GeekBrains

Моделирование жизненного цикла объекта и введение в UML

Диаграммы состояний

Оглавление

Введение

Концепция моделирования — важный инструмент в различных областях, в том числе в ИТ. И как в любой системе у этого подхода есть свой жизненный цикл, который включает разные этапы: от идеи до реализации и эксплуатации.

На лекции мы рассмотрим принципы моделирования жизненного цикла объекта и поговорим про UML (Unified Modeling Language) — стандартный язык моделирования, который широко используется для проектирования систем и разработки программного обеспечения. Мы рассмотрим примеры использования UML на проектах и обсудим его преимущества.

После лекции у вас будет более четкое представление о том, как использовать моделирование жизненного цикла объекта и UML для создания надежных эффективных систем и программного обеспечения.

На этой лекции вы:

- Узнаете, что такое UML и зачем он нужен.
- Познакомитесь с инструментом PlantUML.
- Поработаете с диаграммой состояний.
- На практике примените изученные инструменты.

Термины, используемые в лекции

UML (Unified Modeling Language) — язык графического описания для объектного моделирования в области разработки ПО, для моделирования бизнес-процессов, системного проектирования и отображения организационных структур.

UML

Сегодня мы остановимся на объектном моделировании. Но сперва разберемся, зачем нам это нужно.

Представьте, что собираетесь строить дом. Что сделаете сначала? Скорее всего, представите образ — он поможет определиться с тем, какие характеристики вам нужны. Каким будет дом — большим для семьи или компактным для пары? Он будет

далеко от дороги или нет? Какая нужна звукоизоляция? Отвечая на вопросы, вы соберете **требования** к дому.

Следующий шаг — **смоделировать** дом, а на основе удачной модели сделать **прототип** хотя бы его части, чтобы понять, что ваши теоретические представления соответствовали реальности. Или хотя бы протестировать кровати и кресла, которые вы собрались купить.

Также происходит и с разработкой ПО. После сбора требований мы должны смоделировать систему, чтобы понять, какими будут временные и денежные затраты на разработку. Затем нужно сделать прототип хотя бы в части интерфейсов. В этом нам поможет язык UML.

UML содержит 14 диаграмм. Они делятся на две большие категории:

- **Структурные диаграммы** — статичные. Служат для отображения структуры системы. Например, для описания ее классов.
- **Диаграммы поведения** — динамичные. Показывают, как система действует в тех или иных случаях. Например, демонстрирует обмен данными или изменение статусов сущностей системы.

Диаграмма состояний — одна из представительниц диаграмм поведения. Именно ее мы подробно рассмотрим на этом занятии.

PlantUML

PlantUML — это свободный и открытый язык, а также инструмент для создания UML-диаграмм. Он позволяет создавать графические представления различных типов моделей, включая диаграммы классов, диаграммы последовательностей, диаграммы состояний и многие другие. У него простой и интуитивно понятный синтаксис: можно описывать модели в виде текста, а затем генерировать диаграммы на основе этого описания.

PlantUML поддерживает множество форматов вывода (PNG, SVG, PDF и ASCII Art), что позволяет встраивать диаграммы в разные типы документов и отчетов. Интегрируется с инструментами разработки: Visual Studio Code, IntelliJ IDEA, Eclipse и другими.

С помощью PlantUML специалисты могут создавать диаграммы для проектирования, документирования и отображения своих систем, приложений, процессов.

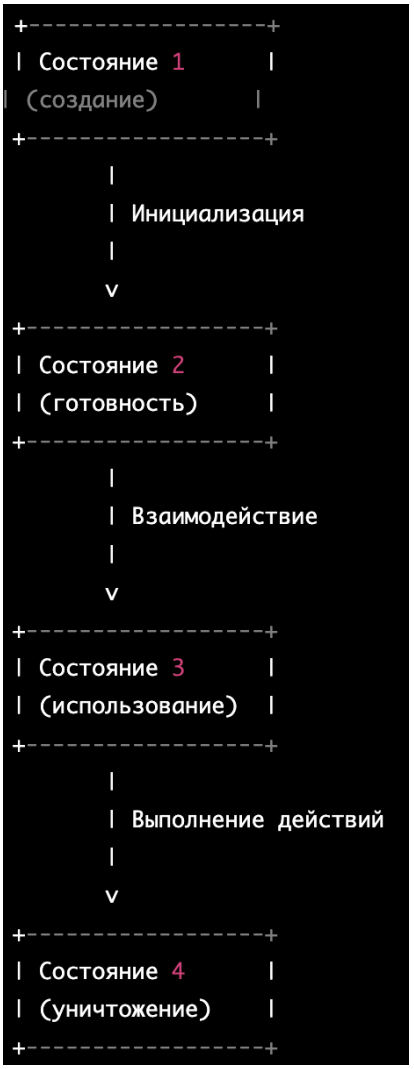
Инструмент будет полезен для разработчиков, аналитиков, архитекторов и других специалистов, работающих в ИТ.

Вы можете вставить код из примеров ниже в онлайн-редактор PlantUML, чтобы отрисовать схемы.

Моделирование жизненного цикла объекта

Диаграммы состояний можно использовать для моделирования жизненного цикла объекта.

