GRASP: шаблоны для распределения обязанностей

Аббревиатура GRASP означает General Responsibility Assignment Software Patterns (Общие шаблоны распределения обязанностей в программных системах).

Принципы объектного проектирования отражены в шаблонах проектирования GRASP, изучение и применение которых позволит освоить методический подход и снять завесу таинственности с процесса разработки.

Шаблоны GRASP позволяют понять основные принципы объектного проектирования и методично применять их.

Обязанности и методы

В UML обязанность (responsibility) определяется как "контракт или обязательство классификатора". Обязанности описывают поведение объекта. В общем случае можно выделить два типа обязанностей.

1. *Знание (knowing).*
2. *Действие (doing).*

К знаниям объекта относятся следующие обязанности.

1. *Наличие информации о закрытых инкапсулированных данных.*
2. *Наличие информации о связанных объектах.*
3. *Наличие информации о следствиях или вычисляемых величинах.*

К действиям объекта относятся следующие обязанности.

- *Выполнение некоторых действий самим объектом, например, создание экземпляра или выполнение вычислений.*
- *Инициирование действий других объектов.*
- *Управление действиями других объектов и их координирование.*

Обязанности присваиваются объектам в процессе объектно-ориентированного проектирования. Например, можно сказать, что объект Sale отвечает за создание экземпляра SalesLineItems (действие) или что объект Sale отвечает за наличие информации о стоимости покупки (знание). Обязанности, относящиеся к разряду "знаний", зачастую вытекают из модели предметной области, поскольку она иллюстрирует атрибуты и ассоциации.

Между методами и обязанностями нельзя ставить знак равенства, однако можно утверждать, что реализация метода обеспечивает выполнение обязанностей. Обязанности реализуются посредством методов, действующих либо отдельно, либо во взаимодействии с другими методами и объектами. Например, для класса Sale можно определить один или несколько методов вычисления стоимости (скажем, метод getTotal). Для выполнения этой обязанности объект Sale должен взаимодействовать с другими объектами, в том числе передавать сообщения getSubtotal каждому объекту SalesLineItem о необходимости предоставления соответствующей информации этими объектами.

Из Рис. 0.1 видно, что обязанностью объектов Sale является создание экземпляров Payment. Для выполнения этой обязанности передается сообщение, реализуемое посредством метода makePayment. Более того, для ее выполнения требуется взаимодействие с объектами SalesLineItem и вызов их конструктора.

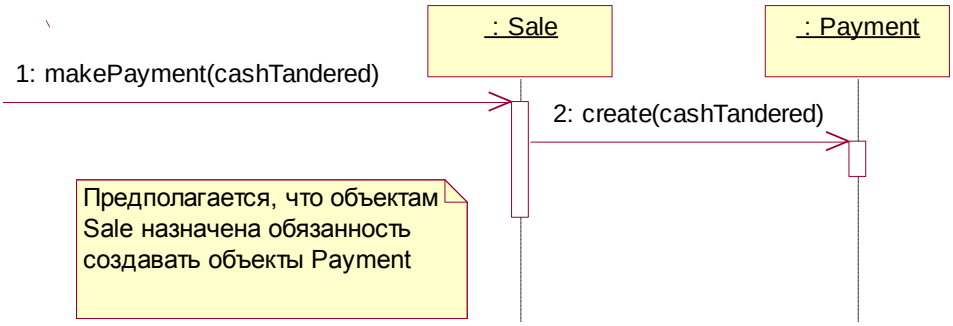

Рис. 0.1 Связь методов и обязанностей

Шаблон Information Expert (информационный эксперт)

Проблема. *Каков наиболее общий принцип распределения обязанностей между объектами при объектно-ориентированном проектировании?*

Решение. *Назначить обязанность информационному эксперту – классу, у которого имеется информация, требуемая для выполнения обязанности.*

В модели системы могут быть определены десятки или сотни программных классов, а в приложении может потребоваться выполнение сотен или тысяч обязанностей. Во время объектно-ориентированного проектирования при формулировке принципов взаимодействия объектов необходимо распределить обязанности между классами. При правильном выполнении этой задачи система становится гораздо проще для понимания, поддержки и расширения. Кроме того, появляется возможность повторного использования уже разработанных компонентов в последующих приложениях.

