1. Основные положения ГОСТ Р ИСО/МЭК 12207

Область применения

* устанавливает общую структуру процессов жизненного цикла программных средств, на которую можно ориентироваться в программной индустрии
* используется при приобретении систем, программных продуктов и услуг, при их поставке, разработке, применении по назначению, сопровождении и прекращении применения программных продуктов и программных компонентов системы как в самой организации, так и вне ее
* устанавливает также процесс, который может использоваться при определении, управлении и совершенствовании процессов жизненного цикла программных средств

Назначение

Предназначен для представления определенной совокупности процессов, облегчающих связи между приобретающими сторонами, поставщиками и другими правообладателями в течение жизненного цикла программных продуктов

Ограничения

* не детализируются процессы жизненного цикла  
  не устанавливает требований к документации (наименование форматы, содержание, носители)
* не устанавливает конкретной модели жизненного цикла  
  не должен противоречить политикам, процедурам и нормам применяющей его организации, национальным законам и регулирующим документам

1. Основные процессы и взаимосвязь между документами в информационной системе

* Процесс соглашения (приобретение, поставка)
* Процесс организации обеспечения проекта (менеджмент жц, структуры, качества и тд)
* Процесс проекта (планирование, управление, оценка проекта, менеджмент рисков, измерений и тд)
* Технические процессы (определение требований, анализ сис.требований, проектирование, реализация, тестирование, инсталляция, поддержка, сопровождение, прекращение применения)

1. Понятие ЖК ПО. Этапы ЖЦ ПО

Жизненный цикл Программного обеспечения – период времени, который начинается с момента принятия решения о необходимости создания программного обеспечения и заканчивается в момент его полного изъятия из эксплуатации

Основные понятия:

* Артефакты – создаваемые человеком информационные сущности. Документы, участвующие в входных данных
* Роль – абстрактная группа заинтересованных лиц, участвующих в деятельности по созданию и эксплуатации системы
* Программный продукт – набор компьютерных программ, процедур и связанных с ним документации и данных

**Этапы ЖЦ ПО**

Жизненный цикл ПО – процесс, состоящий из конкретных этапов который начинается в момент принятия решения о необходимости создания ПО и заканчивается в момент прекращения поддержки ПО разработчиками

7 основных этапов:

* Идея
* Определение требований
* Дизайн системы (определяются используемые технологии, и т.п.)
* Разработка
* Тестирование (отклонения = отчеты о дефектах => исправления)
* Развертывание
* Поддержка (определение => разработка => тестирование => разверт. | самый длительный этап.)
* Закрытие (не обязательный)

Жизнь проектов: простой = пару месяцев, сложный = более 15 лет, большинство = 6-8 лет.

1. Технология структурного программирования

Существует два подхода к составлению сложных программ:

* Сверху вниз / Нисходящее
* Снизу вверх / Восходящее

Структурное программирование – технология разработки ПО, в основе которой лежит представление программы в виде иерархической структуры логически целостных фрагментов

Идея структурного программирования – стандартизация для борьбы с ошибками. Алгоритм называется структурным, если он имеет одну из базовых структур.

Структурное программирование – процесс программирования на алгоритмическом языке с использованием ограниченного набора базовых конструкций – линейной, ветвящейся и циклической.

Модульное программирование – программирование, при котором программа представляется в виде совокупности логически связанных модулей

ООП – программирование, при котором основой программы является объект, для которого определенны совокупности данных и методы их обработки

1. Инструментальные средства оформления и документирования алгоритмов программ

Инструментальное программное обеспечение (ИПО) – программное обеспечение, предназначенное для использования в ходе проектирования, разработки и сопровождения программ

Его задачи:

* Создание теста разрабатываемой программы и использованием специально установленных кодовых слов
* Перевод текста создаваемой программы в машинно-ориентированный код, доступный для распознавание ЭВМ
* Соединение отдельных модулей в единый исполняемый код, с соблюдением необходимой структуры
* Тестирование и контроль созданной программы, выявление и устранение формальных, логических и синтаксических ошибок

