Частное учреждение образования

«Колледж бизнеса и права»

УТВЕРЖДАЮ

Ведущий

методист колледжа

\_\_\_\_\_\_\_\_\_\_\_ Е.В. Паскал

«\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_2021

|  |  |
| --- | --- |
| Специальность: 2-40 01 01 «Программное обеспечение информационных технологий» | Дисциплина: «Технология разработки программного обеспечения» |

**Лабораторная работа № 21**

**Инструкционно-технологическая карта**

Тема: Создание программной системы, использующей порождающие шаблоны

Цель: Научиться создавать программные системы, используя порождающие шаблоны

Время выполнения: 2 часа

**Контрольныевопросы**

1. Назовите и опишите порождающие шаблоны проектирования

**Теоретические сведения для выполнения работы**

Шаблонами проектирования (Design Patterns) называют решения часто встречающихся проблем в области разработки программного обеспечения. Шаблоны проектирования не являются готовыми решениями, которые можно трансформировать непосредственно в код, а представляют общее описание решения проблемы, которое можно использовать в различных ситуациях.

**Порождающие шаблоны проектирования** предназначены для создания объектов, позволяя системе оставаться независимой как от самого процесса порождения, так и от типов порождаемых объектов. Прежде чем рассматривать особенности каждого из порождающих шаблонов, рассмотрим на примере типичные проблемы, встающие перед разработчиками при порождении в системе объектов новых типов.

Рассмотрим порождающие шаблоны и особенности каждого из них:

**Шаблон «Фабричный метод» (Factory Method).** В данном шаблоне вводится полиморфный класс Factory, в котором определяется интерфейс фабричного метода, подобного CreateWarrior( ), а ответственность за создание объектов конкретных классов переносится на производные от Factory классы, в которых этот метод переопределяется;

**Шаблон «Абстрактная фабрика» (Abstract Factory).** Использует несколько фабричных методов и предназначен для создания целого семейства или группы взаимосвязанных объектов. Для приведённого выше примера, шаблон может предоставлять методы для создания всех типов воинов;

**Шаблон «Строитель» (Builder).** Определяет процесс поэтапного конструирования сложного объекта, в результате которого могут получаться разные представления этого объекта;

**Шаблон «Прототип» (Prototype).** Создаёт новые объекты с помощью прототипов (создавая копии самого себя);

**Шаблон «Синглетон» (Singleton).** Позволяет создавать только один экземпляр класса.

**Шаблон «Фабричный метод»**

В системе часто требуется создавать объекты самых разных типов. Шаблон «Фабричный метод» (Factory Method) может быть полезным в решении следующих задач:

Система должна оставаться расширяемой путем добавления объектов новых типов.

Заранее известно, когда нужно создавать объект, но неизвестен его тип.

Для того, чтобы система оставалась независимой от различных типов объектов, Шаблон «Фабричный метод» использует механизм полиморфизма - классы всех конечных типов наследуются от одного абстрактного базового класса, предназначенного для полиморфного использования. В этом базовом классе определяется единый интерфейс, через который пользователь будет оперировать объектами конечных типов.

Существуют две разновидности Шаблон «Фабричный метод»:

**Обобщенный конструктор**, когда в том же самом полиморфном базовом классе, от которого наследуют производные классы всех создаваемых в системе типов, определяется статический фабричный метод. В качестве параметра в этот метод должен передаваться идентификатор типа создаваемого объекта.

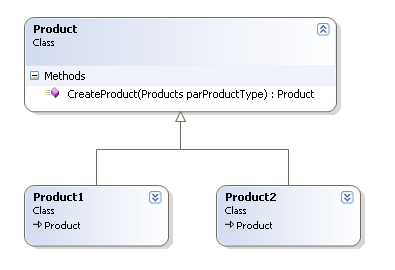


Рисунок 1 Пример диаграммы классов для фабричного метода с обобщённым конструктором

**Классический вариант фабричного метода,** когда интерфейс фабричных методов объявляется в независимом классе-фабрике, а их реализация определяется конкретными подклассами этого класса.