Пример. Некоторому классу необходимо знать общую сумму продажи.

Распределение обязанностей нужно начинать с их четкой формулировки.

Здесь можно сформулировать следующее утверждение. Какой класс должен отвечать за знание общей суммы продажи?

Согласно шаблону Information Expert, нужно определить, объекты каких классов содержат информацию, необходимую для вычисления общей суммы.

Вероятно, на эту роль подойдет класс Sale (Рис. 0.2).

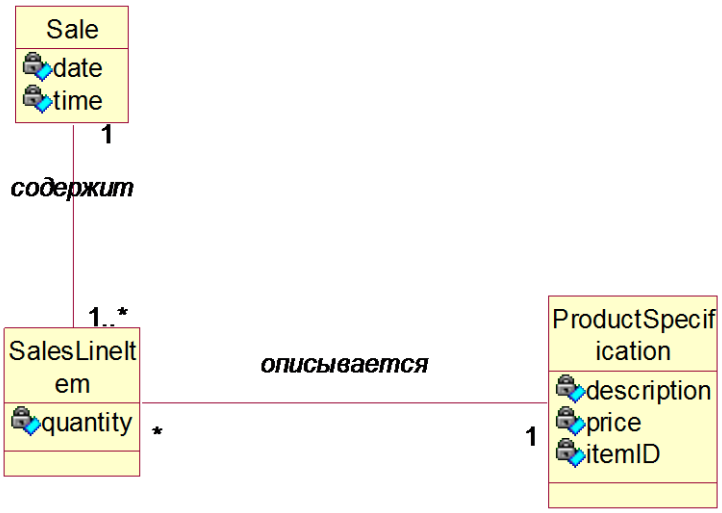

Рис. 0.2. Ассоциации объекта Sale.

Какая информация требуется для вычисления общей суммы? Необходимо узнать стоимость всех проданных товаров SalesLineItem и просуммировать эти промежуточные суммы. Такой информацией обладает лишь экземпляр объекта Sale. Следовательно, с точки зрения шаблона Information Expert объект Sale подходит для выполнения этой

обязанности, т.е. является *информационным экспертом* (information expert). Таким образом, к классу Sale добавляется метод getTotal() (Рис. 0.3).

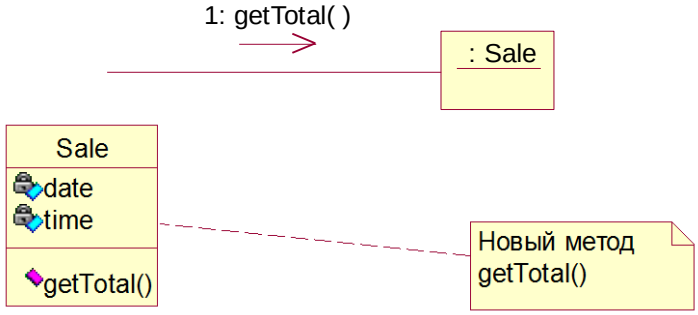

Рис. 0.3. Фрагмент диаграммы взаимодействий

Однако на данном этапе выполнена не вся работа. Какая информация требуется для вычисления промежуточной суммы элементов продажи? Необходимы значения атрибутов SalesLineItem.quantity и SalesLineItem.price. Объекту SalesLineItem известно количество товара и известен связанный с ним объект ProductSpecification. Следовательно, в соответствии с шаблоном Expert, промежуточную сумму должен вычислять объект SalesLineItem. Другими словами, этот объект является *информационным экспертом.*

В терминах диаграмм взаимодействий это означает, что объект Sale должен передать сообщения getSubtotal каждому объекту SalesLineItem, а затем просуммировать полученные результаты,

Для выполнения обязанности, связанной со знанием и предоставлением промежуточной суммы, объекту SalesLineItem должна быть известна стоимость товара.

В данном случае в качестве информационного эксперта будет выступать объект ProductSpecification.

Результаты проектирования представлены на Рис. 0.4.

