Объектно-ориентированный язык программирования C#.

История создания и специфика

1. История создания языка C#

C# (произносится «**Си-шарп**») был создан компанией **Microsoft** как новый язык программирования **в конце 1999 г. – начале 2000 г**. в рамках инициативы **.NET** (см. **Материалы к Практ. зан. Введение_dotNet**).

Sharp (*англ.*) – отточенный.	# - нотный знак, обозначающий повышение звука

Его парадигма программирования очень похожа на **Java**. **C#** представляет собой своего рода систему на основе специального байт-кода, с обещаниями многоплатформенности. В **2000** г. **ECMA** по запросу **Microsoft** одобрила **C#** как **стандартный** язык. Это нетрадиционное решение для **Microsoft**, потому что теперь **C#** — это язык, определенный в стандарте, который Microsoft не сможет менять по своему усмотрению. Многие увидели в этом преимущество **C#**, так как он получил явную независимость от **Microsoft**.

Теория, лежащая в основе **C#**, аналогична **Java** в том, что язык может быть использован для создания многоплатформенных программ. На практике, однако, **Microsoft** видит роль **C#** в создании программ для **Windows**-платформ. Хотя существуют открытые проекты для переноса этой системы на другие платформы, их успех не гарантирован, а **Microsoft** не делает никаких заявлений относительно поддержки Не-**Microsoft** -платформ.

В **Windows** обычная программа состоит из двоичного исполняемого файла. То есть исходный код программы компилируется в машинный формат. **.NET** все еще предоставляет такой тип разработки программного обеспечения, хотя он специально называется *не*управляемым кодом.

Новая модель, используемая в **.NET**, подражает **Java**-платформе: приложение под названием **C**ommon **L**anguage **R**untime (общеязыковая среда выполнения, **CLR**) используется для интерпретации и выполнения байт-кодов инструкций, собранных из различных исходных модулей. Это идентично концепции, в соответствии с которой работает виртуальная машина **Java** (**JVM**). **CLR** имеет два принципиальных отличия от **JVM** (**J**ava **V**irtual **M**achine).

- Основной язык **CLR** — **C#** — теперь стандартизован. Тогда как фирма **Sun** контролирует **Java**, **C#** принят в качестве стандартного языка Европейской ассоциацией производителей компьютеров (**E**uropean **C**omputer **M**anufacturer's **A**ssociation, **ECMA**). Это означает, что Microsoft не может по своей прихоти вносить изменения в язык, а также должна поддерживаться обратная совместимость. Более того, **CLR** поддерживает и другие языки помимо **C#**, такие как **C++**, **COBOL** и Java. Хотя обычно они и не полностью поддерживаются, основные их особенности реализованы.
- **CLR** поддерживает компиляцию в машинный код для увеличения производительности.

Также, подобно **JVM**, **CLR** осуществляет контроль над выполнением кода и имеет возможность останавливать злонамеренные программы. Как и в **Java**, **CLR-**программы имеют тенденцию выполняться медленнее, чем их коллеги в машинном коде. Программы, написанные под **CLR**, называются управляемым кодом (**managed code**).

CLR также предоставляет другие возможности, например, в нем реализован набор стандартных классов и **C**ommon **T**ype **S**ystem (**общая система типов**) для реализации типов данных одинаковыми способами.

2. Специфика языка C#

2.1. Повторно используемый код

C# содержит большое количество повторно используемого кода, как и **Java**. Классы **.NET Framework** предоставляют поддержку соединений с базами данных, обработку изображений, работу с Интернетом и т. д.

2.2. Производительность: интерпретируемый или компилируемый код?

Как и программы на **Java**, программы на **C#** компилируются в специальный байт-код, который может понять и выполнить отдельная программа. **C**ommon **L**anguage **R**untime (**CLR**) — это программа, которая работает с байт-кодом **C#**. Однако **CLR** пошла дальше подхода виртуальной машины, и на самом деле компилирует приложения «на лету» в машинный код, тем самым увеличивая производительность. В случае часто вызываемых приложений для веб-серверов, таких как **ASP.NET**, результаты компиляции кэшируются и повторно используются, опять же для увеличения производительности.

