УДК 004.43 **Информационные технологии**

Поляков Павел Сергеевич, *студент факультета довузовской подготовки и среднего профессионального образования, ФГБОУ ВО «Мордовский государственный университет им. Н. П. Огарёва»*
e-mail: paw.polyakov2010@yandex.ru
Росляева Мария Николаевна, *студентка факультета довузовской подготовки и среднего профессионального образования, ФГБОУ ВО «Мордовский государственный университет им. Н. П. Огарёва»*
e-mail:mar.rosliaewa@yandex.ru
Прокин Александр Александрович, *преподаватель факультета довузовской подготовки и среднего профессионального образования, ФГБОУ ВО «Мордовский государственный университет им. Н. П. Огарёва»*
e-mail: aaprokin90@yandex.ru

НАСЛЕДОВАНИЕ КАК ВАЖНЕЙШИЙ ПРИНЦИП ЯЗЫКА C#

Аннотация: Возможность многократного использования - одно из важнейших преимуществ языка программирования C#.

Классы C# могут быть повторно использованы несколькими способами. После создания родительского (базового) класса он может быть изменен другим программистом в соответствии с его требованиями. Основная идея наследования – создание новых классов, повторное использование свойств существующего базового класса. Механизм получения нового класса из существующего класса (Base / Parent Class) называется наследование [1].

«Старый» класс называется базовым (родительским), а новый – производным (дочерним) или подклассом. Производный класс включает в себя все функции базового класса, а затем добавляет качества, специфичные для производного класса.

В этой статье рассматривается изучение концепции наследования и ее типов с использованием C#.

Ключевые слова: наследование, классы C#, виды наследования.

Annotation: Reusability is one of the most important benefits of the C # programming language.

C # classes can be reused in several ways. After creating the parent (base) class, it can be changed by another programmer in accordance with its requirements. The basic idea of inheritance is the creation of new classes, the reuse of the properties of an existing base class. The mechanism for obtaining a new class from an existing class (Base / Parent Class) is called inheritance [1].

The "old" class is called the base (parent), and the new - the derived (child) or subclass. The derived class includes all the functions of the base class, and then adds the qualities specific to the derived class.

This article explores the concept of inheritance and its types using C #.

Keywords: inheritance, C # classes, types of inheritance.

Объектно-ориентированное программирование прекрасно, и одним из его самых замечательных атрибутов является возможность создавать новые классы, которые наследуют все методы и свойства существующих классов.

На сегодняшний день могут возникнуть некоторые опасения по поводу использования наследования и даже возникают утверждения, что это не особо безопасная конструкция. Однако, многие программисты используют наследование в своих проектах, потому что это намного лучше, чем копирование и вставка, которые используются в интерфейсном подходе [3].

C # использует очень стандартный подход к объектам. В разработке можно спокойно создать базовый класс, а на его основе – производный.

И даже если производный класс будет пуст, у него все еще будут присутствовать все свойства и методы базового класса.

Цель данной исследовательской работы – рассмотреть концепцию наследования в объектно-ориентированных языках на примере C#. Обзорная работа начинается с изучения наследования и возможности повторного использования объектно-ориентированного языка. Наследование играет важную роль для повторного использования кода. Поскольку объектно-ориентированный язык получил широкое признание как технология, которая будет поддерживать создание программного обеспечения многократного использования, особенно из-за функции наследования.

Функции определяются не просто по имени, а по типам их параметров или сигнатуры. Проще говоря, можно определить множество методов с одинаковыми именами, если их списки параметров, т.е. их сигнатура, разные. Стоит отметить, что возвращаемый тип не играет роли в подписи.

Если вызывается перегруженная функция, то фактически используется определение функции с наиболее конкретным соответствием подписи вызова.

В процессе выбора используется иерархия наследования, которая определяет, какую функцию следует использовать. В общем, если несколько перегруженных функций соответствуют сигнатуре вызова, то та, которая имеет самые высокие параметры, т.е. более выше в дереве наследования используется.

Так же, для полного понимания описанной темы, стоит продемонстрировать концепцию наследования в повседневной жизни. Без изменения предыдущих данных новые функции могут быть легко добавлены в класс. Как и человек, дочерний класс может автоматически получать доступ ко всем свойствам родительского класса, как ребенок получает генные данные и какие-то привычки от своих родителей. Так же, в дочерний класс также можно вводить новые методы и параметры, как в будущем человек обретает черты характера, никак не похожие на родительские.