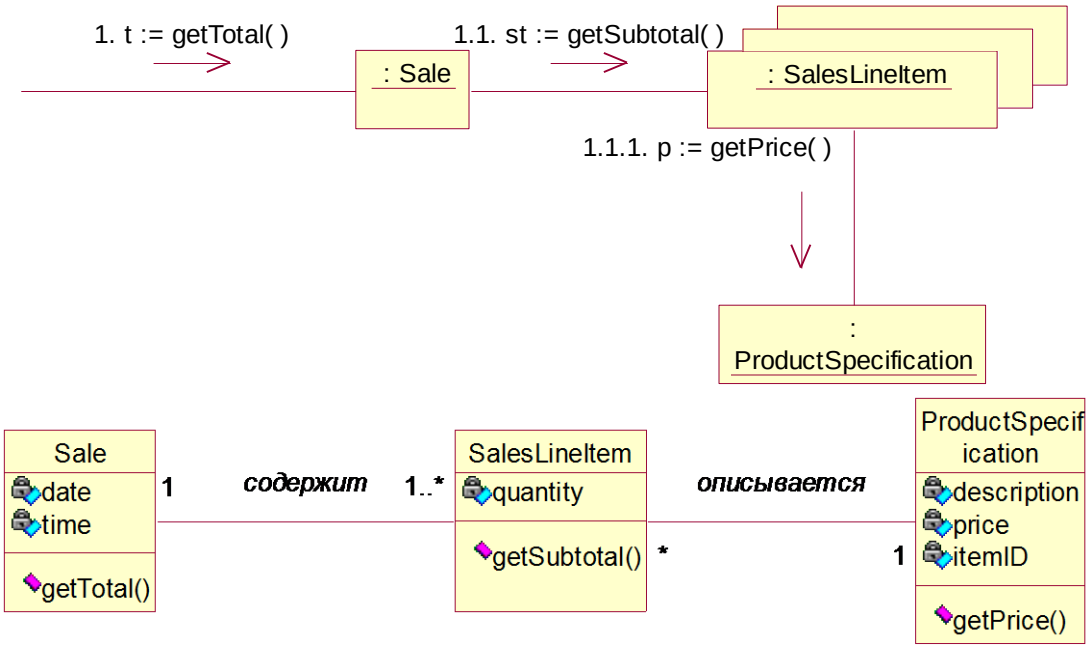

Рис. 0.4. Вычисление общей суммы продажи

В завершение можно сказать следующее. *Для выполнения обязанности "знать и предоставлять общую сумму продажи трем объектам классов" были следующим образом присвоены три обязанности.*

Класс	Обязанность
Sale	Знание общей суммы продажи
SalesLineItem	Знание промежуточной суммы для данного товара
ProductSpecification	Знание цены товара

При распределении обязанностей шаблон Information Expert используется гораздо чаще любого другого шаблона. В нем определены основные принципы, которые уже давно используются в объектно-ориентированном проектировании. Шаблон Expert не содержит неясных или запутанных идей и отражает обычный интуитивно понятный подход. Он заключается в том, что объекты осуществляют действия, связанные с имеющейся у них информацией.

Обратите внимание, что для выполнения обязанности зачастую требуется информация, распределенная между различными классами или объектами. Это предполагает, что должно существовать много "частичных" экспертов, взаимодействующих при выполнении общей задачи. Например, для вычисления общей суммы продажи в конечном счете необходимо взаимодействие трех классов. Если информация распределена между различными объектами, то при выполнении общей задачи они должны взаимодействовать с помощью сообщений.

Преимущества

- *Шаблон Expert поддерживает инкапсуляцию. Для выполнения требуемых задач объекты используют собственные данные*
- *Соответствующее поведение системы обеспечивается несколькими классами, содержащими требуемую информацию. Это приводит к определениям классов, которые гораздо проще понимать и поддерживать.*

Шаблон Creator (создатель)

***Проблема**. Кто должен отвечать за создание нового экземпляра некоторого класса?*

***Решение**. Назначить классу B обязанность создавать экземпляры класса A, если выполняется одно из следующих условий.*

- *Класс B агрегирует объекты A.*
- *Класс B содержит объекты A.*
- *Класс B активно использует объекты A.*
- *Класс B обладает данными инициализации, которые будут передаваться объектам A при их создании (т.е. при создании объектов A класс B является экспертом).*
- *Класс B – создатель объектов A.*

Если выполняется несколько из этих условий, то лучше использовать класс B, агрегирующий или содержащий класс A.

