МИНОБРНАУКИРОССИИ

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

Кафедра программной инженерии

УТВЕРЖДАЮ
Проректор по учебной работе
_____ О.Г. Локтионова
«14» 02 2018 г.

ПАТТЕРН ПРОЕКТИРОВАНИЯ «КОМПОНОВЩИК»

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

Курск 2018

УДК 004.65

Составители: Т.М. Белова, В.Г. Белов

Рецензент
Кандидат технических наук, доцент кафедры программной инженерии ЮЗГУ И.Н. Ефремова

Паттерн проектирования «Компоновщик»: методические указания по выполнению лабораторной 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, В.Г. Белов, – Курск, 2018. – 24 с.: ил. 30.

Изложена последовательность действий по разработке и применению паттерна проектирования «Компоновщик» при работе в интегрированной среде разработки Eclipse.

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 14.02.18. Формат 60x84 1/16.
Усл. печ. л. 1,3. Уч.-изд. л. 1,2. Тираж 50 экз. Заказ 790. Бесплатно.
Юго-Западный государственный университет
305040, Курск, ул.50 лет Октября, 94.

Содержание

1 Цель лабораторной работы 4
2 Основные понятия 4
3 Порядок выполнения лабораторной работы 8
4 Содержание отчета по лабораторной работе 23
5 Вопросы к защите лабораторной работы 23
6 Индивидуальные задания 23
Список использованных источников 24

1 Цель лабораторной работы

Целью лабораторной 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).

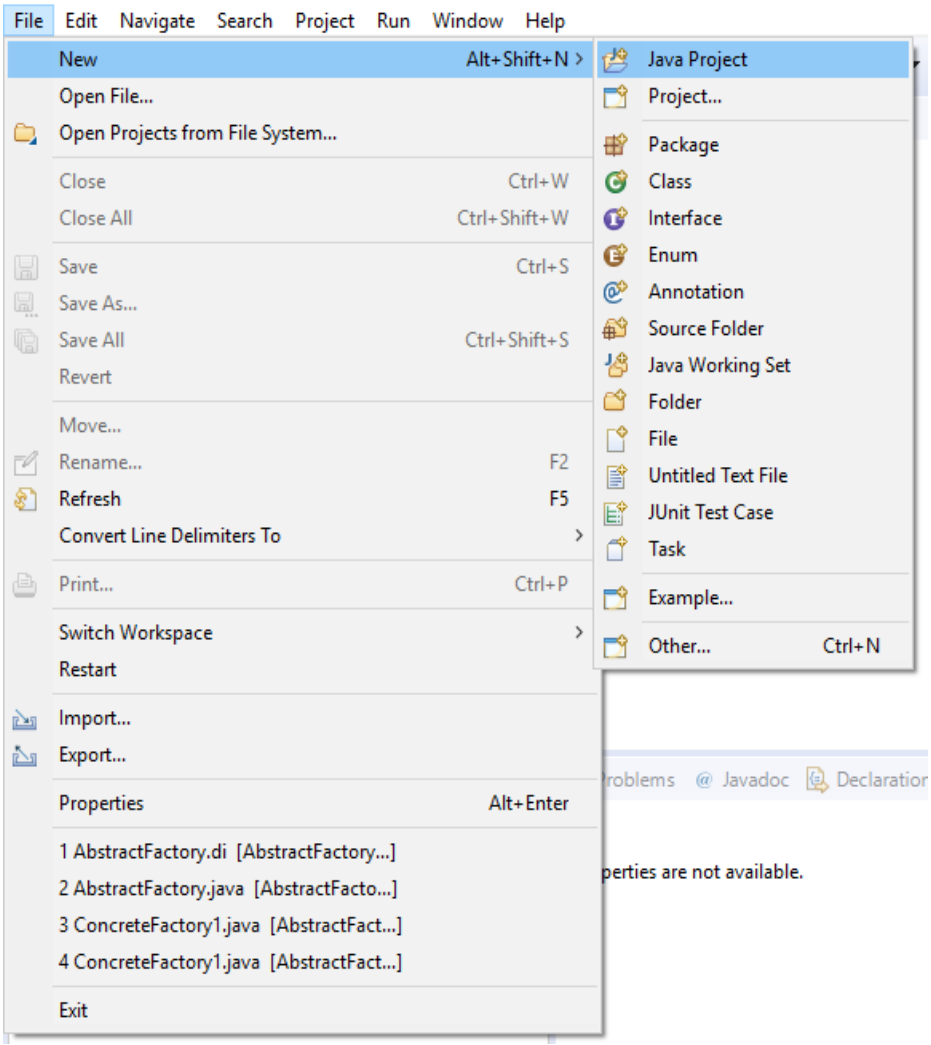

Рисунок 1

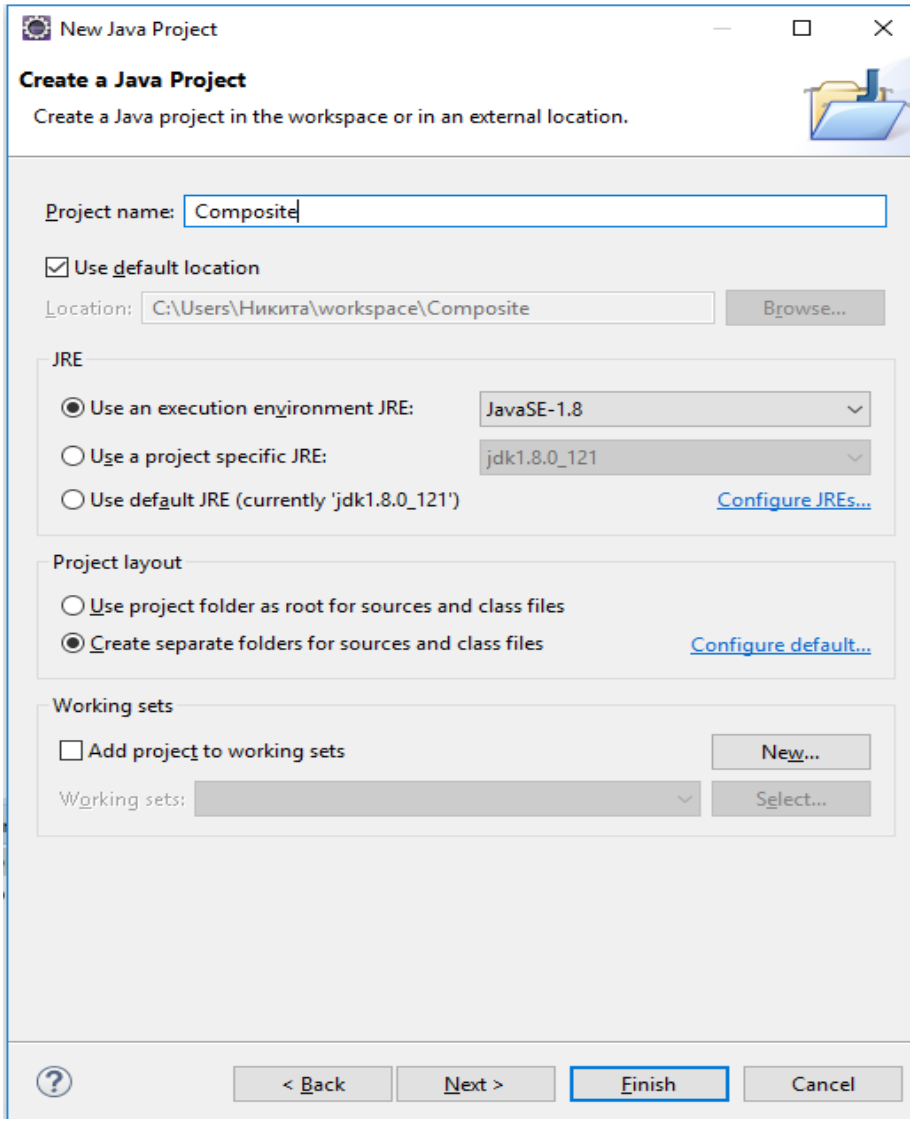

Рисунок 2

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).

