Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

РЕФЕРАТ

По дисциплине «Основы алгоритмизации и программирования»

На тему «Системы состояний»

Выполнила:

Студентка 1 курса 7 группы

Яскевич Валерия Александровна

Проверил:

Белодед Николай Иванович

2023, Минск

**Содержание**

[**Введение** 3](#_Toc163038849)

[**Основные понятия** 3](#_Toc163038850)

[**Проектирование систем состояний в C++** 4](#_Toc163038851)

[**Паттерны проектирования для систем состояний** 6](#_Toc163038852)

[**Паттерн «Состояние»** 6](#_Toc163038853)

[**Паттерн «Конечный автомат»** 7](#_Toc163038854)

[**Моделирование поведения объектов с помощью систем состояний** 9](#_Toc163038855)

[**Сравнение с другими подходами** 10](#_Toc163038856)

[**Системы состояний с использованием условных операторов:** 10](#_Toc163038857)

[**Системы состояний с использованием флагов состояний:** 10](#_Toc163038858)

[**Рекомендации:** 11](#_Toc163038859)

[**Преимущества и ограничения систем состояний** 11](#_Toc163038860)

[**Преимущества в разработке, тестировании и поддержке:** 11](#_Toc163038861)

[**Ограничения и сценарии, когда использование систем состояний нецелесообразно:** 12](#_Toc163038862)

[**Заключение** 13](#_Toc163038863)

[**Значимость и перспективы использования систем состояний:** 13](#_Toc163038864)

[**Рекомендации по дальнейшему изучению и применению концепции систем состояний:** 13](#_Toc163038865)

[**Список литературы** 14](#_Toc163038866)

# **Введение**

Концепция систем состояний является важным инструментом в разработке программного обеспечения, который позволяет моделировать и управлять поведением системы на основе её текущего состояния. Системы состояний состоят из конечного набора состояний, переходов между ними и действий, которые выполняются в каждом состоянии или во время перехода.

Они позволяют эффективно управлять сложным поведением системы, разбивая его на отдельные состояния и определяя переходы между ними. Это делает код более понятным и легко поддерживаемым.

Системы состояний нашли свое применение в множестве областей:

1. Используются для моделирования конечных автоматов, где каждое состояние представляет определенное поведение, а переходы определяют условия для изменения состояний.
2. В разработке пользовательских интерфейсов системы состояний часто применяются для управления различными состояниями интерфейса, например, вводом данных, обработкой действий пользователя и отображением различных экранов.
3. В различных системах, таких как управление ресурсами или процессами, системы состояний используются для моделирования жизненного цикла ресурсов или процессов и определения допустимых действий в каждом состоянии.
4. В бизнес-приложениях они применяются для моделирования бизнес-процессов и управления их выполнением в зависимости от текущих условий и событий.

В целом, системы состояний предоставляют удобный и эффективный способ управления поведением системы в зависимости от её состояния и внешних условий, что делает их незаменимым инструментом при проектировании и разработке различных программных систем.

# **Основные понятия**

Состояние представляет определенное положение системы в конкретный момент времени. Это может быть любое определенное условие, при котором система находится в определенном состоянии своего жизненного цикла. У состояния можно выделить роль и функции:

1. Моделирование поведения. Каждое состояние моделирует определенное поведение системы (например, состояние «заблокировано» может означать, что система не принимает ввод пользователя, а состояние «активно» может означать, что система готова к взаимодействию).
2. Управления действиями. Состояния определяют, какие действия или операции выполняются в системе в определенном состоянии (в состоянии «ожидание» система может ожидать определенного события или ввода пользователя).
3. Определение переходов. Каждое состояние определяет допустимые переходы, то есть условия, при которых система может перейти из текущего состояния в другое (из состояния «ожидание» система может перейти в состояние «активно» после получения определенного события).
4. Реакция на события. Состояния определяют, как система реагирует на внешние события или входные данные (Система может отклонить ввод пользователя в состоянии «заблокировано», но принять его в состоянии «активно»)

# **Проектирование систем состояний в C++**

При реализации систем состояний в C++ с использованием объектно-ориентированного подхода обычно создается набор классов, представляющих состояния и контекст системы, которые управляют переходами между состояниями. Давайте рассмотрим пример простой системы состояний для светофора.