Средства оформления:

* Текст
* Схемы алгоритмов
* Псевдокод
* Запись в форме программы на ЯП
* Диаграммы Нэсси и Дейкстры

1. Основные принципы ООП. Классы C#: основные понятия

ООП – программирование, при котором основой программы является объект, для которого определены совокупности данных и методы их обработки

C# был разработан Microsoft на основе C и C++ для работы с .NET. Можно создавать: Win приложения, Web приложения и службы. Является объектное ориентированным языком (ОО), и поддерживает инкапсуляцию, наследование и полиморфизм (Десятичные через ТОЧКУ)

Пространство имеет – контейнеры для классов

Класс – логическая единица программы, в которой содержатся методы и другие элементы

Метод / функция – программа программы, в которой содержится реализация алгоритма

Типы данных: значимые и ссылочные

Объект – экземпляр класса. Для получения данных объекта или вызова методов используется точка «.»

1. Перегрузка методов C#

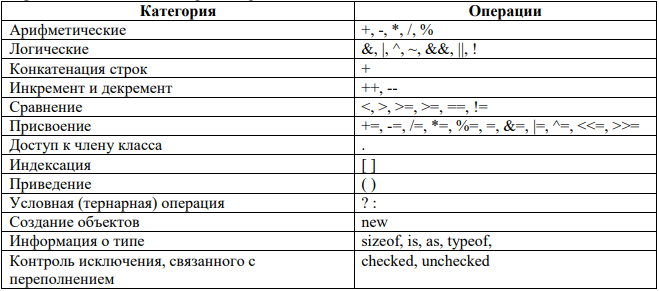
Перегрузка метода – использование метода с одним и тем же именем, но с различным количеством и типов параметров. Компилятор по типу параметров сам определяет, какой метод требуется вызвать (разрешение перегрузки).

Перегрузка методов – простейшая реализация полиморфизма

В C# можно создать в классе несколько методов с одним и тем же именем, но с разной сигнатурой

Сигнатура – совокупность: Имя метода, количество параметров, типы параметров, порядок параметров, модификаторы параметров

1. Операции класса в C#



1. Иерархия классов C#

-

1. Синтаксис интерфейсов в С#

Интерфейс – перечисление методов, которые должны быть обязательно реализованы у класса. Интерфейс дает указать из чего именно должен состоять тот или иной объект.

Интерфейс – ссылочный тип, который может определять некоторый функционал – набор методов и свойств без реализации

Интерфейс содержит: методы, свойства, события, индексаторы  
Интерфейс не содержит: константы, поля, операторы, конструкторы, типы

Интерфейс – список объявлений методов и свойств.

Объявление интерфейса:

[модификатор уровня доступа] interface имя { тело }

Модификатор – public / internal

Имя – с заглавной буквы

Тело – методы, свойства и т.п.

1. Интерфейсы и наследование C#

Наследование – процесс, посредством которого один объект может приобретать свойства другого. Для объекта нужно определить потомков, которые наследуют, корректируют или дополняют его поведение.

Синтаксис: class Производный\_Класс : Базовый\_класс { данные }

Производный класс – разновидность объекта базового класса

Базовый класс – общие характеристики и поведение

При применении наследования интерфейса указывается двоеточие и затем названия применяемых интерфейсов. При этом класс должен реализовать все методы и свойства применяемых интерфейсов, если эти методы и свойства не имеют реализации по умолчанию

Наследование – нельзя применить модификатор sealed, abstract, можно использовать new

1. Структуры в C#

Структура – значимый тип данных. Экземпляр структуры размещается в стеке, а не в динамической памяти

Синтаксис: [спецификаторы] struct <имя> [:интерфейсы] { тело }

Спецификаторы – public, internal, для вложенных структур можно private

Тело может содержать констенты, поля, методы, свойства, события, индексаторы, операции, конструкторы

