**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

Кафедра Вычислительной техники

**ОТЧЕТ**

по учебной практике

по дисциплине «Объектно-ориентированное программирование»

Тема: «Шаблоны проектирования в языке Java»

|  |  |  |
| --- | --- | --- |
| Студент гр. 1308, |  | Лепов А. В. |
| Научный руководитель, |  | Разумовский Г. В. |

Санкт-Петербург,

2023 г.

**ОГЛАВЛЕНИЕ**

[1. ПОРОЖДАЮЩИЙ ФАБРИЧНЫЙ МЕТОД 3](#_Toc139816288)

[1.1. Описание шаблона 3](#_Toc139816289)

[1.2. Случаи использования 3](#_Toc139816290)

[1.3. Пример реализации 3](#_Toc139816291)

[1.4. Диаграмма Классов 3](#_Toc139816292)

[1.5. UML-код диаграммы классов 4](#_Toc139816293)

[1.6. Диаграмма последовательности выполнения 4](#_Toc139816294)

[1.7. UML-код диаграммы последовательности выполнения 5](#_Toc139816295)

[1.8. Исходный код реализации 5](#_Toc139816296)

[1.9. Результат работы 6](#_Toc139816297)

[2. СТРУКТУРНЫЙ КОМПОЗИТНЫЙ МЕТОД 7](#_Toc139816298)

[2.1. Описание шаблона 7](#_Toc139816299)

[2.2. Случаи использования 7](#_Toc139816300)

[2.3. Пример реализации 7](#_Toc139816301)

[2.4. Диаграмма классов 8](#_Toc139816302)

[2.5. UML-код диаграммы классов 8](#_Toc139816303)

[2.6. Диаграмма последовательности выполнения 9](#_Toc139816304)

[2.7. UML-код диаграммы последовательности выполнения 10](#_Toc139816305)

[2.8. Исходный код реализации 11](#_Toc139816306)

[2.9. Результат работы 12](#_Toc139816307)

[3. ПОВЕДЕНЧЕСКИЙ ШАБЛОН СТРАТЕГИЯ 13](#_Toc139816308)

[3.1. Описание шаблона 13](#_Toc139816309)

[3.2. Случаи использования 13](#_Toc139816310)

[3.3. Пример реализации 13](#_Toc139816311)

[3.4. Диаграмма классов 14](#_Toc139816312)

[3.5. UML-код диаграммы классов 14](#_Toc139816313)

[3.6. Диаграмма последовательности взаимодействия 15](#_Toc139816314)

[3.7. UML-код диаграммы последовательности взаимодействия 15](#_Toc139816315)

[3.8. Исходный код реализации 16](#_Toc139816316)

[3.9. Результат работы 17](#_Toc139816317)

1. **ПОРОЖДАЮЩИЙ ФАБРИЧНЫЙ МЕТОД**
   1. **Описание шаблона**

Шаблон проектирования фабричный метод – это порождающий шаблон, который предоставляет интерфейс для создания объектов, но позволяет подклассам решить, какой класс инстанцировать (создать экземпляр класса). Этот шаблон инкапсулирует логику создания объектов в отдельном методе, который реализуется подклассами для создания объектов различных типов.

* 1. **Случаи использования**

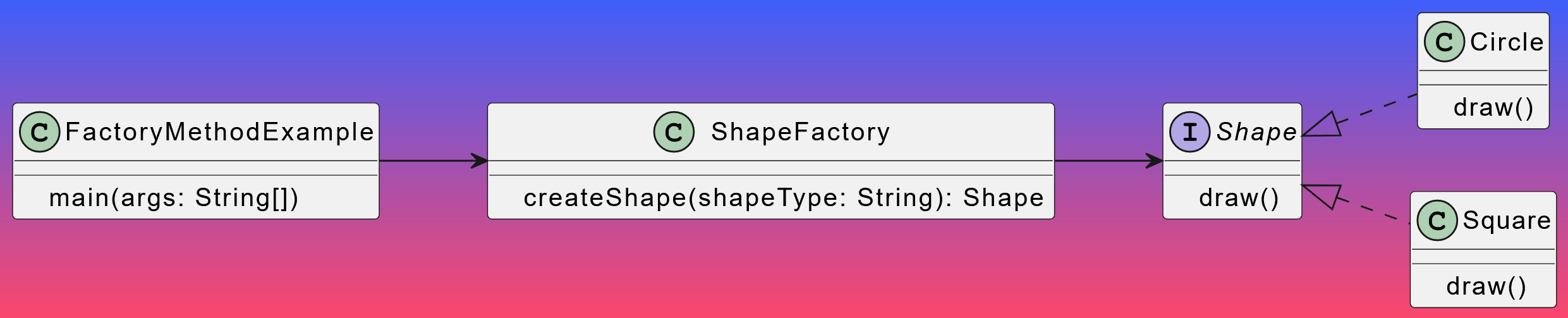
Когда есть суперкласс с несколькими подклассами, и нужно делегировать ответственность за создание объекта подклассам. Или же когда нужно предоставить общий интерфейс для создания объекта, но хочется позволить подклассам определить конкретную реализацию.

* 1. **Пример реализации**

В этом примере мы создадим простое консольное приложение на языке Java, которое демонстрирует использование шаблона Фабричного метода. У нас будет интерфейс Shape, представляющий различные фигуры, и две конкретные реализации: Circle и Square. Класс фабрики, называемый ShapeFactory, будет иметь статический метод createShape, который принимает строковый аргумент, представляющий желаемую фигуру, и возвращает экземпляр соответствующего класса фигуры.