Разберем на примере:



Каждое состояние — это этап жизненного цикла объекта: от создания до уничтожения. Переходы между состояниями — это события или действия, которые могут привести к изменению состояния. Например, объект может перейти из

состояния «Создание» в состояние «Готовность» при завершении инициализации, а при взаимодействии с другими объектами — в состояние «Использование». Кроме того, на схеме показано, что объект может перейти в состояние «Уничтожение» при определенных условиях: например, когда он больше не нужен или когда программа завершается.

Можно представить еще один формат реализации жизненного цикла на примере ЖЦ объекта «Автомобиль»:

Код для отрисовки в PlantUML:

```
@startuml
[*] --> Создание

Создание --> Готовность : Инициализация

Готовность --> Использование : Нажатие педали газа
Готовность --> Уничтожение : Нажатие кнопки "Выключить"

Использование --> Готовность : Отпускание педали газа
Использование --> Уничтожение : Поломка двигателя

Уничтожение --> [*]
@enduml
```

Обозначения:

- **[*]** — начальное и конечное состояния,
- **-->** — переход между состояниями,
- **слова после :** — события, которые могут привести к переходу.

Объект «Автомобиль» начинается с состояния «Создание», затем переходит в состояние «Готовность» после инициализации.

Из состояния «Готовность» автомобиль может перейти в состояние «Использование», если нажать педаль газа, или в состояние «Уничтожение», если нажать кнопку «Выключить».

Из состояния «Использование» автомобиль может вернуться в состояние «Готовность», если отпустить педаль газа, или перейти в состояние «Уничтожение» при поломке двигателя.

Из состояния «Уничтожение» автомобиль возвращается в начальное состояние [*]. Это значит, что он больше не существует.

Таким образом, конечные автоматы могут помочь специалистам лучше понять и описать жизненный цикл объекта в программе.

PlantUML — удобный инструмент создания и отображения конечных автоматов в формате диаграммы. Он упрощает проектирование и разработку программных приложений.

Иерархии состояний

Иерархия состояний — это методология использования диаграмм состояний для моделирования более сложных систем и объектов. Представляет собой набор вложенных состояний, которые могут содержать собственные состояния и переходы. Это делает иерархию более гибкой и расширяемой.

Использование иерархий может помочь специалистам создавать более сложные диаграммы состояний для более точного описания поведения объектов в системе и управления переходами между состояниями.

Разберем несколько примеров.

Управление процессом продажи — иерархия состояний поможет описать процесс от создания заказа до его выполнения.

- Уровень 1 — состояние «Ожидание заказа» с дочерними состояниями «Ожидание подтверждения заказа» и «Обработка заказа».
- Уровень 2 — состояние «Обработка заказа» с дочерними состояниями «Обработка заказа в системе» и «Обработка оплаты».

Управление ресурсами компьютера — иерархия состояний пригодится, чтобы описать работу компьютера в разных режимах.

- Уровень 1 — состояние «Работа компьютера» с дочерними состояниями «Активный режим» и «Режим ожидания».
- Уровень 2 — состояние «Активный режим» с дочерними состояниями «Работа с приложениями» и «Выполнение задач в фоновом режиме».

Управление игровым персонажем — иерархию состояний можно использовать для описания поведения игрового персонажа.

- Уровень 1 — состояние «Игровой персонаж» с дочерними состояниями «Движение» и «Бой».
- Уровень 2 — состояние «Движение» с дочерними состояниями «Ходьба» и «Бег».
- Уровень 3 — состояние «Бой» с дочерними состояниями «Атака» и «Защита».

Управление автомобилем — иерархию состояний можно использовать для описания работы автомобиля.

- Уровень 1 — состояние «Автомобиль» с дочерними состояниями «Движение» и «Остановка».
- Уровень 2 — состояние «Движение» с дочерними состояниями «Едет прямо», «Поворот налево» и «Поворот направо».

> 💡 Для каждого из примеров выше иерархию состояний можно продолжить на следующих уровнях в зависимости от сложности процесса.

Использование иерархии состояний может существенно упростить процесс разработки программного обеспечения, улучшить качество кода, сделать приложение более гибким и расширяемым.

Иерархию состояний можно применить к различным аспектам программирования, включая моделирование процессов, управление ресурсами, управление состоянием объектов и управление пользовательским интерфейсом.

Пример иерархии состояний в формате PlantUML

Рассмотрим пример иерархии состояний, моделирующей ЖЦ заказа в интернет-магазине, в формате PlantUML.

```
@startuml
[*] --> Создание : Начало заказа

Создание --> Ожидание_подтверждения : Завершение формы заказа

Ожидание_подтверждения --> Подтвержден : Подтверждение заказа
Ожидание_подтверждения --> Отменен : Отмена заказа

Подтвержден --> Готов_к_отправке : Оплата
Подтвержден --> Отменен : Отмена заказа

Готов_к_отправке --> Доставляется : Отправка заказа

Доставляется --> Доставлен : Доставка заказа
Доставляется --> Отменен : Отмена заказа

Отменен --> [*] : Отмена заказа
Доставлен --> [*] : Заказ доставлен
@enduml
```

