Контрольные вопросы

26 октября 2022 г.

1 Перечислите и прокомментируйте основные варианты отношений между классами.

Варианты отношений между классами:

- Композиция – отношение (типа часть-целое), при котором объект (класс) и его часть (его член, например, другой класс) удовлетворяют условиям:
  - член является частью объекта;
  - член является частью лишь одного объекта (т.е. частью некоторой сложной сущности);
  - существование члена контролируется объектом: это означает, что объект управляет временем жизни члена таким образом, что пользователю объекта не нужно вмешиваться (обычно при создании композиции создаются и члены, а при её уничтожении – уничтожаются и соответствующие части);
  - член не имеет представления о существовании объекта (unidirectional relationship), который является более сложной сущностью.

  Композиция не характеризует переносимость частей объектов.

- Агрегация – отношение (типа часть-целое), при котором объект и его часть удовлетворяют условиям:
  - член является частью объекта;
  - член может являться частью более, чем одного объекта;
  - существование и продолжительность жизни член не контролируется объектом (или объектами);
  - член не имеет представления о существовании объекта.

- Ассоциация – отношение, при котором:
  - объекты (классы) независимы (т.е. как один объект может являться членом другого, так и другой – членом первого);
  - ассоциированный объект может принадлежать нескольким объектам;
  - существование и продолжительность жизни ассоциированного объекта не контролируется объектом (или объектами);
  - ассоциированный объект может иметь или не иметь представления о существовании объекта (поэтому при ассоциации отношение может быть однонаправленным или двунаправленным – unidirectional or bidirectional relationship).

Иногда объекты могут иметь отношения с другими объектами того же типа. В этом случае говорят о рефлексивной ассоциации (reflexive association).

- Зависимость – отношение, при котором (обычно) один объект (класс) использует другой без хранения связи с последним (это важно, поскольку, например, при ассоциации один класс сохраняет прямую или косвенную связь с ассоциированным классом в качестве члена). Зависимость всегда является однонаправленной.

2 Какие существуют разновидности наследования и для чего они предназначены?

Наследование – создание новых объектов приобретением атрибутов и методов других объектов с их последующим расширением и специализацией (передача чего-либо от родителей потомкам). В C++ наследование возможно между классами. Наследуемый класс называется родителем/базовым (parent class/base class/superclass), наследующий – потомком/производным (child class/derived class/subclass). Существуют две разновидности наследования:

- Одиночное наследование – наследование, при котором каждый наследующий класс (потомок) имеет лишь один наследуемый класс (одного родителя). Одиночное наследование используется, если потомок имеет одного родителей. Зачастую наследование может быть сведено к данному типу наследования.

- Множественное наследование – наследование, при котором наследующий класс (потомок) может иметь более одного наследуемого класса (более одного родителя). Для этой цели могут быть испольхованы т.н. mixin – небольшой класс, от которого можно наследоваться, чтобы добавлять свойства к классу (например, создавать ID для объектов класса).

  Таким образом, множественное наследование используется, если потомок сочетает в себе характеристики нескольких родителей.

  В силу существования опасностей при множественном наследовании его следует избегать (неоднозначность, diamond problem), если альтернативные реализации не приводят к большей сложности.

- Многоуровневое наследование – наследование, при котором наследующий использует другой наследующий класс в качестве родителя. Многоуровневое наследование используется, когда наследующие классы выстраиваются в некоторую цепочку.

- Иерархическое наследование – наследование, при котором родительский класс может являться базовым для нескольких классов.

- Гибридное (виртуальное) наследование – наследование, являющееся комбинацией двух или более типов наследования (перечисленных выше).

3 Что необходимо для корректного функционирования механизма виртуальных функций?

Для корректного функционирования механизма виртуальных функций необходимо:

- Точное совпадение сигнатур функции (имя, типы параметров и константность) производного класса с сигнатурой виртуальной функции базового класса. Тогда можно использовать функцию производного класса. Если функция производного класса имеет другие типы параметров, программа, скорее всего, все равно скомпилируется, но виртуальная функция не будет разрешаться должным образом.