* 1. **Диаграмма Классов**

Построена с помощью онлайн-инструментария: <https://www.planttext.com/>



* 1. **UML-код диаграммы классов**

left to right direction

skinparam{

componentStyle uml2

classAttributeIconSize 1

handwritten false

backgroundcolor #3f5efb-fc466b

}

@startuml

interface Shape {

+ draw()

}

class Circle {

+ draw()

}

class Square {

+ draw()

}

class ShapeFactory {

+ createShape(shapeType: String): Shape

}

class FactoryMethodExample {

+ main(args: String[])

}

Shape <|.. Circle

Shape <|.. Square

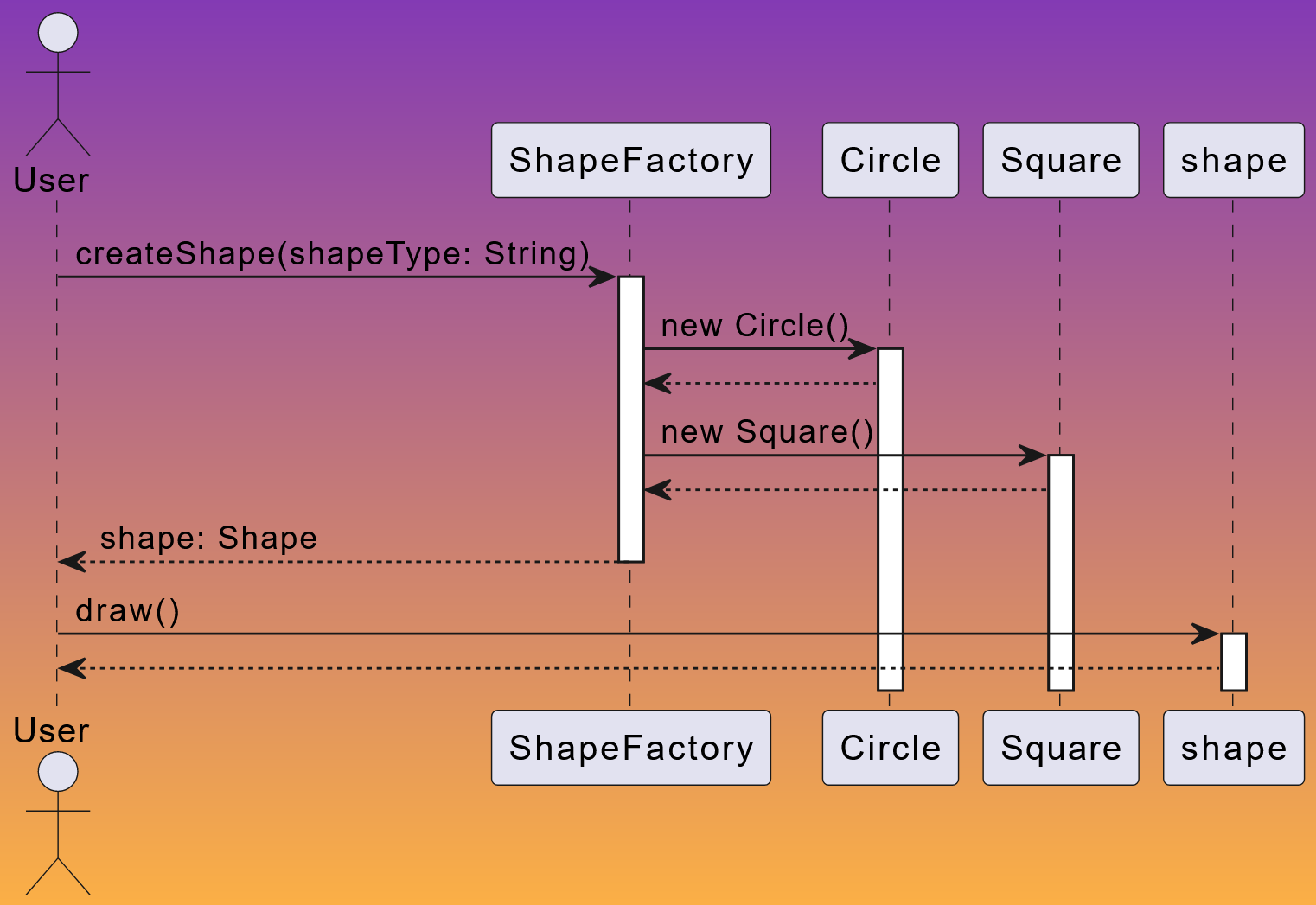
ShapeFactory --> Shape

FactoryMethodExample --> ShapeFactory

@enduml

* 1. **Диаграмма последовательности выполнения**

Построена с помощью онлайн-инструментария: <https://www.planttext.com/>



* 1. **UML-код диаграммы последовательности выполнения**

skinparam{

componentStyle uml2

classAttributeIconSize 1

handwritten false

backgroundcolor #833ab4-fcb045

}

@startuml

actor User

User -> ShapeFactory: createShape(shapeType: String)

activate ShapeFactory

ShapeFactory -> Circle: new Circle()

activate Circle

Circle --> ShapeFactory

ShapeFactory -> Square: new Square()

