Шаблоны проектирования (краткая сводка)

Базовые шаблоны

Делегирование

Некоторая функциональность основного класса выносится в отдельный класс, через ссылку на этот объект осуществляется **делегирование** функционала. **Технологическое решение**, позволяющее «разгрузить» основной класс и сделать программу более модульной.

Интерфейс

Абстрактный суперкласс

Абстрактный класс играет две роли: интерфейс для группы производных классов – многообразие совместимых реализаций + **общий функционал и структуры данных.** Общий функционал использует полиморфные вызовы интерфейсных методов для **отложенного программирования** функционала в производном классе.

Маркерный интерфейс

Интерфейс, не содержащий методов, используется для обозначения наличия у класса какого-либо свойства (например, Serializable – стандартная сериализация). Наличие интерфейса проверяется через instanceof.

Заместитель (Proxy)

Заместитель имеет тот же интерфейс, что и замещаемый класс, а также ссылку или возможность передачи сообщений замещаемому объекту. Изменяет поведение замещаемого объекта в различных аспектах:

- Безопасность, ограничение доступа;
- Удаленный доступ по сети. Класс-заместитель передает сообщение серверу, который создает замещаемый объект и выполняет с ним указанные действия;
- Метод замещаемого объекта выполняется асинхронно в потоке, объект-заместитель завершает метод, не дожидаясь его исполнения в заместителе.
- **Virtual Proxy**. Заместитель не создает замещаемого объекта до тех пор, пока в нем не возникнет реальная необходимость.

Обратный вызов (CallBack)

В процессе делегирования взаимодействие основного объекта и объекта-делегата (целевого объекта) осуществляется через вызов метода во втором объекте в потоке управления первого объекта. Результат этого вызова появляется в момент завершения метода, это может быть синтаксический результат метода, либо результат может быть неявным в виде изменения содержимого объектов-формальных параметров вызова. Такая жесткая схема иногда неудобна по следующим причинам:

- Момент появления результат вызова может быть **асинхронным:** метод запускает поток и завершается, фактические результаты вызова являются результатом работы потока (внутренний параллелизм метода);

- Метод возвращает **результат в виде множества объектов** и параметров. Можно, конечно, создавать специальный класс интегрирующий все результаты, либо использовать несколько объектов-параметров;
- Исключения, порождаемые методом, можно обрабатывать внутри метода, а можно и делегировать. В первом случае появляется дополнительный вариант результата, во втором случае – «выносится сор из избы»;
- Метод возвращает **результат в виде множества однотипных объектов,** в самом методе происходит их накопление, а затем они возвращаются как одно целое, что приводит к большим промежуточным издержкам памяти;
- Метод имеет **множество вариантов** завершения, которые вызывающий код должен обрабатывать.

Во всех случаях в объекте-делегате **прямой вызов метода** может предусматривать обработку результата в контексте вызывающего класса с использованием одного или нескольких методов **обратного вызова,** реализуемых через интерфейс.

Технологически CallBack реализуется на Java следующим образом:

- создается интерфейс, в котором объявляется CallBack-методы с необходимыми параметрами – возвращаемыми данными в обратном вызове;

```
//-------------------------------------------------------------------------------------14_FunctionalExamples
public interface CallBack {
   public void onFinish(String val); }
```

- объект класса, в котором реализуется метод, использующий CallBack, получает в качестве параметра указатель на объект-слушатель с присоединенным интерфейсом обратного вызова. В нем переопределяется CallBack-метод. Для создания объекта-слушателя может использоваться вложенный или анонимный класс, т.к. их объекты «видят» контекст основного класса, в котором производится прямой вызов;

```
//-------------------------------------------------------------------------------------14_FunctionalExamples
// Метод Call с обратным вызовом через интерфейс CallBack
final Slave ss=new Slave();
// Фактический параметр – объект анонимного класса
ss.Call("Call 1 ", new CallBack(){
@Override
public void onFinish(String val) {
  synchronized (ss){      // Синхронизация к основному потоку
     System.out.println(val);
     }
  }
});
// Ссылка на объект анонимного класса (common)
CallBack common=new CallBack(){
@Override
public void onFinish(String val) {
  synchronized (ss){      // Синхронизация к основному потоку
     System.out.println(val);
     }
  }
};
ss.Call("Call 2 ", common);                    // Вызов с объектом-слушателем
ss.Call("Call 3 ", common);
```

