МОДЕЛИ ПРЕДСТАВЛЕНИЯ ПРОГРАММНОГО СРЕДСТВА И ИХ ОПИСАНИЕ

Полезные ссылки

1. Градди Буч. Объектно-ориентированный анализ и проектирование с примерами приложений, 3-е изд.
2. Мартин Фаулер. UML. Основы. Третье издание. Краткое руководство по стандартному языку объектного моделирования.
3. Крэг Ларман. Применение UML 2.0 и шаблонов проектирования. Введение в объектно-ориентированный анализ, проектирование и итеративную разработку.

Краткие теоретические сведения и методические указания

> Представление системы с определенной точки зрения (view point) называется **видом** (view). Вид также называют **моделью системы** с определенной точки зрения.

Каждая модель определяет конкретный аспект системы, использует набор диаграмм и документов заданного формата, а также отражает точку зрения и является объектом деятельности различных людей с конкретными интересами, ролями или задачами. Модель абстрагирует свойства, поведение и структуру системы.

Программная система может рассматриваться **со следующих точек зрения**:

- функциональных требований к системе (use case view);
- логической организации и поведения системы (logical design view);
- функционирования или работы процессов системы (process view);
- компонентной организации и поведения системы (implementation view или component view);
- требований к размещению компонент на аппаратных ресурсах (deployment view).

В зависимости от точки зрения мы получаем различные **виды или модели представления программной системы**:

- ***Use case view*** (представление вариантов использования или прецедентов) – предназначен для моделирования функциональных требований к системе;
- ***Logical view*** (логические представление) – предназначен для моделирования логической структуры и поведения системы;
- ***Component view*** (представление компонент) – предназначен для моделирования архитектуры системы на уровне компонент;
- ***Deployment view*** (представление размещения или развертывания) – предназначен для моделирования развертывания компонент системы на аппаратуре.

Иногда Component view и Deployment view объединяют в один вид, который называют ***Implementation view***.

Ограничимся рассмотрением вида представления **Logical view** ***в целях моделирования поведения программной системы –*** *модели поведения и взаимодействия объектов*. Модель представления ***Use case view*** рассмотрена ранее , модели представления ***Logical view (диаграмма классов), Component view и Deployment view*** будут рассмотрены позже.

Для моделирования **поведения объектов** программной системы используются следующие диаграммы:

- ***диаграмма деятельности*** (activity diagram) – моделируются последовательности действий, исполняемых объектами программной системы;
- ***диаграмма состояний*** (statechart diagram или state machine diagram) – моделируются состояния объектов программной системы, а также переходы между этими состояниями.

Для моделирования **взаимодействия объектов** программной системы используются диаграммы взаимодействия, включающие два вида диаграмм:

- ***диаграмму последовательности*** (sequence diagram) – моделирует упорядоченное во времени взаимодействие объектов системы;
- ***диаграмму взаимодействия*** или ***коммуникации*** (collaboration или communication diagram) – моделирует общую схему взаимодействия объектов и роли объектов во взаимодействии.

Моделирование поведения объектов программной системы

Диаграммы деятельности используются для моделирования поведения системы в рамках различных вариантов использования или потоков управления.

Также могут применяться и для моделирования бизнес-процессов. Однако, применительно к бизнес-процессам, желательно выполнение каждого действия ассоциировать с конкретным подразделением компании, т.е. использовать дорожки. Назначение дорожек состоит в том, чтобы указать зоны ответственности за выполнения отдельных деятельностей в рамках моделируемого бизнес-процесса. В качестве имен дорожек используются либо названия подразделений (департаментов) рассматриваемой компании, либо названия отдельных должностей сотрудников тех или иных подразделений.

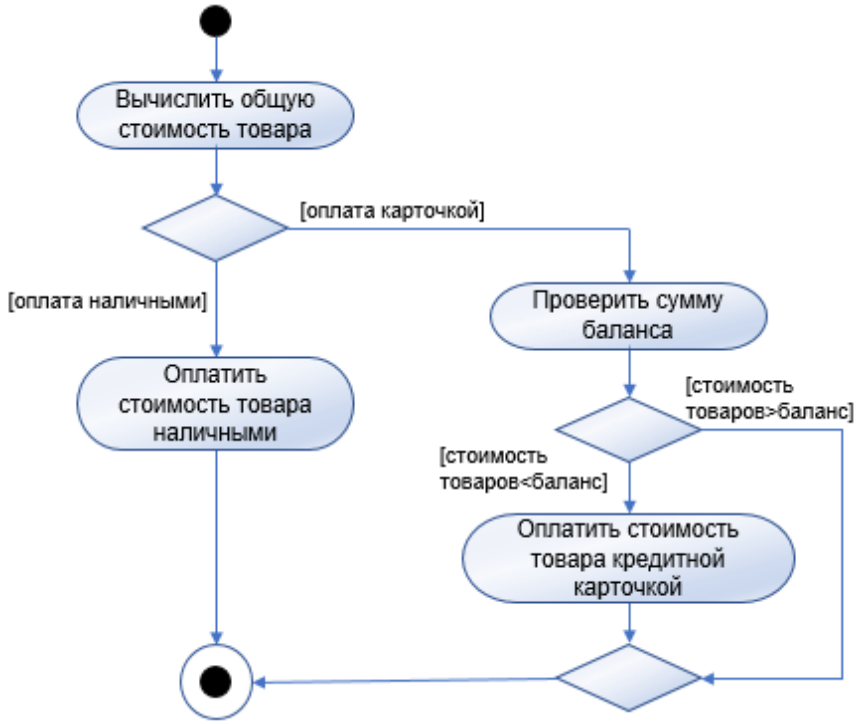
Диаграммы деятельности также полезны и для описании поведения, включающего большое количество параллельных процессов. Не менее важная область их применения связана с моделированием бизнес-процессов при параллельном программировании, поскольку можно графически изобразить все ветви и определить, когда их необходимо синхронизировать.