activate Square

Square --> ShapeFactory

ShapeFactory --> User: shape: Shape

deactivate ShapeFactory

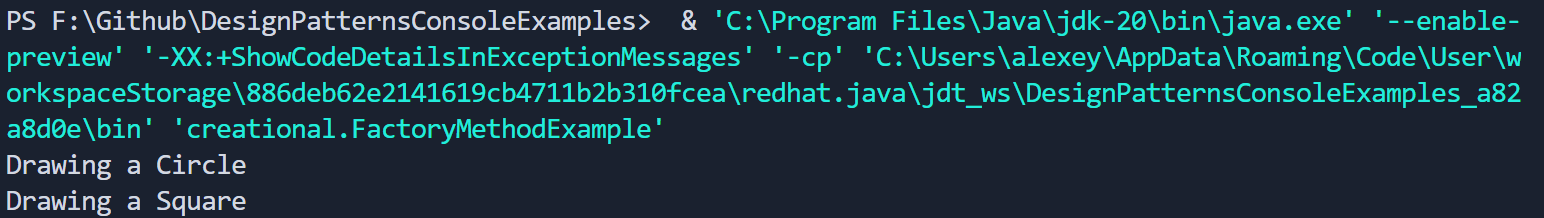
User -> shape: draw()

activate shape

shape --> User

@enduml

* 1. **Исходный код реализации**
     1. package creational;
     2. interface Shape {
     3. void draw();
     4. }
     5. class Circle implements Shape {
     6. @Override
     7. public void draw() {
     8. System.out.println("Drawing a Circle");
     9. }
     10. }
     11. class Square implements Shape {
     12. @Override
     13. public void draw() {
     14. System.out.println("Drawing a Square");
     15. }
     16. }
     17. class ShapeFactory {
     18. public static Shape createShape(String shapeType) {
     19. if (shapeType.equalsIgnoreCase("circle")) {
     20. return new Circle();
     21. } else if (shapeType.equalsIgnoreCase("square")) {
     22. return new Square();
     23. }
     24. return null;
     25. }
     26. }
     27. public class FactoryMethodExample {
     28. public static void main(String[] args) {
     29. Shape circle = ShapeFactory.createShape("circle");
     30. circle.draw();
     31. Shape square = ShapeFactory.createShape("square");
     32. square.draw();
     33. }
     34. }
  2. **Результат работы**



Программа демонстрирует использование паттерна фабричного метода. У нас есть класс ShapeFactory, который действует как фабрика для создания различных фигур. Статический метод createShape принимает строковый аргумент, представляющий желаемую фигуру, и возвращает экземпляр соответствующего класса фигуры. В основном методе мы создаем круг и квадрат с помощью фабричного метода, а затем вызываем метод draw для каждой фигуры, чтобы вывести соответствующие сообщения. Вывод подтверждает, что объекты были созданы и нарисованы правильно.

1. **СТРУКТУРНЫЙ КОМПОЗИТНЫЙ МЕТОД**
   1. **Описание шаблона**

Композитный метод – это шаблон проектирования, который позволяет одинаково обрабатывать отдельные объекты и группы объектов. Он создает древовидную структуру, где как отдельные объекты, так и группы объектов представлены общим интерфейсом.

Этот метод полезен, когда требуется выполнять операции либо над отдельным объектом, либо над группой объектов, без необходимости различать их.

* 1. **Случаи использования**

Представление иерархических структур:

Композитный метод подходит для представления иерархических структур, где объекты могут быть как листовыми узлами, так и составными узлами.

Выполнение операций над группами объектов:

Если требуется выполнять операции над группами объектов, например, вычисление общей стоимости корзины покупок или вывод структуры каталога, композитный метод упрощает реализацию.

