Л а б о р а т о р н а я р а б о т а № 7-8

«Самостоятельная работа по реализации компьютерной игры с применением паттернов проектирования»

Цель работы: Применение паттернов проектирования.
Продолжительность работы - 8 часа.

Содержание

Как решать задачи проектирования с помощью паттернов

Поиск подходящих объектов

Объектно-ориентированные программы состоят из объектов. *Объект* сочетает данные и процедуры для их обработки. Такие процедуры обычно называют *методами* или *операциями.* Объект выполняет операцию, когда получает *запрос* (или *сообщение)* от *клиента.*

Посылка запроса - это *единственный* способ заставить объект выполнить операцию. А выполнение операции - *единственный* способ изменить внутреннее состояние объекта. Имея в виду эти два ограничения, говорят, что внутреннее состояние объекта

инкапсулировано: к нему нельзя получить непосредственный доступ, то есть представление объекта закрыто от внешней программы.

Самая трудная задача в объектно-ориентированном проектировании - разложить систему на объекты. При ее решении приходится учитывать множество факторов: инкапсуляцию, глубину детализации, наличие зависимостей, гибкость, производительность, развитие, повторное использование и т.д. и т.п. Все это влияет на декомпозицию, причем часто противоречивым образом.

Можно построить модель реального мира или перенести выявленные при анализе системы объекты на свой дизайн. Но нередко появляются и классы, у которых нет прототипов в реальном мире. Это могут быть классы как низкого уровня, например массивы, так и высокого. Если придерживаться строгого моделирования и ориентироваться только на реальный мир, то получится система, отражающая сегодняшние потребности, но, возможно, не учитывающая будущего развития. Абстракции, возникающие в ходе проектирования, - ключ к гибкому дизайну.

Паттерны проектирования помогают выявить не вполне очевидные абстракции и объекты, которые могут их использовать. Например, объектов, представляющих процесс или алгоритм, в действительности нет, но они являются неотъемлемыми составляющими гибкого дизайна. Эти объекты редко появляются во время анализа и даже на ранних стадиях проектирования. Работа с ними начинается позже, при попытках сделать дизайн более гибким и пригодным для повторного использования.

Паттерны позволяют изменять отдельные составляющие системы независимо от прочих. Следствие: система менее подвержена влиянию изменений конкретного вида.

Специфицирование интерфейсов объекта

При объявлении объектом любой операции должны быть заданы: имя операции, объекты, передаваемые в качестве параметров, и значение, возвращаемое операцией. эту триаду называют *сигнатурой* операции. Множество сигнатур всех определенных для объекта операций называется *интерфейсом* этого объекта. Интерфейс описывает все множество запросов, которые можно отправить объекту. Любой запрос, сигнатура которого соответствует интерфейсу объекта, может быть ему послан.

В объектно-ориентированных системах интерфейсы фундаментальны. Об объектах известно только то, что они сообщают о себе через свои интерфейсы. Никакого способа получить информацию об объекте или заставить его что-то сделать в обход интерфейса не существует. Интерфейс объекта ничего не говорит о его реализации; разные объекты вправе реализовывать сходные запросы совершенно по-разному. Это означает, что два объекта с различными реализациями могут иметь одинаковые интерфейсы.

Когда объекту посылается запрос, то операция, которую он будет выполнять, зависит как от запроса, так и от объекта-адресата. Разные объекты, поддерживающие одинаковые интерфейсы, могут выполнять в ответ на такие запросы разные операции. Ассоциация запроса с объектом и одной из его операций во время выполнения называется *динамическим связыванием.*

Динамическое связывание означает, что отправка некоторого запроса не опре-деляет никакой конкретной реализации до момента выполнения. Следовательно, допустимо написать программу, которая ожидает объект с конкретным интерфейсом, точно зная, что любой объект с подходящим интерфейсом сможет принять этот запрос. Более того, динамическое связывание позволяет во время выполнения подставить вместо одного объекта другой, если он имеет точно такой же интерфейс. Такая взаимозаменяемость называется *полиморфизмом* и является важнейшей особенностью объектно-ориентированных систем. Она позволяет клиенту не делать почти никаких предположений об объектах, кроме того, что они поддерживают определенный интерфейс. Полиморфизм упрощает определение клиентов, позволяет отделить объекты друг от друга и дает объектам возможность изменять взаимоотношения во время выполнения.

Паттерны проектирования позволяют определять интерфейсы, задавая их основные элементы и то, какие данные можно передавать через интерфейс. Паттерн может также ≪сказать≫, что не должно проходить через интерфейс. Можно описать, как инкапсулировать и сохранить внутреннее состояние объекта таким образом, чтобы в будущем его можно было восстановить точно в таком же состоянии.

Паттерны проектирования специфицируют также отношения между интерфейсами. В частности, нередко они содержат требование, что некоторые классы должны иметь схожие интерфейсы, а иногда налагают ограничения на интерфейсы классов.

Проектирование с учетом будущих изменений

Системы необходимо проектировать с учетом их дальнейшего развития. Для проектирования системы, устойчивой к таким изменениям, следует предположить, как она будет изменяться на протяжении отведенного ей времени жизни.

