МЕТОДИКА ФОРМИРОВАНИЯ ПОНЯТИЯ «МЕТОД» У НАЧИНАЮЩИХ ПРОГРАММИСТОВ C#: ОТ ПРОЦЕДУРНОГО МЫШЛЕНИЯ К ОБЪЕКТНО-ОРИЕНТИРОВАННОМУ ПРОГРАММИРОВАНИЮ

Голубков Н.Д., студент 1 курса ПИ

Тазетдинова Ю.А., к.ф.-м.н., доцент

Тазетдинов Б.И., к.ф.-м.н., доцент

г. Бирск, Бирский филиал Уфимского университета науки и технологий

Аннотация. Данная статья посвящена преодолению ключевой методической проблемы у начинающих программистов — перехода от процедурного мышления к объектно-ориентированному через формирование целостного понятия «метод» в C#. Предлагаемая автором методика выстраивает поэтапный путь освоения этой концепции: от базового синтаксиса до сложных аспектов, таких как перегрузка и работа с исключениями, что позволяет студентам осознать роль метода как фундаментального элемента архитектуры приложения. Такой системный подход обеспечивает не просто заучивание синтаксических конструкций, а формирование глубокого понимания принципов ООП, в итоге подготавливая специалистов, способных

создавать качественное и поддерживаемое программное обеспечение.

Ключевые слова: программирование C#, методы, объектно-ориентированное программирование, синтаксис методов, экземплярные и статические методы, перегрузка методов, обработка исключений.

Введение. Современная парадигма программирования прочно основана на концепции объектно-ориентированного программирования (ООП), где метод выступает в роли фундаментального механизма инкапсуляции поведения объектов. Однако для начинающего программиста, чье мышление часто носит линейно-процедурный характер, переход к пониманию метода как неотъемлемой части объекта представляет значительную методическую сложность. Формирование понятия «метод» в C# становится критически важным этапом, который либо закладывает прочный фундамент для дальнейшего освоения сложных концепций, либо создает непреодолимые барьеры в понимании самой сути программирования.

Проблема методического характера заключается не столько в изучении синтаксиса объявления методов, сколько в формировании у студентов целостного представления о роли методов в архитектуре приложения. Традиционный подход, фокусирующийся исключительно на технических аспектах, оказывается недостаточно эффективным. Студенты, успешно освоившие синтаксис методов, зачастую не могут применить эти знания для решения практических задач, не понимают принципов их композиции и не видят стратегических преимуществ их использования.

В данной работе предлагается комплексная методика формирования понятия «метод», которая рассматривает эту концепцию как многоуровневую систему, последовательно развивающуюся от простых процедурных конструкций к сложным объектно-ориентированным паттернам. Здесь выделены ключевые темы, образующие логическую цепочку обучения. Каждая тема не только раскрывает технические аспекты C#, но и формирует определенный пласт мышления программиста.

Особое внимание уделяется методическим переходам между темами. Изучение синтаксиса и структуры методов создает техническую базу; понимание различий между экземплярными и статическими методами подводит к осознанию принципов ООП;перегрузка методов демонстрирует полиморфизм в действии; работа с областями видимости и исключениями формирует навыки создания надежных приложений.

Предлагаемая методика основана на принципе «от конкретного к абстрактному»: каждое новое понятие вводится через решение практической проблемы, с которой студент мог столкнуться на предыдущем этапе обучения. Такой подход позволяет не только усвоить отдельные темы, но и сформировать целостное представление о месте и роли методов в современной разработке на C#, обеспечивая плавный переход от процедурного мышления к объектно-ориентированному программированию.

Цель работы– представить систематизированную методику, которая может быть непосредственно применена в образовательном процессе, учитывая как когнитивные особенности начинающих программистов, так и требования современной индустрии разработки программного обеспечения.