Таким образом, **C#** действует по-другому: программы, написанные на этом языке, сначала «компилируются» в **CLR**-совместимый формат (похоже на то, что делает **Java**), а затем компилируются в машинный код, когда **CLR** первый раз их выполняет.

C# предоставляет встроенную поддержку **COM**-компонентов и **Windows API**, а также ограниченное использование указателей. Это означает, что хотя программы на этом языке могут быть не такими быстрыми, как приложения, выполняемые напрямую (из-за интерпретации **CLR**), их производительность обычно не такая низкая, как в случае **Java**-приложений.

2.3. Безопасность

C# реализует модель безопасности аналогично **Java**, в формате песочницы, но основной контроль устанавливается сборкой, которая позволяет определить, какие операции могут выполняться некоторой программой или классом. Однако в отличие от **Java** **C#** поддерживает указатели для прямых манипуляций с памятью, хотя их использование также может контролироваться настройками безопасности.

2.4. Переносимость

Теоретически **C#** переносим, но пока никакие операционные системы производства не **Microsoft** не могут использовать его. Это, однако, не означает, что он полностью непереносим. У **Microsoft** также есть операционная система для карманных устройств, которая называется **Windows Mobile** (ранее она называлась **Windows CE**, **Windows PocketPC**, **PocketPC 2000** и т. д.), которая распространяется с **.NET Framework** и **CLR** для выполнения программ на **C#**. Как и **Java**, **C#** можно использовать на веб-серверах, настольных компьютерах и карманных устройствах, если на них есть соответствующая версия **Windows**.

2.5. Сбор мусора

Как и **Java**, **C#** реализует автоматический сбор мусора. Хотя в **C#** можно объявлять деструкторы, важно отметить, что они вызываются, когда процесс сбора мусора определяет, что объект больше недоступен коду и что необходима память. Деструкторы в **C#** имеют такой же формат, как и в **C++:**

~ClassName();

Процесс сбора мусора вызовет этот метод автоматически, когда сочтет это целесообразным.

2.6. Пользовательский интерфейс

C# предоставляет широкую поддержку пользовательского интерфейса, но **C#** ограничен операционной системой Windows. **C#** также предоставляет возможность легко использовать существующие элементы управления **ActiveX.** Как **Java** и его **JavaBeans**, язык **C#** был разработан, чтобы упростить создание компонентов или дополнительных модулей, которые могут быть привязаны к среде разработки программ. Это означает, что вы можете легко создавать новые элементы управления и легко использовать их как встроенную часть компонентов среды разработки.

2.7. Множественное наследование

C# *не* поддерживает множественное наследование, в отличие от **C++**. **C#** поддерживает интерфейсы похожим на **Java** образом. Как и в **Java**, интерфейсы определяют код, который должен быть написан, а не сам повторно используемый код.

2.8. Параметризованные типы и шаблоны

C# не поддерживает шаблоны или параметризованные типы, в отличие от **C++** и **Java 1.5.**

2.9. Сборки

Сборка (**assembly**) представляет собой набор информации для одного или более файлов кода, как показано в табл. 1.

Сборки представляют собой текстовые файлы, которые похожи на исходный код. Они могут быть внедрены в **CLR**-программу или определены вне **CLR** для нескольких файлов. Множество программ могут включать сборку в одном исполняемом файле. Ниже приведен короткий пример сборки для проекта на **C#**.

Таблица 1. Некоторые варианты сборок

Сборка	Описание
Версия	Группирует модули, которые должны иметь одинаковую информацию о версии
Развертывание	Группирует модули кода и ресурсы, которые поддерживают вашу модель развертывания
Повторное использование	Группирует модули, если они логически могут использоваться совместно для каких-то целей. Например, типы и классы, нечасто используемые для обслуживания программы, можно поместить в одну сборку. Кроме того, типы, которые вы собираетесь использовать во многих приложениях, можно сгруппировать в одну сборку, и сборке должно быть присвоено «сильное» имя
Безопасность	Группирует модули, которые должны иметь одинаковые разрешения безопасности
Область видимости	Группирует модули, содержащие типы, видимость которых должна быть ограничена одной и той же сборкой