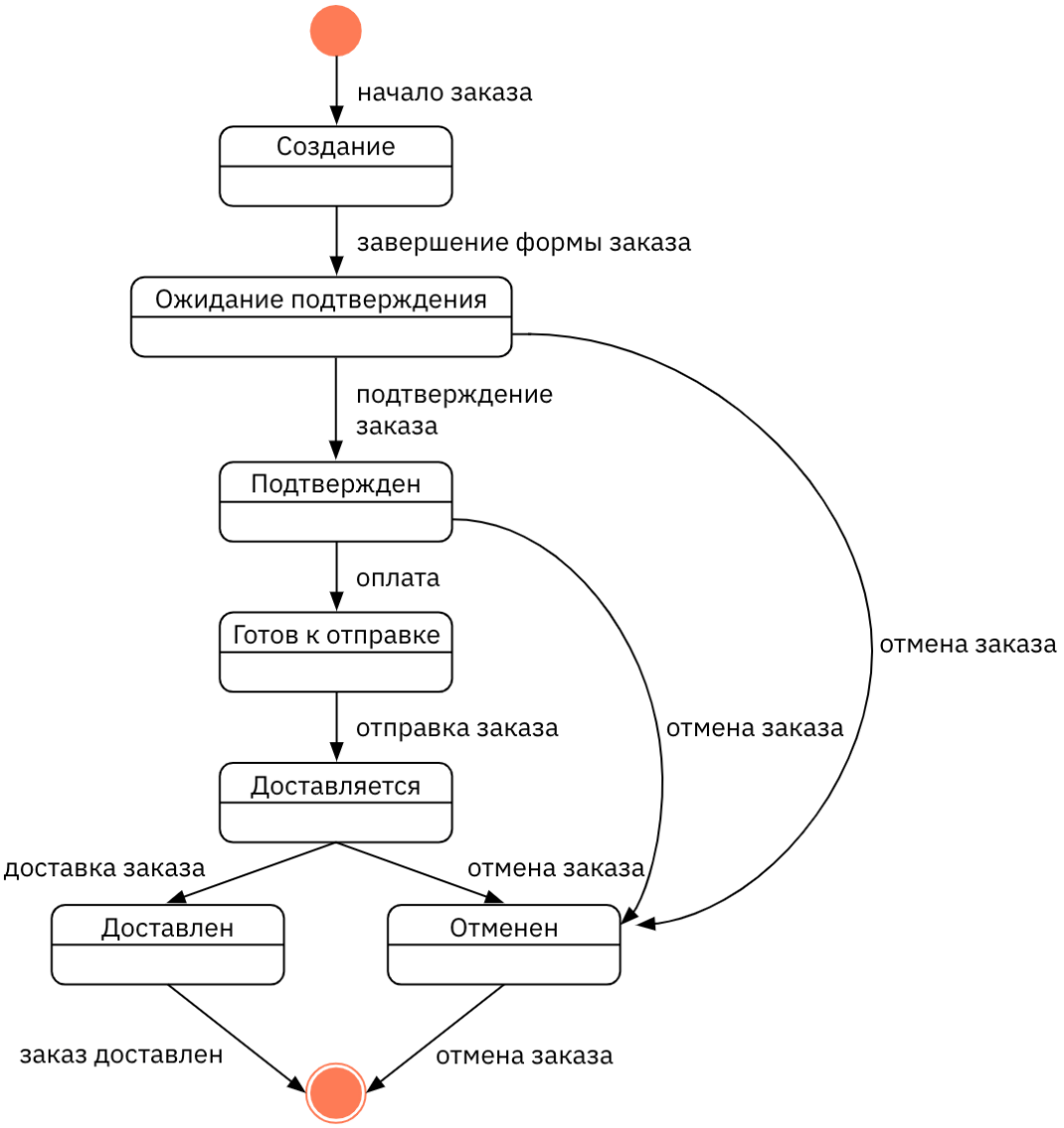
На уровне 1 заказ может находиться в состояниях «Создание», «Ожидание подтверждения», «Подтвержден», «Готов к отправке», «Доставляется», «Доставлен» и «Отменен».

На уровне 2 состояние «Ожидание подтверждения» содержит два дочерних состояния: «Подтвержден» и «Отменен».

На уровне 3 состояние «Подтвержден» содержит два дочерних состояния: «Готов к отправке» и «Отменен».

На уровне 4 состояние «Доставляется» содержит два дочерних состояния: «Доставлен» и «Отменен».

Посмотрим на результат:



Таким образом, использование иерархии состояний позволяет более детально описать жизненный цикл заказа в интернет-магазине и упростить его моделирование в коде.

Работа с исключениями

Диаграммы состояния позволяют описывать поведение объекта в различных состояниях и переходы между ними. Когда возникает исключительная ситуация, объект может перейти в новое состояние, которое позволит ему обработать ошибку и продолжить работу.

Исключение — это ответвление в коде, в котором мы обрабатываем кейсы, идущие не по плану. Диаграммы состояния можно использовать для моделирования этого процесса — они помогут разработчикам точнее определить, какие состояния должен принимать объект в процессе обработки ошибок.

