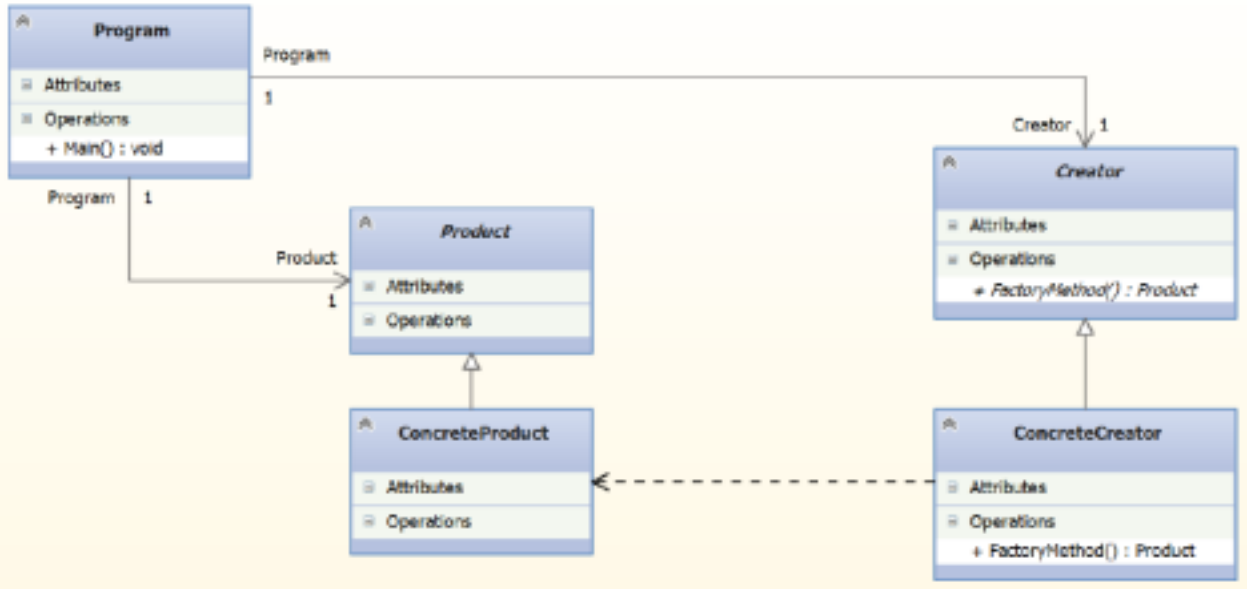
Патерн Factory Method - надає абстрактний інтерфейс (набір методів) для створення об'єкту-продукту, але залишає можливість, розробникам класів, що реалізують цей інтерфейс, самостійно прийняти рішення про те, екземпляр якого конкретного класу-продукту створити. Патерн Factory Method дозволяє базовим абстрактним класам передати відповідальність за створення об'єктів-продуктів своїм похідним класам.

Патерн Factory Method лежить в основі всіх породжують патернів, організовуючи процес породження об'єктів-продуктів. Якщо проектувальник на етапі проектування системи не може відразу визначитися з вибором відповідного патерну для організації процесу породження продукту в конкретній ситуації, то спершу слід скористатися паттерном Factory Method.

Наприклад, якщо проектувальник, не визначився зі складністю продукту або з необхідністю і способом організації взаємодії між декількома продуктами, тоді є сенс спершу скористатися паттерном Factory Method. Пізніше, коли вимоги будуть сформульовані більш чітко, можна буде провести швидку підміну патерну Factory Method на інший твірний патерн, який більш відповідний ситуації.

Важливо пам'ятати, що Factory Method є паттерном рівня класів, і він сфокусований тільки на відносинах між класами. Основним завданням патерну Factory Method є організація техніки делегування відповідальності за створення об'єктів продуктів одним класом (часто абстрактним) іншому класу (похідному конкретному класу).

Структура патерну:



Структура патерну на мові C#:

```
class Program
{
    static void Main()
    {
        Creator creator = null;
        Product product = null;

        creator = new ConcreteCreator();
        product = creator.FactoryMethod();

        creator.AnOperation();
    }
}
```

```
abstract class Creator
{
    Product product;

    public abstract Product FactoryMethod();

    public void AnOperation()
    {
        product = FactoryMethod();
    }
}
```

```
abstract class Product
{
}
```

```
class ConcreteProduct : Product
{
   public ConcreteProduct()
   {
        Console.WriteLine(this.GetHashCode());
   }
}
```

```
class ConcreteCreator : Creator
{
   public override Product FactoryMethod()
   {
       return new ConcreteProduct();
   }
}
```

Патерн Prototype - надає можливість створення нових об'єктів продуктів (клонів), використовуючи техніку клонування (копіювання) створеного раніше об'єкта-оригіналу-продукту (прототипу). Патерн Prototype - дозволяє задати різні види (класи-види) об'єктів-продуктів (клонів), через настройку стану

кожного нового створеного клону. Класифікація клонів-продуктів проводиться на підставі відмінності їх станів.

Патерн Prototype описує процес правильного створення об'єктів клонів на основі наявного об'єкта-прототипу, іншими словами, патерн Prototype описує правильні способи організації процесу клонування.

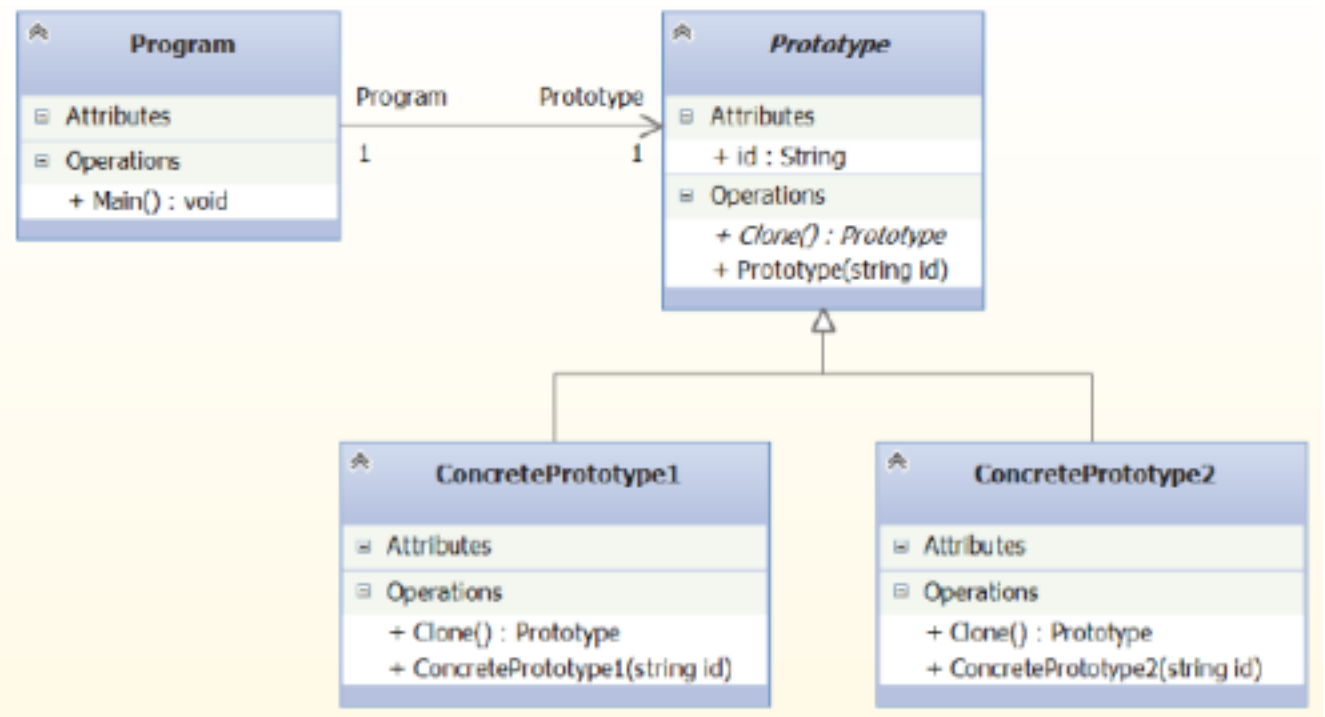
Що таке клонування в об'єктивній реальності? У біології термін «клонування» - позначає процеси копіювання будь-яких живих істот. Але, в інформатиці термін «клонування» має своє специфічне значення, відмінне від його значення в біології. У біології процес клонування відбувається шляхом створення не готовою копії дорослого організму, а шляхом вирощування дорослого клону з немовляти. Спочатку клон-немовля породжується на основі генетичного матеріалу організму-прототипу, і розвивається до дорослого стану поетапно, згідно закладеної в ньому генетичної програми розвитку. В інформатиці ж клонування відбувається «миттєво». У результаті клонування прототипу в інформатиці, виходить відразу «дорослий» клон повністю ідентичний прототипу.

