Министерство образования Республики Беларусь

Учреждение образования  
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет инновационного непрерывного образования

Кафедра проектирования информационных компьютерных систем   
  
Дисциплина "Объектно-ориентированное программирование"

|  |  |
| --- | --- |
|  | К защите допустить:  Руководитель курсовой работы старший преподаватель кафедры  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.В.Михалькевич  27.04.2020 |

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовой работе   
на тему

### Разработка компьютерной игры "Пазл"

БГУИР КР 1-40 05 01-10 № 132 ПЗ

|  |  |  |  |
| --- | --- | --- | --- |
|  | Студент | (подпись студента) | Ю.Э. Фишкина   Курсовая работа  представлена на проверку  27.04.2020  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (подпись студента) |

Минск 2020

## Реферат

БГУИР КР 1-40 05 01-10 № 132 ПЗ, гр. 894351

Ю.Э. Фишкина, Разработка компьютерной игры "Пазл", Минск: БГУИР - 2020.

Пояснительная записка 54 с., 20 рис., 0 табл.

Ключевые слова:

Предмет Объектно-ориентированное программирование, А.В.Михалькевич, Паттерны проектирования, MVC, Git, Щаблонный метод

Игра-головоломка, где нужно вращать фигуры, чтобы собрать правильный узор по примеру игры Noodles! https://apps.apple.com/ru/app/noodles/id967624193

## Содержание

[Введение 5](#_Toc38901800)

[1. Описание проекта 6](#_Toc38901801)

[1.1. Геометрия 6](#_Toc38901802)

[1.2. Координаты системы 7](#_Toc38901803)

[1.3. Интерфейс игры 10](#_Toc38901804)

[2. Обоснование выбора технологий 12](#_Toc38901805)

[2.1. ООП парадигмы 13](#_Toc38901806)

[2.2. Java-библиотеки 16](#_Toc38901807)

[3. Инструментарий 19](#_Toc38901808)

[3.1. Система контроля версий 19](#_Toc38901809)

[3.2. Среда разработки ПО 23](#_Toc38901810)

[4. Архитектурный шаблон проектирования MVC 24](#_Toc38901811)

[4.1. Паттерн Модель-Представление-Контроллер (Model-View-Controller) 24](#_Toc38901812)

[4.2. Паттерн Модель представления (Presentation Model) 26](#_Toc38901813)

[5. Фабричный и шаблонный метод проектирования 29](#_Toc38901814)

[5.1 Фабричный метод 29](#_Toc38901815)

[5.2. Шаблонный метод 29](#_Toc38901816)

[Заключение 33](#_Toc38901817)

[Список использованных источников 34](#_Toc38901818)

[Приложение А (обязательное) Схема алгоритма генерации пазла 35](#_Toc38901819)

[Приложение Б (обязательное) Диаграмма состояний 36](#_Toc38901821)

[Приложение В (обязательное) Диаграмма последовательности 37](#_Toc38901823)

[Приложение Д (обязательное) Листинг кода с комментариями 38](#_Toc38901825)

## 

## Введение

Многие древние задачи представляли собой головоломки, которые использовались в обучении. Более того, все эти задачи были тесно связаны с загадками природы. Решение каких-то из них влекло за собой дальнейшие успехи в математике, что, в свою очередь, способствовало разнообразию самих головоломок, так как расширяло их тематическую содержательность. На данный момент существует огромное количество головоломка, одной категорией из которых являются пазлы.

По мнению психологов, собирание пазлов способствует развитию образного и логического мышления, произвольного внимания, восприятия, в частности, различению отдельных элементов по цвету, форме, размеру и так далее; учит правильно воспринимать связь между частью и целым.

На создание данного программного обеспечения вдохновила игра Noodles, которую можно найти в AppStore. Noodles - это простая игра-головоломка, где нужно собрать пазл. Игроки должны соединить все трубы вместе, вращая их положение, пока каждая часть не соединится. Многие из них могут быть связаны несколькими способами, что затрудняет решение пазла. Игра придерживается минималистической идеологии простых линий, очень мало цветов (только два на экран) и простых геометрических фигур.

Актуальность темы:

В процессе работы с головоломками развивается смекалка и умение мыслить неординарно. Решение головоломок - отличный способ проведения досуга, ведь они развивают интеллект, сообразительность и креативность мышления. Правильно найденный ответ на заковыристое задание поднимает уровень самооценки, вселяет уверенность в собственных умственных способностях и просто повышает настроение.

Цель работы:

* закрепить на практике знания, полученные за время изучения дисциплины «Объектно-ориентированное программирование»
* использование паттернов проектирования
* создание программного продукта на языке Java похожего на игру в AppStore Noodles!  
  Детальная постановка задачи:
* Создание пользовательского интерфейса
* Инициализация новой игры;
* Выбор фигуры(квадрат или шестиугольник), с помощью которой будет создан пазл;
* Выбор размера пазла, массив 3х3 является минимальным;
* Поворот фигуры на определенный угол для составления пазла;
* Математический расчет составления узора из геометрических фигур

## Описание проекта

В данном программном обеспечении «Пазл» для составления головоломки используются фигуры. В этой игре реализованы пазлы на основе шестиугольников и квадратов. В дальнейшем ПО можно расширять и создавать пазлы на основе другие геометрических фигур. Мною были изучены различным математические приемы и программные алгоритмы, которые позволяют создать шестиугольную и квадратную сетку необходимую для головоломки.

В данной игре по умолчанию установлено создание пазла на основе шестиугольной сетки, но есть возможность поменять сетку на квадратную. Квадратная сетка в реализация проста, так как матрица объектов может быть легко представлена в виде матрицы квадратов. С шестиугольной сеткой связаны, некоторые математические трудности реализации, так как существует 2 основные вариации, как расположить шестиугольник, и 4 вариации как составить 6-угольную сетку. С шестиугольной сеткой труднее работать в виде матрицы и определить, какие координаты в матрице дать новому шестиугольнику. Далее будут даны объяснения и теория об построении матрицы из шестиугольников, объяснения для построения матрицы и квадратов будет опущено, так как квадратная сетка не требует дополнительных математических алгоритмов для создания.

Шестиугольники - это шестисторонние многоугольники. Правильные шестиугольники имеют все стороны одинаковой длины. Именно такие многоугольники используются в данной игре для построения шестигранных сеток. Типичные ориентации для шестигранных сеток - вертикальные столбцы (с плоским верхом) и горизонтальные ряды (с остроконечным верхом). Рассмотрим детально шестиугольник.

### 1.1. Геометрия

a) Углы

В правильном шестиугольнике внутренние углы составляют 120°. Есть шесть «клиньев»: каждый равносторонний треугольник с углами 60° внутри. (рисунок 1)

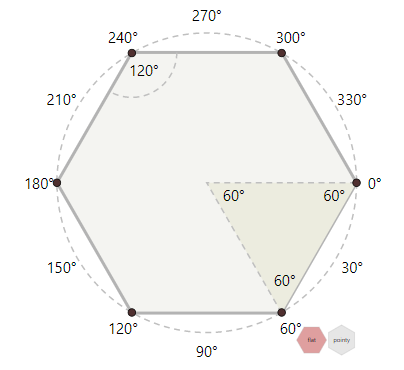


Рисунок 1 – Шестиугольник с разметкой по градусам

b) Размеры

В плоской ориентации шестиугольник имеет:

* Ширину w = 2 \* size
* Высоту h = sqrt (3) \* size,

Где size – это радиус круга описанного вокруг шестиугольника.

Ширина и высота шестиугольника наглядно отображена на рисунке 2.

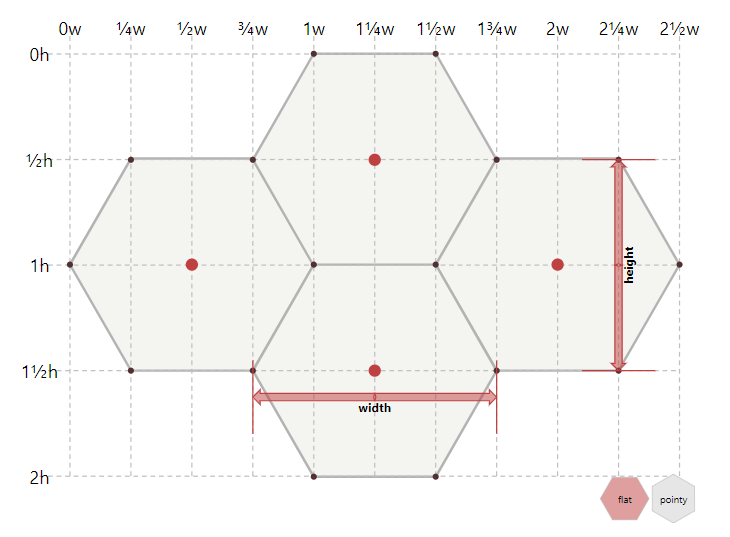


Рисунок 2 – Наглядное отображение ширины и высоты у шестиугольника

Горизонтальное расстояние между центрами соседних шестиугольников составляет w \* 3/4. Вертикальное расстояние между соседними шестиугольными центрами составляет h, где h – высота шестиугольника, а w – ширина шестиугольника.

### 1.2. Координаты системы

Существует множество способов, как создать шестиугольную сетку. В данной игре был использован способ осевых координат. Чтобы разобраться в этом методе, для начала необходимо пояснить об методе кубических координат.

**Метод кубических координат**

Другой способ взглянуть на гексагональные сетки - это увидеть, что есть три основных оси, в отличие от двух, которые мы имеем для квадратных сеток (рисунок 3).

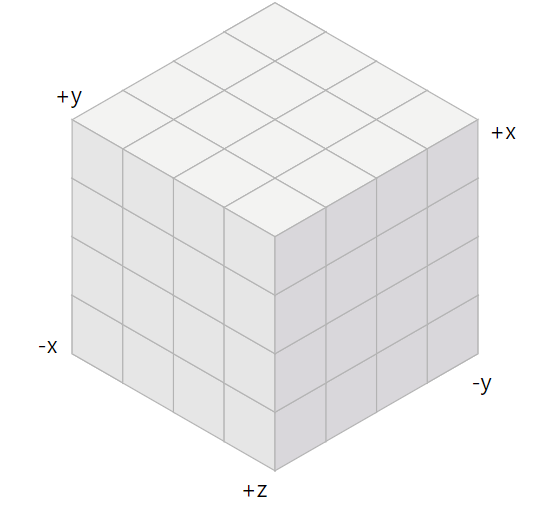


Рисунок 3 – Куб и система координат в трехмерном пространстве

Проведем плоскостное сечение x + y + z = 0 по диагонали куба (рисунок 4). Это действие поможет упростить алгоритмы шестнадцатеричной сетки. В частности, мы можем повторно использовать стандартные операции декартовых координат: сложение координат, вычитание координат, умножение или деление на скаляр и расстояния.

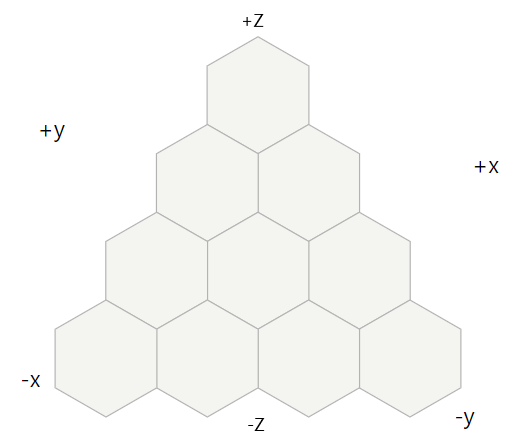


Рисунок 4 – Преобразование трехмерной кубической сетки в двумерную шестигональную сетку

На рисунке 5 изображен метод кубических координат на шестигональной сетке. Заметим, что каждое направление на сетке куба соответствует линии на шестигранной сетке. Каждое направление на шестигранной сетке является комбинацией двух направлений на сетке куба. Например, северо-запад на шестнадцатеричной сетке лежит между + y и -z, поэтому каждый шаг на северо-запад включает добавление 1 к y и вычитание 1 из z.

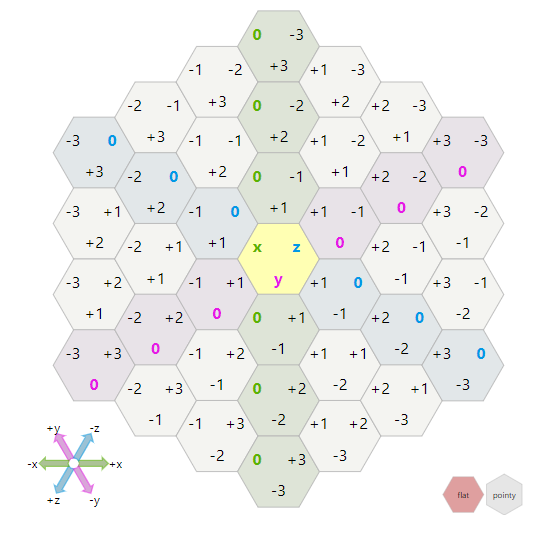


Рисунок 5 – Метод кубических координат на шестигональной сетке

Метод построения матрицы шестиугольников, используя систему координат куба является разумным выбором, но реализация такой модель может быть затруднительной, так как необходимо хранить 3 числа для системы координат. Упрощенной версией метода кубических координат является метод осевых координат, о котором и пойдет речь.