Например, можно создать диаграмму состояния, которая описывает процесс обработки ошибок при работе с базой данных. На этой диаграмме можно показать, как объект переходит в состояние «Обработка ошибки», когда возникает исключительная ситуация, и как он возвращается в состояние «Нормальная работа», когда ошибка успешно обработана. Диаграмма состояния может также показать, какие методы должны быть вызваны в каждом состоянии и какие данные должны быть переданы в эти методы.

Таким образом, использование диаграмм состояния может помочь специалистам эффективнее обрабатывать исключительные ситуации, моделировать процессы обработки ошибок и создавать более надежные программные приложения.

Диаграмма состояний с исключениями подходит для описания поведения системы, которая может находиться в различных состояниях, но в определенных условиях может вызывать исключения. Ниже приведен пример диаграммы состояний с исключениями для простой системы, которая может находиться в состоянии «Выключено» или «Включено»:

- Если пользователь пытается включить систему, когда она выключена, то она переходит в состояние «Включено».
- Если пользователь пытается выключить систему, когда она включена, то она переходит в состояние «Выключено».

Однако в определенных условиях система может вызывать исключения:

- Если в системе нет ресурса для включения, она может вызвать исключение 1.
- Если в системе происходит ошибка при выключении, она может вызвать исключение 2.

Таким образом, диаграмма состояний с исключениями помогает лучше понять поведение системы и предусмотреть возможные исключения.

Диаграмма состояний (State machine UML)

Описание событий и действий — важная часть создания диаграммы состояний (State Machine Diagram) в UML. Эта информация помогает понять, что происходит в системе в каждом ее состоянии и при переходе между ними.

События могут приводить к переходу из одного состояния в другое. Могут быть внешними или внутренними (происходят в системе):

- внешние события: получение входного сигнала, нажатие кнопки или прибытие поезда на станцию.
- внутренние события: истечение времени, завершение вычисления или получение сообщения от другой части системы.

Действия выполняются в системе при переходе из одного состояния в другое. Это могут быть изменения значений переменных, вывод сообщения на экран, вызов метода или отправка сообщения в другую часть системы. Действия помогают определить, что происходит в системе при переходе из одного состояния в другое.

При создании диаграммы состояний нужно определить все возможные события, которые могут произойти в системе, и действия, которые будут выполняться при переходе между состояниями.

Например, при проектировании системы управления стиральной машины можно определить:

- события: загрузка белья, запуск машины, завершение стирки и так далее.
- действия: изменение состояния машины (переход из режима ожидания в режим стирки), включение двигателя, наполнение водой и так далее.

Описание событий и действий поможет определить, как система будет вести себя в каждом состоянии и при переходе между ними. При правильном описании можно создать надежную систему, выполняющую задачи эффективно и без ошибок.

Обработка исключений

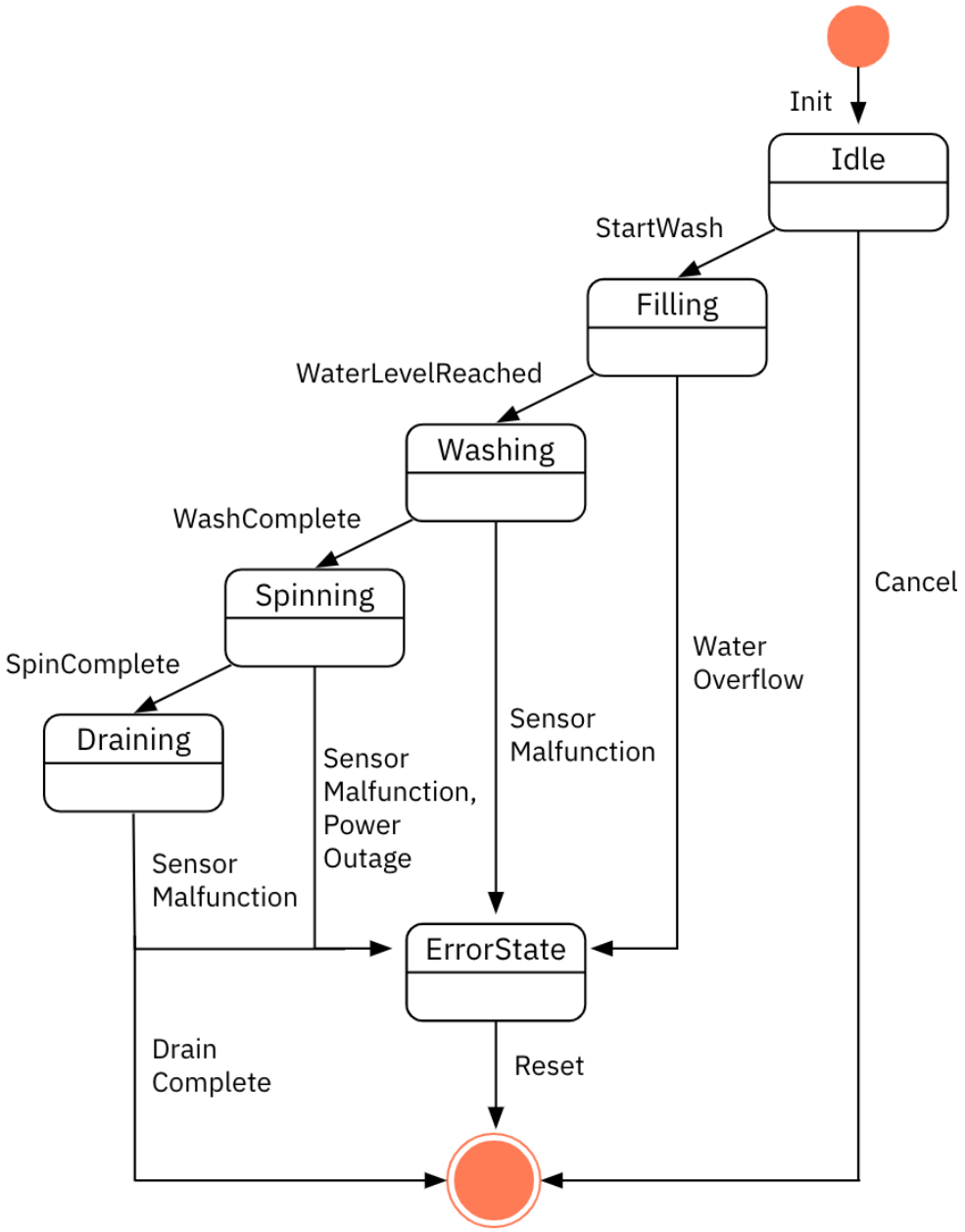
Когда система работает, возможны исключения и неожиданные события, которые могут привести к неправильной работе или даже к сбоям. Поэтому важно учесть возможные ошибки и неожиданные события для каждого состояния и описать, как они будут обработаны.