Правила:

* Не могут участвовать в иерархиях наследования, но могут реализовывать интерфейсы
* Нельзя определить конструктор без параметров
* Нельзя определять декнстуркторы
* Не могут быть абстрактными
* Не может содержать абстрактные или вирт. Методы
* Переопределяется со спецификатором override могут лишь методы, унаследованные от класса object

Объекст структуры можно создать при помощи new:  
<имя структуры> <имя экземпляра> = new <имя структ.>(<параметры>);

Без использования new надо обязательно проанализировать все поля перед получением значений или вызовом методов

Преимущество – обрабатываются напрямую, а не через ссылки => не требует отдельной ссылочкой переменной (Если нужно хранить небольшую группу связанных данный, но не нужно обеспечивать наследование)

1. Делегаты

Делегат - объект, который может ссылаться на метод

Когда создается делегат, создается объект, содержащий ссылку на метод

Синтаксис: delegate возвращ\_тип имя(список\_параметров);

Список параметров – сигнатура (параметры, необходимые для методов вызываемых этим делегатом)

1. Регулярные выражения

Регулярное выражение – шаблон, по которому выполняется поиск соответствующего ему фрагмента текста

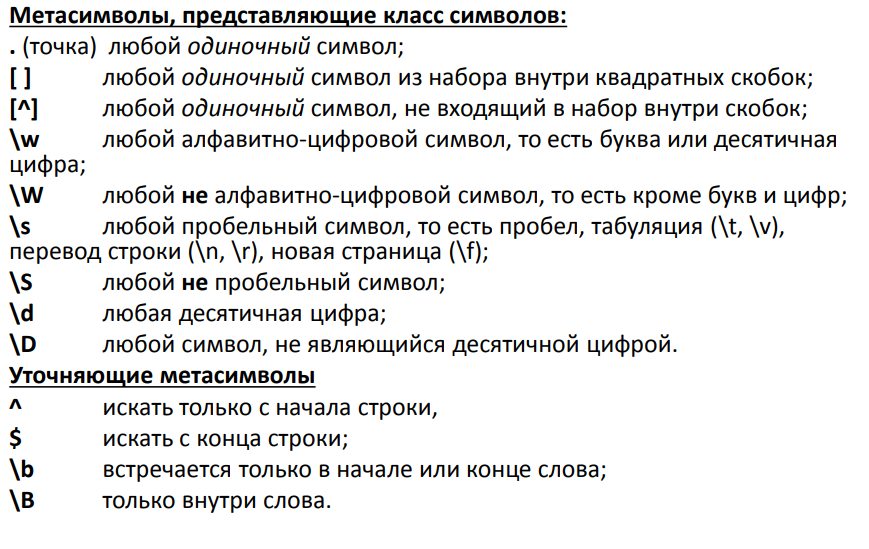
Предназначены для поиска в тесте фрагментов (символов, строк) по заданному шаблону с целью дальнейшей обработки

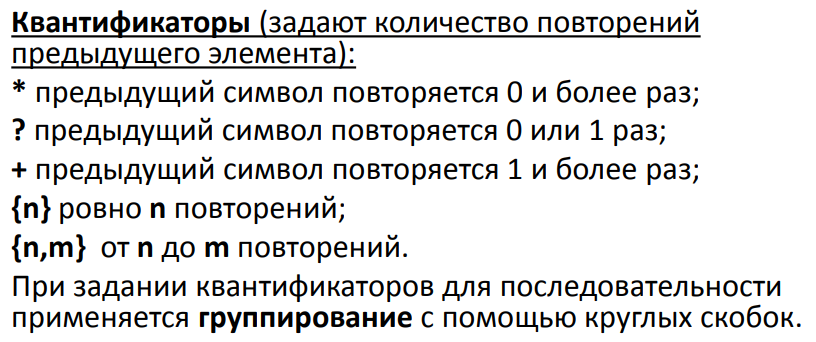
Пространство имен: System.Text.RegularExpressions. Центральный класс – Regex

