C# и Java

Ниже приведен перечень свойств, характерных как для C#, так и для Java. Оба эти языка можно рассматривать как попытку усовершенствовать C++, и нужно признать, что в обоих случаях это удалось. Как можно увидеть из приведенного ниже списка, во многом C# и Java схожи, но было бы неверно отождествлять эти языки.

- Исходный текст программы компилируется в промежуточный код, не зависящий от языка и платформы; этот код в дальнейшем выполняется в специальной управляемой среде.
- Автоматический сбор мусора (GarbageCollection) и запрет на использование указателей. В C# допускается ограниченное использование указателей в блоках кода, помечаемых как "ненадежные" (unsafe).
- Отсутствие заголовочных файлов. Весь код помещается в пакеты (packages) и сборки (assemblies). Никаких проблем с порядком объявления классов в случае наличия перекрестных ссылок.
- Объекты создаются с помощью ключевого слова **new**, выделение памяти производится из "кучи" (heap), находящейся в распоряжении среды выполнения.
- Многопоточность поддерживается путем блокирования объектов
- Интерфейсы, с множественным наследование интерфейсов, однократное наследование реализаций.
- Внутренние классы
- Отсутствие концепции наследования классов с заданным уровнем доступа.
- Отсутствие глобальных функций и констант, все элементы должны принадлежать классам.
- Массивы и строки со встроенной длиной и проверкой границ.
- Не применяются операторы **->**, **::**. Во всех случаях используется оператор "**.**".
- **null** и **boolean**/**bool** являются ключевыми словами.
- Любая величина должна быть проинициализирована до того, как будет использована.
- Нельзя использовать целые числа (integers) для управления операторами **if.**
- Блоки **try** могут иметь заключительное предложение **finally.**
-
-
-
- СВОЙСТВА
- C# предлагает более прозрачный способ реализации свойств, что особенно очевидно для свойств, допускающих чтение и запись. Связь методов Get и Set в C# становится врожденной, в то время как в C++ и Java она лишь поддерживается. У такого подхода есть много преимуществ. Он заставляет программистов мыслить в терминах свойств, независимо от того доступно ли свойство как для чтения, так и для записи, или оно предполагает только чтение. Если вы захотите изменить название свойства, то вам достаточно будет сделать это в одном месте (а часто методы get и set оказываются разделенными сотнями строк кода). Комментарии также достаточно ввести в одном месте, так что они никогда не окажутся рассогласованными.
- Можно возразить, что предлагаемый C# синтаксис не дает реальных преимуществ, так как в случае его использования нельзя с уверенностью сказать, с чем мы имеем дело, с полем или свойством. Но практически никогда реальные классы, спроектированные на Java (и естественно на C#), не имеют общедоступных (public) полей. Поля обычно имеют ограниченный уровень доступа (private/protected/default) и раскрываются только через функции get/set, где C# как

раз и предлагает более удобный синтаксис. Кроме того очевидно, что если класс правильно спроектирован, то пользователя должна интересовать только спецификация класса, а отнюдь не его реализация.

- Еще один аргумент противников использования свойств - снижение эффективности кода. Однако хороший компилятор может свести реализацию простого метода получения значения поля (get) к in-line функции, что сделает ее выполнение столь же быстрым, как и непосредственное считывание значения поля.

Индексаторы (Indexers)

В C# предусмотрены средства для создания пользовательских классов-контейнеров, к внутренним элементам которых можно обращаться с помощью того же оператора индекса, что и к элементам обычного массива встроенных типов. Метод, который обеспечивает такую возможность, получил название **индексатор** (indexer). Индексатор представляет собой слегка измененное свойство C#, в простейшем случае индексатор создается через синтаксическую конструкцию this[], например: