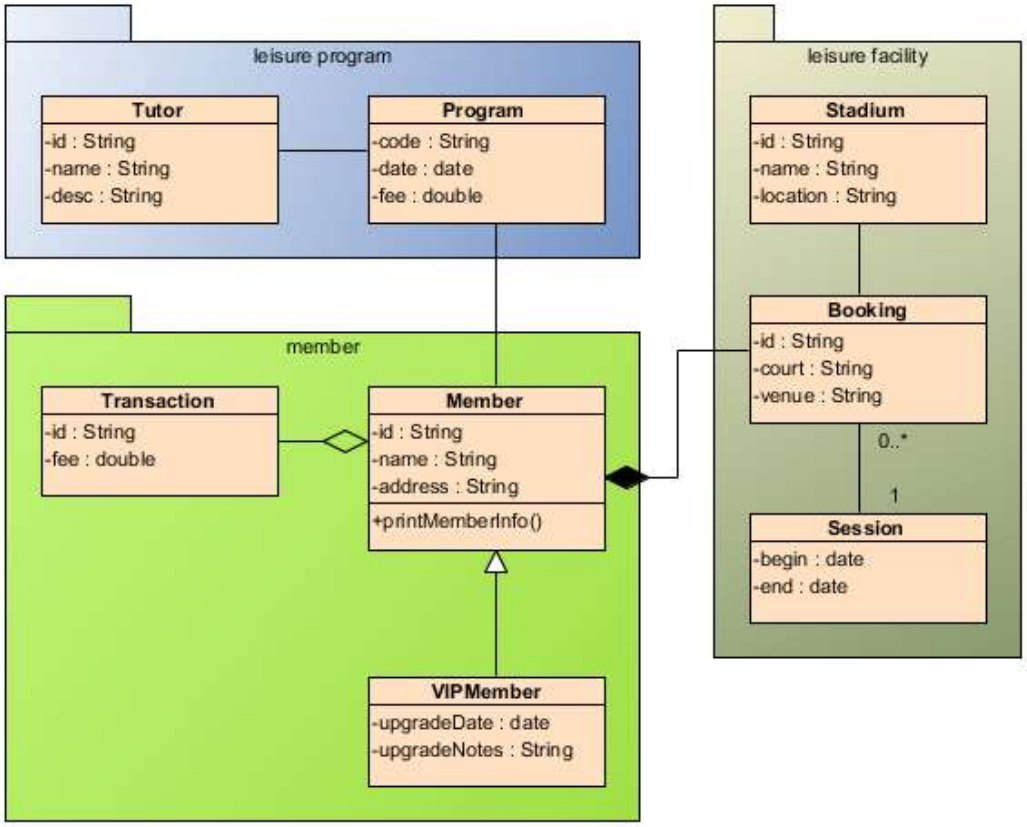
2 Unified Modeling Language

2.1 Понятие и применение UML

2.1.1 UML — является графическим языком для визуализации, описания параметров, конструирования и документирования различных систем (программ в частности)



2.1.2 Типы диаграмм

- Диаграмма вариантов использования (use case diagram)

Проектируемая система представляется в виде множества сущностей или актеров, взаимодействующих с системой с помощью, так называемых прецедентов.

- Диаграмма классов (class diagram)

Диаграмма классов служит для представления статической структуры модели системы в терминологии классов объектно-ориентированного программирования. Диаграмма классов может отражать, в частности, различные взаимосвязи между отдельными сущностями предметной области, такими как объекты и подсистемы, а также описывает их внутреннюю структуру (поля, методы…) и типы отношений (наследование, реализация интерфейсов …).

- Диаграмма состояний (statechart diagram)

Главное предназначение этой диаграммы — описать возможные последовательности состояний и переходов, которые в совокупности характеризуют поведение элемента модели в течение его жизненного цикла. Диаграмма состояний представляет динамическое поведение

сущностей, на основе спецификации их реакции на восприятие некоторых конкретных событий.

- Диаграмма последовательности (sequence diagram)

Для моделирования взаимодействия объектов в языке UML используются соответствующие диаграммы взаимодействия. Взаимодействия объектов можно рассматривать во времени, и тогда для представления временных особенностей передачи и приема сообщений между объектами используется диаграмма последовательности. Взаимодействующие объекты обмениваются между собой некоторой информацией. При этом информация принимает форму законченных сообщений.