#include <iostream>

#include <thread>

#include <chrono>

// Базовый класс состояния

class State {

public:

virtual void handle() = 0;

};

// Конкретные классы состояний

class RedState : public State {

public:

void handle() override {

std::cout << "Red light\n";

std::this\_thread::sleep\_for(std::chrono::seconds(2));

}

};

class GreenState : public State {

public:

void handle() override {

std::cout << "Green light\n";

std::this\_thread::sleep\_for(std::chrono::seconds(2));

}

};

class YellowState : public State {

public:

void handle() override {

std::cout << "Yellow light\n";

std::this\_thread::sleep\_for(std::chrono::seconds(1));

}

};

// Класс контекста светофора

class TrafficLight {

private:

State\* state; // Текущее состояние

public:

TrafficLight(State\* initialState) : state(initialState) {}

// Установка нового состояния

void setState(State\* newState) {

state = newState;

}

// Симуляция работы светофора

void run() {

while (true) {

state->handle(); // Обработка текущего состояния

changeState(); // Переход к следующему состоянию

}

}

// Смена состояния светофора

void changeState() {

if (state == nullptr)

return;

if (dynamic\_cast<RedState\*>(state)) {

setState(new GreenState());

}

else if (dynamic\_cast<GreenState\*>(state)) {

setState(new YellowState());

}

else if (dynamic\_cast<YellowState\*>(state)) {

setState(new RedState());

}

}

};

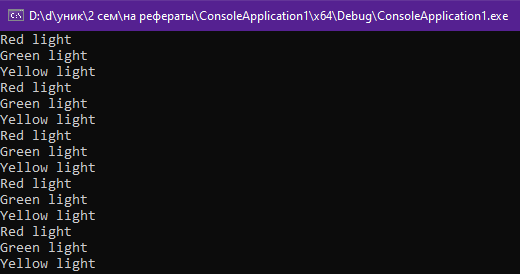
int main() {

TrafficLight trafficLight(new RedState());

trafficLight.run(); // Запуск симуляции работы светофора

return 0;

}



Этот пример демонстрирует реализацию простой системы состояний для светофора. Классы `RedState`, `GreenState` и `YellowState` представляют состояния: красный, зеленый и желтый сигналы. Класс `TrafficLight` представляет контекст светофора, управляющий состояниями и их переходами. Метод `run()` симулирует работу светофора, вызывая метод `handle()` текущего состояния и затем переходя к следующему состоянию.

# **Паттерны проектирования для систем состояний**

## **Паттерн «Состояние»**

Позволяет объекту изменять свое поведение в зависимости от своего состояния. Объект делегирует выполнение определенных действий классам, представляющим его состояния.

Ключевые компоненты:

* Context: Контекст, который содержит объект состояния и использует его для выполнения действий.
* State: Интерфейс или абстрактный класс, представляющий состояние объекта.
* ConcreteState: Конкретные реализации состояний.

Вот пример реализации этого паттерна:

#include <iostream>

class Context;

// Абстрактный класс состояния

class State {

public:

virtual void handle(Context& context) = 0;

};

// Конкретное состояние 1

class ConcreteState1 : public State {

public:

void handle(Context& context) override {

std::cout << "Concrete State 1\n";

}

};

// Конкретное состояние 2

class ConcreteState2 : public State {

public:

void handle(Context& context) override {

std::cout << "Concrete State 2\n";

}

};

// Контекст

class Context {

private:

State\* state;

public:

Context(State\* initialState) : state(initialState) {}

void setState(State\* newState) {

state = newState;

}

void request() {

state->handle(\*this);

}

};

int main() {

ConcreteState1 state1;

ConcreteState2 state2;

Context context(&state1);

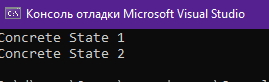
context.request(); // Output: Concrete State 1

context.setState(&state2);

context.request(); // Output: Concrete State 2

return 0;

}



## **Паттерн «Конечный автомат»**

Позволяет моделировать поведение системы с помощью конечного набора состояний, переходов между ними и действий, выполняемых в каждом состоянии или при переходе.

Ключевые компоненты:

* Состояния: Представление состояний системы.
* Переходы: Условия переходов между состояниями.
* Действия: Действия, выполняемые при входе в состояние или при переходе.

