

**Игорь Николаев**, финансовый директор группы компаний «Интернест»

Финансовая модель компании – набор взаимоувязанных показателей, характеризующих ее деятельность. Цель такой модели – мгновенно просчитать, как повлияет изменение экономической ситуации в стране или показателей компании на финансовое состояние и результаты бизнеса.

Все показатели модели можно условно разбить на группы: имущество или активы, обязательства или пассивы, денежные потоки или поступления и платежи денежных средств, доходы и расходы. Все результаты расчетов в модели представим в привычной форме – баланс, отчет о прибылях и убытках и отчет о движении денежных средств по состоянию на произвольную дату в будущем.

Если предстоит создать модель с нуля, руководствуйтесь принципом «от простого к сложному», иначе есть риск запутаться в большом количестве формул и ссылок. Сначала сформируйте простейшую модель с минимальным количеством элементов. Установить связи общего характера между внешними параметрами (спрос на продукцию, стоимость ресурсов) и внутренними показателями деятельности компании (выручка, затраты, денежные потоки и т. д.). В первой итерации можно не заботиться об особой точности. На этом этапе важнее установить правильные взаимосвязи между переменными так, чтобы модель автоматически пересчитывалась после изменения исходных данных и позволяла выстраивать различные сценарии. Уже после этого можно приступить к ее развитию, детализовать показатели, ввести дополнительные уровни аналитики и т. д.

**Как спланировать доходы в финансовой модели**

Как создать финансовую модель, описывающую [операционную деятельность](https://www.1fd.ru/#/document/175/47/) предприятия в Excel, посмотрим на примере компании, занимающейся оказанием услуг.

**Пример финансовой модели**

Начнем финансовую модель с внешних параметров (скачать [модель из примера](https://www.1fd.ru/)). Отправной точкой для дальнейших расчетов послужит план продаж, в примере – оказания услуг. Для этого в Excel на одном из листов книги разместим таблицу с планом продаж в денежном выражении (см. таблицу 1. [План продаж](https://www.1fd.ru/#/document/173/422/qwert111/)). На этом этапе выручку можно указать «навскидку» или использовать данные прошлого года. Пока точность не имеет большого значения. Позднее при детализации модели план продаж придется доработать.

Таблица 1. План продаж, тыс. руб.

| **Показатель** | **Янв.** | **Февр.** | **Март** | **Апр.** | **Май** | **Июнь** | **Июль** | **Авг.** | **Сент.** | **Окт.** | **Нояб.** | **Дек.** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Показатель** | **Янв.** | **Февр.** | **Март** | **Апр.** | **Май** | **Июнь** | **Июль** | **Авг.** | **Сент.** | **Окт.** | **Нояб.** | **Дек.** |
| Услуга 1 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 |
| Услуга 2 | 30 | 33 | 35 | 38 | 40 | 43 | 45 | 48 | 50 | 53 | 55 | 58 |
| Услуга 3 | 65 | 70 | 75 | 80 | 85 | 90 | 95 | 100 | 105 | 110 | 115 | 120 |
| Итого | 110 | 119 | 127 | 136 | 144 | 153 | 161 | 170 | 178 | 187 | 195 | 204 |

**Как планировать расходы в финансовой модели**

Исходя из объема продаж определите размер переменных затрат. В общем виде он равен произведению доли выручки и объема продаж.

Сделаем небольшое допущение и предположим, что в примере переменные только затраты на оплату труда. Заработная плата сотрудников полностью зависит от объема оказанных услуг, на нее уходит примерно 30 процентов выручки от реализации. Кстати, план затрат удобнее разместить на отдельном листе Excel (см. таблицу 2. [План затрат](https://www.1fd.ru/#/document/173/422/qwert40/)). В нем зарплата рассчитывается помесячно как произведение коэффициента 0,3 (30% : 100%) и плана продаж на определенный месяц. Расходы на аренду и управление вводятся на первом этапе создания модели не как расчетные величины, а как фиксированные значения. В дальнейшем при детализации модели их можно будет заменить формулами, увязав с другими показателями.

Таблица 2. План затрат, тыс. руб.

