Частное учреждение образования

«Колледж бизнеса и права»

УТВЕРЖДАЮ

Заместитель директора

по учебной работе

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ И.В.Малафей

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_ 2024

**КОНСТРУИРОВАНИЕ ПРОГРАММ И ЯЗЫКИ ПРОГРАММИРОВАНИЯ**

Вопросы к экзамену для учащихся 3 курса дневной формы получения образования специальности 2-40 01 01 «Программное обеспечение информационных технологий»

Составлены на основании учебной программы по учебному предмету «Конструирование программ и языки программирования», утвержденной директором ЧУО «Колледж бизнеса и права» 06.07.2021

**Теоретические вопросы**

1. Опишите основные этапы обработки программы на основе платформы .NET. Опишите принцип компиляции программ под управлением среды CLR с использованием JIT-компилятора. Напишите код заготовки консольной программы и прокомментируйте все ее части.
2. Опишите каркас платформы .NET. Нарисуйте структуру платформы .NET. Перечислите основные составляющие алфавита языка С#.
3. Опишите каждую составляющую алфавита языка С#. Дайте определение понятию лексем в языке С#. Напишите фрагмент кода программы для расчета значения по заданной формуле



1. Дайте определение понятию переменной. Объясните разницу между объявлением и инициализацией переменной. Опишите особенности названия переменных в С#. Приведите примеры.
2. Поясните что определяет тип данных. Что такое var? Перечислите типы данных в C# с указанием системного типа.
3. Опишите процесс явного и неявного преобразования типов в языке C#. Перечислите все безопасные преобразования типов в С#. Для переменных int b = 300; long a; byte d; осуществите преобразование типа с потерей и без потери данных.
4. Классифицируйте по группам операции языка C#. Опишите принцип работы тернарной условной операции. С помощью тернарного оператора найдите максимальное из двух целых чисел.
5. Перечислите операции, относящиеся к арифметическим. Приведите примеры операций присваивания. Поясните в чем заключается разница между унарными операциями: (++а)-5 и (а++)-5. Вычислите результат при а=10.
6. Перечислите все унарные операции языка C# и приведите примеры. Перечислите битовые операции. Объясните по какому принципу заполняются освобождающиеся позиции при выполнении операций сдвига. Опишите на примере результат выполнения операций &, | , ^ для выражения: а=b…c, если b=1010, c= 0110.
7. Поясните синтаксис и особенности использования условного оператора if в С#. С помощью условного оператора написать программу для расчета корней квадратного уравнения.
8. Поясните синтаксис и особенности использования оператора выбора switch-case-default в С#. Приведите фрагмент программы, позволяющей узнать пору года при вводе числа (месяца) от 1 до 12 (1 – зима, 2 – зима, 3 – весна…). В случае ввода не верных значений выбрасывать ошибку.
9. Поясните синтаксис и особенности использования операторов цикла while и do-while в С#. Опишите разницу между оператором цикла с пред- и постусловием. С помощью операторов цикла while или do-while написать программу для суммы цифр числа N. Число N может иметь любое количество знаков.
10. Поясните синтаксис и особенности использования операторов цикла for и foreach в С#. Опишите разницу между циклами for и foreach. С помощью операторов цикла for или foreach написать программу для суммы всех четных двузначных чисел.
11. Для чего нужны операторы передачи управления? Поясните особенности использования операторов continue, break, go-to и return в С#. С помощью операторов управления найдите в целочисленном массиве сумму между первым положительным и последним нулевым элементами.
12. Опишите назначение методов класса Random. Опишите синтаксис получения псевдослучайной последовательности целых и дробных чисел с изменением и без изменения базы генерации. Напишите фрагмент программы заполнения double-массива из пяти элементов случайными целыми числами в диапазоне от -0,5 до 7,3.
13. Дайте определение понятию массив. Перечислите способы объявления одномерного массива. Напишите код для инициализации одномерного целочисленного массива значениями с консоли. Составьте программу, которая найдет в одномерном массиве минимальной положительный элемент. Значение найденного элемента выведите на консоль.
14. Дайте определение понятию длина массива и индекс элемента массива. Перечислите способы объявления двумерного массива. Напишите код для инициализации двумерного целочисленного массива значениями с консоли. Составьте программу, которая найдет в двумерном массиве сумму элементов на главной диагонали. Значение суммы выведите на консоль.
15. Опишите назначение класса Array. Приведите примеры его методов и/или свойств и укажите их назначение. Составьте программу с использованием методов класса Array, которая найдет в одномерном массиве сумму элементов, расположенных между первым и последним нулевым элементами. Значение суммы выведите на консоль.
16. Дайте определение понятию «метод». Опишите разницу между функцией и методом. Укажите назначение параметра static для методов. Опишите статический метод int SumOfDigits(int n), который принимает некоторое целое число (его длина может быть любой) и возвращает сумму его цифр. Продемонстрируйте в Main вызов и результат работы этого метода. Продемонстрировать вызов метода в Main.
17. Перечислите способы передачи параметров в методах. Укажите разницу между передачей параметров по ссылке и по значению. Поясните разницу между передачей ref- и out-параметра в метод. Опишите статический метод, который меняет первый и последний элемент в массиве. Продемонстрируйте в Main вызов и результат работы этого метода.
18. Перечислите особенности работы со строками в классе String. Приведите примеры его методов и/или свойств и укажите их назначение. Составьте программу, которая посчитает количество цифр в заданной строке (string). Найденное количество выведите на консоль.
19. Перечислите особенности работы со строками в классе StringBuilder. Приведите примеры его методов и/или свойств и укажите их назначение. Составьте программу, которая добавляет к исходной строке (StringBuilder) слово «Hello». Измененную строку выведите на консоль
20. Опишите назначение класса Regex. Приведите перечень категорий символов, операторов и конструкций, которые можно использовать для задания регулярных выражений. Интерпретируйте следующие шаблоны:

\b.\*[.?!;:](\s|\z),

gr[ae]y\s\S+?[\s|\p{P}],

\b[A-Z]\w\*\b,

\bth[^o]\w+\b,

\w)\1,

\b(\w+)(\W){1,2},

\b\w+(e)?s(\s|$),

\b(\S+)\s?,

^\D\d{1,5}\D\*$,

^(\(?\d{3}\)?[\s-])?\d{3}-\d{4}$,

^[0-9-[2468]]+$.

1. Опишите методы для работы с регулярными выражениями в С#. Перечислите основные управляющие символы для построения регулярных выражений. Постройте регулярное выражение для вывода IP-адреса и телефонного номера в формате +375(29|44|25|17)ххх-хх-хх.
2. Опишите синтаксис объявления структуры. Перечислите особенности использования и инициализации объектов структур на языке С#. Создайте структуру для описания класса Человек, полями которого будут имя и возраст, укажите конструктор и метод вывода данных о созданном объекте структуры. Продемонстрируйте работу программы в Main.
3. Для чего предназначена структура DateTime? Перечислите виды форматов даты. Перечислите ее основные методы и свойства. Считайте с компьютера текущую дату и найдите полное количество лет человеку, у которого день рождения 23.08.2000.
4. Дайте определение понятиям «класс» и «объект». Перечислите элементы, которые могут содержаться в классе. Для чего используется ключевое слово this? Приведите пример использования this в конструкторе и в методе класса.

Класс:

Класс — это шаблон для создания объектов в программе.

Он определяет, как объекты данного типа будут вести себя и какие данные они будут содержать.

Класс — это логическая сущность, описывающая общие характеристики объектов.

В классе могут содержаться следующие элементы:

Поля (переменные-члены): Хранят данные, состояние объекта.

Методы (функции-члены): Определяют поведение объекта.

Конструкторы: Инициализируют объект при его создании.

Свойства: Позволяют управлять доступом к полям.

Индексаторы: Позволяют обращаться к объекту как к массиву.

Вложенные типы: Классы, перечисления и интерфейсы, объявленные внутри другого класса.

Объект:

Объект — это экземпляр класса.

Он представляет физическую сущность, созданную на основе класса.

Объекты имеют свои собственные значения полей и могут вызывать методы класса.

Ключевое слово this:

this — это неявная ссылка на текущий экземпляр класса.

Применяется в следующих случаях:

Когда в методе класса существует имя, совпадающее с именем поля класса.

Для вызова конструкторов класса (например, перенаправление на главный конструктор).

При передаче методу ссылки на текущий объект.

