приобретение знаний, умений и навыков для использования паттерна проектирования «Абстрактная фабрика» в проектировании информационно-вычислительных систем.

2 Основные понятия

При реализации проектов по разработке программных систем и моделированию бизнес-процессов встречаются ситуации, когда решение проблем в различных проектах имеют сходные структурные черты. Попытки выявить похожие схемы или структуры в рамках объектно-ориентированного анализа и проектирования привели к появлению понятия паттерна, которое из абстрактной категории превратилось в непременный атрибут современных CASE-средств.

Паттерны различаются степенью детализации и уровнем абстракции. Предлагается следующая общая классификация паттернов по категориям их применения:

- архитектурные паттерны,
- паттерны проектирования,
- паттерны анализа,
- паттерны тестирования,
- паттерны реализации.

Архитектурные паттерны (Architectural patterns) - множество предварительно определенных подсистем со спецификацией их ответственности, правил и базовых принципов установления отношений между ними.

Архитектурные паттерны предназначены для спецификации фундаментальных схем структуризации программных систем. Наиболее известными паттернами этой категории являются паттерны GRASP (General Responsibility Assignment Software Pattern). Эти паттерны относятся к уровню системы и подсистем, но не к уровню классов. Как правило, формулируются в обобщенной форме, используют обычную терминологию и не зависят от области приложения. Паттерны этой категории систематизировал и описал К. Ларман.

Паттерны проектирования (Design patterns) - специальные схемы для уточнения структуры подсистем или компонентов программной системы и отношений между ними.

Паттерны проектирования описывают общую структуру взаимодействия элементов программной системы, которые реализуют

исходную проблему проектирования в конкретном контексте. Наиболее известными паттернами этой категории являются паттерны GoF (Gang of Four), названные в честь Э. Гаммы, Р. Хелма, Р. Джонсона и Дж. Влиссидеса, которые систематизировали их и представили общее описание. Паттерны GoF включают в себя 23 паттерна. Эти паттерны не зависят от языка реализации, но их реализация зависит от области приложения.

Паттерны анализа (Analysis patterns) - специальные схемы для представления общей организации процесса моделирования.

Паттерны анализа относятся к одной или нескольким предметным областям и описываются в терминах предметной области. Наиболее известными паттернами этой группы являются паттерны бизнес-моделирования ARIS (Architecture of Integrated Information Systems), которые характеризуют абстрактный уровень представления бизнес-процессов. Паттерны анализа конкретизируются в типовых моделях с целью выполнения аналитических оценок или имитационного моделирования бизнес-процессов.

Паттерны тестирования (Test patterns) - специальные схемы для представления общей организации процесса тестирования программных систем.

К этой категории паттернов относятся такие паттерны, как тестирование черного ящика, белого ящика, отдельных классов, системы. Паттерны этой категории систематизировал и описал М. Гранд. Некоторые из них реализованы в инструментальных средствах, наиболее известными из которых является IBM Test Studio. В связи с этим паттерны тестирования иногда называют стратегиями или схемами тестирования.

Паттерны реализации (Implementation patterns) - совокупность компонентов и других элементов реализации, используемых в структуре модели при написании программного кода.

Эта категория паттернов делится на следующие подкатегории: паттерны организации программного кода, паттерны оптимизации программного кода, паттерны устойчивости кода, паттерны разработки графического интерфейса пользователя и др. Паттерны этой категории описаны в работах М. Гранда, К. Бека, Дж. Тидвелла и др. Некоторые из них реализованы в популярных интегрированных средах программирования в форме шаблонов создаваемых проектов. В этом случае выбор шаблона программного приложения позволяет получить некоторую заготовку программного кода.

Шаблон проектирования или паттерн (англ. design pattern) в разработке программного обеспечения — повторяемая архитектурная конструкция, представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста.

Технически, паттерны (шаблоны) проектирования - это абстрактные примеры правильного использования небольшого числа комбинаций простейших техник объектно-ориентированного программирования. Паттерны проектирования - это простые примеры, показывающие правильные способы организации взаимодействий между классами или объектами.

Порождающие паттерны (Creational) — это паттерны, которые абстрагируют процесс порождения классов и объектов. Они позволяют сделать систему независимой от способа создания, композиции и представления объектов. Шаблон, порождающий классы, использует наследование, чтобы изменять порождаемый класс, а шаблон, порождающий объекты, делегирует порождение другому объекту.

Среди них выделяются следующие:

- Абстрактная фабрика (Abstract Factory),
- Строитель (Builder),
- Фабричный метод (Factory Method),
- Прототип (Prototype),
- Одиночка (Singleton).

Абстрактная фабрика (Abstract factory) — предоставляет интерфейс для создания объектов, конкретные классы которых неизвестны.

Строитель (Builder) — представляет класс, который представляет собой интерфейс для создания сложного объекта. Отделяет конструирование объекта от представления, позволяя использовать один процесс конструирования для различных представлений.

Фабричный метод (Factory method) — определяет интерфейс для создания объекта, но оставляет подклассам решение о том, какой класс порождать.

Прототип (Prototype) — определяет интерфейс создания объекта через клонирование другого объекта вместо создания через конструктор.

Одиночка (Singleton) — гарантирует, что некоторый класс может иметь только один экземпляр.

«Абстрактная фабрика» является порождающим шаблоном проектирования, который предоставляет интерфейс для создания семейств взаимосвязанных или взаимозависимых объектов без указания их конкретных классов.

Преимущества от применения паттернов проектирования заключаются в следующем:

- Паттерны позволяют суммировать опыт экспертов и сделать его доступным рядовым разработчикам.
- Имена паттернов образуют своего рода словарь, который позволяет разработчикам лучше понимать друг друга.
- Если в документации системы указано, какие паттерны в ней используются, это позволяет читателю быстрее понять систему.
- Паттерны упрощают реструктуризацию системы независимо от того, использовались ли паттерны при ее проектировании.
- Паттерн дает название проблеме и определяет способы решения многих проблем за счет готового набора абстракций.
- Использование шаблонов проектирования аналогично использованию готовых библиотек кода.
- Правильное использование шаблонов помогает разработчикам определить нужный вектор развития и уйти от многих проблем, которые могут возникнуть в процессе разработки.
- Паттерны проектирования не зависят от языка программирования.

Правильно выбранные паттерны проектирования позволяют сделать программную систему более гибкой, ее легче поддерживать и модифицировать, а код такой системы в большей степени соответствует концепции повторного использования.

Проблемы, которые порождают шаблоны проектирования:

- потеря гибкости проектирования и разработки системы;
- использование шаблонов усложняет систему;
- слепое следование определенному шаблону и повсеместное его использование может породить много архитектурных и логических проблем.

3 Порядок выполнения практической работы

1. В интегрированной среде разработки Eclipse необходимо создать Java-проект AbstractFactory для разработки паттерна проектирования «Абстрактная фабрика» (рисунки 1–2).

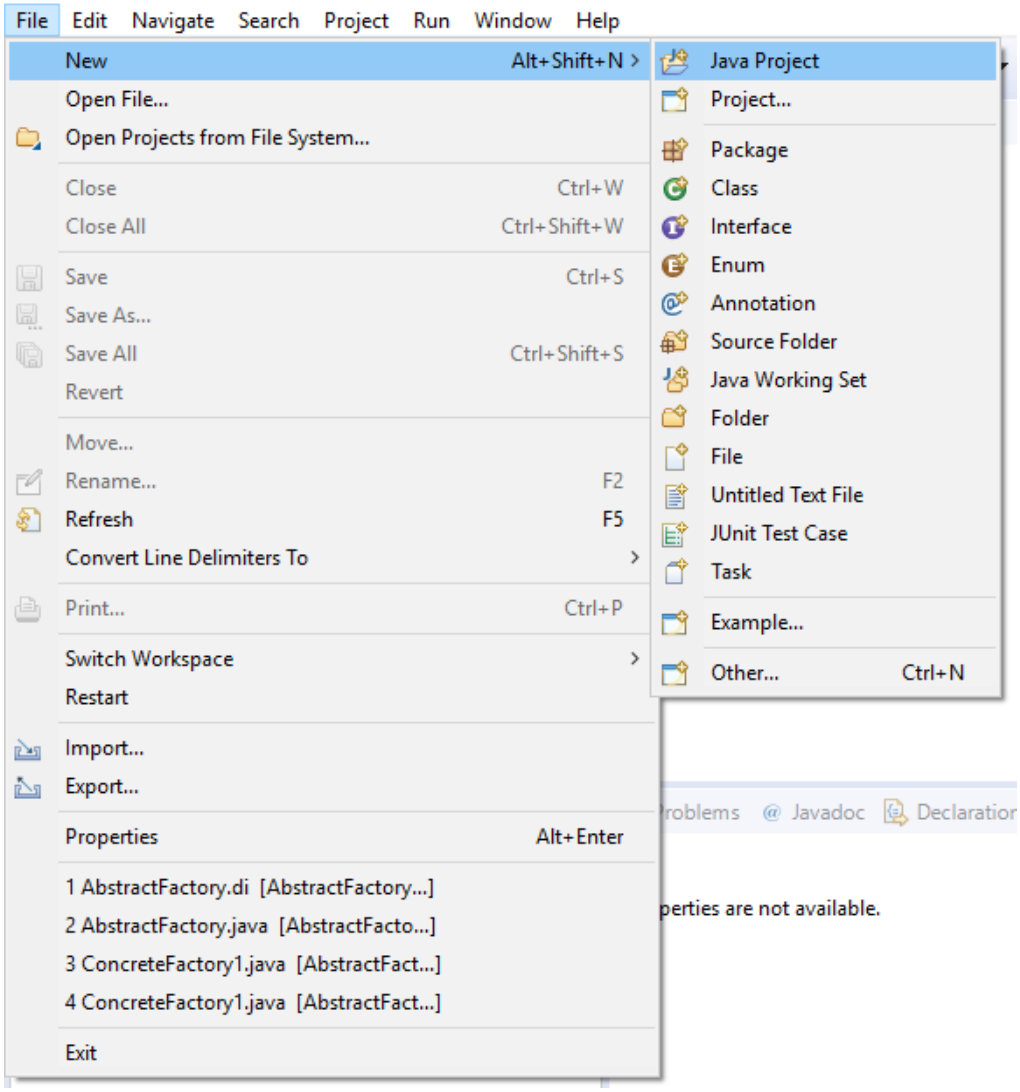

Рисунок 1

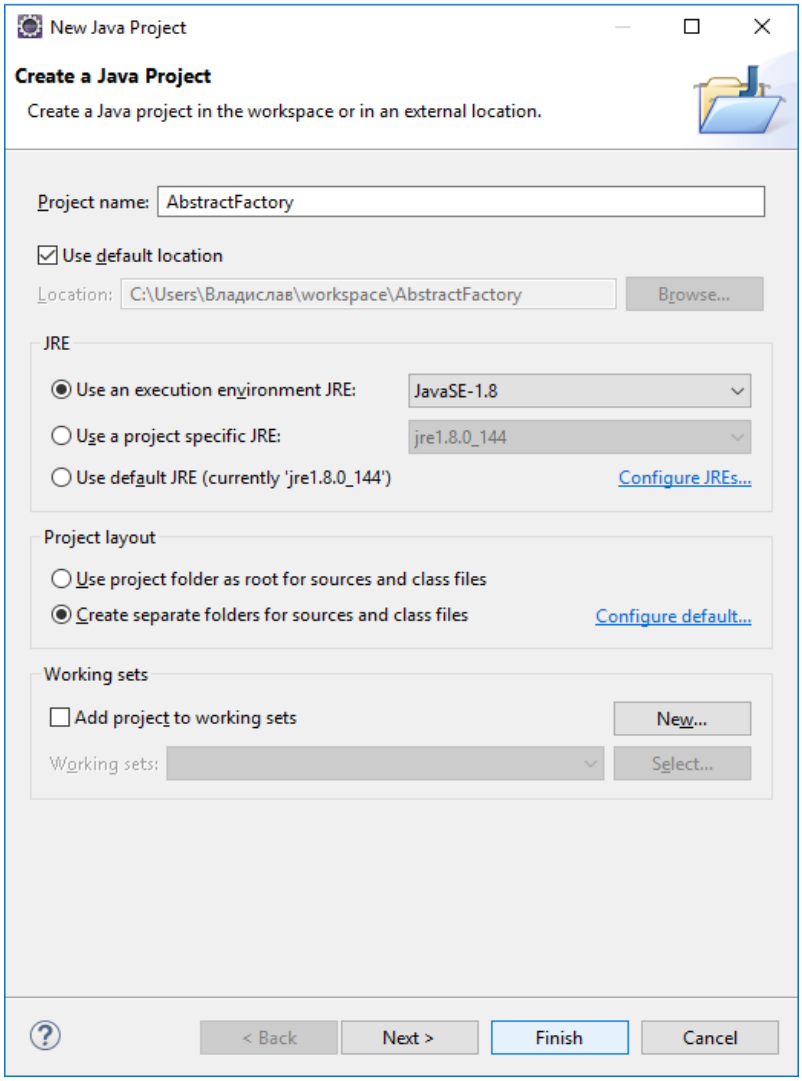

Рисунок 2

2. В проекте необходимо создать папку Model для проектирования диаграммы классов (рисунки 3–4).

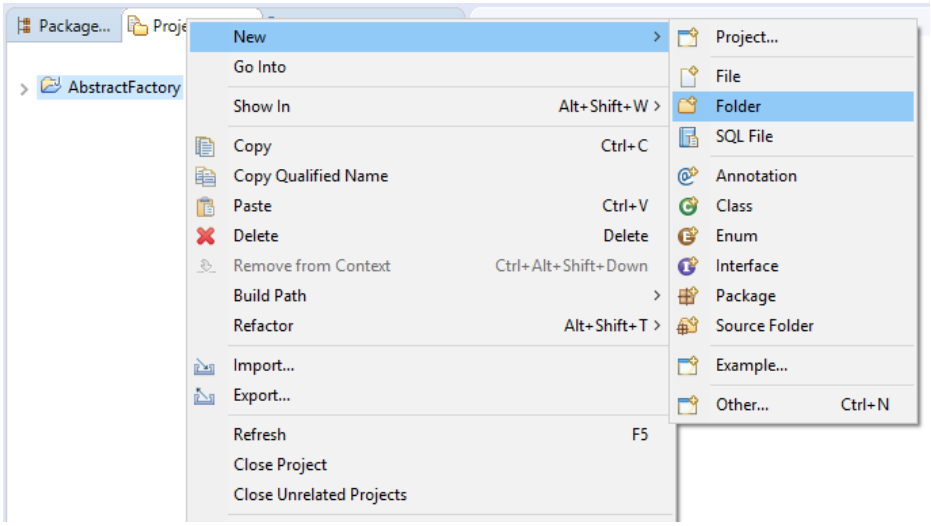

Рисунок 3

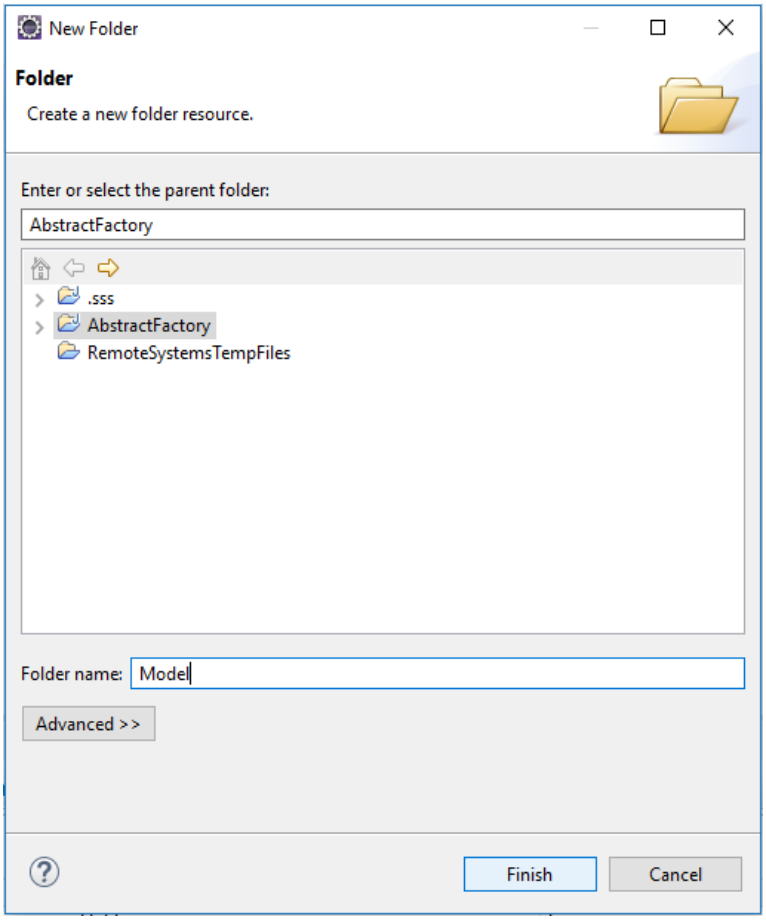

Рисунок 4

3. В созданной папке необходимо добавить файлы для работы с инструментом проектирования UML-диаграмм Papyrus (рисунки 5–10).