Создание объектов в объектно-ориентированной системе является одним из наиболее стандартных видов деятельности. Следовательно, при назначении обязанностей, связанных с созданием объектов, полезно руководствоваться некоторым основным принципом. Правильно распределив обязанности при проектировании, можно создать слабо связанные независимые компоненты с возможностью их дальнейшего использования, упростить их, а также обеспечить инкапсуляцию данных и их повторное использование.

Пример. Кто должен отвечать за создание нового экземпляра объекта SalesLineItem? В соответствии с шаблоном Creator, необходимо найти класс, агрегирующий, содержащий и т.д. экземпляры объектов SalesLineItem.

Рассмотрим фрагмент модели предметной области, представленной на Рис. 0.2.

Поскольку объект Sale содержит (фактически – агрегирует) несколько объектов SalesLineItem, согласно шаблону Creator, он является хорошим кандидатом для выполнения обязанности, связанной с созданием экземпляров объектов SalesLineItem.

Такой подход приводит к необходимости разработки взаимодействия объектов, показанного на следующей диаграмме (Рис. 0.5).

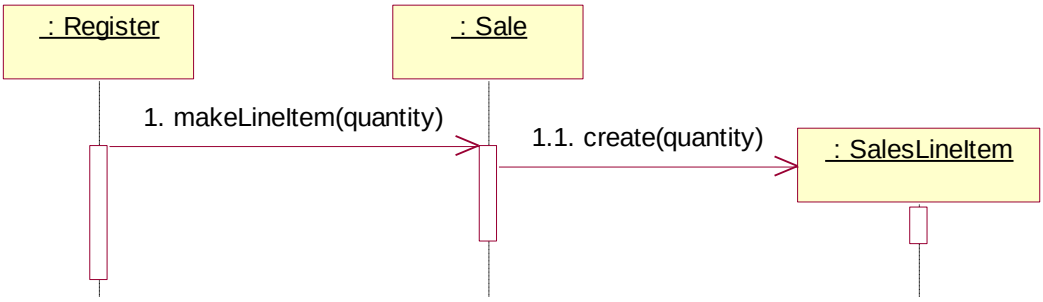

Рис. 0.5. Создание экземпляра объекта SalesLineItem

При таком распределении обязанностей требуется, чтобы в объекте Sale был определен метод makeLineItem.

Рассмотрение и распределение обязанностей выполнялись в процессе создания диаграммы взаимодействий. Затем полученные результаты могут быть реализованы в конкретных методах, помещенных в разделе методов диаграммы классов.

Шаблон Creator определяет способ распределения обязанностей, связанный с процессом создания объектов. В объектно-ориентированных системах эта задача является одной из наиболее распространенных. Основным назначением шаблона Creator является выявление объекта-создателя, который при возникновении любого события должен быть связан со всеми созданными им объектами. При таком подходе обеспечивается низкая степень связанности.

В некоторых случаях в качестве создателя выбирается класс, который содержит данные инициализации, передаваемые объекту во время его создания. На самом деле это пример использования шаблона Expert. В процессе создания инициализирующие данные передаются с помощью метода инициализации некоторого вида, такого как конструктор языка Java с параметрами. Например, предположим, что при создании экземпляр объекта Payment нужно инициализировать с использованием общей суммы, содержащейся в объекте Sale. Поскольку объекту Sale эта сумма известна, он является кандидатом на выполнение обязанности, связанной с созданием экземпляра объекта Payment.

Преимущества

Применение шаблона Creator не повышает степени связанности, поскольку созданный (created) класс, как правило, оказывается видимым для класса-создателя посредством имеющихся ассоциаций.

Шаблон Low Coupling (слабое связывание)

***Проблема**. Как обеспечить зависимость, незначительное влияние изменений и повысить возможность повторного использования?*

***Решение**. Распределить обязанности таким образом, чтобы степень связанности оставалась низкой.*

Степень связанности (coupling) – это мера, определяющая насколько жестко один элемент связан с другими элементами, либо каким количеством данных о других элементах он обладает. *Элемент с низкой степенью связанности (или слабым связыванием) зависит от не очень большого числа других элементов. Выражение "очень много" зависит от контекста, однако необходимо провести его оценку.*

Класс с высокой степенью связанности (или жестко связанный) зависит от множества других классов. Однако наличие таких классов нежелательно, поскольку оно приводит к возникновению следующих проблем.