Пример использования this:

C#

class Car

{

private string color;

public Car(string color)

{

this.color = color; // Использование this для доступа к полю color

}

public void DisplayColor()

{

Console.WriteLine($"Car color: {this.color}");

}

}

// Создание объекта Car

Car myCar = new Car("Blue");

myCar.DisplayColor(); // Выводит: "Car color: Blue"

1. Из чего состоит UML-представление класса? Перечислите и опишите виды связей между классами. Постройте диаграмму классов с полями, методами, событиями, свойствами и др. для классов Колледж, Диплом, Студент, Преподаватель, Оценка, Экзамен, Пересдача, Зав.Отделения, Отчисление, Армия.

Давайте начнем с основных элементов диаграммы классов UML:

Имя класса:

Каждый класс имеет уникальное имя, которое обозначает его.

Имя класса следует начинать с заглавной буквы.

Атрибуты:

Атрибуты — это свойства или переменные, хранящие данные объекта.

На диаграмме классов они располагаются ниже имени класса.

Операции (методы):

Операции — это функции или методы, которые определяют поведение объекта.

Они также отображаются ниже имени класса.

Теперь давайте рассмотрим виды связей между классами:

Ассоциация:

Описывает отношение между двумя классами.

Пример: Студент ассоциирован с Колледжом.

Агрегация:

Описывает “часть-целое” отношение.

Пример: Колледж содержит Студентов.

Композиция:

Также “часть-целое”, но более сильное.

Пример: Диплом состоит из Оценок.

Наследование:

Определяет иерархию классов.

Пример: Преподаватель наследует от Студента.

Зависимость:

Описывает, что один класс зависит от другого.

Пример: Экзамен зависит от Преподавателя.

Теперь построим диаграмму классов:

+---------------------+

| Колледж |

+---------------------+

| - название |

| - адрес |

| + добавитьСтудента()|

+---------------------+

^

|

|

+---------------------+

| Студент |

+---------------------+

| - имя |

| - фамилия |

| + получитьДиплом() |

+---------------------+

^

|

|

+---------------------+

| Диплом |

+---------------------+

| - номер |

| - дата |

| + выдать() |

+---------------------+

^

|

|

+---------------------+

| Преподаватель |

+---------------------+

| - специализация |

| - стаж |

| + провестиЭкзамен() |

+---------------------+

1. Перечислите основные принципы ООП. Опишите каждый из принципов и укажите его назначение. Напишите фрагмент программы, реализующий иерархию классов: Фигура – Квадрат - Прямоугольник. Класс Фигура – абстрактный. В нем опишите абстрактный метод для вывода информации об объекте. Класс Квадрат – наследник класса Фигура. В нем определите поле int a – длина стороны, конструктор с параметрами и реализуйте перегруженный метод для вывода информации об объекте. Класс Прямоугольник – наследник класса Квадрат. В нем определите поле int b – ширина, конструктор с параметрами с использованием конструктора базового класса и реализуйте перегруженный метод для вывода информации об объекте. Продемонстрируйте работоспособность разработанных классов в Main.

Абстракция:

Абстракция позволяет выделить существенные характеристики объекта, скрывая детали реализации.

Она помогает создавать более понятные и гибкие модели данных.

Наследование:

Наследование позволяет создавать иерархии классов, где дочерние классы наследуют свойства и методы от родительских.

Это способ повторного использования кода и создания отношений “является” (например, Квадрат является Фигурой).

Инкапсуляция:

Инкапсуляция скрывает детали реализации объекта, предоставляя только интерфейс для взаимодействия.

Она обеспечивает контроль доступа к данным и методам.

Полиморфизм:

Полиморфизм позволяет использовать один интерфейс для разных типов объектов.

Это способность объектов разных классов реагировать на одинаковые методы.

Теперь давайте реализуем иерархию классов для Фигуры, Квадрата и Прямоугольника:

using System;

abstract class Shape

{

public abstract void DisplayInfo();

}

class Square : Shape

{

private int a;

public Square(int sideLength)

{

a = sideLength;

}

public override void DisplayInfo()

{

Console.WriteLine($"Square: Side length = {a}");

}

}

class Rectangle : Square

{

private int b;

public Rectangle(int sideLength, int width) : base(sideLength)

{

b = width;

}

public override void DisplayInfo()

{

Console.WriteLine($"Rectangle: Side length = {a}, Width = {b}");

}

}

class Program

{

static void Main()

{

Shape square = new Square(5);

square.DisplayInfo();

Shape rectangle = new Rectangle(4, 6);

rectangle.DisplayInfo();

}

}

1. Перечислите принципы SOLID. Опишите каждый из принципов и укажите его назначение. Приведите примеры на каждый из принципов, укажите их преимущества и недостатки.

