**Оглавление**

[1.         Основные понятия архитектуры информационных систем. Архитектура ИС. Структуры ИС (физическая, логическая, программная, функциональная) и их взаимосвязь. 2](#_Toc132197830)

[2. Техническое задание. Состав и содержание 2](#_Toc132197831)

[3. Подсистемы ИС. Основные концептуальные принципы функционирования и построения. Подсистемы обеспечения работоспособности ИС. 3](#_Toc132197832)

[4. Информационное, техническое, программное, математическое и другие виды обеспечения. Их характеристика и состав. 3](#_Toc132197833)

[5. Классификация информационных систем. Требования, предъявляемые к информационным системам. 4](#_Toc132197834)

[6. Платформы серверов ИС и их аппаратно-программные характеристики. 4](#_Toc132197835)

[7. Характеристики аппаратно-программных платформ ИС и их виды 5](#_Toc132197836)

[8. Программное обеспечение ИС и его классификация. Серверное и клиентское программное обеспечение ИС. 5](#_Toc132197837)

[9. Средства автоматизации проектирования и разработки корпоративных систем, их основные типы и классификация. 6](#_Toc132197838)

[10. Корпоративные стандарты и методики: виды стандартов. Методика CDM фирмы Oracle. Международный стандарт ISO/IEC 12207. 6](#_Toc132197839)

[11. Методология и технология разработки информационных систем. Методология RAD. Фазы жизненного цикла в рамках методологии RAD. 7](#_Toc132197840)

[12. Понятие профиля информационной системы. Принципы формирования профиля информационной системы. Структура профилей информационных систем. 7](#_Toc132197841)

[13. Основные фазы проектирования информационной системы. Основные процессы жизненного цикла. 8](#_Toc132197842)

[14. Понятие алгоритмической системы и программного интерфейса. Виды алгоритмов. Блок-схемы. Графическая реализация алгоритмов. 8](#_Toc132197843)

[15. Объектно-ориентированная методология программирования. Понятия объект и класс. Контроль доступа и методы доступа. 9](#_Toc132197844)

[16. Основные принципы объектно-ориентированного программирования. 9](#_Toc132197845)

[17. Язык C# и платформа .NET. 9](#_Toc132197846)

[18. Простейшие операторы языка C#: операторы организации цикла, операторы перехода. 10](#_Toc132197847)

[19. Простейшие операторы языка C#: оператор присваивания, составной оператор, операторы ветвления. 11](#_Toc132197848)

[20. Простейшие действия с данными: оператор и операция. Операции ввода, вывода, преобразование типов, операторы отношений, логические операторы. 12](#_Toc132197849)

[21. Структура программы на языке C#. Приложение (решение), проект, пространство имен, класс, методы, данные. 12](#_Toc132197850)

[22. Массивы, строки и списки в С#. 12](#_Toc132197851)

[23. Обработка исключений в C#. Класс Exception 13](#_Toc132197852)

[24. Платформа WPF, её особенности. 13](#_Toc132197853)

[25. Язык разметки XAML. Структура и основные элементы. 14](#_Toc132197854)

# 1.         Основные понятия архитектуры информационных систем. Архитектура ИС. Структуры ИС (физическая, логическая, программная, функциональная) и их взаимосвязь.

*Архитектура информационных систем* (АИС) - это описание структуры, компонентов, связей и принципов функционирования информационной системы (ИС). АИС определяет, как компоненты ИС взаимодействуют друг с другом, как они организованы и как они обеспечивают выполнение задач ИС.

Физическая структура ИС определяет физические компоненты ИС, такие как компьютеры, серверы, сетевое оборудование и другие устройства, которые используются для хранения, обработки и передачи данных.

Логическая структура ИС определяет логические компоненты ИС, такие как базы данных, приложения, интерфейсы и другие компоненты, которые используются для обработки и управления данными.

Программная структура ИС определяет программные компоненты ИС, такие как операционные системы, приложения, библиотеки и другие программные средства, которые используются для обработки данных и выполнения задач ИС.

Функциональная структура ИС определяет функциональные компоненты ИС, такие как модули, подсистемы и другие компоненты, которые используются для выполнения конкретных задач ИС.

Взаимосвязь между структурами ИС заключается в том, что они взаимодействуют друг с другом для обеспечения выполнения задач ИС. Физическая структура обеспечивает хранение и передачу данных, логическая структура обеспечивает управление данными, программная структура обеспечивает обработку данных, а функциональная структура обеспечивает выполнение конкретных задач ИС. Все эти структуры взаимодействуют друг с другом для обеспечения эффективной работы ИС.

# 2. Техническое задание. Состав и содержание

*Техническое задание* (ТЗ) - это документ, который содержит описание требований к созданию, разработке или модификации продукта или услуги. ТЗ является основным документом, который используется для описания требований заказчика к продукту или услуге, и является основой для дальнейшей работы разработчиков.

Состав технического задания может включать следующие разделы:

1*. Введение* - в этом разделе описывается цель создания продукта или услуги, а также краткое описание требований заказчика.