**Осевая система координат**

Осевая система координат строится путем взятия двух из трех координат из системы координат куба. Поскольку есть ограничение x + y + z = 0, существует некоторая избыточность, таким образом не нужно хранить все три координаты. Эта диаграмма такая же, как и предыдущая, за исключением того, что значение y опущено (рисунок 6):



Рисунок 6 – Метод осевых координат на шестигональной сетке

Также из осевых координат легко перейти в кубические координаты, с помощью математического действия: y = -z – x.

### Интерфейс игры

Интерфейс игры включает несколько панелей:

* Панель отрисовки пазла
* Панель возобновления игры, в котором находится кнопка с надписью "New Puzzle"
* Панель выбора фигуры, где находится 2 кноки JRadioButton: Hexagon(Шестиугольник) и Square(квадрат)
* Панель изменения размера пазла. Минимальным размером пазла является матрица 3х3. В панель встроено 2 JSpinner, с помощью которых можно выбрать размер пазла. Чем больше пазл, тем сложнее его решить.

На рисунках 7-8 представлена реализация интерфейса пазла двух видов.

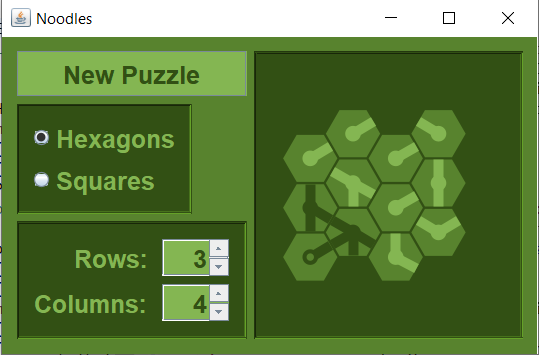


Рисунок 7 – Интерфейс игры с шестиугольными фигурами

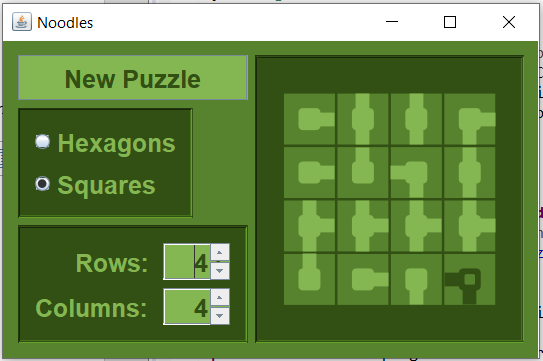


Рисунок 8 - Интерфейс игры с шестиугольными фигурами

Рассмотрим подробнее панель отрисовки пазла. При запуске игры выводиться пазл 3х4 с шестиугольными фигурами. Заметьте, что связь на одной фигуре будет темного цвета и в центре имеет полость, чем отличается от всех остальных фигур. Эта фигура показывает, что с этого элемента удобнее все начать собирать пазл, так как он является корневым и от него отходят «ветви» пазла. При соединении линий фигур, связи отображаются более темным цветом.

Для того, чтобы решить пазл нужно вращать фигуры. Повернуть фигуру можно, нажав на центр фигуры правой или левой кнопкой мыши, таким образом фигура повернется направо или налево соответственно. Также можно воспользоваться подсказкой: для этого, нужно нажать на центр фигуры колесиком мыши. При успешном решении пазла, пользователь увидит всплывающее окно с сообщением об решении пазла (рисунок 9).

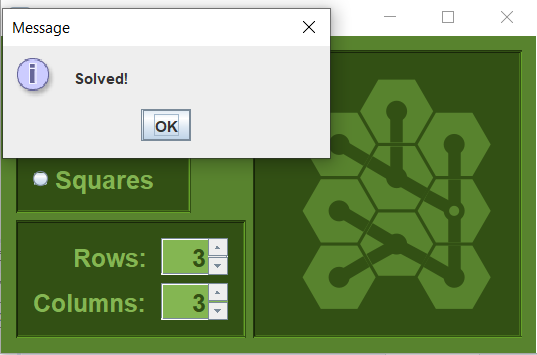


Рисунок 9 – Пазл решен

## Обоснование выбора технологий

При объектно-ориентированном подходе программа представляет собой описание объектов, их свойств (или атрибутов), совокупностей (или классов), отношений между ними, способов их взаимодействия и операций над объектами (или методов).

Несомненным преимуществом данного подхода является концептуальная близость к предметной области произвольной структуры и назначения. Механизм наследования атрибутов и методов позволяет строить производные понятия на основе базовых и таким образом создавать модель сколь угодно сложной предметной области с заданными свойствами.

Еще одним теоретически интересным и практически важным свойством объектно-ориентированного подхода является поддержка механизма обработки событий, которые изменяют атрибуты объектов и моделируют их взаимодействие в предметной области.

В отличие от предыдущих подходов к программированию, объектно-ориентированный подход требует глубокого понимания основных принципов, или, иначе, концепций, на которых он базируется. К числу основополагающих понятий ООП обычно относят абстракцию данных, наследование, инкапсуляцию и полиморфизм.

Объектно-ориентированное программирование в настоящее время является абсолютным лидером в области прикладного программирования.

В качестве основного инструмента разработки применяется Eclipse Java 2019. Язык программирования Java.

Java - объектно-ориентированный язык программирования, разработанный компанией Sun Microsystems. Приложения Java обычно компилируются в специальный байт-код, поэтому они могут работать на любой виртуальной Java-машине (JVM) вне зависимости от компьютерной архитектуры.

Достоинство подобного способа выполнения программ - в полной независимости байт-кода от операционной системы и оборудования, что позволяет выполнять Java-приложения на любом устройстве, для которого существует соответствующая виртуальная машина. Другой важной особенностью технологии Java является гибкая система безопасности благодаря тому, что исполнение программы полностью контролируется виртуальной машиной. Любые операции, которые превышают установленные полномочия программы (например, попытка несанкционированного доступа к данным или соединения с другим компьютером) вызывают немедленное прерывание.

Часто к недостаткам концепции виртуальной машины относят то, что исполнение байт-кода виртуальной машиной может снижать производительность программ и алгоритмов, реализованных на языке Java. В последнее время был внесен ряд усовершенствований, которые несколько увеличили скорость выполнения программ на Java.

### 2.1. ООП парадигмы

С помощью языка Java были реализованы основные ооп парадигмы. В этой программе были использованы такие принципы ООП, как абстрактность, полиморфизм и наследование.

**Абстракция.**

Абстра́кция  — в объектно-ориентированном программировании это придание объекту характеристик, которые отличают его от всех других объектов, четко определяя его концептуальные границы. Основная идея состоит в том, чтобы отделить способ использования составных объектов данных от деталей их реализации в виде более простых объектов, подобно тому, как функциональная абстракция разделяет способ использования функции и деталей её реализации в терминах более примитивных функций, таким образом, данные обрабатываются функцией высокого уровня с помощью вызова функций низкого уровня.

Такой подход является основой объектно-ориентированного программирования. Это позволяет работать с объектами, не вдаваясь в особенности их реализации. В каждом конкретном случае применяется тот или иной подход: инкапсуляция, полиморфизм или наследование. Например, при необходимости обратиться к скрытым данным объекта, следует воспользоваться инкапсуляцией, создав, так называемую, функцию доступа или свойство.

Абстракция данных — популярная и в общем неверно определяемая техника программирования. Фундаментальная идея состоит в разделении несущественных деталей реализации подпрограммы и характеристик существенных для корректного ее использования. Такое разделение может быть выражено через специальный «интерфейс», сосредотачивающий описание всех возможных применений программы.

**Полиморфизм**

Полиморфи́зм — возможность объектов с одинаковой спецификацией иметь различную реализацию.

Язык программирования поддерживает полиморфизм, если классы с одинаковой спецификацией могут иметь различную реализацию — например, реализация класса может быть изменена в процессе наследования.

Кратко смысл полиморфизма можно выразить фразой: «Один интерфейс, множество реализаций».

Полиморфизм — один из четырёх важнейших механизмов объектно-ориентированного программирования (наряду с абстракцией, инкапсуляцией и наследованием).

Полиморфизм позволяет писать более абстрактные программы и повысить коэффициент повторного использования кода. Общие свойства объектов объединяются в систему, которую могут называть по-разному — интерфейс, класс. Общность имеет внешнее и внутреннее выражение:

* внешняя общность проявляется как одинаковый набор методов с одинаковыми именами и сигнатурами (именем методов и типами аргументов и их количеством);
* внутренняя общность — одинаковая функциональность методов. Её можно описать интуитивно или выразить в виде строгих законов, правил, которым должны подчиняться методы. Возможность приписывать разную функциональность одному методу (функции, операции) называется перегрузкой метода (перегрузкой функций, перегрузкой операций).

Формы полиморфизма

Используя Параметрический полиморфизм можно создавать универсальные базовые типы. В случае параметрического полиморфизма, функция реализуется для всех типов одинаково и таким образом функция реализована для произвольного типа. В параметрическом полиморфизме рассматриваются параметрические методы и типы.

* Параметрические методы.
* Если полиморфизм включения влияет на наше восприятие объекта, то параметрический полиморфизм влияет на используемые методы, так как можно создавать методы родственных классов, откладывая объявление типов до времени выполнения. Для избежания написания отдельного метода каждого типа применяется параметрический полиморфизм, при этом тип параметров будет являться таким же параметром, как и операнды.
* Параметрические типы.
* Вместо того, чтобы писать класс для каждого конкретного типа следует создать типы, которые будут реализованы во время выполнения программы то есть мы создаем параметрический тип.
* Полиморфизм переопределения

Абстрактные методы часто относятся к отложенным методам. Класс, в котором определен этот метод может вызвать метод и полиморфизм обеспечивает вызов подходящей версии отложенного метода в дочерних классах. Специальный полиморфизм допускает специальную реализацию для данных каждого типа.

**Наследование**

Наследование — один из четырёх важнейших механизмов объектно-ориентированного программирования (наряду с инкапсуляцией, полиморфизмом и абстракцией), позволяющий описать новый класс на основе уже существующего (родительского), при этом свойства и функциональность родительского класса заимствуются новым классом.

Другими словами, класс-наследник реализует спецификацию уже существующего класса (базовый класс). Это позволяет обращаться с объектами класса-наследника точно так же, как с объектами базового класса.

Простое наследование

Класс, от которого произошло наследование, называется базовым или родительским (англ. base class). Классы, которые произошли от базового, называются потомками, наследниками или производными классами (англ. derived class).

В некоторых языках используются абстрактные классы. Абстрактный класс — это класс, содержащий хотя бы один абстрактный метод, он описан в программе, имеет поля, методы и не может использоваться для непосредственного создания объекта. То есть от абстрактного класса можно только наследовать. Объекты создаются только на основе производных классов, наследованных от абстрактного. Например, абстрактным классом может быть базовый класс «сотрудник вуза», от которого наследуются классы «аспирант», «профессор» и т. д. Так как производные классы имеют общие поля и функции (например, поле «год рождения»), то эти члены класса могут быть описаны в базовом классе. В программе создаются объекты на основе классов «аспирант», «профессор», но нет смысла создавать объект на основе класса «сотрудник вуза».

Множественное наследование

При множественном наследовании у класса может быть более одного предка. В этом случае класс наследует методы всех предков. Достоинства такого подхода в большей гибкости. Множественное наследование реализовано в C++. Из других языков, предоставляющих эту возможность, можно отметить Python и Эйфель. Множественное наследование поддерживается в языке UML.

Множественное наследование — потенциальный источник ошибок, которые могут возникнуть из-за наличия одинаковых имен методов в предках. В языках, которые позиционируются как наследники C++ (Java, C# и др.), от множественного наследования было решено отказаться в пользу интерфейсов. Практически всегда можно обойтись без использования данного механизма. Однако, если такая необходимость все-таки возникла, то, для разрешения конфликтов использования наследованных методов с одинаковыми именами, возможно, например, применить операцию расширения видимости — «::» — для вызова конкретного метода конкретного родителя.

Попытка решения проблемы наличия одинаковых имен методов в предках была предпринята в языке Эйфель, в котором при описании нового класса необходимо явно указывать импортируемые члены каждого из наследуемых классов и их именование в дочернем классе.

Большинство современных объектно-ориентированных языков программирования (C#, Java, Delphi и др.) поддерживают возможность одновременно наследоваться от класса-предка и реализовать методы нескольких интерфейсов одним и тем же классом. Этот механизм позволяет во многом заменить множественное наследование — методы интерфейсов необходимо переопределять явно, что исключает ошибки при наследовании функциональности одинаковых методов различных классов-предков.

В ООП главным элементом является класс, включающий множество объектов с одинаковыми свойствами, операциями и отношениями. Класс имеет внутреннее (реализацию) и внешнее представление - интерфейс. В данной программе создано 3 пакета: noodles, noodles.hex, noodles.square.