Принять во внимание при разработке диаграммы деятельности

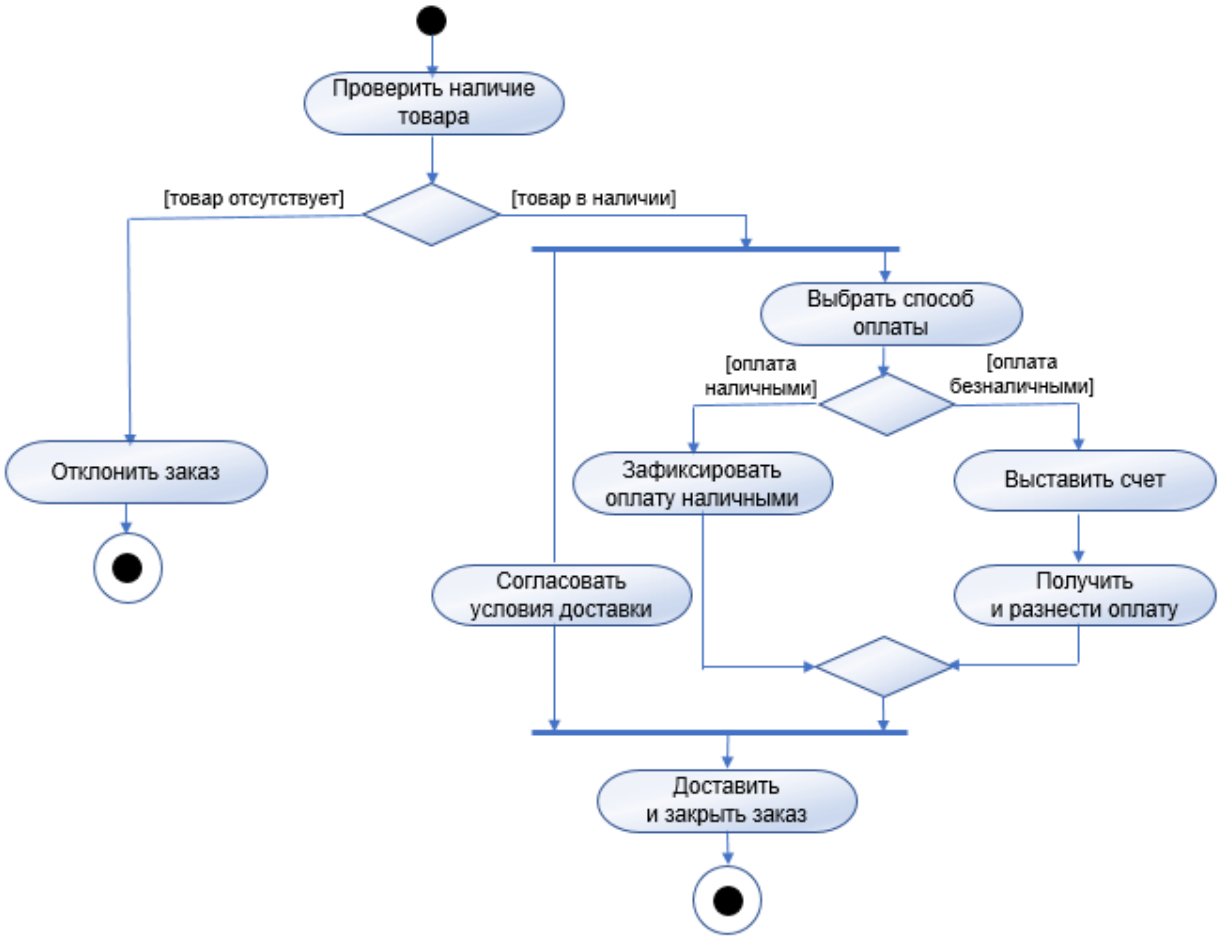
- Деятельность (действие) на диаграмме изображается **прямоугольником со скругленными сторонами (а не углами!!!).**

- С точки зрения топологии стрелок узлы *decision (ромб)* и *fork (утолщенная линия)* схожи (один входной поток и несколько выходных). Разница между ними в параллельности: на выходе *decision-узла* появляется только один поток (из нескольких возможных), а на выходе *fork-узла* – несколько параллельных потоков.
- Хорошо структурированная диаграмма деятельности сконцентрирована на описании одного аспекта динамики системы. Должна содержать только те элементы, которые существенны для понимания этого аспекта.
- Сложные деятельности можно дополнительно детализировать, разбив на действия и изобразив «диаграмму в диаграмме».