[**скачать.xls**](https://www.1fd.ru/)

| **Показатель** | **Янв.** | **Февр.** | **Март** | **Апр.** | **Май** | **Июнь** | **Июль** | **Авг.** | **Сент.** | **Окт.** | **Нояб.** | **Дек.** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Показатель** | **Янв.** | **Февр.** | **Март** | **Апр.** | **Май** | **Июнь** | **Июль** | **Авг.** | **Сент.** | **Окт.** | **Нояб.** | **Дек.** |
| Зарплата | 33 | 36 | 38 | 41 | 43 | 46 | 48 | 51 | 53 | 56 | 59 | 61 |
| Аренда | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 |
| Управленческие расходы | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 |
| Итого | 113 | 116 | 118 | 121 | 123 | 126 | 128 | 131 | 133 | 136 | 139 | 141 |

**Как сформировать сводные планы в финансовой модели**

Как только в модели сформируете два функциональных плана – продаж и затрат, создавайте баланс, план доходов и расходов, а также план движения денежных средств. Данные в них допустимо вносить двумя способами: рассчитать или извлечь напрямую из операционных или функциональных планов – продаж, затрат и других, например дебиторской задолженности, запасов, производства и др. В Excel установите ссылки на соответствующую ячейку и страницу. Ручной ввод цифр тут недопустим за исключением данных о начальных остатках в балансе, иначе модель просто не будет пересчитывать данные.

Не стоит перегружать прогнозный баланс, бюджет доходов и расходов, движения денежных средств показателями. Лучше стремиться к тому, чтобы каждый из них мог уместиться на одном печатном листе. Зачастую трудно удержаться от соблазна расшифровать каждую цифру. Например, в плане доходов и расходов расписать выручку по видам продукции, группам клиентов, каналам сбыта и т. п.. Если в план доходов и расходов включить сотню видов готовой продукции и статей затрат, это значительно затруднит его восприятие. Тем не менее с точки зрения информативности полезно подобные планы дополнять различными относительными показателями. Например, в баланс внести показатели структуры активов и пассивов, в план доходов и расходов – рентабельность.

В нашем примере три листа, на которых разместим по отдельности план доходов и расходов (ПДР), план движения денежных средств (ПДДС) и прогнозный баланс (ПБ). В плане доходов и расходов (см. таблицу 3. [План доходов и расходов](https://www.1fd.ru/#/document/173/422/qwert47/)) строки «Операционные расходы» и «Операционные доходы» заполните при помощи ссылок на соответствующие ячейки функциональных планов. Выручку расшифруйте по видам услуг, затраты – по статьям. Такая расшифровка допустима, поскольку не утяжеляет восприятие отчета и не усложняет его анализ. Кроме того, в отчет включим два аналитических показателя – рентабельность (как отношение прибыли к выручке) и прибыль нарастающим итогом. Если понадобится провести более глубокий анализ, в частности, динамики доли оплаты труда в себестоимости услуг, все необходимые для этого расчеты лучше проводить на отдельном листе.

Таблица 3. План доходов и расходов, тыс. руб.

[**скачать.xls**](https://www.1fd.ru/)

| **Показатели** | **Янв.** | **Февр.** | **Март** | **Апр.** | **Май** | **Июнь** | **Июль** | **Авг.** | **Сент.** | **Окт.** | **Нояб.** | **Дек.** | **Янв.** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Показатели** | **Янв.** | **Февр.** | **Март** | **Апр.** | **Май** | **Июнь** | **Июль** | **Авг.** | **Сент.** | **Окт.** | **Нояб.** | **Дек.** | **Янв.** |
| Операционные доходы, в т. ч. | 110 | 119 | 127 | 136 | 144 | 153 | 161 | 170 | 178 | 187 | 195 | 204 | 1881 |
| услуга 1 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 246 |
| услуга 2 | 30 | 33 | 35 | 38 | 40 | 43 | 45 | 48 | 50 | 53 | 55 | 58 | 525 |
| услуга 3 | 65 | 70 | 75 | 80 | 85 | 90 | 95 | 100 | 105 | 110 | 115 | 120 | 1110 |
| Операционные расходы, в т. ч. | 113 | 116 | 118 | 121 | 123 | 126 | 128 | 131 | 133 | 136 | 139 | 141 | 1524 |
| зарплата | 33 | 36 | 38 | 41 | 43 | 46 | 48 | 51 | 53 | 56 | 59 | 61 | 564 |
| аренда | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 360 |
| управленческие расходы | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 600 |
| Операционная прибыль | -3 | 3 | 9 | 15 | 21 | 27 | 33 | 39 | 45 | 51 | 57 | 62 | 357 |
| Рентабельность, % | -2,73 | 2,49 | 7,01 | 10,96 | 14,44 | 17,54 | 20,31 | 22,80 | 25,06 | 27,10 | 28,97 | 30,69 | 18,96 |
| Прибыль нарастающим итогом | -3 | 0 | 9 | 24 | 45 | 71 | 104 | 143 | 187 | 238 | 294 | 357 | 713 |

