1. **Объясните понятие «атрибут класса».**

**Атрибут** - это фрагмент информации связанный с классом С атрибутами можно связать три основных фрагмента информации: имя атрибута, тип его данных и первоначальное значение. Тип данных атрибута специфичен для используемого языка. При добавлении атрибута к классу каждый экземпляр класса получит свою собственную копию этого атрибута.

1. **Опишите основные типы атрибутов(класса)**

**Статичный атрибут** (static) - это такой атрибут, который используется всеми экземплярами класса. На языке UML статичный атрибут помечают символом «$».

**Производным** (derived) называется атрибут, полученный из одного или нескольких других атрибутов. Например, класс Rectangle (Прямоугольник) может иметь атрибуты Width (Ширина) и Height (Высота). У него также может быть атрибут Area (Площадь), вычисляемый как произведение ширины и высоты. Так как Area получается из этих двух атрибутов, он считается производным атрибутом. В нотации UML производные атрибуты помечают символом «/».

1. **Опишите основные типы операций. Опишите основные стереотипы операций.**

**Операции реализации** (implementor operations) реализуют некоторую бизнес-функциональность. Такие операции можно найти, исследуя диаграммы Взаимодействия. Диаграммы этого типа фокусируются на бизнес-функциональности. Необходимо, чтобы каждую операцию реализации можно было проследить до соответствующего требования.

**Операции управления** (manager operations) управляют созданием и разрушением объектов. В эту категорию попадают конструкторы и деструкторы классов.

Атрибуты обычно бывают закрытыми или защищенными. Тем не менее другие классы иногда должны просматривать или изменять их значения. Для этого предназначены **операции доступа** (access operations). Операции доступа дают возможность безопасно инкапсулировать атрибуты внутри класса, защищая их от других классов, но при этом позволяет осуществлять контролируемый доступ к ним.

**Вспомогательными** (helper operations) называются такие операции класса, которые необходимы ему для выполнения его ответственностей, но о которых другие классы не должны ничего знать. Это закрытые и защищенные операции класса. Как и операции реализации, вспомогательные операции можно обнаружить на диаграммах Последовательности и Кооперативных диаграммах. Часто такие операции являются рефлексивными сообщениями.

Как и в случае других элементов модели, для классификации операций создаются их стереотипы. Существуют четыре наиболее распространенных стереотипа операций:

1) Implementor (операции реализации) - операции, реализующие некоторую бизнес-логику;

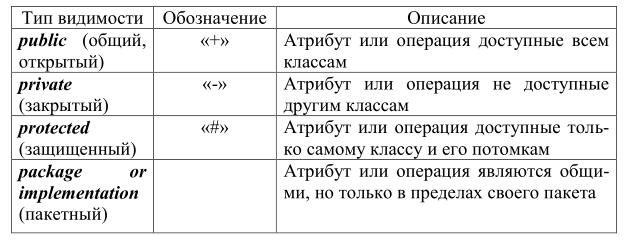
2) Manager (операции управления) - конструкторы, деструкторы и операции управления памятью;

3) Access (операции доступа) - операции, позволяющие другим классам просматривать или редактировать атрибуты данного класса. Как правило, такие операции называют Getter или Setter;

4) Helper (вспомогательные операции) - закрытые или защищенные операции, которые используются классом, но не видны другим классам.

**4) Объясните понятие «видимость» атрибута и операции класса. Перечислите допустимые значения видимости.**

С интерфейсом класса связана такая характеристика, как видимость. Видимость показывает, каким образом данные (атрибуты) и поведение (операции) инкапсулируются в класс.



**5)Опишите основные типы связей между классами (ассоциация, зависимость, агрегация, наследование, полиморфизм). Объясните понятие «дружественный класс».**

1. **Ассоциация** — это семантическая связь между классами. Ассоциация дает классу возможность узнавать об общих атрибутах и операциях другого класса. Ассоциацию часто используют на ранней стадии анализа, чтобы зафиксировать участников отношения. В дальнейшем ассоциации обычно уточняются, становясь более специализированной связью. После того, как классы связали ассоциацией, они могут передавать друг другу сообщения на диаграммах Взаимодействия. Ассоциации могут быть двунаправленными или однонаправленными.

2. **Связь зависимости** (использования) показывает, что один класс(клиент) ссылается на другой (сервер). Типичный случай проявления такого отношения - когда в реализации операции происходит объявление локального объекта используемого класса. Строгое отношение зависимости ограничительно, поскольку клиент имеет доступ только к открытой части интерфейса сервера. Зависимости всегда однонаправлены.

3. **Агрегация** (aggregations) представляет собой более тесную форму ассоциации. Агрегация — это связь между целым и его частями. Как и ассоциации, агрегации могут быть рефлексивными. Рефлексивные агрегации предполагают, что один экземпляр класса состоит из одного или нескольких экземпляров того же класса. Один и тот же класс может быть агрегирован несколькими классами. Один класс может участвовать в нескольких отношениях агрегации с другими классами. **Агрегация по значению** (By Value) предполагает, что целое и часть создаются и разрушаются одновременно. **Агрегация по ссылке** (By Reference) предполагает, что целое и часть могут создаваться и разрушаться в разное время.