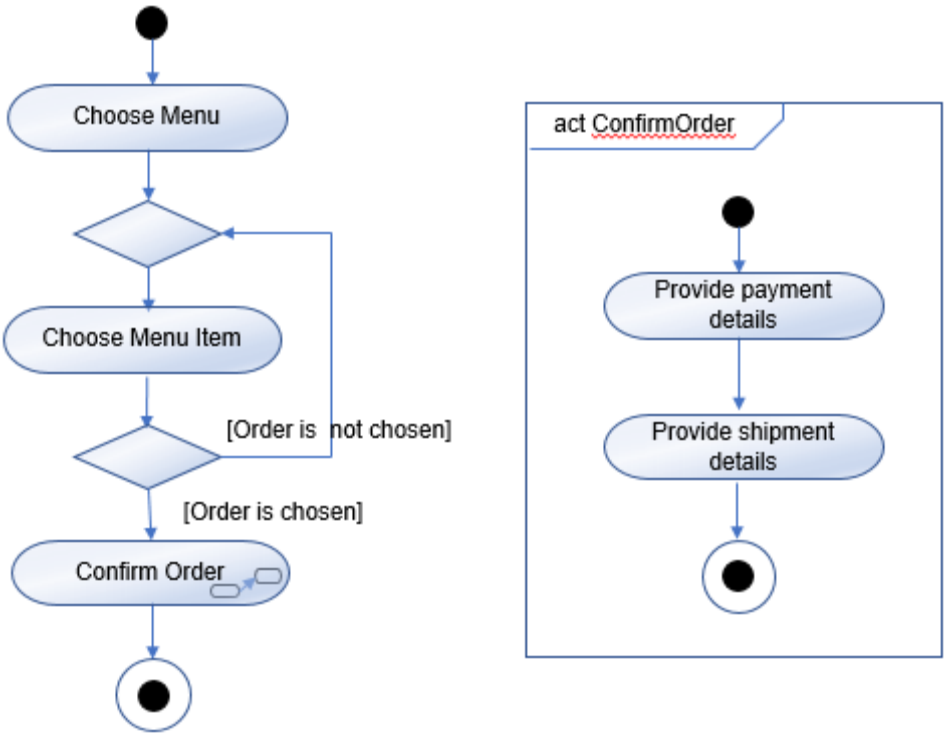
Пример 1



Пример 2 Диаграмма деятельности с параллельными потоками



Пример 3 Графическое изображение состояния под-деятельности



Пример 3 Диаграмма деятельности с дорожками, объектами и с параллельными потоками