План движения денежных средств (см. таблицу 4. [План движения денежных средств](https://www.1fd.ru/#/document/173/422/qwert52/)) в нашем примере формируем со следующими допущениями. Первое: разделы «[Финансовая деятельность](https://www.1fd.ru/#/document/175/49/)» и «[Инвестиционная деятельность](https://www.1fd.ru/#/document/175/48/)» исключаем из плана. Предполагаем, что компания ведет только операционную деятельность, без привлечения заемных средств и без капитальных вложений. Еще одно допущение. Компания оказывает услуги физическим лицам за наличный расчет, а значит, время оказания услуги и ее оплаты совпадает – в итоге у компании нет дебиторской задолженности. Ситуация с платежами по операционной деятельности не так однозначна. Зарплата и аренда выплачиваются в месяце, следующем за месяцем начисления, а управленческие расходы – в месяце их осуществления.

Таблица 4. План движения денежных средств, тыс. руб.

[**скачать.xls**](https://www.1fd.ru/)

| **Показатели** | **Янв.** | **Февр.** | **Март** | **Апр.** | **Май** | **Июнь** | **Июль** | **Авг.** | **Сент.** | **Окт.** | **Нояб.** | **Дек.** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Показатели** | **Янв.** | **Февр.** | **Март** | **Апр.** | **Май** | **Июнь** | **Июль** | **Авг.** | **Сент.** | **Окт.** | **Нояб.** | **Дек.** |
| Поступления по операционной деятельности | 110 | 119 | 127 | 136 | 144 | 153 | 161 | 170 | 178 | 187 | 195 | 204 |
| Оплата услуг | 110 | 119 | 127 | 136 | 144 | 153 | 161 | 170 | 178 | 187 | 195 | 204 |
| Платежи по операционной деятельности | 50 | 113 | 116 | 118 | 121 | 123 | 126 | 128 | 131 | 133 | 136 | 139 |
| Зарплата | 0 | 33 | 36 | 38 | 41 | 43 | 46 | 48 | 51 | 53 | 56 | 59 |
| Аренда | 0 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | 30 |
| Управленческие расходы | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 |
| Сальдо по операционной деятельности | 60 | 6 | 11 | 17 | 23 | 29 | 35 | 41 | 47 | 53 | 59 | 65 |
| Сальдо на начало периода | 10 | 70 | 76 | 87 | 104 | 128 | 157 | 192 | 233 | 281 | 334 | 393 |
| Сальдо на конец периода | 70 | 76 | 87 | 104 | 128 | 157 | 192 | 233 | 281 | 334 | 393 | 458 |

Последнее, что остается сделать, – создать прогнозный баланс (см. таблицу 5. [Прогнозный баланс](https://www.1fd.ru/#/document/173/422/qwert57/)). Данные по оборотам за период берем из плана доходов и расходов, бюджета движения денежных средств, начальные остатки – из баланса за предыдущий период. Здесь допустимо вручную внести цифры.

Таблица 5. Прогнозный баланс, тыс. руб.

[**скачать.xls**](https://www.1fd.ru/)

| **Показатели** | **01.01** | **01.02** | **01.03** | **01.04** | **01.05** | **01.06** | **01.07** | **01.08** | **01.09** | **01.10** | **01.11** | **01.12** | **01.01** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Показатели** | **01.01** | **01.02** | **01.03** | **01.04** | **01.05** | **01.06** | **01.07** | **01.08** | **01.09** | **01.10** | **01.11** | **01.12** | **01.01** |
| Денежные средства | 10 | 70 | 76 | 87 | 104 | 128 | 157 | 192 | 233 | 281 | 334 | 393 | 458 |
| Дебиторская задолженность | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Основные средства | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| АКТИВЫ итого | 10 | 70 | 76 | 87 | 104 | 128 | 157 | 192 | 233 | 281 | 334 | 393 | 458 |
| Кредиторская задолженность | 0 | 63 | 66 | 68 | 71 | 73 | 76 | 78 | 81 | 83 | 86 | 89 | 91 |
| Капитал | 10 | 7 | 10 | 19 | 34 | 55 | 81 | 114 | 153 | 197 | 248 | 304 | 367 |
| ПАССИВЫ итого | 10 | 70 | 76 | 87 | 104 | 128 | 157 | 192 | 233 | 281 | 334 | 393 | 458 |

Остановимся на формировании некоторых строк баланса несколько подробнее и предварительно оговоримся, что в компании остаток денежных средств составляет 10 тыс. руб., а все остальные остатки – нулевые.