- *Изменения в связанных классах приводят к локальным изменениям в данном классе.*
- *Затрудняется понимание каждого класса в отдельности.*
- *Усложняется повторное использование, поскольку для этого требуется дополнительный анализ классов, с которыми связан данный класс.*

Пример. Рассмотрим следующий фрагмент диаграммы классов.

Предположим, что необходимо создать экземпляр класса Payment и связать его с объектом Sale. Какой класс должен отвечать за выполнение этой операции? Поскольку в реальной предметной области регистрация объекта Payment выполняется объектом Register, в соответствии с шаблоном Creator, объект Register является хорошим кандидатом для создания объекта Payment. Затем экземпляр объекта Register должен передать сообщение addPayment объекту Sale, указав в качестве параметра новый объект Payment. Приведенные рассуждения отражены на фрагменте диаграммы взаимодействий, представленной на Рис. 0.6.

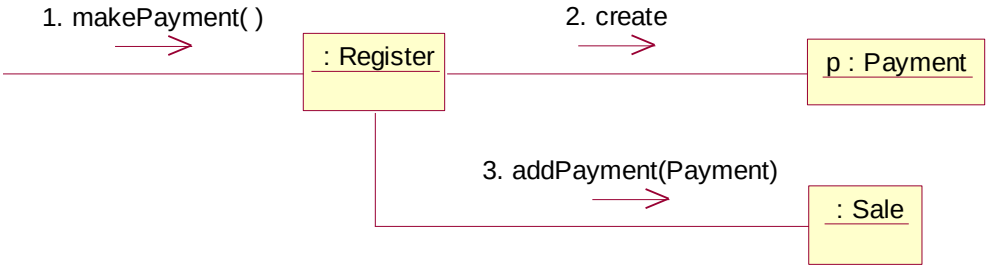

Рис. 0.6. Новый экземпляр объекта Payment создается с помощью объекта Register

Такое распределение обязанностей предполагает, что класс Register обладает знаниями о данных класса Payment (т.е. связывается с ним).

Альтернативный способ создания объекта Payment и его связывания с объектом Sale показан на Рис. 0.7.

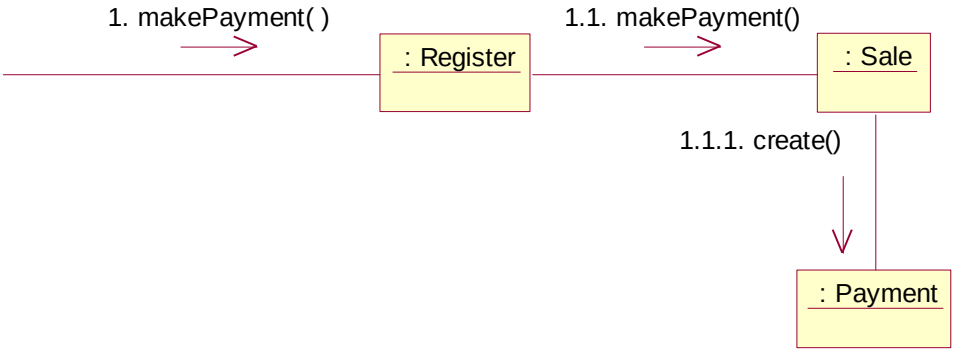

Рис. 0.7. Новый экземпляр объекта Payment создается с помощью объекта Sale

Какой из методов проектирования, основанный на распределении обязанностей, обеспечивает более низкую степень связывания? *В обоих случаях предполагается, что в конечном итоге объекту Sale должно быть известно о существовании объекта Payment. При использовании первого способа, когда объект Payment создается с помощью объекта Register, между этими двумя объектами добавляется новая связь, тогда как второй способ степень связывания объектов не усиливает. С точки зрения числа связей между объектами, более предпочтительным является второй способ, поскольку в этом случае обеспечивается низкая степень связывания. Приведенная иллюстрация является примером того, как при использовании двух различных шаблонов – Low Coupling и Creator – можно прийти к двум различным решениям.*

В шаблоне Low Coupling описывается принцип, о котором нельзя забывать на протяжении всех стадий работы над проектом. Он является объектом постоянного внимания. Шаблон Low Coupling представляет собой средство, которое разработчик применяет при оценке всех принимаемых в процессе проектирования решений.