При проектировании системы управления стиральной машиной могут возникнуть:

- возможные ошибки: неисправность датчика уровня воды или сбой в электронной системе управления.
- неожиданные события: отключение электричества или повреждение сенсоров.

Нарисуем схему в PlantUml:

```
@startuml
scale 750 width

[*] --> Idle: Init
Idle --> Filling: StartWash
Filling --> Washing: WaterLevelReached
Washing --> Spinning: WashComplete
Spinning --> Draining: SpinComplete
Draining --> [*]: DrainComplete
Idle --> [*]: Cancel

Filling --> ErrorState: WaterOverflow
Washing --> ErrorState: SensorMalfunction
Spinning --> ErrorState: SensorMalfunction, PowerOutage
Draining --> ErrorState: SensorMalfunction

ErrorState --> [*]: Reset

@enduml
```



Состояния диаграммы:

- Idle — начальное состояние автомата, ожидание начала стирки.
- Filling — машина заполняется водой до заданного уровня.
- Washing — машина стирает.
- Spinning — машина отжимает белье.
- Draining — машина сливает воду.
- ErrorState — произошла ошибка или неожиданное событие.

Каждый переход обозначен стрелкой с названием события, которое к нему приводит. Например, для начала стирки машины пользователь должен нажать кнопку StartWash, что приведет к переходу в состояние Filling. Также в каждом состоянии обозначены действия, которые происходят в этом состоянии.

Для каждого состояния, где возможны ошибки, мы добавили переходы в состояние ErrorState, где происходит обработка ошибок. Для обработки ошибок мы также добавили переход из ErrorState в начальное состояние Idle после того, как ошибка будет исправлена.

Важно отметить, что обработка ошибок должна быть учтена на всех уровнях разработки системы — не только в коде, но и в пользовательском интерфейсе, базе данных и так далее.

Кроме того, нужно определить, как система будет обрабатывать ошибки и неожиданные события. Например, если машина переполнена водой, система должна остановить наполнение и выдать сообщение об ошибке пользователю. Если происходит сбой в электронной системе управления, система должна выдать сообщение об ошибке и перейти в состояние «Ошибка».

Обработка ошибок — важная часть создания диаграммы состояний, ее нужно учитывать на всех этапах разработки системы. Она помогает создать более надежную и безопасную систему, которая будет работать эффективно и без ошибок.

Правила построения диаграммы состояний в нотации UML

Диаграмма состояний состоит из состояний, переходов между ними и событий, которые вызывают переходы.

В диаграмме состояний определяется начальное состояние системы, конечное состояние, возможные состояния и действия, которые выполняются при переходе между ними.

Каждое состояние может иметь свои свойства и поведение, которые описываются дополнительными диаграммами. Также состояния могут объединяться в группы, чтобы обозначить общее поведение или выполнение одной и той же функции.

При построении UML диаграммы состояний (State Machine) следует соблюдать правила:

1. **Определить название системы**, для которой разрабатывается диаграмма состояний.

   Примеры:

   - Система управления банковским счетом.
   - Система управления графическим редактором.
   - Система управления производственным процессом.
   - Система управления складом.
   - Система управления проектами.
   - Система управления телефонной станцией.
   - Система управления трафиком на перекрестке.
   - Система управления образовательным процессом.
   - Система управления зданием.
   - Система управления логистическими процессами.

2. **Определить начальное состояние** (Initial State), в котором система находится при старте.

   Пример: при старте система находится в состоянии «Неактивный». Для приложения начальным состоянием может быть «Загрузка приложения», в которое система переходит после старта.

3. **Определить конечное состояние** (Final State), в котором система оказывается по завершении работы.

   Пример: состояние «Заказ выполнен» для системы управления заказами в интернет-магазине. Оно означает, что заказ обработан и товары отправлены покупателю — работа системы завершена и больше не может выполнять никаких операций.