2. *Общие требования* - в этом разделе описываются общие требования к продукту или услуге, такие как требования к надежности, безопасности, производительности и т.д.

3. *Функциональные требования* - в этом разделе описываются функциональные требования к продукту или услуге, такие как требования к функциональности, интерфейсу, взаимодействию с другими системами и т.д.

4. *Требования к производительности* - в этом разделе описываются требования к производительности продукта или услуги, такие как требования к скорости работы, времени отклика и т.д.

5. *Требования к безопасности* - в этом разделе описываются требования к безопасности продукта или услуги, такие как требования к защите от несанкционированного доступа, защите от вирусов и т.д.

6. *Требования к документации* - в этом разделе описываются требования к документации, которая должна быть предоставлена вместе с продуктом или услугой, такие как требования к инструкциям по эксплуатации, руководствам пользователя и т.д.

7. *Требования к тестированию* - в этом разделе описываются требования к тестированию продукта или услуги, такие как требования к тестовым сценариям, критериям приемки и т.д.

8. *Требования к сопровождению* - в этом разделе описываются требования к сопровождению продукта или услуги, такие как требования к технической поддержке, обновлениям и т.д.

9*. График работ* - в этом разделе описывается график работ по созданию продукта или услуги, включая сроки выполнения каждого этапа работы.

10. *Бюджет* - в этом разделе описывается бюджет на создание продукта или услуги, включая расходы на материалы, оборудование, трудовые ресурсы и т.д.

Содержание технического задания может варьироваться в зависимости от конкретных требований заказчика и характера проекта.

# 3. Подсистемы ИС. Основные концептуальные принципы функционирования и построения. Подсистемы обеспечения работоспособности ИС.

*Подсистемы информационной системы* (ИС) - это части системы, которые выполняют определенные функции и взаимодействуют друг с другом для обеспечения работы всей системы. Основные концептуальные принципы функционирования и построения подсистем ИС включают:

1. *Разделение функций*: каждая подсистема выполняет определенную функцию, которая не пересекается с функциями других подсистем.

2. *Интеграция*: подсистемы взаимодействуют друг с другом для обеспечения работы всей системы.

3. *Модульность*: подсистемы могут быть разработаны и изменены независимо друг от друга.

4. *Стандартизация*: использование стандартов обеспечивает совместимость и взаимодействие между подсистемами.

Подсистемы обеспечения работоспособности ИС включают:

1. *Подсистема мониторинга*: отслеживает работу системы и выявляет проблемы.

2*. Подсистема резервного копирования*: создает резервные копии данных для защиты от потери информации.

3. *Подсистема восстановления*: восстанавливает работу системы после сбоев.

4. *Подсистема безопасности*: обеспечивает защиту от несанкционированного доступа и вредоносных программ.

5. *Подсистема обновления*: обновляет программное обеспечение и обеспечивает совместимость с другими подсистемами.

# 4. Информационное, техническое, программное, математическое и другие виды обеспечения. Их характеристика и состав.

Информационное, техническое, программное, математическое и другие виды обеспечения - это различные компоненты, которые используются для создания, разработки и поддержки информационных систем.

*Информационное обеспечение* - это компоненты, которые обеспечивают доступность, точность, полноту и своевременность информации. Оно включает в себя базы данных, информационные системы, документацию, отчеты и другие средства для обработки и хранения информации.

*Техническое обеспечение* - это компоненты, которые обеспечивают работоспособность аппаратного обеспечения и средств связи. Оно включает в себя компьютеры, серверы, сетевое оборудование, периферийные устройства и другие технические средства.

*Программное обеспечение* - это компоненты, которые обеспечивают функциональность и возможности информационной системы. Оно включает в себя операционные системы, прикладное программное обеспечение, библиотеки, фреймворки и другие программные средства.

*Математическое обеспечение* - это компоненты, которые обеспечивают математическую обработку данных и анализ. Оно включает в себя алгоритмы, моделирование, статистические методы и другие математические средства.

Другие виды обеспечения могут включать в себя средства для обеспечения безопасности, управления проектами, тестирования, документирования и другие.

Каждый вид обеспечения имеет свои характеристики и состав, которые зависят от конкретных требований и задач информационной системы. Однако, все они взаимодействуют друг с другом для обеспечения работы всей системы.

# 5. Классификация информационных систем. Требования, предъявляемые к информационным системам.

Классификация информационных систем может быть основана на различных критериях, таких как:

1. *По масштабу использования*: персональные, корпоративные, государственные, международные.

2. *По функциональности*: управленческие, бухгалтерские, производственные, научные, медицинские, транспортные и т.д.

3. *По способу обработки информации*: автоматизированные, полуавтоматизированные, ручные.

4. *По типу обрабатываемой информации*: текстовые, графические, звуковые, видео, мультимедийные.

5. *По способу доступа к информации*: локальные, распределенные, интернет-системы.

Требования, предъявляемые к информационным системам, могут включать в себя:

1. *Надежность*: система должна быть стабильной и надежной, чтобы обеспечить бесперебойную работу.

