Вопрос №3.4

Инкапсуляция. Абстрактные типы данных. Модули. Полиморфизм. Параметрический и ad-hoc полиморфизм. Наследование. Принцип подстановки Лисков. Полиморфизм подтипов. Объектно-ориентированное программирование.

1 Модули и инкапсуляция

Модуль — структурная единица программной системы. При использовании различных методов разработки ПО в качестве модулей могут выступать подпрограммы, файлы, пакеты, классы.

Появление модулей связано с борьбой со сложностью программ при помощи *декомпозиции*. По мере того, как росла сложность задач, решаемых программами, появились методы разработки, позволяющие разложить задачу на несколько менее сложных подзадач, достаточно независимых, чтобы можно было продолжать работу над каждой из них в отдельности. Решение каждой из таких подзадач выносится в отдельный модуль программной системы.

Хороший метод декомпозиции и модульная структура должны обеспечивать:

- *модульную непрерывность*, когда незначительное изменение спецификации приводит к изменению одного или небольшого количства модулей; для этого небходимо минимизировать взаимодействие между модулями;
- возможность *повторного использования* модулей — использование созданных ранее модулей без изменений в другом контексте; для этого модуль должен быть самостоятельной синтаксической и семантической единицей.

Этим двум целям служит принцип *инкапсуляции* (или по-другому *скрытия информации*). В соответствии с этим принципом, разработчик каждого модуля должен выбрать некоторое подмножество свойств модуля, которые будут доступны его клиентам (это подмножество свойств также называют *интерфейсом модуля*). Остальные свойства модуля будут скрыты от клиентов.

Если изменения спецификации затрагивают только секретные свойства модуля, то они повлияют только на сам модуль, но не на его клиентов. Чем меньше открытая часть модуля, тем меньше вероятность того,

что клиентские модули подвергнутся модификации. Это ведет к непрерывности. Кроме того, минимизация интерфейсов приводит к тому, что модули становятся более независимыми, а значит увеличивается вероятность того, что они смогут работать в другом контексте.

2 Абстрактные типы данных

В объектно-ориентированном программировании распространенным математическим аппаратом для описания объектов являются абстрактные типы данных. Они позволяют описывать объекты, не раскрывая их внутреннего предсталения, в терминах сервисов, предоставляемых этими объектами.

Абстрактный тип данных (АТД) состоит из конечного множества (возможно, частичных) функций (для каждой функции задается ее сигнатура и, если функция является частичной, предусловие — характеристическая функция области определения) и конечного множества аксиом.

Например, абстрактный тип данных STACK (стек) можно определить следующим образом (символ $\nrightarrow$ используется для обозначения частичной функции):

STACK[G]

- Функции:
  - $put\colon STACK[G] \times G \to STACK[G]$
  - $remove\colon STACK[G] \nrightarrow STACK[G]$
  - $item\colon STACK[G] \nrightarrow G$
  - $empty\colon STACK[G] \to BOOLEAN$
  - $new\colon STACK[G]$
- Предусловия:
  - $remove(s : STACK[G])$ require $\neg empty(s)$
  - $item(s : STACK[G])$ require $\neg empty(s)$
- Аксиомы: для всех $x : G$, $s : STACK[G]$
  - $item(put(s, x)) = x$
  - $remove(put(s, x)) = s$
  - $empty(new)$

– $\neg empty(put(s, x))$

АТД является неполной спецификацией объекта. Во-первых, объект может предоставлять и другие сервисы, кроме тех, что соответствуют функциям АТД. Во-вторых, функции могут обладать и другими свойствами, кроме тех, что перечислены в аксиомах.

3 Полиморфизм

Полиморфизм — взаимозаменяемость объектов с одинаковым интерфейсом; (более узкое) способность присоединять к одной переменной объекты различных типов.

В программировании существует множество различных видов и реализаций полиморфизма, например:

- Макросы. Распространена техника управления компиляцией с помощью макросов, когда в зависимости от контекста подставляются подходящие имена функций, выбираются нужные настройки.
- Перегрузка функций. Подходящая реализация функции выбирается в зависимости от типов аргументов.
- Обобщенность и специализация шаблонов. Обобщенный класс может универсально работать с различными типами, пока они предоставляют необходимый интерфейс. Специализация шаблонов позволяет выбрать специальные алгоритмы для работы с конкретными типами.
- Полиморфизм подтипов. В объектно-ориентированном программирвоании переменная некоторого типа может быть присоединена к объекту любого из его подтипов.

Полиморфизм делят на *статический* (конкретный тип полиморфной сущности становится известным на этапе компиляции) и *динамический* (конкретный тип определяется только на этапе выполнения и может меняться в процессе выполнения). Из приведенного выше списка первые три пункта относятся к статическому, а последний — к динамическому полиморфизму.