### 2.2. Java-библиотеки

Для создания интерфейса программы были использованы библиотека Swing и пакет java.awt, предоставляющий технологию AWT для создания графического (оконного) интерфейса пользователя

**Библиотека AWT**

Ни одна современная программа, предназначенная для пользователя, не обходится без удобного, понятного, в идеале – красивого пользовательского интерфейса. С самой первой версии в Java существует специальная технология для создания GUI. Она называется AWT, Abstract Window Toolkit. Именно о ней пойдет речь в этой лекции. Пакет java.awt претерпел, пожалуй, больше всего изменений с развитием версий Java. Мы рассмотрим дерево компонентов, доступных программисту, специальную модель сообщений, позволяющую гибко обрабатывать пользовательские действия, и другие особенности AWT – работа с цветами, шрифтами, отрисовка графических примитивов, менеджеры компоновки и т.д.

Абстрактный класс Component является базовым для всех компонентов AWT и описывает их основные свойства. Визуальный компонент в AWT имеет прямоугольную форму, может быть отображен на экране и может взаимодействовать с пользователем.

Рассмотрим основные свойства этого класса.

1. Положение

Положение компонента описывается двумя целыми числами (тип int ) x и y. В Java (как и во многих языках программирования) ось x проходит традиционно – горизонтально, направлена вправо, а ось у – вертикально, но направлена вниз, а не вверх, как принято в математике.

Для описания положения компонента предназначен специальный класс – Point (точка). В этом классе определено два public int поля x и y, а также множество конструкторов и вспомогательных методов для работы с ними. Класс Point применяется во многих типах AWT, где надо задать точку на плоскости.

Для компонента эта точка задает положение левого верхнего угла.

Установить положение компонента можно с помощью метода setLocation(), который может принимать в качестве аргументов пару целых чисел, либо Point. Узнать текущее положение можно с помощью метода getLocation(), возвращающего Point, либо с помощью методов getX() и getY(), которые появились с версии Java 1.2.

1. Размер

Как было сказано, компонент AWT имеет прямоугольную форму, а потому его размер описывается также двумя целочисленными параметрами – width (ширина) и height (высота). Для описания размера существует специальный класс Dimension (размер), в котором определено два public int поля width и height, а также вспомогательные методы.

Установить размер компонента можно с помощью метода setSize, который может принимать в качестве аргументов пару целых чисел, либо Dimension. Узнать текущие размеры можно с помощью метода getSize(), возвращающего Dimension, либо с помощью методов getWidth() и getHeight(), которые появились с версии Java 1.2.

Совместно положение и размер компонента задают его границы. Область, занимаемую компонентом, можно описать либо четырьмя числами ( x, y, width, height ), либо экземплярами классов Point и Dimension, либо специальным классом Rectangle (прямоугольник). Как легко догадаться, в этом классе определено четыре public int поля, с которыми можно работать и в виде пары объектов Point и Dimension.

Задать границу объекта можно с помощью метода setBounds, который может принимать четыре числа, либо Rectangle. Узнать текущее значение можно с помощью метода getBounds(), возвращающего Rectangle.

1. Видимость

Существующий компонент может быть как виден пользователю, так и быть скрытым. Это свойство описывается булевским параметром visible. Методы для управления – setVisible, принимающий булевский параметр, и isVisible, возвращающий текущее значение.

Разумеется, невидимый компонент не может взаимодействовать с пользователем.

1. Доступность

Даже если компонент отображается на экране и виден пользователю, он может не взаимодействовать с ним. В результате события от клавиатуры, или мыши не будут получаться и обрабатываться компонентом. Такой компонент называется disabled. Если же компонент активен, его называют enabled. Как правило, компонент некоторым образом меняет свой внешний вид, когда становится недоступным (например, становится серым, менее заметным), но, вообще говоря, это необязательно (хотя очень удобно для пользователя).

Для изменения этого свойства применяется метод setEnabled, принимающий булевский параметр ( true соответствует enabled, false – disabled ), а для получения текущего значения – isEnabled.

1. Цвета

Разумеется, для построения современного графического интерфейса пользователя необходима работа с цветами.

Компонент обладает двумя свойствами, описывающими цвета, – foreground и background цвета. Первое свойство задает, каким цветом выводить надписи, рисовать линии и т.д. Второе – задает цвет фона, которым закрашивается вся область, занимаемая компонентом, перед тем, как прорисовывается внешний вид.

Для задания цвета в AWT используется специальный класс Color. Этот класс обладает довольно обширной функциональностью, поэтому рассмотрим основные характеристики.

Цвет задается 3 целочисленными характеристиками, соответствующими модели RGB, – красный, зеленый, синий. Каждая из них может иметь значение от 0 до 255 (тем не менее, их тип определен как int ). В результате (0, 0, 0) соответствует черному, а (255, 255, 255) – белому.

Класс Color является неизменяемым, то есть, создав экземпляр, соответствующий какому-либо цвету, изменить параметры RGB уже невозможно. Это позволяет объявить в классе Color ряд констант, описывающих базовые цвета: белый, черный, красный, желтый и так далее. Например, вместо того, чтобы задавать синий цвет числовыми параметрами (0, 0, 255), можно воспользоваться константами Color.blue или Color.BLUE (второй вариант появился в более поздних версиях).

Для работы со свойством компонента foreground применяют методы setForeground и getForeground, а для background – setBackground и getBackground.

**Библиотека Swing**

Swing — библиотека для создания графического интерфейса для программ на языке Java. Swing был разработан компанией Sun Microsystems. Он содержит ряд графических компонентов, таких как кнопки, поля ввода, таблицы и т. д.

Swing является одним из простейших вариантов создания интерфейса на Java. Именно поэтому для создания простейшего интерфейса была выбрана библиотека Swing. Существует множество специализированных классов, в этом проекте были использованы наиболее популярные из них:

* JLabel - класс для создания объекта с надписью;
* JTextField - класс для создания текстового поля;
* JButton - класс для создания кнопки;
* JCheckBox - класс для создания кнопки-галочки;
* JRadioButton - класс для создания кнопок выбора.

## Инструментарий

### 3.1. Система контроля версий

Система контроля версий — это система, записывающая изменения в файл или набор файлов в течение времени и позволяющая вернуться позже к определённой версии.

Cистема контроля версий позволяет вернуть файлы к состоянию, в котором они были до изменений, вернуть проект к исходному состоянию, увидеть изменения, кто последний менял что-то и вызвал проблему, кто поставил задачу и когда и многое другое. Использование СКВ также значит в целом, что, если вы сломали что-то или потеряли файлы, вы спокойно можете всё исправить. В дополнение ко всему вы получите всё это без каких-либо дополнительных усилий.

**Локальные системы контроля версий**

Многие люди в качестве метода контроля версий применяют копирование файлов в отдельную директорию (возможно даже, директорию с отметкой по времени, если они достаточно сообразительны). Данный подход очень распространён из-за его простоты, однако он невероятно сильно подвержен появлению ошибок. Можно легко забыть, в какой директории вы находитесь, и случайно изменить не тот файл или скопировать не те файлы, которые вы хотели.

Для того, чтобы решить эту проблему, программисты давным-давно разработали локальные СКВ с простой базой данных, которая хранит записи о всех изменениях в файлах, осуществляя тем самым контроль ревизий(рисунок 10).

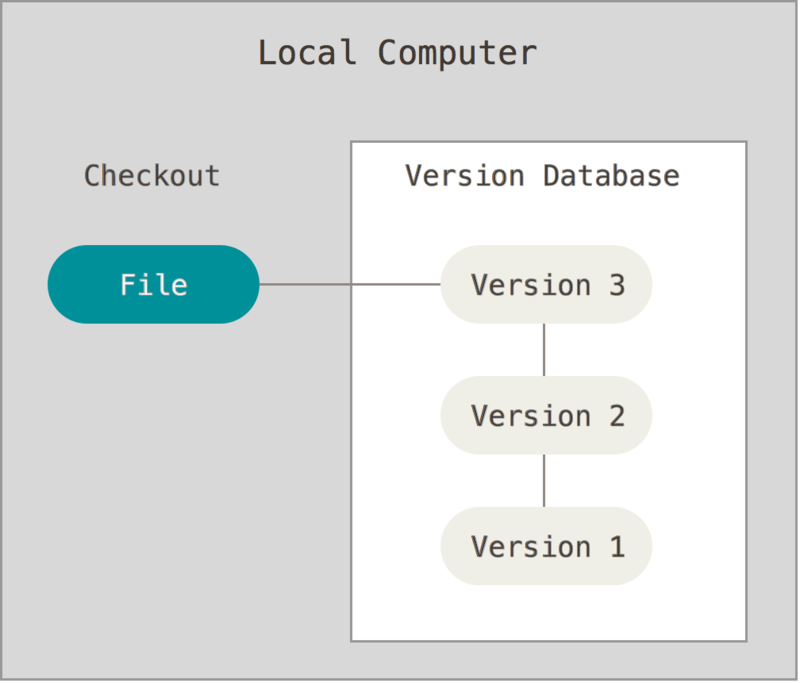


Рисунок 10 - Локальный контроль версий.

Одной из популярных СКВ была система RCS, которая и сегодня распространяется со многими компьютерами. RCS хранит на диске наборы патчей (различий между файлами) в специальном формате, применяя которые она может воссоздавать состояние каждого файла в заданный момент времени.

**Централизованные системы контроля версий**

Следующая серьёзная проблема, с которой сталкиваются люди, — это необходимость взаимодействовать с другими разработчиками. Для того, чтобы разобраться с ней, были разработаны централизованные системы контроля версий (ЦСКВ). Такие системы, как CVS, Subversion и Perforce, используют единственный сервер, содержащий все версии файлов, и некоторое количество клиентов, которые получают файлы из этого централизованного хранилища. Применение ЦСКВ являлось стандартом на протяжении многих лет (рисунок 11).

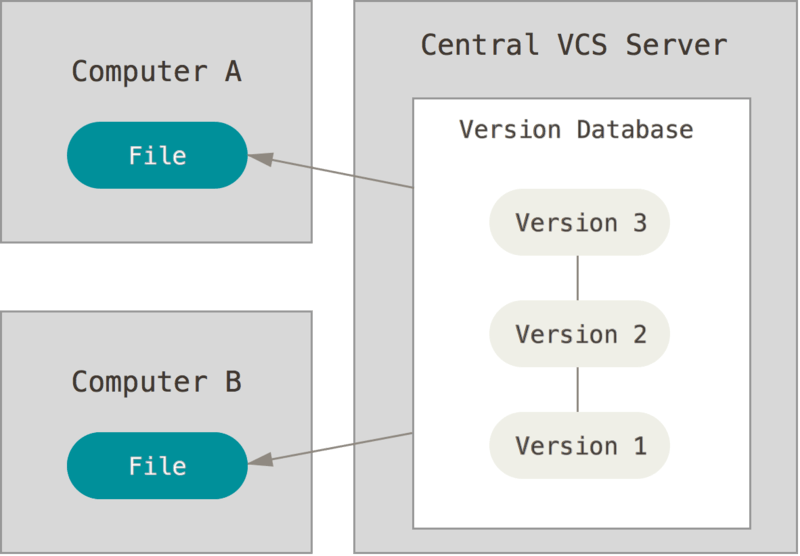


Рисунок 11 - Централизованный контроль версий

Такой подход имеет множество преимуществ, особенно перед локальными СКВ. Например, все разработчики проекта в определённой степени знают, чем занимается каждый из них. Администраторы имеют полный контроль над тем, кто и что может делать, и гораздо проще администрировать ЦСКВ, чем оперировать локальными базами данных на каждом клиенте.

Несмотря на это, данный подход тоже имеет серьёзные минусы. Самый очевидный минус — это единая точка отказа, представленная централизованным сервером. Если этот сервер выйдет из строя на час, то в течение этого времени никто не сможет использовать контроль версий для сохранения изменений, над которыми работает, а также никто не сможет обмениваться этими изменениями с другими разработчиками. Если жёсткий диск, на котором хранится центральная БД, повреждён, а своевременные бэкапы отсутствуют, вы потеряете всё — всю историю проекта, не считая единичных снимков репозитория, которые сохранились на локальных машинах разработчиков. Локальные СКВ страдают от той же самой проблемы: когда вся история проекта хранится в одном месте, вы рискуете потерять всё.

**Распределённые системы контроля версий**

Здесь в игру вступают распределённые системы контроля версий (РСКВ). В РСКВ (таких как Git, Mercurial, Bazaar или Darcs) клиенты не просто скачивают снимок всех файлов (состояние файлов на определённый момент времени) — они полностью копируют репозиторий. В этом случае, если один из серверов, через который разработчики обменивались данными, умрёт, любой клиентский репозиторий может быть скопирован на другой сервер для продолжения работы. Каждая копия репозитория является полным бэкапом всех данных (рисунок 12).