Принципы SOLID — это основополагающие принципы объектно-ориентированного программирования (ООП), которые помогают создавать гибкий, расширяемый и поддерживаемый код. Давайте рассмотрим каждый из них:

1. **Принцип единственной ответственности (Single Responsibility Principle, SRP)**:
   * Каждый класс должен иметь только одну причину для изменения.
   * Назначение: Разделение функциональности для уменьшения связности и улучшения модульности.
   * Преимущества: Упрощение тестирования, легкость поддержки и расширения.
   * Недостатки: Возможно создание большого количества мелких классов.
   * Пример: Разделение класса “Студент” на “Студент” и “ОтчетСтудента”.
2. **Принцип открытости-закрытости (Open/Closed Principle, OCP)**:
   * Программные сущности (классы, модули) должны быть открыты для расширения, но закрыты для изменения.
   * Назначение: Поддержка добавления новой функциональности без изменения существующего кода.
   * Преимущества: Снижение риска внесения ошибок при изменении кода.
   * Недостатки: Требует абстракций и интерфейсов.
   * Пример: Использование паттерна “Стратегия” для разных способов оплаты.
3. **Принцип подстановки Барбары Лисков (Liskov Substitution Principle, LSP)**:
   * Объекты базового класса должны быть заменяемыми объектами производного класса без нарушения корректности программы.
   * Назначение: Гарантия совместимости подклассов с базовым классом.
   * Преимущества: Повышение надежности и предсказуемости кода.
   * Недостатки: Требует внимательного проектирования и тестирования.
   * Пример: Если есть класс “Птица”, то производные классы “Воробей” и “Сокол” также должны быть “Птицами”.
4. **Принцип разделения интерфейса (Interface Segregation Principle, ISP)**:
   * Клиенты не должны зависеть от интерфейсов, которые они не используют.
   * Назначение: Предотвращение “толстых” интерфейсов.
   * Преимущества: Уменьшение связности, улучшение читаемости кода.
   * Недостатки: Требует дополнительного разделения интерфейсов.
   * Пример: Разделение интерфейса “Работник” на “ИнтерфейсРаботника” и “ИнтерфейсМенеджера”.
5. **Принцип инверсии зависимостей (Dependency Inversion Principle, DIP)**:
   * Зависимости должны строиться на абстракциях, а не на конкретных реализациях.
   * Назначение: Уменьшение связности между компонентами системы.
   * Преимущества: Улучшение тестируемости, гибкости и переиспользуемости.
   * Недостатки: Требует дополнительного уровня абстракции.
   * Пример: Использование интерфейсов для взаимодействия между слоями приложения.
6. Опишите назначение инкапсуляции. Перечислите все возможные спецификаторы доступа к пользовательским классам С# с пояснением их назначения. Укажите спецификатор доступа для полей класса, который используется по умолчанию (если он не указан явно). На примере класса Треугольник, для которого можно рассчитать площадь, покажите использование различных типов доступа.

Инкапсуляция в объектно-ориентированном программировании (ООП) — это принцип, который позволяет объединить данные и методы, работающие с этими данными, внутри класса. [Она обеспечивает безопасность, скрывая внутренние детали реализации объекта от внешнего мира1](https://skillbox.ru/media/code/oop_chast_3_modifikatory_dostupa_inkapsulyatsiya/).

Теперь давайте рассмотрим спецификаторы доступа в C#:

1. **public**:
   * Доступ не ограничен.
   * [Методы и свойства с этим модификатором видны из любой внешней программы2](https://learn.microsoft.com/ru-ru/dotnet/csharp/language-reference/keywords/access-modifiers).
   * Пример: public double CalculateArea() { ... }
2. **protected**:
   * Доступ ограничен текущим классом и производными классами (наследниками).
   * [Полезно при наследовании](https://skillbox.ru/media/code/oop_chast_3_modifikatory_dostupa_inkapsulyatsiya/)[2](https://learn.microsoft.com/ru-ru/dotnet/csharp/language-reference/keywords/access-modifiers).
   * Пример: protected double SideLength { get; set; }
3. **internal**:
   * Доступ ограничен текущей сборкой (assembly).
   * [Полезно для внутренних компонентов](https://skillbox.ru/media/code/oop_chast_3_modifikatory_dostupa_inkapsulyatsiya/)[2](https://learn.microsoft.com/ru-ru/dotnet/csharp/language-reference/keywords/access-modifiers).
   * Пример: internal int NumberOfSides { get; set; }
4. **protected internal**:
   * Доступ ограничен текущей сборкой или производными классами.
   * [Комбинация предыдущих двух модификаторов](https://skillbox.ru/media/code/oop_chast_3_modifikatory_dostupa_inkapsulyatsiya/)[2](https://learn.microsoft.com/ru-ru/dotnet/csharp/language-reference/keywords/access-modifiers).
   * Пример: protected internal void SetSides(int sides) { ... }

[По умолчанию, если не указан ни один спецификатор доступа, используется **internal**](https://skillbox.ru/media/code/oop_chast_3_modifikatory_dostupa_inkapsulyatsiya/)[3](https://studopedia.ru/2_6636_spetsifikatori-dostupa.html).

Теперь давайте создадим класс Triangle и продемонстрируем использование различных типов доступа:

**C#**

public class Triangle

{

private double baseLength; // Частный доступ (по умолчанию)

protected double height; // Защищенный доступ

public Triangle(double baseLength, double height)

{

this.baseLength = baseLength;

this.height = height;

}

public double CalculateArea()

{

return 0.5 \* baseLength \* height; // Общедоступный метод

}

}

1. Дайте определение понятию «конструктор». Перечислите особенности конструктора класса. Опишите процесс использования конструктора при создании объекта класса Напишите фрагмент программы описания класса “Прямоугольник”, методами которого представить перегрузку конструкторов. Продемонстрируйте в Main работу с перегруженными конструкторами.

**Конструктор** в объектно-ориентированном программировании (ООП) — это специальный блок инструкций, вызываемый при создании объекта. Его задачи:

1. **Инициализация полей**: Перевести поля объекта в допустимое состояние.
2. **Упростить использование объекта**: Например, передавать параметры при создании объекта[1][1].

Особенности конструктора класса:

* Имеет то же имя, что и класс.
* Не возвращает значения.
* Может быть перегружен (иметь несколько вариантов с разными параметрами).
* Вызывается автоматически при создании объекта.

Пример фрагмента программы для класса “Прямоугольник” с перегрузкой конструкторов:

**C#**

class Rectangle

{

private double width;

private double height;

// Перегруженные конструкторы

public Rectangle(double w, double h)

{

width = w;

height = h;

}

public Rectangle(double side)

{

width = side;

height = side;

}

// Метод для расчета площади

public double CalculateArea()

{

return width \* height;

}

}

class Program

{

static void Main()

{

Rectangle rect1 = new Rectangle(4, 6); // Передаем ширину и высоту

Console.WriteLine($"Area of rect1: {rect1.CalculateArea()}");

Rectangle square = new Rectangle(5); // Передаем длину стороны (квадрат)

Console.WriteLine($"Area of square: {square.CalculateArea()}");

}

}

1. Опишите синтаксис свойств и укажите назначение специальных методов (асессоров) get и set. Опишите процесс использования свойств. Для класса Человек с полями имя и возраст осуществите с помощью set и get проверку вводимых данных (имя – с большой буквы, возраст – от 0 до 100) и измените возраст +10.

**Свойства** в C# — это специальные поля класса, которые позволяют удобно управлять доступом к внутренним данным. Они используются для чтения и записи значений внутренних переменных (полей) класса. [Свойства могут иметь аксессоры get и set, которые обеспечивают доступ к данным1](https://www.bestprog.net/ru/2019/07/26/c-properties-accessors-get-set-examples-of-classes-containing-properties-ru/).

Вот синтаксис объявления свойства:

**C#**

class Person

{

private string name;

private int age;

public string Name

{

get { return name; } // Аксессор get для чтения значения

set { name = char.ToUpper(value[0]) + value.Substring(1); } // Аксессор set для записи значения

}

public int Age

{

get { return age; }

set { age = Math.Clamp(value, 0, 100); } // Ограничение возраста от 0 до 100

}

}

В этом примере:

* Name и Age — это свойства класса Person.
* Аксессор get возвращает значение поля name или age.
* Аксессор set устанавливает значение поля name или age.
* В аксессоре set мы преобразуем имя, чтобы оно начиналось с большой буквы.
* Метод Math.Clamp ограничивает возраст в диапазоне от 0 до 100.