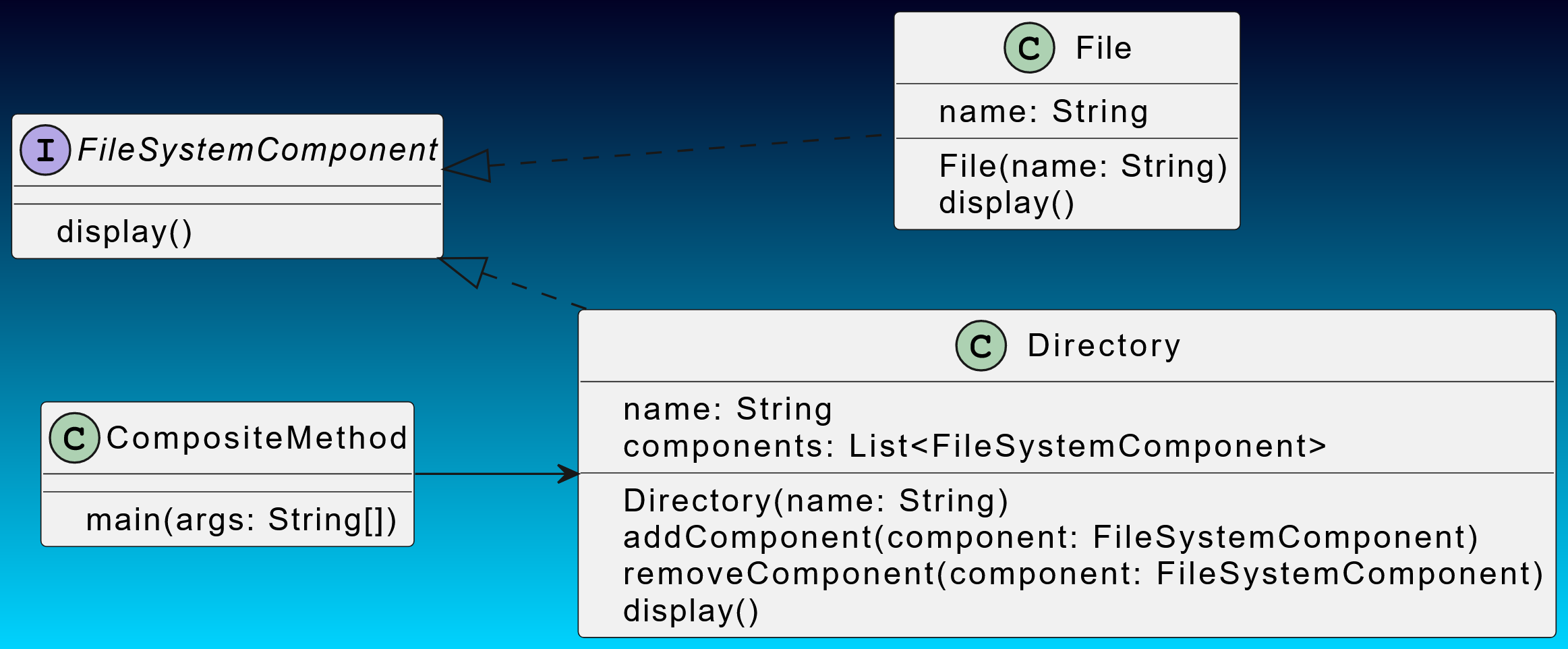
* 1. **Пример реализации**

Данный пример будет рассматривать композитный метод для представления структуры каталога. У нас будет два типа объектов: файл – «File» и каталог – «Directory». Класс «File» представляет отдельные файлы, а класс «Directory» представляет каталоги, которые могут содержать как файлы, так и подкаталоги.

Программа будет создавать структуру каталога, состоящую из файлов и каталогов. Затем она будет отображать всю структуру каталога, используя композитный метод.

* 1. **Диаграмма классов**

Построена с помощью онлайн-инструментария: <https://www.planttext.com/>

****

* 1. **UML-код диаграммы классов**

left to right direction

skinparam{

componentStyle uml2

classAttributeIconSize 1

handwritten false

backgroundcolor #020024-00d4ff

}

@startuml

interface FileSystemComponent {

+display()

}

class File {

-name: String

+File(name: String)

+display()

}

class Directory {

-name: String

-components: List<FileSystemComponent>

+Directory(name: String)

+addComponent(component: FileSystemComponent)

+removeComponent(component: FileSystemComponent)

+display()

}

class CompositeMethod {

+main(args: String[])