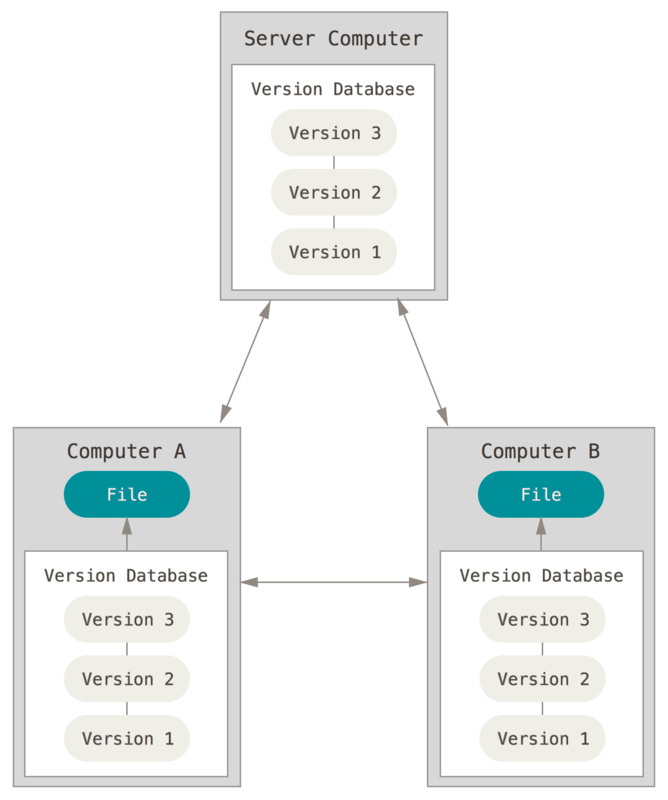


Рисунок 12 - Распределённый контроль версий

Более того, многие РСКВ могут одновременно взаимодействовать с несколькими удалёнными репозиториями, благодаря этому вы можете работать с различными группами людей, применяя различные подходы единовременно в рамках одного проекта. Это позволяет применять сразу несколько подходов в разработке, например, иерархические модели, что совершенно невозможно в централизованных системах.

**Клонирование существующего репозитория**

Для получения копии существующего Git-репозитория, например, необходимо использовать команду git clone. При выполнении git clone с сервера забирается (pulled) каждая версия каждого файла из истории проекта. Фактически, если серверный диск выйдет из строя, возможно использовать любой из клонов на любом из клиентов, для того, чтобы вернуть сервер в то состояние, в котором он находился в момент клонирования.

**Запись изменений в репозиторий**

Каждый файл в рабочем каталоге Git-репозитория может находиться в одном из двух состояний: под версионным контролем (отслеживаемые) и нет (неотслеживаемые). Отслеживаемые файлы — это те файлы, которые были в последнем снимке состояния проекта; они могут быть неизменёнными, изменёнными или подготовленными к коммиту. Если кратко, то отслеживаемые файлы — это те файлы, о которых знает Git.

Неотслеживаемые файлы — это всё остальное, любые файлы в рабочем каталоге, которые не входили в последний снимок состояния и не подготовлены к коммиту. Когда репозиторий впервые клонируется, все файлы будут отслеживаемыми и неизменёнными, потому что Git только что их извлек и пользователь ничего пока не редактировал.

Как только файлы отредактированы, Git будет рассматривать их как изменённые, так как пользователь изменил их с момента последнего коммита. Вы индексируете эти изменения, затем фиксируете все проиндексированные изменения, а затем цикл повторяется (рисунок 13).

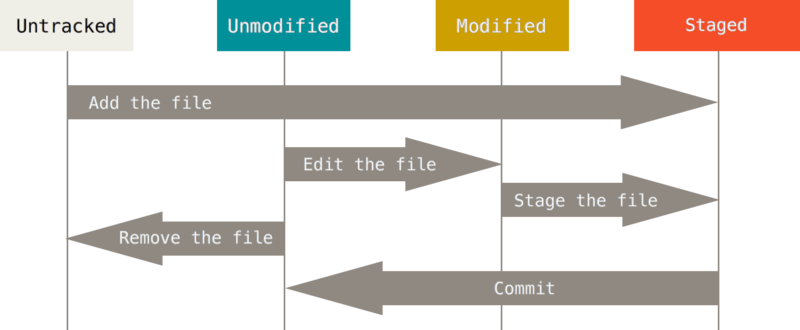


Рисунок 13 - Жизненный цикл состояний файлов

**Определение состояния файлов**

Основной инструмент, используемый для определения, какие файлы в каком состоянии находятся — это команда git status. Предположим, пользователь добавил в свой проект новый файл, простой файл README. Понять, что новый файл README неотслеживаемый можно по тому, что он находится в секции “Untracked files” в выводе команды status.

Статус Untracked означает, что Git видит файл, которого не было в предыдущем снимке состояния (коммите); Git не станет добавлять его в коммиты, пока пользователь его явно об этом не попросит. Это предохранит от случайного добавления в репозиторий сгенерированных бинарных файлов или каких-либо других, которые и не нужно добавлять. Мы хотели добавить README, так давайте сделаем это.

**Отслеживание новых файлов**

Для того чтобы начать отслеживать (добавить под версионный контроль) новый файл, используется команда git add. Чтобы начать отслеживание файла README, нужно можете выполнить следующее:

$ git add README

Команда git add принимает параметром путь к файлу или каталогу, если это каталог, команда рекурсивно добавляет все файлы из указанного каталога в индекс.

**Коммит изменений**

Любые файлы, созданные или изменённые, и для которых пользователь не выполнил git add после редактирования — не войдут в этот коммит. Они останутся изменёнными файлами на диске. Простейший способ зафиксировать изменения — это набрать git commit:

$ git commit

### 3.2. Среда разработки ПО

Наиболее популярной средой разработки программ на языке Java является Eclipse.

Eclipse служит в первую очередь платформой для разработки расширений, чем он и завоевал популярность: любой разработчик может расширить Eclipse своими модулями. Уже существуют Java Development Tools (JDT), Android Development Tools, C/C++ Development Tools (CDT), разрабатываемые инженерами QNX совместно с IBM, и средства для языков Ada (GNATbench, Hibachi), COBOL, FORTRAN, PHP и пр. от различных разработчиков. Множество расширений дополняет среду Eclipse менеджерами для работы с базами данных, серверами приложений и др.

Eclipse JDT (Java Development Tools) — наиболее известный модуль, нацеленный на групповую разработку: среда интегрирована с системами управления версиями — CVS, GIT в основной поставке, для других систем (например, Subversion, MS SourceSafe) существуют плагины. Также предлагает поддержку связи между IDE и системой управления задачами (ошибками). В основной поставке включена поддержка трекера ошибок Bugzilla, также имеется множество расширений для поддержки других трекеров (Trac, Jira и др.). В силу бесплатности и высокого качества, Eclipse во многих организациях является корпоративным стандартом для разработки приложений.

## Архитектурный шаблон проектирования MVC

Ключевым паттерном в подходе проектирования MVC является непосредственно сам шаблон (паттерн) Модель-Представление-Контроллер (Model-View-Controller), который имеет множество вариаций, такие как Модель-Представление-Презентер (Model-View-Presenter), Презентация-Абстракция-Контрол (Presentation-Abstraction-Control), Модель представления (Presentation Model), Модель-Представление-Модель представления (Model-View-ViewModel) и многие другие (некоторые из которых в свою очередь также имеют ряд модификаций), которые, по сути, представляют собой интерпретации «исходного» шаблона (паттерна) MVC, но в тоже время имеют и существенные отличия от него.

### 4.1. Паттерн Модель-Представление-Контроллер (Model-View-Controller)

По историческим причинам аббревиатурой MVC принято называть не один единственный паттерн, а целое семейство паттернов, призванное отделять представление от модели. Произошло это в силу разных обстоятельств. Отчасти из-за того что MVC не просто паттерн, а довольно объемное архитектурное решение, в котором каждый новый разработчик видел что-то свое и ставя во главу угла особенности своего проекта, реализовывал его по своему. Отчасти же из-за возраста данного паттерна, во времена его изобретения и сами приложения, и графические интерфейсы были существенно беднее, чем в наше время, с тех пор они сильно эволюционировали, и вместе с ними изменялся и сам паттерн.

Несмотря на то, что, как уже говорилось, под аббревиатурой MVC обычно понимают целое семейство паттернов, можно выделить первоначальный вариант данного архитектурного решения. На данный момент, применение MVC в оригинальном виде вряд ли будет оправдано, по разным причинам, однако рассмотрим эволюцию паттерна с самого начала.

Впервые паттерн MVC появился в уже упомянутом языке SmallTalk в конце семидесятых. В качестве задачи выступало нахождение архитектурного решения, которое позволяло бы манипулировать графическими представлениями данных некоего приложения, таким образом, чтобы изменение Представления этих данных не влияло на бизнес-логику и данные приложения, а также, чтобы была возможность иметь несколько Представлений для одной Модели. Таким решением и стал паттерн MVC.

В классическом варианте, MVC состоит из трех частей, которые и дали ему название.

**Модель (Model)**

Под Моделью, обычно понимается часть, содержащая в себе функциональную логику приложения, иными словами то, что обычно называется «Business Layer», «Бизнес-слой» или «Слой бизнес логики». Как именно организован этот слой, по большому счету не важно, однако есть ряд ключевых моментов, которые будут рассмотрены ниже.

Основная цель паттерна - сделать так, чтобы Модель была полностью независима от остальных частей и практически ничего не знала об их существовании. Это позволило бы менять и Контроллер и Представление модели, не трогая саму Модель и даже позволить функционирование нескольких экземпляров Представлений и Контроллеров с одной Моделью одновременно. Поэтому Модель ни при каких условиях не может содержать ссылок на объекты Представления или Контроллера.

Обычно различают несколько типов паттернов в зависимости от роли модели.

Passive Model (пассивная модель) - Модель не имеет вообще никаких способов воздействовать на Представление или Контроллер и только используется ими в качестве источника данных для отображения. Все изменения модели отслеживаются Контроллером, и он же отвечает за перерисовку Представления, если это необходимо.

Active Model (активная модель) - Модель имеет возможность оповестить Представление о том, что в ней произошли некие изменения, и Представление может эти изменения отобразить. Как правило, механизм оповещения реализуется на основе паттерна Observer (Наблюдатель), Модель просто генерирует сообщение, а Представления, которые заинтересованы в оповещении, подписываются на эти сообщения, что позволяет сохранить независимость Модели, как от Контроллера, так и от Представления, не нарушая тем самым основного свойства паттерна. Классической реализацией паттерна MVC принято считать версию именно с активной Моделью.

**Представление (View)**

В обязанности Представления входит отображение данных, полученных от Модели. Обычно Представление имеет свободный доступ к Модели и может брать из нее данные, однако это доступ только на чтение, ничего менять в Модели или даже просто вызывать методы, приводящие к изменению ее внутреннего состояния, Представлению позволять нельзя. В случае активной Модели, Представление может подписаться на события изменения Модели и обновляться, забрав измененные данные, при получении соответствующего оповещения. Для взаимодействия с Контроллером, Представление, как правило, реализует некий интерфейс, известный Контроллеру, что позволяет менять Представления независимо и иметь несколько Представлений на Контроллер. Вообще, подмена или изменение Представления самая часто встречающаяся задача, по сути это и есть та причина, по которой создают различные паттерны разделения Модели и Представления.

**Контроллер (Controller)**

В задачи Контроллера входит реакция на внешние раздражители и изменение Модели и/или Представления в соответствии с заложенной в него логикой. Один Контроллер может работать с несколькими Представлениями, в зависимости от ситуации, взаимодействуя с ними через некий заранее известный интерфейс, который эти Представления реализуют. Важный моментом здесь является то обстоятельство, что в классической версии MVC Контроллер не занимается передачей данных из Модели в Представление и не является медиатором (Mediator) между Моделью и Представлениями.

Стоит отметить, что MVC не определяет, каким именно способом Модель взаимодействует с данными и как реализован уровень доступа к данным – это лежит вне зоны ответственности данного паттерна.

Итак, типичная схема взаимодействия компонентов паттерна выглядит примерно так, как показано на рисунке 14. Контроллер перехватывает событие извне и в соответствии с заложенной в него логикой, реагирует на это событие, изменяя Модель, посредством вызова соответствующего метода Модели. После изменения Модель генерирует событие о том, что она изменилась, и все подписанные на это события Представления, получив его, обращаются к Модели за обновленными данными, после чего их и отображают.

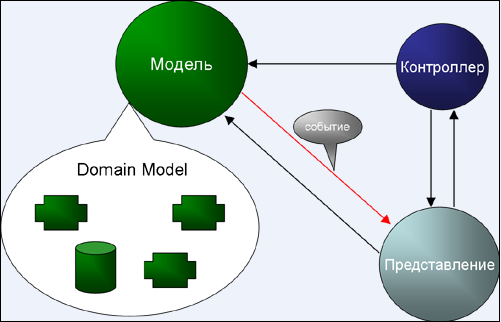


Рисунок 14 - Схема взаимодействия компонентов паттерна MVC.

При этом в описании оригинального паттерна также упоминалось, что выделение отдельного Контроллера не так важно как отделение Представления от Модели, и Контроллер вполне может быть интегрирован в Представление, тем более что в классическом варианте MVC логики в Контроллере не очень много.

Рассмотрим более подходящий архитектурный паттерн для данной разработки – это паттерн модель представления.

### 4.2. Паттерн Модель представления (Presentation Model)

Модель представления (Presentation Model) – это логическое представление UI без привязки к элементам графического интерфейса. Схема паттерна представлена на рисунке 15:

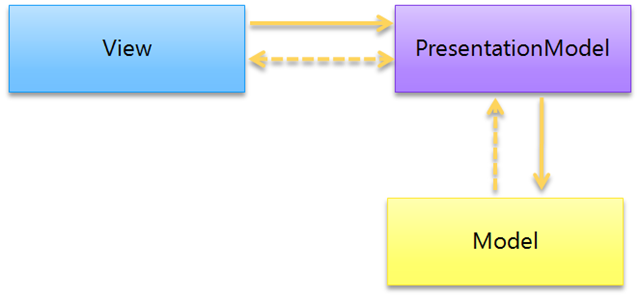


Рисунок 15 - Схема взаимодействия компонентов паттерна PM.

Модель представления имеет здесь следующие функции:

* Содержит логику UI: например, при нажатии на кнопку, Модель представления уведомляется об этом и производит соответствующие действия.
* Отображает данные Модели: производит необходимую конвертацию и форматирование данных Модели для правильного отображения пользователю.
* Хранит состояние UI: хранение текущего выбранного элемента списка, ошибок валидации и прочее.

Представление имеет доступ к данным, хранящимся в Модели представления для дальнейших манипуляций (отображения). Преимуществом такого подхода является то, что можно создавать тесты для UI без привязки к элементам графического интерфейса.

Модель преставления реализована в данной игре на рисунке 16 изображена UML-схема, которая показывает связь между классов Noodles, который является Представлением(View), классом NoodlePanel, который является своего рода Контроллером(ModelPresentor), и класс NoodleStrategy, в котором заложена основная логика игры и является Моделью(Model). В NoodlePanel происходит инициализация GUI-элементов и запуск класса NoodlePanel, который обрабатывает нажатие клавиш и взаимодействует с пользователем непосредственно через них.

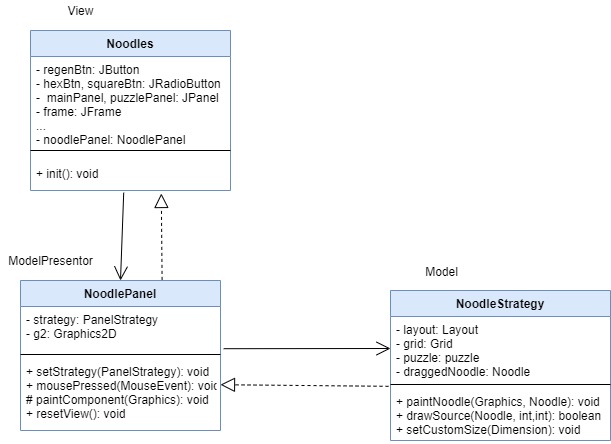


Рисунок 16 – UML-схема взаимодействия компонентов паттерна PM в игре

В приложении A представлена схема алгоритма генерации пазла. Подробное описание также дано на схеме в комментариях к каждому блоку.

Приложении Б схема состояния игровой панели. Обобщенно панель может находится в трех состояниях:

* Запуск панели – инициализация панели в классе Noodles. Далее создается физическая модель фигуры (просчитывается центр фигуры, углы и стороны, класс Layout) и логическая модель фигуры (задаются значение сторон, которые используются при построении пазла, соседние фигуры и поставлена ли фигура правильно, класс Noodle). Затем из физической и логической фигуры строится матрица из фигур (класс Grid) и генерируется случайный пазл (класс Puzzle).
* Отрисовка – отображение матрицы фигур с пазлом на панели (класс PanelStrategy)
* Вращение фигуры – происходит по нажатию фигуры для сбора правильного рисунка и решения пазла. Вращение фигуры может быть выполнена по-разному: правой, средней или левой кнопкой мыши. После чего вызовется мнова метод отрисовки, так как матрица из фигур была изменена.

Приложение В – схема последовательности, которая показывает взаимодействие пользователя с игрой. Рассмотрим схему подробнее.

Пользователь запускает игру. Происходит инициализация кнопок и панелей, если пользователь не выбрал фигуру размер пазла, то игра устанавливает значения по умолчанию. Стратегия игры создает фигуру, а из нее массив фигур, на котором генерируется пазл. После чего случайный пазл отрисовывается на панели. Пользователь может взаимодействовать с панелью путем нажатия на фигуры правой, средней и левой кнопкой мыши, что приведет к повороту фигуры и пазл отрисуется по новой.

## Фабричный и шаблонный метод проектирования

### 5.1 Фабричный метод

**Фабричный метод** — это порождающий паттерн проектирования, который определяет общий интерфейс для создания объектов в суперклассе, позволяя подклассам изменять тип создаваемых объектов. Паттерн Фабричный метод предлагает создавать объекты не напрямую, используя оператор **new**, а через вызов особого фабричного метода.

На первый взгляд, это может показаться бессмысленным: перемещения вызова конструктора из одного конца программы в другой. Но теперь появилась возможность переопределить фабричный метод в подклассе, чтобы изменить тип создаваемого продукта.

Чтобы эта система заработала, все возвращаемые объекты должны иметь общий интерфейс. Подклассы смогут производить объекты различных классов, следующих одному и тому же интерфейсу.

В данном случае, при разработке игры классы **Шестиугольник** и **Квадрат** реализуют интерфейс **Фигуры** с методами **отрисовка и вращения**, но каждый из этих классов реализует метод по-своему. Для клиента фабричного метода нет разницы между этими объектами, так как он будет трактовать их как некий абстрактный **класс Фигуры.** Для него будет важно, чтобы объект имел метод **отрисовка и вращения**, а как конкретно он работает — не важно.

**Преимущества и недостатки**

* Избавляет класс от привязки к конкретным классам продуктов.
* Выделяет код производства продуктов в одно место, упрощая поддержку кода.
* Упрощает добавление новых продуктов в программу.
* Реализует принцип открытости/закрытости.
* Может привести к созданию больших [параллельных иерархий классов](https://refactoring.guru/ru/smells/parallel-inheritance-hierarchies), так как для каждого класса продукта надо создать свой подкласс создателя.

Для данной игры лучшим методом проектирования стал шаблонный метод, так как реальные фигуры должны расширять базовый алгоритм, не меняя его структуры.

### 5.2. Шаблонный метод

**Шаблонный метод** — это поведенческий паттерн проектирования, который определяет скелет алгоритма, перекладывая ответственность за некоторые его шаги на подклассы. Паттерн позволяет подклассам переопределять шаги алгоритма, не меняя его общей структуры.

Паттерн Шаблонный метод предлагает разбить алгоритм на последовательность шагов, описать эти шаги в отдельных методах и вызывать их в одном *шаблонном* методе друг за другом. Это позволит подклассам переопределять некоторые шаги алгоритма, оставляя без изменений его структуру и остальные шаги, которые для этого подкласса не так важны.

Для начала шаги шаблонного метода можно сделать абстрактными. Из-за этого все подклассы должны будут реализовать каждый из шагов по-своему. Затем определить общее для всех классов поведение и вынести его в суперкласс. На примере игры шаги построения и отрисовки сетки из фигур могут отличаться для разных типов фигур, поэтому некоторые методы останутся абстрактными.

Таким образом, получается два вида шагов: *абстрактные*, которые каждый подкласс обязательно должен реализовать, а также шаги *с реализацией по умолчанию*, которые можно переопределять в подклассах, но не обязательно.

Аналогия из жизни

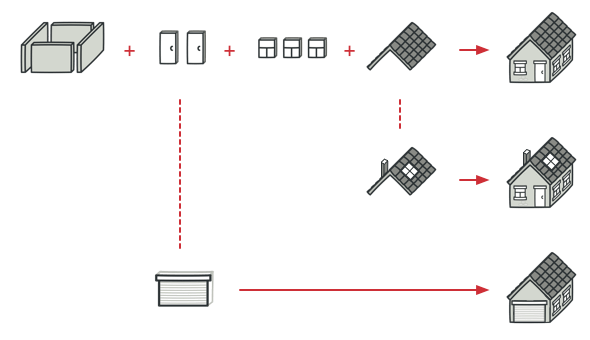


Рисунок 17 – Пример шаблонного метода из жизни

На рисунке 17 изображен проект типового дома, который строители могут немного изменить по желанию клиента.

Строители используют подход, похожий на шаблонный метод при строительстве типовых домов. У них есть основной архитектурный проект, в котором расписаны шаги строительства: заливка фундамента, постройка стен, перекрытие крыши, установка окон и так далее. Но, несмотря на стандартизацию каждого этапа, строители могут вносить небольшие изменения на любом из этапов, чтобы сделать дом чуточку непохожим на другие.

Рассмотрим структуру шаблонного метода из рисунка 18:

1. **Абстрактный класс** определяет шаги алгоритма и содержит шаблонный метод, состоящий из вызовов этих шагов. Шаги могут быть как абстрактными, так и содержать реализацию по умолчанию.
2. **Конкретный класс** переопределяет некоторые (или все) шаги алгоритма. Конкретные классы не переопределяют сам шаблонный метод.

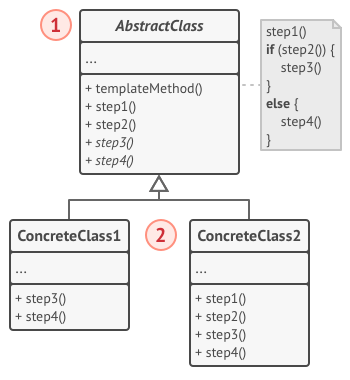


Рисунок 18 – Структура шаблонного метода

PanelStrategy – абстрактный класс, в котором реализована основная логика игры. В этом классе реализована логика нажатия на фигуру, а также отрисовка всей матрицы из определенных фигур. В этом классе определены абстрактные методы масштабирования фигурной сетки, которые реализованы в классах HexPanel и SquarePanel (рисунок 19).

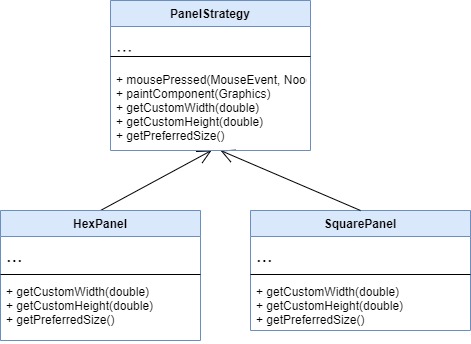


Рисунок 19 – UML-схема применения шаблонного метода для расширения класса PanelStrategy

Такая же логика применена для абстрактного класса Noodle – это логическая модель фигуры, хранящая такие параметры, как:

* какие стороны фигуры пересекаются связями;
* правильно ли поставлена фигура;
* насколько повернута фигура.

Этот класс расширяется классами Hex и Square, которые задают разные фигуры.

HexLayout и SquareLayout расширяют абстрактный класс Layout, который является физической моделью фигуры, просчитывается центр фигуры, углы и стороны.

Такая же структура показана на рисунке 20 построения матрицы фигур.

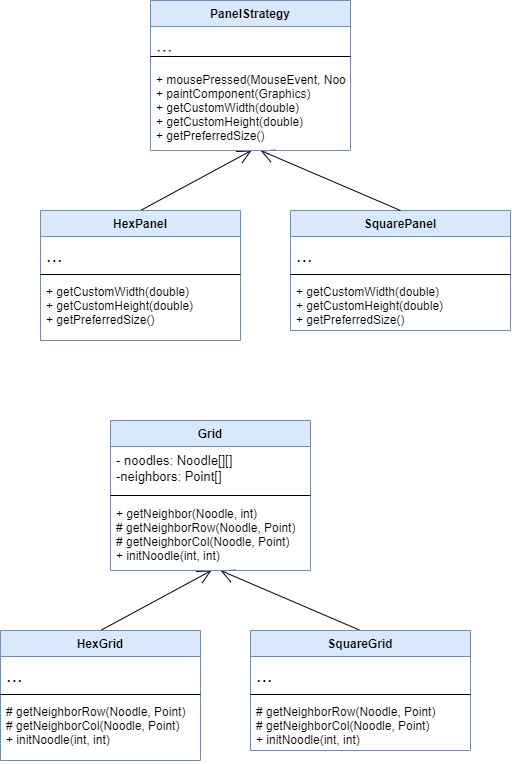


Рисунок 20 – UML-схема применения шаблонного метода для расширения класса Grid

**Применимость**

* Когда подклассы должны расширять базовый алгоритм, не меняя его структуры.
* Шаблонный метод позволяет подклассам расширять определённые шаги алгоритма через наследование, не меняя при этом структуру алгоритмов, объявленную в базовом классе.
* Когда есть несколько классов, делающих одно и то же с незначительными отличиями. Если вы редактируете один класс, то приходится вносить такие же правки и в остальные классы.
* Паттерн шаблонный метод предлагает создать для похожих классов общий суперкласс и оформить в нём главный алгоритм в виде шагов. Отличающиеся шаги можно переопределить в подклассах.
* Это позволит убрать дублирование кода в нескольких классах с похожим поведением, но отличающихся в деталях.