Синтаксис и структура методов. Метод в C# – это функция, которая принадлежит классу. Он имеет имя, возвращаемый тип, параметры и тело. Синтаксис объявления метода выглядит следующим образом:

```
[модификатор_доступа] [static]  возвращаемый_тип Имя Метода(параметры)

{

// Тело  метода

}
```

Модификатор доступа определяет видимость метода (public, private, protected и т.д.), static если метод статический, его можно вызывать без создания экземпляра класса, возвращаемый тип – тип данных, который метод возвращает (например, int, string, void – если ничего не возвращает), параметры – входные данные, разделенные запятыми.

Пример простого метода

```
Publicint Sum(int a, int  b)

{return a + b;}
```

Этот метод принимает два целых числа и возвращает их сумму. Вызов метода

```
Int result = Sum(5,3); //  result = 8.
```

Методы могут иметь параметры по умолчанию, что делает их более гибкими

```
public void static PrintMessage(string message = "Hello, World!")

{Console.WriteLine(message);}
```

Здесь параметр message имеет значение по умолчанию, по этому метод можно вызвать как PrintMessage()или PrintMessage("Custom message").

Параметры по умолчанию в C# могут быть полезны, например, при работе с анкетными данными, где большинство полей имеют типичные значения, но иногда требуются исключения. Рассмотрим практические примеры из разных сфер. В системах сбора контактной информации часто встречается ситуация, когда нужно сохранить данные клиента. Большинство людей проживают в одной стране, не имеют второго телефона или отчества. Вместо того чтобы постоянно передавать одни и те же значения, можно создать метод с параметрами по умолчанию. Например, метод создания контакта может принимать обязательные параметры – имя, фамилию и основной телефон, а страну проживания, второй телефон и отчество сделать необязательными со значениями по умолчанию. Это позволяет быстро добавлять типичных клиентов, просто указывая основные данные, а для иностранных клиентов или людей с двумя телефонами – передавать дополнительные параметры.

Экземплярные и сатические методы. Экземплярные методы принадлежат объекту класса. Для их вызова нужно создать экземпляр класса.

```
class Calculator

{

public int Add(int x, int  y)

{

return x + y;

    }

}
```

```
Calculator calc = new Calculator();

int sum = calc.Add(2, 3); // 5
```

Статические методы не требуют экземпляра. Они вызываются через имя класса и часто

используются для утилитарных функций.

```
class MathUtils

{

public static double Square(doublenum)

{

return num * num;

    }

}
```

```
double sq = MathUtils.Square(4); // 16
```

Понимание различий между статическими и экземплярными методами является фундаментальным для эффективного проектирования классов в C#. Статические методы, принадлежащие классу в целом, оптимальны для реализации утилитарных функций и операций, не зависящих от состояния конкретного объекта. Напротив, экземплярные методы, связанные с отдельным объектом, обеспечивают работу с его индивидуальным состоянием и поведением. Это разделение позволяет создавать

логичную и эффективную архитектуру приложения, где статические методы служат для общей функциональности, а экземплярные – для специфического поведения объектов. Грамотное применение обоих типов методов способствует написанию чистого, поддерживаемого и объектно-ориентированного кода.

Перегрузка методов. Перегрузка позволяет создавать несколько методов с одинаковым именем, но разными параметрами. Компилятор выбирает подходящий вариант на основе типов аргументов.

```
publicint Multiply(int a, intb)

{

return a * b;

}

public int Multiply(int a, int  b, int c)

{

return a * b * c;
```

}

Multiply(2, 3) // 6

Multiply(2, 3, 4) // 24.

Перегрузка методов представляет собой мощный механизм в C#, позволяющий создавать несколько методов с одинаковым именем, но различными параметрами. Ключевая ценность перегрузки заключается в способности предоставить единый интерфейс для выполнения логически родственных операций над различными типами данных или с разным количеством аргументов. Это не только улучшает читаемость кода, избавляя разработчика от необходимости запоминать множество различных имен методов для схожих действий, но и способствует созданию интуитивно понятных API. Важно отметить, что перегрузка демонстрирует принцип полиморфизма на этапе компиляции, когда компилятор самостоятельно определяет, какая именно версия метода должна быть вызвана на основе переданных аргументов. Таким образом, грамотное использование перегрузки методов является важным шагом на пути к созданию гибкого, поддерживаемого и профессионального кода, соответствующего принципам объектно-ориентированного программирования.