4. **Определить возможные состояния** (States), в которых система может находиться во время работы.

5. **Определить переходы между состояниями** (Transitions). Переходы могут быть условным (зависят от некоторого условия) или безусловным.

Пример: состояния и переходы для системы продажи билетов в кино.

- «Начальное состояние» — система находится в этом состоянии при старте;
- «Выбор фильма» — пользователь может выбрать фильм из списка доступных;
- «Выбор места» — пользователь может выбрать место в зале;
- «Оплата» — пользователь должен оплатить билет;
- «Выдача билета» — пользователь получил билет;
- «Отмена операции» — операция может быть отменена пользователем в любой момент.

6. **Определить действия** (Actions), которые выполняются при переходе из одного состояния в другое. Действие может быть как кодом, который должен быть выполнен, так и изменением некоторых параметров системы.

   Пример: действия в диаграмме состояний для онлайн-магазина, которые могут выполняться при переходе из одного состояния в другое.

   - Добавление товара в корзину — выполняется при переходе из состояния «Просмотр товаров» в «Корзина»;
   - Удаление товара из корзины — выполняется при переходе из состояния «Корзина» в «Просмотр товаров»;
   - Изменение количества товаров в корзине — выполняется при переходе из состояния «Корзина» в «Корзина»;
   - Оформление заказа — выполняется при переходе из состояния «Корзина» в «Оформление заказа»;
   - Отмена заказа — выполняется при переходе из состояния «Оформление заказа» в «Корзина»;
   - Подтверждение заказа — выполняется при переходе из состояния «Оформление заказа» в «Заказ подтвержден».

   Действия могут включать изменение состояний параметров, таких как количество товаров в корзине, а также выполнение некоторых кодовых операций, например, вычисление стоимости заказа.

7. **Определить события** (Events), которые вызывают переходы между состояниями. Событие может быть как внутренним, так и внешним воздействием на систему.

   Пример: автоматизированная система управления складом.

   - Внешнее событие — на склад приходит новый товар. Происходит переход из состояния «Склад пуст» в состояние «Склад частично заполнен».
   - Внутреннее событие — изменение статуса заказа на доставку. Вызывает переход из состояния «Склад частично заполнен» в состояние «Склад заполнен на 70%».

8. **Определить группы состояний** (State groups), которые имеют общее поведение или выполняют одну и ту же функцию.

   Пример: диаграмма состояний для автомата по продаже напитков. Можно определить несколько групп состояний.

   - Группа состояний для выбора напитка — «Выбор напитка», «Выбор размера» и «Выбор дополнительных ингредиентов». Все эти состояния относятся к выбору напитка и выполняют одну и ту же функцию.
   - Группа состояний для обработки платежа — «Внесение денег», «Подтверждение платежа» и «Выдача сдачи». Все эти состояния относятся к обработке платежа и выполняют одну и ту же функцию.

9. **Определить иерархию состояний** (State hierarchy), в которой одно состояние может содержать другие.

   Пример: иерархия состояний для системы управления электронным замком:

   - Состояние «Закрытый замок» — замок закрыт. Начальное состояние системы.
   - Состояние «Открытый замок» — замок открыт. Для перехода из состояния «Закрытый замок» в состояние «Открытый замок» должен выполниться определенный набор условий.
   - Состояние «Замок в процессе открытия» — описывает процесс открытия замка. Система проверяет условия и совершает действия, необходимые для открытия замка.

- Состояние «Замок в процессе закрытия» — описывает процесс закрытия замка. Система проверяет условия и совершает действия, необходимые для закрытия замка.

Каждое из состояний «Замок в процессе открытия» и «Замок в процессе закрытия» может содержать дополнительные состояния для более подробного описания открытия или закрытия замка: например, состояние «Проверка ключа» или «Проверка кодового слова».

Таким образом, у этой системы есть иерархия состояний, в которой одно состояние может содержать другие для детального описания процесса.

10. **Определить ограничения** (Constraints), которые накладываются на переходы или состояния.

    Пример: диаграмма состояний для системы управления автомобилем. В системе ограничением может быть указано, что переход в состояние «Включенный» возможен, только если в замке зажигания есть ключ.

    Другое ограничение — если автомобиль разгонится до скорости более 120 км/ч, система автоматически перейдет в состояние «Аварийное», независимо от внешних событий.