Язык описания содержит обычные символы и метасимволы

Обычный – представляет в шаблоне сам себя

Метасимвол – класс символов, уточняющий символ или квантификатор (например \n или A{3} – букву а повторить 3 раза)





1. Коллекции

Коллекция – некоторая группа объектов. Упрощают реализацию многих задач программирования

.NET Framework поддерживает 3 типа коллекций:

* Общего назначения
* Специализированные
* Ориентированные на побитовую организацию данных

Коллекция – объединение произвольного количество объектов, возможно разного типа

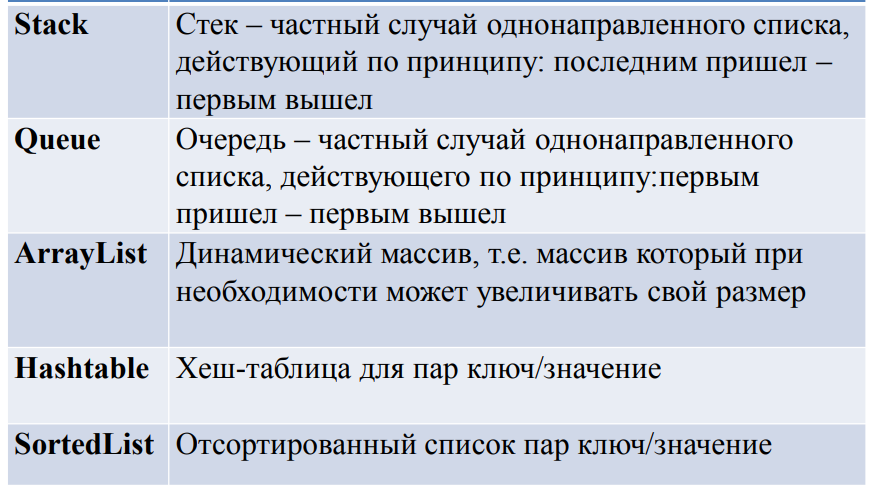
Коллекция – динамическая структура данных. Память под кеоллекцию не фиксируется.

Коллекция – более гибкая организация данных, чем массив. Но работа с массивом ведется быстрее

Пространство имен: System.Collections / System.Colletions.Generic

Структуры данных: сетки, очереди, динамические массивы, словари, отсортированный список

Коллекции специального назначения: System.Collections.Specialized



Наиболее используеммые коллекции: List, Dictionary, ArrayList, Hashtable, Stack, Queue

1. Указатели C#

-

1. Операции со списками C#

-

1. Назначение и виды паттернов

Паттерн проектирования – определённый способ построение программного кода для решения часто встречающихся проблем

Плюсы:

* Каждый паттерн описывает решение целого класса проблем
* Каждый паттерн имеет известное имя
* Шаблоны не зависят от языка

Идиома – низкоуровневый шаблон

Программная идиома – выражение, обозначающее элементарную конструкцию.

Паттерн состоит из:

* Проблема
* Мотивация к решению
* Структуры классов
* Пример на одном из языков
* Особенность реализации
* Связи с другими паттернами

Классификация:

* Идиомы – самые низкоуровневые
* Универсальные – архитектурные паттерны
* Порождающие
* Структурные
* Поведенческие

1. Событийно – управляемое программирование C#

Инетрфейс пользователя – набор средств для взаимодействия

Взаимодействие объектов программы организуется по принципу «источник-наблюдатель»

Собыие – член класса, который позволяет компактно реализовать взаимодействие «источник-наблюдатель»

Событие – синтаксическая обертка над частным делегатным полем, реализующая add и remove

Синтаксис: [модификаторы] event тип имя

1. Элементы управления. Диалоговые окна. Обработчики событий

Form - добавляет значительную функциональность базовому классу Control (Производный класс от Container Control)