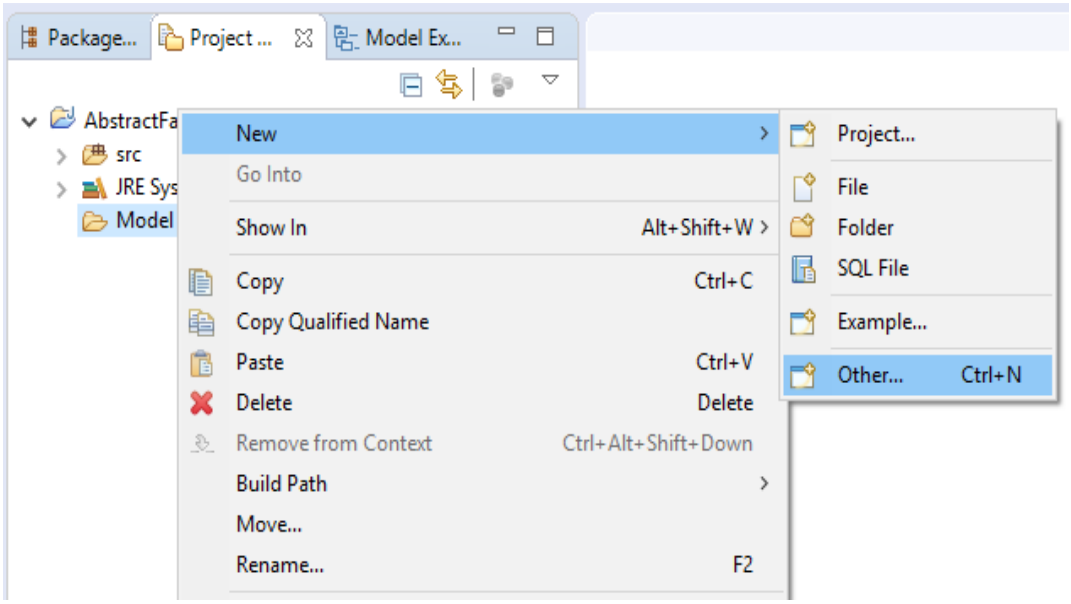

Рисунок 5

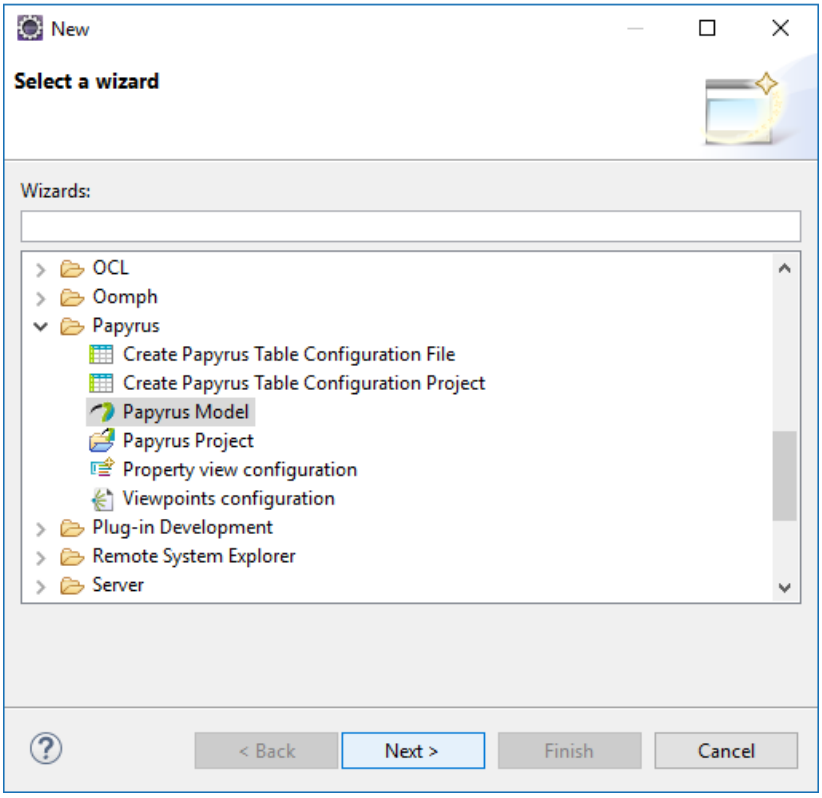

Рисунок 6

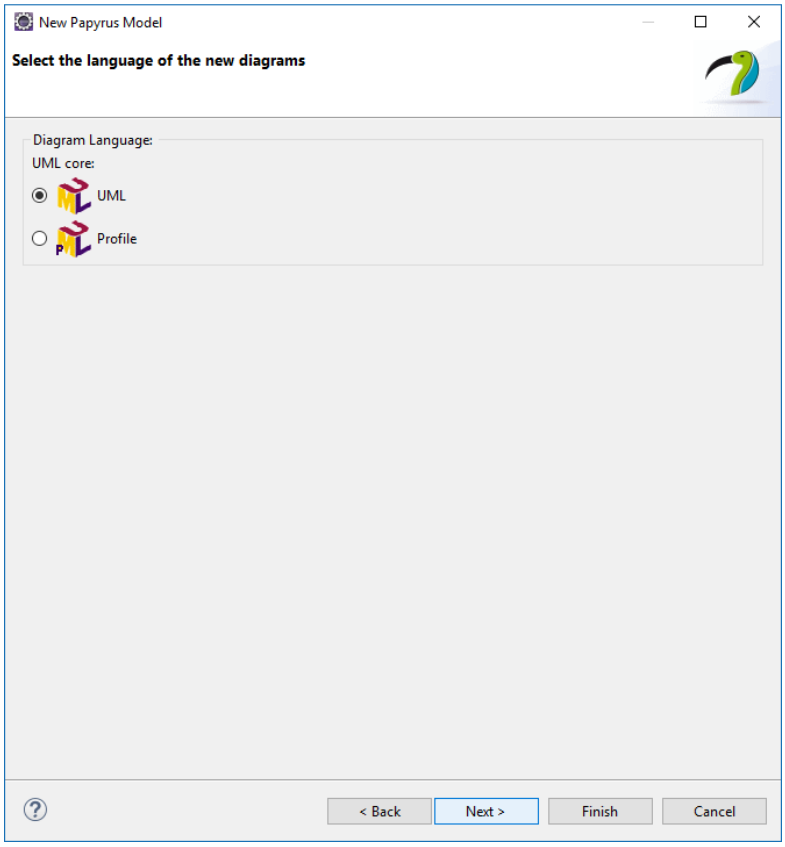

Рисунок 7

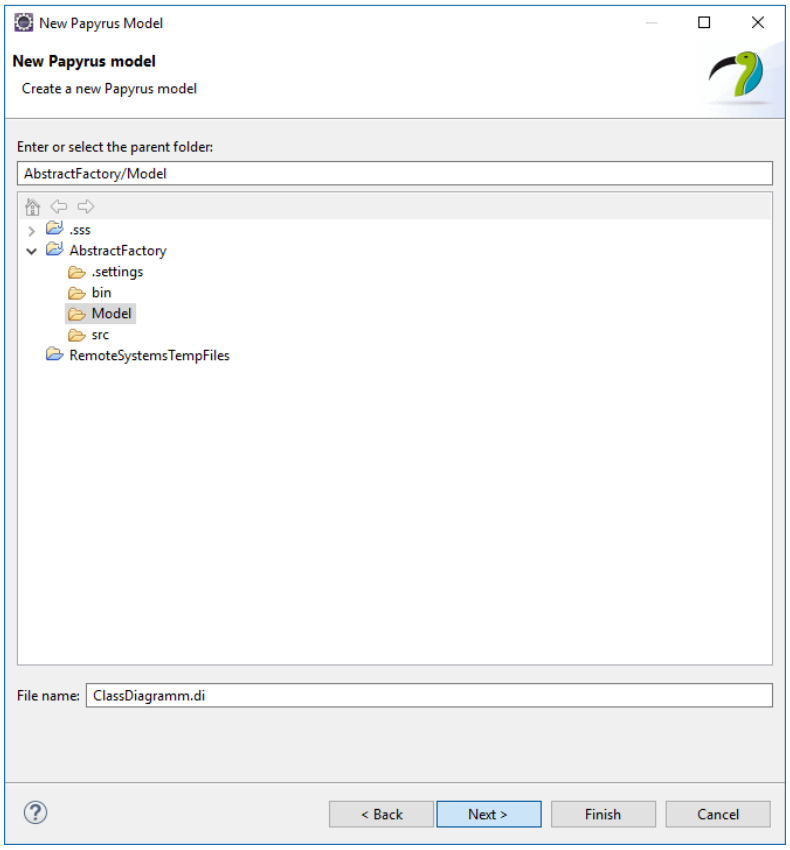

Рисунок 8

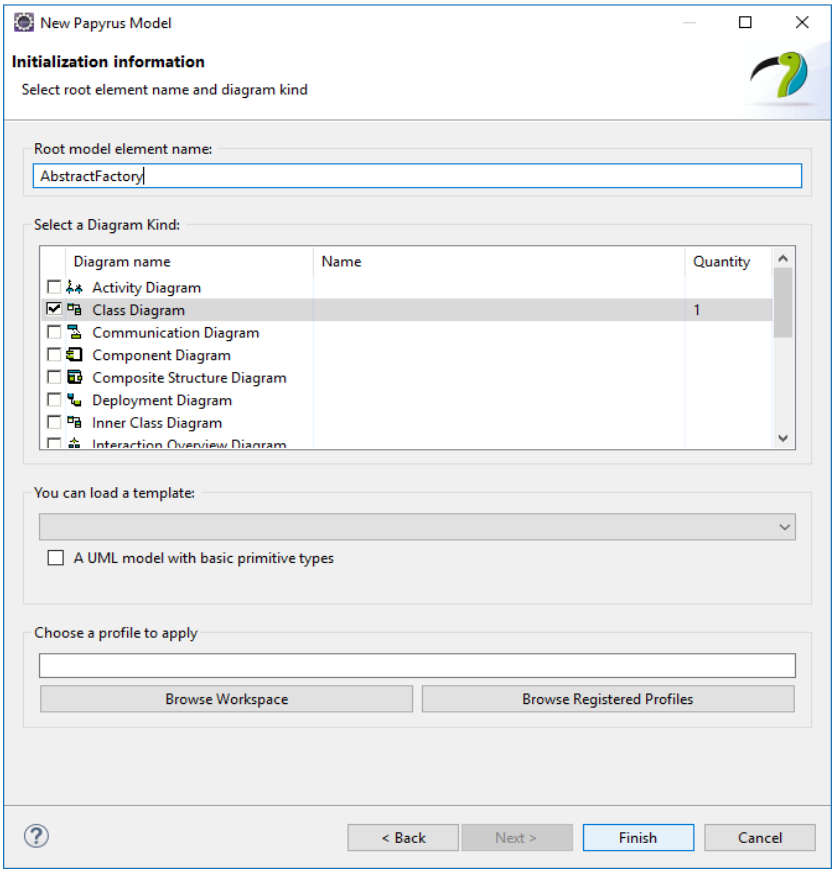

Рисунок 9

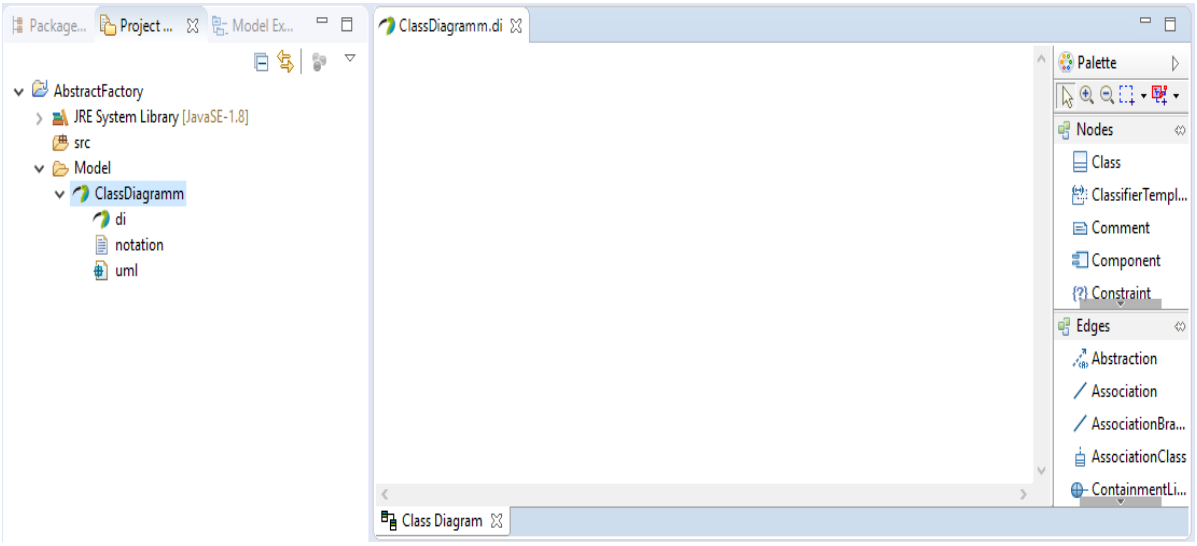

Рисунок 10

4. Необходимо разработать в проектировщике UML-диаграмм Papyrus диаграмму классов паттерна проектирования «Абстрактная фабрика» для конкретной задачи (рисунок 12) по схеме, представленной на рисунке 11.

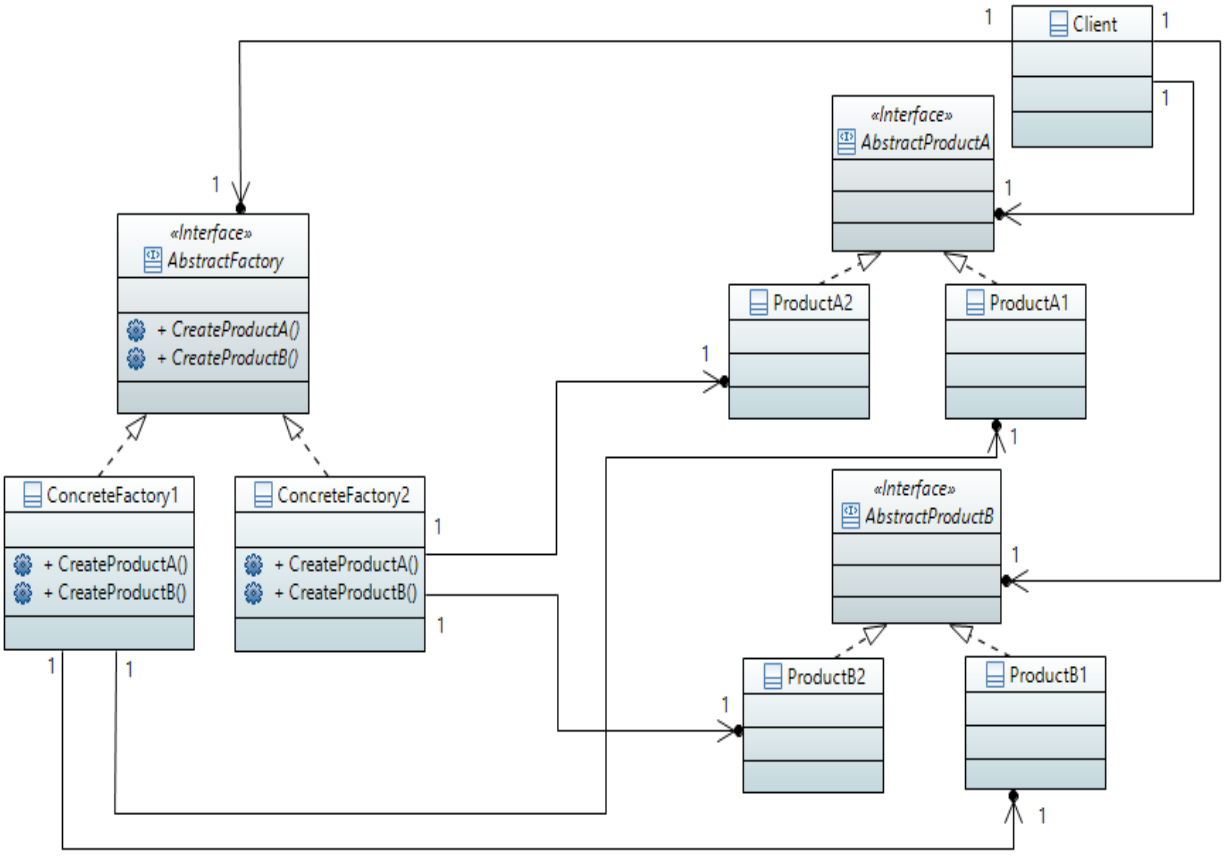

Рисунок 11

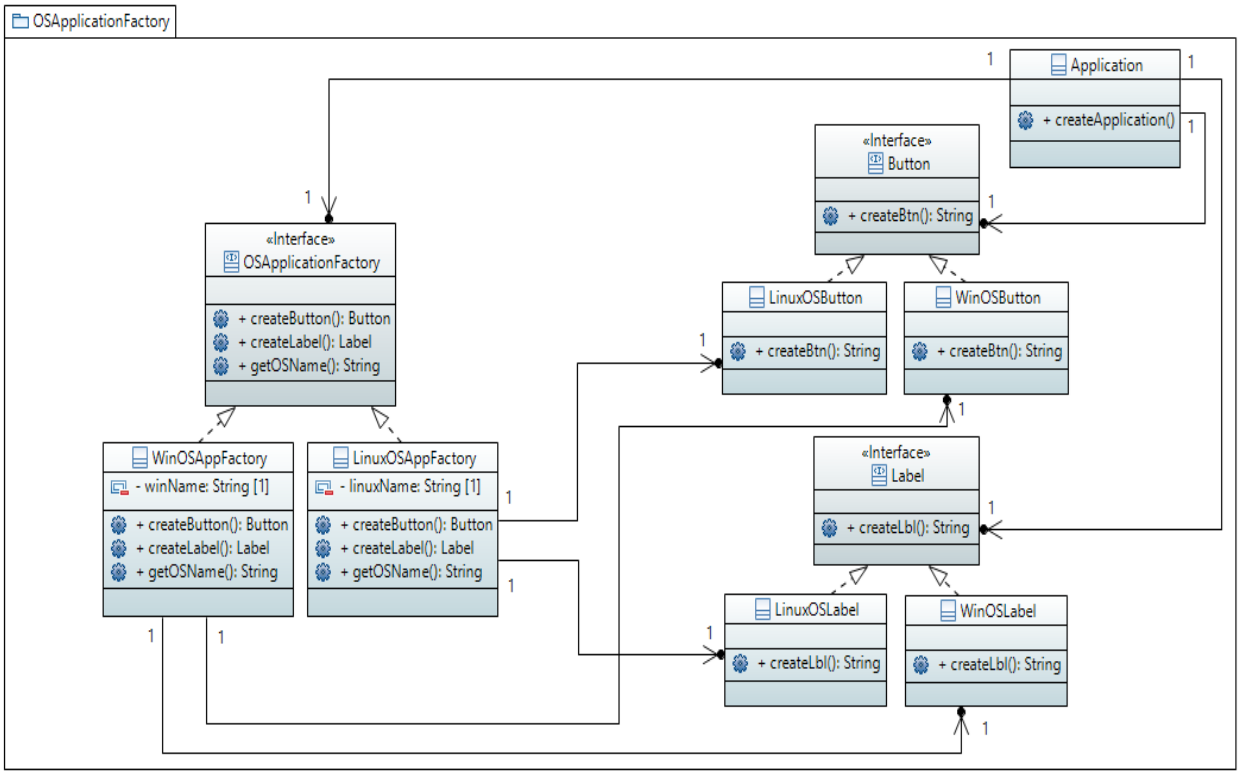

Рисунок 12

5. Полученную диаграмму классов необходимо сгенерировать в Java-код. Сгенерированные файлы объектов по диаграммам классов можно увидеть в папке src проекта (рисунок 13).

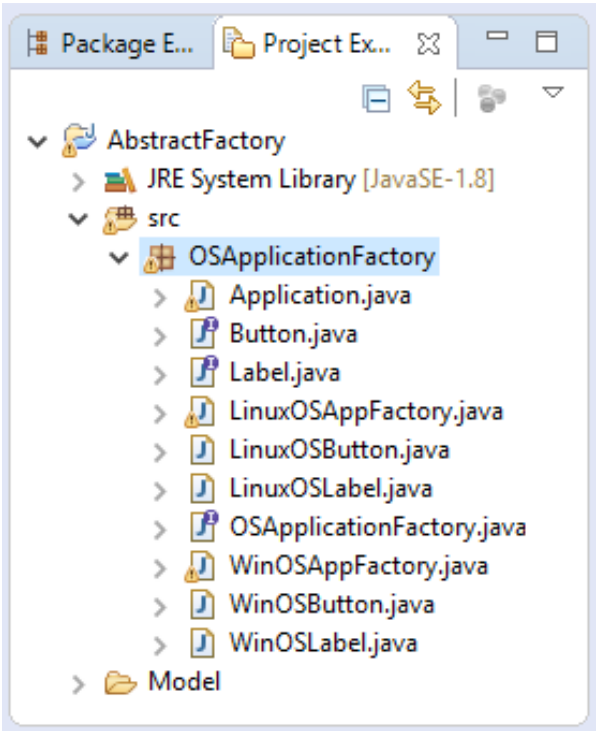

Рисунок 13

6. Необходима корректировка сгенерированного кода классов LinuxOSButton и WinOSButton (рисунки 14–15), являющихся наследниками интерфейса Button (рисунок 16). Интерфейс Button играет роль первого продукта, производимого конкретной фабрикой. Скорректированный код представлен на рисунках 17–18.