- метод прямого вызова в момент получения результата (в собственном потоке управления или в отдельном запускаемом потоке) через ссылку на объект-слушатель производит вызов CallBack-метода.

```
//-----------------------------------------------------------------------------------14_FunctionalExamples
public class Slave {
   public void Call(final String src, final CallBack fin){
      // Поток – анонимный класс
      new Thread(){
         public void run(){
            int wait=(int)(Math.random()*10000);
            try { sleep(wait); } catch(Exception ee){}
            fin.onFinish(src+" "+wait);
                     // Вызов интерфейсного метода в объекте-слушателе
                     // CallBack из потока
            }
         }.start();
   }
}
```

Перечислим ряд технологических решений, для которых может использоваться обратный вызов:

- При обращении к сети непосредственно из GUI могут иметь место существенные задержки, блокирующие клиента, которые внешне воспринимаются как «подвисание» программы. Вызываемый метод, работающий с сетью, запускает поток, в котором выполняется необходимое взаимодействие, по окончании которого производится **асинхронный** обратный вызов. **Замечание.** Необходима, как минимум, синхронизация обратного вызова, выполняемого в отдельном потоке, с основным потоком управления, а если основной поток является потоком GUI, то эта синхронизация делается стандартными методами оконного класса. Кроме того, необходимо заблокировать возможность повторного прямого вызова этого же метода, пока не завершился текущий, а также «разнести» данные, с которыми работают основной поток и поток в вызове;

- Вызываемый метод обращается к БД, от которой получает в цикле последовательности выбранных записей, которые должны обрабатываться вызывающим объектом. В этом случае **каждую запись можно возвращать в виде обратного вызова**, а по завершении цикла выполнять еще один обратный вызов специального **метода завершения** в том же интерфейсе;

- При вызове метода возможны **различные варианты ответа**, которые можно реализовать в виде **группы обратных вызовов** в общем интерфейсе. Тогда вызывающему коду не потребуется их повторная селекция;

- При использовании обратного вызова **обработка исключений** может быть реализована в два этапа – в вызываемом методе, а также с виде **дополнительного метода обратного вызова** (onError);

- Если метод переопределен в группе классов, а в некоторых из них требуется использование **дополнительных параметров,** то эти параметры можно не передавать в общем списке, а запрашивать в нужный момент **методами обратного вызова** из контекста вызывающего объекта. CallBack-интерфейс с методами запроса параметров можно сделать **расширением** базового интерфейса, передавая его объектам только тех классов, которым это необходимо (они, в свою очередь, должны будут выполнять у себя явное сужение до этого интерфейса).

Порождающие шаблоны

Метод – «фабрика» (Factory Method)

Имеется группа родственных классов на основе абстрактного базового или интерфейса. Метод-фабрика создает и возвращает объект одного из классов, руководствуясь своими параметрами (например, по типу файла, передаваемого в имени), либо имеет набор статических методов, возвращающих объекты разных классов.

Абстрактная фабрика (Abstract Factory)

Интерфейс создает подсистему из группы классов с присоединенной группой интерфейсов. Классы конкретных фабрик, присоединяющих этот интерфейс, создают набор объектов конкретных классов, реализующих эту группу интерфейсов.

Строитель (Builder)

Аналогичен «фабрике», только создает не отдельный класс, а систему из основного класса и связанных с ним объектов других классов. Кроме того, в базовом (абстрактном) классе имеется статический метод, который создает объект производного класса, руководствуясь параметрами вызова.

Прототип (Prototype)

Группа родственных классов имеет в интерфейсе метод копирования (клонирования), программа создает и использует копию объекта-прототипа, при этом класс объекта-прототипа и класс создаваемой копии являются неопределенными. В Java на примитивном уровне можно использовать интерфейс clone, средства рефлексии (Class.newInstance).

Синглетон (*Singleton*)

Обеспечивает единственность экземпляра объекта некоторого класса. В отличие от статической ссылки на объект, создается при первом доступе к объекту и закрыт (приватен) для пользователей.