Данные о дебиторской задолженности (ДЗ) по состоянию на 1 января вносятся из соответствующей статьи баланса за предыдущий период, а на каждую последующую дату рассчитываются как сумма дебиторской задолженности на предыдущую дату и отгрузки периода, за минусом поступлений от покупателей за период.

В примере дебиторская задолженность на 1 января равна нулю. В последующие периоды она не возникает. Кредиторская задолженность (КЗ) формируется из-за зарплаты и арендных расходов, поскольку они выплачиваются в месяце, следующем за месяцем начисления. Кредиторскую задолженность можно рассчитать как сумму кредиторской задолженности на предыдущую дату и начислений затрат текущего периода, за минусом оплаты задолженности в текущем периоде.

После того как построение баланса подобным образом завершено, остается удостовериться, что все было сделано правильно – активы равны пассивам. В нашем примере они совпадают, значит, модель работает, выдает корректные результаты.

Построенная таким образом финансовая модель обозначает основные группы показателей, характеризующих деятельность компании (доходы, расходы, денежные средства и т. п.), увязывает их в три сводных плана компании. Даже эту простейшую на первый взгляд модель можно использовать для сценарного анализа. В частности, если исключить из плана продаж услугу № 1 (соответствующую строку удалять не нужно, достаточно проставить по ней нули), то можно увидеть, насколько ухудшатся показатели рентабельности и ликвидности.

**Какие аналитические показатели внести в финансовую модель**

Чтобы превратить модель в полноценный инструмент сценарного анализа, потребуется «насытить» ее аналитикой, детализировать исходную информацию до показателей, которыми можно управлять на практике. Например, в случае с компанией, оказывающей услуги, очевидна необходимость детализации плана продаж, внесенного ранее в модель в денежном выражении. Выручку по каждому виду услуг можно рассчитать как произведение цены единицы услуги и количества указанных услуг. На практике, естественно, план продаж формируется исходя из конъюнктуры рынка, ожидаемого спроса, предполагаемой цены реализации, достигнутых договоренностей с ключевыми клиентами, запланированных маркетинговых мероприятий, ценовой и кредитной политики и т. д. (подробнее о подготовке плана продаж см. [Как планировать продажи при составлении бюджета](https://www.1fd.ru/#/document/173/646/)).

Аналогично детализируйте и другие исходные данные. Например, арендную плату можно было бы разложить на площадь арендуемого помещения и стоимость одного квадратного метра, зарплату расписать по сотрудникам, управленческие расходы разбить по видам. В итоге функциональность модели развивается до такого уровня, что можно увидеть, как влияет изменение любого, даже самого незначительного параметра на конечный результат.

Использование финансовой модели при планировании деятельности помогает увидеть, как те или иные планы развития отражаются на структуре активов, пассивов, доходов и расходов предприятия, а также определить, от каких факторов в наибольшей степени зависят будущая прибыль, ликвидность и финансовая устойчивость. Модель служит, скорее, инструментом мониторинга текущей ситуации в компании и выработки адекватной финансовой политики.

Финансовую модель стоит использовать в процессе бюджетирования сразу же после утверждения плана продаж. Если его «прогнать» через модель, то полученный финансовый результат можно показать акционерам, чтобы установить целевые значения по затратам, прибыли, дивидендам. Если планируемая выручка не обеспечивает необходимой прибыли с точки зрения акционеров, прямо в модели корректируются влияющие показатели. Окончательный вариант расчетов модели определяет целевые значения бюджетных лимитов для всех центров финансовой ответственности. В течение года модель можно будет корректировать, проставлять по пройденным месяцам фактические данные вместо плановых и контролировать таким образом финансовые результаты, отслеживать негативные тенденции и четко понимать, к чему они приведут компанию.