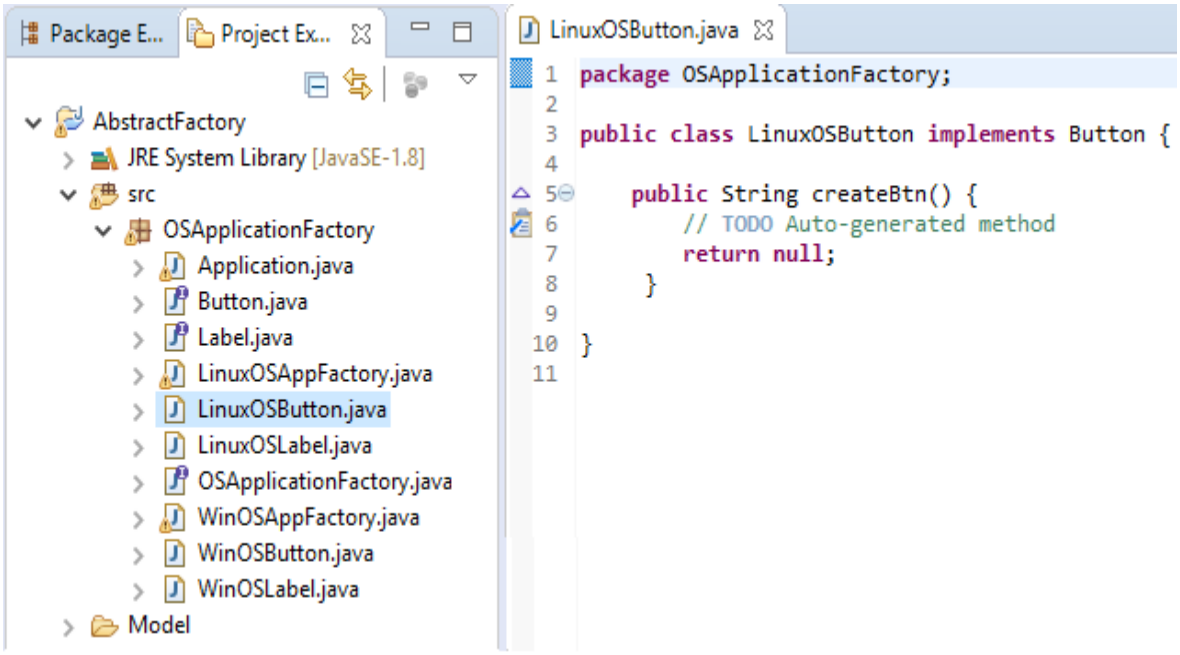

Рисунок 14

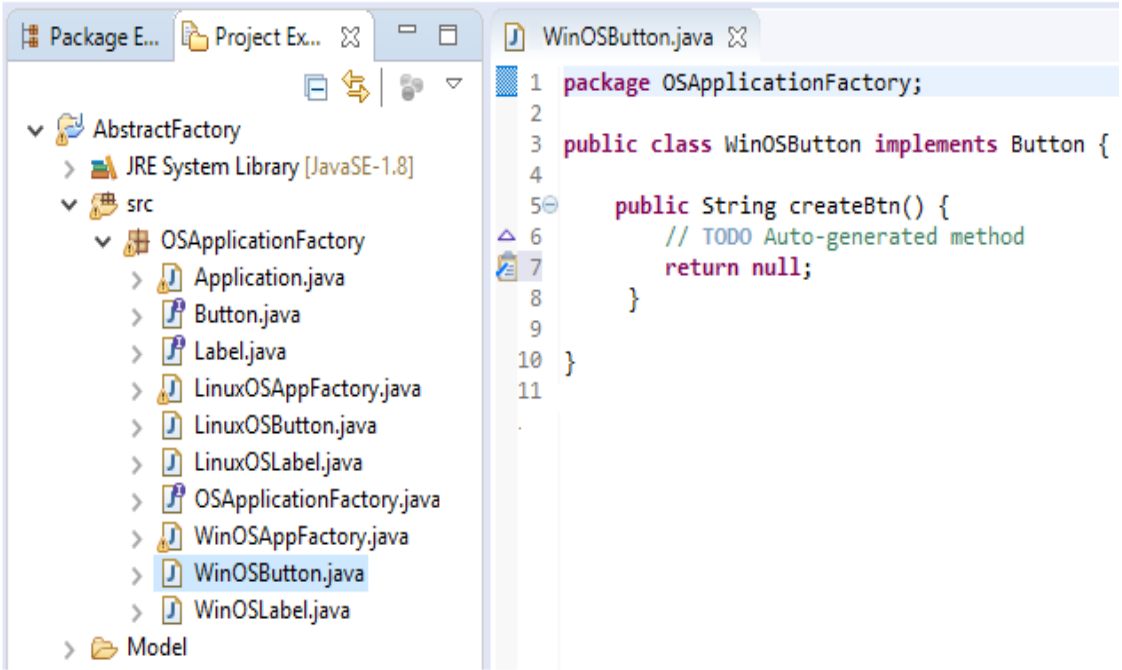

Рисунок 15

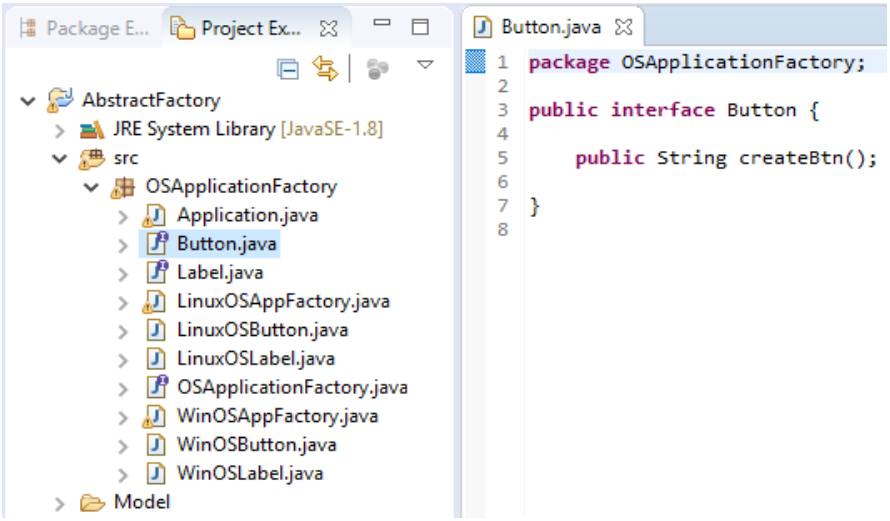

Рисунок 16

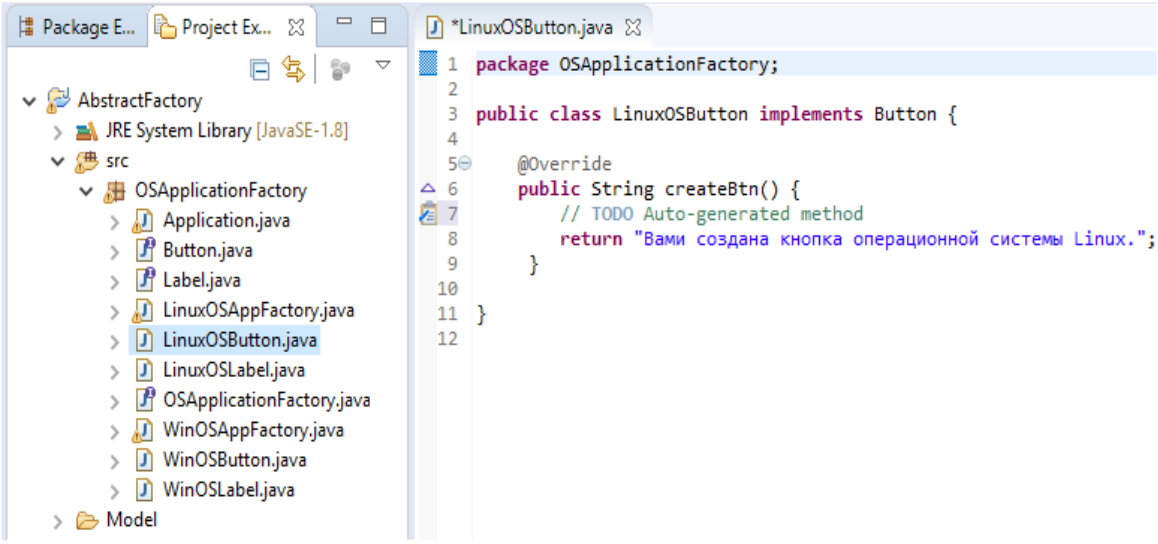

Рисунок 17

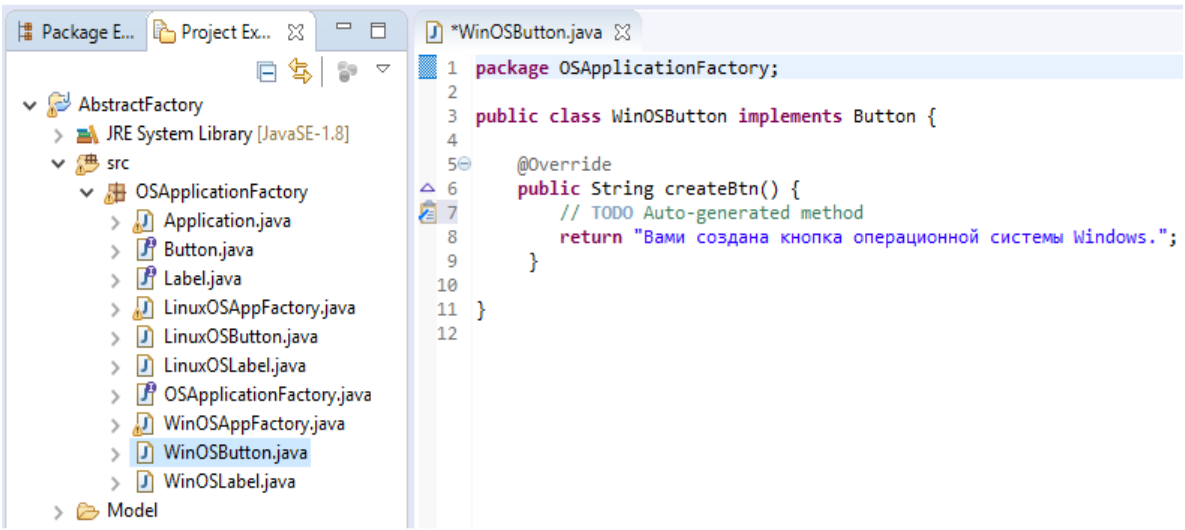

Рисунок 18

7. Необходима корректировка сгенерированного кода классов LinuxOSLabel и WinOSLabel (рисунки 19–20), являющихся наследниками интерфейса Label (рисунок 21). Интерфейс Label играет роль второго продукта, производимого конкретной фабрикой. Скорректированный код представлен на рисунках 22–23.

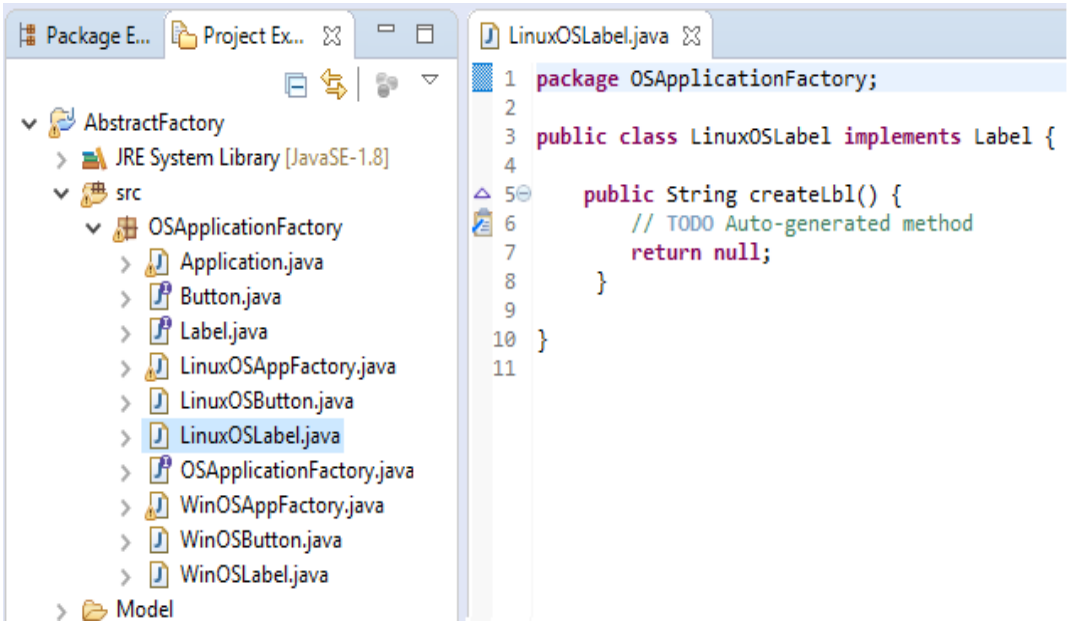

Рисунок 19

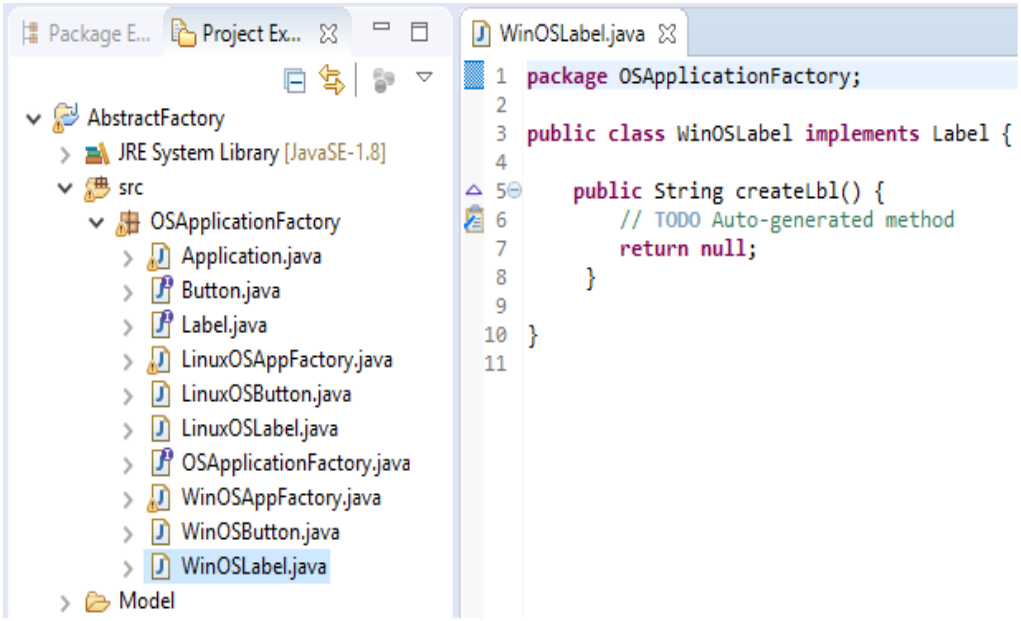

Рисунок 20

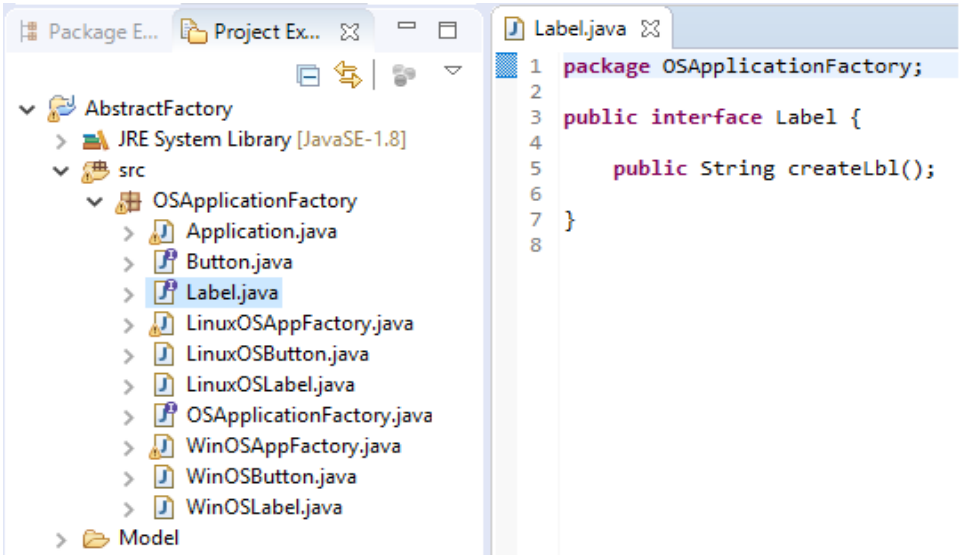

Рисунок 21

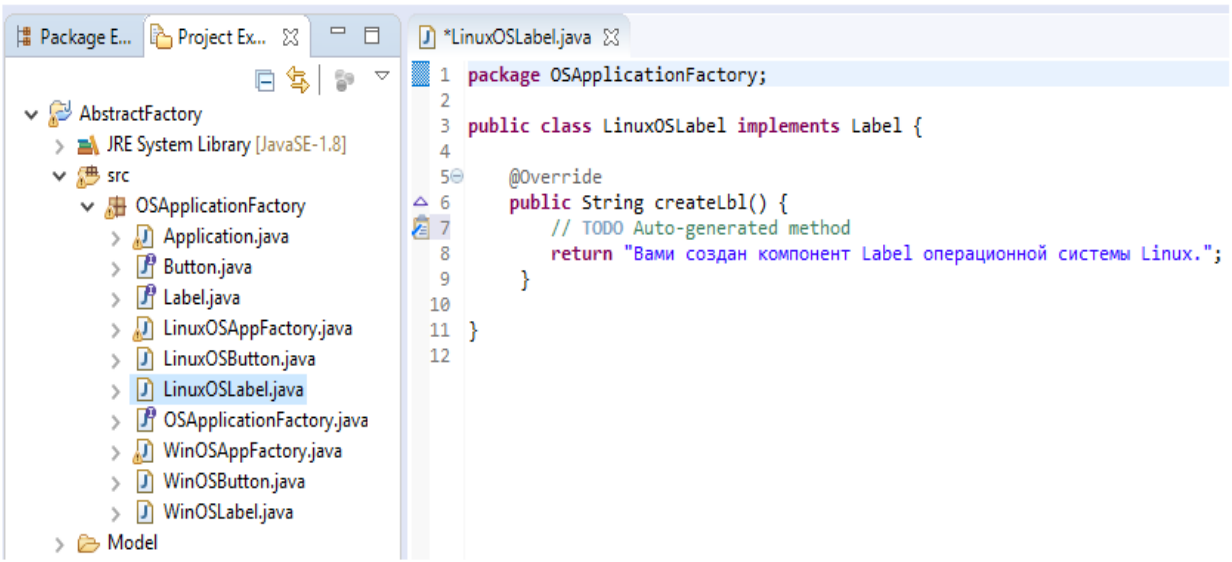

Рисунок 22

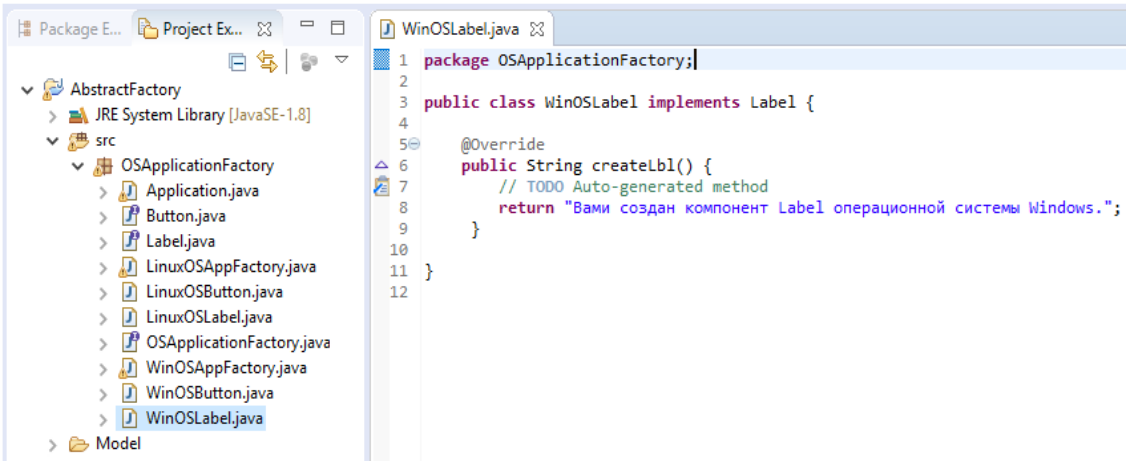

Рисунок 23

8. Необходима корректировка сгенерированного кода классов LinuxOSAppFactory и WinOSAppFactory (рисунки 24–25), являющихся наследниками интерфейса OSApplicationFactory (рисунок 26). Интерфейс OSApplicationFactory реализуется всеми конкретными фабриками и состоит из методов создания продуктов Button и Label. Скорректированный код представлен на рисунках 27–28.