Типы Наследования

1. Наследование одного уровня

Если производный класс создается только из одного базового класса, то такое наследование называется одноуровневым наследованием. На рисунке 1 показан простой пример одиночного наследования. Приведенная диаграмма показывает принцип одиночного наследования. Класс A является базовым классом, а класс B считается производным классом. Класс B имеет все свои собственные свойства, а также свойства базового класса (т.е. A).

Рисунок 1 — Одноуровневое наследование

2. Множественное наследование

Если производный класс создается из более чем одного базового класса, такое наследование называется множественным наследованием. На рисунке 2 показан простой пример множественного наследования.

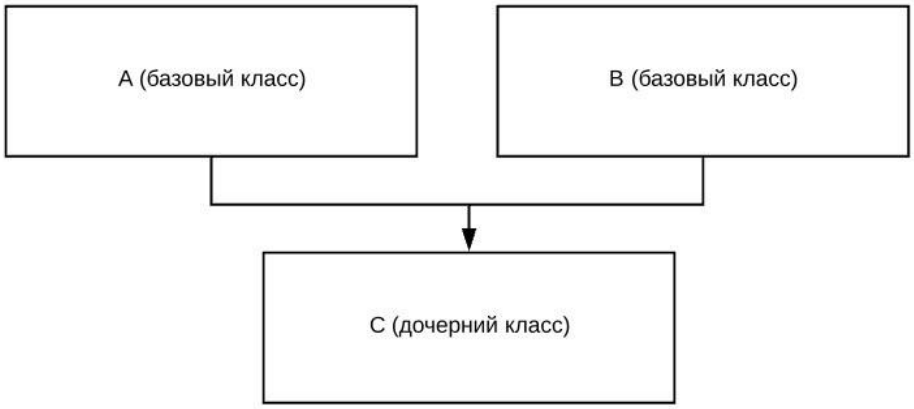

Рисунок 2 — Множественное наследование

Приведенная выше диаграмма показывает множественное наследование. Класс A является базовым классом, а класс B также является базовым классом, а класс C создается из двух базовых классов A и B. Поэтому класс C называется производным классом. Класс C имеет все свои собственные свойства, а также свойства классов A и B. Множественное наследование позволяет объединять свойства нескольких базовых классов в качестве отправной точки для определения новых классов. Это подобно тому, как ребенок наследует некоторые свойства отца и матери обоих.

3. Иерархическое наследование

Если из одного базового класса создано более одного производного класса, такое наследование называется иерархическим. На рисунке 3 показан простой пример иерархического наследования.

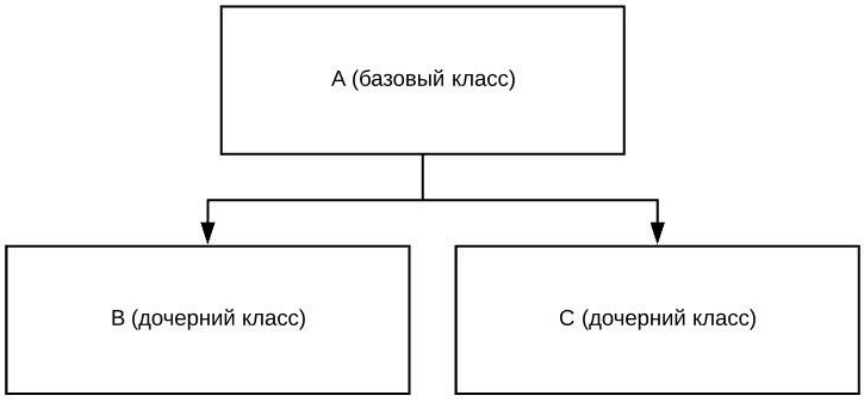

Рисунок 3 — Иерархическое наследование

Приведенная выше диаграмма показывает иерархическое наследование. Класс A является базовым классом, а класс B и класс C являются производными классами. Классы C и B имеют собственные независимые свойства, а также свойства класса A.

4. Многоуровневое наследование

Если производный класс создается из другого производного класса (промежуточного базового класса), то такое наследование называется многоуровневым наследованием. На рисунке 4 показан простой пример многоуровневого наследования.

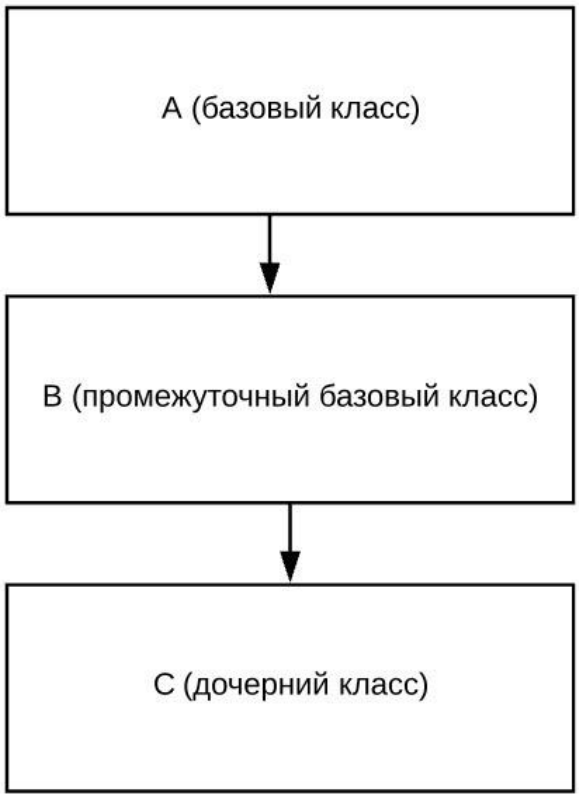

Рисунок 4 — Иерархическое наследование

Приведенная выше диаграмма показывает многоуровневое наследование. Класс A является базовым классом, а класс B создан из класса A. Класс B имеет свои собственные свойства, а так же свойства класса A.

Класс C создан из класса B (промежуточный базовый класс), поэтому класс C обладает собственными свойствами, а также свойствами как класса A, так и класса B.

5. Гибридное наследование

Комбинация одного или нескольких наследований называется гибридным наследованием. На рисунке 5 можно увидеть схему такого наследования.

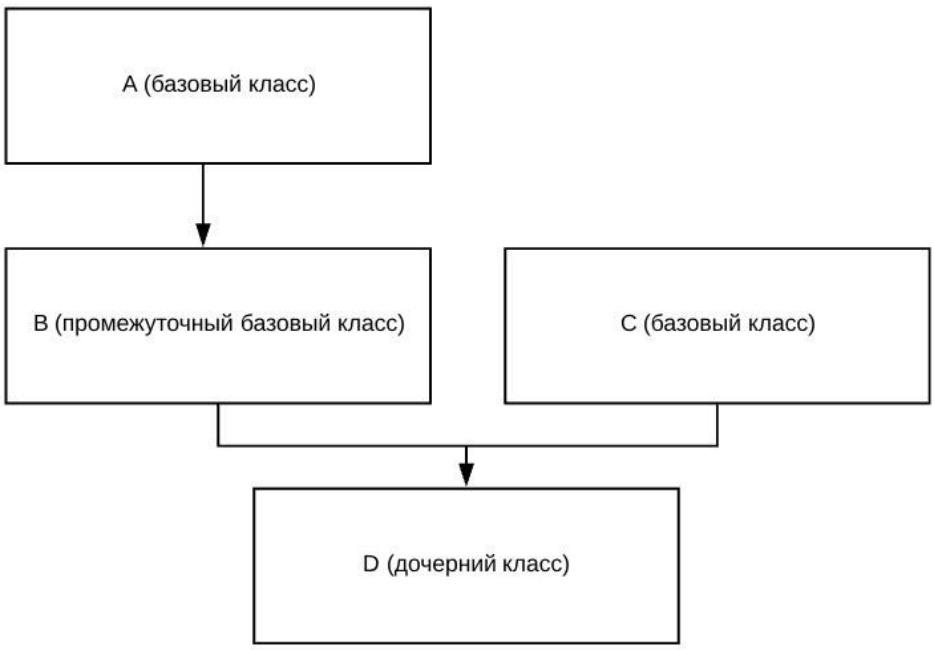

Рисунок 5 — Гибридное наследование

Вывод

Механизм (процесс) получения нового класса из существующего (старого) класса называется наследованием. Используя наследование, можно повторно использовать возможности существующего класса, и это является наиболее важной концепцией в C#. Все наследование имеет свои особенности и его использование для предоставления пользователям возможности многократного использования концепций, экономии времени и уменьшения сложности.

Библиографический список:

1. Вайсфельд М. Объектно-ориентированное мышление. — СПб.: Питер, 2014. — 304 с.: ил. — (Серия «Библиотека программиста) ISBN 978-5-496-00793-1.

2. Васильев Алексей C#. Объектно-ориентированное программирование / Алексей Васильев. - М.: Питер, 2012. - 320 с.

3. Макконнелл Стив. Совершенный код. Практическое руководство по разработке программного обеспечения — Русская Редакция, Питер, 2005. -896 с.

4. Комлев, Николай Юрьевич Объектно Ориентированное Программирование. Хорошая книга для Хороших Людей / Комлев Николай Юрьевич. - М.: Солон-Пресс, 2014. - 770 с.