}

FileSystemComponent <|.. File

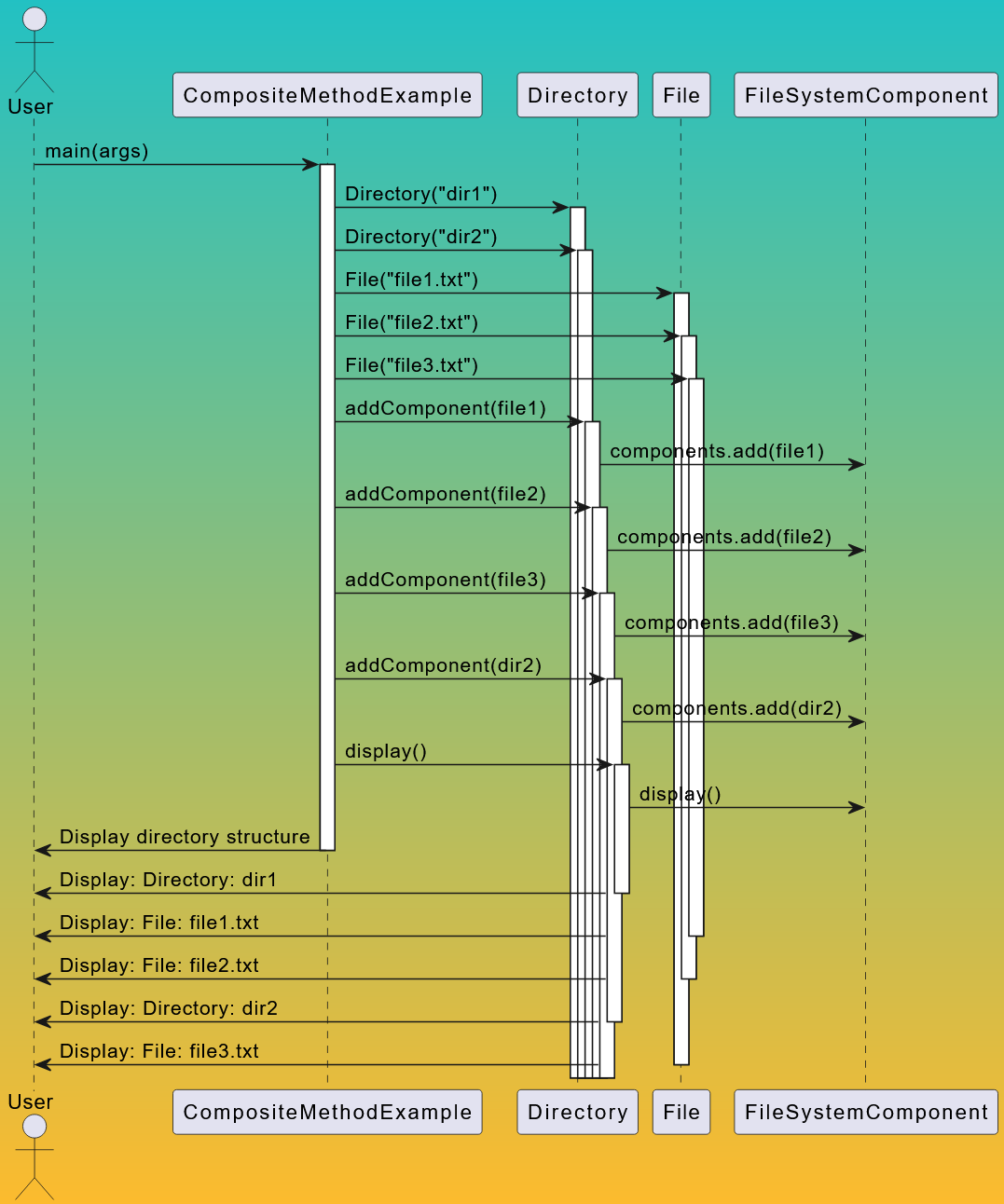
FileSystemComponent <|.. Directory

CompositeMethod --> Directory

@enduml

* 1. **Диаграмма последовательности выполнения**

Построена с помощью онлайн-инструментария: <https://www.planttext.com/>

****

* 1. **UML-код диаграммы последовательности выполнения**

skinparam{

componentStyle uml2

classAttributeIconSize 1

handwritten false

backgroundcolor #22c1c3-fdbb2d

}

@startuml

actor User

User -> CompositeMethodExample: main(args)

activate CompositeMethodExample

CompositeMethodExample -> Directory: Directory("dir1")

activate Directory

CompositeMethodExample -> Directory: Directory("dir2")

activate Directory

CompositeMethodExample -> File: File("file1.txt")

activate File

CompositeMethodExample -> File: File("file2.txt")

activate File

CompositeMethodExample -> File: File("file3.txt")

activate File

CompositeMethodExample -> Directory: addComponent(file1)

activate Directory

Directory -> FileSystemComponent: components.add(file1)

CompositeMethodExample -> Directory: addComponent(file2)

activate Directory

Directory -> FileSystemComponent: components.add(file2)

CompositeMethodExample -> Directory: addComponent(file3)

activate Directory

Directory -> FileSystemComponent: components.add(file3)

CompositeMethodExample -> Directory: addComponent(dir2)

activate Directory

Directory -> FileSystemComponent: components.add(dir2)

CompositeMethodExample -> Directory: display()

activate Directory

Directory -> FileSystemComponent: display()

User <- CompositeMethodExample: Display directory structure

deactivate CompositeMethodExample

User <- Directory: Display: Directory: dir1

deactivate Directory

User <- Directory: Display: File: file1.txt

deactivate File

User <- Directory: Display: File: file2.txt

deactivate File

User <- Directory: Display: Directory: dir2

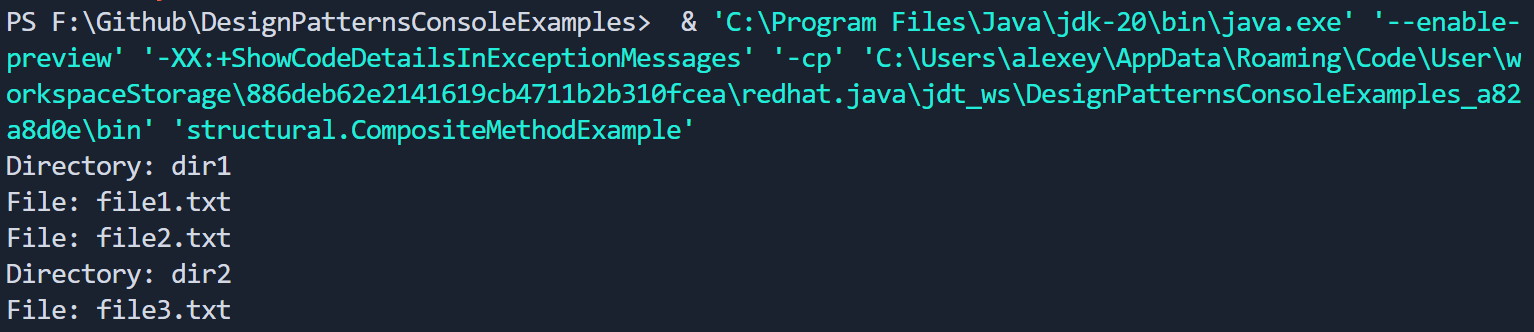
deactivate Directory

User <- Directory: Display: File: file3.txt

deactivate File

@enduml

* 1. **Исходный код реализации**
     1. package structural;
     2. import java.util.ArrayList;
     3. import java.util.List;
     4. interface FileSystemComponent {
     5. void display();
     6. }
     7. class File implements FileSystemComponent {
     8. private String name;
     9. public File(String name) {
     10. this.name = name;
     11. }
     12. public void display() {
     13. System.out.println("File: " + name);
     14. }
     15. }
     16. class Directory implements FileSystemComponent {
     17. private String name;
     18. private List<FileSystemComponent> components;
     19. public Directory(String name) {
     20. this.name = name;
     21. this.components = new ArrayList<>();
     22. }
     23. public void addComponent(FileSystemComponent component) {
     24. components.add(component);
     25. }
     26. public void removeComponent(FileSystemComponent component) {
     27. components.remove(component);
     28. }
     29. public void display() {
     30. System.out.println("Directory: " + name);
     31. for (FileSystemComponent component : components) {
     32. component.display();
     33. }
     34. }
     35. }
     36. public class CompositeMethodExample {
     37. public static void main(String[] args) {
     38. File file1 = new File("file1.txt");
     39. File file2 = new File("file2.txt");
     40. File file3 = new File("file3.txt");
     41. Directory dir1 = new Directory("dir1");
     42. Directory dir2 = new Directory("dir2");
     43. dir1.addComponent(file1);
     44. dir1.addComponent(file2);
     45. dir2.addComponent(file3);
     46. dir1.addComponent(dir2);
     47. dir1.display();
     48. }
     49. }
  2. **Результат работы**



Вывод демонстрирует, как композитный метод позволяет одинаково обращаться к файлам и каталогам. Мы можем добавлять файлы и каталоги друг к другу, и при отображении структуры каталога правильно отображается иерархия файлов и каталогов.