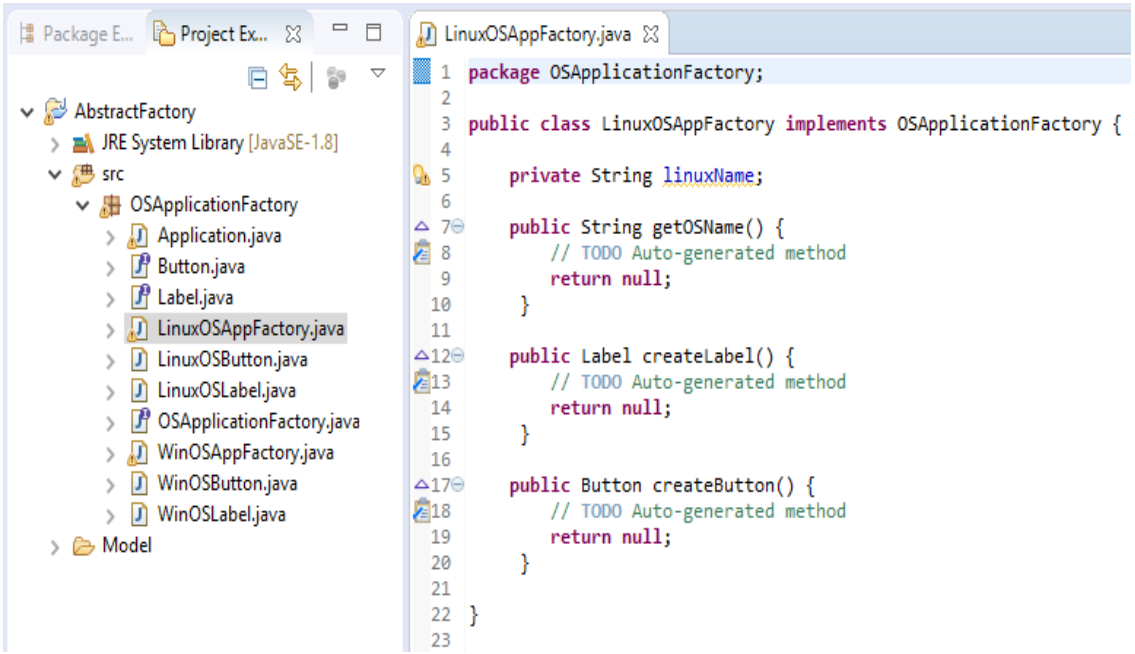

Рисунок 24

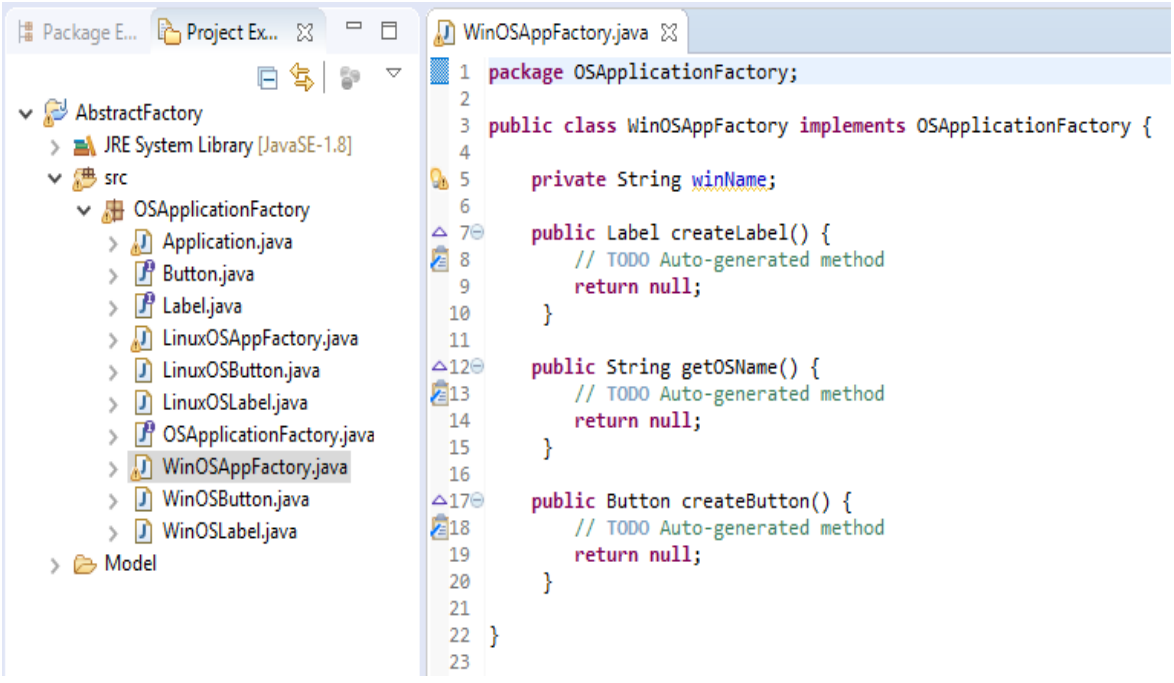

Рисунок 25

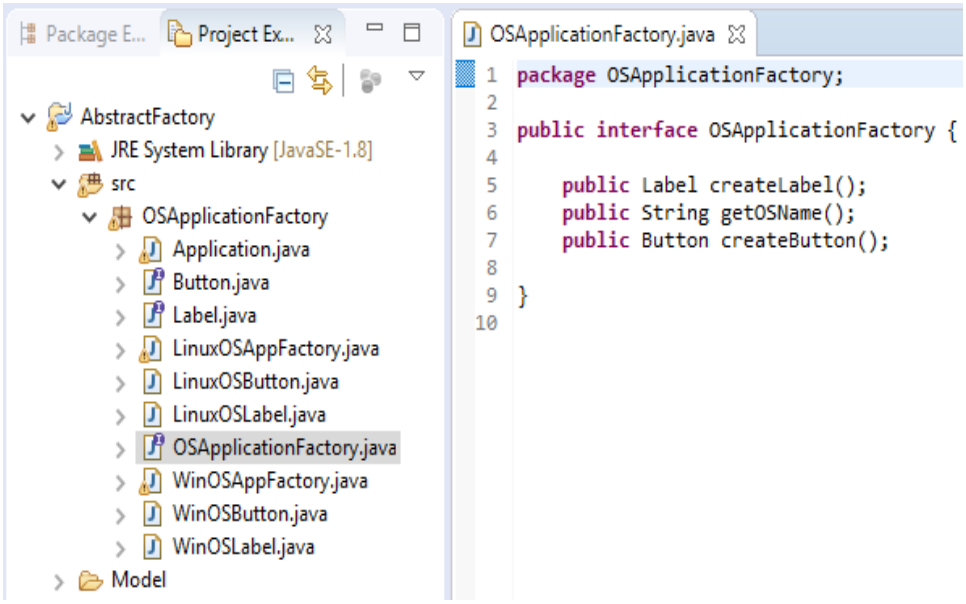

Рисунок 26

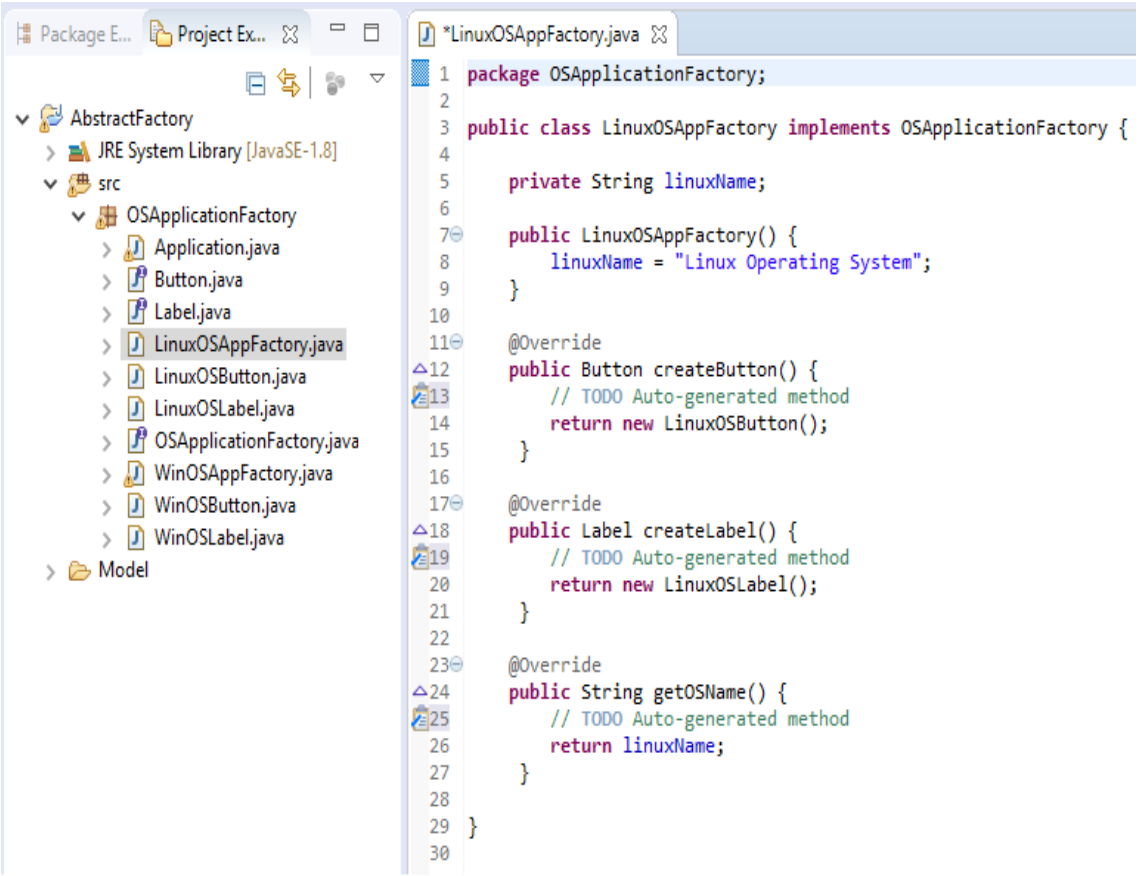

Рисунок 27

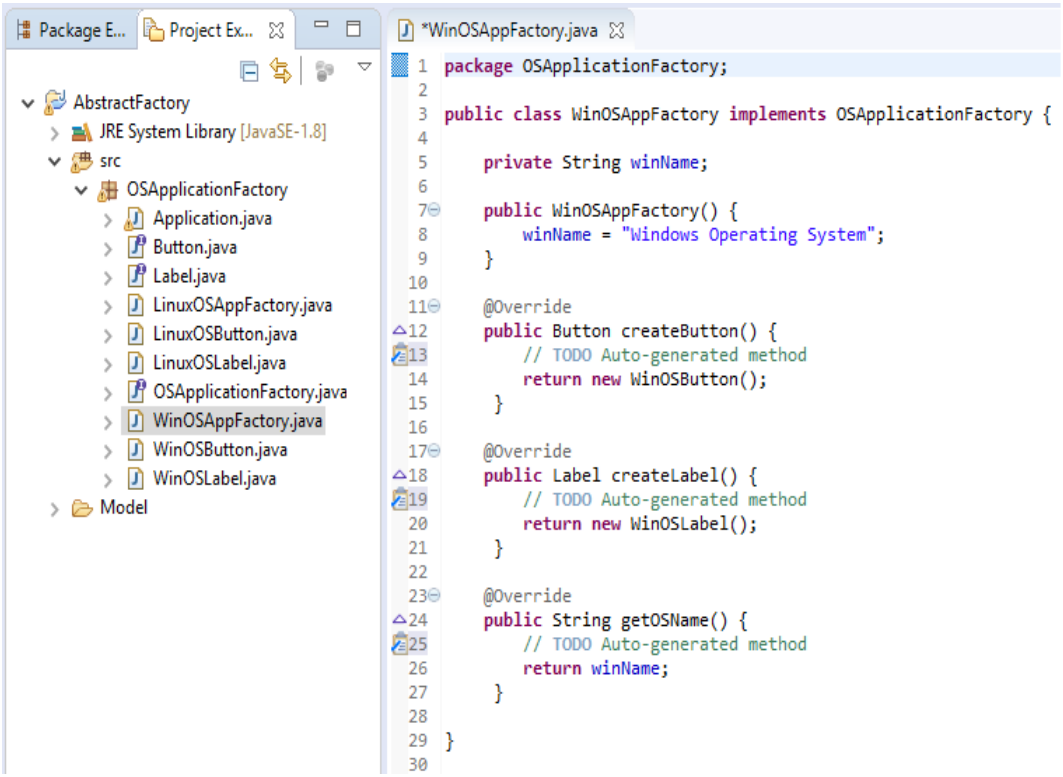

Рисунок 28

9. Необходима корректировка сгенерированного кода класса Application (рисунок 29). Данный класс написан для абстрактной фабрики OSApplicationFactory, а затем во время выполнения связывается с одной из конкретных фабрик LinuxOSAppFactory и WinOSAppFactory. Скорректированный код представлен на рисунке 30.

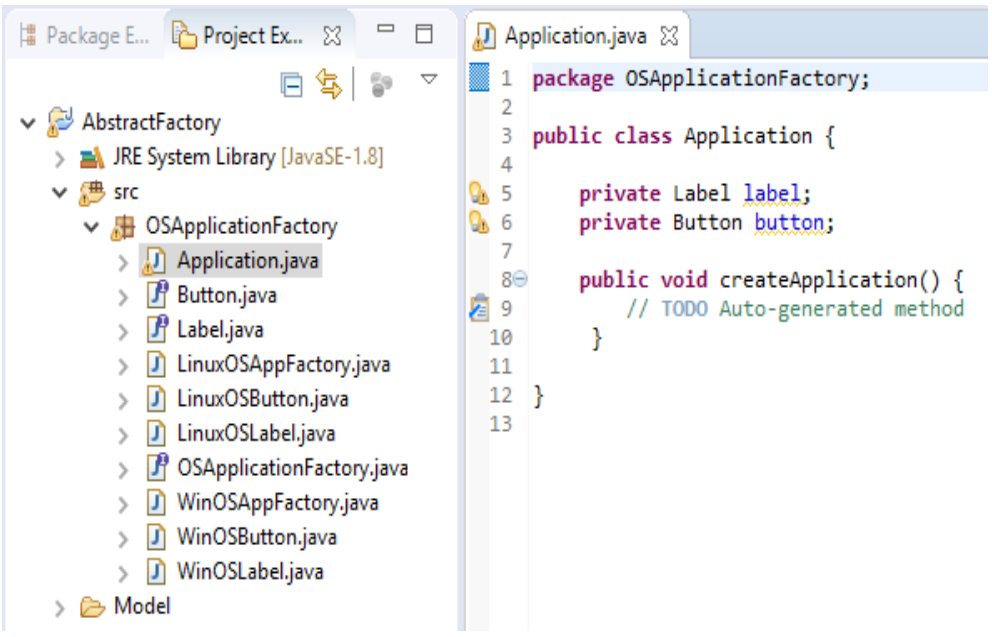

Рисунок 29

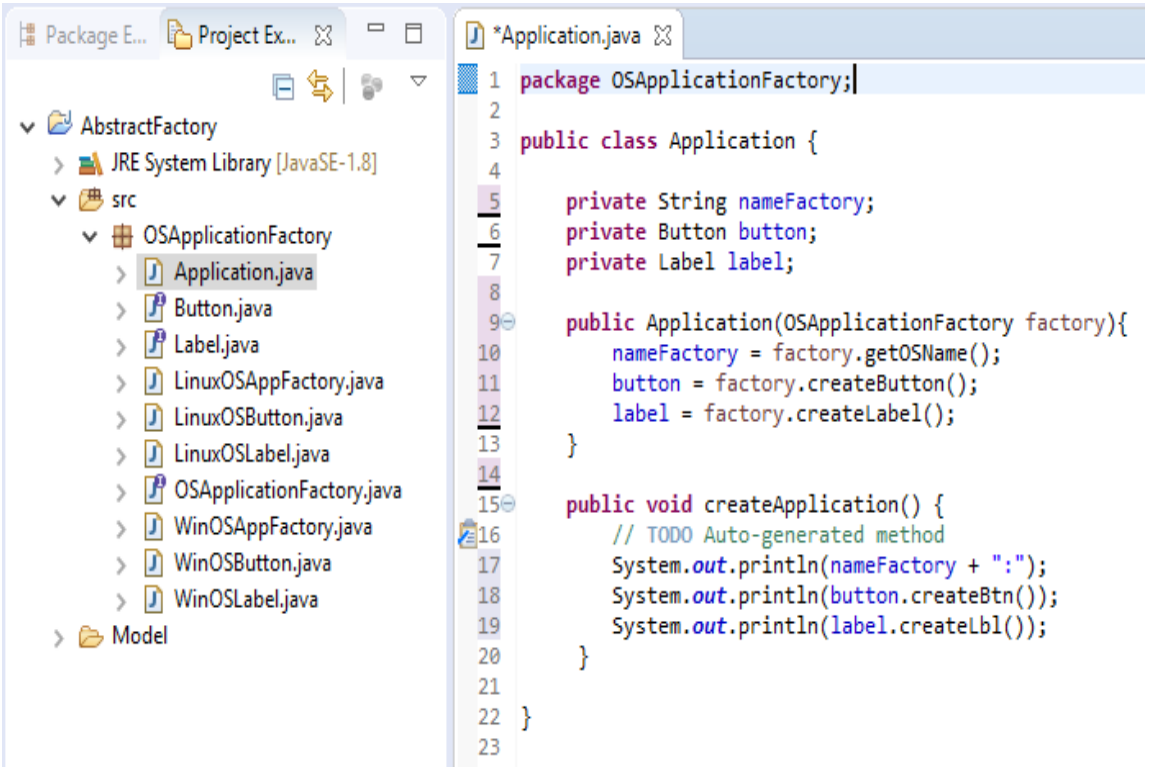

Рисунок 30

10. В пакете проекта OSApplicationFactory реализовать класс TestAbstractFactory для тестирования паттерна проектирования «Абстрактная фабрика» (рисунки 31–33).

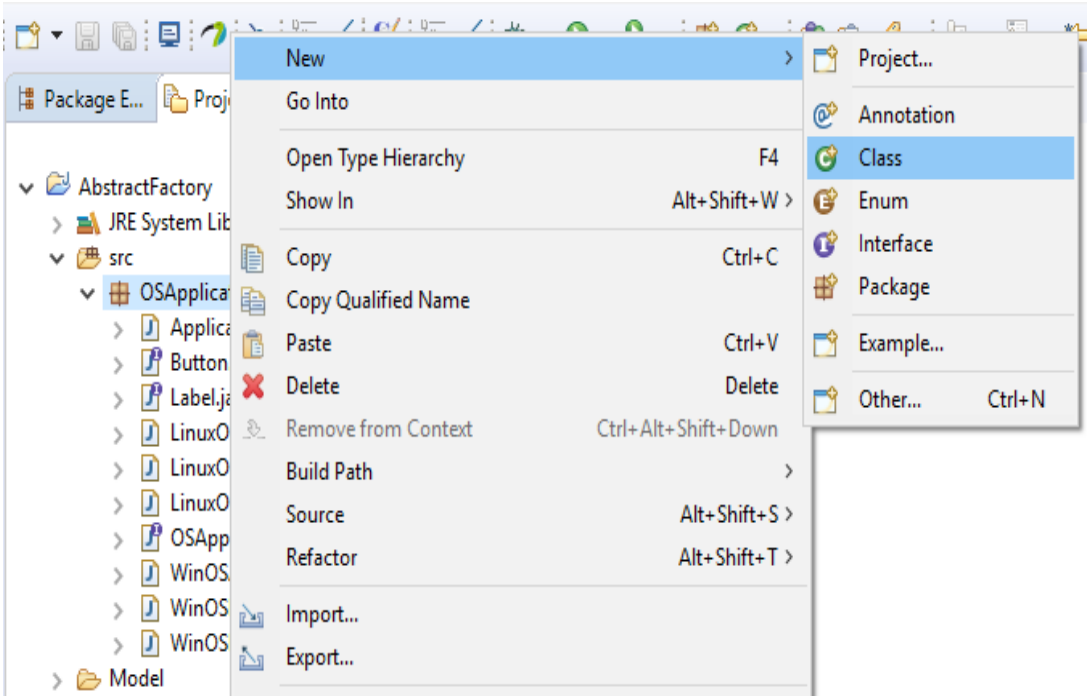

Рисунок 31

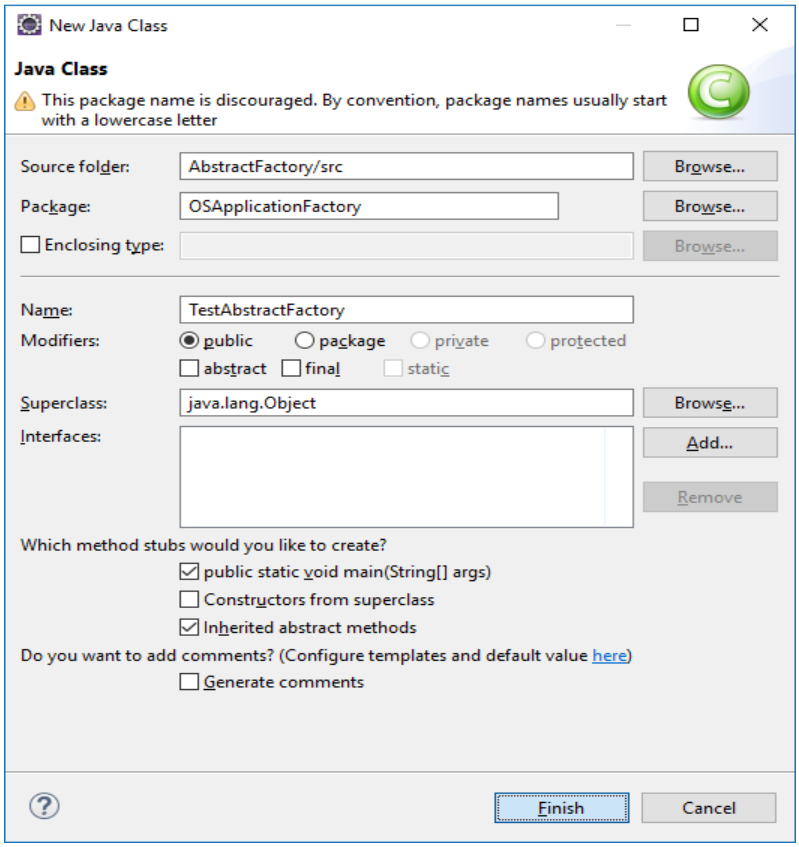

Рисунок 32

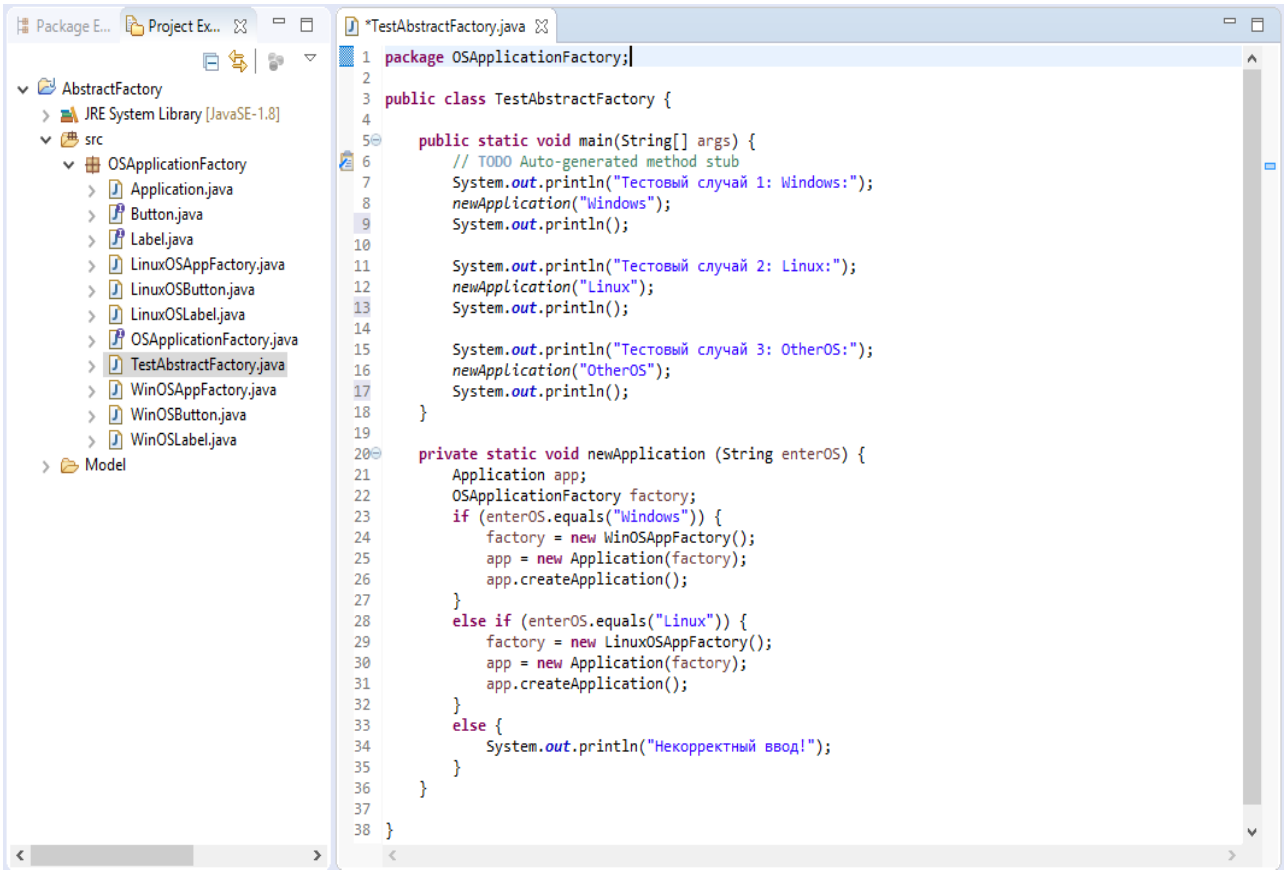

Рисунок 33

11. Для компиляции класса TestAbstractFactory нажать кнопку Run , чтобы запустить тестирование программы. В окне Console можно увидеть результаты тестирования (рисунок 34).