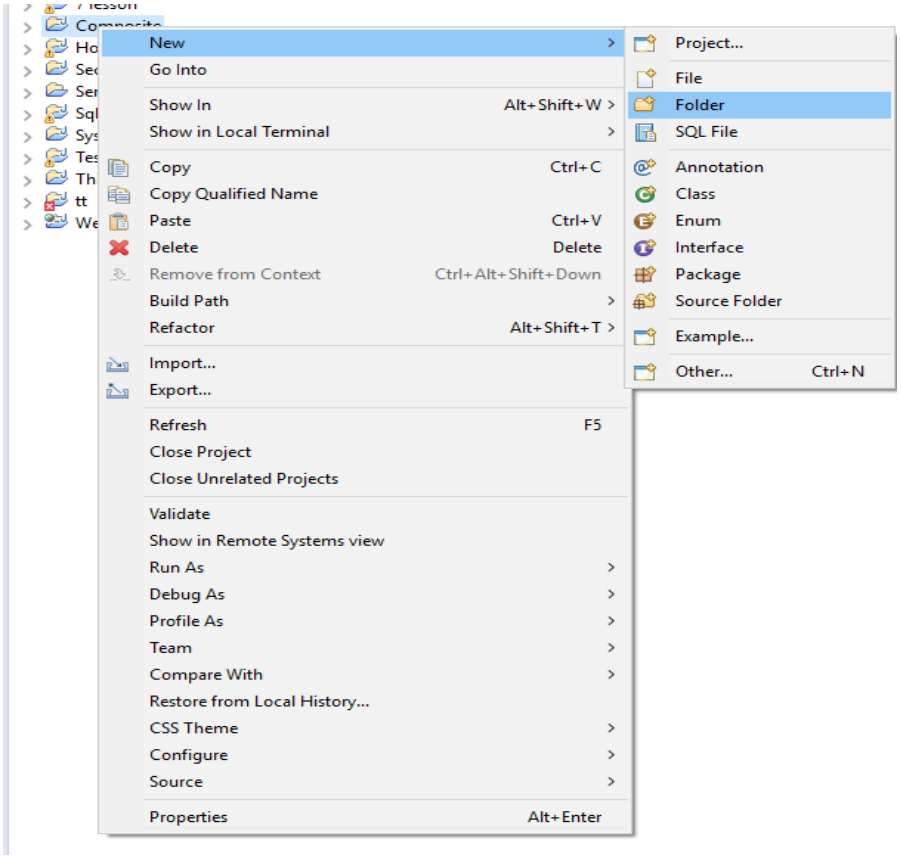

Рисунок 3

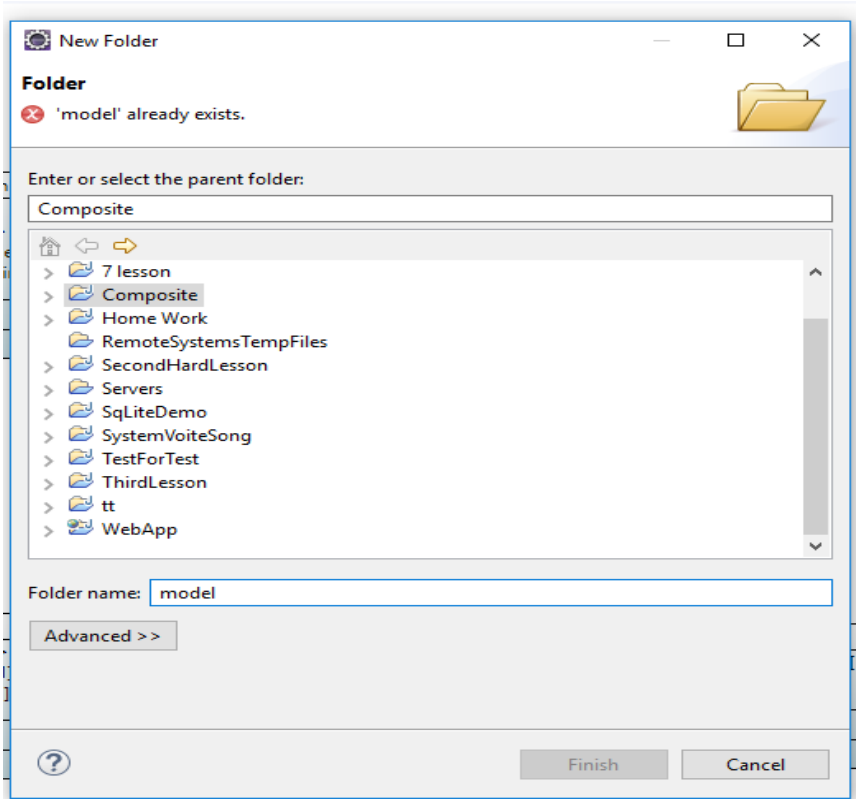

Рисунок 4

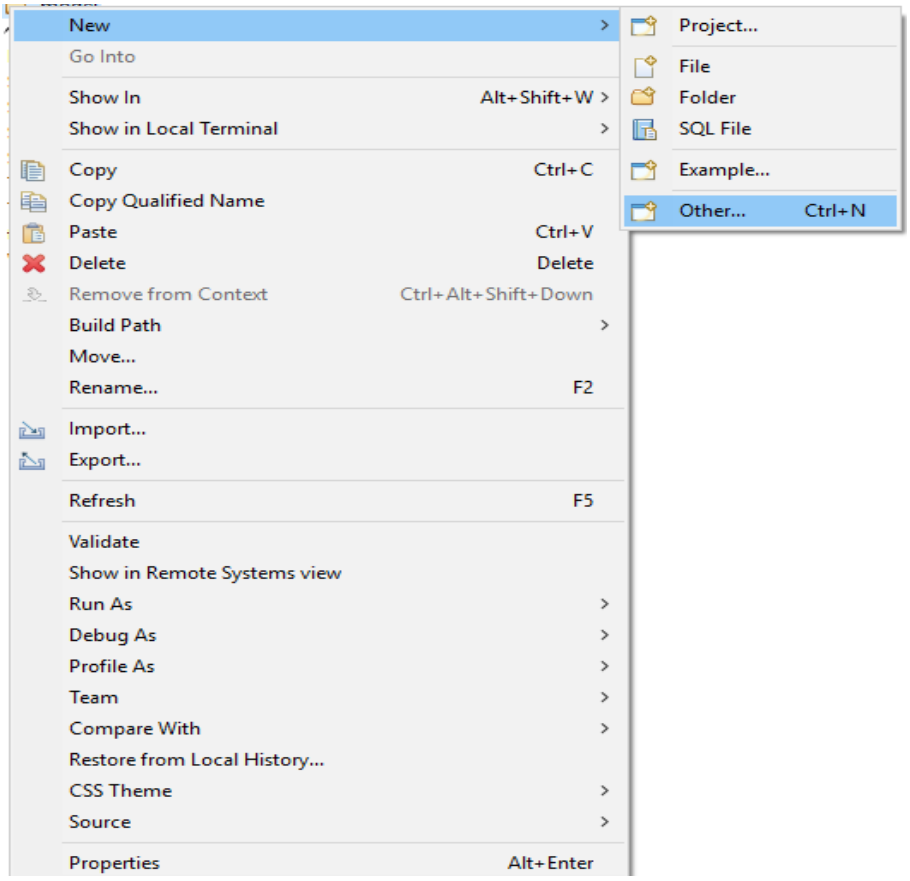

Рисунок 5

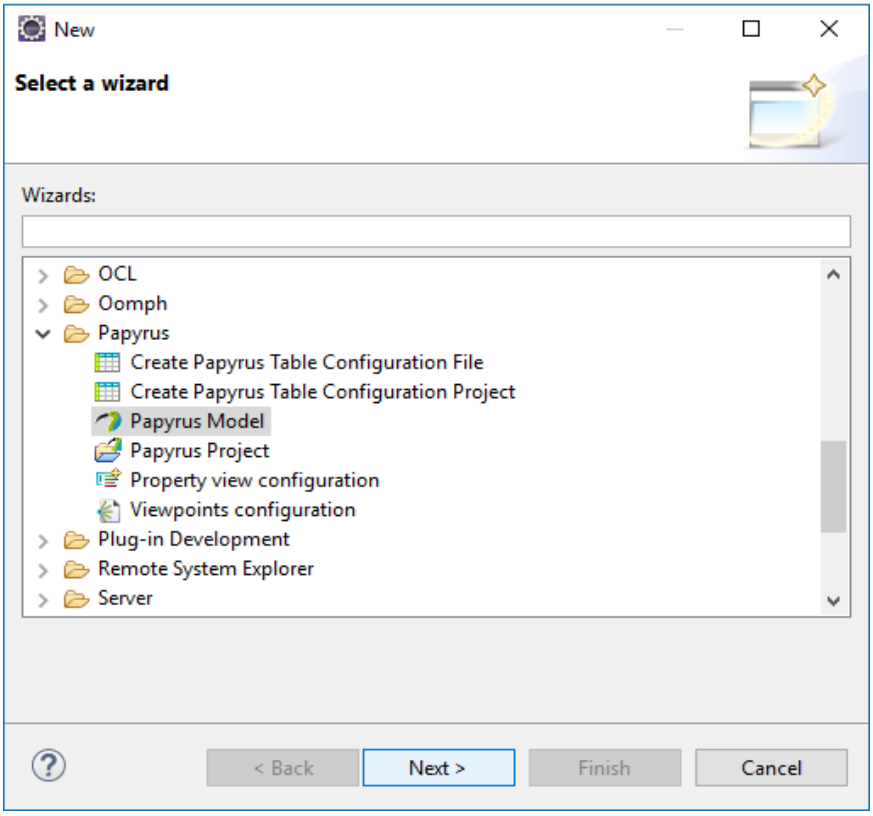

Рисунок 6

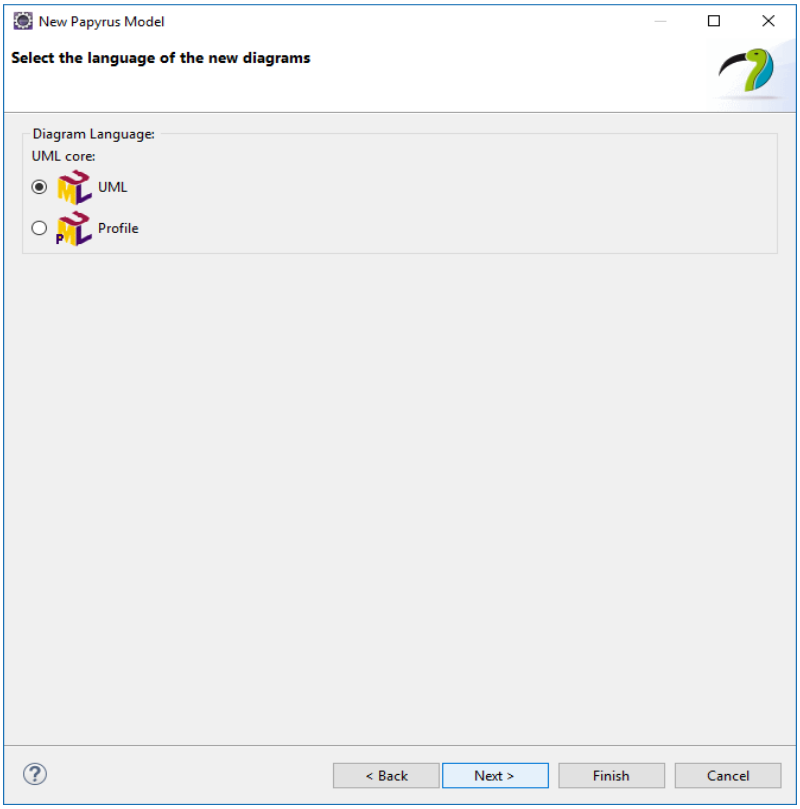

Рисунок 7