Параметры-значения и параметры-ссылки в методах. В C# существует два принципиально разных способа передачи параметров в методы: по значению и по ссылке.

Параметры-значения – это способ по умолчанию. Когда вы передаете переменную таким образом, метод получает только копию ваших данных. Что бы метод ни делал с этой копией, это никак не повлияет на вашу исходную переменную. Это похоже на то,

как если бы вы отправили другу фотографию документа – он может делать с фотографией что угодно, но оригинальный документ у вас останется неизменным.

Параметры-ссылки (с модификаторами ref или out) работают иначе. В этом случае метод получает не копию, а доступ к оригинальной переменной. Это можно сравнить с тем, что вы даете другу не фотографию документа, а сам документ – любые изменения, которые он сделает, останутся в оригинале.

Ключевые различия между параметрами-значениями и параметрами-ссылками заключаются в следующем: параметры-значения безопасны – вы защищены от случайного изменения своих данных, тогда как параметры-ссылки эффективны для больших объектов, так как не создают копий. Кроме того, параметры-ссылки позволяют методу возвращать несколько результатов.

Когда что использовать: используйте параметры-значения для простых данных и когда метод не должен менять исходные переменные. Используйте параметры-ссылки когда нужно изменить исходную переменную или когда передача копии больших данных неэффективна. Правильное понимание этих механизмов помогает писать более эффективный и безопасный код в C#.

Работа с иключениями. Методы могут генерировать исключения. Используйте try-catch для обработки ошибок.

```
public int Divide(int a, int  b)

{
```

```
if(b == 0)

throw new DivideByZeroException("Делениенаноль!");

return a / b;

}




try

{

int result = Divide(10, 0);

}

catch(DivideByZeroException  ex)

{

Console.WriteLine(ex.Message);
```

}

Изучайте распространенные исключения и пишите тесты для проверки методов.

Заключение. Подводя итоги, можно утверждать, что проблема формирования понятия «метод» у студентов-программистов лежит не в плоскости запоминания синтаксиса, а в необходимости перестройки их мышления. Традиционные подходы, фокусирующиеся на технических аспектах, не способны решить эту задачу, что приводит к формальному знанию без понимания сути.

В качестве решения предлагается методика, которая выстраивает обучение как последовательную цепочку, где каждая тема логически вытекает из предыдущей и решает конкретную практическую проблему. Такой путь – от написания простой процедуры до использования методов как части объектов – обеспечивает естественный и осознанный переход от процедурного подхода к объектно-ориентированному.

Главный результат внедрения данной методики – это студент, который не просто «знает», что такое метод, а понимает, зачем он нужен,как методы взаимодействуют в рамках приложения и какие архитектурные преимущества они дают. Это превращает изучение C# из освоения набора правил в осмысленное овладение инструментом для создания качественного, надежного и масштабируемого программного обеспечения, что в полной мере отвечает требованиям современной IT-индустрии.

Литература

1. Павловская Т.А. C#. Программирование на языке высокого уровня. – СПб.: Питер, 2009. – 432 с.

2. Информационно-справочная система ресурса Microsoft. URL: https://msdn.microsoft.com/uk-ua/library/ms173121(v=vs.90).aspx (дата обращения: 08.10.2025).

3. Ресурс METANIT.COM. URL: https://metanit.com/sharp/tutorial/ (дата обращения: 15.10.2025).

4. Ресурс internetурок. URL: https://interneturok.ru/lesson/algebra/podgotovka-k-ege/tema-2-uravneniya-i-neravenstva/kvadratnye-uravneniya-lineynye-

uravneniya
-
praktika
(дата обращения: 16.10.2025).