Пул объектов (Object Pool)

Программа имеет ограничение на создание объектов определенного класса (например, соединений с БД, сетевых соединений, открытых файлов). Либо имеются существенные временные затраты на создание (утилизацию) объектов (буферный пул для режима реального времени). Варианты реализации:

- Предварительная генерация объектов, либо по требованию (запросу);
- Хранение только свободных, либо свободных и выделенных;
- Наличие или отсутствие ограничения на размер пула;
- Действие при отсутствии свободного: создание дополнительного объекта или блокировка запрашивающего потока до появления свободного (см. шаблоны параллелизма);
- Действие при освобождении: помещение в буферный пул, либо потеря ссылки (для последующего сбора мусора) при ограничении размера пула.

Структурные шаблоны

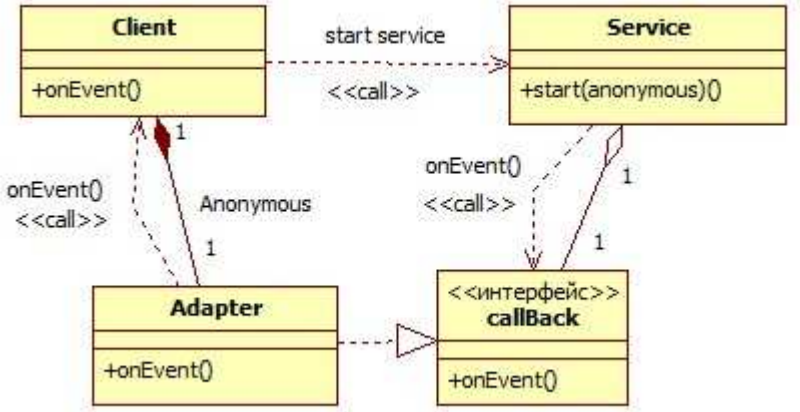
Фильтр (Filter)

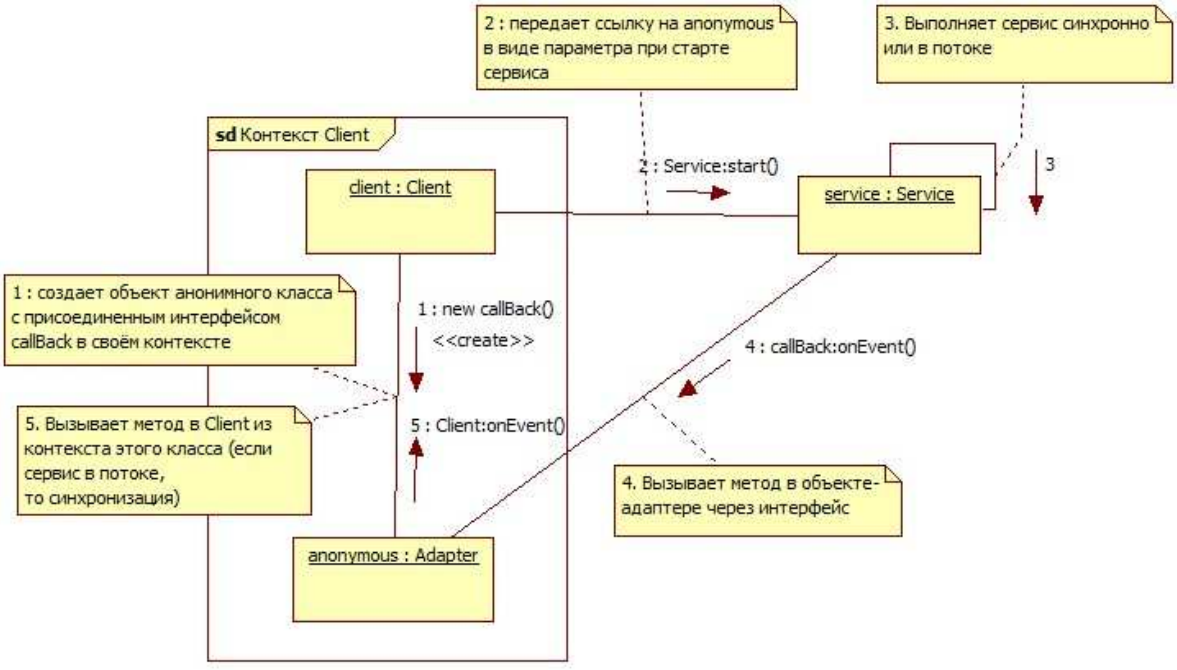
Класс имеет тот же самый интерфейс, что и замещаемый. Фильтр получает данные из замещаемого (делегируемого) класса и производит из преобразование (фильтрацию), возвращая отфильтрованные данные через собственный интерфейс.

