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

1.1. 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.

1.2. Случаи использования

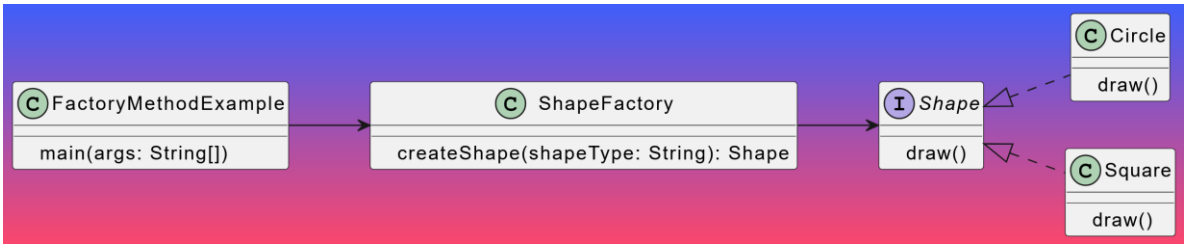
Когда есть суперкласс с несколькими подклассами, и нужно делегировать ответственность за создание объекта подклассам. Или же когда нужно предоставить общий интерфейс для создания объекта, но хочется позволить подклассам определить конкретную реализацию.

1.3. 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.

1.4. Диаграмма Классов

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

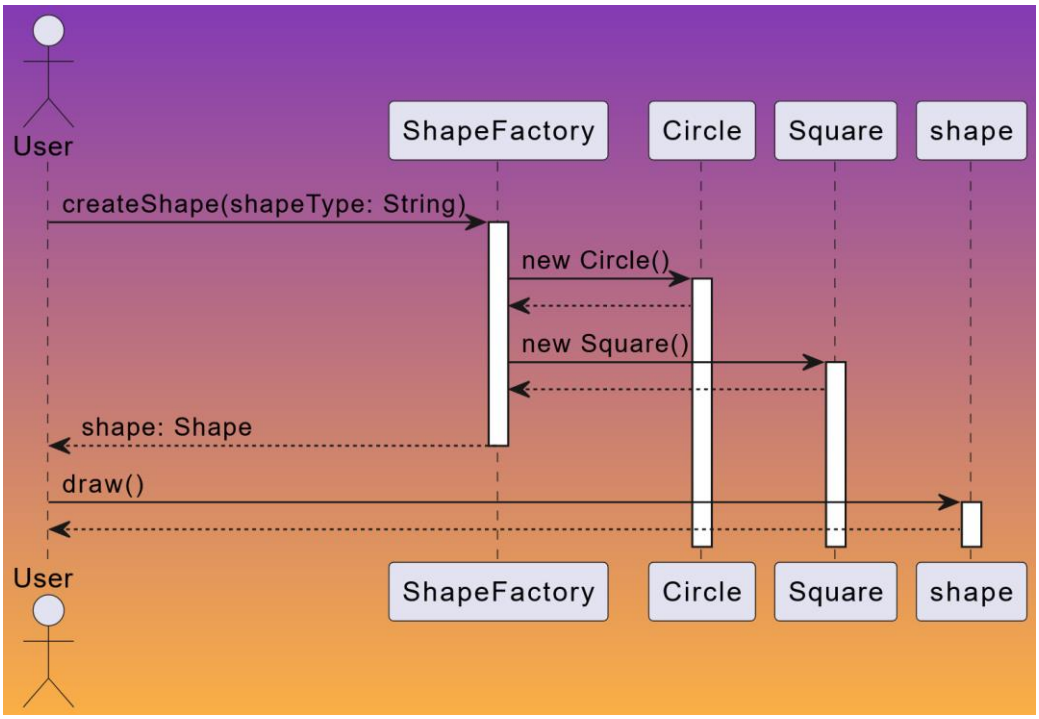


1.5. UML-код диаграммы классов

```
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
```

1.6. Диаграмма последовательности выполнения

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



1.7. 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
```

1.8. Исходный код реализации

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

```
32.    public class FactoryMethodExample {
33.        public static void main(String[] args) {
34.            Shape circle = ShapeFactory.createShape("circle");
35.            circle.draw();
36.
37.            Shape square = ShapeFactory.createShape("square");
38.            square.draw();
39.        }
40.    }
```

1.9. Результат работы

