**Занятие № 19**

**Номер учебной группы:** П-16.

**Фамилия, инициалы учащегося:** Новалихина С.К.

**Дата выполнения работы:** 30.11.2022.

**Тема работы:** «Использование шаблонов при разработке программного продукта»

**Цель работы:** разрабатывать и применять на практике различного типа шаблоны проектирования и реализации.

**Ход работы**

**Задание 1**

Изучила теоретический материал по теме «Шаблоны проектирования».

**Задание 2**

Разработала шаблон проектирования и реализации для будущего использования при разработке ПО.

Шаблон:

**using System;**

**namespace RefactoringGuru.DesignPatterns.Prototype.Conceptual**

**{**

**public class Person**

**{**

**public int Age;**

**public DateTime BirthDate;**

**public string Name;**

**public IdInfo IdInfo;**

**public Person ShallowCopy()**

**{**

**return (Person) this.MemberwiseClone();**

**}**

**public Person DeepCopy()**

**{**

**Person clone = (Person) this.MemberwiseClone();**

**clone.IdInfo = new IdInfo(IdInfo.IdNumber);**

**clone.Name = String.Copy(Name);**

**return clone;**

**}**

**}**

**public class IdInfo**

**{**

**public int IdNumber;**

**public IdInfo(int idNumber)**

**{**

**this.IdNumber = idNumber;**

**}**

**}**

**class Program**

**{**

**static void Main(string[] args)**

**{**

**Person p1 = new Person();**

**p1.Age = 42;**

**p1.BirthDate = Convert.ToDateTime("1977-01-01");**

**p1.Name = "Jack Daniels";**

**p1.IdInfo = new IdInfo(666);**

**// Выполнить поверхностное копирование p1 и присвоить её p2.**

**Person p2 = p1.ShallowCopy();**

**// Сделать глубокую копию p1 и присвоить её p3.**

**Person p3 = p1.DeepCopy();**

**// Вывести значения p1, p2 и p3.**

**Console.WriteLine("Original values of p1, p2, p3:");**

**Console.WriteLine(" p1 instance values: ");**

**DisplayValues(p1);**

**Console.WriteLine(" p2 instance values:");**

**DisplayValues(p2);**

**Console.WriteLine(" p3 instance values:");**

**DisplayValues(p3);**

**// Изменить значение свойств p1 и отобразить значения p1, p2 и p3.**

**p1.Age = 32;**

**p1.BirthDate = Convert.ToDateTime("1900-01-01");**

**p1.Name = "Frank";**

**p1.IdInfo.IdNumber = 7878;**

**Console.WriteLine("\nValues of p1, p2 and p3 after changes to p1:");**

**Console.WriteLine(" p1 instance values: ");**

**DisplayValues(p1);**

**Console.WriteLine(" p2 instance values (reference values have changed):");**

**DisplayValues(p2);**

**Console.WriteLine(" p3 instance values (everything was kept the same):");**

**DisplayValues(p3);**

**}**

**public static void DisplayValues(Person p)**

**{**

**Console.WriteLine(" Name: {0:s}, Age: {1:d}, BirthDate: {2:MM/dd/yy}",**

**p.Name, p.Age, p.BirthDate);**

**Console.WriteLine(" ID#: {0:d}", p.IdInfo.IdNumber);**

**}**

**}**

**}**

Шаблон реализован в базовой библиотеке C# посредством интерфейса ICloneable. Используется для клонирования объектов любой сложности без привязки к их конкретным классам.

**Задание 3**

Ответила на контрольные вопросы:

1. Что вы понимаете под шаблонами проектирования?

Ответ: Шаблон проектирования или паттерн (англ. design pattern) в разработке программного обеспечения — повторяемая архитектурная конструкция, представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста.

2. Перечислите плюсы и минусы применения шаблонов.

Ответ:

**Плюсы:**

1) Снижении сложности разработки за счёт готовых абстракций для решения целого класса проблем.

2) Снижение количества ошибок.

**Минусы:**

1) Слепое следование некоторому выбранному шаблону может привести к усложнению программы.

2) У разработчика может возникнуть желание попробовать некоторый шаблон в деле без особых оснований.

3. Перечислите типы шаблонов проектирования и реализации.