2. *Безопасность*: система должна быть защищена от несанкционированного доступа и вредоносных программ.

3. *Производительность*: система должна обеспечивать быстродействие и эффективность при обработке информации.

4. *Масштабируемость*: система должна быть способна масштабироваться в зависимости от изменения объема обрабатываемой информации.

5. *Удобство использования*: система должна быть удобной и простой в использовании для пользователей.

6. *Совместимость*: система должна быть совместима с другими системами и программным обеспечением.

7. *Доступность*: система должна быть доступна для использования в любое время и из любой точки мира.

8. *Соответствие стандартам*: система должна соответствовать стандартам и требованиям, установленным в отрасли.

9. *Поддержка*: система должна иметь поддержку и обслуживание для обеспечения ее бесперебойной работы.

# 6. Платформы серверов ИС и их аппаратно-программные характеристики.

Платформы серверов информационных систем могут иметь различные аппаратно-программные характеристики в зависимости от требований к производительности, масштабируемости, надежности и других факторов.

Некоторые из популярных платформ серверов включают в себя:

1. *Windows Server* - операционная система сервера от Microsoft, которая поддерживает множество приложений и сервисов, таких как Active Directory, Exchange Server, SharePoint и другие. Она может работать на различных аппаратных платформах, включая x86, x64 и ARM.

2. *Linux* - свободная операционная система, которая также может использоваться в качестве платформы сервера. Она поддерживает множество приложений и сервисов, таких как Apache, MySQL, PostgreSQL и другие. Linux может работать на различных аппаратных платформах, включая x86, x64, ARM и другие.

3*. IBM Power Systems* - платформа серверов от IBM, которая использует процессоры Power и операционную систему AIX. Она обеспечивает высокую производительность и масштабируемость, а также поддерживает множество приложений и сервисов.

4. *Oracle SPARC* - платформа серверов от Oracle, которая использует процессоры SPARC и операционную систему Solaris. Она обеспечивает высокую производительность и надежность, а также поддерживает множество приложений и сервисов.

Аппаратно-программные характеристики платформ серверов могут включать в себя процессоры, оперативную память, жесткие диски, сетевые интерфейсы, операционные системы, программное обеспечение для управления серверами и другие компоненты. Важно выбирать платформу сервера, которая соответствует требованиям информационной системы и обеспечивает необходимую производительность, масштабируемость и надежность.

7. Характеристики аппаратно-программных платформ ИС и их виды**.**

Аппаратно-программные платформы информационных систем (ИС) могут иметь различные характеристики в зависимости от требований к производительности, масштабируемости, надежности и других факторов.

Некоторые из характеристик аппаратно-программных платформ ИС включают в себя:

1. *Процессоры* - это основной компонент аппаратной платформы ИС, который обеспечивает выполнение вычислительных задач. Характеристики процессоров могут включать в себя такие параметры, как тактовая частота, количество ядер, кэш-память и другие.

2. *Оперативная память* - это компонент аппаратной платформы ИС, который используется для временного хранения данных и программ во время их выполнения. Характеристики оперативной памяти могут включать в себя такие параметры, как объем, скорость и тип.

3. *Жесткие диски* - это компонент аппаратной платформы ИС, который используется для хранения данных на постоянной основе. Характеристики жестких дисков могут включать в себя такие параметры, как объем, скорость, интерфейс и другие.

4. *Сетевые интерфейсы* - это компонент аппаратной платформы ИС, который используется для обмена данными между различными компонентами ИС. Характеристики сетевых интерфейсов могут включать в себя такие параметры, как скорость передачи данных, тип интерфейса и другие.

5. *Операционные системы* - это программное обеспечение, которое управляет аппаратной платформой ИС и обеспечивает выполнение приложений и сервисов. Характеристики операционных систем могут включать в себя такие параметры, как производительность, масштабируемость, надежность и другие.

6. *Программное обеспечение для управления ИС* - это программное обеспечение, которое используется для управления и настройки аппаратной платформы ИС и ее компонентов. Характеристики программного обеспечения для управления ИС могут включать в себя такие параметры, как функциональность, производительность, масштабируемость и другие.

Виды аппаратно-программных платформ ИС могут включать в себя серверы, рабочие станции, мобильные устройства, встроенные системы и другие. Каждый тип платформы ИС имеет свои уникальные характеристики и применения в зависимости от требований к ИС.

# 8. Программное обеспечение ИС и его классификация. Серверное и клиентское программное обеспечение ИС.

*Программное обеспечение информационной системы* (ИС) - это набор программных компонентов, которые обеспечивают функционирование ИС и позволяют ей выполнять свои задачи.

Классификация программного обеспечения ИС может быть основана на различных критериях, таких как тип обрабатываемой информации, функциональность, масштаб использования, метод обработки и другие.

*Серверное программное обеспечение ИС* - это программное обеспечение, которое устанавливается на серверах и обеспечивает функционирование ИС на стороне сервера. Оно может включать в себя операционную систему, базы данных, веб-серверы, приложения для обработки данных и другие компоненты.