Разрабатываем диаграмму состояний для системы управления термостатом

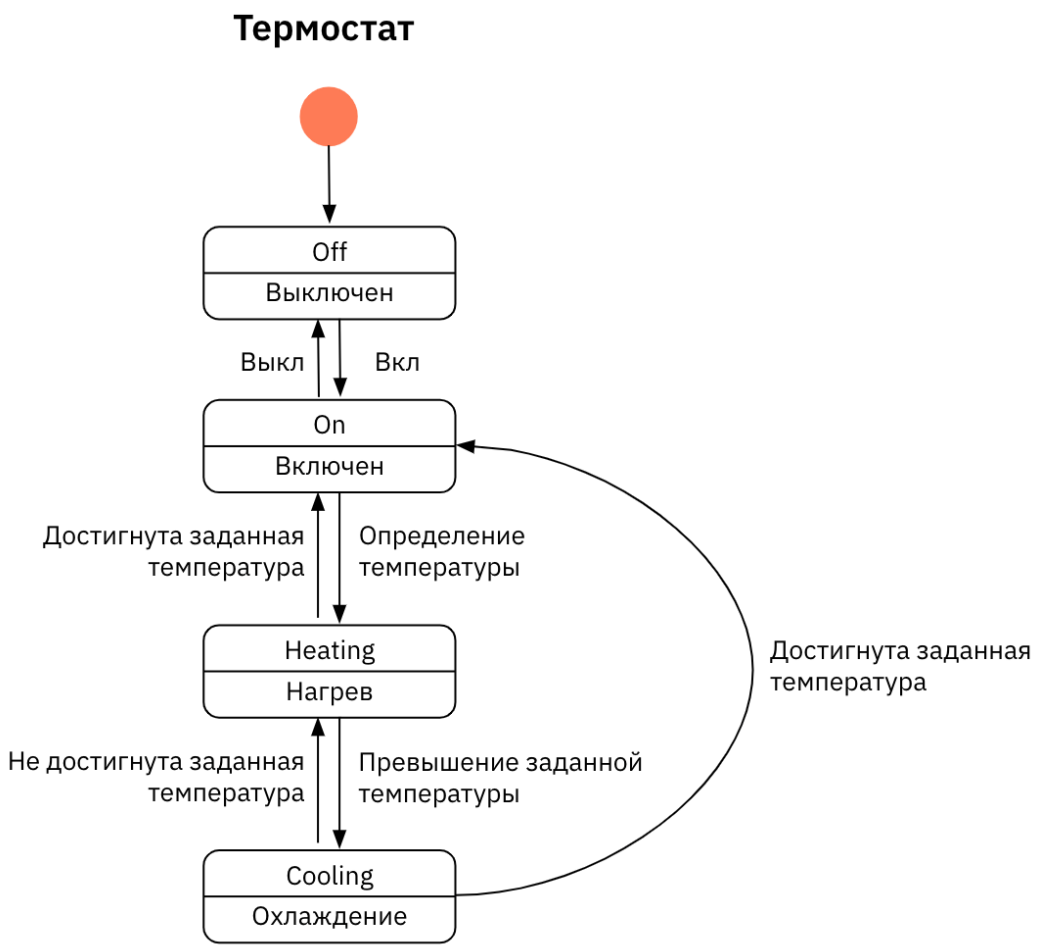
Допустим, перед нами стоит задача — разработать диаграмму состояний для системы управления термостатом. Следуя правилам построения диаграмм в нотации UML, получим следующую диаграмму состояний:

```
@startuml
title Термостат

[*] --> Off

Off : ВЫКЛ

Off --> On : ВКЛ
On : ВКЛ

On --> Off : ВЫКЛ
On --> Heating : Определение температуры

Heating : Нагрев
Heating --> On : Достигнута заданная температура
Heating --> Cooling : Превышение заданной температуры

Cooling : Охлаждение
Cooling --> On : Достигнута заданная температура
Cooling --> Heating : Не достигнута заданная температура

@enduml
```

На этой диаграмме состояний можно увидеть:

- Off — начальное состояние системы.
- On — состояние, в котором система находится после включения.
- Heating и Cooling — состояния, в которые система переходит в зависимости от определения температуры.
- [*] — символ начала диаграммы.
- --> — безусловный переход между состояниями.
- Текст после : — описание состояния.

Также можно добавить действия и события для каждого перехода и состояния, чтобы более детально описать работу системы.



Применение изученных инструментов

Представим, что мы работаем в ИТ-компании и к нам на вход приходит такое описание программы.

Кейс 1: программа для записи к врачу

Программное обеспечение для онлайн-записи к врачу позволяет пациентам записываться на прием через интернет без необходимости обращаться в медицинское учреждение лично.

Функционал, который может включать программа:

- просмотр доступных врачей и их расписания,
- выбор времени и даты приема,
- подтверждение записи через электронную почту или SMS,
- отмена или изменение записи,
- напоминание о записи,

- предоставление доступа с информацией о медицинской карте пациента и истории записей.

Вы можете дополнять этот кейс недостающей информацией.

Это кейс, на базе которого мы будем прорабатывать наше решение.

Конечный автомат

Составим конечный автомат с **описанием состояния приложения для записи к врачу.**