## Заключение

В ходе работы был выполнен анализ литературных исследований, выявлены основные недостатки и преимущества существующих алгоритмов, а также были разработаны алгоритмы построения геометрических фигур, проведен анализ и сравнение результатов, полученных при использовании разработанных алгоритмов и существующих уже алгоритмов. Во время выполнения задания курсовой работы, были углублены знания по многим аспектам языка Java. Основная цель работы была выполнена, т.е. была написана логическая игра «Пазл» с графическим интерфейсом.

По итогу данного проекта, было разработано программное обеспечение похожее на игру «Noodles!», которую можно найти в AppStore. В дальнейшем данный проект может быть развит и абстрактные классы могут быть расширены другими фигурами. Данная работа помогла мне разобраться в паттернах проектирования, практически применить архитектурный шаблон проектирования MVC и шаблонный метод проектирования.

## Список использованных источников

[1] Иван Бодягин. Model-View-Controller в .Net. Model-View-Presenter и сопутствующие паттерны. - журнал RSDN Magazine #2-2006.

[2] The difference between Model-View-ViewModel and other separated presentation patterns. [Электронный ресурс]. – Режим доступа: <https://www.beyondjava.net/model-view-whatever>

[3] Composite Application Guidance for WPF - June 2008. Presentation Model: msdn.

[4] Git Документация [Электронный ресурс]. – Режим доступа: <https://git-scm.com/book/ru/>

[6] Паттерны проектирования [Электронный ресурс]. – Режим доступа: <https://refactoring.guru/ru/design-patterns/>

[7] Пакет java.awt. [Электронный ресурс]. – Режим доступа: <https://wm-help.net/lib/b/book/3683783285/34>

[8] Hexagonal Grid [Электронный ресурс]. – Режим доступа: <https://www.redblobgames.com/grids/hexagons/>

## ПРИЛОЖЕНИЕ А (обязательное)

## Схема алгоритма генерации пазла

## ПРИЛОЖЕНИЕ Б (обязательное)

## Диаграмма состояний игровой панели

## ПРИЛОЖЕНИЕ В (обязательное)

## Диаграмма последовательности игры «Пазл»

## Приложение Д (обязательное) Листинг кода с комментариями

**Пакет noodles**

**ColorTheme.java**

package noodles;

import java.awt.Color;

import java.util.\*;

public class ColorTheme{

private static final Map<String, Color[]> THEMES = initThemes();

private static Map<String, Color[]> initThemes(){

Map<String, Color[]> themes = new LinkedHashMap<>();

themes.put("Brick Red", initTheme(new Color(92, 47, 35)));

themes.put("Copper Orange", initTheme(new Color(112, 64, 16)));

themes.put("Forest Green", initTheme(new Color(50, 80, 20)));

themes.put("Sea Blue", initTheme(new Color(15, 60, 100)));

themes.put("Lavender Purple", initTheme(new Color(70, 40, 100)));

themes.put("Stone Gray", initTheme(new Color(67, 64, 58)));

themes.put("Stranded", initTheme(new Color(20,110,140), new Color(180, 150, 100), new Color(220,192,146)));

themes.put("Volcano", initTheme(new Color(55, 40, 40), new Color(250, 100, 10), new Color(150, 40, 20)));

return Collections.unmodifiableMap(themes);

}//end initThemes

//get methods

public static String[] getThemeNames(){

Set<String> keys = THEMES.keySet();

String[] sortedKeys = new String[keys.size()];

sortedKeys = keys.toArray(sortedKeys);

return sortedKeys;

}

public static Color[] getTheme(String theme){return THEMES.get(theme);}

private static Color[] initTheme(Color color){

Color[] colors = new Color[3];

float[] hsbVals = new float[3];

Color.RGBtoHSB(color.getRed(), color.getGreen(), color.getBlue(), hsbVals);

colors[0] = color;

colors[1] = Color.getHSBColor(hsbVals[0], Math.max(hsbVals[1] - 0.2f, 0.0f), Math.min(hsbVals[2] + 0.4f, 1.0f));

colors[2] = Color.getHSBColor(hsbVals[0], Math.max(hsbVals[1] - 0.1f, 0.0f), Math.min(hsbVals[2] + 0.2f, 1.0f));

return colors;

}//end initTheme (single color)

private static Color[] initTheme(Color powerColor, Color noodleColor, Color baseColor){

Color[] colors = new Color[3];

colors[0] = powerColor;

colors[1] = noodleColor;

colors[2] = baseColor;

return colors;

}//end initTheme (multiple colors)

}//end ColorTheme

**Grid.java**

**package** noodles;

**import** java.awt.\*;

**public** **abstract** **class** Grid{

**private** Noodle[][] noodles;

**private** Point[] neighbors;

**public** Grid(**int** rows, **int** cols, Point[] neighbors){

**this**.neighbors = neighbors;

**this**.initNoodles(rows, cols);

}

//get methods

**public** Noodle getNoodle(**int** row, **int** col){**return** **this**.noodles[row][col];}

**public** Noodle[][] getNoodles(){**return** **this**.noodles;}

**public** **int** getRows(){**return** **this**.noodles.length;}

**public** **int** getCols(){**return** **this**.noodles[0].length;}

**public** **abstract** **int** getRow(Noodle noodle);

**public** **abstract** **int** getCol(Noodle noodle);

**public** Noodle getNeighbor(Noodle noodle, **int** dir){

Point coords = **this**.neighbors[dir];

**int** row = getNeighborRow(noodle, coords);

**int** col = getNeighborCol(noodle, coords);

**int** rows = **this**.noodles.length;

**int** cols = **this**.noodles[0].length;

**if**(row >= 0 && row < rows && col >= 0 && col < cols)

**return** **this**.noodles[row][col];

**return** **null**;

}

**public** Point[] getNeighbors(){**return** **this**.neighbors;}

**protected** **abstract** **int** getNeighborRow(Noodle noodle, Point coords);

**protected** **abstract** **int** getNeighborCol(Noodle noodle, Point coords);

//setMethods

**public** **void** setNoodles(Noodle[][] noodles){**this**.noodles = noodles;}

**public** **void** setNeighbors(Point[] neighbors){**this**.neighbors = neighbors;}

**public** **void** initNoodles(**int** rows, **int** cols){

**this**.noodles = **new** Noodle[rows][cols];

**for**(**int** row = 0; row < rows; row++){

**for**(**int** col = 0; col < cols; col++){

noodles[row][col] = initNoodle(row, col);

}

}

}//end initNoodles

**public** **void** resetActiveSides(**int** sideCount){

**for**(Noodle[] row : **this**.noodles){

**for**(Noodle noodle : row){

noodle.setActiveSides(**new** **boolean**[sideCount]);

noodle.setRotation(0);

}

}

}//end resetActiveSides

**public** **void** resetPower(){

**for**(Noodle[] row : **this**.noodles){

**for**(Noodle noodle : row){

noodle.setPowered(**false**);

}

}

}//end resetPower

**public** **abstract** Noodle initNoodle(**int** row, **int** col);

}//end Grid class

**Layout.java**

package noodles;

import java.awt.Shape;

import java.awt.geom.\*;

public abstract class Layout{

public static final double PADDINGRATIO = 0.05;

public static final double NOODLERATIO = 0.375;

private double size, marginX, marginY, innerScale, outerScale, cornerAngle, edgeAngle;

private int segments;

public Layout(double innerScale, double outerScale, double cornerAngle, double edgeAngle, int segments){

this.size = 27;

this.marginX = 0;

this.marginY = 0;

this.innerScale = innerScale;

this.outerScale = outerScale;

this.cornerAngle = cornerAngle;

this.edgeAngle = edgeAngle;

this.segments = segments;

}

//get methods

public double getSize(){return this.size;}

public double getPadding(){return this.size \* PADDINGRATIO;}

public double getMarginX(){return this.marginX;}

public double getMarginY(){return this.marginY;}

public double getCornerAngle(){return this.cornerAngle;}

public int getSegments(){return this.segments;}

//set methods

public void setSize(double size){this.size = (size >= 1 ? size : 1);}

public void setMarginX(double marginX){this.marginX = marginX;}

public void setMarginY(double marginY){this.marginY = marginY;}

public void setSegments(int segments){this.segments = segments;}

public Shape getBase(Noodle noodle){

Point2D.Double[] corners = this.getCorners(this.cornerAngle, noodle, this.outerScale);

Path2D.Double path = new Path2D.Double();

path.moveTo(corners[0].getX(), corners[0].getY());

for(int i = 1; i < segments; i++){

path.lineTo(corners[i].getX(), corners[i].getY());

}

path.lineTo(corners[0].getX(), corners[0].getY());

return path;

}//end getBase

private Point2D.Double[] getCorners(double startAngle, Noodle noodle, double scale){

Point2D.Double[] corners = new Point2D.Double[this.segments];

Point2D.Double center = noodleToPixel(noodle);

for(int i = 0; i < this.segments; i++){

Point2D.Double offset = cornerOffset(startAngle, i, scale);

corners[i] = new Point2D.Double(center.getX() + offset.getX(), center.getY() + offset.getY());

}

return corners;

}//end getCorners

//Draw line

public Shape getNoodleSide(Noodle noodle, int side){

Point2D.Double[] edges = this.getCorners(this.edgeAngle, noodle, this.innerScale);

Point2D.Double center = noodleToPixel(noodle);

Line2D.Double edge = new Line2D.Double(center, edges[side]);

return edge;

}//end getNoodleSide

//get hex corner points

private Point2D.Double cornerOffset(double startAngle, int corner, double scale){

double angle = 2 \* Math.PI \* (corner + startAngle) / segments;

return new Point2D.Double(this.size \* scale \* Math.cos(angle), this.size \* scale \* Math.sin(angle));

}//end hexCornerOffset

public abstract Point2D.Double getOrigin();

public abstract int getStroke();

public abstract Shape getNoodleCenter(Noodle noodle, double scale);

public abstract Point2D.Double noodleToPixel(Noodle noodle);

public abstract Noodle pixelToNoodle(Point2D.Double point);

}//end Layout class

**Noodle.java**

**package** noodles;

**public** **abstract** **class** Noodle{

**private** **boolean** powered;

**private** **boolean**[] activeSides;

**private** **int** rotation;

**public** Noodle(**boolean**[] activeSides){

**this**.powered = **false**;

**this**.activeSides = activeSides;

**this**.rotation = 0;

}

**public** Noodle(**boolean** powered, **boolean**[] activeSides){

**this**.powered = powered;

**this**.activeSides = activeSides;

}

//get methods

**public** **boolean** getPowered(){**return** **this**.powered;}

**public** **boolean**[] getActiveSides(){**return** **this**.activeSides;}

**public** **int** getRotation(){**return** **this**.rotation;}

**public** **int** getActiveCount(){

**int** count = 0;

**for**(**boolean** bool : **this**.activeSides){

**if**(bool)

count++;

}

**return** count;

}

//set methods

**public** **void** setPowered(**boolean** powered){**this**.powered = powered;}

**public** **void** setActive(**int** dir, **boolean** active){**this**.activeSides[dir] = active;}

**public** **void** setActiveSides(**boolean**[] activeSides){**this**.activeSides = activeSides;}

**public** **void** setRotation(**int** rotation){**this**.rotation = rotation;}

**public** **boolean** isVisited(){

**boolean** visited = **false**;

**for**(**boolean** bool : **this**.activeSides){

**if**(bool)

visited = **true**;

}

**return** visited;

}//end isVisited

**public** **void** rotate(**int** amount){

**int** size = **this**.activeSides.length;

amount %= size;

**if**(amount < 0)

amount += size;

**for**(**int** i = 0; i < amount; i++){

**boolean**[] temp = **new** **boolean**[size];

**for**(**int** j = 0; j < size; j++)

temp[j] = **this**.activeSides[((j + size) - 1) % size];

**this**.activeSides = temp;

}

**this**.rotation += amount;

}//end rotate

}//end Noodle class

**NoodlePanel.java**

**package** noodles;

**import** java.awt.\*;

**import** javax.swing.\*;

**import** java.awt.event.\*;

**public** **class** NoodlePanel **extends** JPanel **implements** MouseListener, MouseMotionListener, ComponentListener{

**private** PanelStrategy strategy;

**private** Graphics2D g2;

**public** NoodlePanel(PanelStrategy strategy){

**this**.strategy = strategy;

addMouseListener(**this**);

addMouseMotionListener(**this**);

addComponentListener(**this**);