Чтобы не возникло серьезных проблем при использовании модели, будет вполне уместно:

* унифицировать ее сводные планы (бюджеты доходов и расходов, движения денежных средств, прогнозный баланс) с используемыми в компании формами управленческой отчетности;
* максимально приблизить алгоритмы расчета показателей модели к учетной политике предприятия;
* формы представления исходных данных модели (например, план продаж в натуральном выражении, прайс-листы и т. п.) унифицировать с формой соответствующих документов, применяемых в компании. К примеру, если план продаж в модели будет сформирован в том же виде, в котором этот план в компании формирует отдел продаж, существенно упростится процедура оценки и утверждения этих планов: достаточно будет внести их в модель и увидеть результат деятельности всей компании.

Если задействуете в бюджете сводные таблицы, заранее сделайте так, чтобы они обновлялись автоматически.

**Видеосеминар. Как заставить автоматически обновляться сводные таблицы**

«Как разработать финансовую модель компании в Excel». И. Николаев  
© Материал из ФСС "Система Финансовый директор".  
Подробнее: <https://www.1fd.ru/#/document/173/422/bssPhr2/?of=copy-9a1c476f10>

## Абстрактная фабрика (Abstract Factory)

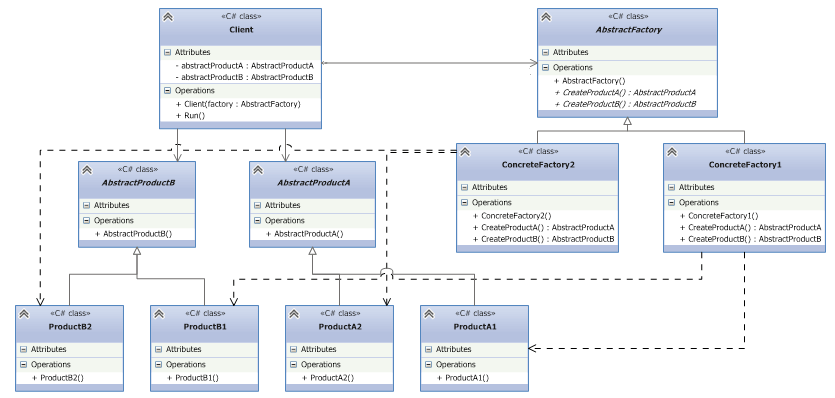
**Последнее обновление: 31.10.2015**

Паттерн "Абстрактная фабрика" (Abstract Factory) предоставляет интерфейс для создания семейств взаимосвязанных объектов с определенными интерфейсами без указания конкретных типов данных объектов.

### Когда использовать абстрактную фабрику

* Когда система не должна зависеть от способа создания и компоновки новых объектов
* Когда создаваемые объекты должны использоваться вместе и являются взаимосвязанными

С помощью UML абстрактную фабрику можно представить следующим образом:



Формальное определение паттерна на языке C# может выглядеть следующим образом:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61 | abstract class AbstractFactory  {      public abstract AbstractProductA CreateProductA();      public abstract AbstractProductB CreateProductB();  }  class ConcreteFactory1: AbstractFactory  {      public override AbstractProductA CreateProductA()      {          return new ProductA1();      }        public override AbstractProductB CreateProductB()      {          return new ProductB1();      }  }  class ConcreteFactory2: AbstractFactory  {      public override AbstractProductA CreateProductA()      {          return new ProductA2();      }        public override AbstractProductB CreateProductB()      {          return new ProductB2();      }  }    abstract class AbstractProductA  {}    abstract class AbstractProductB  {}    class ProductA1: AbstractProductA  {}    class ProductB1: AbstractProductB  {}    class ProductA2: AbstractProductA  {}    class ProductB2: AbstractProductB  {}    class Client  {      private AbstractProductA abstractProductA;      private AbstractProductB abstractProductB;        public Client(AbstractFactory factory)      {          abstractProductB = factory.CreateProductB();          abstractProductA = factory.CreateProductA();      }      public void Run()      { }  } |

Паттерн определяет следующих участников:

* Абстрактные классы **AbstractProductA** и **AbstractProductB** определяют интерфейс для классов, объекты которых будут создаваться в программе.
* Конкретные классы **ProductA1 / ProductA2** и **ProductB1 / ProductB2** представляют конкретную реализацию абстрактных классов
* Абстрактный класс фабрики **AbstractFactory** определяет методы для создания объектов. Причем методы возвращают абстрактные продукты, а не их конкретные реализации.
* Конкретные классы фабрик **ConcreteFactory1** и **ConcreteFactory2** реализуют абстрактные методы базового класса и непосредственно определяют какие конкретные продукты использовать
* Класс клиента **Client** использует класс фабрики для создания объектов. При этом он использует исключительно абстрактный класс фабрики AbstractFactory и абстрактные классы продуктов AbstractProductA и AbstractProductB и никак не зависит от их конкретных реализаций

Посмотрим, как мы можем применить паттерн. Например, мы делаем игру, где пользователь должен управлять некими супергероями, при этом каждый супергерой имеет определенное оружие и определенную модель передвижения. Различные супергерои могут определяться комплексом признаков. Например, эльф может летать и должен стрелять из арбалета, другой супергерой должен бегать и управлять мечом. Таким образом, получается, что сущность оружия и модель передвижения являются взаимосвязанными и используются в комплексе. То есть имеется один из доводов в пользу использования абстрактной фабрики.

И кроме того, наша задача при проектировании игры абстрагировать создание супергероев от самого класса супергероя, чтобы создать более гибкую архитектуру. И для этого применим абстрактную фабрику:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109 | class Program  {      static void Main(string[] args)      {          Hero elf = new Hero(new ElfFactory());          elf.Hit();          elf.Run();            Hero voin = new Hero(new VoinFactory());          voin.Hit();          voin.Run();            Console.ReadLine();      }  }  //абстрактный класс - оружие  abstract class Weapon  {      public abstract void Hit();  }  // абстрактный класс движение  abstract class Movement  {      public abstract void Move();  }    // класс арбалет  class Arbalet : Weapon  {      public override void Hit()      {          Console.WriteLine("Стреляем из арбалета");      }  }  // класс меч  class Sword : Weapon  {      public override void Hit()      {          Console.WriteLine("Бьем мечом");      }  }  // движение полета  class FlyMovement : Movement  {      public override void Move()      {          Console.WriteLine("Летим");      }  }  // движение - бег  class RunMovement : Movement  {      public override void Move()      {          Console.WriteLine("Бежим");      }  }  // класс абстрактной фабрики  abstract class HeroFactory  {      public abstract Movement CreateMovement();      public abstract Weapon CreateWeapon();  }  // Фабрика создания летящего героя с арбалетом  class ElfFactory : HeroFactory  {      public override Movement CreateMovement()      {          return new FlyMovement();      }        public override Weapon CreateWeapon()      {              return new Arbalet();      }  }  // Фабрика создания бегущего героя с мечом  class VoinFactory : HeroFactory  {      public override Movement CreateMovement()      {          return new RunMovement();      }        public override Weapon CreateWeapon()      {          return new Sword();      }  }  // клиент - сам супергерой  class Hero  {      private Weapon weapon;      private Movement movement;      public Hero(HeroFactory factory)      {          weapon = factory.CreateWeapon();          movement = factory.CreateMovement();      }      public void Run()      {          movement.Move();      }      public void Hit()      {          weapon.Hit();      }  } |

Таким образом, создание супергероя абстрагируется от самого класса супергероя. В то же время нельзя не отметить и недостатки шаблона. В частности, если нам захочется добавить в конфигурацию супергероя новый объект, например, тип одежды, то придется переделывать классы фабрик и класс супергероя. Поэтому возможности по расширению в данном паттерне имеют некоторые ограничения.

## Одиночка

**Последнее обновление: 23.12.2018**

Одиночка (Singleton, Синглтон) - порождающий паттерн, который гарантирует, что для определенного класса будет создан только один объект, а также предоставит к этому объекту точку доступа.

Когда надо использовать Синглтон? Когда необходимо, чтобы для класса существовал только один экземпляр

Синглтон позволяет создать объект только при его необходимости. Если объект не нужен, то он не будет создан. В этом отличие синглтона от глобальных переменных.

Классическая реализация данного шаблона проектирования на C# выглядит следующим образом:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | class Singleton  {      private static Singleton instance;        private Singleton()      {}        public static Singleton getInstance()      {          if (instance == null)              instance = new Singleton();          return instance;      }  } |

В классе определяется статическая переменная - ссылка на конкретный экземпляр данного объекта и приватный конструктор. В статическом методе getInstance() этот конструктор вызывается для создания объекта, если, конечно, объект отсутствует и равен null.

Для применения паттерна Одиночка создадим небольшую программу. Например, на каждом компьютере можно одномоментно запустить только одну операционную систему. В этом плане операционная система будет реализоваться через паттерн синглтон:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41 | class Program  {      static void Main(string[] args)      {          Computer comp = new Computer();          comp.Launch("Windows 8.1");          Console.WriteLine(comp.OS.Name);            // у нас не получится изменить ОС, так как объект уже создан          comp.OS = OS.getInstance("Windows 10");          Console.WriteLine(comp.OS.Name);            Console.ReadLine();      }  }  class Computer  {      public OS OS { get; set; }      public void Launch(string osName)      {          OS = OS.getInstance(osName);      }  }  class OS  {      private static OS instance;        public string Name { get; private set; }        protected OS(string name)      {          this.Name=name;      }        public static OS getInstance(string name)      {          if (instance == null)              instance = new OS(name);          return instance;      }  } |

### Синглтон и многопоточность

При применении паттерна синглтон в многопоточным программах мы можем столкнуться с проблемой, которую можно описать следующим образом:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | static void Main(string[] args)  {      (new Thread(() =>      {          Computer comp2 = new Computer();          comp2.OS = OS.getInstance("Windows 10");          Console.WriteLine(comp2.OS.Name);        })).Start();        Computer comp = new Computer();      comp.Launch("Windows 8.1");      Console.WriteLine(comp.OS.Name);      Console.ReadLine();  } |

Здесь запускается дополнительный поток, который получает доступ к синглтону. Параллельно выполняется тот код, который идет запуска потока и кторый также обращается к синглтону. Таким образом, и главный, и дополнительный поток пытаются инициализровать синглтон нужным значением - "Windows 10", либо "Windows 8.1". Какое значение сиглтон получит в итоге, пресказать в данном случае невозможно.

Вывод программы может быть такой:

Windows 8.1

Windows 10

Или такой:

Windows 8.1

Windows 8.1

В итоге мы сталкиваемся с проблемой инициализации синглтона, когда оба потока одновременно обращаются к коду:

|  |  |
| --- | --- |
| 1  2 | if (instance == null)      instance = new OS(name); |

Чтобы решить эту проблему, перепишем класс синглтона следующим образом:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | class OS  {      private static OS instance;        public string Name { get; private set; }      private static object syncRoot = new Object();        protected OS(string name)      {          this.Name = name;      }        public static OS getInstance(string name)      {          if (instance == null)          {              lock (syncRoot)              {                  if (instance == null)                      instance = new OS(name);              }          }          return instance;      }  } |

Чтобы избежать одновременного доступа к коду из разных потоков критическая секция заключается в блок **lock**.

### Другие реализации синглтона

Выше были рассмотрены общие стандартные реализации: потоконебезопасная и потокобезопасная реализации паттерна. Но есть еще ряд дополнительных реализаций, которые можно рассмотреть.

### Потокобезопасная реализация без использования lock

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | public class Singleton  {      private static readonly Singleton instance = new Singleton();        public string Date { get; private set; }        private Singleton()      {          Date = System.DateTime.Now.TimeOfDay.ToString();      }        public static Singleton GetInstance()      {          return instance;      }  } |

Данная реализация также потокобезопасная, то есть мы можем использовать ее в потоках так:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | (new Thread(() =>  {      Singleton singleton1 = Singleton.GetInstance();      Console.WriteLine(singleton1.Date);  })).Start();    Singleton singleton2 = Singleton.GetInstance();  Console.WriteLine(singleton2.Date); |

### Lazy-реализация

Определение объекта синглтона в виде статического поля класса открывает нам дорогу к созданию Lazy-реализации паттерна Синглтон, то есть такой реализации, где данные будут инициализироваться только перед непосредственным использованием. Поскольку статические поля инициализируются перед первым доступом к статическому членам класса и перед вызовом статического конструктора (при его наличии). Однако здесь мы можем столкнуться с двумя трудностями.