1. **ПОВЕДЕНЧЕСКИЙ ШАБЛОН СТРАТЕГИЯ**
   1. **Описание шаблона**

Метод поведенческой стратегии является шаблоном проектирования, который позволяет инкапсулировать набор алгоритмов и делает их взаимозаменяемыми внутри одной программы. Этот шаблон предоставляет гибкий способ определения семейства алгоритмов и динамического выбора одного из них во время выполнения, в зависимости от конкретного контекста или условий.

* 1. **Случаи использования**

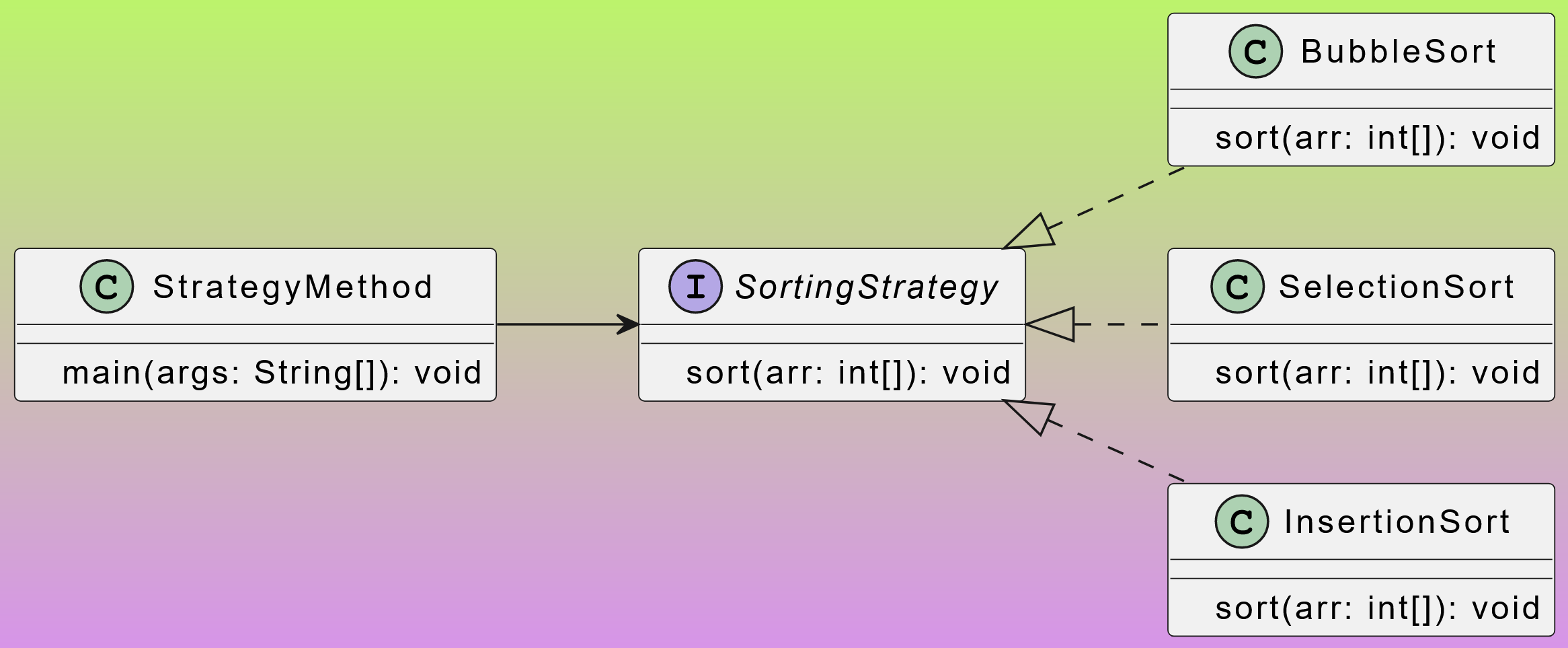
Используется в случаях когда:

* У вас есть несколько алгоритмов, которые могут взаимозаменяться для выполнения конкретной задачи.
* Вы хотите отделить алгоритмы от клиентского кода, что упрощает добавление, удаление или изменение алгоритмов без влияния на общую структуру.
* Вам нужно динамически выбирать алгоритм во время выполнения, исходя из различных факторов или условий.
  1. **Пример реализации**

Программа демонстрирует метод поведенческой стратегии, реализуя различные алгоритмы сортировки (сортировка пузырьком, сортировка выбором и сортировка вставками) и позволяя пользователю выбрать алгоритм сортировки во время выполнения. Программа предлагает пользователю ввести список чисел, а затем сортирует числа с использованием выбранного алгоритма.