```
PS F:\Github\DesignPatternsConsoleExamples>  & 'C:\Program Files\Java\jdk-20\bin\java.exe' '--enable-
preview' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\alexey\AppData\Roaming\Code\User\w
orkspaceStorage\886deb62e2141619cb4711b2b310fcea\redhat.java\jdt_ws\DesignPatternsConsoleExamples_a82
a8d0e\bin' 'creational.FactoryMethodExample'
Drawing a Circle
Drawing a Square
```

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.

2. СТРУКТУРНЫЙ КОМПОЗИТНЫЙ МЕТОД

2.1. 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.

2.2. 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.

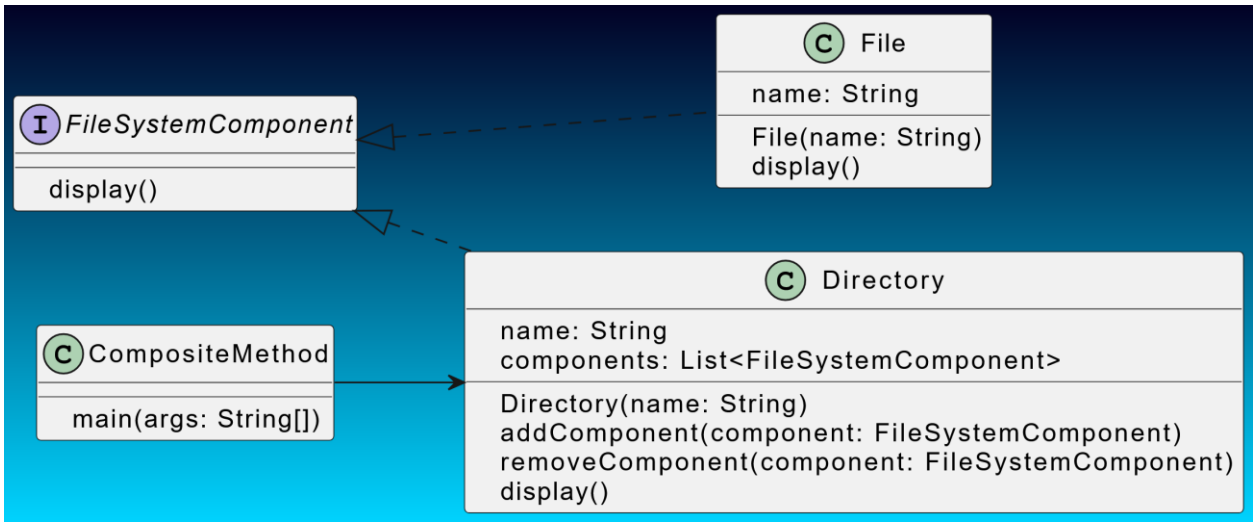
2.3. Пример реализации

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.

2.4. Диаграмма классов

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



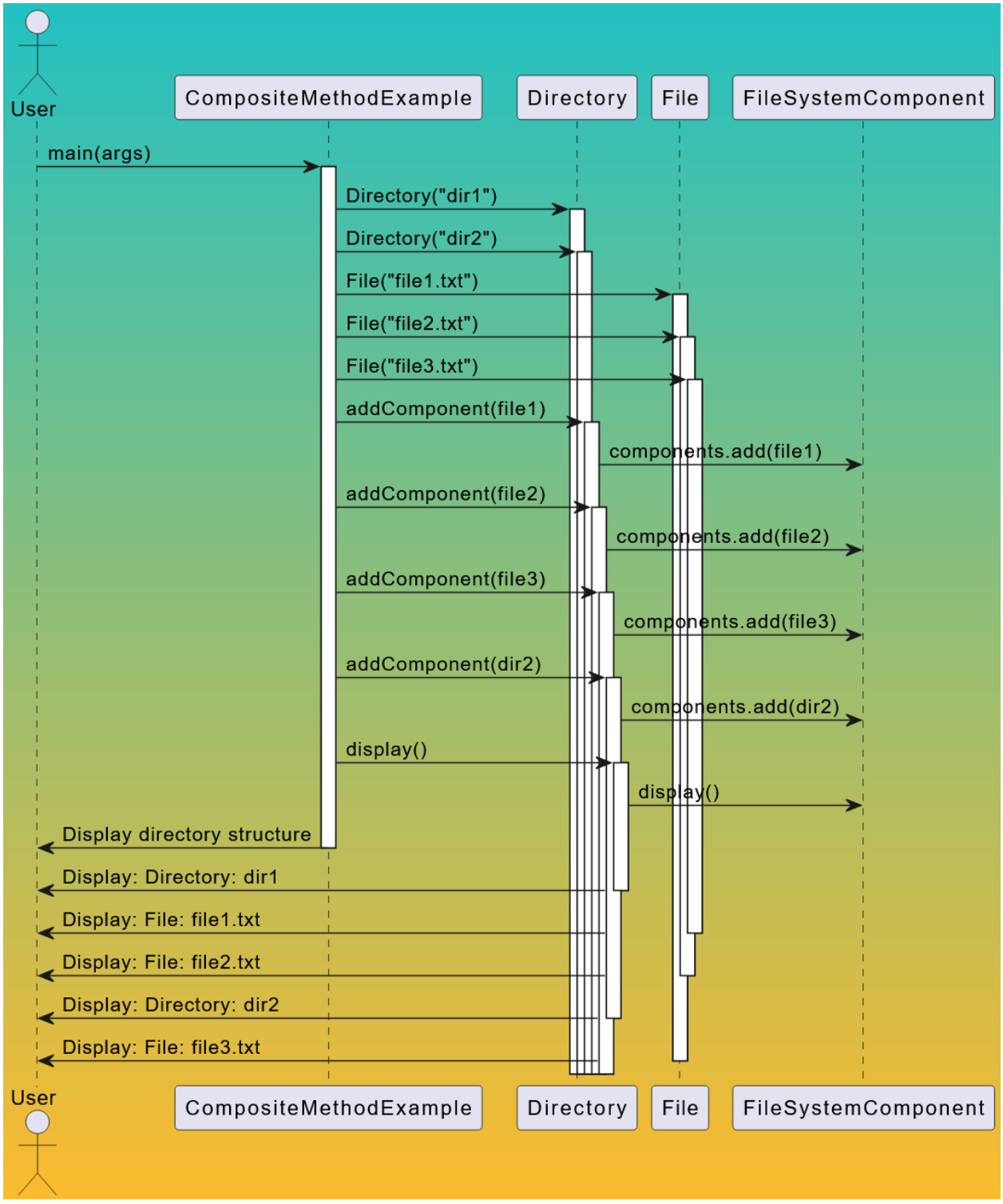
2.5. UML-код диаграммы классов