Теперь давайте продемонстрируем использование свойств:

**C#**

class Program

{

static void Main()

{

Person person = new Person();

person.Name = "alice"; // Установка имени

person.Age = 25; // Установка возраста

Console.WriteLine($"Name: {person.Name}, Age: {person.Age}");

// Вывод: "Name: Alice, Age: 25"

}

}

1. Опишите назначение наследования. Дайте определение понятию «абстрактный класс». Укажите особенности абстрактных классов. Приведите пример описания абстрактного класса и производного от него класса, создайте конкретный объект.

**Наследование** в объектно-ориентированном программировании (ООП) — это ключевой принцип, позволяющий создавать иерархии классов. При наследовании один класс (подкласс) получает свойства и методы другого класса (суперкласса). [Это уменьшает дублирование кода, упрощает структуру программы и создает более логичные иерархии объектов1](https://tproger.ru/articles/osnovnye-principy-oop-nasledovanie-v-programmirovanii).

**Абстрактный класс** — это базовый класс, который не предполагает создания экземпляров. Он служит основой для других классов, предоставляя им общую структуру и набор методов. Особенности абстрактных классов:

1. **Не может быть инициализирован**: Нельзя создать объект абстрактного класса напрямую.
2. **Может содержать абстрактные методы**: Эти методы объявляются, но не реализуются в самом абстрактном классе. Они должны быть реализованы в его неабстрактных потомках.
3. [**Объекты создаются на основе производных классов**: Абстрактные классы служат в качестве обобщенных концепций, на основе которых можно создавать более конкретные классы2](https://ru.wikipedia.org/wiki/%D0%90%D0%B1%D1%81%D1%82%D1%80%D0%B0%D0%BA%D1%82%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BB%D0%B0%D1%81%D1%81).

Пример описания абстрактного класса и производного от него класса на языке C#:

**C#**

abstract class Shape

{

public abstract double CalculateArea();

}

class Circle : Shape

{

private double radius;

public Circle(double r)

{

radius = r;

}

public override double CalculateArea()

{

return Math.PI \* radius \* radius;

}

}

class Program

{

static void Main()

{

Circle circle = new Circle(5);

double area = circle.CalculateArea();

Console.WriteLine($"Area of the circle: {area}");

}

}

1. Опишите назначение полиморфизма. Укажите назначение ключевых слов virtual и override. Опишите особенности использования виртуальных методов. Продемонстрируйте работу виртуальных методов для расчета площади на примере иерархии фигур Квадрат -> Прямоугольник с виртуальным методом площадь.

**Полиморфизм** в объектно-ориентированном программировании (ООП) — это принцип, который позволяет разным сущностям выполнять одни и те же действия, независимо от их внутренней реализации. [Он облегчает написание кода, позволяет масштабировать решения, делает код более читаемым и предсказуемым1](https://blog.skillfactory.ru/glossary/polimorfizm/)[2](https://bing.com/search?q=%D0%BF%D0%BE%D0%BB%D0%B8%D0%BC%D0%BE%D1%80%D1%84%D0%B8%D0%B7%D0%BC+%D0%BD%D0%B0%D0%B7%D0%BD%D0%B0%D1%87%D0%B5%D0%BD%D0%B8%D0%B5).

Ключевые слова virtual и override используются для работы с виртуальными методами:

1. virtual:
   * Используется в базовом классе для объявления метода как виртуального (полиморфного).
   * Позволяет переопределять метод в производных классах.
   * Обеспечивает динамическое связывание (выбор правильной реализации метода во время выполнения).
   * Пример: virtual double CalculateArea() { ... }
2. override:
   * Используется в производных классах для указания, что метод переопределяет виртуальный метод из базового класса.
   * Гарантирует, что метод будет вызываться вместо базовой реализации.
   * Предотвращает ошибки при случайном создании нового метода с другим именем.
   * Пример: override double CalculateArea() { ... }

Пример иерархии фигур с виртуальным методом площади:

**C#**

using System;

abstract class Shape

{

public abstract double CalculateArea();

}

class Square : Shape

{

private double side;

public Square(double s)