Адаптер (Adapter)

Версия 1. Требуется реализовать в классе клиента некоторый интерфейс, например, обратный вызов (callBack) по асинхронному событию. По технологическим соображениям этот интерфейс нельзя навесить на класс-клиент (например, есть несколько источников событий и в самом интерфейсе они не идентифицируются). Тогда создается класс-адаптер, который присоединяет к себе указанный интерфейс, и при этом «видит» класса-клиента. Способы видения могут быть разными:

- Конструктор получает ссылку на объект класса-клиента;
- Класс адаптера является вложенным;
- Класс адаптера является анонимным.





Версия 2. Класс, который обеспечивает реализацию имеющегося интерфейса к классу, имеющему интерфейс, отличный от первого. Класс – адаптер присоединяет к себе первый интерфейс и создает (получает) объект класса со вторым интерфейсом, преобразуя вызовы методов первого в вызовы второго.

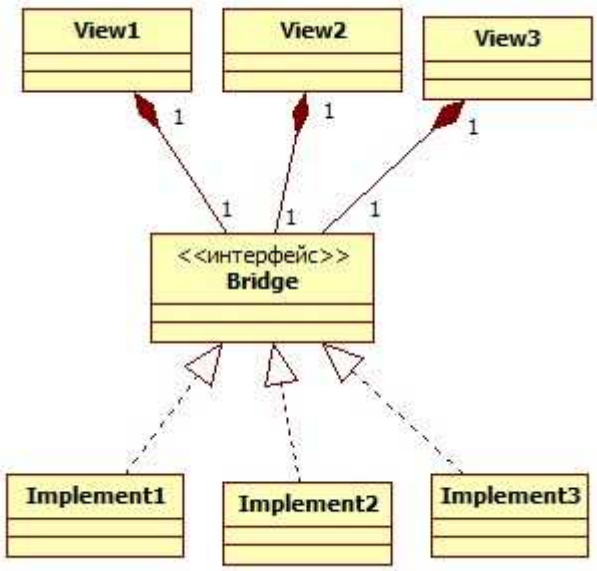
Итератор (*Iterator*) **(или поведенческий)**

Класс-движок по элементам структуры данных, обеспечивающий интерфейс последовательного перемещения по ее элементам (начало, конец, вперед, назад, текущее значение). В реализации используются 4 интерфейса: элемента данных, итератора, структуры данных и callback-вызова набора действий «для каждого». Класс элемента структуры данных (элемента списка) и самого итератора – вложенные в основной класс. Вложенность класса итератора обеспечивает «видимость» им своего родителя – т.е. заголовка списка.

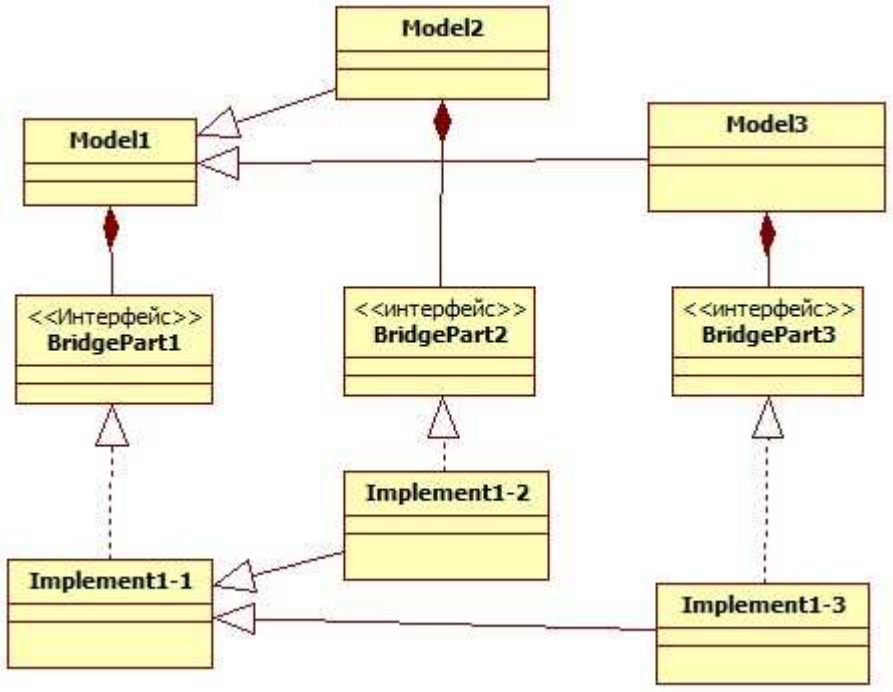
Мост (Bridge)

Разделение многообразия логических и физических представлений на основе общего интерфейса.

Вариант 1. Множество классов логического представления и множество классов физической реализации с промежуточным интерфейсом.



Вариант 2. Группа связанных классов логического уровня и аналогичные группы классов физического уровня (реализаций), связанных по одной схеме. Связывание логического и физического уровней через интерфейсы.



Фасад (*Facade*)

Технологическое решение. Вместо того, чтобы хранить набор ссылок на требуемые объекты, они переносятся в промежуточный класс, обычно оставаясь открытыми. Появляется возможность использовать фасад в других компонентах как единое целое, а не таскать все ссылки. Кроме того, в фасаде можно реализовать методы, данные для которых локализованы в фасаде.

Декоратор (Decorator)

Расширение функциональности исходного класса. Класс-декоратор присоединяет интерфейс исходного класса + имеет собственную функциональность. В отличие от наследования, он получает/создает объект исходного класса, к которому делегирует старую функциональность, а сам дополняет ее.

Динамическое связывание (загрузка, компоновка) (*Dynamic Linkage*)

В Java благодаря рефлексии имеется возможность динамической загрузки классов (ClassLoader). Естественно, загружаемый класс должен иметь оговоренный интерфейс использования, поддерживаемый программой.

Композиция (Composite)

Древовидная (иерархическая) структура объектов. Общий интерфейс Component предполагает несколько частных реализаций однокомпонентных классов, а также класс MultipleComponent, содержащий вектор ссылок на вложенные компоненты, а также методы для манипулирования ими (добавление, удаление).

Приспособленец (Flyweight)

Обеспечивает разделение объектов с одинаковым содержимым путем раздачи ссылок. Изменяемое содержимое объектов выносится в декораторы (внешние объекты). Имеется общий абстрактный класс объектов, на основе которого создаются классы разделяемых и уникальных объектов. Класс Factory, производящий объекты, может создавать уникальные объекты, либо копировать ссылки на разделяемые. Проблема: при манипуляции разделяемым объектом, если его содержимое со стороны использующего его класса меняется, необходимо заменить разделяемый объект на уникальный.

Управление кэшем (Cashe Management)

Класс-менеджер объектов контролирует процесс получения (загрузки) объектов и сохраняет их копии (или ссылки) в кэше. При повторном обращении к тому же объекту по идентификатору он повторно извлекается (копируется) из кэша. Размер кэша ограничен. Стратегии вытеснения – FIFO, LRU.

Объект-значение (Value-Object)

Реализация передачи результата операции в виде объекта-копии (по значению). Значение оригинала при этом не меняется. В Си++ - результат метода в виде объекта-значения с поддержкой конструктора копирования. В Java – возврат ссылки на объект с неизменяемым содержимым (аналогично классу String).

Поведенческие шаблоны

Конечный автомат *(State)*

Модель конечного автомата. Табличная реализация для технологии ООП не подходит, поэтому используется базовый класс состояния и производные классы конкретных состояний. Для событий автомата можно использовать отдельный класс (с доп. параметрами события), а для выполняемых действий callback-интерфейс. Классы состояний автомата – вложенные, чтобы видеть данные родителя – класса-автомата.

Последовательность команд (Command)

Шаблон Command. Группа команд обработки объекта с общим интерфейсом Do/ReDo/UnDo. Производный класс запоминает параметры, необходимые для выполнения прямой и обратной команды. Класс – менеджер команд поддерживает очередь команд ограниченной длины, текущий обрабатываемый объект, методы Do/ReDo/UnDo, выбирая их из очереди и вызывая соответствующие методы в объектах-командах.

Моментальный снимок (SnapShot)

Сохранение существенного текущего содержимого объекта (класса и связанных с ним данных) с возможностью последующего восстановления. Может использоваться сериализация, либо коллекции именованных объектов для сохранения необходимых значений данных (Пример: Android: события для сохранения и восстановления произвольного объекта при повороте экрана, сохранение/восстановление данных в именованной коллекции Bundle при снятии Activity операционной системой).

Null-объект (NullObject)

В группе классов на основе интерфейса создается класс с «нулевой» функциональностью, замещающий null-ссылку. В качестве null-объекта может использоваться и объект базового класса с «нулевой» функциональностью.

Стратегия (Strategy)

Базовый класс реализует структуры данных и базовый функционал их обслуживания. Производные классы реализуют различные варианты решения задачи (алгоритмы, стратегии).

Метод шаблона (Template Method)

Аналог **абстрактного суперкласса.** Абстрактный метод является дополнением основного алгоритма в базовом классе и работает по принципу **отложенного программирования** при реализации производного класса.

Встраиваемый объект (Pluggable Object)

Аналогичен **методу шаблона.** При наличии небольших вариаций функциональности основной класс может получать при конструировании ссылку на объект-делегат, реализующий дополнительную функциональность.

Встраиваемый переключатель (Pluggable Selector)

Аналогичен **методу шаблона,** но при зашитых вариантах функциональности в самом классе (наборе однотипных методов), которые вызываются **по имени с помощью рефлексии.**

Цепочка ответственности (Chain of Responsibility)

Объекты классов с подключенным интерфейсом обработки сообщения (события), образуют линейную или древовидную структуру данных. Основной класс последовательно вызывает метод обработки сообщения во всех объектах в порядке обхода структуры данных. Если один из объектов обработал сообщение, обход прекращается. В моделях систем контроля сообщения также могут передаваться вверх по дереву иерархии от классов-источников (сенсоров) к классам элементов управления. Элемент управления реагирует на сообщение, либо пересылает его выше.

Наблюдатель (Observer)

Средства **широковещательной или селектируемой рассылки** сообщений между объектами группы классов с общим интерфейсом Observer. Объект «подписывается» у класса-диспетчера сообщений (с интерфейсом MultiCast), передавая интерфейс Observer. Класс широковещательной рассылки, получая сообщение, рассылает его всем подписавшимся объектам. Наблюдатель также может быть источником сообщений.

Маленький язык, интерпретатор (Little Language, Interpreter)

Интепретатор произвольного языка с универсальной (настраиваемой) лексикой и синтаксисом.

Посредник (Mediator)

Класс, согласующий состояния связанной с ним группы объектов (разрешения, активность, видимость). Реализует общее поведение, исходя из которого и управляет

состояниями объектов (Пример, форма с изменяющимися разрешениями и видимостью полей в зависимости от результата редактирования других).

Посетитель (Visitor)

Класс, выполняющий обход структуры данных (см. шаблон Композиция) и реализующий некоторый общий алгоритм.

Шаблоны параллелизма (потоковые)

Однопотоковое исполнение *(Single Threaded Execution)*

Синхронизированное исполнение группы действий «друг за другом», как правило, при разделении ими общего ресурса. Возможны различные реализации в зависимости от требования – исполнять планируемые действия в одном потоке, или нет.

1. Блоки synchronized в различных классах с общим объектом синхронизации, либо метод synchronized, вызываемый в одном объекте несколькими потоками. В этом случае не создается общий поток, поэтому каждое действие выполняется в текущем потоке. Недостатки: если синхронизируемое действие занимает много времени, то это тормозит тот поток, в котором оно выполняется, например, поток GUI.
2. Операции, требующие много времени, запускаются в отдельном потоке, параллельно с GUI. См. шаблон Планировщик.

Объект блокировки (Lock Object)

Создается один общий объект блокировки (синглетон) на всю структуру данных вместо множества независимых элементов синхронизации по отдельным данным и операциям над ними. Менее эффективен, но более прост и надежен.

Планировщик (Scheduler)

Действия, инициируемые независимыми потоками, выполняются последовательно, для этого организуется очередь запросов с некоторой процедурой планирования последовательности исполнения (простейшая - FIFO).

Блокировка (Read/Write Lock)

Операция read к ресурсу выполняется без блокировки (параллельно), операция write монополизирует ресурс, синхронизируя все параллельные операции.

Двойная буферизация (Double Buffering)

Стандартная система взаимоотношений «поставщик-потребитель» при отсутствии колебаний производительности. Пока потребитель обрабатывает содержимое одного буфера, поставщик готовит другой. По окончании обработки потребитель меняет буферы и пробуждает поставщика.

Двухфазное завершение (Two-phase Termination)

Главный поток устанавливает признак завершения и уведомляет потоки о начале shutdown. Потоки завершаются асинхронно.

