Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук

Кафедра Програмної інженерії

Контрольна робота

з дисципліни «Технології компонентного ПЗ»

Студент

гр. ПІз-11-1 Нікулін А.В.

Викладач

К.т.н., доцент Лещинський В. О.

Харків, 2014 р.

15. Правила іменування подій.

1. Используйте суффикс *EventHandler* для делегатов, являющихся типами событий. Другие классы не должны использовать этот суффикс.
2. Создавая событие в компонентах и control-ах, старайтесь описывать их по следующей схеме. Определите два параметра с именами *sender* и *e.* Параметр *sender* описывает объект, инициировавший событие, и всегда должен быть типа object, даже если возможно использование более конкретного типа. Второй параметр, *e,* должен содержать состояние и дополнительную информацию, соответствующую событию. Этот параметр должен быть конкретного типа, относящегося к событию.
3. Делайте тип, описывающий связанную с событием информацию, производным от *EventArgs,* и используйте суффикс *EventArgs*. Другие классы, не описывающие информацию о событии, не должны использовать этот суффикс.
4. Для имен событий старайтесь использовать глаголы, которые описывают производимое над объектом действие (например, Click, GotFocus или FontChanged).
5. Не используйте суффиксы наподобие *On, Before, After* для идентификатора события. Используйте соответствующую форму глагола, например *Closing* перед закрытием и Closed после закрытия.
6. При описании события также предоставляйте виртуальный protected-метод, который можно переопределить в классе-наследнике. Называйте такой метод OnXxx, где Xxx – имя события. В качестве параметров таких методов не следует передавать *sender*, так как – это всегда текущий объект (this).
7. Пытайтесь подобрать стандартный делегат и название для своих событий. Например, если ваш элемент управления должен реагировать на нажатие кнопки мыши, следует использовать стандартное событие Click. Для элементов управления, обычно, такие события уже объявлены в базовом классе Control.

Пример:

|  |
| --- |
| public delegate void MouseEventHandler(object sender, MouseEventArgs e);  public class MouseEventArgs : EventArgs  {  private int x;  private int y;  public MouseEventArgs(int x, int y)  {  this.x = x;  this.y = y;  }  public int X { get { return x; } }  public int Y { get { return y; } }  }  public class Control  {  public event MouseEventHandler MouseMove;  protected virtual void OnMouseMove(MouseEventArgs e)  {  if (MouseMove != null)  MouseMove(this, e);  }  } |

16. Рекомендації щодо розробки типів.

Используйте пространства имен для организации типов в иерархию связанных предметных областей.

Избегайте слишком глубокой иерархии пространств имен. Такие иерархии сложны для обзора, так как пользователю приходится часто возвращаться назад.

Избегайте слишком большого числа пространств имен.

Типы, используемые в одних и тех же сценариях, должны по возможности располагаться в тех же самых пространствах имен. Пользователи не должны быть вынуждены импортировать слишком большое число пространств имен для разрабатываемых сценариев.

Не допускайте, чтобы типы для сложных сценариев располагались в том же пространстве имен, что и типы, предназначенные для обычных сценариев.

В общем случае, следует размещать расширенные типы в пространстве имен внутри основного пространства имен и использовать Advanced в качестве последнего идентификатора имени. Например, общецелевые типы, относящиеся к XML-сериализации, находятся в пространстве имен [System.Xml.Serialization](http://msdn.microsoft.com/ru-ru/library/vstudio/system.xml.serialization%28v=vs.100%29.aspx), а расширенные типы – в пространстве имен [System.Xml.Serialization.Advanced](http://msdn.microsoft.com/ru-ru/library/vstudio/system.xml.serialization.advanced%28v=vs.100%29.aspx).

Не определяйте типы без указания их пространства имен.