Пример реализации паттерна:

#include <iostream>

enum class State {

STATE\_A,

STATE\_B,

STATE\_C

};

class Context {

private:

State currentState;

public:

Context(State initialState) : currentState(initialState) {}

void setState(State newState) {

currentState = newState;

}

void handleRequest() {

switch (currentState) {

case State::STATE\_A:

std::cout << "Handling request in state A\n";

// Дополнительные действия для состояния A

break;

case State::STATE\_B:

std::cout << "Handling request in state B\n";

// Дополнительные действия для состояния B

break;

case State::STATE\_C:

std::cout << "Handling request in state C\n";

// Дополнительные действия для состояния C

break;

}

}

};

int main() {

Context context(State::STATE\_A);

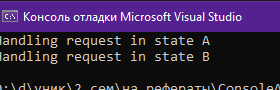
context.handleRequest(); // Output: Handling request in state A

context.setState(State::STATE\_B);

context.handleRequest(); // Output: Handling request in state B

return 0;

}



Оба этих паттерна позволяют эффективно управлять состояниями объекта или системы и упрощают разработку и поддержку кода благодаря разделению логики на отдельные состояния и переходы между ними.

# **Моделирование поведения объектов с помощью систем состояний**

Моделирование поведения объектов с помощью систем состояний позволяет эффективно управлять состояниями и переходами между ними в приложениях. Вот несколько примеров использования и реализации управления состояниями в различных типах приложений:

1. Управление пользовательским интерфейсом:

- Веб-приложения: Моделирование состояний страницы, формы или элементов управления, таких как кнопки или поля ввода.

- Мобильные приложения: Управление различными экранами и состояниями приложения, например, экранами входа, регистрации, профиля и т. д.

2. Управление бизнес-процессами:

- CRM или ERP системы: Моделирование жизненного цикла заказов, клиентов, сделок и т. д.

- Банковские системы: Управление состояниями банковских транзакций, счетов, заявок на кредит и т. д.

3. Управление игровым поведением:

- В компьютерных играх: Моделирование состояний игровых персонажей, объектов, игровых уровней и т. д.

- Виртуальные миры: Управление поведением NPC (неперсонажных персонажей), ресурсами, мирами и т. д.

4. Управление устройствами и оборудованием:

- Промышленные автоматизированные системы: Моделирование состояний и управление процессами производства, манипуляторами, роботами и т. д.

- Умные устройства: Управление состояниями и поведением умных устройств, таких как умные дома, умные автомобили и т. д.

Реализация управления состояниями в приложениях обычно осуществляется путем создания классов для каждого состояния и контекста, который управляет переходами между состояниями и исполняет действия, связанные с текущим состоянием. Здесь ключевыми элементами являются абстрактные классы или интерфейсы для состояний и контекста, а также конкретные реализации этих классов для каждого состояния и их переходов.

Важно также учитывать проектирование архитектуры приложения для обеспечения гибкости и расширяемости системы состояний, так что при необходимости вы могли легко добавлять новые состояния или изменять поведение существующих.

# **Сравнение с другими подходами**

## **Системы состояний с использованием условных операторов:**

При использовании этого подхода каждое состояние объекта обычно моделируется отдельным классом или функцией, которые содержат логику обработки действий и переходов между состояниями. Управление состояниями происходит путем явной проверки текущего состояния объекта и выполнения соответствующих действий с использованием условных операторов (if, switch, etc.).

Преимущества:

1. Простота: Код может быть простым и прямолинейным, что упрощает его понимание и поддержку.

2. Явность: Логика переходов и действий явно выражается в коде, что делает его более понятным для разработчиков.

Недостатки:

1. Ограниченность: При добавлении новых состояний или изменении логики переходов код может стать сложным и сложным для поддержки.

2. Дублирование кода: Логика переходов может дублироваться в различных местах кода, что приводит к возможным ошибкам и усложнению изменений.

## **Системы состояний с использованием флагов состояний:**

При использовании этого подхода объект содержит набор флагов, каждый из которых представляет одно из состояний объекта. Управление состояниями осуществляется путем изменения и проверки этих флагов.

Преимущества:

1. Гибкость и расширяемость: Легко добавлять новые состояния и изменять поведение объекта без изменения логики переходов.

2. Управление сложным поведением: Можно моделировать сложное поведение с множеством состояний и динамическими условиями переходов.

Недостатки:

1. Потенциальная сложность и потеря читаемости: В зависимости от количества и сложности состояний флаги состояний могут привести к усложнению кода и потере его читаемости и понятности.

2. Управление состояниями в одном месте: В некоторых случаях логика управления состояниями может размазаться по всему коду, что делает его сложнее понимать и изменять.

## **Рекомендации:**

- Если приложение имеет ограниченное количество простых состояний и небольшой объем логики переходов, использование условных операторов может быть достаточным и более прямолинейным.

- Если приложение имеет сложное поведение с множеством состояний и/или динамическими условиями переходов, использование систем состояний с флагами состояний может быть более предпочтительным для обеспечения гибкости и расширяемости.