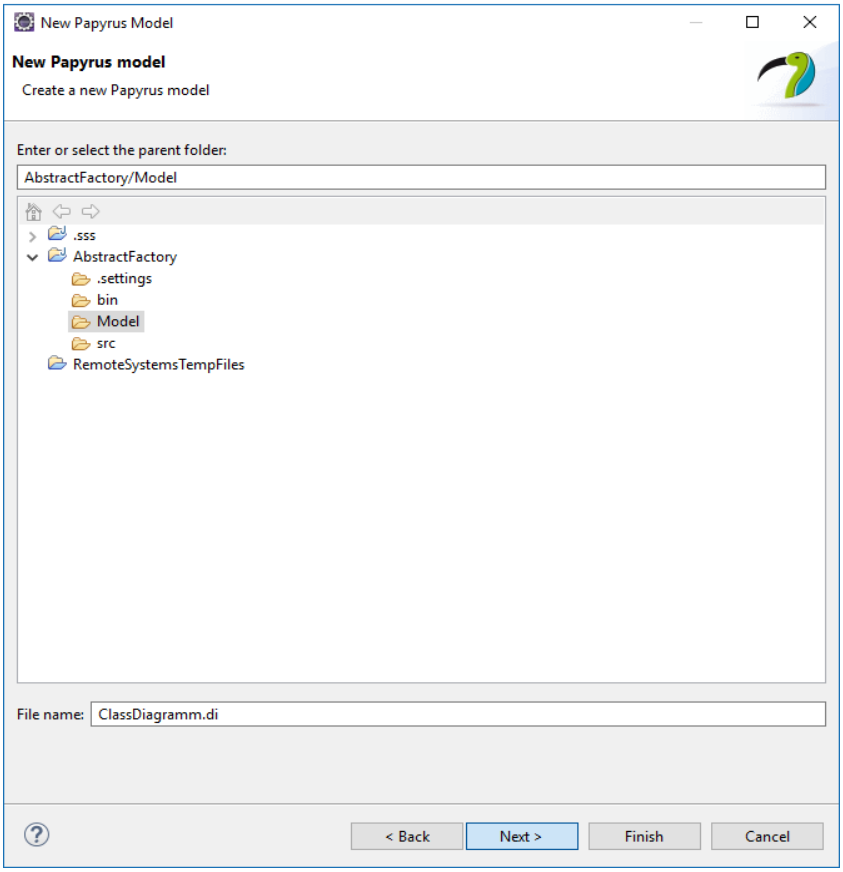

Рисунок 8

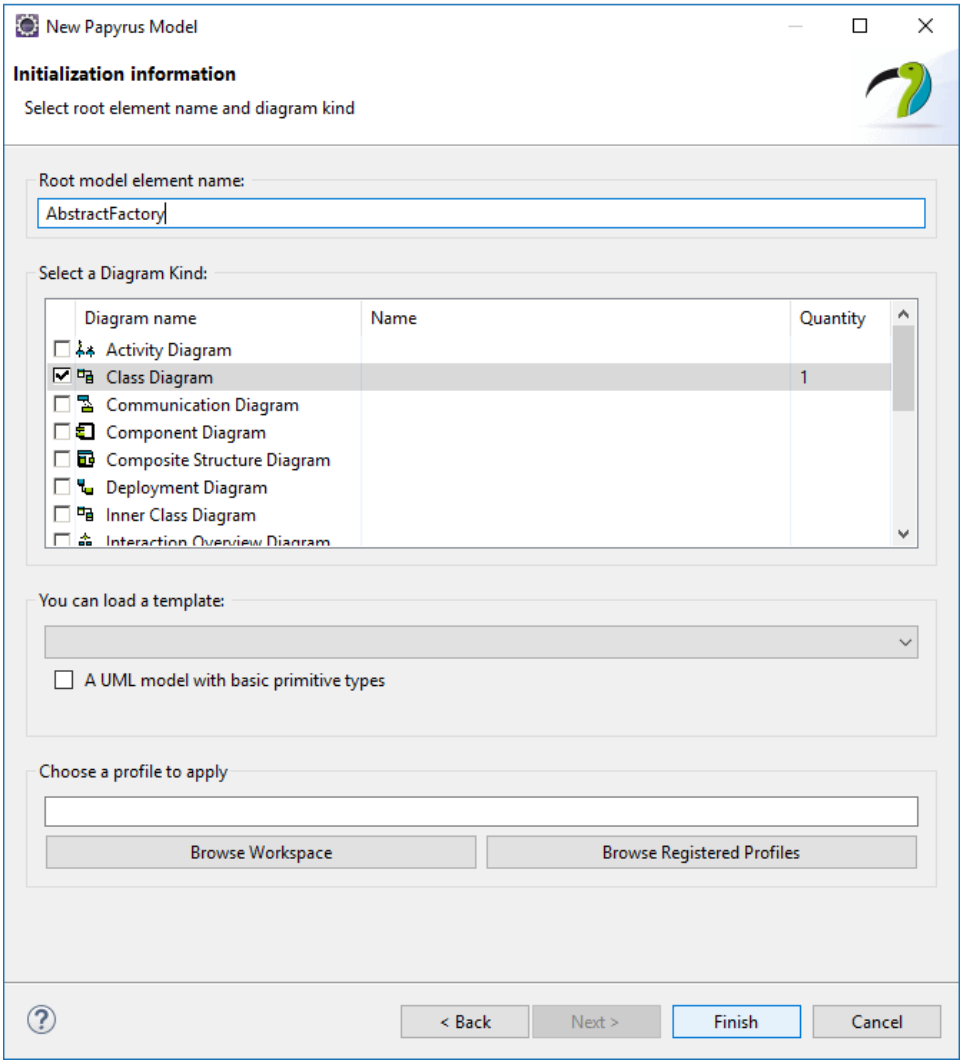

Рисунок 9

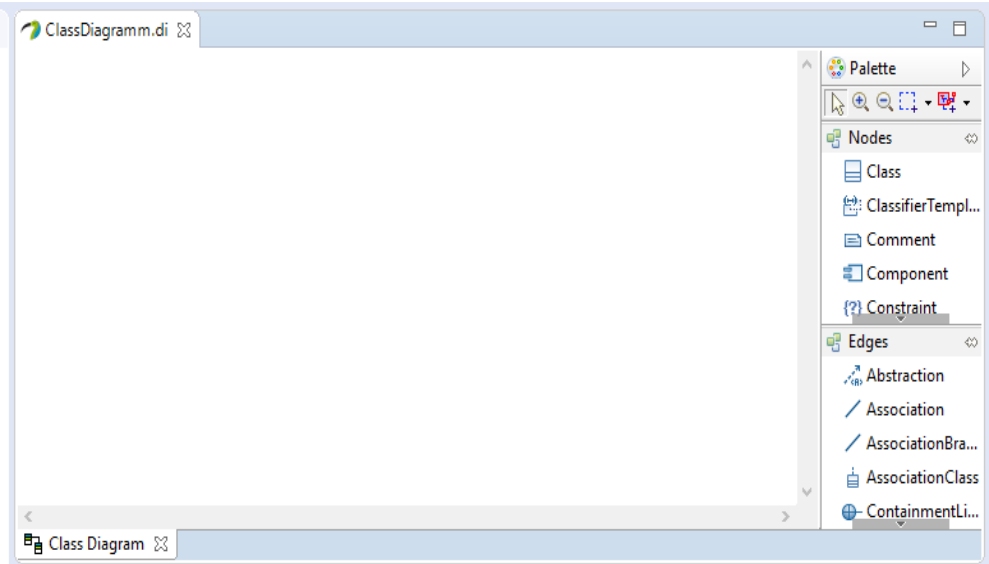

Рисунок 10

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

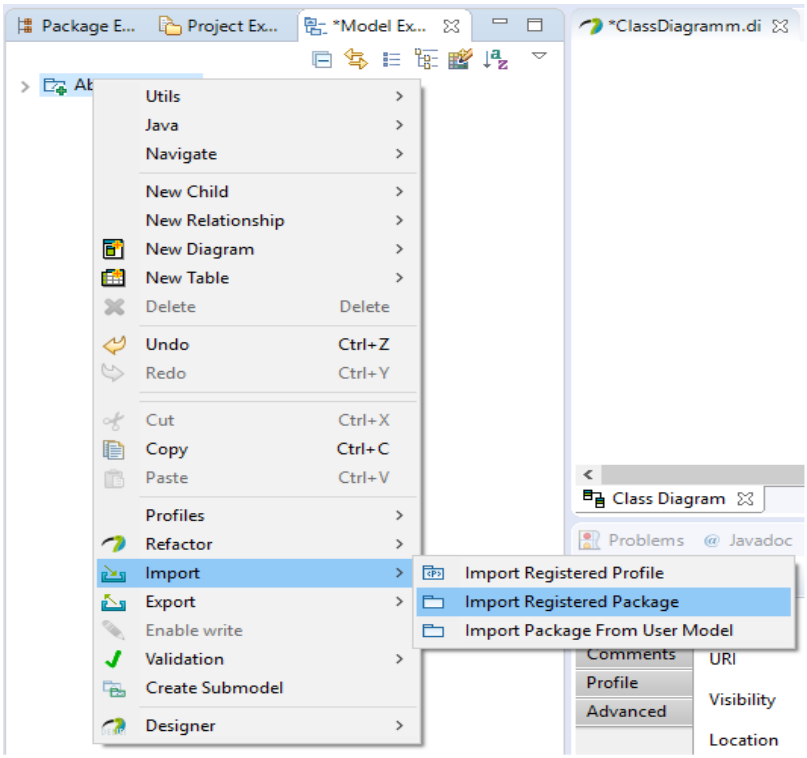

Рисунок 11