Типы, которым не назначены пространства имен, располагаются в глобальном пространстве имен. Так как они не находятся в специальном пространстве имен, типы в глобальном пространстве имен сложно найти при помощи инструментов разработчика. Кроме того, коллизии по именам в глобальном пространстве имен не могут быть разрешены

**Выбор между классами и структурами**

Классы являются ссылочными типами, а структуры — типами значения. Ссылочные типы размещаются в куче, а управление памятью осуществляется сборщиком мусора. Типы значения размещаются на стеке или непосредственно в коде и становятся недоступны, когда находятся вне области видимости. В общем случае для размещения и освобождения типов значения требуется меньше затрат. Однако, если эти типы используются в сценарии, в котором требуется значительное количество преобразований упаковки-распаковки, они работают хуже по сравнению со ссылочными типами

Рекомендуется определять структуру вместо класса, если экземпляры типа малы и имеют короткое время существования или внедрены в другие объекты.

Определяйте структуру только в том случает, если тип обладает всеми приведенными ниже характеристиками.

* Тип логически представляет единичное значение подобно примитивным типам (integer, double и т.п.).
* Размер экземпляра не превышает 16 байт.
* Тип является неизменяемым.
* Тип не нуждается в частом выполнении упаковка-преобразований.

Если одно или несколько этих условий не выполняется, создавайте вместо структуры ссылочный тип. Не следование этой рекомендации может негативно сказаться на производительности.

# Выбор между классами и интерфейсами

Интерфейс определяет сигнатуры для набора членов, предоставляемых разработчиками. Интерфейсы не могут предоставить членам подробные сведения о реализации. Например, в интерфейсе [ICollection](http://msdn.microsoft.com/ru-ru/library/vstudio/system.collections.icollection%28v=vs.100%29.aspx) определяются члены, относящиеся к работе с коллекциями. Каждый класс, реализующий интерфейс, должен предоставить этим членам подробные сведения о реализации. Классы способны реализовывать несколько интерфейсов.

Классы определяют как сигнатуры членов, так и подробные сведения о реализации для каждого члена. Классы Abstract (MustInherit в Visual Basic) могут вести себя, как интерфейсы или обычные классы, в которых они могут определять члены; также они могут предоставлять подробные сведения о реализации, но это не является обязательным. Если абстрактный класс не предоставляет подробные сведения о реализации, для этого требуются конкретные классы, являющиеся производными от абстрактного класса.

И абстрактные классы, и интерфейсы поддерживают отделение контракта от реализации, однако интерфейсы не могут описывать новые члены в более поздних версиях, тогда как абстрактные классы могут добавлять члены по мере необходимости для поддержания дополнительной функциональности.

Предпочтительнее определение классов, а не интерфейсов.

В более поздних версиях библиотеки можно безопасно добавлять в классы новые члены; добавить члены в интерфейсы невозможно, не нарушая существующего кода.

Используйте абстрактные ("MustInherit" в Visual Basic) классы вместо интерфейсов для отделения контракта от реализаций.

Определяйте интерфейс для предоставления полиморфной иерархии типов значений.

Типы значений должны наследовать от класса [ValueType](http://msdn.microsoft.com/ru-ru/library/vstudio/system.valuetype%28v=vs.100%29.aspx) и могут наследовать только от класса [ValueType](http://msdn.microsoft.com/ru-ru/library/vstudio/system.valuetype%28v=vs.100%29.aspx), поэтому они не могут использовать классы для отделения контракта от реализации. В этом случае, если для типов значений требуется полиморфное поведение, следует использовать интерфейс.

Рекомендуется определять интерфейсы для достижения результата, подобного множественному наследованию.

Если тип должен реализовать несколько контрактов или контракт применим к большому числу типов, используйте интерфейс. Например, интерфейс [IDisposable](http://msdn.microsoft.com/ru-ru/library/vstudio/system.idisposable%28v=vs.100%29.aspx) реализуется типами, используемыми в различных сценариях. Требование, чтобы классы, производные от базового, можно было освободить, лишает иерархию классов гибкости. Такие классы, как [MemoryStream](http://msdn.microsoft.com/ru-ru/library/vstudio/system.io.memorystream%28v=vs.100%29.aspx), которые должны наследовать свои основанные на потоках контракты от родительских классов, не смогут этого сделать и также будут освобождаться.