У житті програмному клонуванню аналогію не знайти, так як біологічний організм-клон ніколи не буде ідентичний своєму організму прототипу. Порушення ідентичності відбувається за рахунок втручання в розвиток клону зовнішніх факторів середовища його перебування. З втратою ідентичності у біологічного клону розвивається яскраво виражена індивідуальність. В інформатиці ж, легко домогтися ідентичності клону і прототипу за рахунок можливості копіювання всього стану прототипу в клон. Але можна відійти від підходу до клонування із суворим контролем

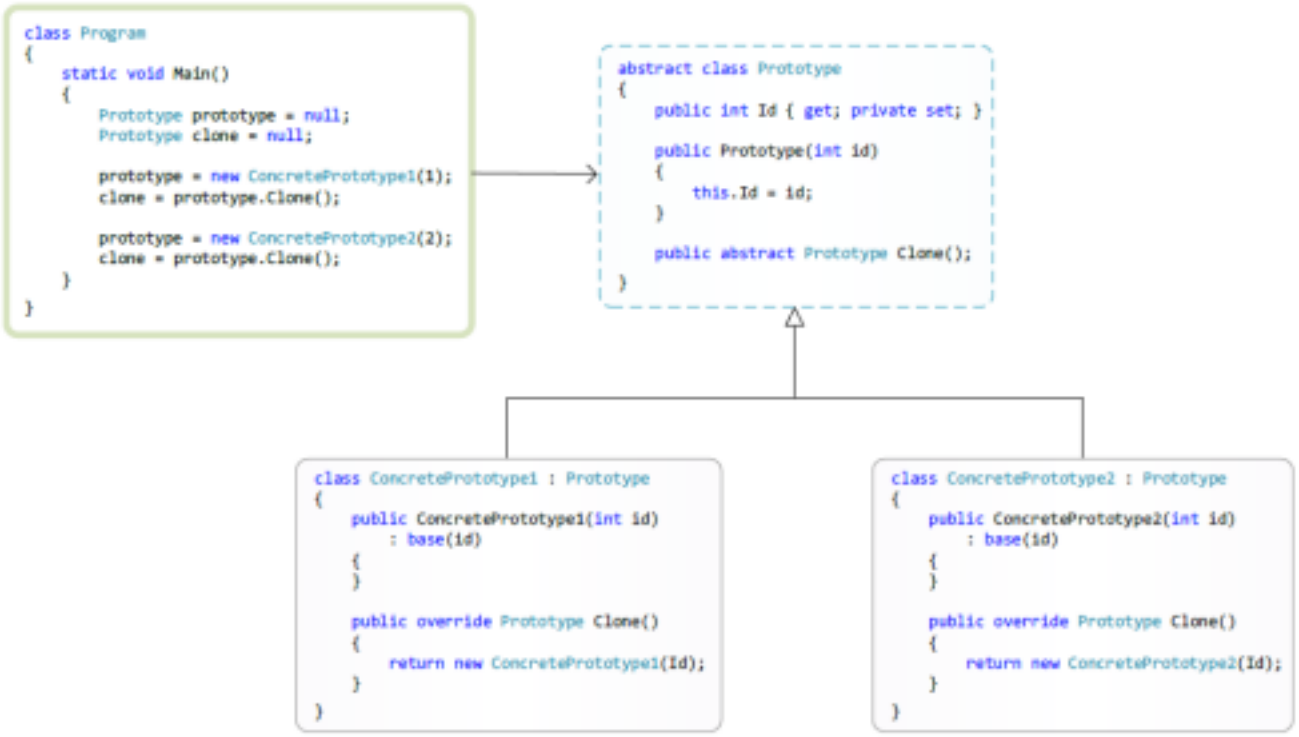
ідентичності і представити клон, як систему, створену за зразком іншої. Клон повинен зберігати основні властивості вихідної системи-прототипу не руйнуючи абстракції (генотипу) прототипу, тобто збірного поняття вихідної системи. Індивідуальні риси (фенотип), які в процесі розвитку набуває клон, не повинні руйнувати абстракцію (генотип) прототипу і такими індивідуальними рисами клону можна знехтувати.

Наприклад, при клонуванні вівці Доллі, сама Доллі і вівця-донор мали стовідсоткову ідентичність генотипу, але не мали стовідсоткової ідентичності фенотипу. Іншими словами, і донор-вівця і клон-вівця були вівцями-близнюками однієї породи (збереження генотипу), а не вівцями різних порід, але у них міг відрізнятися вага, зріст, відтінок вовни, а також набутий досвід визначальний індивідуальну поведінку (різниця фенотипів).

Структура патерну:



Структура патерну на мові C#:



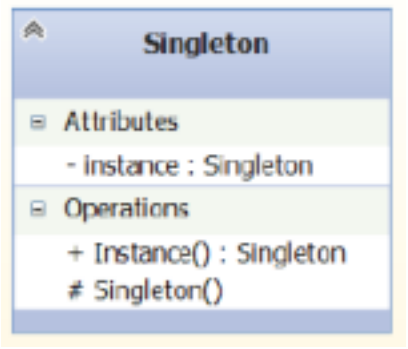
Патерн Singleton - гарантує, що у класу може бути тільки один екземпляр. В окремому випадку надається можливість наявності, заздалегідь визначеного числа примірників.