Ответ:

### Типы шаблонов проектирования

### Основные

|  |  |  |  |
| --- | --- | --- | --- |
| **Название** | **Оригинальное название** | **Описание** | **Описан в** [***Design Patterns***](https://ru.wikipedia.org/wiki/Design_Patterns) |
| **Основные шаблоны (Fundamental)** | | | |
| [Шаблон делегирования](https://ru.wikipedia.org/wiki/%D0%A8%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD_%D0%B4%D0%B5%D0%BB%D0%B5%D0%B3%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) | Delegation pattern | Объект внешне выражает некоторое поведение, но в реальности передаёт ответственность за выполнение этого поведения связанному объекту. | Н/д |
| [Шаблон функционального дизайна](https://ru.wikipedia.org/wiki/%D0%A8%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD_%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B3%D0%BE_%D0%B4%D0%B8%D0%B7%D0%B0%D0%B9%D0%BD%D0%B0) | Functional design | Гарантирует, что каждый модуль компьютерной программы имеет только одну обязанность и исполняет её с минимумом побочных эффектов на другие части программы. | Н/д |
| [Интерфейс](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_(%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)) | Interface | Общий метод для структурирования компьютерных программ для того, чтобы их было проще понять. | Н/д |
| [**Порождающие шаблоны**](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%80%D0%BE%D0%B6%D0%B4%D0%B0%D1%8E%D1%89%D0%B8%D0%B5_%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD%D1%8B_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) **([Creational](https://en.wikipedia.org/wiki/Creational_pattern" \o "en:Creational pattern))** — шаблоны проектирования, которые абстрагируют процесс инстанцирования. Они позволяют сделать систему независимой от способа создания, композиции и представления объектов. Шаблон, порождающий классы, использует наследование, чтобы изменять инстанцируемый класс, а шаблон, порождающий объекты, делегирует инстанцирование другому объекту. | | | |
| [Абстрактная фабрика](https://ru.wikipedia.org/wiki/%D0%90%D0%B1%D1%81%D1%82%D1%80%D0%B0%D0%BA%D1%82%D0%BD%D0%B0%D1%8F_%D1%84%D0%B0%D0%B1%D1%80%D0%B8%D0%BA%D0%B0_(%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)) | Abstract factory | Класс, который представляет собой интерфейс для создания компонентов системы. | Да |
| [Строитель](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D0%BE%D0%B8%D1%82%D0%B5%D0%BB%D1%8C_(%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)) | Builder | Класс, который представляет собой интерфейс для создания сложного объекта. | Да |
| [Фабричный метод](https://ru.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B1%D1%80%D0%B8%D1%87%D0%BD%D1%8B%D0%B9_%D0%BC%D0%B5%D1%82%D0%BE%D0%B4_(%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)) | Factory method | Определяет интерфейс для создания объекта, но оставляет подклассам решение о том, какой класс инстанцировать. | Да |
| [Прототип](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%82%D0%BE%D1%82%D0%B8%D0%BF_(%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)) | Prototype | Определяет интерфейс создания объекта через клонирование другого объекта вместо создания через конструктор. | Да |
| [**Структурные шаблоны**](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%BD%D1%8B%D0%B5_%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD%D1%8B_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) **(Structural)** определяют различные сложные структуры, которые изменяют [интерфейс](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_(%D0%BE%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) уже существующих объектов или его реализацию, позволяя облегчить разработку и оптимизировать программу. | | | |
| [Адаптер](https://ru.wikipedia.org/wiki/%D0%90%D0%B4%D0%B0%D0%BF%D1%82%D0%B5%D1%80_(%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)) | Adapter / Wrapper | Объект, обеспечивающий взаимодействие двух других объектов, один из которых использует, а другой предоставляет несовместимый с первым интерфейс. | Да |
| [Фасад](https://ru.wikipedia.org/wiki/%D0%A4%D0%B0%D1%81%D0%B0%D0%B4_(%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)) | Facade | Объект, который абстрагирует работу с несколькими классами, объединяя их в единое целое. | Да |
| [**Поведенческие шаблоны**](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B5_%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD%D1%8B_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) **(Behavioral)** определяют взаимодействие между объектами, увеличивая таким образом его гибкость. | | | |
| [Наблюдатель](https://ru.wikipedia.org/wiki/%D0%9D%D0%B0%D0%B1%D0%BB%D1%8E%D0%B4%D0%B0%D1%82%D0%B5%D0%BB%D1%8C_(%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)) или[Publish/subscribe](https://ru.wikipedia.org/w/index.php?title=Publish/subscribe&action=edit&redlink=1)[[en]](https://en.wikipedia.org/wiki/Publish/subscribe) | Observer | Определяет зависимость типа «один ко многим» между объектами таким образом, что при изменении состояния одного объекта все зависящие от него оповещаются об этом событии. | Да |

4. Дайте характеристику шаблону проектирования «Делегирование»

Ответ: Делегирование (англ. Delegation) — основной шаблон проектирования, в котором объект внешне выражает некоторое поведение, но в реальности передаёт ответственность за выполнение этого поведения связанному объекту. Шаблон делегирования является фундаментальной абстракцией, на основе которой реализованы другие шаблоны - композиция (также называемая агрегацией), примеси (mixins) и аспекты (aspects).