Шаблон Low Coupling поддерживает независимость классов, что, в свою очередь, повышает возможности повторного использования и обеспечивает более высокую эффективность приложения. Его нельзя рассматривать изолированно от других шаблонов, таких как Expert и High Cohesion. Скорее, он обеспечивает один из основных принципов проектирования, применяемых при распределении обязанностей.

Подкласс жестко связан со своим суперклассом. Поэтому, принимая решение о наследовании свойств объектов, следует учитывать, что отношение наследования повышает степень связанности классов.

Не существует абсолютной меры для определения слишком высокой степени связывания. Важно лишь понимать степень связанности объектов на текущий момент и не упустить тот момент, когда дальнейшее повышение степени связанности может привести к возникновению проблем. В целом, следует руководствоваться таким принципом: классы, которые являются достаточно общими по своей природе и с высокой вероятностью будут повторно использоваться в дальнейшем, должны иметь минимальную степень связанности с другими классами.

Крайним случаем при реализации шаблона Low Coupling является полное отсутствие связывания между классами. Такая ситуация тоже нежелательна, поскольку базовой идеей объектного подхода является система связанных объектов, которые "общаются" между собой посредством передачи сообщений. При слишком частом использовании принципа слабого связывания система будет состоять из нескольких изолированных сложных активных объектов, самостоятельно выполняющих все операции, и множества пассивных объектов, основная функция которых сводится к хранению данных. Поэтому при создании объектно-ориентированной системы должна присутствовать некоторая оптимальная степень связывания между объектами, позволяющая выполнять основные функции посредством взаимодействия этих объектов.

Преимущества

- *Изменения компонентов мало сказываются на других объектах.*
- *Принципы работы и функции компонентов можно понять, не изучая другие объекты.*
- *Удобство повторного использования.*

Шаблон High Cohesion (высокое зацепление)

***Проблема**. Как обеспечить возможность управления сложностью?*

***Решение**. Распределение обязанностей, поддерживающее высокую степень зацепления.*

***Зацепление (функциональное зацепление) – это мера связанности и сфокусированности обязанностей класса**. Считается, что элемент обладает высокой степенью зацепления, если его обязанности тесно связаны между собой и он не выполняет непомерных объемов работы. В роли таких элементов могут выступать классы, подсистемы и т.д.*

Класс с низкой степенью зацепления выполняет много разнородных функций или несвязанных между собой обязанностей. Такие классы создавать нежелательно, поскольку они приводят к возникновению следующих проблем.

- *Трудность понимания.*
- *Сложности при повторном использовании.*
- *Сложности поддержки.*
- *Ненадежность, постоянная подверженность изменениям.*

Классы со слабым зацеплением, как правило, являются слишком "абстрактными" или выполняют обязанности, которые можно легко распределить между другими объектами.

Пример. Предположим, необходимо создать экземпляр объекта Payment и связать его с текущей продажей. Какой класс должен выполнять эту обязанность? Поскольку в реальной предметной области сведения о платежах записываются в реестре, согласно шаблону Creator, для создания экземпляра объекта Payment можно использовать объект Register. Тогда экземпляр объекта Register сможет отправить сообщение addPayment объекту Sale, передавая в качестве параметра новый экземпляр объекта Payment, как показано на Рис. 0.8.

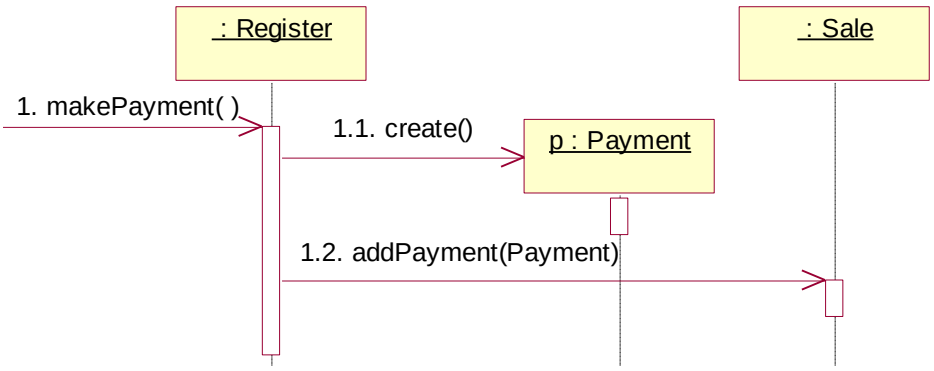

Рис. 0.8. Экземпляр объекта Register создает объект Payment

При таком распределении обязанностей платежи выполняет объект Register, т.е. объект Register частично несет ответственность за выполнение системной операции makePayment.