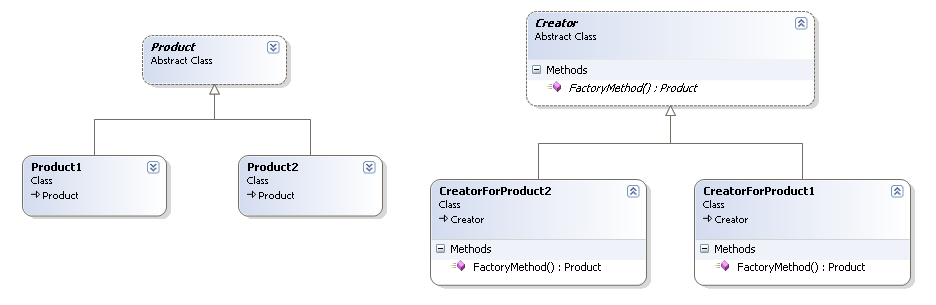


Рисунок 2 Пример классического фабричного метода

**Достоинства:**

* Создает объекты разных типов, позволяя системе оставаться независимой как от самого процесса создания, так и от типов создаваемых объектов.

**Недостатки:**

* В случае классического варианта паттерна даже для порождения единственного объекта необходимо создавать соответствующую фабрику

**Шаблон «Абстрактная фабрика»**

Данный шаблон позволяет изменять поведение системы, оперируя создаваемыми объектами, при этом сохраняя интерфейсы. Он позволяет создавать целые группы взаимосвязанных объектов, которые, будучи созданными одной фабрикой, реализуют общее поведение. Шаблон реализуется созданием абстрактного класса Factory, который представляет собой интерфейс для создания компонентов системы (например, для оконного интерфейса он может создавать окна и кнопки). Затем пишутся классы, реализующие этот интерфейс.

Данный шаблон рекомендуется использовать в следующих случаях:

* Система должна оставаться независимой как от процесса создания новых объектов, так и от типов порождаемых объектов. Непосредственное использование выражения new в коде приложения;
* Необходимо создавать группы или семейства взаимосвязанных объектов, исключая возможность одновременного использования объектов из разных семейств в одном контексте.

Для решения задачи по созданию семейств взаимосвязанных объектов шаблон «Абстрактная фабрика» вводит понятие абстрактной фабрики. Абстрактная фабрика представляет собой некоторый полиморфный базовый класс, назначением которого является объявление интерфейсов фабричных методов, служащих для создания продуктов всех основных типов (один фабричный метод на каждый тип продукта). Производные от него классы, реализующие эти интерфейсы, предназначены для создания продуктов всех типов внутри семейства или группы.

Обобщённая диаграмма классов выглядит следующим образом:

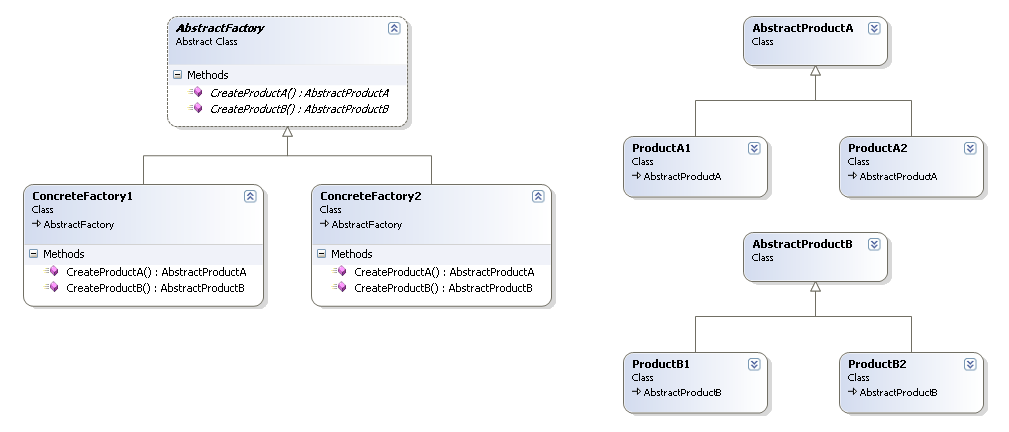


Рисунок 3 Диаграмма классов для реализации абстрактной фабрики

В данном примере есть отдельные иерархии наследования для продуктов «A» и «B». Абстрактный класс фабрики реализует два метода по созданию объектов классов «A» и «B». Конкретные реализации фабрики переопределяют данные методы и создают каждый свой вид продуктов.

**Достоинства:**

- Скрывает сам процесс порождения объектов, а также делает систему независимой от типов создаваемых объектов, специфичных для различных семейств или групп (пользователи оперируют этими объектами через соответствующие абстрактные интерфейсы);

- Позволяет быстро настраивать систему на нужное семейство создаваемых объектов.

**Недостатки:**

- Трудно добавлять новые типы создаваемых объектов или заменять существующие, так как интерфейс базового класса абстрактной фабрики фиксирован.

**Шаблон «Строитель»**

Шаблон «Строитель» (Builder) позволяет не только создавать объекты, но и конструировать сложные объекты по заложенному в него алгоритму.

Может помочь в решении следующих задач:

- В системе могут существовать сложные объекты, создание которых за одну операцию затруднительно или невозможно. Требуется поэтапное построение объектов с контролем результатов выполнения каждого этапа.

- Данные должны иметь несколько представлений.

Шаблон «Строитель» определяет алгоритм поэтапного создания продукта в специальном классе Director (распорядитель), а ответственность за координацию процесса сборки отдельных частей продукта возлагает на иерархию классов. В этой иерархии базовый класс объявляет интерфейсы для построения отдельных частей продукта, а соответствующие подклассы их реализуют подходящим образом, например, создают или получают нужные ресурсы, сохраняют промежуточные результаты, контролируют результаты выполнения операций.

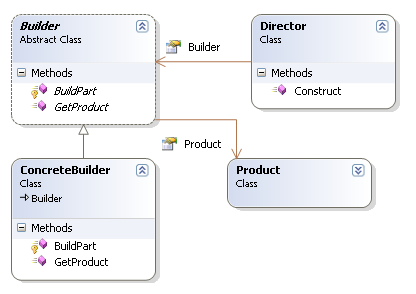


Рисунок 4 Диаграмма классов для шаблона «Строитель»

Класс Director содержит указатель или ссылку на экземпляр одного из дочерних классов Builder, определяющего соответствующе представление. После этого, класс Director может обрабатывать клиентские запросы на создание объекта. Получив такой запрос, с помощью имеющегося экземпляра строителя, Director строит продукт по частям, а затем возвращает его пользователю.

**Достоинства:**

- Возможность контролировать процесс создания сложного продукта.

- Возможность получения разных представлений некоторых данных.

**Недостатки:**

- ConcreteBuilder и создаваемый им продукт жестко связаны между собой, поэтому при внесеннии изменений в класс продукта скорее всего придется соответствующим образом изменять и класс ConcreteBuilder.

**Шаблон «Прототип»**

Для создания новых объектов шаблон «Прототип» использует клонирование. Прототип - это уже существующий в системе объект, который поддерживает операцию клонирования, то есть умеет создавать копию самого себя. Таким образом, для создания объекта некоторого класса достаточно выполнить операцию clone() соответствующего прототипа.

Шаблон «Прототип» реализует подобное поведение следующим образом: все классы, объекты которых нужно создавать, должны быть подклассами одного общего абстрактного базового класса. Этот базовый класс должен объявлять интерфейс метода clone(). Также здесь могут объявляться виртуальными и другие общие методы, например, initialize() в случае, если после клонирования нужна инициализация вновь созданного объекта. Все производные классы должны реализовывать метод clone().