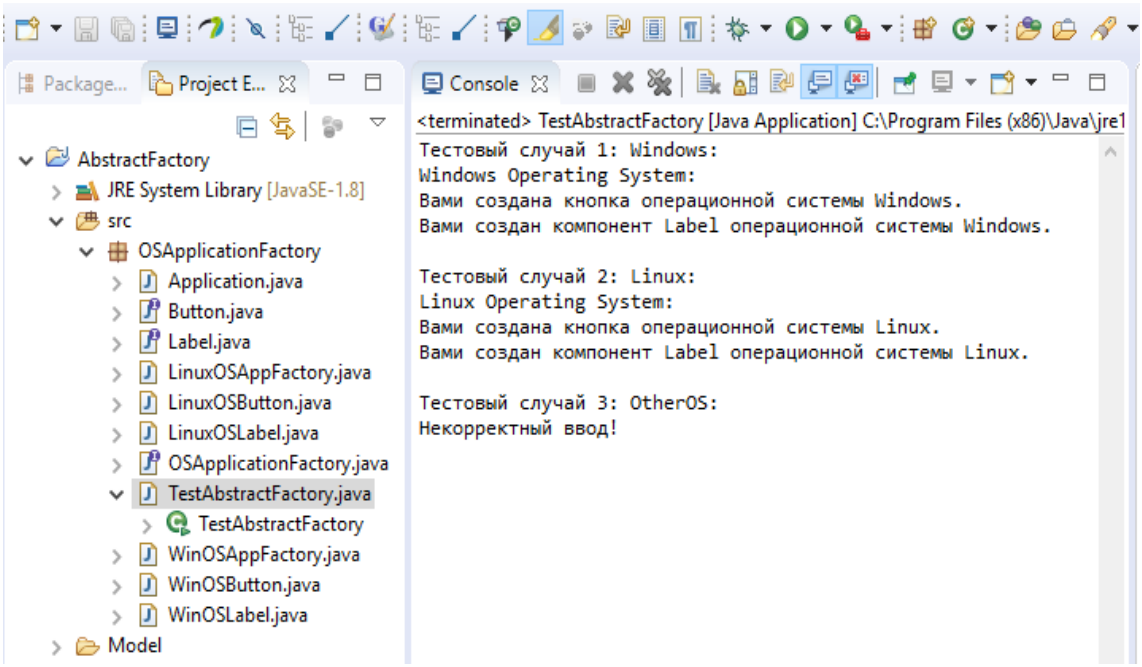

Рисунок 34

4 Содержание отчета по лабораторной работе

Отчет по лабораторной работе включает:

- титульный лист;
- условие задания;
- диаграммы классов для решения задачи;
- диаграммы последовательности для решения задачи;
- текст программы реализации паттерна проектирования «Абстрактная фабрика» для индивидуального задания;
- результаты тестирования программы.

5 Вопросы к защите практической работы

1. Что такое паттерн проектирования?
2. Для чего предназначен паттерн проектирования «Абстрактная фабрика»?
3. К какому типу паттернов проектирования относится «Абстрактная фабрика»?

4. Какие классы/интерфейсы являются участниками «Абстрактная фабрика»?

5. Назовите родственные для паттерна «Абстрактная фабрика» паттерны проектирования.

6. Какие методы использует интерфейс AbstractFactory в данном паттерне проектирования?

7. В чем отличие шаблона проектирования «Абстрактная фабрика» от шаблона «Фабричный Метод»?

8. Какие достоинства и недостатки имеет паттерн «Абстрактная фабрика»?

9. В каких случаях применяется данный шаблон проектирования?

6 Индивидуальные задания

1. Автомобилестроительная компания Ford производит легковой автомобиль Ford Focus и микроавтобус Ford Transit, а автомобилестроительная компания Mercedes-Benz производит легковой автомобиль Mercedes-Benz C-klasse и микроавтобус Mercedes-Benz Sprinter. Использовав паттерн проектирования «Абстрактная фабрика», реализовать программный продукт, демонстрирующий произведенные автомобили в каждой из перечисленных компаний.

2. На мебельной фабрике производят комплекты, каждый из которых состоит из кресла, дивана и стола, в трех разных стилях: Викторианском, Ампир и Модерн. Использовав паттерн проектирования «Абстрактная фабрика», реализовать программный продукт, демонстрирующий произведенный на фабрике комплект мебели в каждом из перечисленных стилях.

3. У каждого государства есть символика в виде флага и гимна и есть столица. Использовав паттерн проектирования «Абстрактная фабрика», реализовать программный продукт, демонстрирующий символику и столицу для каждого из следующих государств: Российская Федерация, Республика Беларусь, Индия.

4. В операционной системе Linux используется офисный пакет LibreOffice, содержащий текстовый процессор LibreOffice Writer и табличный редактор LibreOffice Calc, а в операционной системе macOS используется офисный пакет iWork, содержащий текстовый процессор iWork Pages и табличный редактор iWork Numbers. Использовав паттерн проектирования «Абстрактная фабрика», реализовать программный

продукт, демонстрирующий набор офисных приложений для каждой из перечисленных операционных систем.

5. В стратегической игре для каждой империи используют три типа персонажей: император, воин и крестьянин. Использовав паттерн проектирования «Абстрактная фабрика», реализовать программный продукт, демонстрирующий персонажей для каждой из следующих империй: Римская империя, Британская империя.

6. Для приготовления пиццы «Маргарита» используют в качестве начинки томаты и сыр моцарелла, а для приготовления гавайской пиццы – ананас и сыр пармезан. Использовав паттерн проектирования «Абстрактная фабрика», реализовать программный продукт, демонстрирующий основные ингредиенты начинки для каждой из перечисленной пиццы.

7. Интерфейс пользователя одного проекта состоит из следующих компонентов: текст интерфейса, изображения и справка пользователя. Использовав паттерн проектирования «Абстрактная фабрика», реализовать программный продукт, демонстрирующий интерфейс пользователя для каждого из следующих языков пользователя: русский, французский, итальянский.

8. На мебельной фабрике производят комплекты для спальни, каждый из которых состоит из кровати, шкафа и тумбочки, в четырех разных стилях: Классицизм, Неоклассицизм, Эклектика и Шале. Использовав паттерн проектирования «Абстрактная фабрика», реализовать программный продукт, демонстрирующий произведенный на фабрике комплект мебели в каждом из перечисленных стилях.

9. На мебельной фабрике производят комплекты для гостиной, каждый из которых состоит из дивана, двух кресел и журнального столика, в четырех разных стилях: Шебби-шик, Винтаж, Хай-тек и Контемпорари. Использовав паттерн проектирования «Абстрактная фабрика», реализовать программный продукт, демонстрирующий произведенный на фабрике комплект мебели в каждом из перечисленных стилях.

10. Интерфейс пользователя одного проекта состоит из следующих компонентов: текст интерфейса, изображения и справка пользователя. Использовав паттерн проектирования «Абстрактная фабрика», реализовать программный продукт, демонстрирующий интерфейс пользователя для каждого из следующих языков пользователя: русский, английский, немецкий.

11. Интерфейс пользователя одного проекта состоит из следующих компонентов: текст интерфейса, изображения и справка пользователя.

Использовав паттерн проектирования «Абстрактная фабрика», реализовать программный продукт, демонстрирующий интерфейс пользователя для каждого из следующих языков пользователя: русский, французский, испанский.

12. У каждого государства есть символика в виде флага и гимна и есть столица. Использовав паттерн проектирования «Абстрактная фабрика», реализовать программный продукт, демонстрирующий символику и столицу для каждого из следующих государств: Российская Федерация, Китай, Индия.

13. У каждого государства есть символика в виде флага и гимна и есть столица. Использовав паттерн проектирования «Абстрактная фабрика», реализовать программный продукт, демонстрирующий символику и столицу для каждого из следующих государств: Российская Федерация, Испания, Франция.

14. У каждого государства есть символика в виде флага и гимна и есть столица. Использовав паттерн проектирования «Абстрактная фабрика», реализовать программный продукт, демонстрирующий символику и столицу для каждого из следующих государств: Российская Федерация, Исландия, Мексика.

15. У каждого государства есть символика в виде флага и гимна и есть столица. Использовав паттерн проектирования «Абстрактная фабрика», реализовать программный продукт, демонстрирующий символику и столицу для каждого из следующих государств: Российская Федерация, Австралия, Бразилия.

Список использованных источников

1 Ларман, К. Применение UML 2.0 и шаблонов проектирования/ К. Ларман. - М.: Издательский дом «Вильямс», 2013. -736 с. - Текст: непосредственный

2 Османи, Э. Паттерны для масштабируемых JavaScript-приложений/ Э. Османи. - М.: Техносфера, 2015. -188 с. - Текст: непосредственный

3 Фримен Э. Паттерны проектирования/ Э. Фримен, Э. Фримен, К. Сьерра, Б. Бейтс. – СПб.: Питер, 2016. - 653 с. - Текст: непосредственный

4 Перл, И. А. Введение в методологию программной инженерии : учебное пособие: [16+]/ И. А. Перл, О. В. Калёнова. – Санкт-Петербург : Университет ИТМО, 2019. – 53 с. : ил., схем. – Режим доступа: по подписке. – URL: https://biblioclub.ru/index.php?page=book&id=566776 (дата обращения: 28.08.2025). – Бибиогр. в кн. – Текст : электронный.

5 Шуваев, А. В. Программная инженерия: учебное пособие для магистрантов направления подготовки 09.04.02 – Информационные системы и технологии: [16+]/ А. В. Шуваев; Ставропольский государственный аграрный университет, Кафедра информационных систем. – Ставрополь: Ветеран, 2020. – 84 с. : ил., табл. – Режим доступа: по подписке. – URL: https://biblioclub.ru/index.php?page=book&id=700960 (дата обращения: 28.08.2025). – Библиогр. в кн. – Текст : электронный.

МИНОБРНАУКИРОССИИ

Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Юго-Западный государственный университет»
(ЮЗГУ)

МИНОБРНАУКИРОССИИ

Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Юго-Западный государственный университет»
(ЮЗГУ)

Кафедра программной инженерии

УТВЕРЖДАЮ
Проректор по учебной работе
_____О.Г. Локтионова
«_4_»_09_____ 2025 г.

ПАТТЕРН ПРОЕКТИРОВАНИЯ «КОМПОНОВЩИК»

Методические указания по выполнению практической работы по дисциплине "Методология программной инженерии" для студентов направления подготовки магистров 09.04.04 "Программная инженерия"

Курск 2025

УДК 004.65

Составители: Т.М. Белова

Рецензент
Кандидат технических наук, доцент кафедры вычислительной техники ЮЗГУ Т.И. Лапина

Паттерн проектирования «Компоновщик»: методические указания по выполнению практической работы по дисциплине "Методология программной инженерии" для студентов направления подготовки магистров 09.04.04 "Программная инженерия"/ Юго-Зап. гос. ун-т; сост.: Т.М. Белова – Курск, 2025. – 26 с.: ил. 30.

Изложена последовательность действий по разработке и применению паттерна проектирования «Компоновщик».

Материал предназначен для студентов направления подготовки магистров 09.04.04 «Программная инженерия», а также будет полезен студентам всех направлений подготовки, изучающим технологии разработки программных продуктов с использованием паттернов.

Текст печатается в авторской редакции.

Подписано в печать . Формат 60x84 1/16.
Усл. печ. л. 1,5. Уч.-изд. л. 1,4. Тираж 50 экз. Заказ . Бесплатно.
Юго-Западный государственный университет
305040, Курск, ул.50 лет Октября, 94.

Содержание

1 Цель практической работы

Целью практической работы является приобретение знаний, умений и навыков для использования возможностей шаблона проектирования «Компоновщик».

2 Основные понятия

При реализации проектов по разработке программных систем и моделированию бизнес-процессов встречаются ситуации, когда решение проблем в различных проектах имеют сходные структурные черты. Попытки выявить похожие схемы или структуры в рамках объектно-ориентированного анализа и проектирования привели к появлению понятия паттерна, которое из абстрактной категории превратилось в непременный атрибут современных CASE-средств.

Паттерны различаются степенью детализации и уровнем абстракции. Предлагается следующая общая классификация паттернов по категориям их применения:

- архитектурные паттерны,
- паттерны проектирования,
- паттерны анализа,
- паттерны тестирования,
- паттерны реализации.

Архитектурные паттерны (Architectural patterns) - множество предварительно определенных подсистем со спецификацией их ответственности, правил и базовых принципов установления отношений между ними.

Архитектурные паттерны предназначены для спецификации фундаментальных схем структуризации программных систем. Наиболее известными паттернами этой категории являются паттерны GRASP (General Responsibility Assignment Software Pattern). Эти паттерны относятся к уровню системы и подсистем, но не к уровню классов. Как правило, формулируются в обобщенной форме, используют обычную терминологию и не зависят от области приложения. Паттерны этой категории систематизировал и описал К. Ларман.

Паттерны проектирования (Design patterns) - специальные схемы для уточнения структуры подсистем или компонентов программной системы и отношений между ними.

Паттерны проектирования описывают общую структуру взаимодействия элементов программной системы, которые реализуют

исходную проблему проектирования в конкретном контексте. Наиболее известными паттернами этой категории являются паттерны GoF (Gang of Four), названные в честь Э. Гаммы, Р. Хелма, Р. Джонсона и Дж. Влиссидеса, которые систематизировали их и представили общее описание. Паттерны GoF включают в себя 23 паттерна. Эти паттерны не зависят от языка реализации, но их реализация зависит от области приложения.

Паттерны анализа (Analysis patterns) - специальные схемы для представления общей организации процесса моделирования.

Паттерны анализа относятся к одной или нескольким предметным областям и описываются в терминах предметной области. Наиболее известными паттернами этой группы являются паттерны бизнес-моделирования ARIS (Architecture of Integrated Information Systems), которые характеризуют абстрактный уровень представления бизнес-процессов. Паттерны анализа конкретизируются в типовых моделях с целью выполнения аналитических оценок или имитационного моделирования бизнес-процессов.

Паттерны тестирования (Test patterns) - специальные схемы для представления общей организации процесса тестирования программных систем.

К этой категории паттернов относятся такие паттерны, как тестирование черного ящика, белого ящика, отдельных классов, системы. Паттерны этой категории систематизировал и описал М. Гранд. Некоторые из них реализованы в инструментальных средствах, наиболее известными из которых является IBM Test Studio. В связи с этим паттерны тестирования иногда называют стратегиями или схемами тестирования.

Паттерны реализации (Implementation patterns) - совокупность компонентов и других элементов реализации, используемых в структуре модели при написании программного кода.

Эта категория паттернов делится на следующие подкатегории: паттерны организации программного кода, паттерны оптимизации программного кода, паттерны устойчивости кода, паттерны разработки графического интерфейса пользователя и др. Паттерны этой категории описаны в работах М. Гранда, К. Бека, Дж. Тидвелла и др. Некоторые из них реализованы в популярных интегрированных средах программирования в форме шаблонов создаваемых проектов. В этом случае выбор шаблона программного приложения позволяет получить некоторую заготовку программного кода.

Шаблон проектирования или паттерн (англ. design pattern) в разработке программного обеспечения — повторяемая архитектурная конструкция, представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста.

Технически, паттерны (шаблоны) проектирования - это абстрактные примеры правильного использования небольшого числа комбинаций простейших техник объектно-ориентированного программирования. Паттерны проектирования - это простые примеры, показывающие правильные способы организации взаимодействий между классами или объектами.

Паттерн "Компоновщик" относится к группе структурных паттернов. Структурные паттерны (Structural) определяют различные сложные структуры, которые изменяют интерфейс уже существующих объектов или его реализацию, позволяя облегчить разработку и оптимизировать программу.

К таким шаблонам относятся:

- Адаптер (Adapter),
- Мост (Bridge),
- Компоновщик (Composite),
- Декоратор (Decorator),
- Фасад (Facade),
- Приспособленец (Flyweight),
- Заместитель (Proxy).

Адаптер (Adapter) — предоставляет объект, обеспечивающий взаимодействие двух других объектов, один из которых использует, а другой предоставляет несовместимый с первым интерфейс.

Мост (Bridge) — представляет структуру, позволяющую изменять интерфейс обращения и интерфейс реализации класса независимо.

Компоновщик (Composite) — объект, который объединяет в себе объекты, подобные ему самому.

Декоратор или Обёртка (Decorator или Wrapper) — класс, расширяющий функциональность другого класса без использования наследования. Он динамически добавляет новые обязанности объекту. Декораторы являются гибкой альтернативой порождению подклассов для расширения функциональности.

Фасад (Facade) — объект, который абстрагирует работу с несколькими классами, объединяя их в единое целое.

Приспособленец (Flyweight) —объект, представляющий себя как уникальный экземпляр в разных местах программы, но по факту не являющийся таковым. Позволяет использовать разделяемые объекты сразу в нескольких контекстах. Данный паттерн используется преимущественно для оптимизации работы с памятью.

Заместитель (Proxy) — объект, который является посредником между двумя другими объектами, и который реализует/ограничивает доступ к объекту, к которому обращаются через него.

Преимущества от применения паттернов проектирования заключаются в следующем:

- Паттерны позволяют суммировать опыт экспертов и сделать его доступным рядовым разработчикам.
- Имена паттернов образуют своего рода словарь, который позволяет разработчикам лучше понимать друг друга.
- Если в документации системы указано, какие паттерны в ней используются, это позволяет читателю быстрее понять систему.
- Паттерны упрощают реструктуризацию системы независимо от того, использовались ли паттерны при ее проектировании.
- Паттерн дает название проблеме и определяет способы решения многих проблем за счет готового набора абстракций.
- Использование шаблонов проектирования аналогично использованию готовых библиотек кода.
- Правильное использование шаблонов помогает разработчикам определить нужный вектор развития и уйти от многих проблем, которые могут возникнуть в процессе разработки.
- Паттерны проектирования не зависят от языка программирования.