{

side = s;

}

public override double CalculateArea()

{

return side \* side;

}

}

class Rectangle : Shape

{

private double width;

private double height;

public Rectangle(double w, double h)

{

width = w;

height = h;

}

public override double CalculateArea()

{

return width \* height;

}

}

class Program

{

static void Main()

{

Shape square = new Square(5);

Shape rectangle = new Rectangle(4, 6);

Console.WriteLine($"Area of square: {square.CalculateArea()}");

Console.WriteLine($"Area of rectangle: {rectangle.CalculateArea()}");

}

}

1. Дайте определение понятию «интерфейс». Опишите синтаксис и укажите особенности использования интерфейсов. Реализуйте интерфейс ввода-вывода IPrint для класса Студент, полями которого являются ФИО и его оценки за семестр.
2. **Синтаксис интерфейсов**:
   1. Интерфейсы объявляются с помощью ключевого слова interface.
   2. Они содержат только методы (без реализации) и свойства.
   3. Классы, реализующие интерфейс, обязаны предоставить реализацию всех его методов.
3. **Преимущества интерфейсов**:
   1. Позволяют реализовать множественное наследование методов.
   2. Обеспечивают гибкость и расширяемость кода.
   3. Стандартизируют поведение классов.

Пример реализации интерфейса IPrint для класса Student с полями ФИО и оценками за семестр:

**C#**

using System;

interface IPrint

{

void PrintInfo();

}

class Student : IPrint

{

private string fullName;

private double[] semesterGrades;

public Student(string name, double[] grades)

{

fullName = name;

semesterGrades = grades;

}

public void PrintInfo()

{

Console.WriteLine($"Student: {fullName}");

Console.WriteLine("Semester Grades:");

foreach (var grade in semesterGrades)

{

Console.WriteLine($" - {grade}");

}

}

}

class Program

{

static void Main()

{

double[] grades = { 4.5, 3.8, 4.2, 4.0 };

Student student = new Student("Иванов Иван", grades);

// Используем интерфейс IPrint для вывода информации о студенте

student.PrintInfo();

}

}

1. Перечислите стандартные интерфейсы .Net. Приведите синтаксис и опишите назначение стандартного интерфейса IComparable. Приведите пример реализации данного интерфейса для класса Человек с полями имя и возраст. Сравнение производить по возрасту.

В .NET существует несколько стандартных интерфейсов, которые предоставляют общие функциональные возможности для типов. Один из таких интерфейсов — IComparable. Давайте рассмотрим его синтаксис и назначение.

**Интерфейс**IComparable определяет метод сравнения объектов. Этот метод позволяет упорядочивать или сравнивать объекты по определенному критерию. Классы, реализующие IComparable, могут выполнять сравнение на равенство, больше или меньше.

Вот синтаксис интерфейса IComparable:

**C#**

public interface IComparable<T>

{

int CompareTo(T other);

}

Код, созданный ИИ. Внимательно просмотрите и используйте. [Дополнительные сведения о часто задаваемых вопросах](https://www.bing.com/new#faq).

* T — это тип объектов, которые мы хотим сравнивать.
* Метод CompareTo принимает в качестве параметра другой объект типа T и возвращает целое число:
  + Отрицательное число, если текущий объект меньше переданного.
  + Ноль, если объекты равны.
  + Положительное число, если текущий объект больше переданного.

Давайте реализуем интерфейс IComparable для класса Person, где сравнение будет производиться по возрасту:

**C#**

using System;

class Person : IComparable<Person>

{

public string Name { get; set; }

public int Age { get; set; }

public int CompareTo(Person other)

{

// Сравниваем по возрасту

return Age.CompareTo(other.Age);

}

}

class Program

{

static void Main()

{

Person person1 = new Person { Name = "Alice", Age = 30 };

Person person2 = new Person { Name = "Bob", Age = 25 };

int result = person1.CompareTo(person2);

if (result < 0)

Console.WriteLine($"{person1.Name} is younger than {person2.Name}");

else if (result > 0)

Console.WriteLine($"{person1.Name} is older than {person2.Name}");

else

Console.WriteLine($"{person1.Name} and {person2.Name} are of the same age");

}

}

Код, созданный ИИ. Внимательно просмотрите и используйте. [Дополнительные сведения о часто задаваемых вопросах](https://www.bing.com/new#faq).