- Диаграмма кооперации (collaboration diagram)

На диаграмме кооперации в виде прямоугольников изображаются участвующие во взаимодействии объекты, содержащие имя объекта, его класс и, возможно, значения атрибутов. Как и на диаграмме классов, указываются ассоциации между объектами в виде различных соединительных линий. При этом можно явно указать имена ассоциации и ролей, которые играют объекты в данной ассоциации.

В отличие от диаграммы последовательности, на диаграмме кооперации изображаются только отношения между объектами, играющими определенные роли во взаимодействии

- Диаграмма компонентов (component diagram)

Диаграмма компонентов, в отличие от ранее рассмотренных диаграмм, описывает особенности физического представления системы. Диаграмма компонентов позволяет определить архитектуру разрабатываемой системы, установив зависимости между программными компонентами, в роли которых может выступать исходный, бинарный и исполняемый код. Во многих средах разработки модуль или компонент соответствует файлу. Пунктирные стрелки, соединяющие модули, показывают отношения взаимозависимости, аналогичные тем, которые имеют место при компиляции исходных текстов программ. Основными графическими элементами диаграммы компонентов являются компоненты, интерфейсы и зависимости между ними.

- Диаграмма развертывания (deployment diagram)

Диаграмма развертывания предназначена для визуализации элементов и компонентов программы, существующих лишь на этапе

ее исполнения (runtime). При этом представляются только компоненты-экземпляры программы, являющиеся исполнимыми файлами или динамическими библиотеками. Те компоненты, которые не используются на этапе исполнения, на диаграмме развертывания не показываются.

Диаграмма развертывания содержит графические изображения процессоров, устройств, процессов и связей между ними. В отличие от диаграмм логического представления, диаграмма развертывания является единой для системы в целом, поскольку должна всецело отражать особенности ее реализации. Эта диаграмма, по сути, завершает процесс ООАП для конкретной программной системы и ее разработка, как правило, является последним этапом спецификации модели.

2.2 Основные обозначения в UML

2.2.1 Видимость

Для задания видимости членов класса (то есть любой атрибут или метод), эти обозначения должны быть размещены перед именем участника:

+	Публичный (Public)
-	Приватный (Private)
#	Защищённый (Protected)
/	Производный (Derived) (может быть совмещён с другими)
~	Пакет (Package)

2.2.2 Взаимосвязи

Взаимосвязь — это особый тип логических отношений между сущностями, показанных на диаграммах классов и объектов. В UML представлены следующие виды отношений:

Зависимость - обозначает такое отношение между классами, что изменение спецификации класса-поставщика может повлиять на работу зависимого класса, но не наоборот.

Ассоциация показывает, что объекты одной сущности (класса) связаны с объектами другой сущности таким образом, что можно перемещаться от объектов одного класса к другому. Является общим случаем композиции и агрегации.