# Разработка абстрактных классов

Абстрактные классы никогда не должны порождать экземпляры, поэтому важно правильно определять их конструкторы. Крайне важно обеспечить правильную работу абстрактного класса и возможность расширения его функциональности. Выполнение следующих рекомендаций позволит быть уверенными в том, что абстрактные классы разработаны правильно и при реализации будут работать, как ожидается.

Не определяйте конструкторы с атрибутами "public" или "protected internal" ("Protected Friend" в Visual Basic) в абстрактных типах.

Конструкторы с видимостью public или protected internal используются для типов, для которых могут создаваться экземпляры. Из абстрактных типов никогда не могут создаваться экземпляры.

Определите защищенный или внутренний конструктор в абстрактных классах.

При определении защищенного конструктора в абстрактном классе базовый класс может выполнить инициализацию при создании экземпляров производного класса. Внутренний конструктор не позволяет использовать абстрактный класс в качестве базового класса для типов, находящихся за пределами сборки, содержащей данный абстрактный класс.

Предоставьте хотя бы один конкретный тип, наследующий от каждого поставляемого абстрактного класса.

Такой метод помогает разработчикам библиотек обнаруживать проблемы или недоработки в коде абстрактного класса. Этот также означает, что при написании сценариев высокого уровня разработчики, которые не понимают принципов абстрактных классов и наследования, могут использовать конкретный класс, не изучая понятий абстрактного класса. Например, платформа .NET Framework содержит абстрактные классы [WebRequest](http://msdn.microsoft.com/ru-ru/library/vstudio/system.net.webrequest%28v=vs.100%29.aspx) и [WebResponse](http://msdn.microsoft.com/ru-ru/library/vstudio/system.net.webresponse%28v=vs.100%29.aspx) для отправки и получения запросов от универсального идентификатора ресурса (URI). В качестве одной из конкретных реализаций для этих абстрактных классов в среде .NET Framework содержатся классы [HttpWebRequest](http://msdn.microsoft.com/ru-ru/library/vstudio/system.net.httpwebrequest%28v=vs.100%29.aspx) и [HttpWebResponse](http://msdn.microsoft.com/ru-ru/library/vstudio/system.net.httpwebresponse%28v=vs.100%29.aspx), которые реализуют эти абстрактные классы для протокола HTTP.

# Разработка статичных классов

Статические классы — это классы, которые не содержат экземпляров членов, кроме производных от [Object](http://msdn.microsoft.com/ru-ru/library/vstudio/system.object%28v=vs.100%29.aspx), и не имеют вызываемого конструктора. Следующие рекомендации помогут обеспечить правильность разработки статических классов.

Используйте статические классы только в случае необходимости.

Статические классы следует использовать только для поддержки объектно-ориентированной модели платформы.

Не используйте статические классы как контейнер для всевозможных объектов.

Класс [Environment](http://msdn.microsoft.com/ru-ru/library/vstudio/system.environment%28v=vs.100%29.aspx) является хорошим примером правильного использования статического класса. Этот класс предоставляет доступ к сведениям о текущем пользовательском окружении.

Не объявляйте и не переопределяйте члены экземпляра в статических классах.

Если в классе предусмотрены члены экземпляра, класс не должен быть объявлен статическим.

Объявляйте статические классы абстрактными и запечатанными и добавляйте конструктор частных экземпляров, если в используемом языке программирования не предусмотрена встроенная поддержка статических классов.

# Разработка интерфейса

Интерфейс определяет сигнатуры для набора членов, предоставляемых разработчиками. Интерфейсы не могут предоставить членам подробные сведения о реализации. Например, в интерфейсе [ICollection](http://msdn.microsoft.com/ru-ru/library/vstudio/system.collections.icollection%28v=vs.100%29.aspx) определяются члены, относящиеся к работе с коллекциями. Каждый конкретный класс, реализующий интерфейс, должен предоставить подробные сведения о реализации этих членов. Классы могут наследовать только от одного класса, однако они могут реализовать несколько интерфейсов. Представленные ниже рекомендации помогут обеспечить правильность разработки интерфейсов.

Определяйте интерфейс, если необходима поддержка некоторой общей функциональности набором типов, содержащим типы значения.