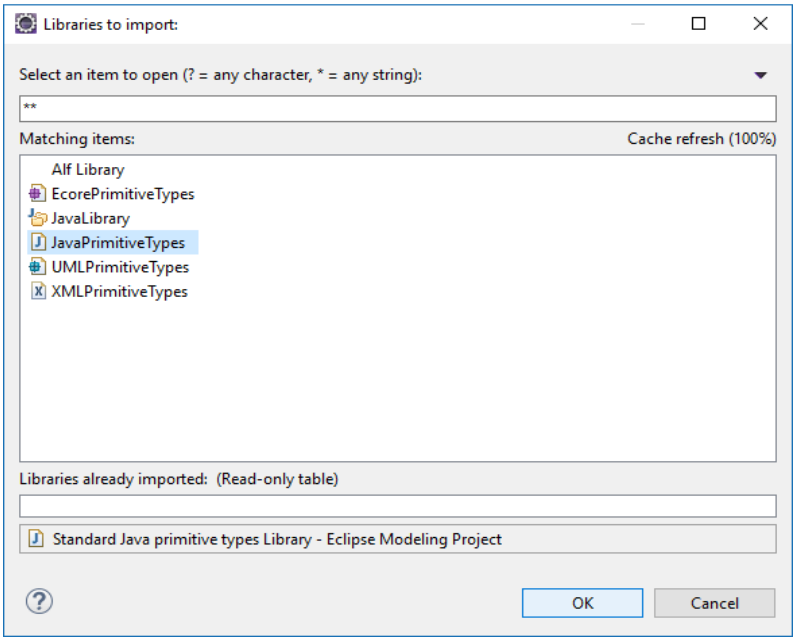

Рисунок 12

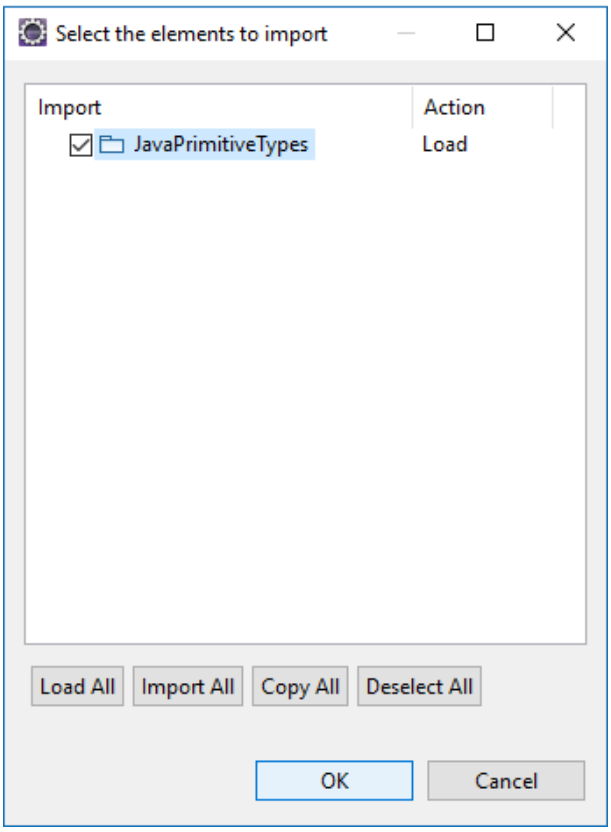

Рисунок 13

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

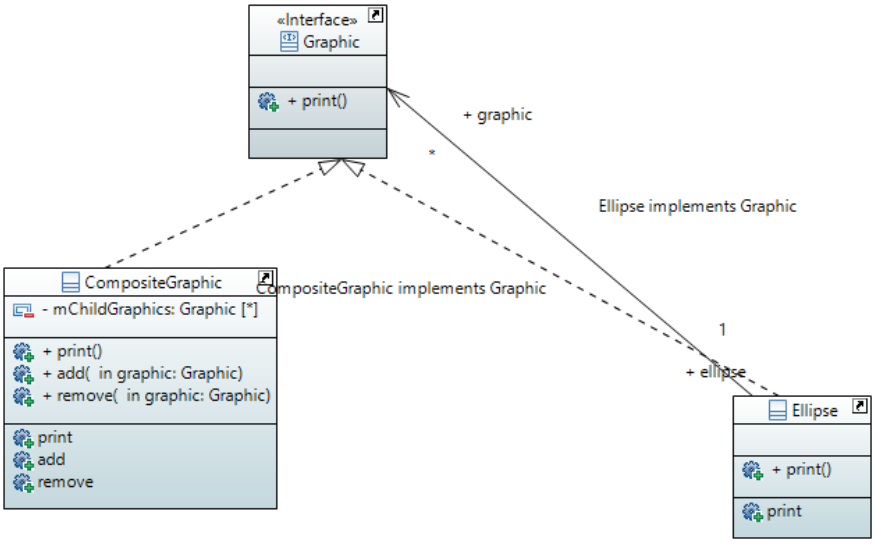

Рисунок 14

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

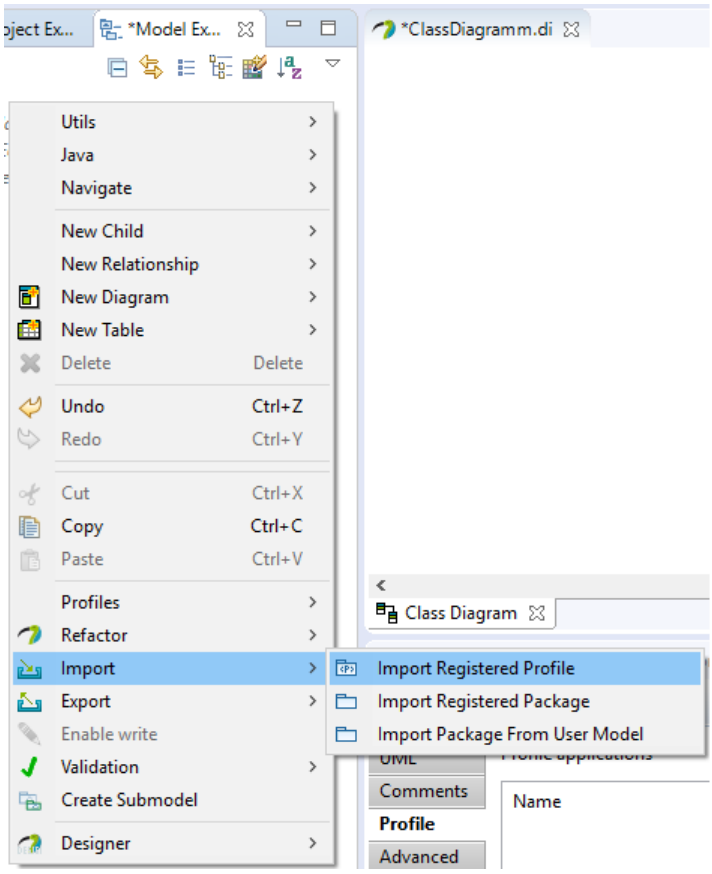