Правильно выбранные паттерны проектирования позволяют сделать программную систему более гибкой, ее легче поддерживать и модифицировать, а код такой системы в большей степени соответствует концепции повторного использования.

Проблемы, которые порождают шаблоны проектирования:

- потеря гибкости проектирования и разработки системы;
- использование шаблонов усложняет систему;
- слепое следование определенному шаблону и повсеместное его использование может породить много архитектурных и логических проблем.

3 Порядок выполнения лабораторной работы

Данный шаблон предназначен, прежде всего, для использования в таких случаях, когда нужно обратиться к отдельным объектам и к группам объектов одинаково. Рассматриваемая задача заключается в реализации алгоритмов добавления, удаления и просмотра элементов в древовидной структуре.

1. В интегрированной среде разработки Eclipse нужно создать новый Java-проект Coposite для разработки паттерна проектирования «Компоновщик» (рисунки 1–2).

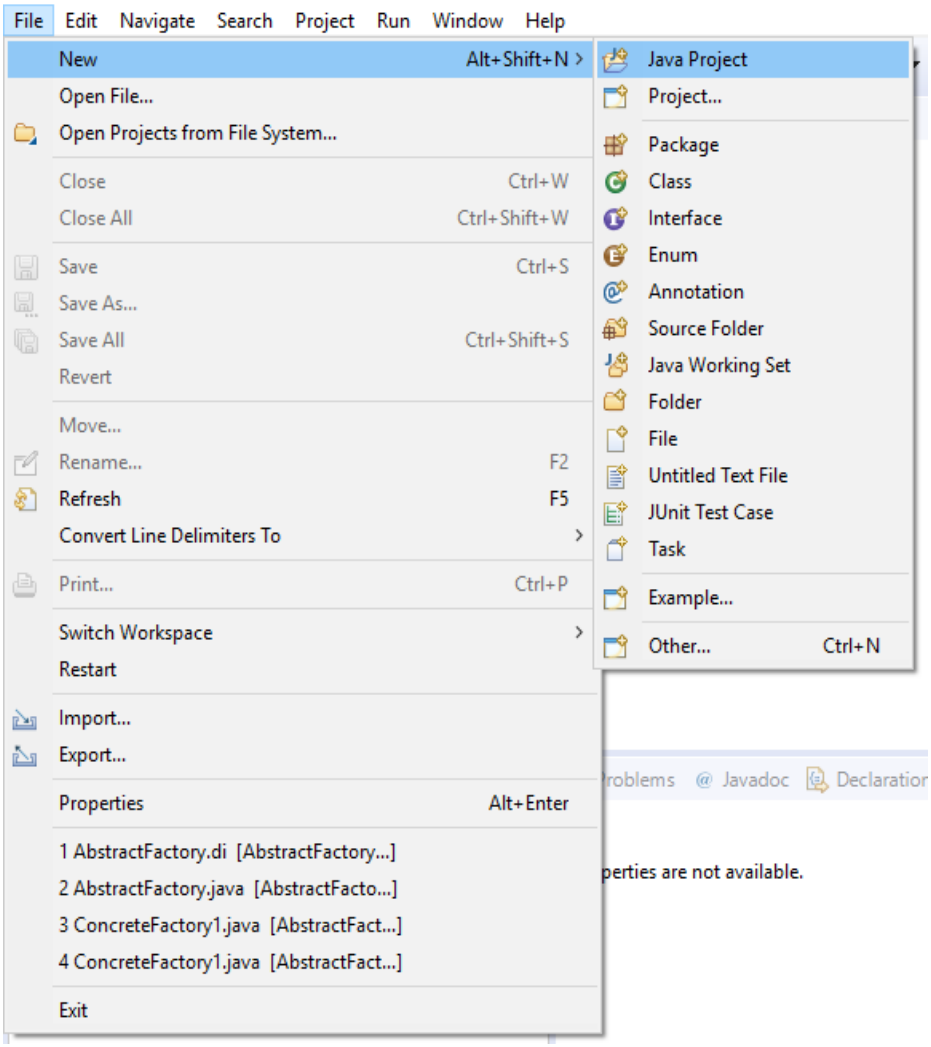

Рисунок 1

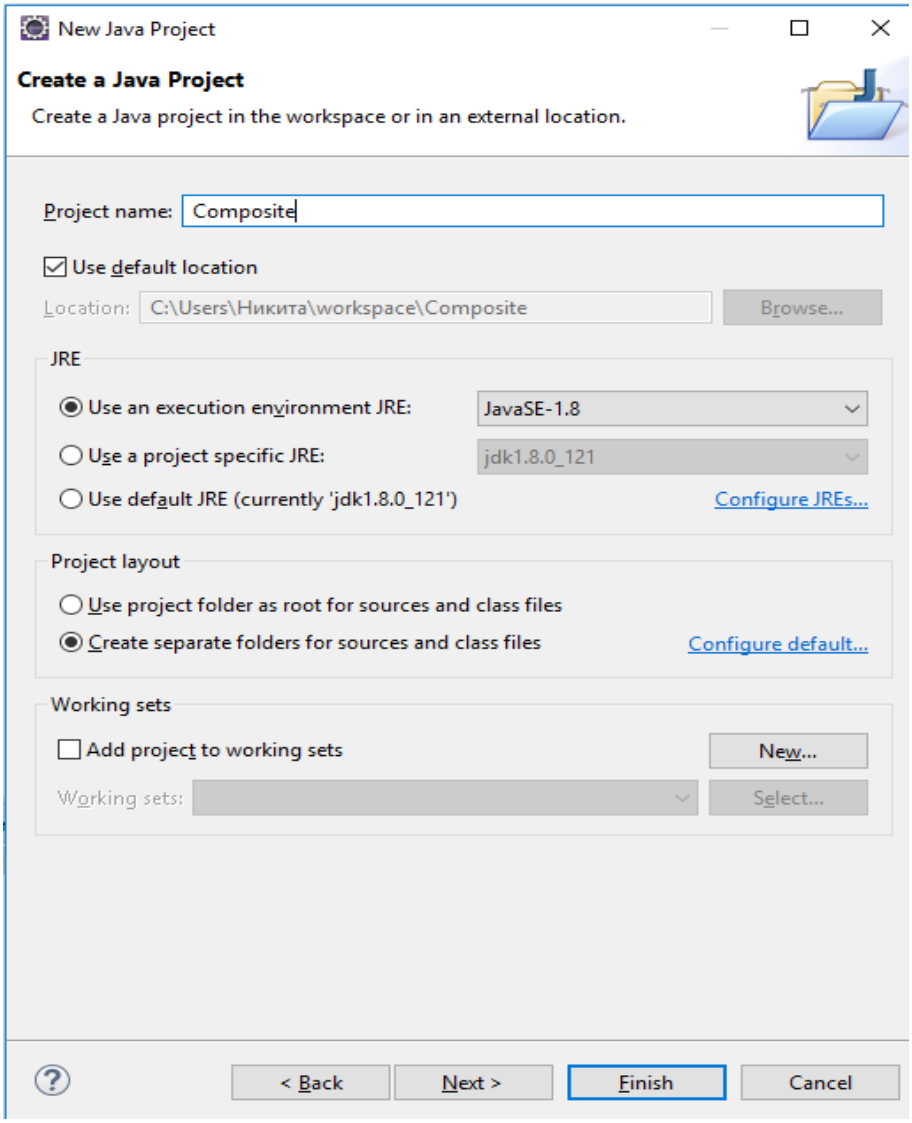

Рисунок 2

2. В проекте нужно создать папку Model для проектирования диаграммы классов (рисунки 3–4). В созданной папке добавьте необходимые файлы для работы с инструментом проектирования UML-диаграмм Papyrus (рисунки 5–10).