  Данную ошибку бывает сложно обнаружить, поэтому при использовании виртуальных функций добавляют ключевое слово override. Если требуется ограничить дальнейшие переопределения для виртуальных функций, используется ключевое слово final (причём при ограничении функций оно используется после override, но может быть использовано и при объявлении класса – тогда будет запрещено наследование всего данного класса).

- Совпадение при нормальных условиях типов возвращаемого значения виртуальной функции и ее переопределения.

  В особом случае (covariant return types) переопределение виртуальной функции производного класса может иметь тип возвращаемого значения, отличный от типа возвращаемого значения виртуальной функции базового класса, и по-прежнему считаться соответствующим переопределением: если возвращаемый тип виртуальной функции является производным от возвращаемого типа базовой виртуальной функции (например, если возвращаемый тип является указателем или ссылкой на некоторый класс, функции переопределения могут возвращать указатель или ссылку на производный класс).

- Отсутствие вызовов виртуальных функций из конструкторов или деструкторов. Дело в том что при создании наследующего класса сначала создается базовая часть. Если бы виртуальную функцию вызывалась из базового конструктора, а производная часть класса ещё даже не была бы создана, то она не смогла бы вызвать производную версию функции, потому что для работы производной функции нет производного объекта. В C++ вместо этого будет вызываться базовая версия.

  Аналогичная проблема существует для деструкторов. Если вызывается виртуальная функцию в деструкторе базового класса, она всегда будет разрешаться в версию функции базового класса, потому что производная часть класса уже будет уничтожена.

4 Какую проблему решают виртуальные базовые классы при множественном наследовании?

При множественном наследовании виртуальные базовые классы решают т.н. проблему ромба (diamond problem). Эта проблема возникает, когда класс многократно наследуется от двух классов, каждый из которых наследуется от одного базового класса. Это приводит к ромбовидному типу наследования в иерархии классов (что хорошо видно, если нарисовать иерархическую диаграмму наследования классов).

В связи с данной проблемой появлются вопросы: должен ли последний в иерархии классов производный класс иметь одну или две копии исходного базового класса и как разрешать определенные типы неоднозначных ссылок (например, если первые производные классы имели индетичные функции). Большинство этих проблем можно решить с помощью явного определения области видимости, но расходы на обслуживание, добавленные к классам для решения дополнительной сложности, могут привести к резкому увеличению времени разработки. Поэтому виртуальные базовые классы действительно полезны.

При этом за создание виртуального базового класса становится ответственным последний производный класс в иерархии классов. А обращение к членам последнего потомка предшествующими производными классами (к которым они могут обращаться) обычно осуществляется с помощью виртуальной таблицы.

5 Перечислите и прокомментируйте основные категории паттернов проектирования.

Паттерны проектирования (Design Patterns) – решения часто встречающихся проблем в области разработки программного обеспечения. Основные категории паттернов проектирования:

- Порождающие паттерны предназначены для создания новых объектов в системе. При этом они позволяют ей оставаться независимой как от самого процесса порождения, так и от типов порождаемых объектов. При этом предлагаются различные фабрики и фабричные функции, оптимизирующие процесс создания новых объектов.
- Структурные паттерны прендназначены для решения задачи компоновки системы на основе классов и объектов с возможным использованием механизмов:
  - Наследования, когда базовый класс определяет интерфейс, а подклассы – реализацию. Структуры на основе наследования получаются статичными.
  - Композиции, когда структуры строятся путем объединения объектов некоторых классов. Композиция позволяет получать структуры, которые можно изменять во время выполнения.
- Паттерны поведения предназначены для распределения обязанностей между объектами в системе. Для этого могут использоваться механизмы, основанные как на наследовании, так и на композиции.

Литература

[1] https://www.learncpp.com/

[2] https://www.geeksforgeeks.org/inheritance-in-c/

[3] http://cpp-reference.ru