Если при проектировании системы не принималась во внимание возможность изменений, то есть вероятность, что в будущем ее придется полностью перепроек-тировать. Перепроектирование отражается на многих частях системы, поэтому непредвиденные изменения всегда оказываются дорогостоящими.

Благодаря паттернам систему всегда можно модифицировать определенным образом. Каждый паттерн позволяет изменять некоторый аспект системы независимо от всех прочих, таким образом, она менее подвержена влиянию изменений конкретного вида. Вот некоторые типичные причины перепроектирования, а также паттерны, которые позволяют этого избежать.

1. При создании объекта явно указывается класс. Задание имени класса привязывает нас к конкретной реализации, а не к конкретному интерфейсу. Это может осложнить изменение объекта в будущем. Чтобы уйти от такой проблемы, нужно создавать объекты косвенно.

Паттерны проектирования: абстрактная фабрика, фабричный метод, прототип.

2. Зависимость от конкретных операций. Задавая конкретную операцию, мы ограничиваем себя единственным способом выполнения запроса. Если не включать запросы в код, то будет проще изменить способ удовлетворения запроса как на этапе компиляции, так и на этапе выполнения.

Паттерны проектирования: цепочка обязанностей, команда.

3. Зависимость от аппаратной и программной платформ.

Внешние интерфейсы операционной системы и интерфейсы прикладных программ (API) различны на разных программных и аппаратных платформах. Если программа зависит от конкретной платформы, ее будет труднее перенести на другие. Поэтому при проектировании систем важно ограничивать платформенные зависимости.

Паттерны проектирования: абстрактная фабрика, мост.

4. Зависимость от представления или реализации объекта. Если клиент “знает”, как объект представлен, хранится или реализован, то при изменении объекта может оказаться необходимым изменить и клиента. Сокрытие этой информации от клиентов поможет уберечься от каскада изменений.

Паттерны проектирования: абстрактная фабрика, мост, хранитель, заместитель.

5. Зависимость от алгоритмов. Во время разработки и последующего использования алгоритмы часто расширяются, оптимизируются и заменяются. Зависящие от алгоритмов объекты придется переписывать при каждом изменении алгоритма. Поэтому алгоритмы, вероятность изменения которых высока, следует изолировать.

Паттерны проектирования: мост, итератор, стратегия, шаблонный метод, посетитель.

6. Сильная связанность. Сильно связанные между собой классы трудно использовать порознь, так как они зависят друг от друга. Сильная связанность приводит к появлению монолитных систем, в которых нельзя ни изменить, ни удалить класс без знания деталей и модификации других классов. Такую систему трудно изучать, переносить на другие платформы и сопровождать.

Слабая связанность повышает вероятность того, что класс можно будет повторно использовать сам по себе. При этом изучение, перенос, модификация и сопровождение системы намного упрощаются. Для поддержки слабо связанных систем в паттернах проектирования применяются такие методы, как абстрактные связи и разбиение на слои.

Паттерны проектирования: абстрактная фабрика, мост, цепочка обязанностей, команда, фасад, посредник, наблюдатель.

7. расширение функциональности за счет порождения подклассов. Специализация объекта путем создания подкласса часто оказывается непростым делом. С каждым новым подклассом связаны фиксированные издержки реализации

(инициализация, очистка и т.д.). Для определения подкласса необходимо также ясно представлять себе устройство родительского класса. Например, для замещения одной операции может потребоваться заместить и другие. Замещение операции может оказаться необходимым для того, чтобы можно было вызвать унаследованную операцию. Кроме того, порождение подклассов ведет к комбинаторному росту числа классов, поскольку даже для реализации простого расширения может понадобиться много новых подклассов.

Композиция объектов и делегирование - гибкие альтернативы наследованию для комбинирования поведений. Приложению можно добавить новую функциональность, меняя способ композиции объектов, а не определяя новые подклассы уже имеющихся классов. С другой стороны, при интенсивном использовании композиции объектов проект может оказаться трудным для понимания. С помощью многих паттернов проектирования удается построить такое решение, где специализация достигается за счет определения одного подкласса и комбинирования его экземпляров с уже существующими.

Паттерны проектирования: мост, цепочка обязанностей, компоновщик,

декоратор, наблюдатель, стратегия.

8. неудобства при изменении классов. Иногда нужно модифицировать класс, но делать это неудобно. Допустим, вам нужен исходный код, а его нет (так обстоит дело с коммерческими библиотеками классов). Или любое изменение тянет за собой модификации множества существующих подклассов. Благодаря паттернам проектирования можно модифицировать классы и при таких условиях.

Паттерны проектирования: адаптер, декоратор, посетитель.

4. Задание

В группах по 4-5 человек разработать игру по выбору: пошаговую стратегию, логигу, «бродилку» с применением нескольких паттернов проектирование в комрлексе. Разработать UML-диаграммы (диаграмму классов и диаграмму последовательности) решаемой задачи.

Результатом лабораторной работы должна быть работающая игра.