*Клиентское программное обеспечение ИС* - это программное обеспечение, которое устанавливается на компьютерах пользователей и обеспечивает доступ к ИС. Оно может включать в себя приложения для работы с данными, браузеры, почтовые клиенты и другие компоненты.

Обычно серверное и клиентское программное обеспечение ИС взаимодействуют друг с другом, чтобы обеспечить полноценное функционирование ИС.

# 9. Средства автоматизации проектирования и разработки корпоративных систем, их основные типы и классификация.

*Средства автоматизации проектирования и разработки корпоративных систем* - это программные инструменты, которые позволяют автоматизировать процессы проектирования, разработки, тестирования и сопровождения корпоративных систем. Они помогают ускорить и упростить процессы разработки, повысить качество и надежность системы, а также снизить затраты на ее создание и сопровождение.

Основные типы средств автоматизации проектирования и разработки корпоративных систем:

1. *Интегрированные среды разработки (IDE*) - это программные средства, которые объединяют в себе различные инструменты для разработки, отладки и тестирования программного обеспечения. Они позволяют ускорить процесс разработки и повысить качество кода.

2. *Системы управления версиями (VCS)* - это программные инструменты, которые позволяют управлять изменениями в исходном коде программного обеспечения. Они позволяют отслеживать изменения, вносимые в код, и восстанавливать предыдущие версии кода.

3*. Средства моделирования* - это программные инструменты, которые позволяют создавать модели системы и ее компонентов. Они позволяют ускорить процесс проектирования и снизить риски ошибок в реализации системы.

4. *Средства автоматического тестирования* - это программные инструменты, которые позволяют автоматизировать процесс тестирования системы. Они позволяют ускорить процесс тестирования и повысить качество системы.

Классификация средств автоматизации проектирования и разработки корпоративных систем может быть основана на различных критериях, таких как тип задач, которые они решают, методы автоматизации, используемые в них, и другие. Например, средства автоматизации могут быть классифицированы как CASE-системы (Computer-Aided Software Engineering), которые позволяют автоматизировать процессы проектирования и разработки систем, или как DevOps-системы, которые позволяют автоматизировать процессы разработки и сопровождения системы.

# 10. Корпоративные стандарты и методики: виды стандартов. Методика CDM фирмы Oracle. Международный стандарт ISO/IEC 12207.

*Корпоративные стандарты и методики* - это наборы правил, процедур и рекомендаций, которые используются компаниями для обеспечения единого подхода к разработке, тестированию и сопровождению программного обеспечения. Существует несколько видов корпоративных стандартов, включая стандарты, разработанные отдельными компаниями, стандарты, разработанные отраслевыми организациями, и международные стандарты.

*Методика CDM (Custom Development Methodology) фирмы Oracle* - это методика разработки программного обеспечения, которая предоставляет набор инструментов и процедур для разработки приложений на платформе Oracle. Методика CDM включает в себя шаги, такие как анализ требований, проектирование, разработка, тестирование и внедрение, а также рекомендации по использованию инструментов Oracle для каждого из этих шагов.

*Международный стандарт ISO/IEC 12207* - это стандарт, который определяет процессы жизненного цикла программного обеспечения, которые должны быть выполнены при разработке, тестировании и сопровождении программного обеспечения. Стандарт ISO/IEC 12207 включает в себя процессы, такие как управление проектом, управление конфигурацией, разработка, тестирование, сопровождение и документирование. Этот стандарт является международным и широко используется в индустрии программного обеспечения.

# 11. Методология и технология разработки информационных систем. Методология RAD. Фазы жизненного цикла в рамках методологии RAD.

*Методология и технология разработки информационных систем* - это набор процессов, методов и инструментов, которые используются для создания информационных систем. Она включает в себя различные методологии, такие как RAD, Agile, Waterfall и другие, а также технологии, такие как базы данных, языки программирования, среды разработки и т.д.

*Методология RAD (Rapid Application Development)* - это методология разработки программного обеспечения, которая основана на быстрой разработке итеративных прототипов. Она позволяет быстро создавать и тестировать прототипы, чтобы получить обратную связь от заказчика и уточнить требования к системе. Методология RAD включает в себя несколько фаз жизненного цикла.

Фазы жизненного цикла в рамках методологии RAD:

1. *Планирование* - определение целей и требований к системе, а также определение ресурсов и сроков.

2*. Анализ* - сбор и анализ требований к системе, определение функциональности и проектирование архитектуры.

3. *Дизайн* - создание детального дизайна системы, включая интерфейсы пользователя, базы данных и алгоритмы.

4*. Разработка* - создание кода и тестирование системы.

5. *Тестирование* - проверка работоспособности системы и выявление ошибок.

6. *Внедрение* - установка и настройка системы на рабочих местах пользователей.

7. *Сопровождение* - поддержка и обновление системы после ее внедрения.

Методология RAD позволяет быстро создавать итеративные прототипы, что ускоряет процесс разработки и позволяет быстро получать обратную связь от заказчика. Однако, она может быть неэффективной для больших и сложных проектов, где требуется более детальное планирование и управление рисками.

# 12. Понятие профиля информационной системы. Принципы формирования профиля информационной системы. Структура профилей информационных систем.

*Профиль информационной системы* - это описание ее характеристик и свойств, которые определяют ее функциональность, производительность, надежность, безопасность и другие аспекты. Профиль информационной системы включает в себя описание ее аппаратной и программной составляющей, а также ее функциональных возможностей и требований к окружающей среде.

Принципы формирования профиля информационной системы включают в себя следующие аспекты:

1. *Анализ требований к системе*. Необходимо определить функциональные и нефункциональные требования к системе, а также ее цели и задачи.

2. *Определение архитектуры системы*. Необходимо определить аппаратную и программную архитектуру системы, а также ее компоненты и связи между ними.

3. *Определение требований к производительности*. Необходимо определить требования к производительности системы, такие как скорость работы, объем обрабатываемых данных и другие параметры.

4. *Определение требований к безопасности*. Необходимо определить требования к безопасности системы, такие как защита от несанкционированного доступа, защита от вирусов и других угроз.

Структура профилей информационных систем может включать в себя следующие элементы:

1. Описание аппаратной и программной архитектуры системы.

2. Описание функциональных возможностей системы.

3. Описание требований к производительности системы.

4. Описание требований к безопасности системы.

5. Описание требований к окружающей среде.

6. Описание требований к поддержке и обслуживанию системы.

7. Описание требований к документации и обучению пользователей.

# 13. Основные фазы проектирования информационной системы. Основные процессы жизненного цикла.

Основные фазы проектирования информационной системы включают:

1. *Планирование* - определение целей и задач, выбор методологии и технологий, определение требований к системе и ее функциональности.

2. *Анализ* - изучение существующих процессов и систем, определение потребностей пользователей, выявление проблем и возможностей для улучшения.

3. *Проектирование* - разработка архитектуры системы, выбор технологий и инструментов, создание диаграмм и моделей, определение интерфейсов и функциональности.

4. *Разработка* - создание кода и тестирование системы, интеграция компонентов, отладка и оптимизация.

5. *Внедрение* - установка и настройка системы, обучение пользователей, перенос данных и запуск системы в реальной среде.

6. *Сопровождение* - поддержка и обновление системы, исправление ошибок и улучшение функциональности.

Основные процессы жизненного цикла информационной системы включают:

1. *Планирование* - определение целей и задач, выбор методологии и технологий, определение требований к системе и ее функциональности.

2. *Анализ* - изучение существующих процессов и систем, определение потребностей пользователей, выявление проблем и возможностей для улучшения.

3. *Проектирование* - разработка архитектуры системы, выбор технологий и инструментов, создание диаграмм и моделей, определение интерфейсов и функциональности.

4. *Разработка* - создание кода и тестирование системы, интеграция компонентов, отладка и оптимизация.

5*. Внедрение* - установка и настройка системы, обучение пользователей, перенос данных и запуск системы в реальной среде.

6. *Сопровождение* - поддержка и обновление системы, исправление ошибок и улучшение функциональности.

# 14. Понятие алгоритмической системы и программного интерфейса. Виды алгоритмов. Блок-схемы. Графическая реализация алгоритмов.

*Алгоритмическая система* - это система, которая использует алгоритмы для решения задач. Она состоит из алгоритмов, структур данных, программного обеспечения и аппаратных средств, необходимых для выполнения алгоритмов.

*Программный интерфейс* - это набор функций, процедур и протоколов, которые позволяют программам взаимодействовать друг с другом и с аппаратным обеспечением.

Виды алгоритмов:

1. *Последовательный алгоритм* - это алгоритм, который выполняет действия последовательно, одно за другим.

2. *Рекурсивный алгоритм* - это алгоритм, который вызывает сам себя для решения задачи.

3. *Итеративный алгоритм* - это алгоритм, который использует циклы для повторения действий.

4. *Рандомизированный алгоритм* - это алгоритм, который использует случайные числа для принятия решений.

*Блок-схемы* - это графическое представление алгоритма, которое использует блоки для представления действий и стрелки для связи между блоками.

*Графическая реализация алгоритмов* - это способ представления алгоритмов в виде графических элементов, таких как блоки, стрелки и текстовые поля. Это позволяет легче понимать и отлаживать алгоритмы, а также упрощает их визуализацию и документирование. Примерами графических языков программирования являются блок-схемы, UML-диаграммы и диаграммы потоков данных.

# 15. Объектно-ориентированная методология программирования. Понятия объект и класс. Контроль доступа и методы доступа.

*Объектно-ориентированная методология программирования (ООП)* - это подход к программированию, основанный на использовании объектов, которые являются экземплярами классов. ООП позволяет создавать более гибкие, модульные и расширяемые программы.

*Объект* - это экземпляр класса, который имеет свои собственные значения свойств и может выполнять методы, определенные в классе.

*Класс* - это шаблон, определяющий свойства и методы, которые будут использоваться объектами, созданными на основе этого класса. Класс может быть использован для создания множества объектов.

*Контроль доступа* - это механизм, который определяет, какие свойства и методы класса могут быть доступны извне. В ООП используются три уровня контроля доступа:

1. *Public* - свойства и методы, объявленные как public, могут быть доступны из любого места программы.

2. *Private* - свойства и методы, объявленные как private, могут быть доступны только внутри класса.

3. *Protected* - свойства и методы, объявленные как protected, могут быть доступны только внутри класса и его наследников.

*Методы доступа* - это методы, которые позволяют получить или изменить значения свойств объекта. В ООП используются два метода доступа:

1. *Геттеры (get*) - методы, которые позволяют получить значение свойства объекта.

2. *Сеттеры (set)* - методы, которые позволяют изменить значение свойства объекта.

Геттеры и сеттеры могут быть объявлены как public, private или protected, в зависимости от уровня контроля доступа, который необходим для данного свойства.

# 16. Основные принципы объектно-ориентированного программирования.

Основные принципы объектно-ориентированного программирования (ООП) включают в себя:

1. *Инкапсуляция*: это принцип, который позволяет скрыть детали реализации объекта от других объектов. Это достигается путем объединения данных и методов, которые работают с этими данными, в одном объекте.

2. *Наследование*: это принцип, который позволяет создавать новые классы на основе уже существующих классов. Новый класс наследует свойства и методы родительского класса и может добавлять свои собственные свойства и методы.

3. *Полиморфизм*: это принцип, который позволяет объектам одного класса использовать методы другого класса. Это достигается путем создания методов с одинаковыми именами в разных классах.

4. *Абстракция*: это принцип, который позволяет создавать абстрактные классы и интерфейсы, которые определяют общие свойства и методы для группы классов. Абстрактные классы и интерфейсы не могут быть созданы в качестве объектов, но могут быть использованы для создания объектов.

Эти принципы помогают создавать более гибкие, модульные и расширяемые программы, которые легче поддерживать и изменять.

# 17. Язык C# и платформа .NET.

*C#* - это объектно-ориентированный язык программирования, разработанный компанией Microsoft. Он был создан в 2000 году как часть платформы .NET Framework и быстро стал одним из самых популярных языков программирования в мире.

*Платформа .NET* - это программная платформа, разработанная компанией Microsoft для создания и выполнения приложений. Она включает в себя среду выполнения, библиотеки классов и инструменты разработки, которые позволяют разработчикам создавать приложения для различных платформ, включая Windows, Linux и macOS.

C# является одним из основных языков программирования, используемых для разработки приложений на платформе .NET. Он обладает многими преимуществами, такими как:

- Строгая типизация и проверка типов во время компиляции, что позволяет обнаруживать ошибки на ранних этапах разработки.

- Поддержка объектно-ориентированного программирования, что позволяет создавать более гибкие и расширяемые приложения.

- Большое количество библиотек классов, которые позволяют разработчикам использовать готовый код для решения различных задач.

- Поддержка многопоточности, что позволяет создавать приложения, которые могут выполнять несколько задач одновременно.

Платформа .NET также имеет множество преимуществ, таких как:

- Поддержка множества языков программирования, включая C#, F# и Visual Basic.

- Большое количество библиотек классов, которые позволяют разработчикам использовать готовый код для решения различных задач.

- Среда выполнения, которая обеспечивает независимость от аппаратной платформы и операционной системы.

- Инструменты разработки, такие как Visual Studio, которые облегчают процесс разработки приложений на платформе .NET.

# 18. Простейшие операторы языка C#: операторы организации цикла, операторы перехода.

*Операторы организации цикла в языке C#* позволяют повторять выполнение блока кода несколько раз. В языке C# существует три оператора организации цикла:

1. *Оператор for*: позволяет повторять выполнение блока кода заданное количество раз. Синтаксис оператора for выглядит следующим образом:

for (инициализация; условие; итерация){

 // блок кода, который нужно повторять

}

2*. Оператор while:* позволяет повторять выполнение блока кода до тех пор, пока заданное условие истинно. Синтаксис оператора while выглядит следующим образом:

while (условие){

 // блок кода, который нужно повторять

}

3. *Оператор do-while:* позволяет повторять выполнение блока кода до тех пор, пока заданное условие истинно, но гарантирует, что блок кода будет выполнен хотя бы один раз. Синтаксис оператора do-while выглядит следующим образом:

do{

 // блок кода, который нужно повторять

} while (условие);

*Операторы перехода в языке C#* позволяют изменять порядок выполнения программы, переходя к другим участкам кода. В языке C# существует несколько операторов перехода:

1. *Оператор break:* позволяет выйти из цикла или перейти к следующему оператору после цикла. Оператор break может использоваться только внутри цикла. Синтаксис оператора break выглядит следующим образом:

for (int i = 0; i < 10; i++){

 if (i == 5) {

 break;

 }

 Console.WriteLine(i);