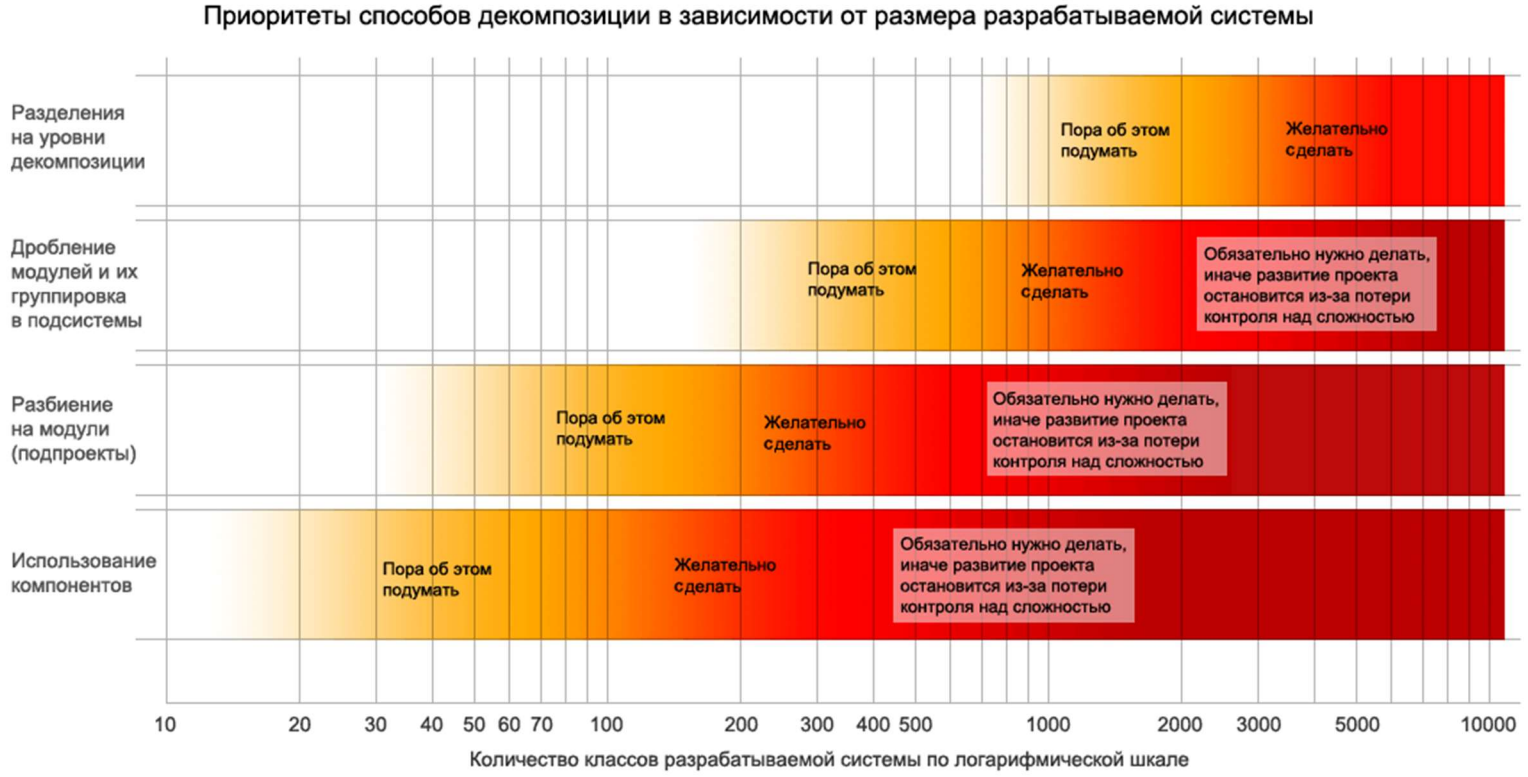
Агрегация — это разновидность ассоциации при отношении между целым и его частями. Как тип ассоциации агрегация может быть именованной. Одно отношение агрегации не может включать более двух классов (контейнер и содержимое)

Композиция — более строгий вариант агрегации. Известна также как агрегация по значению.

Композиция имеет жёсткую зависимость времени существования экземпляров класса контейнера и экземпляров содержащихся классов. Если контейнер будет уничтожен, то всё его содержимое будет также уничтожено.

2.3 Декомпозиция предметной области приложения, построение архитектуры приложения

2.3.1 Объектно-ориентированная декомпозиция включает мероприятия, направленные на разделение сложности, актуальность которых главным образом зависит от размера разрабатываемой системы. На следующей диаграмме показаны приоритеты наиболее важных декомпозиционных мероприятий в зависимости от общего числа классов:



2.3.2 Архитектура приложений

Архитектура приложений покрывает достаточно широкую область, которая начинается с идентификации того, какие прикладные системы нужны предприятию для выполнения бизнес-процессов, и включает такие аспекты, как проектирование, разработка (или приобретение) и интеграция прикладных систем

Критерии хорошей архитектуры

- Эффективность системы.
- Гибкость системы.
- Расширяемость системы.
- Масштабируемость процесса разработки.
- Тестируемость
- Возможность повторного использования.
- Хорошо структурированный, читаемый и понятный код. Сопровождаемость