Ортогональная классификация: *ad-hoc* (ограниченный) и *параметрический* полиморфизм. В первом случае множество различных типов (поведений), которые могут сопоставляться полиморфной сущности, конечно и должно быть явно задано перед использованием. Во втором случае

один и тот же код, написанный без упоминания конкретных типов, корректно и прозрачно работает с потенциально бесконечным множеством типов. Из приведенного выше списка только обобщенность принадлежит к параметрическому типу полиморфизма.

4 Наследование

Наследование — механизм расширения классов, при котором класс добавляет все свои компоненты классам, объявившим его в качестве родителя.

Наследование наряду с клиентским отношением является одним из двух главных отношений между классами в объектно-ориентированном программировании. Возможность специальным образом изменять (переопределять) унаследованные компоненты является решением известной дилеммы «повторно использовать или переделать»: чаще всего существующий класс не полностью удовлетворяет нашим нуждам, однако всегда переписывать все с нуля нецелесообразно.

Наследование часто путают с другим важным понятием объектно-ориентированного программирования — *подтипизацией*. Отношение наследования относится к классам как модулям и является механизмом повторного использования кода на стороне поставщика. Подтипизация — это отношения на классах как типах, оно тоже является механизмом повторного использования, но уже клиентского кода. Общее определение подтипизации называют принципом подстановки Лисков (по фамилии предложившей его Барбары Лисков).

Принцип подстановки Лисков. Тип B является подтипом типа A, если везде, где используются объекты типа A вместо них можно подставить объекты типа B, не нарушая обязательств, данных клиенту.

В терминах классов это означает, что класс B должен предоставлять по крайней мере те же операции, что и класс A, причем сигнатуры и семантика этих операций должна быть совместимой. Типы аргументов операций могут переопределяться только *контравариантно* (типы аргументов в B должны быть супертипами типов аргументов в A), а тип результата — *ковариантно* (тип результата в B должны быть подтипом типа результата в A). Кроме того, класс B имеет право только усиливать инвариант и постусловия операций и только ослаблять предусловия операций.

Подтипизация является поведенческим, а не структурным отношением (типу B необязательно объявлять себя подтипом A). В частности, из-за сложности поиска подтипов на этапе компиляции в практических

объектно-ориентированных языках подтипизацию привязывают к наследованию: класс как тип может быть подтипом другого класса, только если как модуль он наследник этого класса. Однако наследование подтипов — не единственный вид наследования, есть и другие, не порождающие подтипизации.

Поскольку объекты подтипа можно использовать везде, где требуются объекты супертипа, возникает *полиморфизм подтипов*. Клиент может единообразно работать с объектами различных типов, обращаясь с ними как с объектами их общего супертипа. Это главный вид полиморфизма в объектно-ориентированном программирвоании.

5 Объектно-ориентированное программирование

Объектно-ориентированная парадигма создания программного обеспечения характеризируется следующими основными концепциями:

- *Объектная декомпозиция.* Этот вид декомпозиции основан не на вопросе «Что делает система?», а на вопросе «Кто в системе что-нибудь делает?». Поскольку этот аспект системы является более устойчивым, то и результирующая архитектура хорошо приспособлена к эволюции. В результате модули системы рождаются из сущностей, которые, однако, характеризуются не внутренним содержанием, а теми сервисами, которые они предоставляют (такие сущности называются абстрактными типами данных или АТД).

- *Классы.* Класс — это абстрактный тип данных, снабженный полной или частичной реализацией. Классы - единственный вид модулей в объектно-ориентированной системе. Классы общаются между собой через интерфейсы (с точностью до обозначений интерфейс соответствует понятию АТД, на котором основан класс). Интерфейс класса содержит сигнатуры его компонентов: запросов и команд (аналог операций АТД), а также семантические свойства: предусловия запросов и команд (аналог предусловий операций АТД), постусловия запросов и команд и инварианты класса (аналог аксиом АТД).

  В отличие от модулей в структурной парадигме (например, в языках Pascal и Ada), класс одновременно является модулем и типом. Любой тип в объектно-ориентированной программной системе основан на некотором классе.

- *Объекты.* Если класс полностью реализует АТД, то он может иметь экземпляры — объекты. Связь между объектом и классом имеет двоякую природу, как и сам класс: с одной стороны класс - это тип объекта (и понятие типа здесь практически ничем не отличается от понятия типа переменной в не объектно-ориентированных языках), а с другой — класс как модуль определяет те операции, которые применимы к объекту.

  Объект — это единственный вид сущности, существующей в объектно-ориентированной программной системе во время выполнения. Вызов компонента на некотором объекте – это основной вид операции в объектно-ориентированных вычислениях.

- *Наследование, подтипизация и полиморфизм подтипов* (разд. 4).