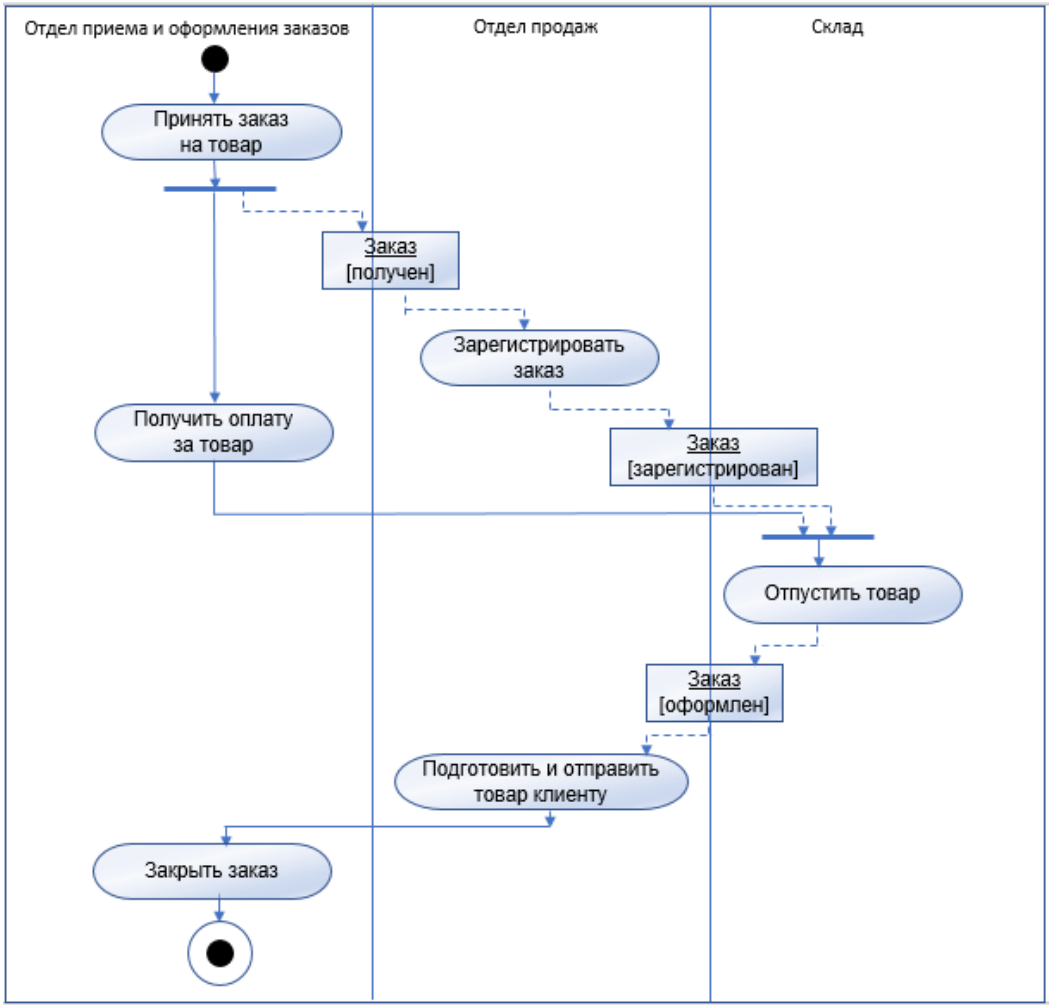


Диаграмма состояний предназначена для моделирования динамического поведения объектов модели, используя их состояния и переходы между этими состояниями. Диаграмма состояний определяет все возможные состояния, в которых может находиться конкретный объект, а также процесс смены состояний объекта в результате наступления некоторых событий.

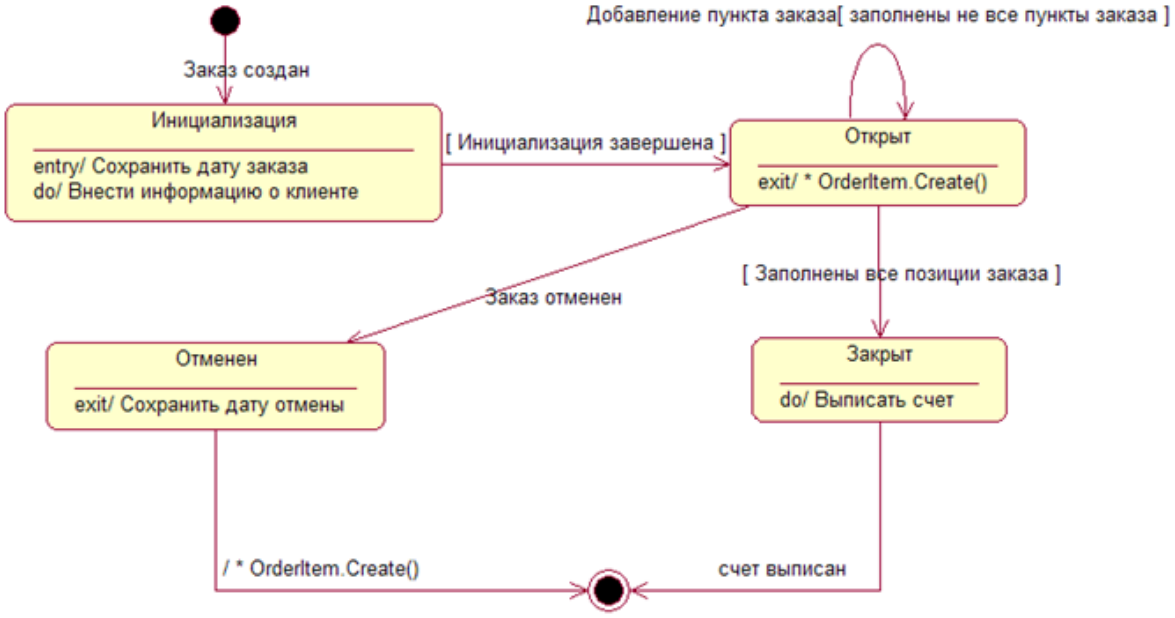
Диаграммы состояний чаще всего используются для *описания поведения отдельных объектов, а также спецификации функциональности других компонентов моделей, таких как варианты использования, актеры, подсистемы, системы*.