Данный шаблон можно использовать в следующих случаях:

- Система должна оставаться независимой как от процесса создания новых объектов, так и от типов порождаемых объектов. Непосредственное использование выражения new в коде приложения считается нежелательным;

- Необходимо создавать объекты, точные классы которых становятся известными уже на стадии выполнения программы.

Шаблон «Фабричный метод» также делает систему независимой от типов порождаемых объектов, но для этого он вводит параллельную иерархию классов: для каждого типа создаваемого объекта должен присутствовать соответствующий класс-фабрика, что может быть нежелательно. «Прототип» лишен этого недостатка.

Диаграмма классов выглядит следующим образом:

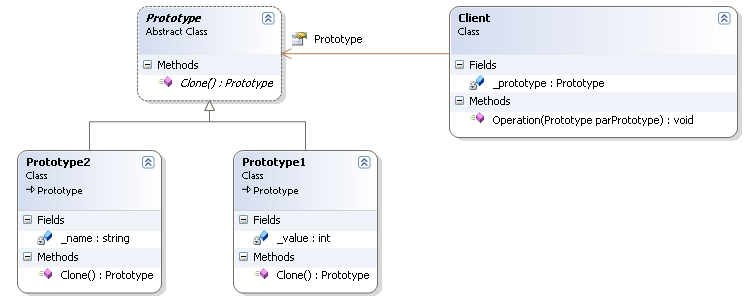


Рисунок 5 Диаграмма классов для шаблона «Прототип»

Для порождения объекта некоторого типа в системе должен существовать его прототип. Прототип представляет собой объект того же типа, единственным назначением которого является создание подобных ему объектов.

**Достоинства:**

- Для создания новых объектов клиенту необязательно знать их конкретные классы.

- Возможность гибкого управления процессом создания новых объектов за счет возможности динамических добавления и удаления прототипов в реестр.

**Недостатки:**

- Каждый тип создаваемого продукта должен реализовывать операцию клонирования clone(). В случае, если требуется глубокое копирование объекта (объект содержит ссылки или указатели на другие объекты), это может быть непростой задачей.

**Шаблон «Одиночка»**

Часто в системе могут существовать сущности только в единственном экземпляре, например, система ведения системного журнала сообщений или драйвер дисплея. В таких случаях необходимо уметь создавать единственный экземпляр некоторого типа, предоставлять к нему доступ извне и запрещать создание нескольких экземпляров того же типа.

Шаблон «Одиночка» (Singleton) предоставляет такие возможности.

Для того, чтобы у класса существовал единственный экземпляр, необходимо запретить создавать экземпляр класса извне и контролировать единственность внутри кода самого класса.

Это обеспечивается при помощи следующего решения:

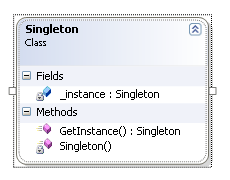


Рисунок 6 Диаграмма классов для реализации шаблона «Одиночка»

В данном примере реализован закрытый конструктор, который не даст создавать экземпляр класса извне.

Создание единственного экземпляра контролируется закрытым статическим полем \_instance, которое инициализируется при первом обращении к получению экземпляра класса через метод GetInstance(), который предоставляет клиентам единственный экземпляр класса.

Достоинства и недостатки

**Достоинства:**

* Класс сам контролирует процесс создания единственного экземпляра;
* Шаблон легко адаптировать для создания нужного числа экземпляров;
* Возможность создания объектов классов, производных от Singleton.

**Недостатки:**

* В случае использования нескольких взаимозависимых одиночек их реализация может резко усложниться.

**Порядок выполнения работы**

1. Изучите теоретические сведения.

2. Реализовать программу, использующую один из порождающих шаблонов по теме индивидуального задания.

Преподаватель \_\_\_\_\_\_\_\_ А.С. Комаровский

Рассмотрено на заседании цикловой

комиссии ПОИТ №10

Протокол № от «\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2021

Председатель ЦК \_\_\_\_\_\_\_\_ В.Ю. Михалевич