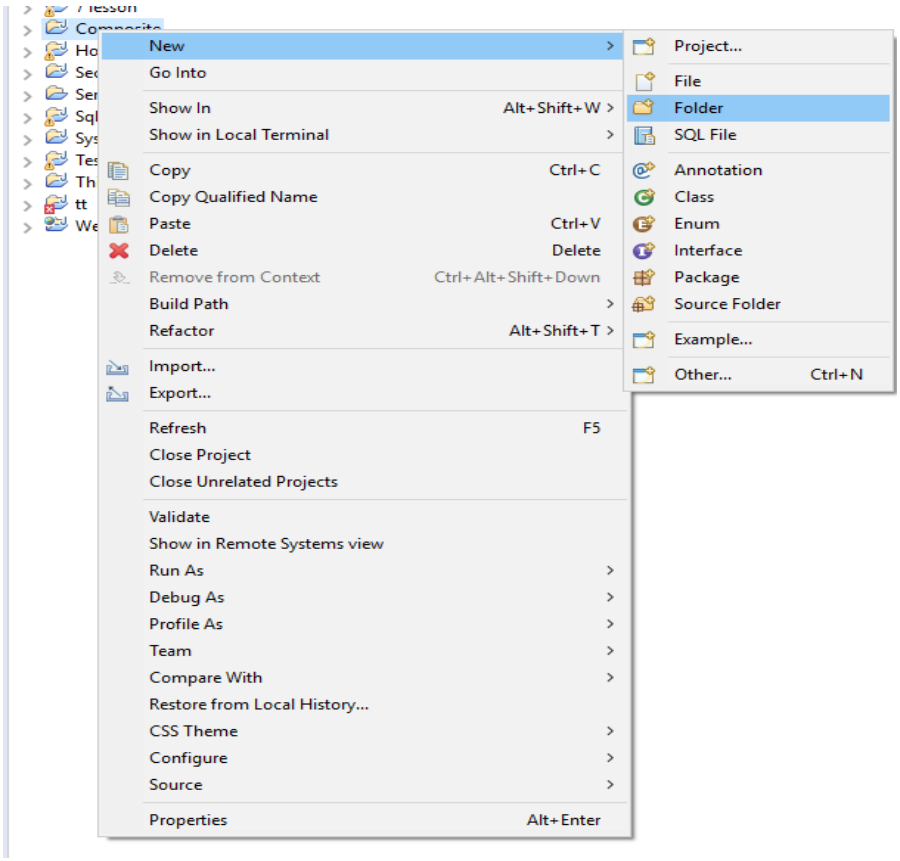

Рисунок 3

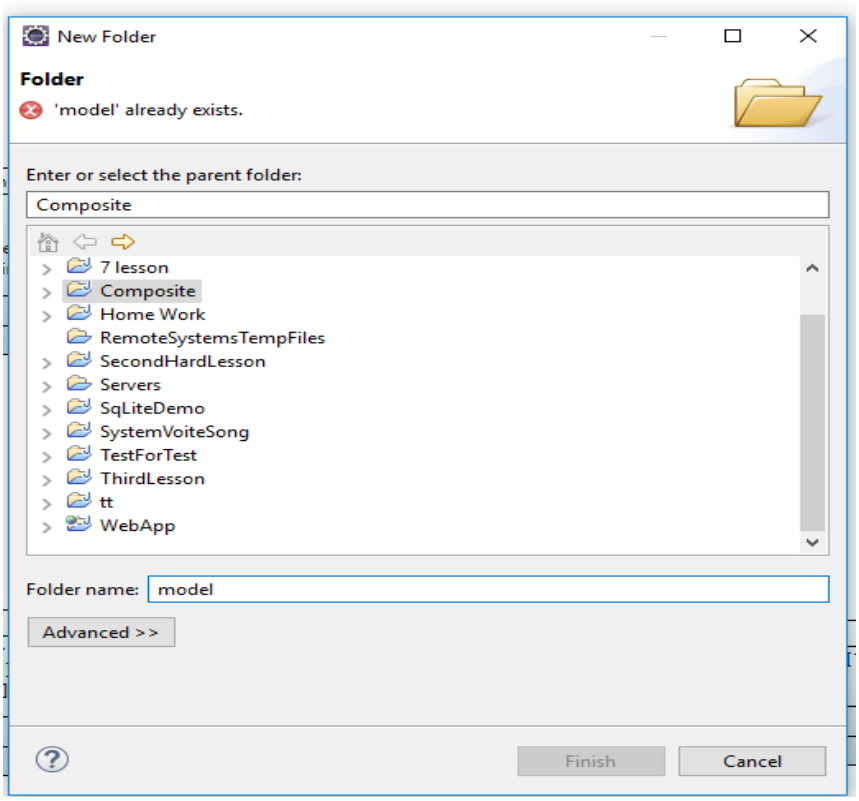

Рисунок 4

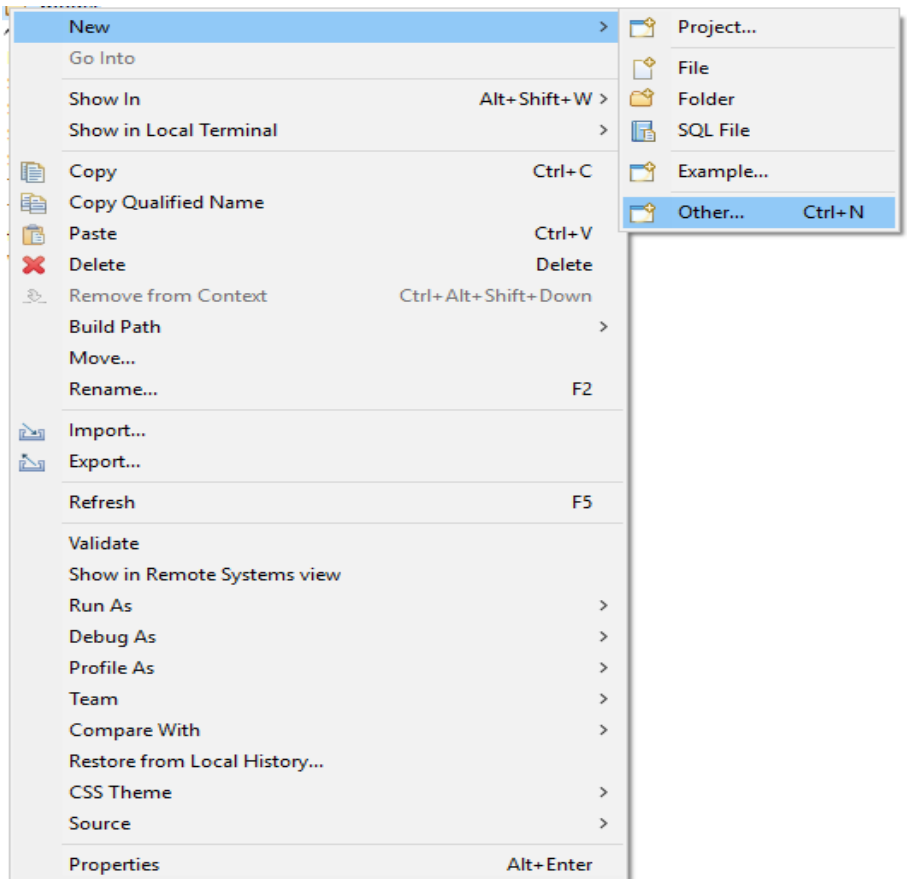

Рисунок 5

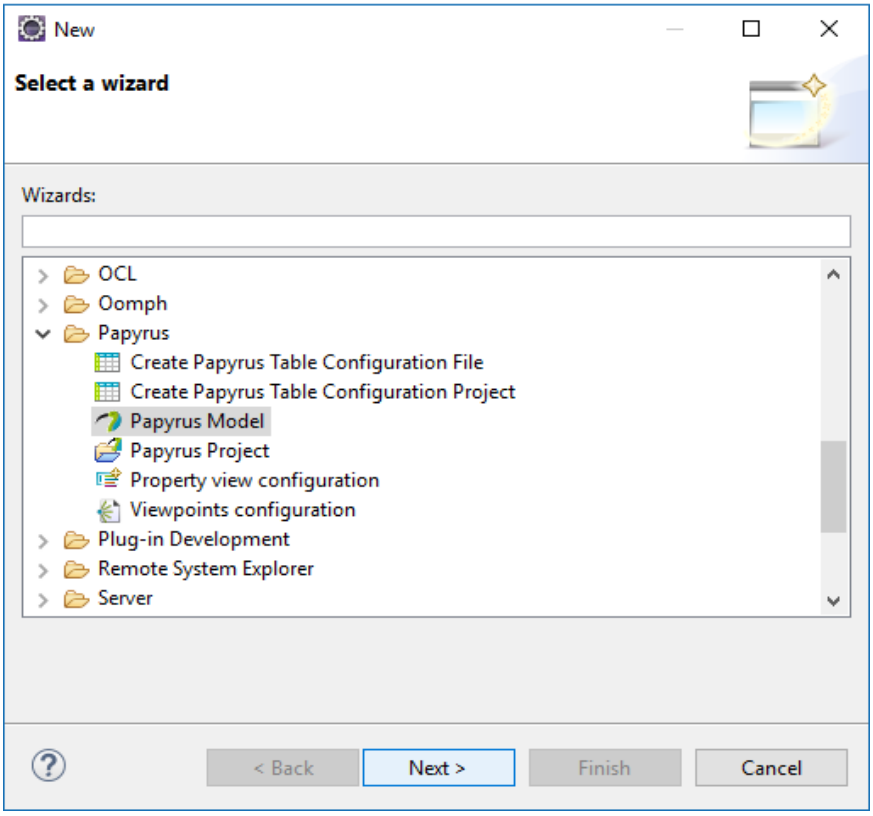

Рисунок 6

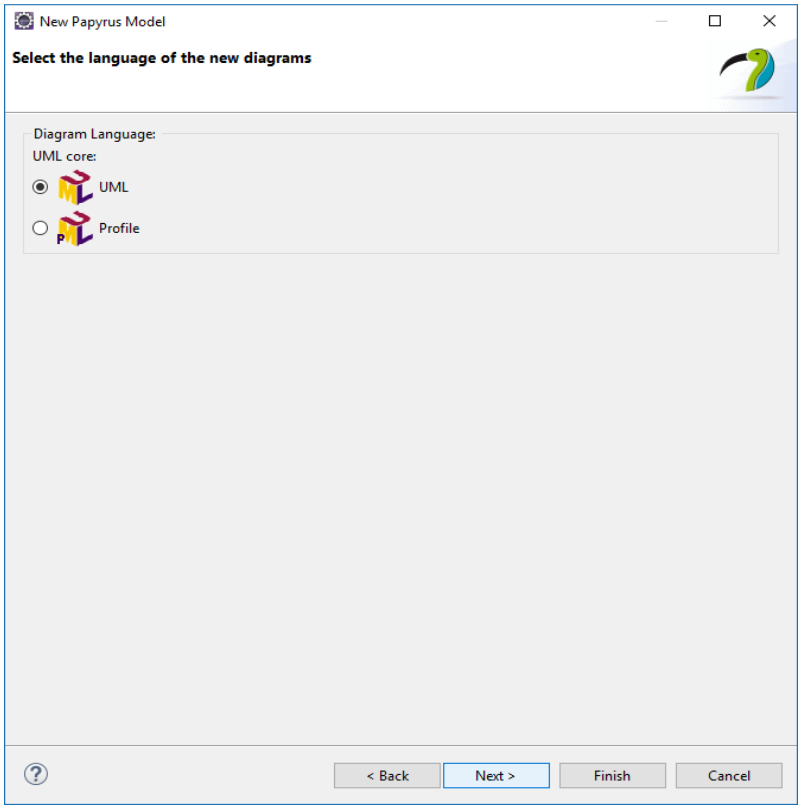

Рисунок 7

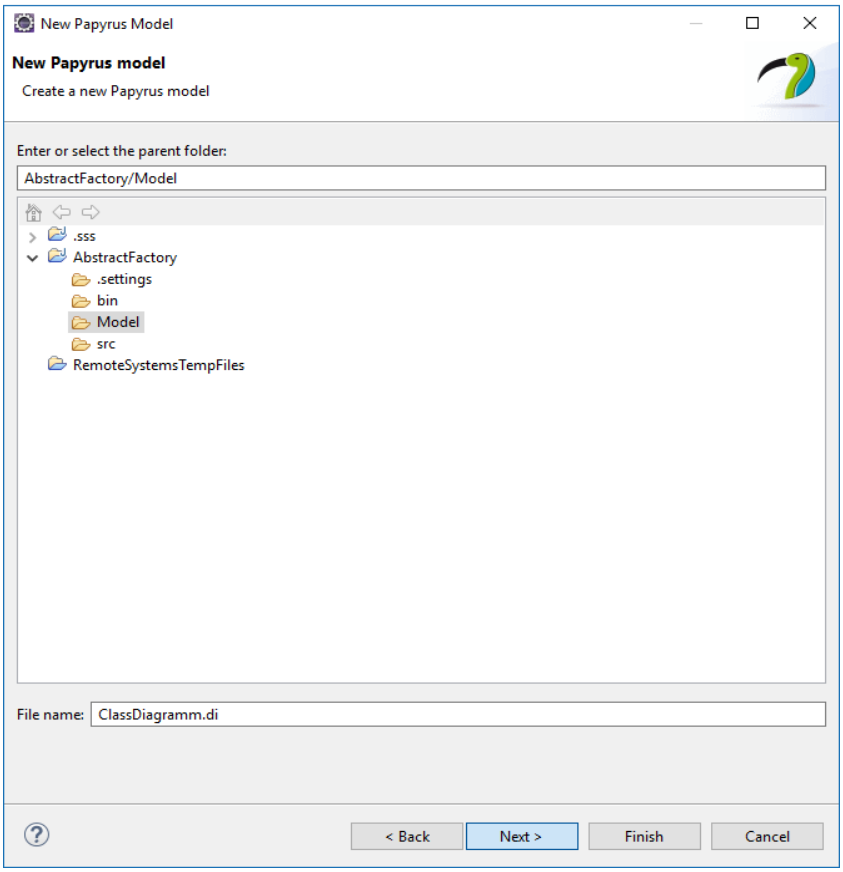

Рисунок 8

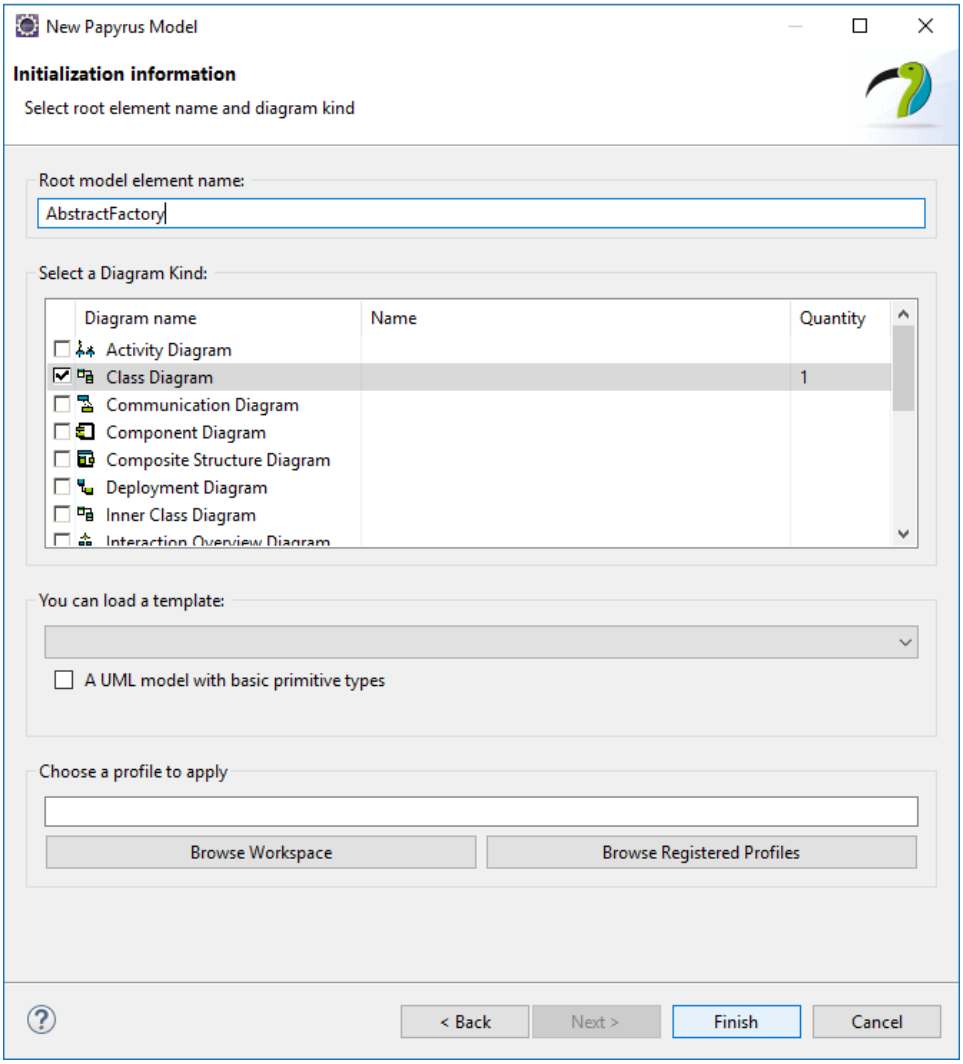

Рисунок 9

Рисунок 10

3. Для того, чтобы использовать типы данных языка Java в диаграмме классов, необходимо подключить к проекту диаграмм Coposite библиотеку JavaPrimitiveTypes (рисунок 11–13).

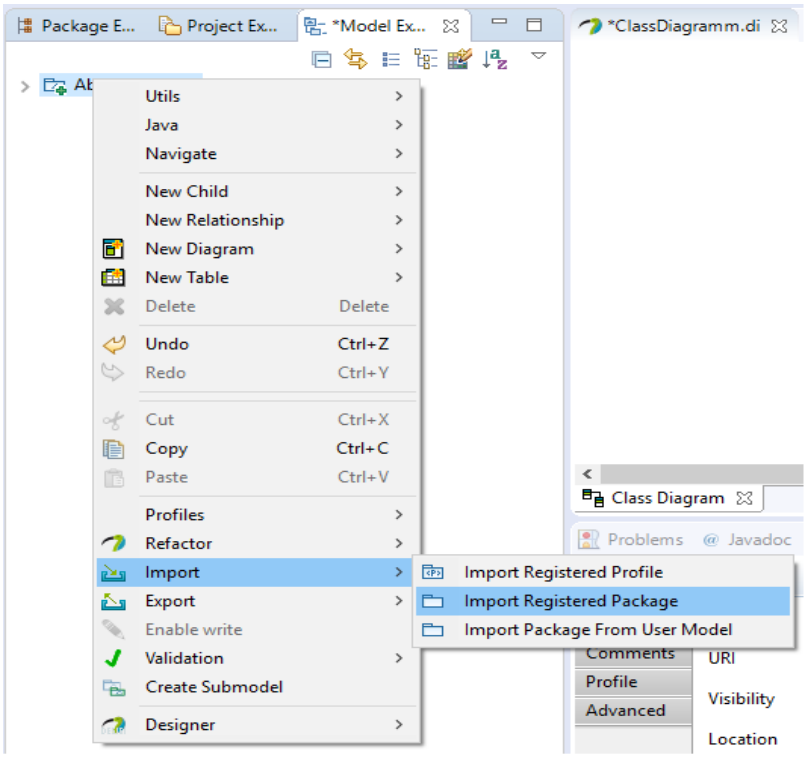

Рисунок 11

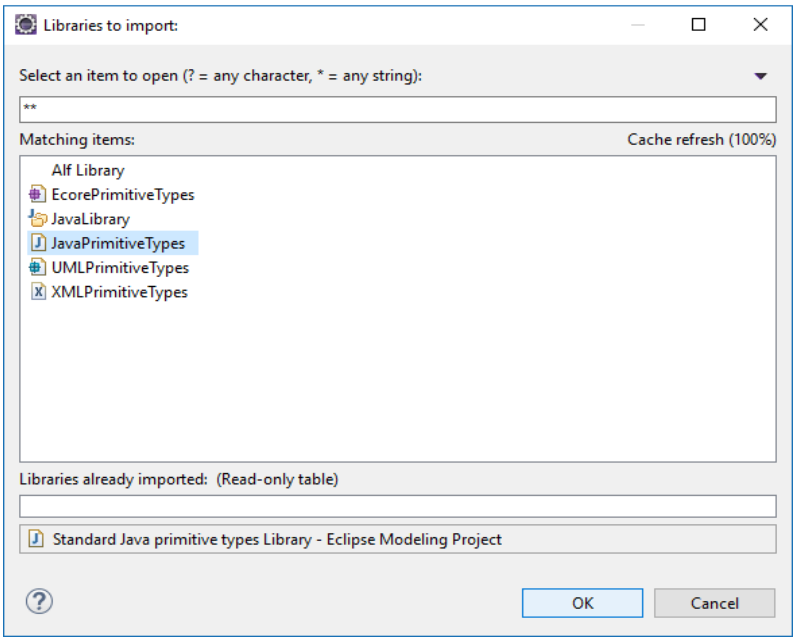

Рисунок 12

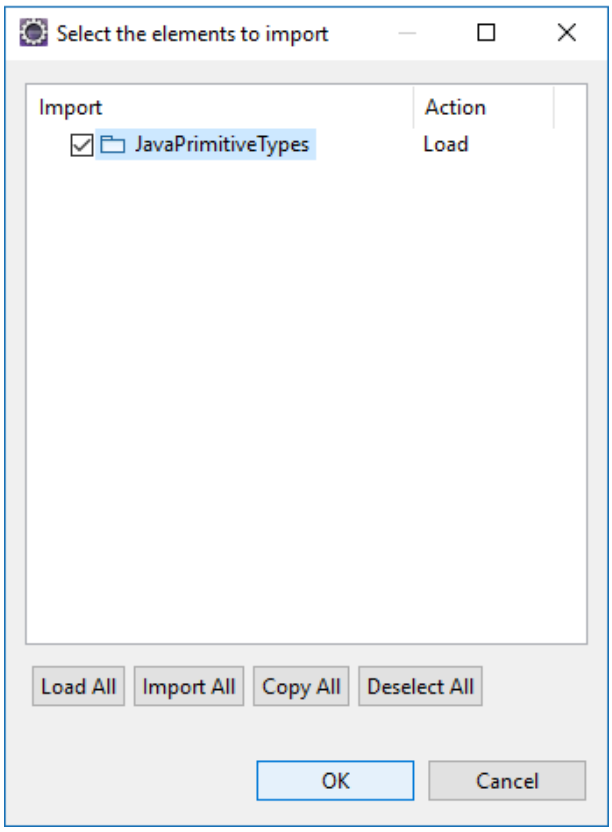

Рисунок 13

4. Далее следует разработать в проектировщике UML-диаграмм Papyrus диаграмму классов шаблона проектирования «Компоновщик», как показано на рисунке 14.

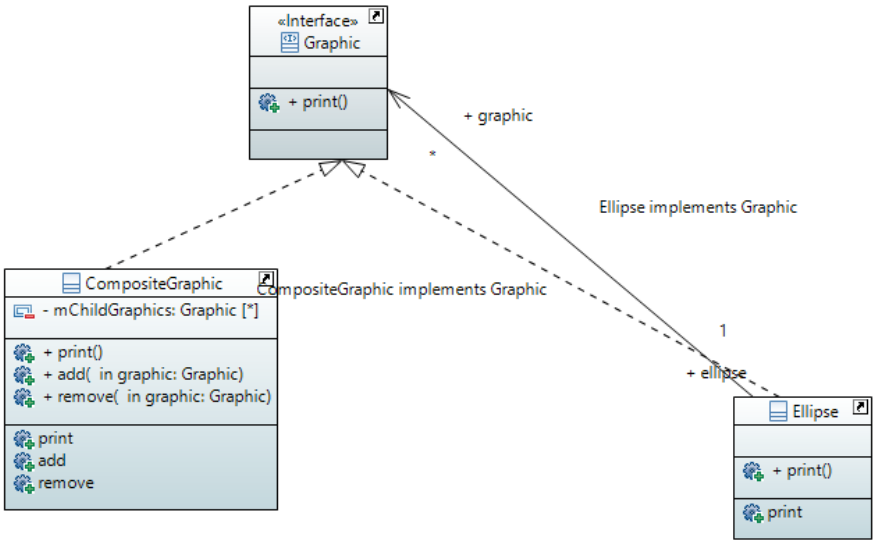

Рисунок 14

5. Для того, чтобы сгенерировать Java-код из полученной диаграммы классов, необходимо подключить к проекту диаграмм Coposite профиль Java (рисунок 15–16).

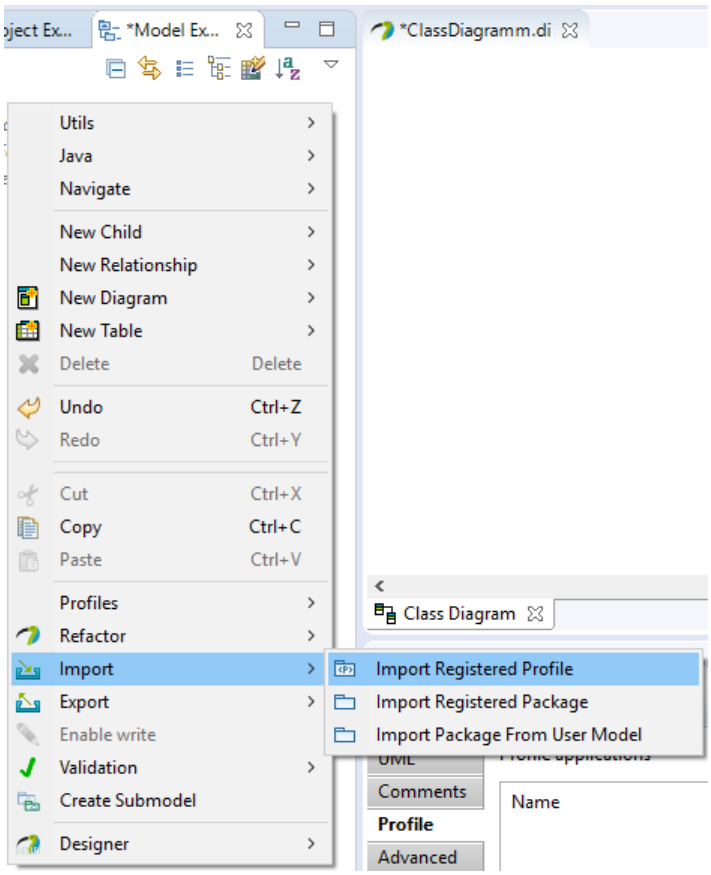

Рисунок 15

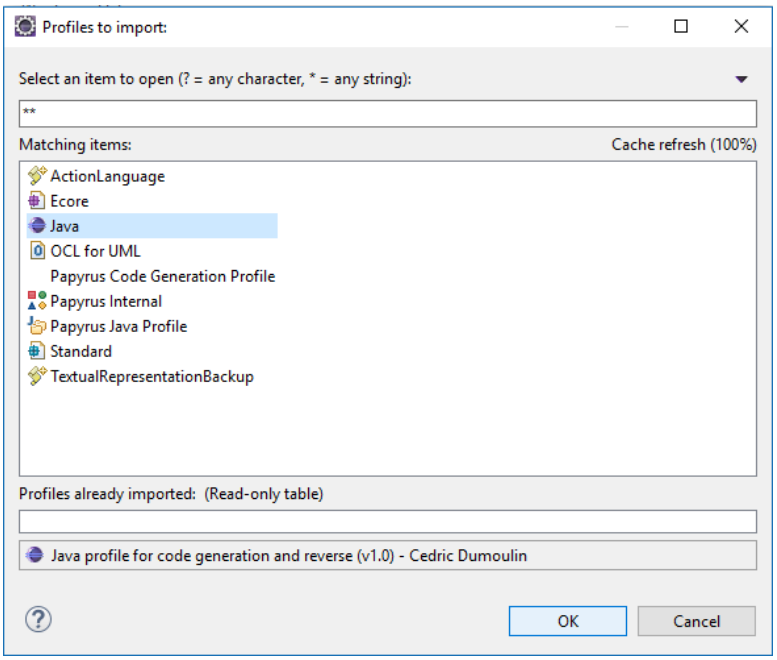

Рисунок 16

6. В свойствах пакета OSCoposite из диаграммы классов следует добавить профиль приложения Java (рисунки 17–19) и стереотип пакета JavaPackage_ (рисунки 20–21).