}

//get methods

**public** PanelStrategy getStrategy(){**return** **this**.strategy;}

**public** **int** getRows(){**return** **this**.strategy.getRows();}

**public** **int** getCols(){**return** **this**.strategy.getCols();}

**public** String getThemeName(){**return** **this**.strategy.getThemeName();}

**public** Color[] getTheme(){**return** **this**.strategy.getTheme();}

//set methods

**public** **void** setRows(**int** rows){

**this**.strategy.setRows(rows);

**this**.resetView();

}

**public** **void** setCols(**int** cols){

**this**.strategy.setCols(cols);

**this**.resetView();

}

**public** **void** setStrategy(PanelStrategy strategy){

**this**.strategy = strategy;

**this**.resetView();

}

**public** **void** setTheme(String theme){

**this**.strategy.setTheme(theme);

**this**.repaint();

}

**public** **void** regenerate(){

**this**.strategy.regenerate();

**this**.repaint();

}//end regenerate

**public** **void** resetView(){

**this**.strategy.setCustomSize(**this**.getSize());

**this**.revalidate();

**this**.repaint();

}//end resetView

**public** **void** mouseClicked(MouseEvent e){}

**public** **void** mouseEntered(MouseEvent e){}

**public** **void** mouseExited(MouseEvent e){}

**public** **void** mouseMoved(MouseEvent e){}

**public** **void** mousePressed(MouseEvent e){

**this**.strategy.mousePressed(e, **this**);

**this**.repaint();

}

**public** **void** mouseReleased(MouseEvent e){

**this**.strategy.mouseReleased();

**this**.repaint();

}

**public** **void** mouseDragged(MouseEvent e){

**this**.strategy.mouseDragged(e);

**this**.repaint();

}

**public** **void** massageSolved() {

JOptionPane.*showMessageDialog*(**null**, "Solved!");

}

**public** **void** componentResized(ComponentEvent e){

**this**.strategy.setCustomSize(e.getComponent().getSize());

**this**.repaint();

}

**public** **void** componentHidden(ComponentEvent e){}

**public** **void** componentMoved(ComponentEvent e){}

**public** **void** componentShown(ComponentEvent e){}

**protected** **void** paintComponent(Graphics g){

**this**.g2 = (Graphics2D)g;

g2.setRenderingHint(RenderingHints.***KEY\_ANTIALIASING***, RenderingHints.***VALUE\_ANTIALIAS\_ON***);

**super**.paintComponent(g2);

strategy.paintComponent(g2);

}//end paintComponent

**public** Dimension getPreferredSize(){

**return** **this**.strategy.getPreferredSize();

}//end getPreferredSize

}//end NoodlePanel class

**Noodles.java**

**package** noodles;

**import** java.awt.\*;

**import** javax.swing.\*;

**import** java.awt.event.\*;

**public** **class** NoodlePanel **extends** JPanel **implements** MouseListener, MouseMotionListener, ComponentListener{

**private** PanelStrategy strategy;

**private** Graphics2D g2;

**public** NoodlePanel(PanelStrategy strategy){

**this**.strategy = strategy;

addMouseListener(**this**);

addMouseMotionListener(**this**);

addComponentListener(**this**);

}

//get methods

**public** PanelStrategy getStrategy(){**return** **this**.strategy;}

**public** **int** getRows(){**return** **this**.strategy.getRows();}

**public** **int** getCols(){**return** **this**.strategy.getCols();}

**public** String getThemeName(){**return** **this**.strategy.getThemeName();}

**public** Color[] getTheme(){**return** **this**.strategy.getTheme();}

//set methods

**public** **void** setRows(**int** rows){

**this**.strategy.setRows(rows);

**this**.resetView();

}

**public** **void** setCols(**int** cols){

**this**.strategy.setCols(cols);

**this**.resetView();

}

**public** **void** setStrategy(PanelStrategy strategy){

**this**.strategy = strategy;

**this**.resetView();

}

**public** **void** setTheme(String theme){

**this**.strategy.setTheme(theme);

**this**.repaint();

}

**public** **void** regenerate(){

**this**.strategy.regenerate();

**this**.repaint();

}//end regenerate

**public** **void** resetView(){

**this**.strategy.setCustomSize(**this**.getSize());

**this**.revalidate();

**this**.repaint();

}//end resetView

**public** **void** mouseClicked(MouseEvent e){}

**public** **void** mouseEntered(MouseEvent e){}

**public** **void** mouseExited(MouseEvent e){}

**public** **void** mouseMoved(MouseEvent e){}

**public** **void** mousePressed(MouseEvent e){

**this**.strategy.mousePressed(e, **this**);

**this**.repaint();

}

**public** **void** mouseReleased(MouseEvent e){

**this**.strategy.mouseReleased();

**this**.repaint();

}

**public** **void** mouseDragged(MouseEvent e){

**this**.strategy.mouseDragged(e);

**this**.repaint();

}

**public** **void** massageSolved() {

JOptionPane.*showMessageDialog*(**null**, "Solved!");

}

**public** **void** componentResized(ComponentEvent e){

**this**.strategy.setCustomSize(e.getComponent().getSize());

**this**.repaint();

}

**public** **void** componentHidden(ComponentEvent e){}

**public** **void** componentMoved(ComponentEvent e){}

**public** **void** componentShown(ComponentEvent e){}

**protected** **void** paintComponent(Graphics g){

**this**.g2 = (Graphics2D)g;

g2.setRenderingHint(RenderingHints.***KEY\_ANTIALIASING***, RenderingHints.***VALUE\_ANTIALIAS\_ON***);

**super**.paintComponent(g2);

strategy.paintComponent(g2);

}//end paintComponent

**public** Dimension getPreferredSize(){

**return** **this**.strategy.getPreferredSize();

}//end getPreferredSize

}//end NoodlePanel class

**Orientation.java**

**package** noodles;

**public** **class** Orientation{

**private** **double**[][] forward;

**private** **double**[][] inverse;

**private** **double** startAngle;

**public** Orientation(**double** f0, **double** f1, **double** f2, **double** f3, **double** b0, **double** b1, **double** b2, **double** b3){

**this**.forward = **new** **double**[][]{{f0, f1}, {f2, f3}};

**this**.inverse = **new** **double**[][]{{b0, b1}, {b2, b3}};

}

//get methods

**public** **double**[][] getForward(){**return** **this**.forward;}

**public** **double**[][] getInverse(){**return** **this**.inverse;}

}//end Orientation class

PanelStrategy.java

**package** noodles;

**import** noodles.NoodlePanel;

**import** java.awt.\*;

**import** java.awt.event.\*;

**import** java.awt.geom.\*;

**public** **abstract** **class** PanelStrategy{

**private** **int** rows, cols, segment;

**private** Layout layout;

**private** Grid grid;

**private** Puzzle puzzle;

**private** BasicStroke noodleStroke, baseStroke;

**private** String themeName;

**private** Color[] theme;

**private** Point2D.Double clickPoint, mousePoint;

**private** Noodle draggedNoodle;

**public** PanelStrategy(**int** rows, **int** cols, Layout layout, Grid grid, String themeName){

**this**.rows = rows;

**this**.cols = cols;

**this**.segment = -1;

**this**.layout = layout;

**this**.grid = grid;

**this**.baseStroke = **new** BasicStroke();

**this**.setTheme(themeName);

}

//get methods

**public** **int** getRows(){**return** **this**.rows;}

**public** **int** getCols(){**return** **this**.cols;}

**public** Layout getLayout(){**return** **this**.layout;}

**public** Grid getGrid(){**return** **this**.grid;}

**public** Puzzle getPuzzle(){**return** **this**.puzzle;}

**public** String getThemeName(){**return** **this**.themeName;}

**public** Color[] getTheme(){**return** **this**.theme;}

//set methods

**public** **void** setLayout(Layout layout){**this**.layout = layout;}

**public** **void** setGrid(Grid grid){**this**.grid = grid;}

**public** **void** setPuzzle(Puzzle puzzle){**this**.puzzle = puzzle;}

**public** **void** setNoodleStroke(BasicStroke noodleStroke){**this**.noodleStroke = noodleStroke;}

**public** **void** setRows(**int** rows){

**this**.rows = rows;

**this**.grid.initNoodles(**this**.rows, **this**.cols);

**this**.regenerate();

}

**public** **void** setCols(**int** cols){

**this**.cols = cols;

**this**.grid.initNoodles(**this**.rows, **this**.cols);

**this**.regenerate();

}

**public** **void** setTheme(String themeName){

**this**.themeName = themeName;

**this**.theme = ColorTheme.getTheme(themeName);

}

**public** **void** regenerate(){

**this**.puzzle.initPuzzle(**this**.grid);

}//end regenerate

**public** **int** getSegment(Point2D.Double p1, Point2D.Double p2){

**int** segments = **this**.layout.getSegments();

**double** angle = Math.*toDegrees*(Math.*atan2*(p1.getY() - p2.getY(), p1.getX() - p2.getX()));

angle += (360 / segments) \* **this**.layout.getCornerAngle();

angle %= 360;

**if**(angle < 0)

angle +=360;

**return** (**int**)((angle / (360 / segments)) % segments);

}

**public** **void** mousePressed(MouseEvent e, NoodlePanel panel){

**if**(**this**.puzzle.isSolved()) {

**this**.puzzle.initPuzzle(**this**.grid);

panel.massageSolved();

}

**else**{

Point2D.Double point = **new** Point2D.Double(e.getX(), e.getY());

Noodle noodle = **this**.layout.pixelToNoodle(point);

**int** row = **this**.grid.getRow(noodle);

**int** col = **this**.grid.getCol(noodle);

**if**(row >= 0 && row < **this**.rows && col >= 0 && col < **this**.cols){

noodle = **this**.grid.getNoodle(row, col);

**if**(**this**.layout.getBase(noodle).contains(point)){

**if**(e.getButton() == 1){

**if**(**this**.layout.getNoodleCenter(noodle, .2).contains(point))

noodle.rotate(-1);

**else**{

**this**.clickPoint = point;

**this**.draggedNoodle = noodle;

}

}

**else** **if**(e.getButton() == 2)

noodle.rotate(-1 \* noodle.getRotation());

**else** **if**(e.getButton() == 3)

noodle.rotate(1);

}

**this**.puzzle.testPowered(**this**.grid);

}

}

}//end mousePressed

**public** **void** mouseReleased(){

**if**(**this**.draggedNoodle != **null**){

**this**.segment = -1;

**this**.draggedNoodle = **null**;

**this**.clickPoint = **null**;

**this**.mousePoint = **null**;

}

**this**.puzzle.testPowered(**this**.grid);

}//end mouseReleased

**public** **void** mouseDragged(MouseEvent e){

**if**(clickPoint != **null**){

**if**(**this**.segment < 0)

**this**.segment = **this**.getSegment(**this**.clickPoint, **this**.layout.noodleToPixel(draggedNoodle));

**this**.mousePoint = **new** Point2D.Double(e.getX(), e.getY());

**int** difference = **this**.getSegment(**this**.mousePoint, **this**.layout.noodleToPixel(draggedNoodle)) - **this**.segment;

**this**.draggedNoodle.rotate(difference);

**this**.segment += difference;

}

}//end mouseDragged

**public** **void** paintComponent(Graphics2D g2){

**this**.noodleStroke = **new** BasicStroke(**this**.layout.getStroke(), BasicStroke.***CAP\_BUTT***, BasicStroke.***JOIN\_MITER***);

g2.setStroke(**this**.baseStroke);

g2.setColor(**this**.theme[2]);

Noodle[][] noodles = **this**.grid.getNoodles();

Noodle cur;

**for**(**int** row = 0; row < **this**.rows; row++){

**for**(**int** col = 0; col < **this**.cols; col++){

cur = noodles[row][col];

**this**.paintNoodle(g2, cur, row, col);

}

}

}//end paintComponent

**public** **void** paintNoodle(Graphics2D g2, Noodle noodle, **int** row, **int** col){

g2.fill(**this**.layout.getBase(noodle));

g2.setColor(noodle.getPowered() ? **this**.theme[0] : **this**.theme[1]);

g2.setStroke(**this**.noodleStroke);

**boolean**[] activeSides = noodle.getActiveSides();

**for**(**int** i = 0; i < activeSides.length; i++){

**if**(activeSides[i]){

g2.draw(**this**.layout.getNoodleSide(noodle, i));

}

}

g2.fill(**this**.layout.getNoodleCenter(noodle, .2));

**if**(**this**.drawSource(noodle, row, col)){

g2.setColor(**this**.theme[2]);

g2.fill(**this**.layout.getNoodleCenter(noodle, -.1));

}

g2.setColor(**this**.theme[2]);

}//end paintNoodle

**public** **boolean** drawSource(Noodle noodle, **int** row, **int** col){///?????

**if**(noodle.equals(**this**.puzzle.getSource()))

**return** **true**;

**if**(row == **this**.grid.getRow(**this**.puzzle.getSource()) && (col == **this**.grid.getNoodles()[0].length && **this**.grid.getCol(puzzle.getSource()) == 0))

**return** **true**;

**if**(col == **this**.grid.getCol(**this**.puzzle.getSource()) && (row == **this**.grid.getNoodles().length && **this**.grid.getRow(puzzle.getSource()) == 0))

**return** **true**;

**return** **false**;

}

**public** **void** setCustomSize(Dimension newSize){

Dimension oldSize = **this**.getPreferredSize();