Рисунок 15

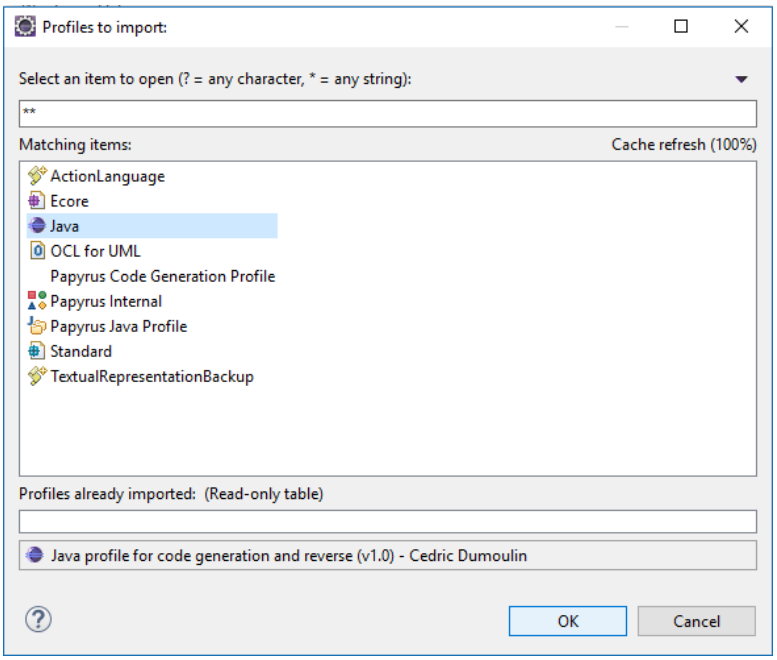

Рисунок 16

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

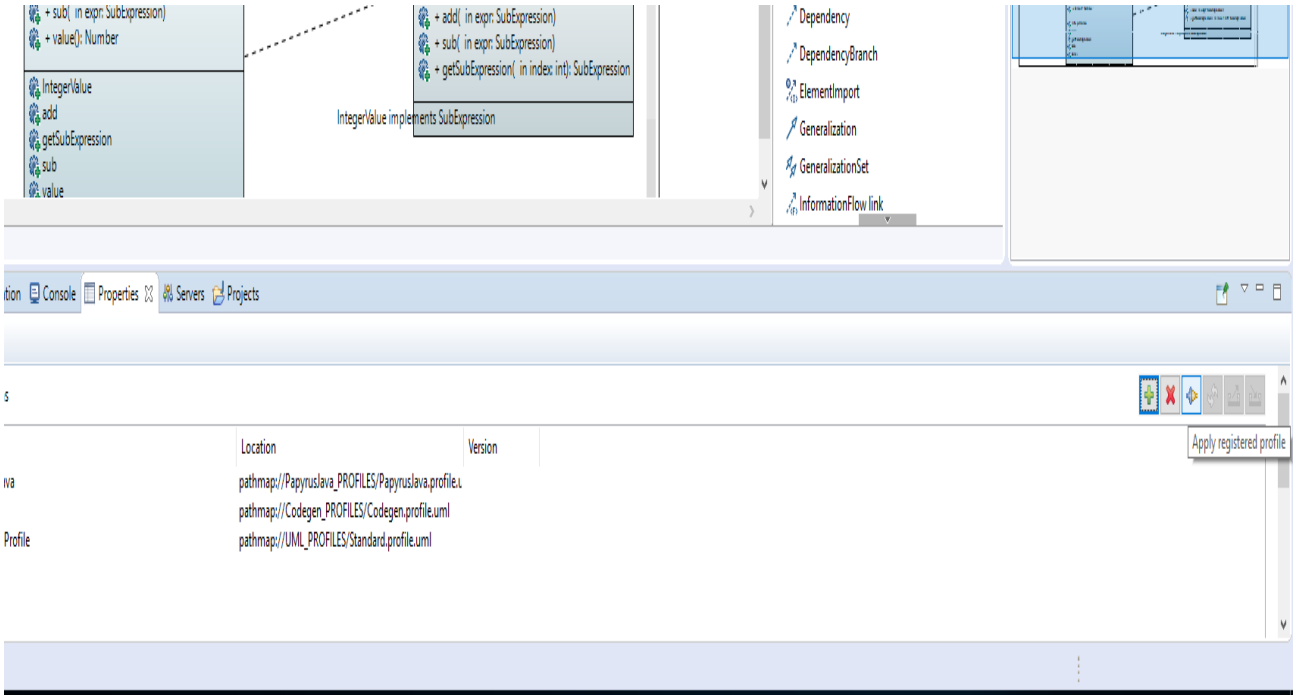

Рисунок 17

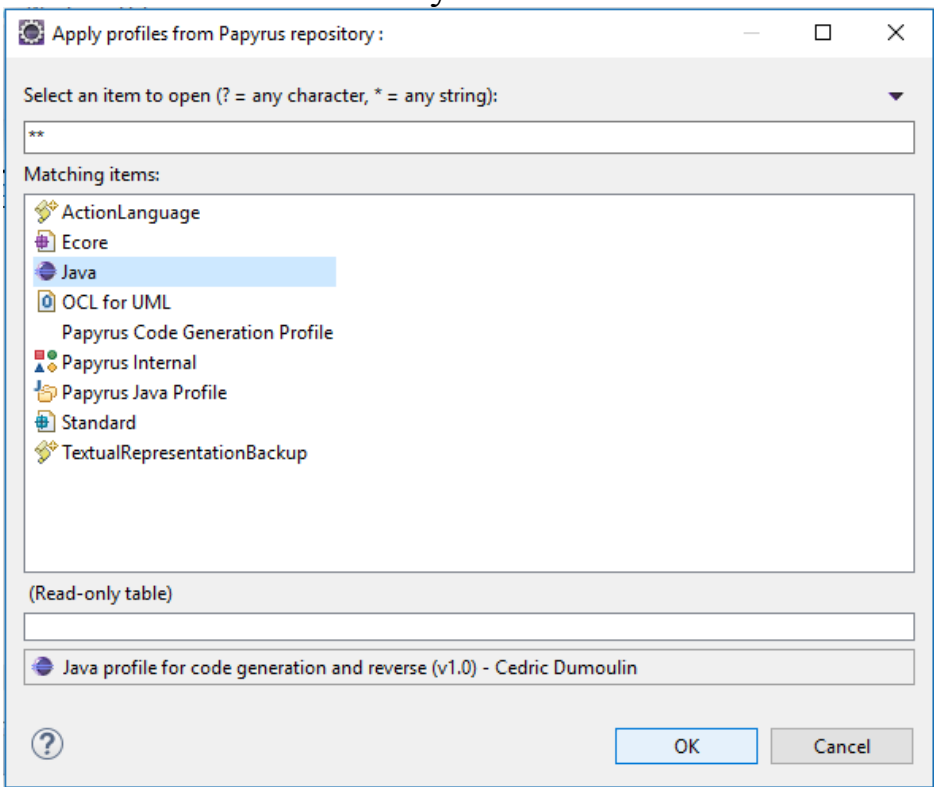

Рисунок 18

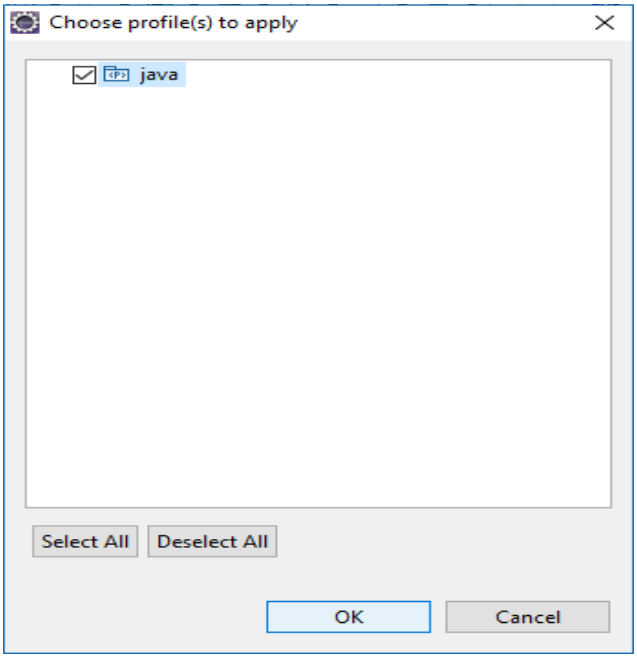