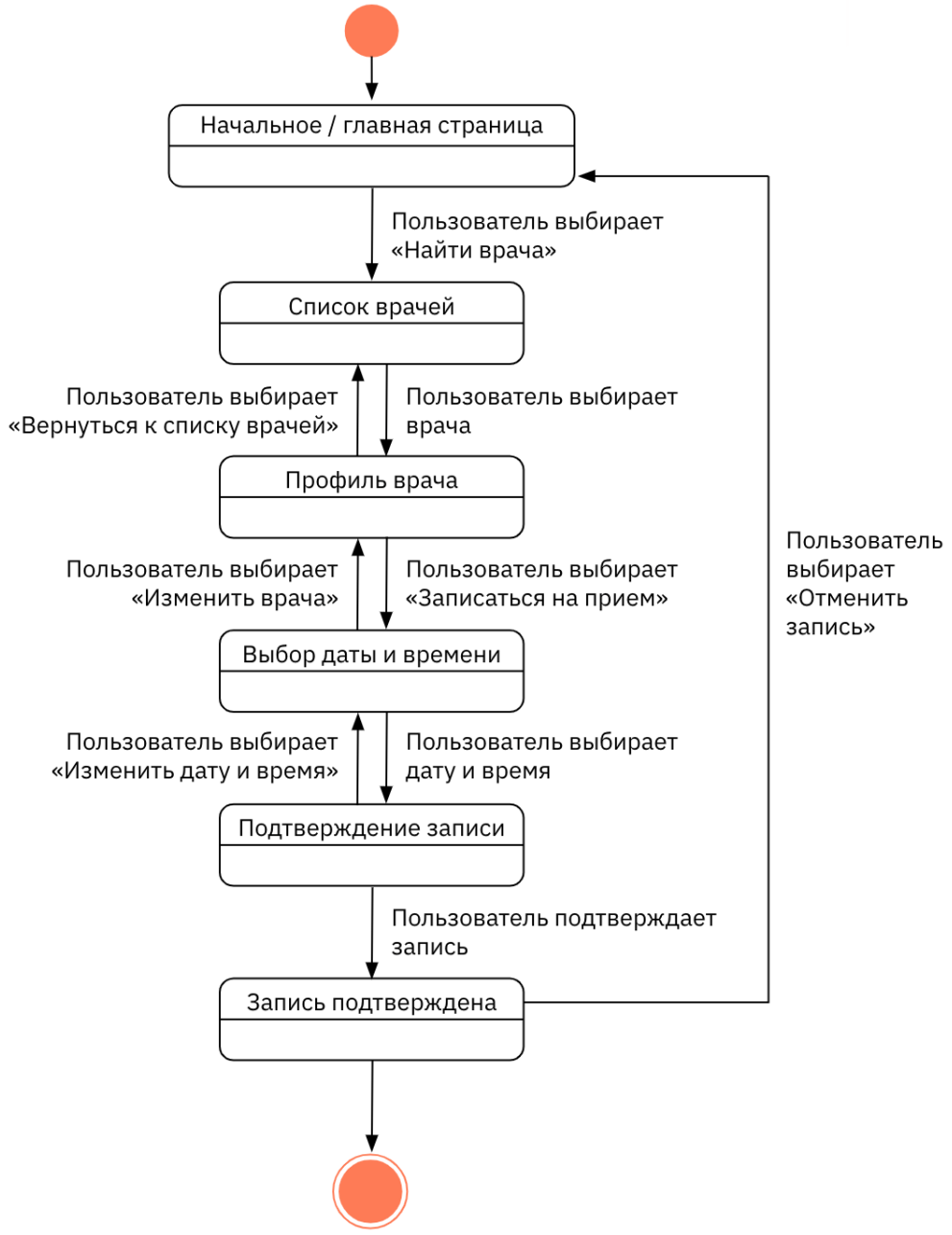
Пример таблицы переходов:

Текущее состояние	Событие	Следующее состояние
Начальное / главная страница	Пользователь выбирает «Найти врача»	Список врачей
Список врачей	Пользователь выбирает врача	Профиль врача
Профиль врача	Пользователь выбирает «Записаться на прием»	Выбор даты и времени
Выбор даты и времени	Пользователь выбирает дату и время	Подтверждение записи
Подтверждение записи	Пользователь подтверждает запись	Запись подтверждена
Запись подтверждена	Пользователь выбирает «Вернуться на главную»	Начальное / Главная страница

В этой таблице переходов отображаются все возможные переходы между состояниями конечного автомата, а также соответствующие события и действия, которые происходят при переходе между состояниями.

Диаграмма состояний

Отрисуем состояние системы с помощью нотации UML:



Пояснение:

- **Начальное / главная страница** — это первое состояние, которое пользователь видит при открытии приложения. Здесь он может перейти к списку врачей для выбора врача.

- **Список врачей** — в этом состоянии пользователь просматривает список доступных врачей и может выбрать одного из них для более подробной информации.
- **Профиль врача** — здесь пользователь просматривает информацию о выбранном враче и может решить записаться на прием. Из этого состояния пользователь может перейти в предыдущее, нажав на кнопку «Вернуться к списку врачей».
- **Выбор даты и времени** — когда пользователь выбрал врача, он переходит к выбору даты и времени приема. Из этого состояния пользователь может перейти в предыдущее, нажав на кнопку «Изменить врача».
- **Подтверждение записи** — в этом состоянии пользователь подтверждает свою запись, просматривая выбранную информацию о враче, дате и времени. Из этого состояния пользователь может перейти в предыдущее, нажав на кнопку «Изменить дату и время».
- **Запись подтверждена** — после подтверждения записи пользователь видит окно с подтверждением. Он может отменить запись после чего его система перекинет на главную страницу.

Что можно почитать еще?

1. GUID — Википедия
2. Использование диаграммы классов UML при проектировании и документировании программного обеспечения / Хабр
3. Полное руководство по диаграмме классов UML — Кибермедиана

Используемая литература

1. UML-диаграммы классов : сущности, связи, интерфейсы