Цель применения. Возможность изменить поведение конкретного экземпляра объекта вместо создания нового класса путём наследования.

5. Дайте характеристику шаблону проектирования «Функциональный дизайн».

Ответ: Функциональный дизайн (англ. Functional design) — основной шаблон проектирования, использующийся для упрощения проектирования ПО. Функциональный дизайн гарантирует, что каждый модуль компьютерной программы имеет только одну обязанность и исполняет её с минимумом побочных эффектов на другие части программы. Функционально разработанные модули имеют низкое зацепление.

Преимущества. Системы с функционально-спроектированными частями легче модифицировать, потому что каждая часть делает только то, для чего она предназначена. Так как поддержка программы занимает больше 3/4 жизни успешной системы, эта особенность является решающим преимуществом. Это также делает систему лёгкой для понимания и документирования, что также упрощает обучение. Результатом является то, что практическое время жизни функциональной системы больше.

Преимуществом для реализации является то, что если программный модуль имеет единственное предназначение, он будет проще и тем самым — легче и менее дорогостоящим для проектирования и реализации.

В программных системах функциональный модуль будет легче использовать многократно, потому что он менее вероятно будет иметь побочные эффекты, которые проявятся в других частях системы.

Стандартный способ обеспечения функционального дизайна — это обзор описания модуля. Если описание включает связи, такие, как «и» либо «или», тогда дизайн имеет более чем одно предназначение, и соответственно возможно будет иметь побочные эффекты. Предназначения должны быть разделены в отдельные модули для того, чтобы функциональный дизайн был бы достижим.

Каждая компьютерная система имеет части, которые не могут быть функционально чисты, потому что они существуют для распределения тактов процессора или других ресурсов различным модулям. Например, большинство систем имеют раздел «инициализации», который запускает модули. Другие хорошо известные примеры включают в себя таблицу векторов прерываний и главный цикл.

Некоторые функции, в сущности, имеют смешанную семантику. Например, функция «вывести автомобиль из гаража», по сути, имеет побочный эффект изменения «положения автомобиля». В некоторых случаях смешанная семантика может быть расширена на большое топологическое дерево или граф связанных понятий. В этих необычных случаях некоторые

авторитеты не рекомендуют использовать функциональный дизайн. Вместо этого попробуйте полиморфизм и наследование.

6. Дайте характеристику шаблону проектирования «Интерфейс».

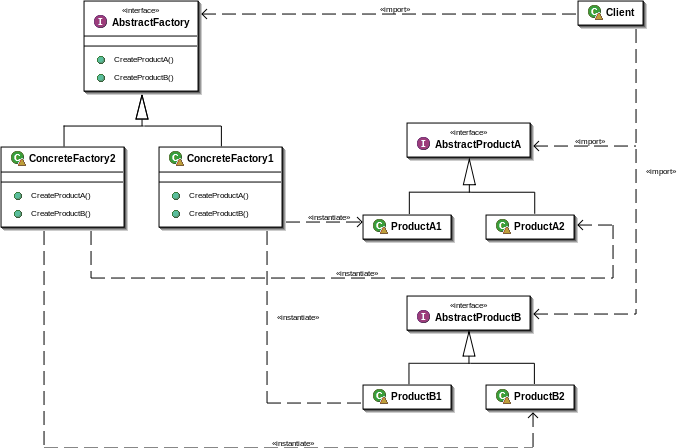
Ответ: Интерфейс (англ. Interface) — основной шаблон проектирования, являющийся общим методом для структурирования компьютерных программ для того, чтобы их было проще понять. В общем, интерфейс — это класс, который обеспечивает программисту простой или более программно-специфический способ доступа к другим классам. Интерфейс может содержать набор объектов и обеспечивать простую, высокоуровневую функциональность для программиста (например, Шаблон Фасад); он может обеспечивать более чистый или более специфический способ использования сложных классов («класс-обёртка»); он может использоваться в качестве «клея» между двумя различными API (Шаблон Адаптер); и для многих других целей

7. Дайте характеристику шаблону проектирования «Абстрактная фабрика».

Ответ: Шаблон проектирования «Абстрактная фабрика»

**Назначение шаблона:** предоставляет интерфейс для создания семейств взаимосвязанных или взаимозависимых объектов, не специфицируя их конкретных классов. 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.