Рисунок 19

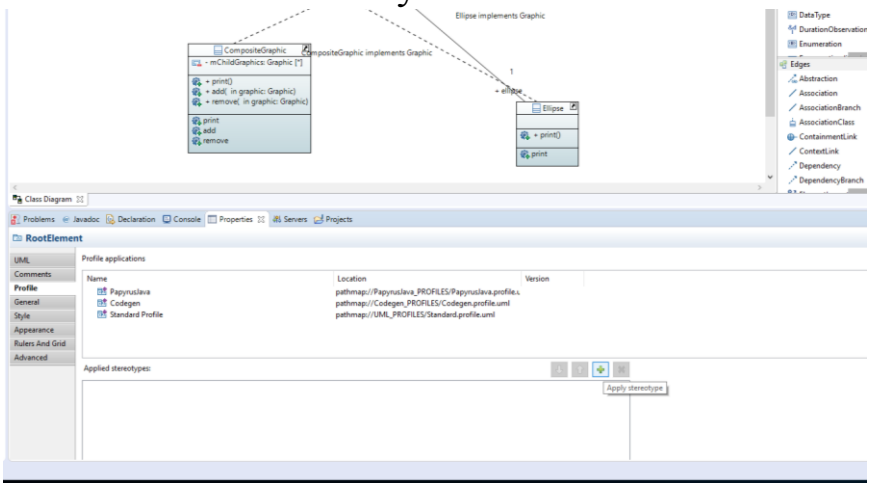

Рисунок 20

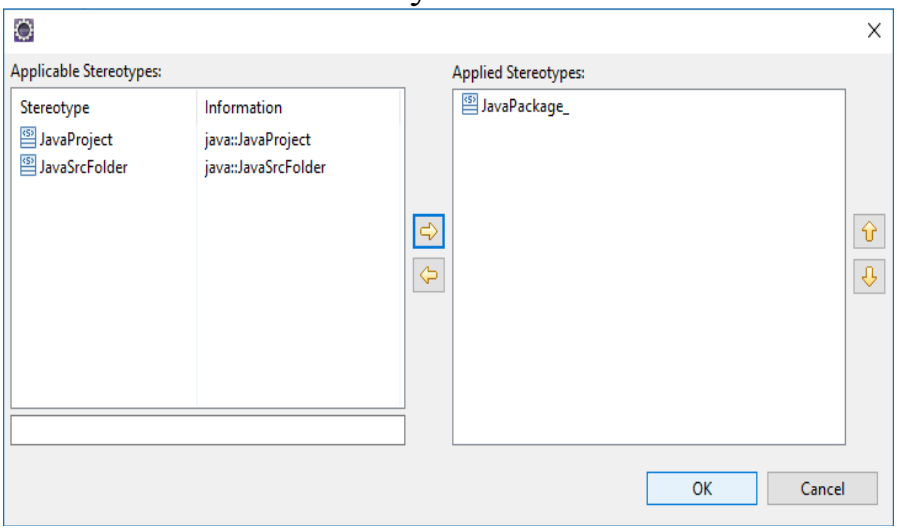

Рисунок 21

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).

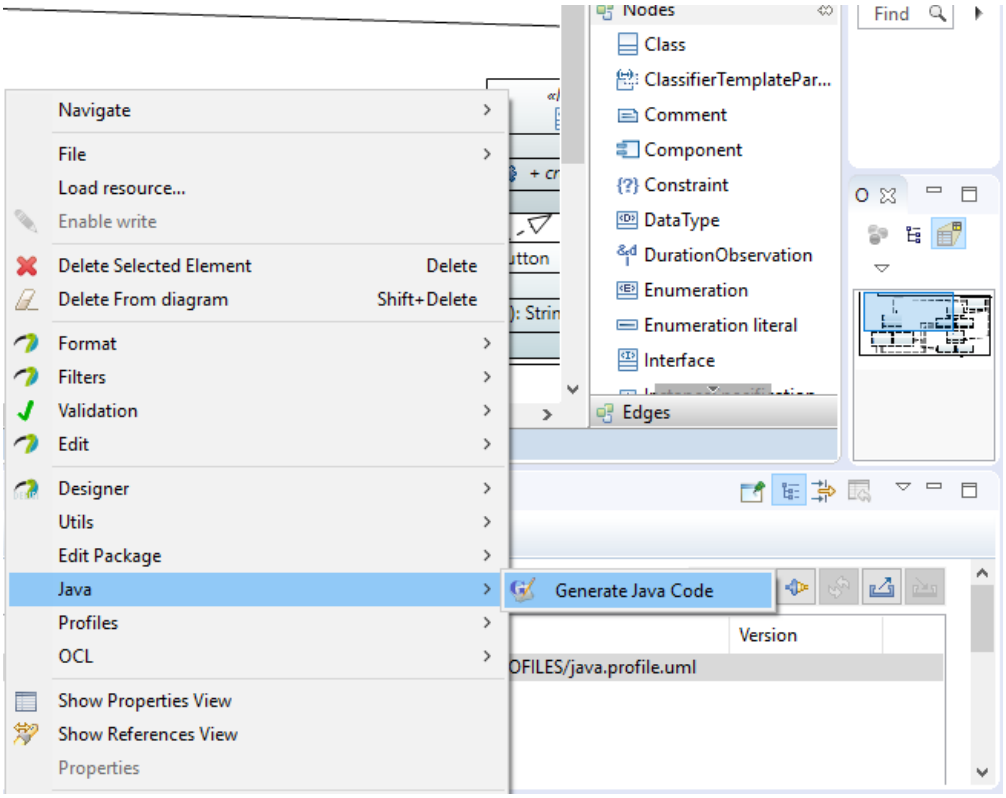

Рисунок 22

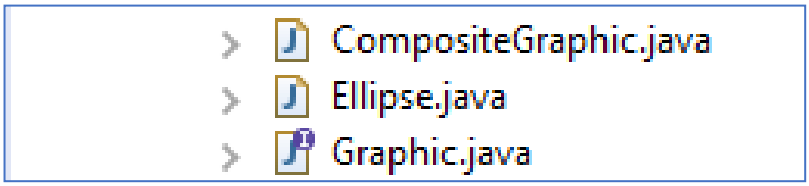

Рисунок 23

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

```
1
2  public class Ellipse implements Graphic {
3      public void print() {
4          System.out.println("Ellipse");
5      }
6  }
7
```

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).