В данном обособленном примере это приемлемо. Однако если и далее возлагать на класс Register обязанности по выполнению все новых и новых функций, связанных с другими системными операциями, то этот класс будет слишком перегружен и будет обладать низкой степенью зацепления.

Предположим, приложение должно выполнять пятьдесят системных операций и все они возложены на класс Register. Если этот объект будет выполнять все операции, то

он станет чрезмерно "раздутым" и не будет обладать свойством зацепления. И дело не в том, что одна задача создания экземпляра объекта Payment сама по себе снизила степень зацепления объекта Register; она является частью общей картины распределения обязанностей.

На Рис. 0.9 представлен другой вариант распределения обязанностей. *Здесь функция создания экземпляра платежа делегирована объекту Sale. Благодаря этому поддерживается более высокая степень зацепления объекта Register. Поскольку такой вариант распределения обязанностей обеспечивает низкий уровень связывания и более высокую степень зацепления, он является более предпочтительным.*

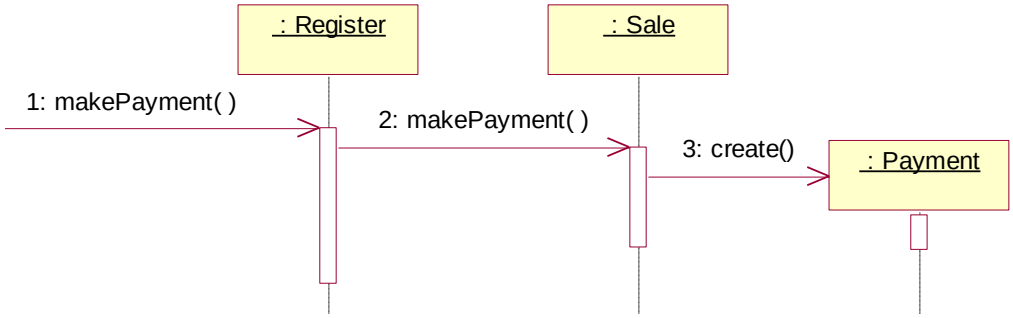

Рис. 0.9. Объект Sale создает экземпляр объекта Payment

На практике уровень зацепления не рассматривают изолированно от других обязанностей и принципов, обеспечиваемых шаблонами Expert и Low Coupling.

Как и о принципе слабого связывания, о высокой степени зацепления следует помнить в течение всего процесса проектирования. Этот шаблон необходимо применять при оценке эффективности каждого проектного решения.

Следующие сценарии, иллюстрируют различную степень функционального зацепления.

1. *Очень слабое зацепление. Только один класс отвечает за выполнение множества операций в самых различных функциональных областях.*
2. *Слабое зацепление. Класс несет единоличную ответственность за выполнение сложной задачи из одной функциональной области.*
3. *Среднее зацепление. Класс имеет среднее количество обязанностей из одной функциональной области и для выполнения своих задач взаимодействует с другими классами.*
4. *Сильное зацепление. Класс имеет несложные обязанности в нескольких различных областях, логически связанных с концепцией этого класса, но не связанных между собой. Пример. Существует класс Company, который несет полную ответственность за (а) знание всех сотрудников компании и (б) всю финансовую информацию. Эти две области не слишком связаны между собой, однако обе логически связаны с понятием "компания". К тому же предполагается, что такой класс содержит небольшое число открытых методов и требует незначительных объемов кода для их реализации.*

Как правило, класс с высокой степенью зацепления содержит сравнительно небольшое число методов, которые функционально тесно связаны между собой, и не выполняет слишком много функций. Он взаимодействует с другими объектами для выполнения более сложных задач.

Преимущества

- *Повышаются ясность и простота проектных решений.*
- *Упрощаются поддержка и доработка.*
- *Зачастую обеспечивается слабое связывание.*
- *Улучшаются возможности повторного использования, поскольку класс с высокой степенью зацепления выполняет конкретную задачу.*