**Структура:**



**Применяется в случаях:** когда программа должна быть независимой от процесса и типов создаваемых новых объектов.

Когда необходимо создать семейства или группы взаимосвязанных объектов исключая возможность одновременного использования объектов из разных этих наборов в одном контексте

**Плюсы:** изолирует конкретные классы, упрощает замену семейств продуктов, гарантирует сочетаемость продуктов.

**Минусы:** сложно добавить поддержку нового вида продуктов.

8. Дайте характеристику шаблону проектирования «Строитель».

Ответ: **Строитель** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Builder*) — [порождающий шаблон проектирования](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%80%D0%BE%D0%B6%D0%B4%D0%B0%D1%8E%D1%89%D0%B8%D0%B5_%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD%D1%8B_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) предоставляет способ создания составного объекта.

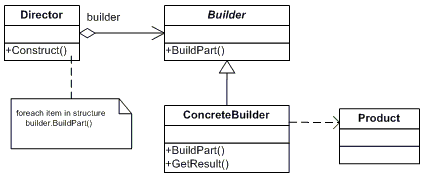
Отделяет конструирование сложного объекта от его представления, так что в результате одного и того же процесса конструирования могут получаться разные представления.

Плюсы: позволяет изменять внутреннее представление продукта, изолирует код, реализующий конструирование и представление, дает более тонкий контроль над процессом конструирования.

Применение:

Алгоритм создания сложного объекта не должен зависеть от того, из каких частей состоит объект и как они стыкуются между собой;

процесс конструирования должен обеспечивать различные представления конструируемого объекта.

[](https://ru.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB:Builder.gif)

9. Дайте характеристику шаблону проектирования «Фабричный метод»

Ответ: **Фабричный метод** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Factory Method* также известен как Виртуальный конструктор ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Virtual Constructor*)) — [порождающий шаблон проектирования](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%80%D0%BE%D0%B6%D0%B4%D0%B0%D1%8E%D1%89%D0%B8%D0%B5_%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD%D1%8B_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), предоставляющий подклассам интерфейс для создания экземпляров некоторого класса. В момент создания наследники могут определить, какой класс создавать. Иными словами, Фабрика [делегирует](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%BB%D0%B5%D0%B3%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) создание объектов наследникам родительского класса. Это позволяет использовать в коде программы не специфические классы, а манипулировать абстрактными объектами на более высоком уровне.

Определяет [интерфейс](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_(%D0%BE%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) для создания объекта, но оставляет подклассам решение о том, какой класс инстанцировать. Фабричный метод позволяет классу делегировать создание подклассов. Используется, когда:

* классу заранее неизвестно, объекты каких подклассов ему нужно создавать.
* класс спроектирован так, чтобы объекты, которые он создаёт, специфицировались подклассами.
* класс делегирует свои обязанности одному из нескольких вспомогательных подклассов, и планируется локализовать знание о том, какой класс принимает эти обязанности на себя

**Product** — продукт

определяет интерфейс объектов, создаваемых абстрактным методом;

**ConcreteProduct** — конкретный продукт

реализует интерфейс *Product*;

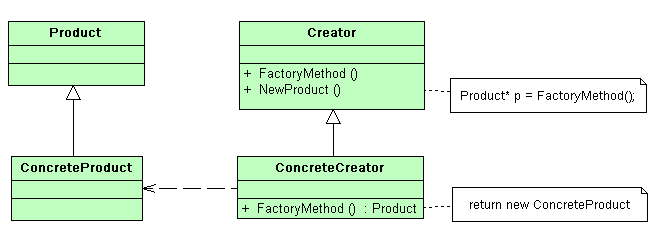
**Creator** — создатель

объявляет фабричный метод, который возвращает объект типа *Product*. Может также содержать реализацию этого метода «по умолчанию»;

может вызывать фабричный метод для создания объекта типа *Product*;

**ConcreteCreator** — конкретный создатель

переопределяет фабричный метод таким образом, чтобы он создавал и возвращал объект класса *ConcreteProduct*.



Позволяет сделать код создания объектов более универсальным, не привязываясь к конкретным классам (ConcreteProduct), а оперируя лишь общим интерфейсом (Product).

Позволяет установить связь между параллельными иерархиями классов.

10. Дайте характеристику шаблону проектирования «Прототип».

Ответ: **Прототип**, **(**[**англ.**](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) ***Prototype*)** — [порождающий шаблон проектирования](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%80%D0%BE%D0%B6%D0%B4%D0%B0%D1%8E%D1%89%D0%B8%D0%B5_%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD%D1%8B_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F).