```
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
```

2.6. Диаграмма последовательности выполнения

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



2.7. 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
```

2.8. Исходный код реализации

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

```
58.         dir1.display();
59.     }
60. }
```

2.9. Результат работы

```
PS F:\Github\DesignPatternsConsoleExamples>  & 'C:\Program Files\Java\jdk-20\bin\java.exe' '--enable-
preview' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\alexey\AppData\Roaming\Code\User\w
orkspaceStorage\886deb62e2141619cb4711b2b310fcea\redhat.java\jdt_ws\DesignPatternsConsoleExamples_a82
a8d0e\bin' 'structural.CompositeMethodExample'
Directory: dir1
File: file1.txt
File: file2.txt
Directory: dir2
File: file3.txt
```

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.

3. ПОВЕДЕНЧЕСКИЙ ШАБЛОН СТРАТЕГИЯ

3.1. 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.

3.2. Случаи использования

Используется в случаях когда:

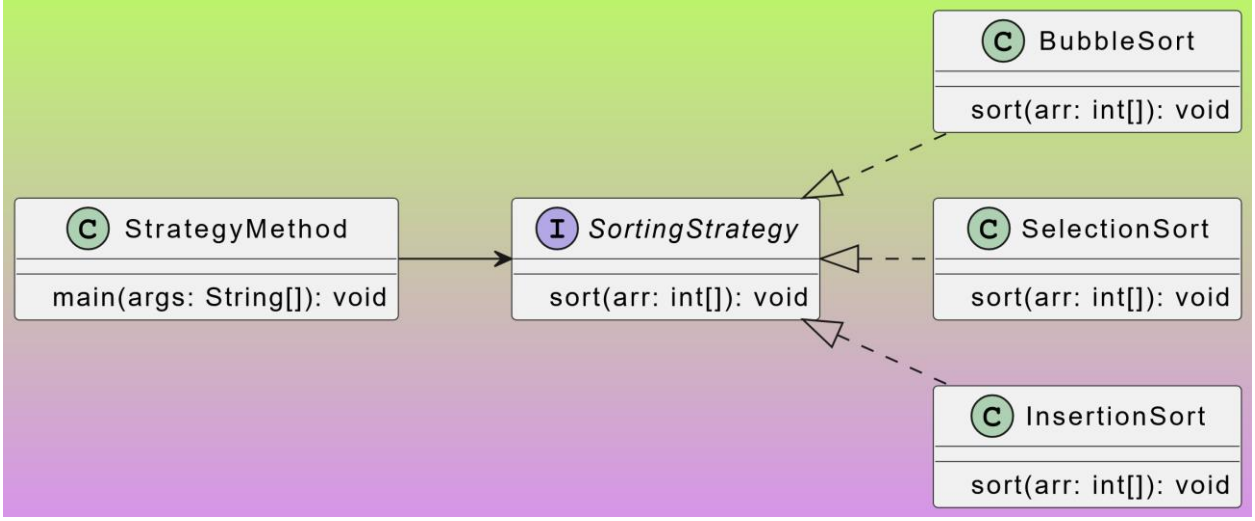
- У вас есть несколько алгоритмов, которые могут взаимозаменяться для выполнения конкретной задачи.
- Вы хотите отделить алгоритмы от клиентского кода, что упрощает добавление, удаление или изменение алгоритмов без влияния на общую структуру.
- Вам нужно динамически выбирать алгоритм во время выполнения,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

3.3. 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.