}

2. *Оператор continue:* позволяет перейти к следующей итерации цикла, игнорируя оставшуюся часть текущей итерации. Оператор continue может использоваться только внутри цикла. Синтаксис оператора continue выглядит следующим образом:

for (int i = 0; i < 10; i++){

 if (i == 5){

 continue;

 }

 Console.WriteLine(i);

}

3. *Оператор return*: позволяет вернуть значение из метода и завершить его выполнение. Оператор return может использоваться в любом месте метода. Синтаксис оператора return выглядит следующим образом:

public int Sum(int a, int b){

 return a + b;

}

# 19. Простейшие операторы языка C#: оператор присваивания, составной оператор, операторы ветвления.

1. *Оператор присваивания* (=) используется для присвоения значения переменной. Например:

int x = 5; // переменная x получает значение 5

string name = "John"; // переменная name получает значение "John"

2. *Составной оператор* ({}) используется для группировки нескольких операторов в один блок. Например:

if (x > 0) // если x больше 0

{

 Console.WriteLine("x is positive"); // выводим сообщение

 x++; // увеличиваем значение x на 1

}

3. *Операторы ветвления* (if, else if, else) используются для выполнения различных действий в зависимости от условий. Например:

if (x > 0) // если x больше 0

{

 Console.WriteLine("x is positive"); // выводим сообщение

}

else if (x < 0) // если x меньше 0

{

 Console.WriteLine("x is negative"); // выводим сообщение

}

else // если x равен 0

{

 Console.WriteLine("x is zero"); // выводим сообщение

}

# 20. Простейшие действия с данными: оператор и операция. Операции ввода, вывода, преобразование типов, операторы отношений, логические операторы.

1. *Оператор* - это символ или комбинация символов, которые выполняют определенное действие. Например, оператор присваивания (=) используется для присваивания значения переменной.

2. *Операция* - это выполнение действия с помощью оператора. Например, операция присваивания (x = 5) присваивает переменной x значение 5.

3. *Операции ввода и вывода* используются для взаимодействия с пользователем и файлами. Например, операция Console.WriteLine() используется для вывода сообщения на консоль, а операция Console.ReadLine() используется для чтения введенных пользователем данных.

4*. Операции преобразования типов* используются для преобразования значений из одного типа в другой. Например, операция Convert.ToInt32() используется для преобразования строки в целое число.

5. *Операторы отношений* используются для сравнения значений. Например, операторы == (равно), != (не равно), > (больше), < (меньше), >= (больше или равно) и <= (меньше или равно) используются для сравнения числовых значений.

6. *Логические операторы* используются для выполнения логических операций. Например, операторы && (логическое И), || (логическое ИЛИ) и ! (логическое НЕ) используются для выполнения логических операций над значениями типа bool.

# 21. Структура программы на языке C#. Приложение (решение), проект, пространство имен, класс, методы, данные.

Структура программы на языке C# включает в себя несколько уровней организации кода, которые являются основными элементами языка. Рассмотрим каждый из них подробнее:

1. *Приложение (решение)* - это набор проектов, которые связаны между собой и образуют единую программу. В Visual Studio, приложение представляет собой решение (solution), которое может содержать один или несколько проектов.

2*. Проект* - это набор файлов, которые содержат код, ресурсы и другие элементы, необходимые для создания конечного приложения. Каждый проект имеет свой собственный набор настроек, включая настройки компиляции, ссылки на другие библиотеки и т.д.

3*. Пространство имен* - это механизм, который позволяет организовать код в логически связанные группы. Пространства имен используются для предотвращения конфликтов имен и облегчения организации кода. Например, пространство имен System содержит классы и методы, связанные с системными операциями.

4. *Класс* - это шаблон, который определяет состояние и поведение объектов. Классы содержат поля (данные) и методы (операции), которые могут быть вызваны для выполнения определенных действий. Например, класс Console содержит методы для ввода и вывода данных на консоль.

5. *Методы* - это операции, которые могут быть вызваны для выполнения определенных действий. Методы могут принимать параметры и возвращать значения. Например, метод Console.WriteLine() используется для вывода данных на консоль.

6. *Данные* - это переменные, которые содержат значения, используемые в программе. Данные могут быть определены внутри класса или метода и могут иметь различные типы, такие как целые числа, строки, булевы значения и т.д.

# 22. Массивы, строки и списки в С#.

*Массивы* - это упорядоченные коллекции элементов одного типа. Они могут быть одномерными, многомерными или зубчатыми. Для создания массива в C# используется ключевое слово "new", за которым следует тип элементов массива и его размерность. Например, следующий код создает одномерный массив целых чисел:

int[] numbers = new int[5];

*Строки* - это последовательности символов Unicode. Они могут быть созданы с помощью литералов строк или с помощью конструктора класса String. Например, следующий код создает строку с текстом "Hello, world!":

string message = "Hello, world!";

*Списки* - это динамические коллекции элементов, которые могут изменять свой размер во время выполнения программы. В C# есть несколько типов списков, таких как List<T>, LinkedList<T> и другие. Например, следующий код создает список целых чисел и добавляет в него несколько элементов:

List<int> numbers = new List<int>();

numbers.Add(1);

numbers.Add(2);

numbers.Add(3);

Списки обычно используются вместо массивов, когда необходимо добавлять или удалять элементы в коллекции во время выполнения программы.