- При выборе подхода следует также учитывать требования к производительности, обработку ошибок, а также особенности архитектуры вашего приложения.

# **Преимущества и ограничения систем состояний**

Использование систем состояний в разработке программного обеспечения может принести ряд преимуществ, но также существуют ограничения и сценарии, когда использование систем состояний может быть нецелесообразным. Давайте рассмотрим это более подробно:

## **Преимущества в разработке, тестировании и поддержке:**

1. Четкость и модульность кода: Системы состояний позволяют четко определить различные состояния и переходы между ними, что способствует модульности и улучшает читаемость кода.

2. Гибкость и расширяемость: Добавление новых состояний или изменение логики переходов обычно требует минимальных изменений в коде благодаря отделению логики состояний от основной логики приложения.

3. Упрощение тестирования: Тестирование отдельных состояний и переходов может быть проще и более надежно, что уменьшает количество ошибок и облегчает отладку.

4. Улучшение масштабируемости и обслуживаемости: Использование систем состояний способствует созданию гибких и легко расширяемых систем, что делает их более легкими в поддержке и сопровождении в долгосрочной перспективе.

## **Ограничения и сценарии, когда использование систем состояний нецелесообразно:**

1. Простота и незначительность поведения: В случае, если поведение объекта является простым и не включает множество состояний и переходов, использование систем состояний может быть излишним усложнением.

2. Низкий объем логики переходов: Если логика переходов между состояниями ограничена и не требует сложных проверок условий, применение систем состояний может быть излишним.

3. Малый объем проекта: В небольших проектах применение систем состояний может быть излишним, особенно если поведение объекта не предполагает дальнейшего расширения и изменения.

4. Сложность понимания системы состояний: Если система состояний является сложной и труднопонимаемой для разработчиков, это может увеличить сложность поддержки и отладки приложения.

5. Высокие накладные расходы: Реализация системы состояний может быть излишне сложной и занимать больше времени и ресурсов, чем альтернативные подходы, особенно в случае небольших и простых проектов.

В целом, применение систем состояний целесообразно в случае, когда они помогают упростить разработку, улучшить модульность и гибкость приложения, а также улучшить его обслуживаемость в долгосрочной перспективе. Однако важно оценить конкретные потребности и особенности проекта, чтобы определить, насколько целесообразно использование систем состояний в данном случае.

# **Заключение**

Использование систем состояний в разработке программного обеспечения представляет собой значимый инструмент, который может значительно улучшить модульность, гибкость и обслуживаемость приложения. Ниже приведены основные выводы о значимости и перспективах использования систем состояний, а также рекомендации по дальнейшему изучению и применению данной концепции:

## **Значимость и перспективы использования систем состояний:**

1. Улучшение модульности: Системы состояний позволяют четко разделить логику состояний и переходов между ними, что способствует созданию более модульного и легко расширяемого кода.

2. Повышение гибкости: Применение систем состояний делает приложение более гибким и способствует его адаптации к изменяющимся требованиям и условиям.

3. Улучшение обслуживаемости: Системы состояний делают код более читаемым и понятным, что упрощает его поддержку и обслуживание в долгосрочной перспективе.

4. Улучшение тестируемости: Каждое состояние и переход может быть протестировано отдельно, что упрощает тестирование и обнаружение ошибок.

## **Рекомендации по дальнейшему изучению и применению концепции систем состояний:**

1. Глубже изучите паттерн "Состояние": Ознакомьтесь с основами и примерами применения паттерна "Состояние", чтобы лучше понять его концепцию и принципы работы.

2. Исследуйте альтернативные подходы: Рассмотрите различные способы реализации систем состояний, такие как использование конечных автоматов или флагов состояний, и определите, какой подход лучше подходит для вашего проекта.

3. Практикуйтесь на реальных проектах: Примените концепцию систем состояний на практике, реализуя её в своих проектах или участвуя в открытых исследовательских проектах.

4. Изучите сопутствующие темы: Углубитесь в сопутствующие темы, такие как динамическое программирование, управление состояниями в пользовательском интерфейсе, управление жизненным циклом объектов и т. д., чтобы лучше понять контекст и применение систем состояний.

5. Обменивайтесь опытом с сообществом: Обсуждайте свои идеи и опыт с другими разработчиками в онлайн-форумах, митапах и конференциях, чтобы узнать о лучших практиках и получить обратную связь о своих проектах.

# **Список литературы**

1. <https://learn.microsoft.com/ru-ru/windows/win32/taskschd/displaying-task-names-and-state--c--->
2. <https://radioprog.ru/post/1503>