* 1. **Диаграмма классов**

Построена с помощью онлайн-инструментария: <https://www.planttext.com/>



* 1. **UML-код диаграммы классов**

left to right direction

skinparam{

componentStyle uml2

classAttributeIconSize 1

handwritten false

backgroundcolor #bcf46b-d794e9

}

@startuml

interface SortingStrategy {

+sort(arr: int[]): void

}

class BubbleSort {

+sort(arr: int[]): void

}

class SelectionSort {

+sort(arr: int[]): void

}

class InsertionSort {

+sort(arr: int[]): void

}

class StrategyMethod {

+main(args: String[]): void

}

SortingStrategy <|.. BubbleSort

SortingStrategy <|.. SelectionSort

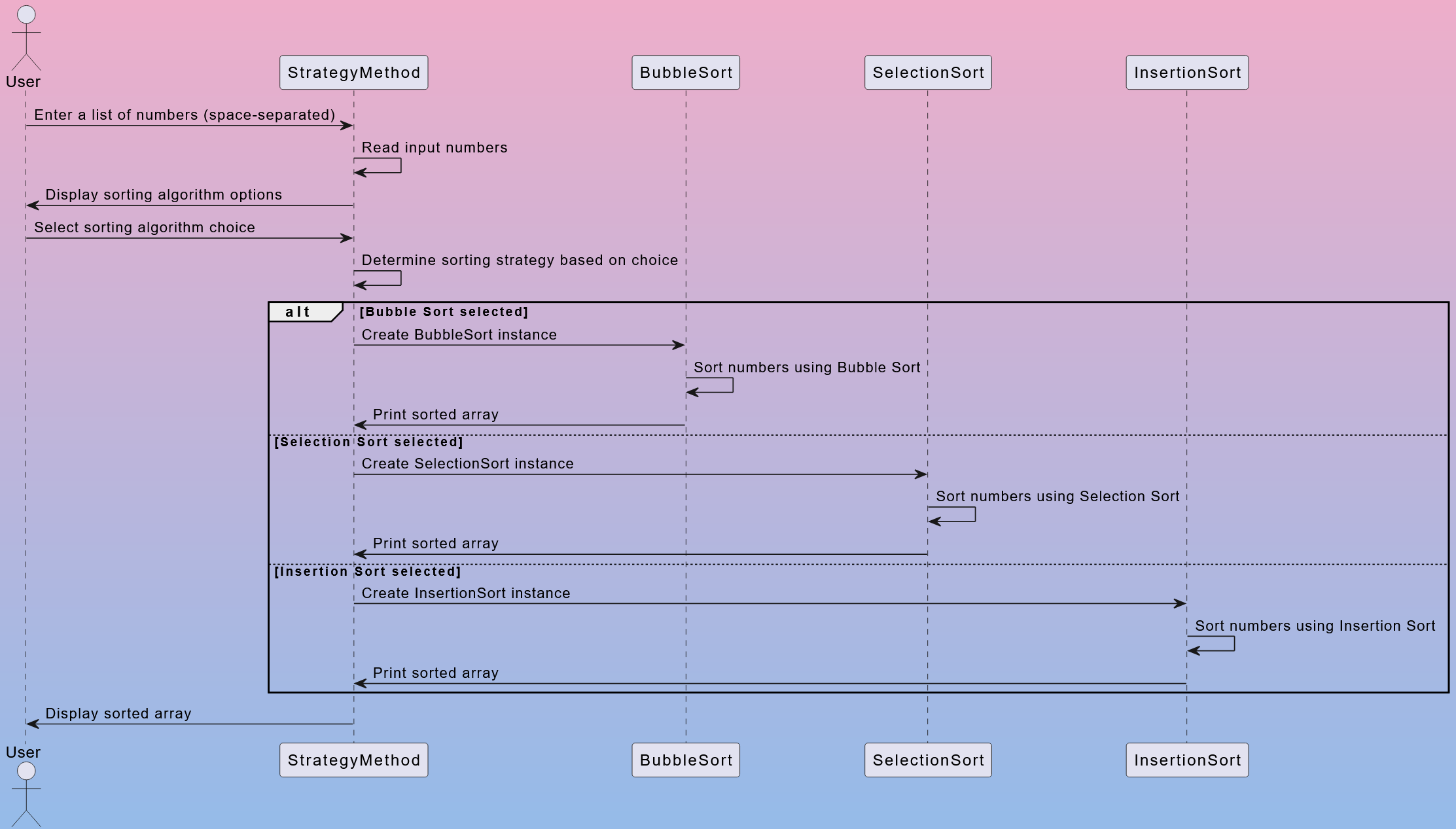
SortingStrategy <|.. InsertionSort

StrategyMethod --> SortingStrategy

@enduml

* 1. **Диаграмма последовательности взаимодействия**

Построена с помощью онлайн-инструментария: <https://www.planttext.com/>



* 1. **UML-код диаграммы последовательности взаимодействия**

skinparam{

componentStyle uml2

classAttributeIconSize 1

handwritten false

backgroundcolor #eeaeca-94bbe9

}

@startuml

actor User

participant StrategyMethod

participant BubbleSort

participant SelectionSort

participant InsertionSort

User -> StrategyMethod: Enter a list of numbers (space-separated)