Типы значения должны наследовать от типа [ValueType](http://msdn.microsoft.com/ru-ru/library/vstudio/system.valuetype%28v=vs.100%29.aspx). Поэтому абстрактные классы нельзя использовать для описания контракта для типов значения; вместо этого следует использовать интерфейсы.

Рекомендуется определять интерфейс, если требуется поддержка его функциональности в типах, являющихся производными от какого-либо другого типа.

Избегайте использования интерфейсов-маркеров (интерфейсов без членов).

Одним из способов маркировки типов являются настраиваемые атрибуты. Настраиваемые атрибуты предпочтительнее, когда можно отложить проверку атрибута до выполнения кода. Если для сценария требуется проверка во время компиляции, эта рекомендация не может выполняться.

Предоставьте хотя бы один тип, являющийся реализацией интерфейса.

Это позволит удостовериться в правильности разработки интерфейса и может быть реализовано без особых сложностей. Класс [Int32](http://msdn.microsoft.com/ru-ru/library/vstudio/system.int32%28v=vs.100%29.aspx) предоставляет реализацию интерфейса [IComparable](http://msdn.microsoft.com/ru-ru/library/vstudio/system.icomparable%28v=vs.100%29.aspx).

Предоставьте хотя бы один член, использующий каждый из определенных интерфейсов (например, метод, принимающий интерфейс в качестве параметра, или свойство с типом интерфейса).

Это еще один механизм, который позволит удостовериться в правильности разработки интерфейса и может быть использован без особых сложностей.

Не добавляйте члены к ранее поставленному интерфейсу.

Добавление новых членов приведет к неработоспособности кода, реализующего предыдущую версию интерфейса. Это является одной из основных причин того, почему в общем случае классы являются предпочтительнее интерфейсов. Если для поставляющего определения интерфейса требуются дополнительные члены, можно реализовать новый интерфейс и соответствующие члены для его использования.

# Разработка структуры

Структуры имеют тип значения. Они размещаются на стеке или в области данных и становятся недоступны, когда выходят из области видимости. В общем случае тип значения эффективен для размещения и освобождения, однако если он используется в сценарии, который требует значительного количества преобразований упаковки-распаковки, этот тип работает хуже по сравнению со ссылочным типом

Не предоставляйте конструктор по умолчанию для структуры.

Если в структуре определен конструктор по умолчанию, то при создании массивов структуры среда CLR автоматически вызовет конструктор по умолчанию для каждого элемента массива.

Некоторые компиляторы, такие как компилятор C#, не позволяют структурам иметь конструктор по умолчанию.

Не реализуйте System.IEquatable`1 на типах значения.

[IEquatable<T>](http://msdn.microsoft.com/ru-ru/library/vstudio/ms131187%28v=vs.100%29.aspx) предпочтительнее, чем [Equals](http://msdn.microsoft.com/ru-ru/library/vstudio/system.object.equals%28v=vs.100%29.aspx) для сравнения двух типов значения. При использовании интерфейса предотвращается негативное влияние преобразования упаковки и управляемого отражения.

Обязательно убедитесь, что состояние, в котором все данные экземпляра имеют значения false, null иле нулевые (в зависимости от конкретного случая) является приемлемым.

Если вы будете следовать этому указанию, вновь созданные вами экземпляры типов значения не останутся в неиспользуемом состоянии. Например, следующая структура спроектирована неправильно. Параметризованный конструктор предназначен для обеспечения корректного состояния, но конструктор не выполняется, когда создается массив структур. Это означает, что экземпляр поля label инициализируется в null (Nothing в Visual Basic), что недопустимо для реализации этой структуры для метода **ToString**.

# Разработка перечислений

Перечисления предоставляют набор постоянных значений, которые применяются для строго типизированных членов и повышают удобочитаемость кода. Перечисления могут быть либо простыми, либо флаговыми. Простые перечисления содержат значения, которые не комбинируются и не используются в побитовых операциях сравнения. Флаговые перечисления предназначены для комбинирования при помощи побитовых операций OR. Комбинации значений флаговых перечислений проверяются с использованием побитовых операций AND.

