Новиков 1302

1. Вопрос 34. Операторы, перегрузка операторов.

Многие из них поддерживаются встроенными типами и позволяют выполнять базовые операции со значениями этих типов. В число этих операторов входят следующие группы:

1. Арифметические операторы:

- + (сложение)
- - (вычитание)
- * (умножение)
- / (деление)
- % (остаток от деления)
- ++ (инкремент)
- -- (декремент)

2. Операторы сравнения:

- == (равно)
- != (не равно)
- < (меньше)
- > (больше)
- <= (меньше или равно)
- >= (больше или равно)

3. Логические операторы:

- && (логическое И)
- || (логическое ИЛИ)
- ! (логическое НЕ)

4. Побитовые операторы:

- & (побитовое И)
- | (побитовое ИЛИ)

- ^ (побитовое исключающее ИЛИ)
- ~ (побитовое НЕ)
- << (побитовый сдвиг влево)
- >> (побитовый сдвиг вправо)

5. Операторы равенства

- == (равно)
- != (не равно)

6. Операторы присваивания:

- = (присваивание)
- += (сложение с присваиванием)
- -= (вычитание с присваиванием)
- *= (умножение с присваиванием)
- /= (деление с присваиванием)
- %= (остаток от деления с присваиванием)
- &= (побитовое И с присваиванием)
- |= (побитовое ИЛИ с присваиванием)
- ^= (побитовое исключающее ИЛИ с присваиванием)
- <<= (побитовый сдвиг влево с присваиванием)
- >>= (побитовый сдвиг вправо с присваиванием)

7. Условный оператор:

- ? : (тернарный условный оператор)

8. Другие операторы:

- x.y (доступ к члену объекта)
- x[y] (оператор индексатора)
- ?? и ??= (операторы объединения со значением NULL)

(приведены не все операторы)

Как правило, можно выполнить перегрузку этих операторов, то есть указать поведение оператора для операндов определяемого пользователем типа.

Перегрузка операторов осуществляется с помощью ключевого слова operator.

Синтаксис:

```
public static int operator Op (ClassName a, ClassName b)
```

Op – оператор

ClassName – имя класса, в котором реализован данный метод

Метод, перегружающий оператор должен иметь идентификаторы static и public.

2. Вопрос 65. Принципы объектно-ориентированного проектирования SOLID. Принцип единственности обязанности

SOLID это акроним, который описывает пять основных принципов объектно-ориентированного проектирования. Каждая буква SOLID соответствует одному из этих принципов:

S - Принцип единственной ответственности (Single Responsibility Principle). Этот принцип гласит, что класс должен быть ответственным только за одну вещь. Все ресурсы, необходимые для его осуществления, должны быть инкапсулированы в этот класс и подчинены только этой задаче

O - Принцип открытости/закрытости (Open/Closed Principle). Этот принцип гласит, что классы должны быть открыты для расширения и закрыты для изменения. Это достигается через использование абстракций и интерфейсов.

L - Принцип подстановки Барбары Лисков (Liskov Substitution Principle). Этот принцип гласит, что функции, которые используют базовый тип, должны иметь возможность использовать подтипы базового типа не зная об этом.

I - Принцип разделения интерфейса (Interface Segregation Principle). Этот принцип гласит, что интерфейсы классов должны быть маленькими и специфичными для конкретных задач. Большие интерфейсы следует разбивать на более мелкие и специализированные.

D - Принцип инверсии зависимостей (Dependency Inversion Principle). Этот принцип гласит, что модули верхнего уровня не должны зависеть от модулей нижнего уровня. Вместо этого, оба уровня должны зависеть от абстракций. Это позволяет лучше управлять зависимостями и делает код более гибким и расширяемым.

Принцип единственности обязанности гласит, что каждый класс должен иметь только одну ответственность и делать только одну вещь. Если класс имеет несколько ответственностей, то любые изменения в одной из них могут повлиять на другие ответственности, что может привести к ошибкам,

сложностям в тестировании и поддержке кода. Кроме того, классы с несколькими ответственностями часто являются признаком плохого дизайна.

Принцип единственной ответственности помогает разделить функциональность программы на более мелкие и легко управляемые блоки, что упрощает поддержку и облегчает расширение приложения.

В C# принцип SRP может быть реализован, например, через разделение методов и свойств в классах на отдельные модули или классы с конкретными задачами. Также возможна выделение интерфейсов, которые могут использоваться несколькими классами и обладать единственной обязанностью.