StrategyMethod -> StrategyMethod: Read input numbers

StrategyMethod -> User: Display sorting algorithm options

User -> StrategyMethod: Select sorting algorithm choice

StrategyMethod -> StrategyMethod: Determine sorting strategy based on choice

alt Bubble Sort selected

StrategyMethod -> BubbleSort: Create BubbleSort instance

BubbleSort -> BubbleSort: Sort numbers using Bubble Sort

BubbleSort -> StrategyMethod: Print sorted array

else Selection Sort selected

StrategyMethod -> SelectionSort: Create SelectionSort instance

SelectionSort -> SelectionSort: Sort numbers using Selection Sort

SelectionSort -> StrategyMethod: Print sorted array

else Insertion Sort selected

StrategyMethod -> InsertionSort: Create InsertionSort instance

InsertionSort -> InsertionSort: Sort numbers using Insertion Sort

InsertionSort -> StrategyMethod: Print sorted array

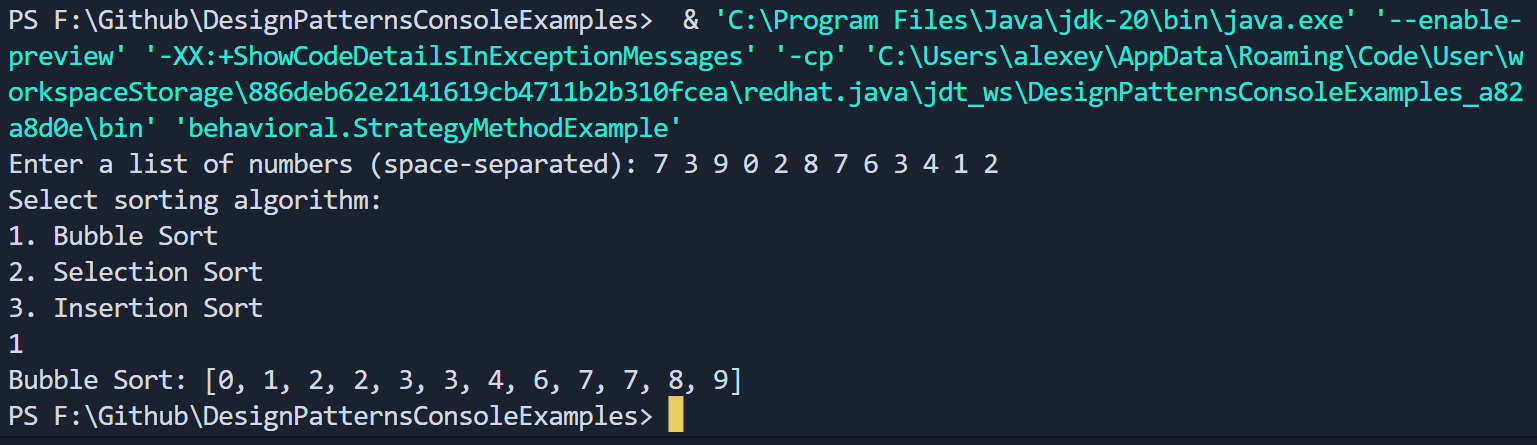
End

StrategyMethod -> User: Display sorted array

@enduml

* 1. **Исходный код реализации**
     1. package behavioral;
     2. import java.util.Arrays;
     3. import java.util.Scanner;
     4. interface SortingStrategy {
     5. void sort(int[] arr);
     6. }
     7. class BubbleSort implements SortingStrategy {
     8. public void sort(int[] arr) {
     9. int n = arr.length;
     10. for (int i = 0; i < n - 1; i++) {
     11. for (int j = 0; j < n - i - 1; j++) {
     12. if (arr[j] > arr[j + 1]) {
     13. int temp = arr[j];
     14. arr[j] = arr[j + 1];
     15. arr[j + 1] = temp;
     16. }
     17. }
     18. }
     19. System.out.println("Bubble Sort: " + Arrays.toString(arr));
     20. }
     21. }
     22. class SelectionSort implements SortingStrategy {
     23. public void sort(int[] arr) {
     24. int n = arr.length;
     25. for (int i = 0; i < n - 1; i++) {
     26. int minIndex = i;
     27. for (int j = i + 1; j < n; j++) {
     28. if (arr[j] < arr[minIndex]) {
     29. minIndex = j;
     30. }
     31. }
     32. int temp = arr[minIndex];
     33. arr[minIndex] = arr[i];
     34. arr[i] = temp;
     35. }
     36. System.out.println("Selection Sort: " + Arrays.toString(arr));
     37. }
     38. }
     39. class InsertionSort implements SortingStrategy {
     40. public void sort(int[] arr) {
     41. int n = arr.length;
     42. for (int i = 1; i < n; ++i) {
     43. int key = arr[i];
     44. int j = i - 1;
     45. while (j >= 0 && arr[j] > key) {
     46. arr[j + 1] = arr[j];
     47. j = j - 1;
     48. }
     49. arr[j + 1] = key;
     50. }
     51. System.out.println("Insertion Sort: " + Arrays.toString(arr));
     52. }
     53. }
     54. public class StrategyMethodExample {
     55. public static void main(String[] args) {
     56. Scanner scanner = new Scanner(System.in);
     57. System.out.print("Enter a list of numbers (space-separated): ");
     58. String input = scanner.nextLine();
     59. int[] numbers = Arrays.stream(input.split(" ")).mapToInt(Integer::parseInt).toArray();
     60. SortingStrategy sortingStrategy;
     61. System.out.println("Select sorting algorithm:");
     62. System.out.println("1. Bubble Sort");
     63. System.out.println("2. Selection Sort");
     64. System.out.println("3. Insertion Sort");
     65. int choice = scanner.nextInt();
     66. switch (choice) {
     67. case 1:
     68. sortingStrategy = new BubbleSort();
     69. break;
     70. case 2:
     71. sortingStrategy = new SelectionSort();
     72. break;
     73. case 3:
     74. sortingStrategy = new InsertionSort();
     75. break;
     76. default:
     77. System.out.println("Invalid choice. Using Bubble Sort.");
     78. sortingStrategy = new BubbleSort();
     79. }
     80. sortingStrategy.sort(numbers);
     81. }
     82. }
  2. **Результат работы**

После запуска программы она предлагает пользователю ввести список чисел. Предположим, что пользователь вводит "7 3 9 0 2 8 7 6 3 4 1 2". Затем программа представляет меню для выбора алгоритма сортировки. Предположим, что пользователь выбирает сортировку пузырьком. Программа выполняет алгоритм сортировки пузырьком для введенного списка и отображает отсортированный массив следующим образом:



Вывод показывает список чисел, отсортированных с использованием выбранного алгоритма (в данном случае сортировка пузырьком).