Асинхронная обработка (Asynchronous Task)

Выполнение продолжительных операций в отдельном потоке с асинхронным завершением и callback-интерфейсом, синхронизированным с вызывающим потоком (например, GUI).

Поставщик-потребитель (Producer-Consumer)

Стандартная задача синхронизации потоков генератора и обработчика данных при вариациях производительности. Между поставщиком и потребителем имеет буферный пул (n блоков), в которые поставщик предварительно подгружает данные. Имеется блокировка поставщика по заполнению буферного пула и блокировка потребителя по отсутствию данных в пуле.

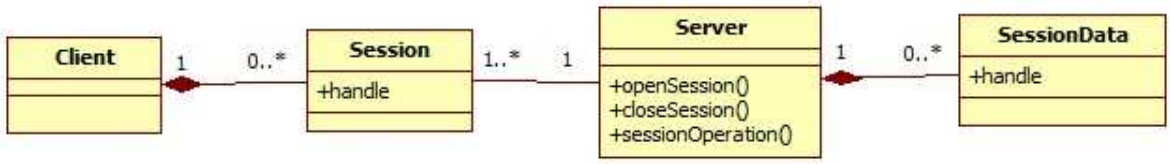
Системные шаблоны

Модель-представление-контроллер (MVC-Model-View-Controller)

Между моделью (представлением данных и внутреннего поведения) и внешним представлением располагается контроллер, связывающий события и отображения в представлении с методами модели.

Сессия (Session)

Позволяет обслуживающему классу/серверу выполнять логически связанную последовательность методов/команд с сохранением в классе/сервере промежуточных данных (например, прав, полученных при авторизации). При выполнении команды/метода авторизации (подключения, открытия) в классе/сервере создается структура данных - внутренний объект сессии, с которым ассоциируется некоторый уникальный идентификатор сессии – handle, возвращаемый классу-пользователю. С помощью handle клиент и сервер идентифицируют сессию и связанные с ней данные. Клиент может использовать класс-представитель сессии, хранящий этот handle.



Подтверждаемое обновление (Successive Update)

Синхронное и **асинхронное** информирование клиента о произошедших изменениях на сервере.

Транзакция (Transaction)

Выполнение последовательности методов в серверном классе единим блоком. При возникновении ошибки во время выполнения одного из методов производится откат к начальному состоянию транзакции. Кроме того, транзакция реализуется как неделимая операция (критическая секция), синхронизируясь с другими транзакциями.

Шаблоны разработки через тестирование

Тестовый метод – эмулятор исключений

Для тестирования обработки исключений вместо исходного метода подставляется метод, генерирующий исключения, аналогично тестируемому.

Тестовый метод – подделка

Метод, подставляемый взамен исходного или отсутствующего, возвращающий заведомо верное значение. Используется при отладке для локализации дефекта, при интеграционном тестировании.

Тестовый метод – эмулятор ресурса

Эмулирует отсутствующий на этапе модульного тестирования ресурс, например, выборку БД или данные из файла.

От одного ко многим

Последовательный рефакторинг кода с тестированием с заменой одиночного объекта на первом шаге на коллекцию (увеличение размерности данных в разрабатываемом классе).

Триангуляция

Использование не менее 2 тестов, корректных на текущем шаге, перед выделением абстракции из разрабатываемого класса (разделением класса на абстрактный базовый и производный от него).

Шаблоны рефакторинга

Перемещение кода в метод (Move Method)

Предпосылка. В классе A имеется кусок кода, в котором много обращений к методам класса B для одного объекта. **Рецепт.** Создать в B новый метод, в который перенести кусок кода, при необходимости добавив в формальные параметры ссылку на объект A, либо его данные.

Выделение метода в отдельный объект (Method Object)

Предпосылка. Имеется объемистый метод с большим количеством формальных параметров. **Рецепт.** Создается отдельный класс, в котором формальные параметры метода становятся внутренними данными класса (задаются в конструкторе), в него же копируется исходный метод, параметры удаляются.

Обратный вызов вместо возврата

Предпосылка. Даже если метод является синхронным, у него может быть много вариантов завершения (например, select, cancel, exception). **Рецепт.** Создать callback-интерфейс и анонимный объект-адаптер для обработки ответа. Объект-адаптер включить в список параметров вызова метода.