Задаёт виды создаваемых объектов с помощью экземпляра-прототипа и создаёт новые объекты путём копирования этого прототипа. Он позволяет уйти от реализации и позволяет следовать принципу «программирование через интерфейсы». В качестве возвращающего типа указывается интерфейс/абстрактный класс на вершине иерархии, а классы-наследники могут подставить туда наследника, реализующего этот тип.

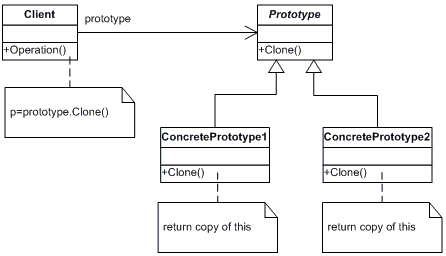
Проще говоря, это паттерн создания объекта через клонирование другого объекта вместо создания через конструктор.

Паттерн используется чтобы:

* избежать дополнительных усилий по созданию объекта стандартным путём (имеется в виду использование конструктора, так как в этом случае также будут вызваны конструкторы всей иерархии предков объекта), когда это непозволительно дорого для приложения.
* избежать наследования создателя объекта (object creator) в клиентском приложении, как это делает паттерн [abstract factory](https://ru.wikipedia.org/wiki/%D0%90%D0%B1%D1%81%D1%82%D1%80%D0%B0%D0%BA%D1%82%D0%BD%D0%B0%D1%8F_%D1%84%D0%B0%D0%B1%D1%80%D0%B8%D0%BA%D0%B0_(%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)).

Используйте этот шаблон проектирования, когда системe безразлично как именно в ней создаются, компонуются и представляются продукты:

* инстанцируемые классы определяются во время выполнения, например с помощью динамической загрузки;
* избежать построения иерархий классов или фабрик, параллельных иерархии классов продуктов;
* экземпляры класса могут находиться в одном из нескольких различных состояний. Может оказаться удобнее установить соответствующее число прототипов и клонировать их, а не инстанцировать каждый раз класс вручную в подходящем состоянии.

[](https://ru.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB:Prototype.gif)

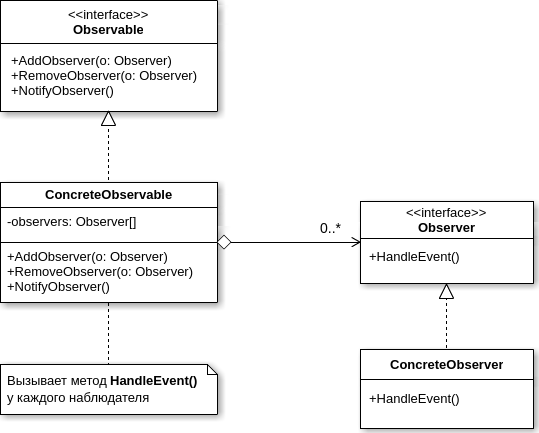
11. Дайте характеристику шаблону проектирования «Наблюдатель»

Ответ: **Наблюдатель** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Observer*) — [поведенческий](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B5_%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD%D1%8B_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) [шаблон проектирования](https://ru.wikipedia.org/wiki/%D0%A8%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F). Также известен как «подчинённые» (Dependents). Создает механизм у класса, который позволяет получать экземпляру объекта этого класса оповещения от других объектов об изменении их состояния, тем самым наблюдая за ними.

Определяет зависимость типа «один ко многим» между объектами таким образом, что при изменении состояния одного объекта все зависящие от него оповещаются об этом событии.

При реализации шаблона «наблюдатель» обычно используются следующие классы:

* **Observable** — интерфейс, определяющий методы для добавления, удаления и оповещения наблюдателей;
* **Observer** — интерфейс, с помощью которого наблюдатель получает оповещение;
* **ConcreteObservable** — конкретный класс, который реализует интерфейс Observable;
* **ConcreteObserver** — конкретный класс, который реализует интерфейс Observer.



Шаблон «наблюдатель» применяется в тех случаях, когда система обладает следующими свойствами:

* существует, как минимум, один объект, рассылающий сообщения;
* имеется не менее одного получателя сообщений, причём их количество и состав могут изменяться во время работы приложения;
* нет надобности очень сильно связывать взаимодействующие объекты, что полезно для повторного использования.

Данный шаблон часто применяют в ситуациях, в которых отправителя сообщений не интересует, что делают получатели с предоставленной им информацией.