Принять во внимание при разработке диаграммы состояний

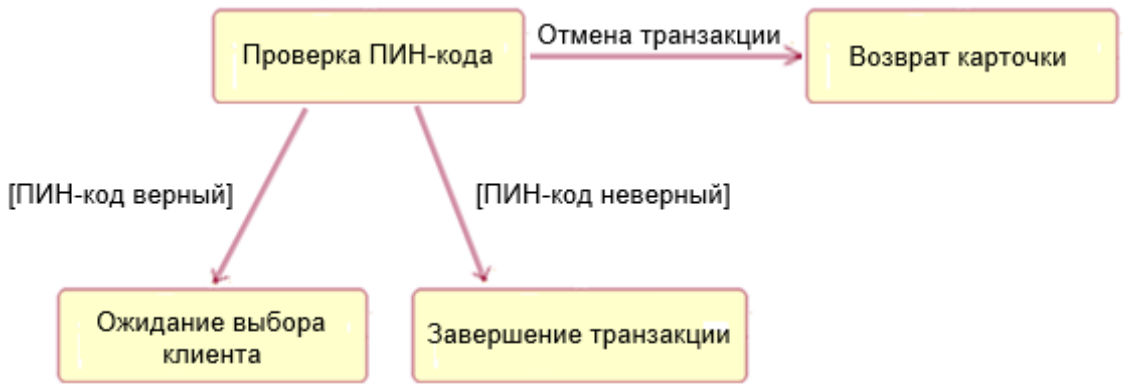
- Состояние на диаграмме изображается прямоугольником ***со скругленными вершинами (а не сторонами!!!).***
- ***На диаграмме состояний decision-узлы не используем!!!***
- Диаграммы состояний не требуется создавать для каждого класса. Если объект класса может существовать в нескольких состояниях и в каждом из них ведет себя по-разному, то для него может потребоваться диаграмма состояний. Диаграммы состояний *создаются для описания объектов с высоким уровнем динамического поведения*.

- Из каждого состояния на диаграмме не может быть самопроизвольного перехода в какое бы то ни было другое состояние. *Все переходы должны быть явно специфицированы, в противном случае построенная диаграмма состояний является либо неполной (неадекватной), либо ошибочной с точки зрения нотации языка UML.*
- При построении диаграммы необходимо выполнить требование отсутствия конфликтов у всех переходов, выходящих из одного и того же состояния. Наличие такого конфликта может служить признаком ошибки, либо параллельности или ветвления рассматриваемого процесса. Если параллельность по замыслу разработчика отсутствует, то следует ввести дополнительные сторожевые условия либо изменить существующие, чтобы исключить конфликт переходов. При наличии параллельности следует заменить конфликтующие переходы одним параллельным переходом типа ветвления.

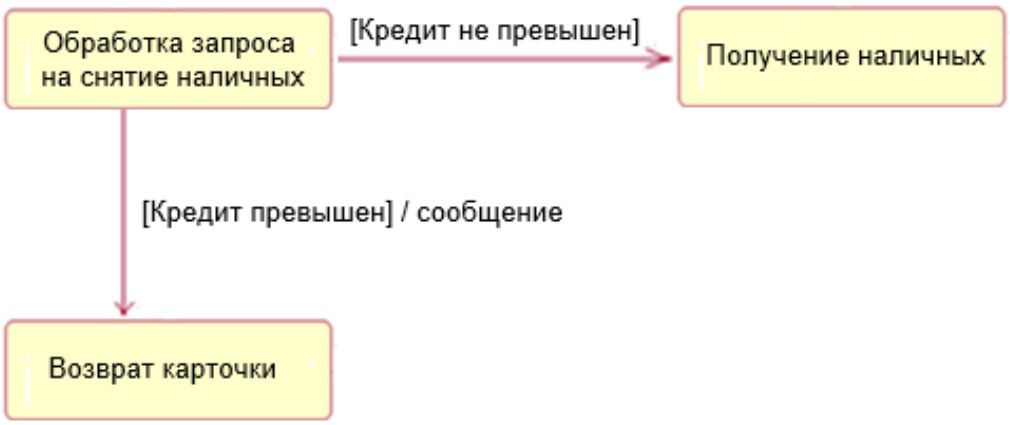
Пример 1



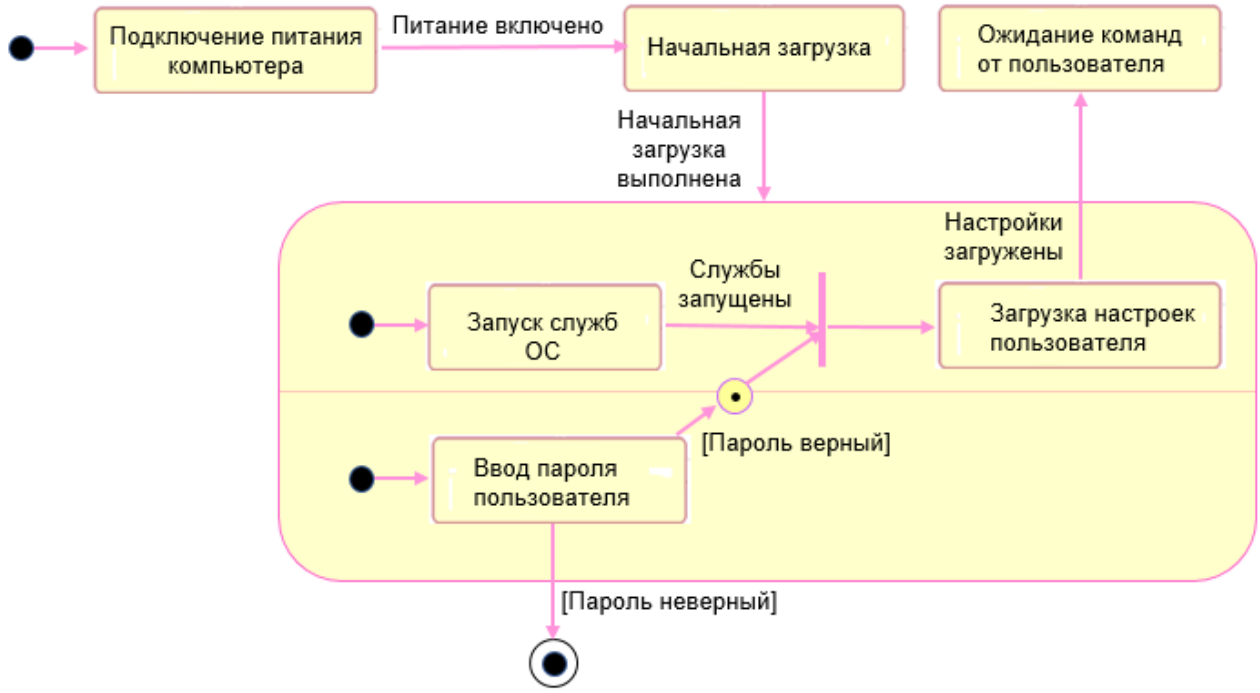
Пример 2 Триггерные и нетриггерные переходы на диаграмме состояний