В этом примере:

* Мы реализуем интерфейс IComparable<Person> для класса Person.
* Сравнение производится по возрасту.
* Метод CompareTo возвращает отрицательное число, если текущий объект младше переданного, и положительное число, если старше.

1. Укажите назначение операторов для обработки исключений try-catch-finally. Опишите процесс создания пользовательского класса исключений. Укажите назначение оператора throw. Опишите класс Оценка за семестр с полем оценка (int), конструктором с параметром, свойством, проверяющим, что оценка лежит в диапазоне от 0 до 10, и методом вывода информации об объекте. Реализуйте свой класс для обработки исключений для данного класса. Продемонстрируйте работоспособность разработанной программы в Main.
2. **Назначение операторов для обработки исключений:**
   1. **try-catch-finally** используются для обработки исключений:
      1. **try**: В этом блоке размещается код, который может вызвать исключение.
      2. **catch**: Здесь указывается, как обрабатывать исключение, если оно возникло в блоке try.
      3. **finally**: Этот блок выполняется независимо от того, произошло исключение или нет.
3. **Создание пользовательского класса исключений:**
   1. Чтобы создать свой класс исключений, достаточно определить его как производный от класса Exception.
4. Дайте определение понятию «делегат». Укажите синтаксис объявления делегата, создание экземпляров делегата и их использование. Перечислите операции, которые можно использовать с делегатами. Приведите пример передачи делегата в качестве параметра в функцию.
5. Дайте определение понятию «событие» и «обработчик события». Приведите синтаксис события. Укажите как события связано с делегатом. Создайте класс Мешок, полем которого будет количество шаров. Создайте событие добавления и удаления шаров из мешка, предусматривающих различные случаи. Продемонстрируйте работу с событиями в Main.
6. Дайте определение понятию «анонимный метод». Укажите чем лямбда-выражение отличается от анонимного метода. Приведите синтаксис анонимного метода и лямбда-выражения. Напишите сортировку массива по убыванию с использованием лямбда-выражения.
7. Опишите разницу между List, ArrayList и массивом. Перечислите обобщенные и не обобщенные коллекции. Удалите положительные четные элементы из целочисленного листа.
8. Перечислите виды параметризованных коллекций. Опишите коллекцию Dictionary<T,K>. Напишите фрагмент кода с использованием коллекции Dictionary<T,K>.
9. Приведите классификацию файлов в С# с указанием стандартных классов пространства имен System.IO для их обработки. Приведите примеры методов и/или свойств для каждого из классов и укажите их назначение. Приведите примеры записи и чтения текстового файла и байтового файла (на основе потока Stream).
10. Перечислите классы для работы с дисками, каталогами, файлами. Укажите их основные функции. Напишите программу, которая выведет все имена и свойства дисков, а также перечень файлов и папок, находящихся на диске С: текущего компьютера.
11. Что такое сериализация? Как исключить поля из сериализации? Перечислите классы, которые реализуют сериализацию, и укажите их особенности. Приведите примеры.
12. Для чего нужны dll-библиотеки? Опишите процесс создания библиотеки динамической компоновки и подключения ее к проекту на С#. Создайте библиотеку для решения линейных уравнений вида ах=b и продемонстрируйте ее подключение и работу в Main.
13. Опишите модель документа XML. Перечислите основные методы и свойства для обращения к элементам XML-документа. Приведите пример создания XML-документа через класс XmlDocument.
14. Для чего предназначен класс System.Thread? Опишите основные функции для работы с потоком. Приведите пример использования нескольких потоков.
15. Опишите структуру проекта WPF и XAML-кода. Перечислите основные типы контейнеров компоновки и элементов управления. Напишите программу, использующую контейнер Grid, элементы TextBox и Button для поиска самого длинного слова в строке.
16. Опишите процесс привязки базы данных к проекту WPF с помощью технологии ADO.NET или Entity Framework.
17. Опишите структуру паттерна MVVM. Укажите и поясните составляющие паттерна MVVM. Приведите пример реализации этого паттерна на C#.

Преподаватели: Е.Н. Коропа

Е.В. Багласова

В.Ю. Михалевич

А.В. Кривошеина

Д.О. Елинская

Рассмотрено на заседании цикловой комиссии программирования №5

Протокол № 10 от «16» мая 2024

Председатель ЦК Е.Н.Коропа