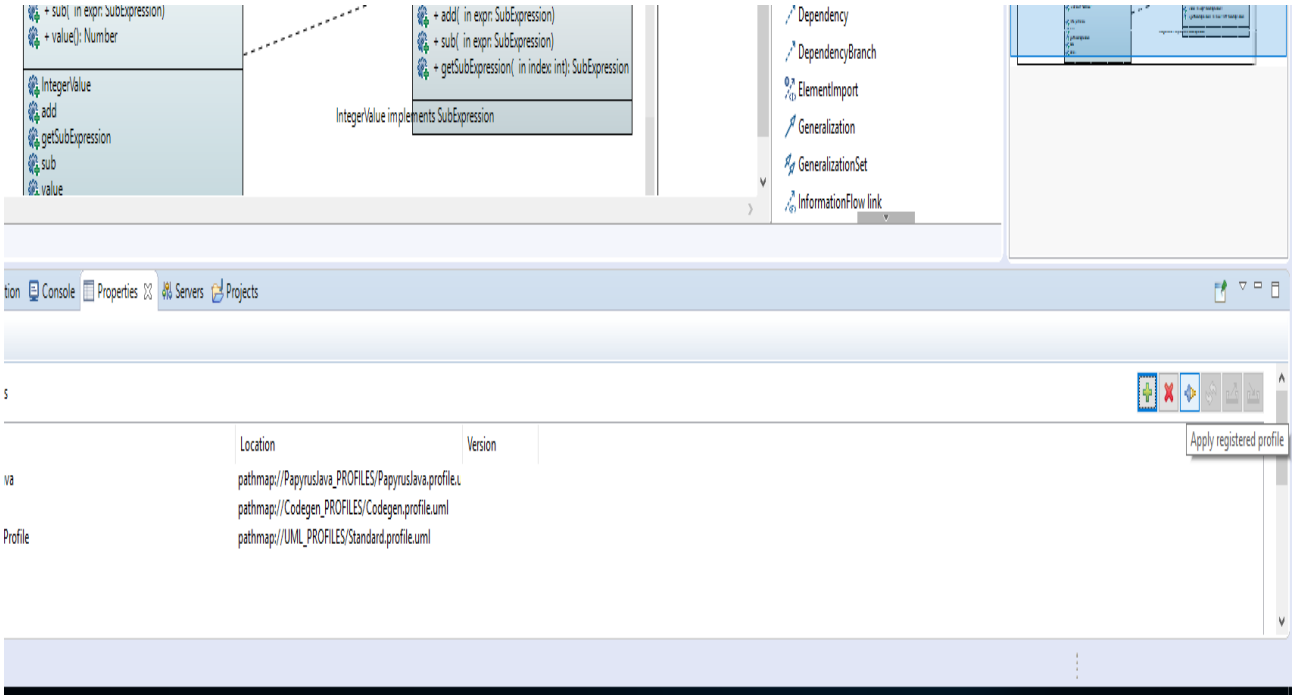

Рисунок 17

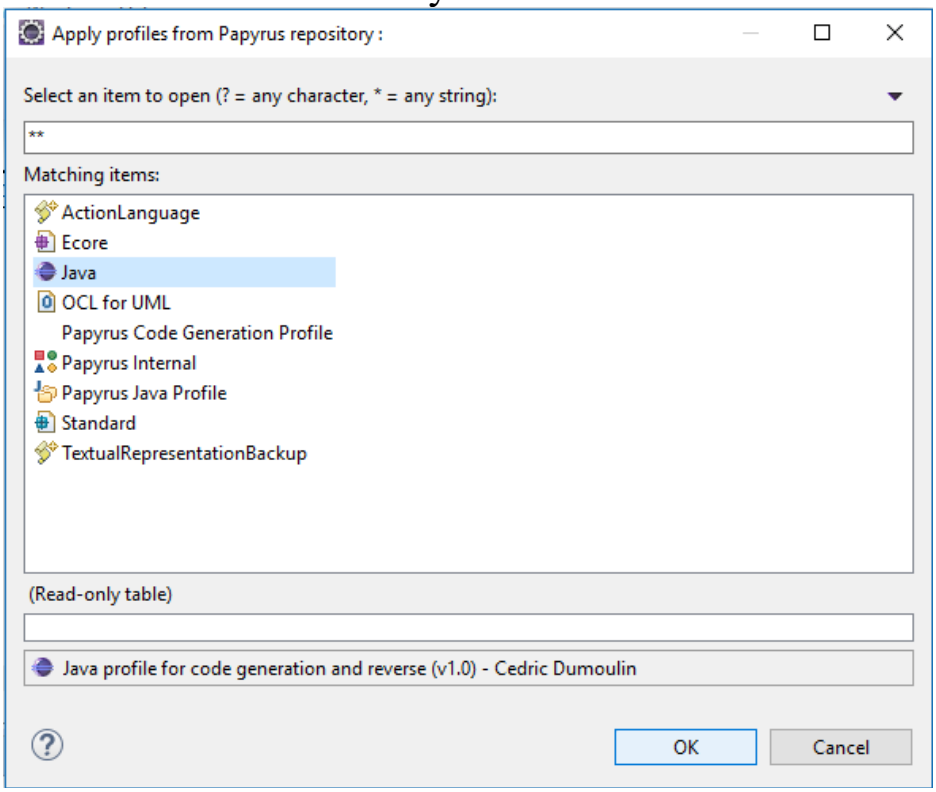

Рисунок 18

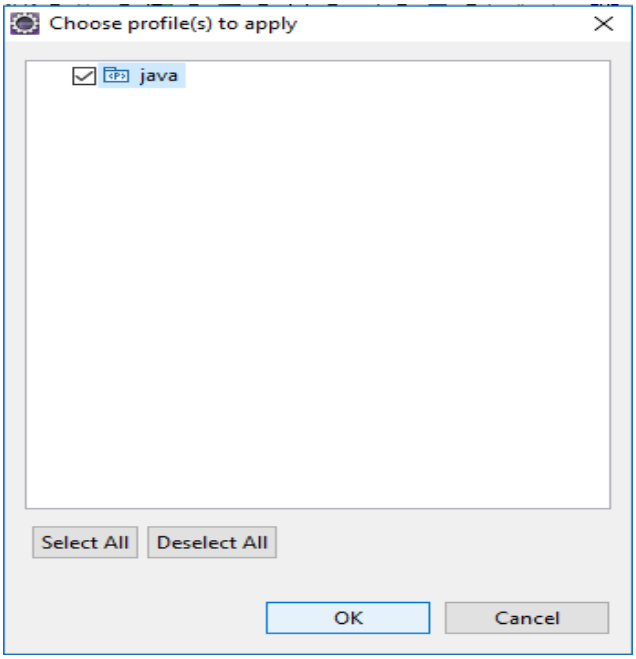

Рисунок 19

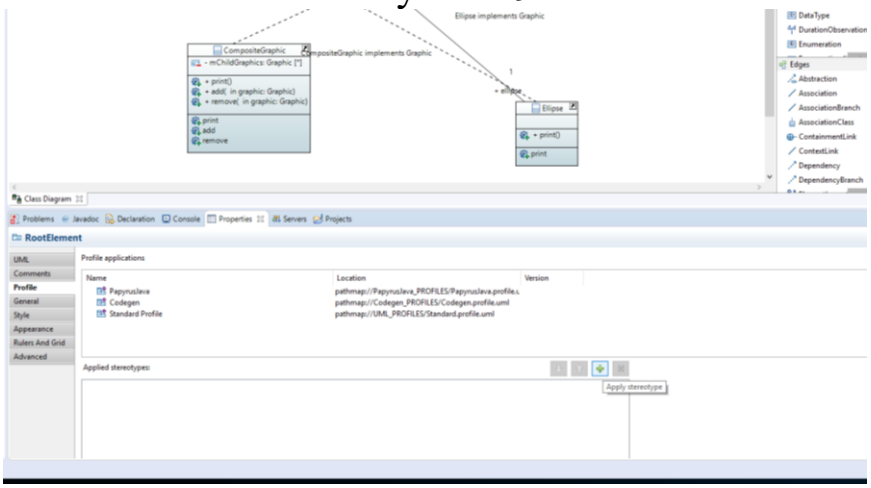

Рисунок 20

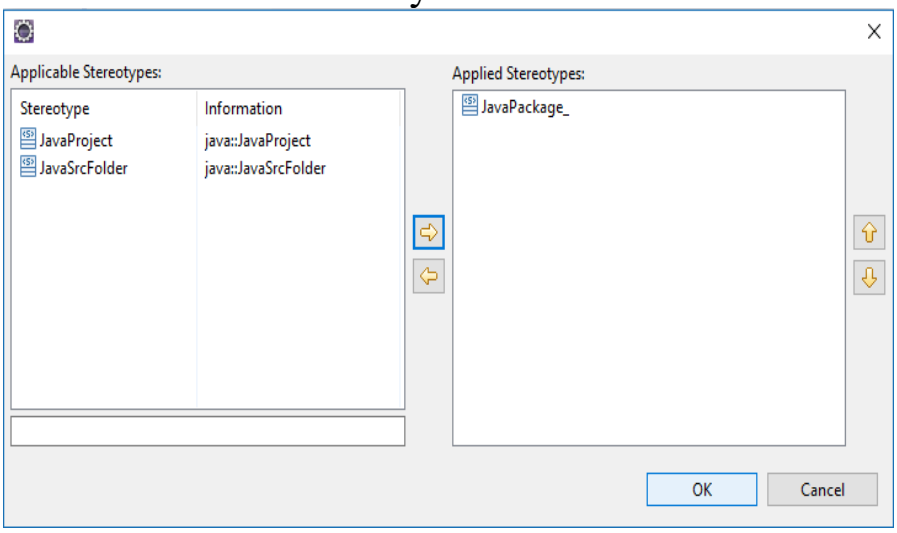

Рисунок 21

7. Сгенерировать Java-код по диаграмме классов. Для этого необходимо нажать правой кнопкой мыши на диаграмму классов и выбрать Generate Java Code из выпадающего меню (рисунок 22). В результате сгенерированные файлы с классами и интерфейсами паттерна будут находиться в папке src в пакете OSCoposite (рисунок 23).

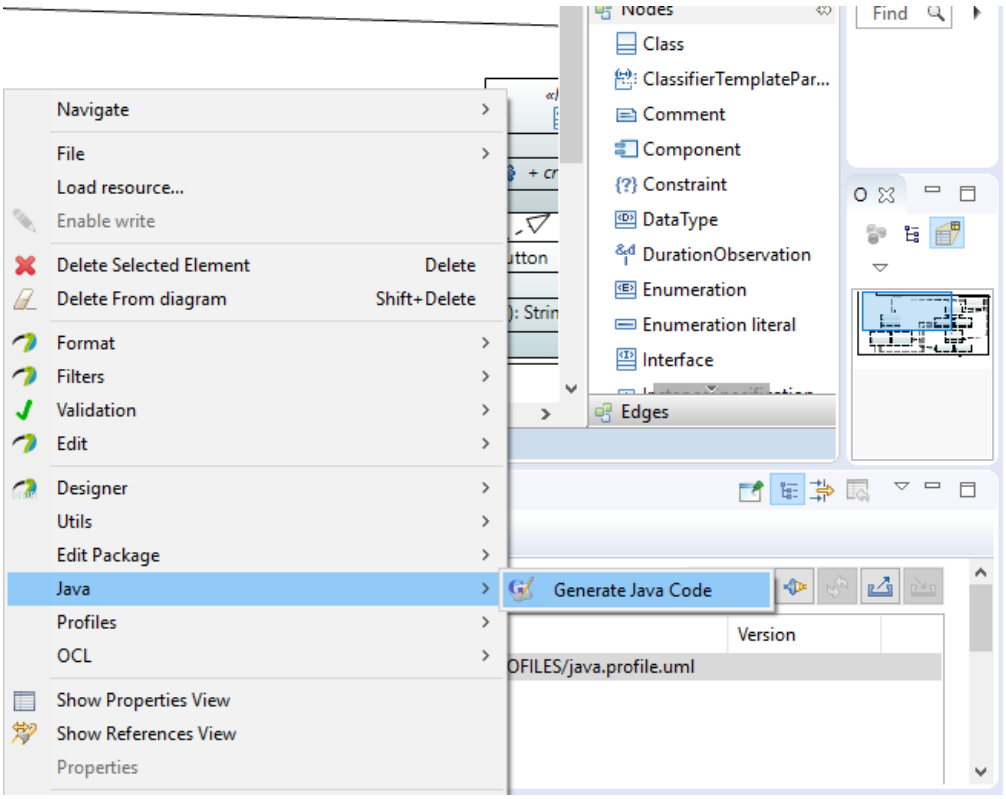

Рисунок 22

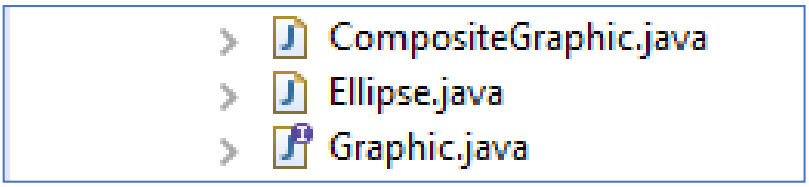

Рисунок 23

8. Необходима корректировка сгенерированного кода абстрактного интерфейса Graphic (рисунок 24) и классов-наследников CompositeGraphic, Ellipse (рисунки 25, 26). Ellipse играет роль листьев в древовидной системе, а CompositeGraphic группирует все листья в единую систему.

```
1
2  public interface Graphic {
3      public void print();
4  }
```

Рисунок 24

```
public class Ellipse implements Graphic {
    public void print() {
        System.out.println("Ellipse");
    }
}
```

Рисунок 25

```
1  import java.util.List;
2  import java.util.ArrayList;
3
4  public class CompositeGraphic implements Graphic {
5      private List<Graphic> mChildGraphics = new ArrayList<Graphic>();
6
7      //Prints the graphic.
8      public void print() {
9          for (Graphic graphic : mChildGraphics) {
10             graphic.print();
11         }
12     }
13
14     //Adds the graphic to the composition.
15     public void add(Graphic graphic) {
16         mChildGraphics.add(graphic);
17     }
18
19     //Removes the graphic from the composition.
20     public void remove(Graphic graphic) {
21         mChildGraphics.remove(graphic);
22     }
23 }
```

Рисунок 26

9. Далее нужно создать в проекте класс ComposieExaple для тестирования паттерна проектирования «Компоновщик» (рисунки 27-29).

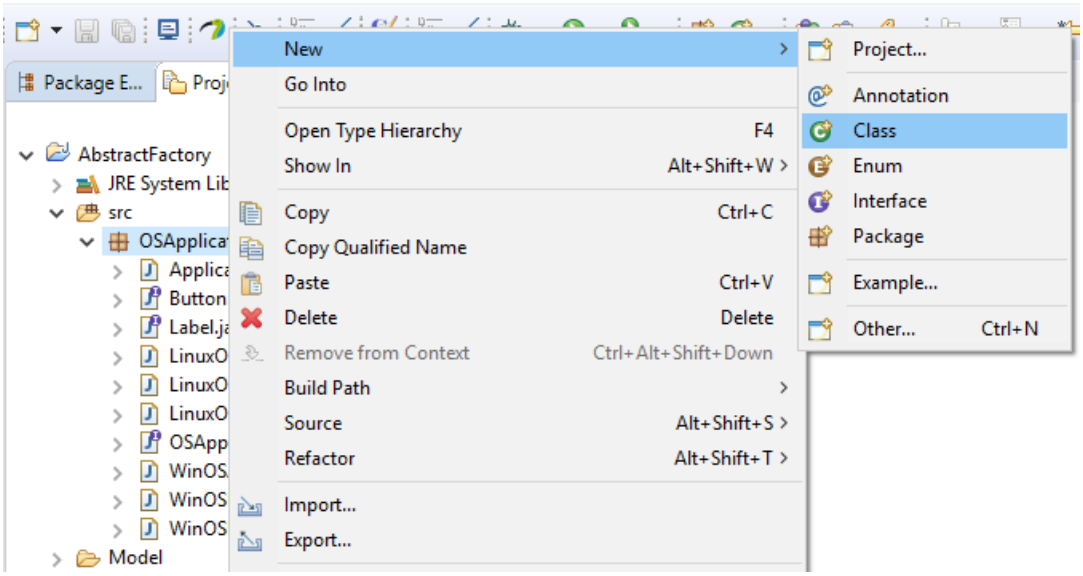

Рисунок 27

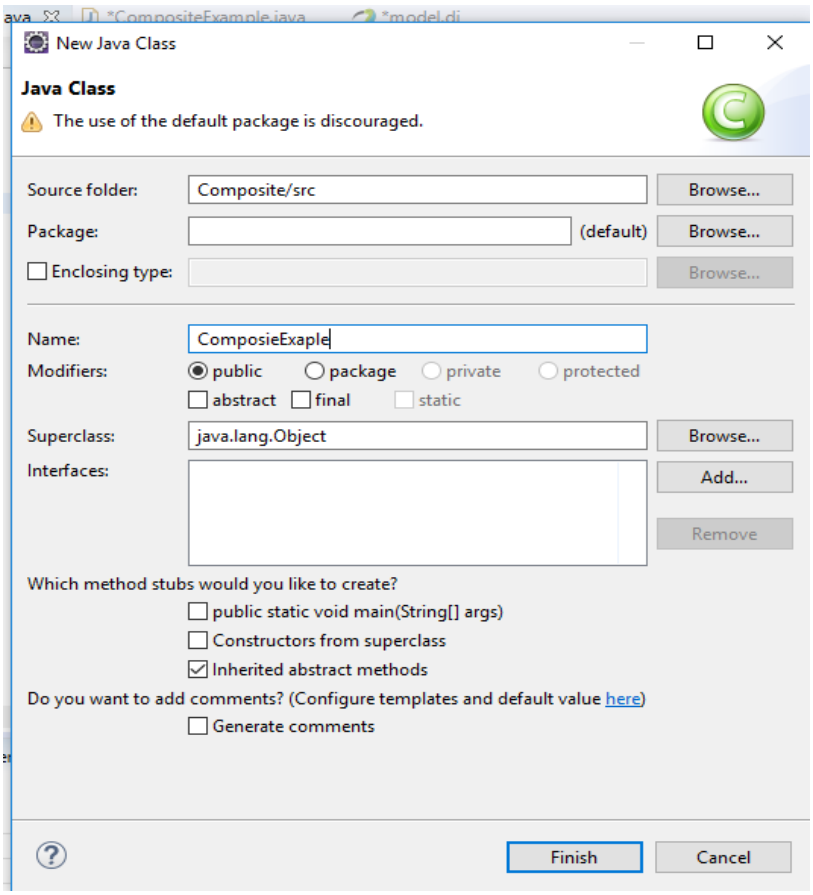

Рисунок 28

```
public class CompositeExample {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
            Ellipse ellipse1 = new Ellipse();
            Ellipse ellipse2 = new Ellipse();
            Ellipse ellipse3 = new Ellipse();
            Ellipse ellipse4 = new Ellipse();

            //Initialize three composite graphics
            CompositeGraphic graphic = new CompositeGraphic();
            CompositeGraphic graphic1 = new CompositeGraphic();
            CompositeGraphic graphic2 = new CompositeGraphic();

            //Composes the graphics
            graphic1.add(ellipse1);
            graphic1.add(ellipse2);
            graphic1.add(ellipse3);

            graphic2.add(ellipse4);

            graphic.add(graphic1);
            graphic.add(graphic2);

            //Prints the complete graphic (four times the string "Ellipse").
            graphic.print();
    }

}
```

Рисунок 29

10. Для компиляции класса ComposieExaple нажмите кнопку Run , чтобы запустить тестирование программы. В окне Console можно увидеть результаты тестирования (рисунок 30).

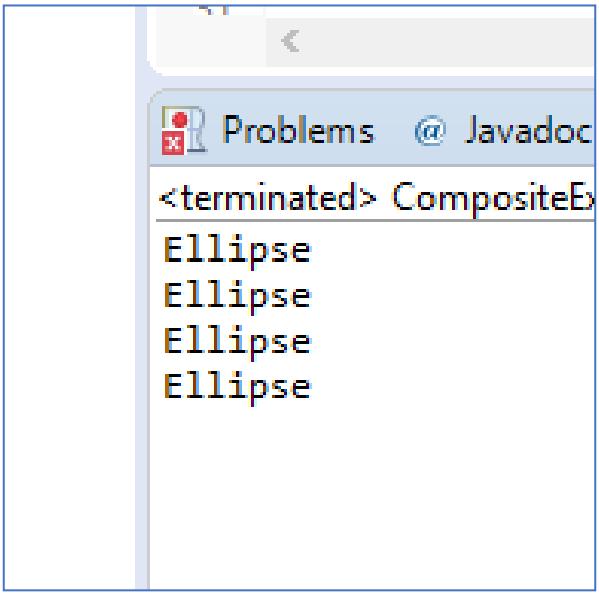

Рисунок 30

4 Содержание отчета по практической работе

Отчет по практической работе включает:

- титульный лист;
- условие задания;
- диаграммы классов для решения задачи;
- диаграммы последовательности для решения задачи;
- текст программы реализации паттерна проектирования «Компоновщик» для индивидуального задания;
- результаты тестирования программы.

5 Вопросы к защите практической работы

1. Что такое паттерн проектирования?
2. Для чего предназначен паттерн проектирования «Компоновщик»?
3. К какому типу паттернов проектирования относится «Компоновщик»?
4. Какие классы/интерфейсы являются участниками «Компоновщик»?
5. Назовите родственные для «Компоновщик» паттерны проектирования.
6. Выделите достоинства и недостатки этого паттерна проектирования.
7. В каком случае необходимо наследовать интерфейс доступа?

6 Индивидуальные задания

При разработке использовать паттерн проектирования «Компоновщик».

1. Есть три типа героев – король, воин и маг. У каждого своя атака, тип оружия и наносимый урон. Каждый игрок может собирать свою армию, на основе 3 типов героев. Реализовать программный продукт, позволяющий формировать армию.
2. Существуют различные легковые машины, которые используют разные источники энергии: электричество, бензин, газ. Есть гибридные автомобили. Каждый автосалон продает различные автомобили.

Реализовать программный продукт, позволяющий каждому автосалону сформировать список автомобилей с разными источниками энергии.

3. Есть различные продукты, каждый продукт имеет марку, название и цену. Продукты продаются в магазинах. Реализовать программный продукт, позволяющий добавлять в магазин различные продукты.

4. Для того, чтобы пойти в школу, ученики собирают портфель. В каждом портфеле должны быть дневник, пенал, тетради и учебники. Реализовать программный продукт, позволяющий сформировать портфель ученика.

5. В офисе работают люди. У каждого человека есть свои трудовые обязанности. Реализовать программный продукт, который формирует работников в офисе.

6. В каждой квартире есть мебель. Каждая мебель имеет название и занимает определенную площадь. Реализовать программный продукт, который позволяет разместить мебель в комнате

7. Рабочий стол в компьютере состоит из иконок. Каждая иконка имеет название и размер, который она занимает на жестком диске. Реализовать программный продукт, который позволит сформировать иконки на рабочем столе.

8. Для того, чтобы пойти в поход, турист собирает рюкзак. В рюкзаке должны быть спальный мешок, белье, куртка и аптечка. Реализовать программный продукт, позволяющий сформировать рюкзак туриста.

9. Есть различные продукты, каждый продукт имеет название, вес, и срок хранения. Продукты размещаются в холодильнике. Реализовать программный продукт, позволяющий разместить в холодильнике различные продукты.

10. В квартире есть кухонный шкаф для бытовой техники, хранящейся в коробках. Каждая коробка имеет название и занимает определенную площадь. Реализовать программный продукт, который позволяет разместить коробки в шкафу.

11. Существуют различные грузовые машины, которые используют разные источники энергии: электричество, бензин, газ. Есть гибридные автомобили. Каждый автосалон продает различные автомобили. Реализовать программный продукт, позволяющий каждому автосалону сформировать список автомобилей с разными источниками энергии.

12. На городской автобазе имеется автотранспорт для перевозки пассажиров. Автотранспорт использует разные источники энергии:

электричество, бензин, газ. Есть гибридные автобусы. Реализовать программный продукт, позволяющий автобазе сформировать список автомобилей с разными источниками энергии.

13. Для того, чтобы поехать на отдых, турист собирает чемодан. В чемодане должны быть уложены белье, купальник, куртка, аптечка. Реализовать программный продукт, позволяющий сформировать чемодан туриста.

14. В квартире есть антресоль для обуви, хранящейся в коробках. Каждая коробка имеет название и занимает определенную площадь. Реализовать программный продукт, который позволяет разместить коробки на антресоли.

15. В офисе есть мебель: компьютерные столы, компьютерные кресла, шкафы для документов. Каждая мебель имеет название и занимает определенную площадь. Реализовать программный продукт, который позволяет разместить мебель в офисе

Список использованных источников

1. Ларман, К. Применение UML 2.0 и шаблонов проектирования/ К. Ларман. - М.: Издательский дом «Вильямс», 2013. -736 с. - Текст: непосредственный
2. Османи, Э. Паттерны для масштабируемых JavaScript-приложений/ Э. Османи. - М.: Техносфера, 2015. -188 с. - Текст: непосредственный
3. Фримен Э. Паттерны проектирования/ Э. Фримен, Э. Фримен, К. Сьерра, Б. Бейтс. – СПб.: Питер, 2016. -653 с. - Текст: непосредственный
4. Перл, И. А. Введение в методологию программной инженерии : учебное пособие : [16+] / И. А. Перл, О. В. Калёнова. – Санкт-Петербург : Университет ИТМО, 2019. – 53 с. : ил., схем. – Режим доступа: по подписке. – URL: https://biblioclub.ru/index.php?page=book&id=566776 (дата обращения: 28.08.2025). – Бибиогр. в кн. – Текст : электронный.
5. Шуваев, А. В. Программная инженерия : учебное пособие для магистрантов направления подготовки 09.04.02 – Информационные системы и технологии : [16+] / А. В. Шуваев ; Ставропольский государственный аграрный университет, Кафедра информационных систем. – Ставрополь: Ветеран, 2020. – 84 с. : ил., табл. – Режим доступа: по подписке. – URL:

https://biblioclub.ru/index.php?page=book&id=700960 (дата обращения: 28.08.2025). – Библиогр. в кн. – Текст : электронный.