Пример 3 Выражение действия перехода на диаграмме состояний



Пример 4 Моделирование параллельного поведения



Моделирование взаимодействия объектов программной системы

Диаграммы взаимодействия описывают поведение взаимодействующих групп объектов (в рамках варианта использования или некоторой операции класса). Как правило, диаграмма взаимодействия охватывает поведение объектов в рамках только одного потока событий варианта использования. На такой диаграмме отображается ряд объектов и те сообщения, которыми они обмениваются между собой.

Для моделирования **взаимодействия объектов** программного средства следует разработать только **диаграмму последовательности**.

Диаграммы последовательности и коммуникации, по сути, отображают одну и ту же информацию, но представляют ее с различных точек зрения. Подобно диаграммам последовательности кооперативные диаграммы отображают

поток событий варианта использования. Диаграммы последовательности упорядочены по времени, а кооперативные диаграммы концентрируют внимание на связях между объектами. Большинство Case-средств позволяет после построения одной из диаграмм автоматически получить другую, а также выполнять синхронизацию этих диаграмм между собой.

Диаграмма последовательности описывает поведение взаимодействующих объектов **в рамках реализации конкретного варианта использования**. Ее используют *для уточнения варианта использования, более детального описания логики сценариев использования (рисунок 1)*.

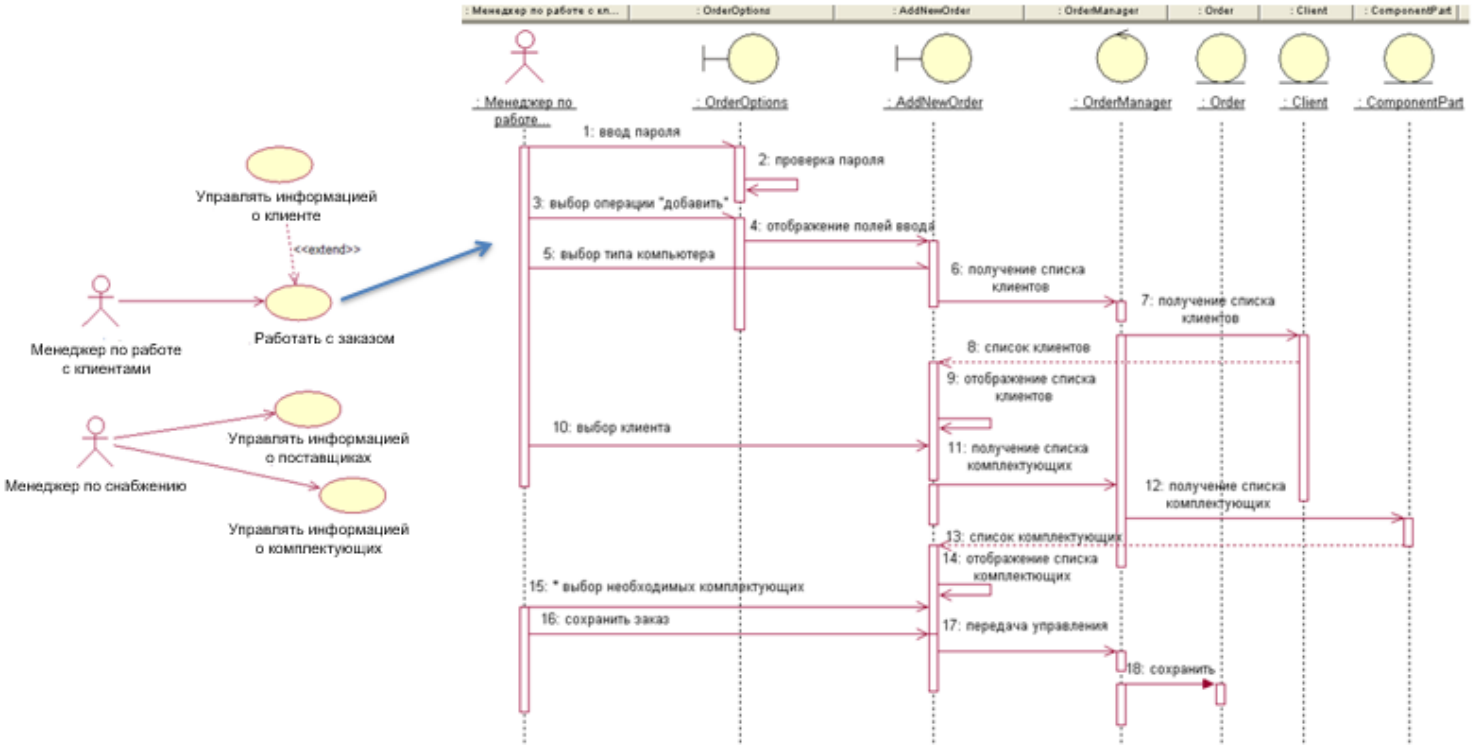


Рисунок 1 – Диаграмма последовательности для варианта использования Работать с заказом (поток событий Добавить новый заказ)

Вариант использования **Работать с заказом** предусматривает несколько возможных потоков событий, таких как *добавить новый заказ; изменить заказ; удалить заказ; просмотреть заказ*.

Представленная на рисунке 1 диаграмма последовательности охватывает поведение объектов в рамках *только одного потока событий* **Добавить новый заказ** *варианта использования* **Работать с заказом**.

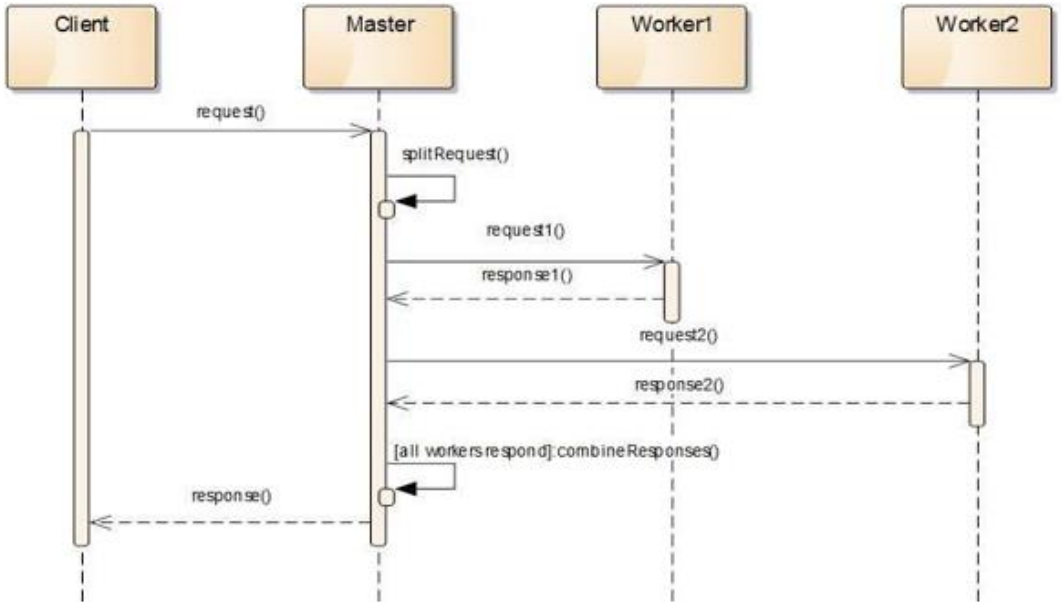
Пример описания сообщений

Номер сообщения	Объект-отправитель	Объект-получатель	Имя сообщения
1	Менеджер по работе с клиентами	OrderOptions	ввод пароля
2	OrderOptions	OrderOptions	проверка пароля
3	Менеджер по работе с клиентами	OrderOptions	выбор операции "добавить"
4	OrderOptions	AddNewOrder	отображение полей ввода
5	Менеджер по работе с клиентами	AddNewOrder	выбор типа компьютера
6	AddNewOrder	OrderManager	получение списка клиентов
7	OrderManager	Client	получение списка клиентов
8	Client	AddNewOrder	список клиентов
9	AddNewOrder	AddNewOrder	отображение списка клиентов
10	Менеджер по работе с клиентами	AddNewOrder	выбор клиента
…	…	…	…