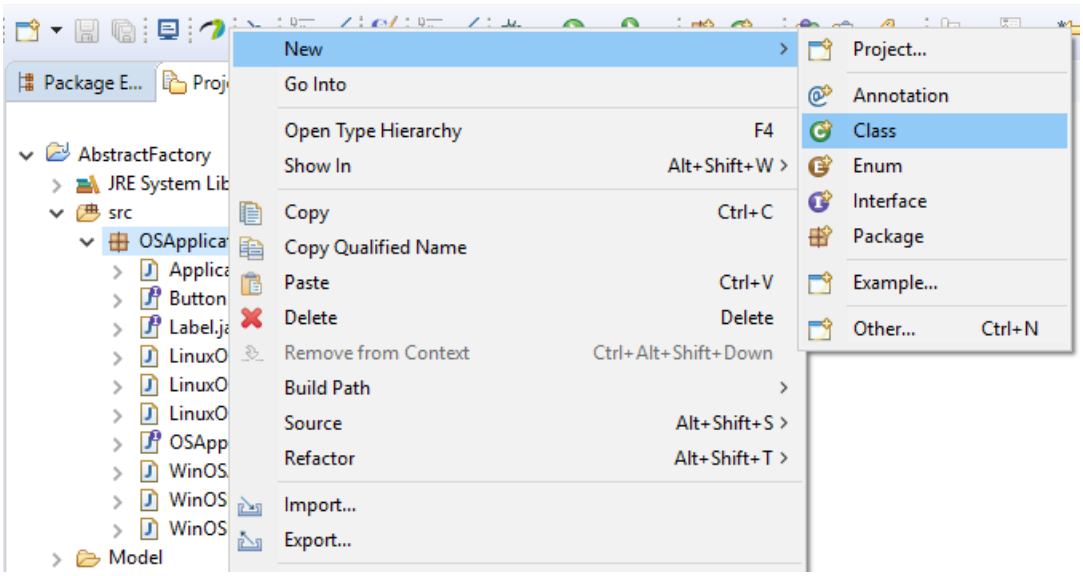

Рисунок 27

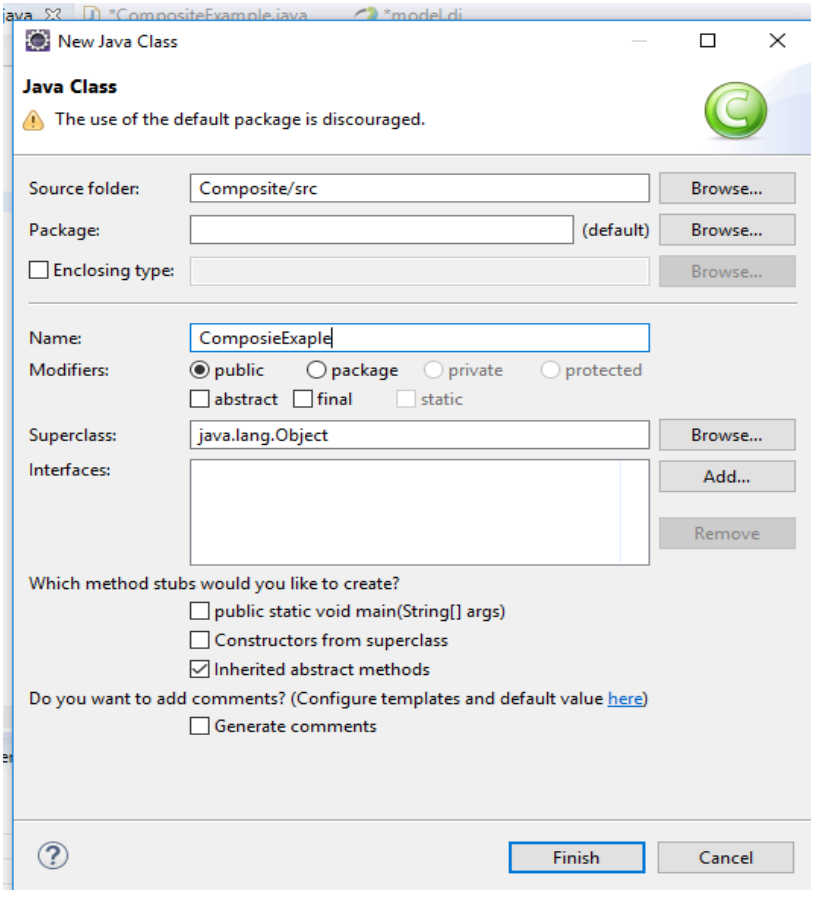

Рисунок 28

```
1
2  public class CompositeExample {
3
4      public static void main(String[] args) {
5          // TODO Auto-generated method stub
6              Ellipse ellipse1 = new Ellipse();
7              Ellipse ellipse2 = new Ellipse();
8              Ellipse ellipse3 = new Ellipse();
9              Ellipse ellipse4 = new Ellipse();
0
1              //Initialize three composite graphics
2              CompositeGraphic graphic = new CompositeGraphic();
3              CompositeGraphic graphic1 = new CompositeGraphic();
4              CompositeGraphic graphic2 = new CompositeGraphic();
5
6              //Composes the graphics
7              graphic1.add(ellipse1);
8              graphic1.add(ellipse2);
9              graphic1.add(ellipse3);
0
1              graphic2.add(ellipse4);
2
3              graphic.add(graphic1);
4              graphic.add(graphic2);
5
6              //Prints the complete graphic (four times the string "Ellipse").
7              graphic.print();
8      }
9
0  }
```

Рисунок 29

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

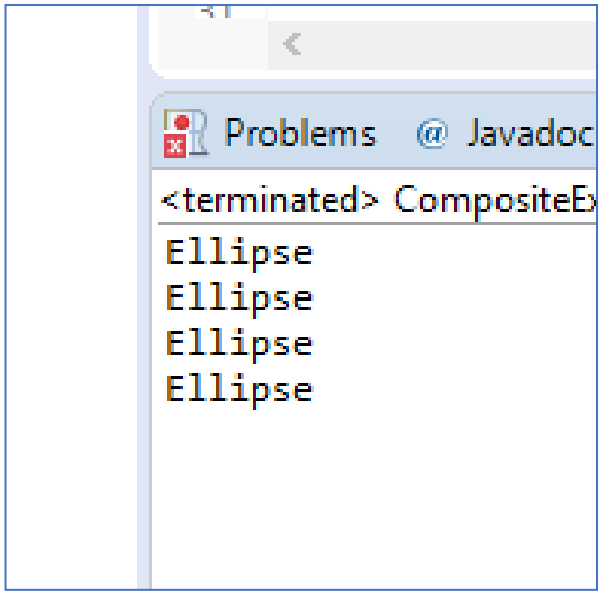

Рисунок 30

4 Содержание отчета по 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 «Компоновщик» для индивидуального задания;
- результаты тестирования программы.

5 Вопросы к защите лабораторной 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

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. При разработке использовать паттерн проектирования «Компоновщик».
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. При разработке использовать паттерн проектирования «Компоновщик».

3. Есть различные продукты, каждый продукт имеет марку, название и цену. Продукты продаются в магазинах. Реализовать программный продукт, позволяющий добавлять в магазин различные продукты. При разработке использовать паттерн проектирования «Компоновщик».

4. Для того, что бы пойти в школу, ученики собирают портфель. В каждом портфеле должны быть дневник, пенал, тетради и учебники. Реализовать программный продукт, позволяющий сформировать портфель ученика. При разработке использовать паттерн проектирования «Компоновщик».

5. В офисе работают люди. У каждого человека есть свои трудовые обязанности. Реализовать программный продукт, который формирует работников в офисе. При разработке использовать паттерн проектирования «Компоновщик».

6. В каждой квартире есть мебель. Каждая мебель имеет название и занимает определенную площадь. Реализовать программный продукт, который позволяет разместить мебель в комнате. При разработке использовать паттерн проектирования «Компоновщик».

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. При разработке использовать паттерн проектирования «Компоновщик».

Список использованных источников

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

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

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