Во-первых, класс синглтона может иметь множество статических переменных. Возможно, мы вообще не будем обращаться к объекту синглтона, а будем использовать какие-то другие статические переменные:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | public class Singleton  {      private static readonly Singleton instance = new Singleton();      public static string text = "hello";      public string Date { get; private set; }        private Singleton()      {          Console.WriteLine($"Singleton ctor {DateTime.Now.TimeOfDay}");          Date = System.DateTime.Now.TimeOfDay.ToString();      }        public static Singleton GetInstance()      {          Console.WriteLine($"GetInstance {DateTime.Now.TimeOfDay}");          Thread.Sleep(500);          return instance;      }  }  class Program  {      static void Main(string[] args)      {          Console.WriteLine($"Main {DateTime.Now.TimeOfDay}");          Console.WriteLine(Singleton.text);          Console.Read();      }  } |

В данном случае идет только обращение к переменной text, однако статическое поле instance также будет инициализировано. Например, консольный вывод в данном случае мог бы выглядеть следующим образом:

Singleton ctor 16:05:54.1469982

Main 16:05:54.2920316

hello

В данном случае мы видим, что статическое поле instance инициализировано.

Для решения этой проблемы выделим отдельный внутренний класс в рамках класса синглтона:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | public class Singleton  {      public string Date { get; private set; }      public static string text = "hello";      private Singleton()      {          Console.WriteLine($"Singleton ctor {DateTime.Now.TimeOfDay}");          Date = DateTime.Now.TimeOfDay.ToString();      }        public static Singleton GetInstance()      {          Console.WriteLine($"GetInstance {DateTime.Now.TimeOfDay}");          return Nested.instance;      }        private class Nested      {          internal static readonly Singleton instance = new Singleton();      }  }  class Program  {      static void Main(string[] args)      {          Console.WriteLine($"Main {DateTime.Now.TimeOfDay}");          Console.WriteLine(Singleton.text);          Console.Read();      }  } |

Теперь статическая переменная, которая представляет объект синглтона, определена во вложенном классе Nested. Чтобы к этой переменной можно было обращаться из класса синглтона, она имеет модификатор internal, в то же время сам класс Nested имеет модификатор private, что позволяет гарантировать, что данный класс будет доступен только из класса Singleton.

Консольный вывод в данном случае мог бы выглядеть следующим образом:

Main 16:11:40.1320873

hello

Далее мы сталкиваемся со второй проблемой: статические поля инициализируются перед первым доступом к статическому членам класса и перед вызовом статического конструктора (при его наличии). Но когда именно? Если класс содержит статические поля, не содержит статического конструктора, то время инициализации статических полей зависит от реализации платформы. Нередко это непосредственно перед первым использованием, но тем не менее момент точно не определен - это может быть происходить и чуть раньше. Однако если класс содержит статический конструктор, то статические поля будут инициализироваться непосредственно либо при создании первого экземпляра класса, либо при первом обращении к статическим членам класса.

Например, рассмотрим выполнение следующей программы:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | static void Main(string[] args)  {      Console.WriteLine($"Main {DateTime.Now.TimeOfDay}");      Console.WriteLine(Singleton.text);        Singleton singleton1 = Singleton.GetInstance();      Console.WriteLine(singleton1.Date);      Console.Read();  } |

Ее возможный консольный вывод:

Main 16:33:33.1404818

hello

Singleton ctor 16:33:33.1564802

GetInstance 16:33:33.1574824

16:33:33.1564802

Мы видим, что код метода GetInstance, который идет до вызова конструктора класса Singleton, выполняется после выполнения этого конструктора. Поэтому добавим в выше определенный класс Nested статический конструктор:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | public class Singleton  {      public string Date { get; private set; }      public static string text = "hello";      private Singleton()      {          Console.WriteLine($"Singleton ctor {DateTime.Now.TimeOfDay}");          Date = DateTime.Now.TimeOfDay.ToString();      }        public static Singleton GetInstance()      {          Console.WriteLine($"GetInstance {DateTime.Now.TimeOfDay}");          Thread.Sleep(500);          return Nested.instance;      }        private class Nested      {          static Nested() { }          internal static readonly Singleton instance = new Singleton();      }  } |

Теперь при выполнении той же программы мы получим полноценную Lazy-реализацию:

Main 16:37:18.4108064

hello

GetInstance 16:37:18.4208062

Singleton ctor 16:37:18.4218065

16:37:18.4228061

### Реализация через класс Lazy<T>

Еще один способ создания синглтона представляет использование класса Lazy<T>:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | public class Singleton  {      private static readonly Lazy<Singleton> lazy =          new Lazy<Singleton>(() => new Singleton());        public string Name { get; private set; }        private Singleton()      {          Name = System.Guid.NewGuid().ToString();      }        public static Singleton GetInstance()      {          return lazy.Value;      }  } |