**double** newRatio = newSize.getWidth() / newSize.getHeight();

**double** oldRatio = oldSize.getWidth() / oldSize.getHeight();

**if**(newRatio < oldRatio){

**this**.layout.setSize(getCustomWidth(newSize.getWidth()));

**this**.layout.setMarginX(0);

**this**.layout.setMarginY((newSize.getHeight() - **this**.getPreferredSize().getHeight()) / 2);

}

**else**{

**this**.layout.setSize(getCustomHeight(newSize.getHeight()));

**this**.layout.setMarginX((newSize.getWidth() - **this**.getPreferredSize().getWidth()) / 2);

**this**.layout.setMarginY(0);

}

}//end setCustomSize

**public** **abstract** **double** getCustomWidth(**double** newWidth);

**public** **abstract** **double** getCustomHeight(**double** newHeight);

**public** **abstract** Dimension getPreferredSize();

}//end PanelStrategy class

**Пакет noodles.hex:**

**FractionalHex.java**

**package** noodles.hex;

**public** **class** FractionalHex {

**private** **double** q;

**private** **double** r;

**public** FractionalHex(**double** q, **double** r){

**this**.q = q;

**this**.r = r;

}//end EVC

//get methods

**public** **double** getQ(){**return** **this**.q;}

**public** **double** getR(){**return** **this**.r;}

**public** **double** getS(){**return** -**this**.q - **this**.r;}

**public** **static** Hex hexRound(FractionalHex frac){

**int** q = (**int**)(Math.*round*(frac.getQ()));

**int** r = (**int**)(Math.*round*(frac.getR()));

**int** s = (**int**)(Math.*round*(frac.getS()));

**double** qDiff = Math.*abs*(q - frac.getQ());

**double** rDiff = Math.*abs*(r - frac.getR());

**double** sDiff = Math.*abs*(s - frac.getS());

**if**(qDiff > rDiff && qDiff > sDiff)

q = -r - s;

**else** **if**(rDiff > sDiff)

r = -q - s;

**else**

s = -q - r;

**return** **new** Hex(q, r);

}//end hexRound

}//end FractionalHex class

Hex.java

**package** noodles.hex;

**import** noodles.Noodle;

**public** **class** Hex **extends** Noodle{

**private** **int** q;

**private** **int** r;

**public** Hex(**int** q, **int** r){

**super**(**new** **boolean**[6]);

**this**.q = q;

**this**.r = r;

}

**public** Hex(**int** q, **int** r, **boolean** powered, **boolean**[] activeSides){

**super**(powered, activeSides);

**this**.q = q;

**this**.r = r;

}

//get methods

**public** **int** getQ(){**return** **this**.q;}

**public** **int** getR(){**return** **this**.r;}

**public** **int** getS(){**return** -**this**.q - **this**.r;}

}//end Hex class

**HexGrid.java**

package noodles.hex;

import noodles.\*;

import java.awt.\*;

public class HexGrid extends Grid{

public HexGrid(int rows, int cols){

super(rows, cols, new Point[]{new Point(1, 0), new Point(0, 1), new Point(-1, 1), new Point(-1, 0), new Point(0, -1), new Point(1, -1)});

}

//get methods

public int getRow(Noodle noodle){return ((Hex)noodle).getR() + (((Hex)noodle).getQ()+1)/2;}

public int getCol(Noodle noodle){return ((Hex)noodle).getQ();}

public int getNeighborRow(Noodle noodle, Point coords){return (((Hex)noodle).getR() + coords.y) + ((((Hex)noodle).getQ() + coords.x)+1)/2;}

public int getNeighborCol(Noodle noodle, Point coords){return ((Hex)noodle).getQ() + coords.x;}

public Noodle initNoodle(int row, int col){

int q = col;

int r = row - (col+1)/2;

return new Hex(q, r);

}

}//end HexGrid class

**HexLayout.java**

package noodles.hex;

import noodles.\*;

import java.awt.Shape;

import java.awt.geom.\*;

public class HexLayout extends Layout{

public static final Orientation FLAT = new Orientation(3/2d, 0, Math.sqrt(3)/2d, Math.sqrt(3), 2/3d, 0, -1/3d, Math.sqrt(3)/3d);

public static final Orientation POINTY = new Orientation(Math.sqrt(3), Math.sqrt(3)/2d, 0, 3/2d, Math.sqrt(3)/3d, -1/3d, 0, 2/3d);

public HexLayout(){

super(Math.sqrt(3)/2, 1, 0, 0.5d, 6); //new

}

public Point2D.Double getOrigin(){

double xOrigin = super.getSize() + super.getPadding() + super.getMarginX();

double yOrigin = 2 \* ((super.getSize() + super.getPadding()) \* Math.sqrt(3)/2d) + super.getMarginY();

return new Point2D.Double(xOrigin, yOrigin);

}//end getOrigin

public int getStroke(){

return super.getSize() \* Layout.NOODLERATIO > 0 ? (int)(super.getSize() \* Layout.NOODLERATIO) : 1;

}//end getStroke

public Shape getNoodleCenter(Noodle noodle, double scale){

Point2D.Double center = noodleToPixel(noodle);

double size = super.getSize() \* (Layout.NOODLERATIO + scale);

return new Ellipse2D.Double(center.getX() - size/2, center.getY() - size/2, size, size);

}//end getNoodleCenter

public Point2D.Double noodleToPixel(Noodle noodle){

Hex hex = (Hex)noodle;

double[][] matrix = HexLayout.FLAT.getForward();

double x = (matrix[0][0] \* hex.getQ() + matrix[0][1] \* hex.getR()) \* (super.getSize() + super.getPadding());

double y = (matrix[1][0] \* hex.getQ() + matrix[1][1] \* hex.getR()) \* (super.getSize() + super.getPadding());

return new Point2D.Double(x + this.getOrigin().getX(), y + this.getOrigin().getY());

}//end noodleToPixel

public Noodle pixelToNoodle(Point2D.Double point){

double x = (point.getX() - this.getOrigin().getX()) / (super.getSize() + super.getPadding());

double y = (point.getY() - this.getOrigin().getY()) / (super.getSize() + super.getPadding());

double[][] matrix = this.FLAT.getInverse();

double q = (matrix[0][0] \* x + matrix[0][1] \* y);

double r = (matrix[1][0] \* x + matrix[1][1] \* y);

return FractionalHex.hexRound(new FractionalHex(q, r));

}//end pixelToNoodle

}//end HexLayout class

**HexPanel.java**

package noodles.hex;

import noodles.\*;

import java.awt.\*;

import java.awt.event.\*;

import java.awt.geom.\*;

public class HexPanel extends PanelStrategy{

public HexPanel(int rows, int cols, String themeName){

super(rows, cols, new HexLayout(), new HexGrid(rows, cols), themeName);

super.setPuzzle(new HexPuzzle(super.getGrid()));

}

public double getCustomWidth(double newWidth){

double size = newWidth / ((((3/4d) \* (super.getCols() + 1))) \* 2);

return size / (Layout.PADDINGRATIO + 1);

}//end getCustomWidth

public double getCustomHeight(double newHeight){

double size = newHeight / (Math.sqrt(3) \* (super.getRows() + 1/2d));

return size / (Layout.PADDINGRATIO + 1);

}//end getCustomHeight

public Dimension getPreferredSize(){

double width = (super.getLayout().getSize() + super.getLayout().getPadding())\*2;

double height = (Math.sqrt(3)/2d) \* width;

int x = (int)(width \* (((3/4d) \* (super.getCols()))) + 1);

int y = (int)((height \* super.getRows()) + (height / 2));

return new Dimension(x, y);

}//end getPreferredSize

}//end HexPanel class

**HexPuzzle.java**

**package** noodles.hex;

**import** noodles.\*;

**public** **class** HexPuzzle **extends** Puzzle{

**public** HexPuzzle(Grid grid){

**super**(grid, **new** **int**[]{0, 1, 2, 3, 4, 5}, 4);

}

}//end HexPuzzle class

**Пакет noodles.square:**

**Square.java**

**package** noodles.square;

**import** noodles.Noodle;

**public** **class** Square **extends** Noodle{

**private** **int** row;

**private** **int** col;

**public** Square(**int** row, **int** col){

**super**(**new** **boolean**[4]);

**this**.row = row;

**this**.col = col;

}

**public** Square(**int** row, **int** col, **boolean** powered, **boolean**[] activeSides){

**super**(powered, activeSides);

**this**.row = row;

**this**.col = col;

}

//get methods

**public** **int** getRow(){**return** **this**.row;}

**public** **int** getCol(){**return** **this**.col;}

}//end Square class

**SquareGrid.java**

package noodles.square;

import noodles.\*;

import java.awt.\*;

public class SquareGrid extends Grid{

public SquareGrid(int rows, int cols){

super(rows, cols, new Point[]{new Point(1, 0), new Point(0, 1), new Point(-1, 0), new Point(0, -1)});

}

//get methods

public int getRow(Noodle noodle){return ((Square)noodle).getRow();}

public int getCol(Noodle noodle){return ((Square)noodle).getCol();}

public int getNeighborRow(Noodle noodle, Point coords){return ((Square)noodle).getRow() + coords.y;}

public int getNeighborCol(Noodle noodle, Point coords){return ((Square)noodle).getCol() + coords.x;}

public Noodle initNoodle(int row, int col){

return new Square(row, col);

}

}//end SquareGrid class

**SquareLayout.java**

package noodles.square;

import noodles.\*;

import java.awt.Shape;

import java.awt.geom.\*;

public class SquareLayout extends Layout{

public SquareLayout(){

super(1, Math.sqrt(2), 0.5d, 0, 4);

}

public Point2D.Double getOrigin(){

double xOrigin = super.getSize() + super.getPadding() + super.getMarginX();

double yOrigin = super.getSize() + super.getPadding() + super.getMarginY();

return new Point2D.Double(xOrigin, yOrigin);

}//end getOrigin

public int getStroke(){

return super.getSize() \* 1.5 \* Layout.NOODLERATIO > 0 ? (int)(super.getSize() \* 1.5 \* Layout.NOODLERATIO) : 1;

}//end getStroke

public Shape getNoodleCenter(Noodle noodle, double scale){

Point2D.Double center = this.noodleToPixel(noodle);

double size = super.getSize() \* 1.5 \* (Layout.NOODLERATIO + scale);

double round = size - this.getStroke();

return new RoundRectangle2D.Double(center.getX() - size/2, center.getY() - size/2, size, size, round, round);

}//end getNoodleCenter

public Point2D.Double noodleToPixel(Noodle noodle){

Square square = (Square)noodle;

double x = square.getCol() \* ((super.getSize() + super.getPadding()) \* 2);

double y = square.getRow() \* ((super.getSize() + super.getPadding()) \* 2);

return new Point2D.Double(x + this.getOrigin().getX(), y + this.getOrigin().getY());

}//end noodleToPixel

public Noodle pixelToNoodle(Point2D.Double point){

double x = (point.getX() - this.getOrigin().getX()) / ((super.getSize() + super.getPadding()) \* 2);

double y = (point.getY() - this.getOrigin().getY()) / ((super.getSize() + super.getPadding()) \* 2);

int row = (int)(Math.round(y));

int col = (int)(Math.round(x));

return new Square(row, col);

}//end pixelToNoodle

}//end SquareLayout class

SquarePanel.java

**package** noodles.square;

**import** noodles.\*;

**import** java.awt.\*;

**import** java.awt.event.\*;

**import** java.awt.geom.\*;

**public** **class** SquarePanel **extends** PanelStrategy{

**public** SquarePanel(**int** rows, **int** cols, String themeName){

**super**(rows, cols, **new** SquareLayout(), **new** SquareGrid(rows, cols), themeName);

**super**.setPuzzle(**new** SquarePuzzle(**super**.getGrid()));

}

**public** Noodle getGhost(**int** row, **int** col){

**return** **new** Square(row, col, **false**, **new** **boolean**[]{**false**, **false**, **false**, **false**});

}//end getGhostNoodle

**public** Noodle getGhost(**int** row, **int** col, **boolean** powered, **boolean**[] activeSides){

**return** **new** Square(row, col, powered, activeSides);

}//end getGhostNoodle

**public** **double** getCustomWidth(**double** newWidth){

**double** size = newWidth / (**super**.getCols() \* 2);

**return** size / (Layout.***PADDINGRATIO*** + 1);

}//end getCustomWidth

**public** **double** getCustomHeight(**double** newHeight){

**double** size = newHeight / (**super**.getRows() \* 2);

**return** size / (Layout.***PADDINGRATIO*** + 1);

}//end getCustomHeight

**public** Dimension getPreferredSize(){

**double** square = (**super**.getLayout().getSize() + **super**.getLayout().getPadding()) \* 2;

**int** x = (**int**)(square \* **super**.getCols());

**int** y = (**int**)(square \* **super**.getRows());

**return** **new** Dimension(x, y);

}//end getPreferredSize

}//end SquarePanel

SquarePuzzle.java

**package** noodles.square;

**import** noodles.\*;

**public** **class** SquarePuzzle **extends** Puzzle{

**public** SquarePuzzle(Grid grid){

**super**(grid, **new** **int**[]{0, 1, 2, 3}, 3);

}

}//end SquarePuzzle