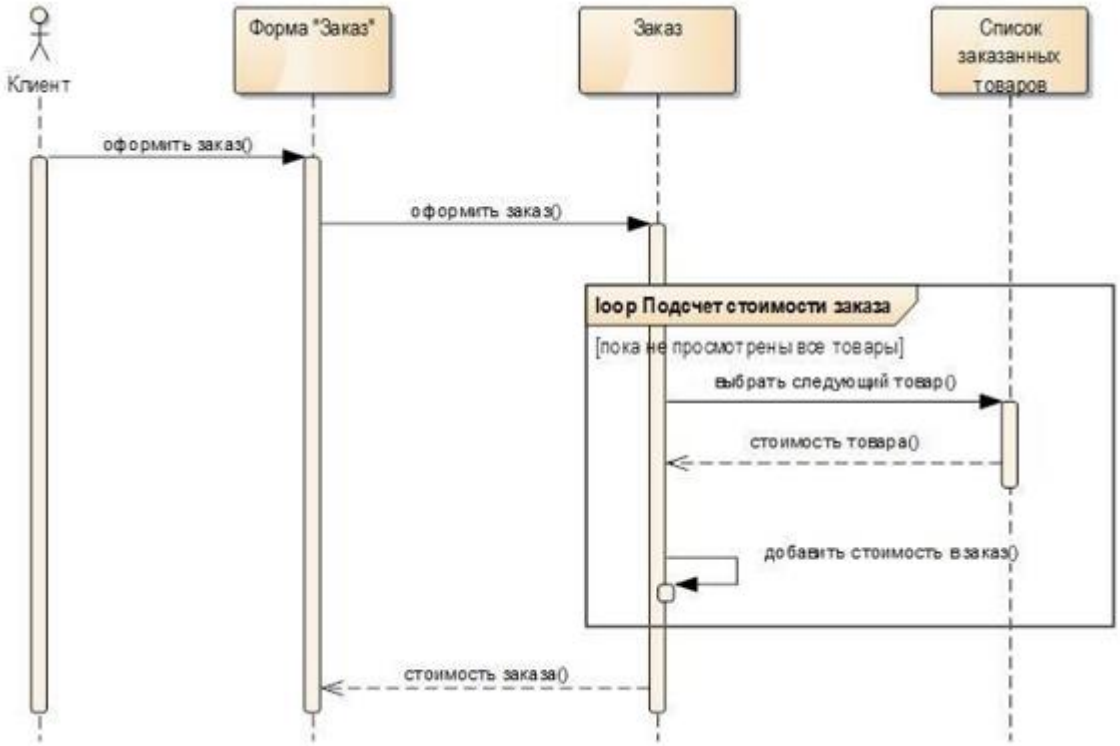
Принять во внимание при разработке диаграммы последовательности

- Как правило, диаграмма последовательности охватывает поведение объектов в рамках *только одного потока событий варианта использования*. Для отображения других вариантов развития событий следует построить несколько диаграмм.
- Диаграммы последовательности обязательно опираются на диаграмму классов. Объекты должны принадлежать классам, описанным в диаграмме классов. Если создаваемый объект не может принадлежать ни одному из существующих классов, возможно, следует доработать диаграмму классов. Поэтому в диаграммах последовательности нет необходимости создавать новые объекты, их достаточно просто перетащить в рабочую область текущей диаграммы из соответствующей диаграммы классов и определить имя экземпляра данного класса.
- Построение диаграммы последовательности целесообразно начинать с выделения из всей совокупности тех и только тех классов, объекты которых участвуют в моделируемом взаимодействии. После этого все объекты наносятся на диаграмму с соблюдением некоторого порядка инициализации сообщений. Когда объекты визуализированы, приступают к спецификации сообщений.

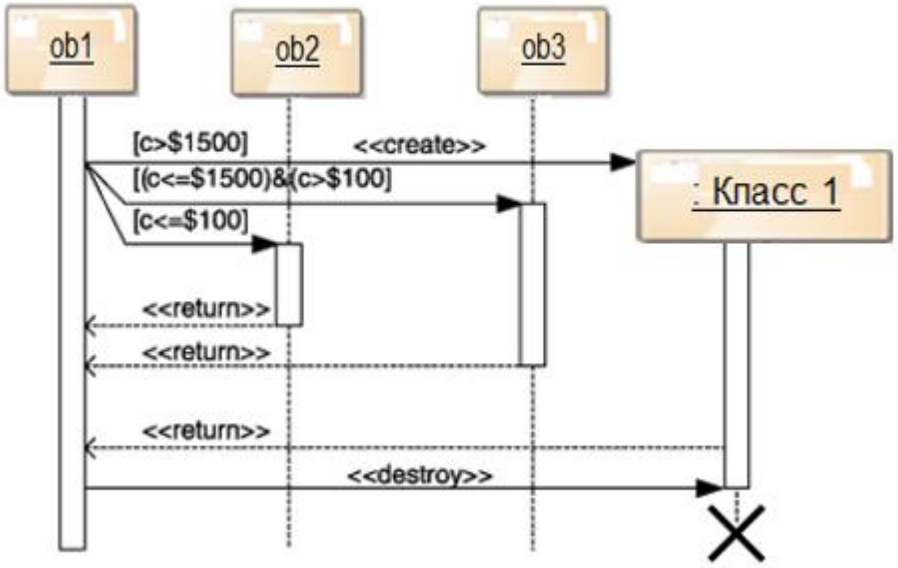
Пример 1



Пример 2



Пример 3


ob1
ob2
ob3
: Класс 1
[c>$1500]
<<create>>
[(c<=$1500)&(c>$100]
[c<=$100]
<<return>>
<<return>>
<<return>>
<<destroy>>