3.4. Диаграмма классов

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

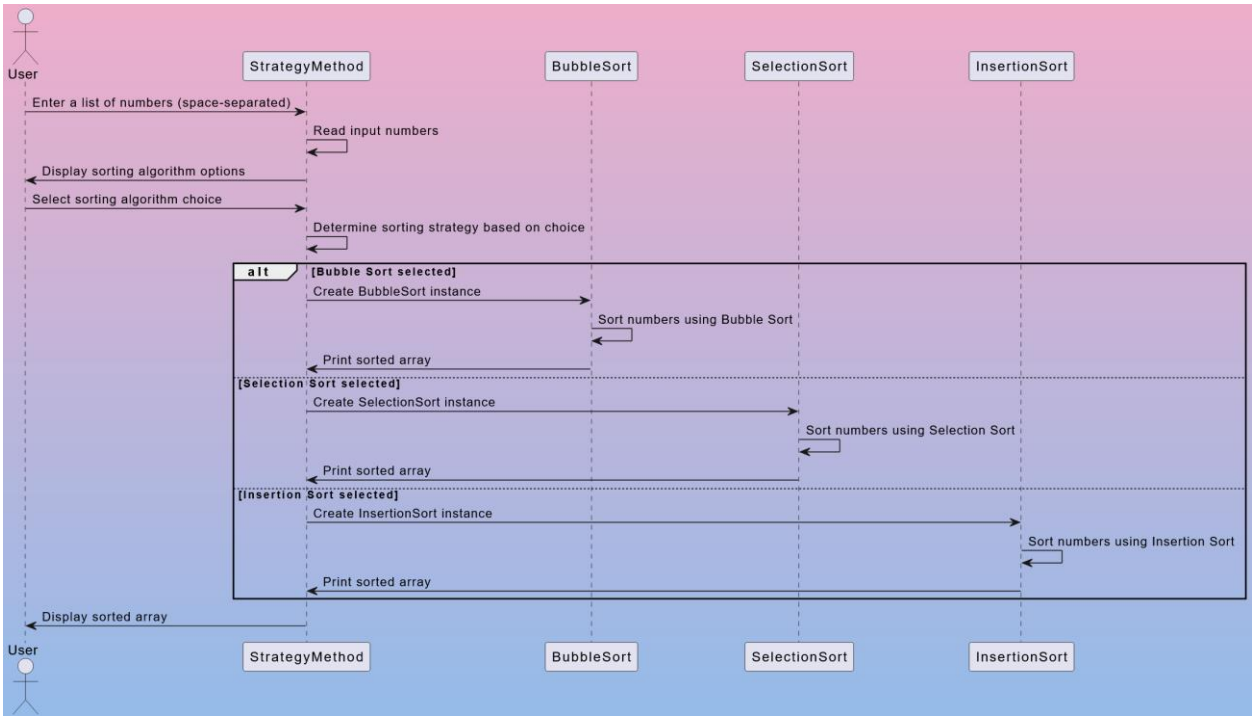


3.5. UML-код диаграммы классов

```
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
```

3.6. Диаграмма последовательности взаимодействия

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



3.7. 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
```

3.8. Исходный код реализации

```java
package behavioral;

import java.util.Arrays;
import java.util.Scanner;

interface SortingStrategy {
    void sort(int[] arr);
}

class BubbleSort implements SortingStrategy {
    public void sort(int[] arr) {
        int n = arr.length;
        for (int i = 0; i < n - 1; i++) {
            for (int j = 0; j < n - i - 1; j++) {
                if (arr[j] > arr[j + 1]) {
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                }
            }
        }
        System.out.println("Bubble Sort: " + Arrays.toString(arr));
    }
}

class SelectionSort implements SortingStrategy {
    public void sort(int[] arr) {
        int n = arr.length;
        for (int i = 0; i < n - 1; i++) {
            int minIndex = i;
            for (int j = i + 1; j < n; j++) {
                if (arr[j] < arr[minIndex]) {
                    minIndex = j;
                }
            }
            int temp = arr[minIndex];
            arr[minIndex] = arr[i];
            arr[i] = temp;
        }
        System.out.println("Selection Sort: " + Arrays.toString(arr));
    }
}
```

```
43.
44. class InsertionSort implements SortingStrategy {
45.     public void sort(int[] arr) {
46.         int n = arr.length;
47.         for (int i = 1; i < n; ++i) {
48.             int key = arr[i];
49.             int j = i - 1;
50.             while (j >= 0 && arr[j] > key) {
51.                 arr[j + 1] = arr[j];
52.                 j = j - 1;
53.             }
54.             arr[j + 1] = key;
55.         }
56.         System.out.println("Insertion Sort: " + Arrays.toString(arr));
57.     }
58. }
59.
60. public class StrategyMethodExample {
61.     public static void main(String[] args) {
62.         Scanner scanner = new Scanner(System.in);
63.         System.out.print("Enter a list of numbers (space-separated): ");
64.         String input = scanner.nextLine();
65.         int[] numbers = Arrays.stream(input.split("
    ")).mapToInt(Integer::parseInt).toArray();
66.
67.         SortingStrategy sortingStrategy;
68.         System.out.println("Select sorting algorithm:");
69.         System.out.println("1. Bubble Sort");
70.         System.out.println("2. Selection Sort");
71.         System.out.println("3. Insertion Sort");
72.         int choice = scanner.nextInt();
73.         switch (choice) {
74.             case 1:
75.                 sortingStrategy = new BubbleSort();
76.                 break;
77.             case 2:
78.                 sortingStrategy = new SelectionSort();
79.                 break;
80.             case 3:
81.                 sortingStrategy = new InsertionSort();
82.                 break;
83.             default:
84.                 System.out.println("Invalid choice. Using Bubble Sort.");
85.                 sortingStrategy = new BubbleSort();
86.         }
87.         sortingStrategy.sort(numbers);
88.     }
89. }
```

3.9. 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:

```
PS F:\Github\DesignPatternsConsoleExamples>  & 'C:\Program Files\Java\jdk-20\bin\java.exe' '--enable-preview' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\alexey\AppData\Roaming\Code\User\workspaceStorage\886deb62e2141619cb4711b2b310fcea\redhat.java\jdt_ws\DesignPatternsConsoleExamples_a82a8d0e\bin' 'behavioral.StrategyMethodExample'
Enter a list of numbers (space-separated): 7 3 9 0 2 8 7 6 3 4 1 2
Select sorting algorithm:
1. Bubble Sort
2. Selection Sort
3. Insertion Sort
1
Bubble Sort: [0, 1, 2, 2, 3, 3, 4, 6, 7, 7, 8, 9]
```

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