В приведенных ниже рекомендациях приводятся оптимальные методы разработки перечислений.

Используйте перечисления для строго типизированных параметров, свойств и возвращаемых значений, которые представляют собой наборы значений.

Предпочтительнее использовать перечисления, а не статические константы.

Не используйте перечисление для открытых наборов, таких как версия операционной системы.

Добавление значений в уже поставленное перечисление может привести к неработоспособности существующего кода. Иногда это может быть приемлемо, но при наличии подобных проблем разработка перечислений нецелесообразна.

Не определяйте зарезервированные значения перечислений, которые предполагается использовать в будущем.

В некоторых ситуациях может оказаться, что необходимо добавить значения к поставленному перечислению, несмотря на риск возникновения неработоспособности существующего кода. Можно также определить новое перечисление и члены, которые работают с его значениями.

Избегайте перечислений с открытым доступом, содержащих только одно значение.

Не включайте в перечисления значения-метки.

Значения-метки используются для определения границ значений перечисления. Обычно значение-метка используется при проверке диапазонов и не является допустимым значением данных. В следующем примере кода определяется перечисление, в котором используется значение-метка.

Битовое перечисление используется для маскирования битовых полей и выполнения битовых сравнений. При корректном дизайне они используются для задания множественных перечислимых значений за один раз. Например, можно комбинировать любые из перечислимых значений [GenericUriParserOptions](http://msdn.microsoft.com/ru-ru/library/vstudio/system.genericuriparseroptions%28v=vs.100%29.aspx) для конфигурации базового синтаксического анализатора Uniform Resource Identifier (URI).

Используйте System.FlagsAttribute для битовых перечислений. Не применяйте этот атрибут для простых перечислений.

Используйте степень числа 2 для значений битовых перечислений, так чтобы их можно было свободно комбинировать с использованием битовой операции OR.

*Если Вы не используете степень числа 2 или комбинации степеней числа 2, битовые операции не выполняются, как предусмотрено.*

Для наиболее используемых комбинаций битов следует рассмотреть возможность создания специальных значений перечислений.

Комбинирование значений битовых перечислений — это средний уровень мастерства, который не следует требовать от разработчиков обычных сценариев. Например, перечисление [FileShare](http://msdn.microsoft.com/ru-ru/library/vstudio/system.io.fileshare%28v=vs.100%29.aspx) содержит значение [ReadWrite](http://msdn.microsoft.com/ru-ru/library/vstudio/system.io.fileshare%28v=vs.100%29.aspx) для указания того, что разделяемый файл может быть открыт для чтения или записи. Это показывает разработчикам, что они могут открыть разделяемый файл для чтения или записи, и исключает необходимость комбинирования значений перечислений из отдельных значений.

Избегайте создания битовых перечислений, когда определенные комбинации значений недопустимы.

Эта проблема обычно указывает на то, что смысл перечисления недостаточно определен. Следует рассмотреть возможность разделения перечисления на два или более перечисления, в каждом из которых будет более определенный набор значений. Например, рассмотрим следующее плохо определенное перечисление.

[Flags]

public enum PurchaseTypes

{

SalePrice,

RegularPrice,

Book,

CompactDisk

}

Рекомендуется добавлять значения к перечислениям, несмотря на небольшой риск возникновения несовместимости.

Эта рекомендация становится важной, если предполагается поставлять несколько версий библиотеки. Чтобы снизить до минимума случаи сбоя кода в результате добавления новых значений к существующему перечислению, можно реализовать новые члены, которые возвращают полный набор значений, и пометить существующие члены (которые возвращают первичный набор значений), используя атрибут [ObsoleteAttribute](http://msdn.microsoft.com/ru-ru/library/vstudio/system.obsoleteattribute%28v=vs.100%29.aspx). В случае потенциально опасных изменений можно определить новое перечисление и связанные с ним члены и пометить существующие члены и перечисление как устаревшие.

Перелік посилань

1. Бібліотека MSDN [Електронний ресурс] - Режим доступу : www/ URL: http://msdn.microsoft.com/ru-ru/library/vstudio/ms229042(v=vs.100).aspx