Организованный интерфейс бывает: SDI – иерархия форм отсутствует, все формы равны; MDi – есть иерархия форм: есть родительская и дочерние формы (кона будут внутри родительской)

Модальная форма – не позволяет переключатсья на другие формы этого же приложения, пока не будет завершена работа с текущим | ShowDialog

Немодальная – поволяет переключаться на другие окна приложения | Show

Временно скрыть форму – Visible

Закрыть – Close

Оборатиться к форме из класса – This

**События**

Load – инициализация

Activated – форма видима и текущая, но пользовательская область не отрисована

Deactivated – форма не активна

Cloasing – процесс закрытия

Closed – после закрытия

**Виды окон:**

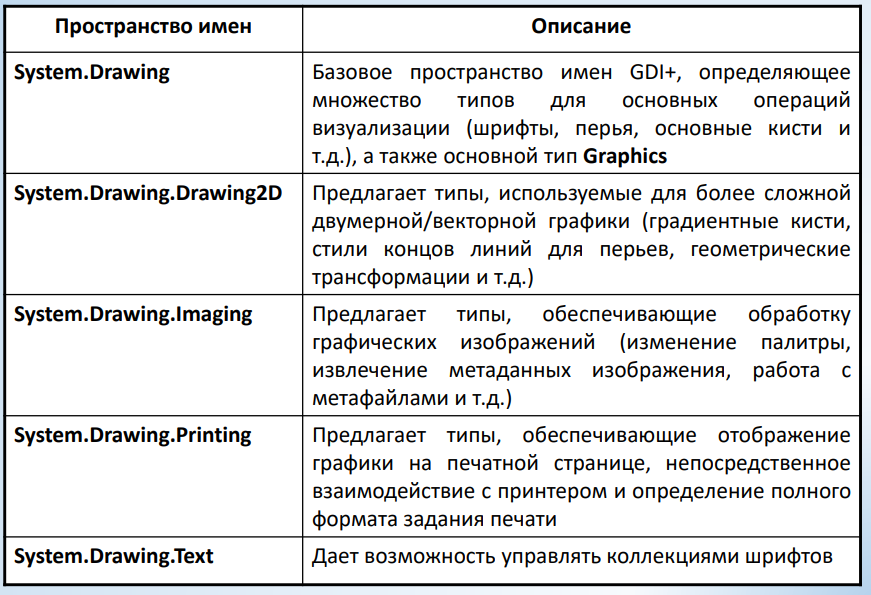
* Основные окна
* Диалоговые окна
* Элементы управления

Диалоговое окно – неизменяемые размеры, отсутствие кнопок восстановления и свертывания, наличие кнопок «Ок» и им подобным, установленное свойство DialogResult

1. Введение в графику C#

Для работы с графикой есть специальный набор функций – GDI. .NET обеспечивает набор пространств имен, которые формируют набор возможностей .NET – GDI+

Три направления применения GDI+: текст, растровая и векторная графики



1. Методы оптимизации программного кода

Оптимизация программы – обработка, связанная с переупорядочиванием и изменением операций в компилируемой программе, с целью получения более эффективной результирующей объектной программы.

Может выполнятся несколько раз на этапе генерации кода. Два вида оптимизирующих преобразований: преобразования исходной программы и результирующей объектной программы

Общие принципы:

* Выделение оптимизации в отдельный этап генерации кода
* Оптимизация – необязательный этап компиляции
* Может выполнятся для следующих типовых конструкций:
  + Линейные участки
  + Лог выражения
  + Циклы
  + Вызов процедур функций
  + Другие конструкции входного языка

1. Цели и методы рефакторинга

Рефакторинг – постепенная работа над кодом, не влияющая на функциональность ПО, но делающая его более простым для чтения и гибким

Рефакторинг нужен для того, что бы сделать код программы легче для понимания. Рефакторинг != оптимизация / реинжеринга. Результат рефакторинга – чистый код и простой дизайн.

Чистый код – код, который просто читать и понимать

1. Правила разработки интерфейсов пользователя