Для работы с массивами, строками и списками в C# есть множество методов и свойств, которые позволяют выполнять различные операции, такие как добавление, удаление, поиск, сортировка и т.д.

# 23. Обработка исключений в C#. Класс Exception

В C# обработка исключений осуществляется с помощью конструкции try-catch-finally. Код, который может вызвать исключение, помещается в блок try. Если исключение произошло, то управление передается в блок catch, где можно обработать исключение. Блок finally выполняется в любом случае, независимо от того, произошло исключение или нет.

Класс Exception является базовым классом для всех исключений в C#. Он содержит свойства и методы, которые позволяют получить информацию об исключении, такую как тип исключения, сообщение об ошибке, стек вызовов и т.д. Кроме того, класс Exception позволяет создавать пользовательские исключения, наследуясь от него.

Пример использования класса Exception:

try{

 // Код, который может вызвать исключение

}

catch (Exception ex){

 // Обработка исключения

 Console.WriteLine("Исключение: " + ex.Message);

}

finally{

 // Код, который должен выполниться в любом случае

}

# 24. Платформа WPF, её особенности.

*Windows Presentation Foundation (WPF*) - это платформа для создания графических пользовательских интерфейсов (GUI) в приложениях Windows. Она была введена в .NET Framework 3.0 и является одной из основных технологий для создания приложений Windows в .NET.

Основные особенности WPF:

1. *Визуальные элементы*: WPF предоставляет богатый набор визуальных элементов, таких как кнопки, текстовые поля, изображения, таблицы и т.д. Эти элементы могут быть настроены с помощью стилей и шаблонов.

2. *Разметка XAML*: WPF использует язык разметки XAML для определения пользовательского интерфейса. XAML позволяет разработчикам создавать интерфейсы, используя декларативный подход, что упрощает разработку и поддержку кода.

3. *Графические возможности*: WPF предоставляет мощные графические возможности, такие как векторная графика, анимация, эффекты и т.д. Это позволяет создавать более интерактивные и привлекательные пользовательские интерфейсы.

4. *Привязка данных*: WPF предоставляет механизм привязки данных, который позволяет связывать данные с элементами пользовательского интерфейса. Это упрощает работу с данными и уменьшает количество кода, необходимого для обновления пользовательского интерфейса.

5. *Поддержка стилей и шаблонов*: WPF позволяет создавать стили и шаблоны, которые могут быть применены к различным элементам пользовательского интерфейса. Это упрощает создание единообразного пользовательского интерфейса и уменьшает количество кода, необходимого для его создания.

6. *Поддержка многопоточности*: WPF поддерживает многопоточность, что позволяет создавать более отзывчивые приложения и улучшать производительность.

7. *Поддержка аппаратного ускорения*: WPF использует аппаратное ускорение для отображения графики, что улучшает производительность и качество отображения.

8. *Поддержка масштабирования*: WPF позволяет создавать приложения, которые могут масштабироваться на различных устройствах и разрешениях экрана.

В целом, WPF предоставляет мощные инструменты для создания современных и привлекательных пользовательских интерфейсов в приложениях Windows. Она является одной из основных технологий для создания приложений на платформе .NET и позволяет разработчикам создавать более отзывчивые и привлекательные приложения.

# 25. Язык разметки XAML. Структура и основные элементы.

*Язык разметки XAML (eXtensible Application Markup Language)* используется для определения пользовательского интерфейса в приложениях Windows Presentation Foundation (WPF) и других технологиях Microsoft. Он позволяет разработчикам создавать графические элементы интерфейса, определять их свойства и расположение на экране.

Структура XAML-документа состоит из корневого элемента, который может содержать другие элементы вложенные в него. Каждый элемент имеет имя и может иметь атрибуты, которые определяют его свойства. Например, элемент Button имеет атрибуты Content, Width, Height, Background и т.д.

Основные элементы XAML включают в себя:

1. *Window* - элемент, который представляет окно приложения.

2. *Grid* - элемент, который представляет сетку, в которой можно размещать другие элементы.

3. *StackPanel* - элемент, который представляет панель, в которой элементы располагаются друг за другом в заданном порядке.

4. *TextBlock* - элемент, который представляет текстовую строку.

5. *Button* - элемент, который представляет кнопку.

6. *TextBox* - элемент, который представляет текстовое поле для ввода данных.

7. *ComboBox* - элемент, который представляет выпадающий список.

8. *ListBox* - элемент, который представляет список элементов.

Пример XAML-кода для создания кнопки:

<Button Content="Нажми меня" Width="100" Height="50" Background="Blue" />

Этот код создаст кнопку с текстом "Нажми меня", шириной 100 пикселей, высотой 50 пикселей и синим фоном.