У реальному житті аналогією об'єкта в єдиному екземплярі може служити Сонце. Як би ми не дивилися на нього, ми завжди будемо бачити одне й те ж Сонце - зірку, навколо якої обертається планета Земля, так як не може бути іншого примірника Сонця для нашої планети.

Важливо зауважити, що аналогія з Сонцем не є повною відносно патерну Singleton. Так як жоден об'єкт в реальному житті не може контролювати процес свого породження, а, отже, і існування себе в єдиному екземплярі. Така можливість з'являється тільки у віртуальних об'єктів програмних систем.

При розробці додатків буває необхідно, щоб існував тільки один екземпляр певного класу. Наприклад, в іграх від першої особи персонаж анімат, яким управляє гравець, повинен бути одним єдиним, оскільки такий анімат, є образом-проекцією, живого гравця на ігровий світ.

Структура патерну:



Структура патерну на мові C#:

```
class Program
{
    static void Main()
    {
        Singleton instance1 = Singleton.Instance();
        Singleton instance2 = Singleton.Instance();
        Console.WriteLine(ReferenceEquals(instance1, instance2));

        instance1.SingletonOperation();
        string singletonData = instance1.GetSingletonData();
        Console.WriteLine(singletonData);
    }
}
```

```
class Singleton
{
    static Singleton uniqueInstance;
    string singletonData = string.Empty;

    protected Singleton()
    {
    }

    public static Singleton Instance()
    {
        if (uniqueInstance == null)
            uniqueInstance = new Singleton();

        return uniqueInstance;
    }

    public void SingletonOperation()
    {
        singletonData = "SingletonData";
    }

    public string GetSingletonData()
    {
        return singletonData;
    }
}
```

Варіант №1

Салон з продажу автомобілів працює за системою гнучких знижок і розрізняє три види клієнтів: ті що купують автомобілі одразу з повною оплатою; ті, що купують автомобілі у кредит та клієнти, які купують автомобіль з розтермінуванням. Кожен з трьох видів клієнтів отримує свою кінцеву ціну на авто, свою схему страхування та свій варіант гарантійного обслуговування. При оформленні документів на автомобіль в БД автосалону зберігаються персональні дані покупця, інформація про модель, марку, комплектацію та ціну авто, варіанти страховки та сервісного обслуговування. Реалізувати шаблон проектування Фабричний метод на наведеному прикладі.

Варіант №2

Написати програму, яка б реалізувала сортування одновимірного масиву за допомогою трьох алгоритмів: сортування бульбашкою, сортування Шелла та швидке сортування. Кожен алгоритм виконується в окремому потоці, а час сортування записується у один спільний для трьох алгоритмів файл. На даному прикладі реалізувати шаблон проектування Одинак з лінивою ініціалізацією.

Варіант №3

Розробити програму, яка зберігає в контейнері об’єкти будинків (багатоквартирні та котеджі). Кожен будинок характеризується площею, кількістю поверхів, адресою та персональними даними власника (або власників квартир для багатоквартирного будинку). Програма повинна надавати можливість переглядати інформацію про попередньо збережені будинки та корегувати її. Заповнення контейнера об’єктів здійснювати опираючись на шаблон проектування Прототип.

Варіант №4

Продемонструвати шаблон проектування Одинак з ранньою ініціалізацією на наступному прикладі. Розробити програму, в якій існує лічильник для підрахунку кількості проданих товарів трьох груп(продукти харчування, ліки, одяг). Вартість продукту визначається випадковим чином з надбавкою 5% на продукти харчування, 10% на ліки та 15% на одяг. Окрім кількості проданих товарів об’єкт лічильника повинен
зберігати сумарну вартість проданих товарів та файл, в якому повинен
записуватись час продажу кожного товару і найменування групи товарів.

Варіант №5

Розробити програму для обліку кімнат в гуртожитку. Кожна кімната може бути двомісною або трьохмісною. Необхідно зберігати інформацію про тип

кімнати та всіх її мешканців: ПІБ, дата народження, факультет, група, форма навчання(державне замовлення чи за кошти фізичних осіб). Комендант гуртожитку повинен формувати звіт у вигляді текстового файлу про кожну кімнату(тип кімнати, кількість мешканців, квартплата для кожного мешканця та інформація про всіх мешканців). Поставлене завдання реалізувати за допомогою шаблону проектування Прототип.

Варіант №6

За допомогою шаблону проектування Фабричний метод написати програму, яка б реалізувала прорисовку геометричних фігур. Користувач обирає фігуру через користувацький інтерфейс та вибирає місце прорисовки на формі. Усі геометричні фігури мають однакові можливості: обчислення площі, відображення кольору та виводу координат центру описаного кола.