4. С помощью **обобщений** (generalization) показывают связи наследования между двумя классами. **Наследование** — это такое отношение между классами, когда один класс повторяет структуру и поведение другого класса (одиночное наследование) или других классов (множественное наследование). Наследование позволяет одному классу наследовать все атрибуты, операции и связи другого. Класс, структура и поведение которого наследуются, называется суперклассом (предком). Производный от суперкласса класс называется подклассом (потомком). Наследование устанавливает между классами иерархию «общего/частного».

Проектирование структур классов с множественным наследованием осуществляется путем последовательных приближений. Есть две специфические для множественного наследования проблемы: как разрешить конфликты имен между суперклассами и что делать с повторным наследованием.

Конфликт имен происходит, когда в двух или более суперклассах случайно оказывается элемент (переменная или метод) с одинаковым именем. Для устранения этого к именам элементов добавляют префиксы, указывающие на имена классов-источников.

Вторая проблема множественного наследования возникает, когда один класс является наследником другого по нескольким линиям. Например, класс D наследует от B и C, которые, в свою очередь, наследуют от A. Эта ситуация называется повторным наследованием.

Проблема повторного наследования решается двумя способами. Во-первых, можно разделить две копии унаследованного элемента, добавив к именам префиксы в виде имени класса-источника. Во-вторых, можно рассматривать множественные ссылки на один и тот же класс как обозначающие один и тот же класс.

5. Наследование связано с явлением **полиморфизма.**

Полиморфизм допускает отнесение значений и переменных к нескольким типам. Полиморфизм позволяет обойтись без операторов выбора, поскольку объекты сами знают свой тип. Полиморфизм тесно связан с механизмом позднего связывания. При полиморфизме связь метода и имени определяется только в процессе выполнения программ. В C++ программист имеет возможность выбирать между ранним и поздним связыванием имени с операцией.

В C++ операции для позднего связывания объявляются виртуальными (virtual), а все остальные обрабатываются компилятором как обычные вызовы подпрограмм. Для управления виртуальными функциями в C++ используется концепция vtable (виртуальных таблиц), которые формируются для каждого объекта при его создании (то есть когда класс объекта уже известен). Такая таблица содержит список указателей на виртуальные функции. Во время исполнения программы происходит косвенное обращение через соответствующий указатель и сразу выполняется нужный код без всякого поиска.

**Дружественная** (friend) связь предполагает, что класс-клиент имеет право доступа к атрибутам и операциям класса-сервера, которые не являются общими. Это свойство можно задать для ассоциаций, агрегаций, зависимостей и обобщений. В исходный код класса-сервера войдет логика, поддерживающая дружественную видимость для клиента.

**6) Объясните следующие понятия: «стереотип связи», «имя связи» и «элемент связи».**

Как и другим элементам модели, связям разрешается назначать **стереотипы**. Они применяются для классификации связей. Например, Вы используете два типа ассоциаций, им можно задать стереотипы. Стереотипы пишут вдоль линии ассоциации в двойных угловых скобках (<< >>).

С помощью **имен связей** (ролевых имен) уточняют связи. Имя связи - это обычно глагол (глагольная фраза), описывающая, зачем нужна связь.

**Элементом связи** (link element), известным также как класс Ассоциаций (Association class), называется место, где хранятся относящиеся к ассоциации атрибуты.

**7) Опишите основные критерии, используемые для оценки качества классов и объектов.**

1. **Сцепление** является мерой внешней независимости между отдельными модулями. Систему с сильной зависимостью между модулями гораздо сложнее воспринимать и модифицировать. Сложность системы может быть уменьшена путем уменьшения сцепления между отдельными модулями. При объектно-ориентированной разработке, кроме сцепления между модулями, существенную роль играет сцепление между классами и объектами.

2.**Связность** является мерой независимости внутренних частей модуля. Наиболее желательной является функциональная связность, при которой все элементы класса или модуля выполняют одну специфическую функцию. Эта функция может быть представлена набором элементарных функций, однако каждая из них не будет являться самостоятельной с учетом общей выполняемой работы. Так, например, класс Rose (Роза) будет функционально связным, если он описывает только поведение розы и ничего, кроме этого.

3. Под **достаточностью** подразумевается наличие в классе или модуле всего необходимого для реализации логичного и эффективного поведения. Компоненты должны быть полностью готовы к использованию. Допустим, в классе Set (множество) имеется операция удаления элемента из множества, будет ошибкой не включить в него также операцию добавления элемента.

4. Под **полнотой** подразумевается наличие в интерфейсной части класса всех характеристик абстракции. Идея достаточности предъявляет к интерфейсу минимальные требования, а идея полноты охватывает все аспекты абстракции. Полнотой характеризуется такой класс или модуль, интерфейс которого гарантирует все для взаимодействия с пользователями.

5. **Примитивными** являются только такие операции, которые требуют доступа к внутренней реализации абстракции. Так, в примере с классом Set операция Add (добавление к множеству элемента) примитивна, а операция добавления четырех элементов не будет примитивной, так как вполне эффективно реализуется через операцию добавления одного элемента. Операция, которая требует прямого доступа к структуре данных, примитивна по определению.