**1. Что такое методология программирования? Какие есть методологии?**

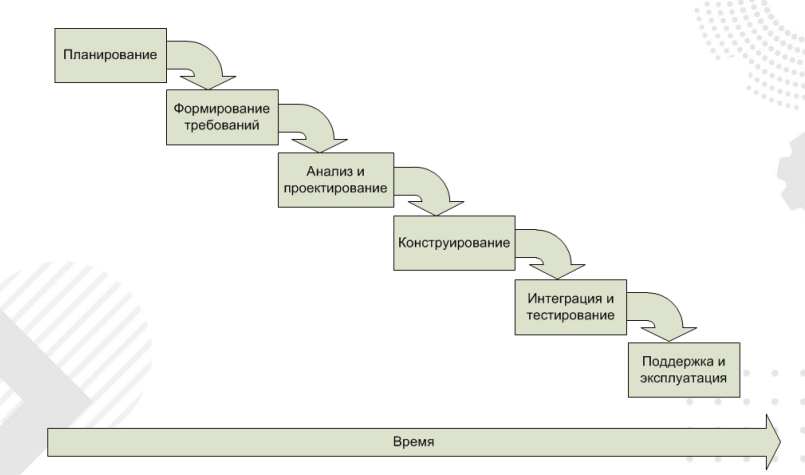
*Методология программирования* – это совокупность идей, понятий, принципов, способов и средств, определяющая стиль написания, отладки и сопровождения программ.

Виды методологий программирования:

* Водопадная модель
* Итеративная модель
* Спиральная модель
* Lean
* XP (Extreme Programming)
* FDD (Feature-Driven Development)
* TDD (Test Driven Development)
* BDD (Behavior Driven Development)
* Agile
* Scrum
* Kanban

**2. Как выполняется разработка по водопадной модели? Какие у нее преимущества и недостатки?**

*Водопадная модель разработки* – это линейный подход к разработке программного обеспечения.



1. Сначала полностью завершается этап «сбор требований»
2. Переход к проектированию, в ходе которого создаются документы, подробно описывающие для программистов способ и план реализации указанных требований
3. Реализация полученного проекта
4. Интеграция отдельных компонентов, разрабатываемых различными командами программистов
5. Тестирование и отладка продукта
6. Программный продукт внедряется и обеспечивается его поддержка – внесение новой функциональности и устранение ошибок

*Преимущества:*

* Эта методология хорошо подходит для проектов со строгими и понятными требованиями
* Применима к проектам любого масштаба – от небольших до крупных
* Много внимания уделяется тщательному документированию каждого этапа разработки
* Модель простая, четкая и понятная. Её легко внедрить в рабочий процесс благодаря простым фазам, которые идут строго по порядку.

*Недостатки*

* Недостаточно гибкости. После завершения какого-либо этапа сложно вернуться назад и внести изменения, не нарушая последующие этапы
* Может быть проблематично применять в проектах, где нужны итерации и частая обратная связь от владельца продукта
* Фаза тестирования наступает в конце проекта, поэтому проблемы и ошибки могут быть выявлены не сразу – а устранять их будет дорого и сложно.

**3. Как выполняется разработка по итерационной модели? Какие у нее преимущества и недостатки?**

*Итеративная модель* – это процесс создания программного обеспечения, который осуществляется небольшими этапами, в ходе которых ведется анализ полученных промежуточных результатов, выдвигаются новые требования и корректируются предыдущие этапы работы.

При итерационной разработке проект разбит на последовательность итераций, каждая из которых является проектом в миниатюре (то есть включает в себя все процессы разработки ПО (сбор и анализ требований, составление спецификаций, непосредственную реализацию, тестирование и запуск), но в рамках одной итерации разрабатывается не весь проект, а только его версия или отдельная часть).

*Преимущества*

* Снижение рисков – раннее обнаружение конфликтов между требованиями, моделями и реализацией проекта; большая фокусировка на основных задачах
* Динамическое формирование требований и управление ими
* Организация эффективной обратной связи проектной команды с потребителем, создание продукта, реально отвечающего его потребностям
* Быстрый выпуск минимально ценного продукта (MVP) и возможность вывести продукт на рынок и начать эксплуатацию гораздо раньше.

*Недостатки*

* Проблемы с архитектурой и накладные расходы – при работе с хаотичными требованиями и без проработанного глобального плана архитектура приложения может пострадать, а на её приведение к адекватному виду могут потребоваться дополнительные ресурсы
* Нет фиксированного бюджета и сроков, а также нужна сильная вовлеченность Заказчика в процесс.

**4. Как выполняется разработка по спиральной модели? Что такое риск, какие существуют риски?**

Спиральная модель – это модель процесса разработки программного обеспечения с учетом рисков. Это комбинация водопадной и итеративной моделей.

Спиральная модель состоит из четырех главных повторяющихся стадий. В ходе процесса разработки проект несколько раз проходит через все эти фазы. Каждая такая итерация называется спиралью.

*Фазы спиральной модели:*

1. Определение целей, альтернатив, ограничений, или фаза планирования
2. Анализ, определение и разрешение рисков
3. Фаза разработки
4. Планирование следующей фазы

Отличительной особенностью этой модели является специальное внимание рискам, влияющим на организацию жизненного цикла. Выделяют десять наиболее распространённых (по приоритетам) рисков:

1. Дефицит специалистов
2. Нереалистичные сроки и бюджет
3. Реализация несоответствующей функциональности
4. Разработка неправильного пользовательского интерфейса
5. «Золотая сервировка», перфекционизм, ненужная оптимизация и оттачивание деталей
6. Непрекращающийся поток изменений
7. Нехватка информации о внешних компонентах, определяющих окружение системы или вовлечённых в интеграцию
8. Недостатки в работах, выполняемых внешними (по отношению к проекту) ресурсами
9. Недостаточная производительность получаемой системы
10. Разрыв между квалификацией специалистов и требованиями проекта.

**5. Чем подходы FDD, TDD и BDD отличается от других подходов разработки?**

Подходы **FDD (Feature-Driven Development)**, **TDD (Test-Driven Development)** и **BDD (Behavior-Driven Development)** относятся к гибким (agile) методологиям разработки.

Эти три подхода (FDD, TDD, BDD) фокусируются на конкретных аспектах разработки, а не на управлении проектом в целом (как Scrum или Waterfall). Их главная цель — структурировать процесс написания кода и улучшить его качество.

1. FDD — Feature-Driven Development (разработка, основанная на функциональности)

Основные черты:

* Фокус на разработке конкретных функций, ценных для клиента.
* Работа организуется вокруг малых, измеримых задач — "features".
* Используются модели домена, планирование по функциям, дизайн и построение каждой функции поэтапно.

Отличие от других подходов:

* Строгая структура и пошаговая реализация каждой фичи.
* Подходит для больших проектов с постоянной командой.
* Меньше акцента на тестах и поведении, больше на планировании и архитектуре

2. TDD — Test-Driven Development (разработка через тестирование)

Основные черты:

* Сначала пишется тест, который не проходит.
* Затем — код, который проходит тест.
* После — рефакторинг без изменения поведения.

Отличие от других подходов:

* Код пишется "от теста", а не "от задачи".
* Позволяет создать высокопокрытый, тестируемый и модульный код.
* Чаще всего используется в backend-разработке и для библиотек/компонентов.

3. BDD — Behavior-Driven Development (разработка, ориентированная на поведение)

Основные черты:

* Расширяет TDD, добавляя описание поведения системы на естественном языке (обычно Gherkin: Given-When-Then).
* Вовлекает бизнес-аналитиков, тестировщиков и разработчиков.
* Позволяет выразить требования в форме поведенческих сценариев.

Отличие от других подходов:

* Улучшает взаимопонимание между техническими и нетехническими участниками.
* Фокус на ожидаемом поведении системы, а не только на технических аспектах.
* Часто используется в автоматизированном acceptance-тестировании.

**6. В чем заключается методология Agile? Что включает в себя Манифест Agile?**

*Методология Agile* – это подход к управлению проектами, предполагающий разбивку проекта на этапы, а также непрерывное сотрудничество и совершенствование. В рамках этого подхода команды циклично проводят планирование, выполнение и оценку.

*Манифест Agile включает 4 ценности и 12 принципов:*

*Принципы:*

1. Удовлетворение клиентов – приоритетная задача при разработке продукта. Клиенты должны своевременно и в полном объёме получать качественное программное обеспечение и его обновления.
2. Изменения в процессе разработки приветствуются. Гибкие процессы позволяют наделить продукт конкурентными преимуществами для клиентов
3. Рабочее ПО нужно доставлять клиенту часто, в рамках 2–16 недель
4. Руководители и разработчики должны трудиться вместе на протяжении всего рабочего процесса
5. В основе проекта – мотивированные люди. Обеспечьте им необходимые условия работы, поддержку и доверие
6. Лучший способ передачи информации в команде – личная беседа
7. Основное мерило прогресса – работающее ПО. А не часы, трудозатраты и другие критерии
8. Гибкие процессы – основа устойчивого развития. Они позволяют поддерживать нужный рабочий темп как на спринтерской, так и на марафонской дистанции
9. Важно уделять внимание техническому совершенству и качественному дизайну продукта
10. Важно сокращать до минимума лишнюю работу и не переусложнять проект и рабочие процессы
11. Самые лучшие продукты рождаются у самоорганизующихся команд. Нет микроменеджменту, да – свободе управления
12. Команда должна регулярно оценивать работу и корректировать своё поведение.

*Ценности*

1. Люди и взаимодействия важнее процессов и инструментов
2. Работающий продукт важнее точной и подробной документации
3. Сотрудничество с заказчиком важнее условий договора
4. Готовность к изменениям важнее следования изначальному плану

**7. В чем заключается методология Scrum? Что такое спринт, дейлик, ретроспектива?**

*Scrum* – методика организации совместного рабочего процесса, в основе которой – поэтапная разработка и совершенствование продукта небольшой командой специалистов различного профиля. Scrum относится к семейству гибких Agile-методик.

*Спринт* – период, в течение которого команда создает и совершенствует отдельную часть продукта, чтобы быть готовым к изменениям условий и исключить глобальную доработку.

*Дейлик* – Короткая (до 15 минут) ежедневная встреча команды. Цель встречи – услышать от каждого участника ответ на три вопроса: Что я сделал с прошлой встречи? Что я буду делать сегодня? Что мешает выполнению задачи?

*Ретроспектива* — это мероприятие, направленное на улучшение командных процессов за счет обсуждение предыдущих событий и процессов в команде, которые наблюдались в течение спринта. Это мероприятие, которое завершает спринт.

**8. В чем заключается методология Kanban? Чем она отличается от Scrum?**

*Kanban* – это популярный подход к реализации принципов Agile и DevOps при разработке ПО, предполагающий обсуждение ресурсов в реальном времени и полную прозрачность процессов.

Рабочие задачи выводятся на доску Kanban, благодаря чему участники команд могут просматривать статус каждой из них в любой момент времени.

Отличия Kanban и Scrum:

* Scrum – структурированный подход с заданными этапами создания продукта, а Kanban – сбалансированный, основная цель которого – обеспечить всех членов команды одинаковым количеством работы.
* В Scrum предусмотрены организованные периоды работы с задачами на период. В Kanban новые задачи могут появляться в любой день.
* Scrum-команды работают в течение заданного отрезка времени, а в Kanban задачи поступают непрерывно.
* Kanban гораздо мягче, чем Scrum. Он позволяет начать с того, что есть сейчас: взять принципы, уже присутствующие в компании, и постепенно их улучшать.

**9. Что включает в себя инструментарий программирования?**

*Инструменты для программирования* – это программные средства, которые помогают разработчикам создавать, тестировать и поддерживать программное обеспечение (примеры: Jira, Trello, Asana).

*Основные типы инструментов включают:*

* Интегрированные среды разработки (IDE)
* Системы контроля версий
* Инструменты для тестирования и отладки
* Инструменты для управления проектами
* Инструменты для автоматизации сборки
* Инструменты для анализа и мониторинга кода

**10.Что такое CI/CD? Что такое CD-конвейер?**

CI/CD (Continuous Integration, Continuous Delivery – непрерывная интеграция и доставка) – это технология автоматизации тестирования и доставки новых модулей разрабатываемого проекта заинтересованным сторонам (практика разработки программного обеспечения, обеспечиваемая автоматизацией).

* CI (Continuous Integration) — непрерывная интеграция. Это процесс, при котором изменения кода регулярно (несколько раз в день) интегрируются в общую кодовую базу и автоматически тестируются.
* CD (Continuous Delivery) — непрерывная доставка. Это автоматическая подготовка проверенного кода к релизу в любое время.

Типичный CD-конвейер состоит из этапов сборки, тестирования и развертывания.

(гпт)

*CD-конвейер* — это последовательность автоматических этапов, через которые проходит код после коммита: от сборки и тестов до выката на сервер (развёртывания). Он включает в себя шаги вроде:

* Сборка (build)
* Юнит-тесты
* Интеграционные тесты
* Статический анализ кода
* Развёртывание (deployment)

**11.Зачем нужны архитектурные паттерны? Какие знаете архитектурные паттерны?**

Архитектурные паттерны определяют архитектуру приложения, задают его логику и помогают разработчику понять, как устроен продукт внутри.

*Архитектурные паттерны:*

* Многоуровневый шаблон
* Модель-представление-контроллер
* Клиент-серверный шаблон
* Каналы и фильтры
* Управляемая событиями архитектура
* Архитектура на основе микросервисов
* CQRS

**12.Какие особенности у многоуровневая архитектуры?**

* Паттерн позволяет создать понятное и хорошо задокументированное разделение задач, чтобы модули системы можно было разрабатывать и поддерживать независимым образом.
* Разрабатываемое программное обеспечение разделяется так, чтобы модули можно было разрабатывать и развивать отдельно – с минимальным взаимодействием между частями, обеспечивая переносимость, модифицируемость и повторное использование.
* Уровни (состоящие из модулей, предоставляющих взаимосвязанный набор сервисов) полностью разделяют ПО, причем каждая часть доступна через публичный интерфейс.

Уровни:

* Представление (также известен как пользовательский интерфейс)
* Бизнес-логика (также известен как уровень предметной области)
* Доступ к данным (также известен как уровень хранения данных)
* База данных

**13.Какие особенности у MVC архитектуры?**

Паттерн MVC предоставляет простой способ построения структуры приложения через разделение бизнес-логики и пользовательского интерфейса.

1. Модель – содержит бизнес-логику приложения
2. Представление (вид) – используется для отображения данных пользователю от модели
3. Контроллер является связующем звеном между видом (видами), моделью (моделями), и другими компонентами приложения и отвечает за обработку запросов пользователя

**14.Какие особенности у клиент-серверной архитектуры?**

* Данный шаблон представляет проект в виде двух приложений: сервера и клиента. Сервер отвечает за хранение и обработку данных. Клиент обеспечивает взаимодействие пользователя с сервером.
* Несколько компьютеров-клиентов (удалённые системы) посылают запросы и получают услуги от централизованной служебной машины – сервера, которая также может называться хост-системой.

**15.Какие типы паттернов проектирования знаете? Какие паттерны каждого типа можете назвать?**

*Паттерны проектирования* – это решения часто встречающихся проблем в области разработки ПО.

Паттерны не являются готовыми решениями, которые можно преобразовать в код, а представляют общее описание решения проблемы, которое можно использовать в различных ситуациях.

*Типы паттернов:*

1. *Структурные паттерны* (предназначены для выстраивания правильной иерархии классов (упрощение создания новых классов, расширение существующих) и правильного взаимодействия между классами (уменьшение взаимосвязанности и т.п.))
2. *Порождающие паттерны* (создания объектов, позволяя системе оставаться независимой как от самого процесса порождения, так и от типов порождаемых объектов.)
3. *Поведенческие паттерны* (рассматривают вопросы о связях между объектами и распределением обязанностей между ними)

*Структурные паттерны:*

* Паттерн Adapter (позволяет объектам с несовместимыми интерфейсами работать вместе)
* Паттерн Bridge
* Паттерн Composite
* Паттерн Decorator
* Паттерн Facade
* Паттерн Flyweight
* Паттерн Proxy

*Порождающие паттерны:*

* Паттерн Factory Method
* Паттерн Abstract Factory
* Паттерн Builder
* Паттерн Prototype
* Паттерн Singleton (контролирует создание единственного экземпляра некоторого класса и предоставляет доступ к нему. Возлагает контроль над созданием единственного объекта на сам класс. Доступ к этому объекту осуществляется через статическую функцию-член класса, которая возвращает указатель или ссылку на него. Этот объект будет создан только при первом обращении к методу, а все последующие вызовы просто возвращают его адрес)
* Паттерн Object Pool

*Паттерны поведения:*

* Паттерн Chain of Responsibility (позволяет передавать запросы последовательно по цепочке обработчиков. Каждый последующий обработчик решает, может ли он обработать запрос сам и стоит ли передавать запрос дальше по цепи)
* Паттерн Command
* Паттерн Iterator
* Паттерн Interpreter
* Паттерн Mediator
* Паттерн Memento
* Паттерн Observer
* Паттерн State
* Паттерн Strategy
* Паттерн Template Method
* Паттерн Visitor

**16.Сериализация. Что такое сериализация? Что такое десериализация? Область применения?**

*Сериализация* – это процесс преобразования объекта в поток байтов для сохранения или передачи в память, базу данных или файл. Объект сериализуется в поток, который служит для передачи данных. Поток также может содержать сведения о типе объекта

*Десериализация* – процесс, обратный процессу сериализации данных, т.е. восстановление состояния объекта с целью последующего использования. Последовательность сохраняемых данных должна содержать в себе всю информацию, необходимую для восстановления.

*Посредством сериализации можно выполнять следующие действия:*

* отправка объекта в удаленное приложение с помощью веб-службы
* передача объекта из одного домена в другой
* передача объекта через брандмауэр в виде строки (JSON или XML, например)
* хранение сведений о безопасности и пользователях между приложениями

**17.Подходы сериализации. Какие есть подходы сериализации?**

Подходы

* Базовый (упрощенный)
* Пользовательский (настраиваемый)

Базовый: допускаются настройки для класса, указывающие, что он может быть сериализован и настройки для полей, что в процессе сериализации они не участвуют и информацию с них не сохранять. В процессе сериализации сохраняется информация со всех (либо только публичных) полей класс (исключение составляют поля, которые были явно указаны как несериализуемые), в зависимости от настроек выбранного типа сериализации. Настройки сериализации класса могут не распространятся на производные классы от этого класса. В случае, если в классе имеются поля-ссылки на объекты других классов, то эти поля не должны участвовать в процессе сериализации, если у других классов не настроена сериализация.

Пользовательский подход: подразумевает применение различных манипуляций с данными объекта класса перед сериализацией данных, в процессе сериализации, и в процессе или по окончании процесса десериализации данных. Под манипуляцией подразумевают преобразование значений (например, шифрование значений) для сокрытия важной информации или структуры класса, сохранения сведений о событии сериализации (например, даты и времени сериализации) или для версионности данных (например, если поменялась структура класса и надо поддерживать данные, сохраненные по старой структуре). Для реализации пользовательского подхода в классе потребуется определить 2 метода, первый должен вызываться при сериализации объекта, а второй при десериализации.

**18.Типы сериализации. Какие есть типы сериализации?**

Типы сериализаций:

* + Бинарная (двоичная) сериализация – генерирует компактный поток байтов.
* Не скрывает структуру класса
* Сохраняет даже приватные поля (нарушение инкапсуляции)
* Сохраняет информацию об имени класса (могут быть проблемы при десериализации получаемого объекта другим классом с тем же набором полей и даже тем же именем)
  + SOAP (Simple Object Access Protocol) – это облегченный протокол для обмена информацией в децентрализованной распределенной среде. Это протокол на основе XML. Состоит из 3 частей:
* Конструкция конверта SOAP. Определяет общую структуру для выражения содержимого сообщения; кто должен заниматься этим, и является ли это необязательным или обязательным.
* Правила кодирования SOAP. Определяют механизм сериализации, который можно использовать для обмена экземплярами типов данных, определяемых приложением
* Представление SOAP RPC. Определяет соглашение, которое можно использовать для представления удаленных вызовов процедур и ответов.
* XML
* JSON

**19.XML. Что такое XML? Каковы особенности записи?**

XML (eXtensible Markup Language) – расширяемый язык разметки, используемый для хранения и передачи данных. Язык называется расширяемым, поскольку он не фиксирует разметку, используемую в документах: разработчик волен создать разметку в соответствии с потребностями к конкретной области, будучи ограниченным лишь синтаксическими правилами языка.

Все элементы XML оформлены в виде тегов. Между открывающим и закрывающим тегами хранится значение, которое привязано к этому тегу, либо вложенные элементы. Также у элемента могут быть атрибуты. Они указываются внутри отрывающегося тега после названия тега через пробел и в формате «имя атрибута = значение атрибута».

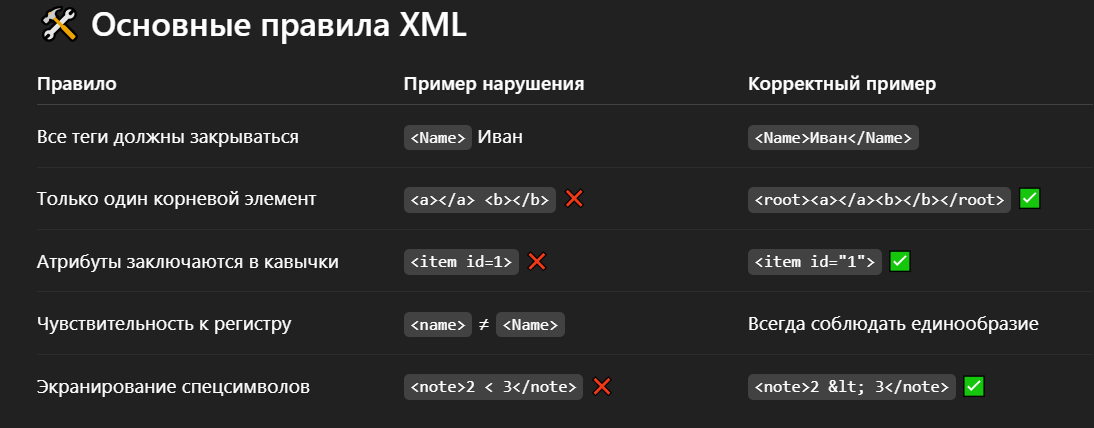
Состоит из:

* Пролог
* Включает объявления, инструкции обработки, комментарии
* Является не обязательной частью документа, но если он есть, то это должна быть первая строка XML документа.
* Показывает версию XML
* Показывает кодировку
* Корневой элемент
* Обязательная часть документа, составляющая всю его суть
* Может включать вложенные в него элементы, символьные данные и комментарии

*XSD* – это язык описания структуры XML документа. При использовании такой схемы XML парсер может проверить не только правильность синтаксиса XML документа, но также его структуру, модель содержания и типы данных.

*Атрибуты* – чтобы класс корректно сериализовался в xml-объект, к нему самому и его полям можно применять специальные атрибуты. Атрибуты можно использовать для управления сериализацией XML объекта или для создания альтернативного потока XML из того же набора классов.

* XmlRootAttribute
* XmlElementAttribute
* XmlAttributeAttribute
* XmlArrayAttribute
* XmlArrayItemAttribute
* XmlEnumAttribute
* XmlIncludeAttribute
* XmlIgnoreAttribute



**20.JSON. Что такое JSON? Каковы особенности записи?**

*JavaScript Object Notation* – текстовый формат обмена данными, основанный на JavaScript. Формат считается независимым от языка и может использоваться практически с любым языком программирования.

*В качестве значений в JSON могут быть использованы:*

* JSON-объект
* Массив
* Число (целое или вещественное)
* Литералы
* Строка

*JSON-объект* – это неупорядоченное множество пар «ключ:значение». Ключ – это название параметра объекта. Имя ключа заключают в кавычки. В качестве значение может выступать любой элемент JSON. Если в объекте более одной пары, то пары разделяются запятой. Объект заключается в фигурные скобки ’{’ ’}’.

{

“ParamName”: Value,

“SecondParam”: Value

}

*Массив (одномерный)* – это упорядоченное множество значений. Массив может быть пустым. Значения в пределах одного массива могут иметь разный тип. В массиве нет ключей, как в объекте, поэтому обращаться к массиву можно только по номеру элемента. И поэтому менять местами данные внутри нельзя.

*Значения*:

* *Число*. Число тоже очень похоже на Javascript-число, за исключением того, что используется только десятичный формат (с точкой в качестве разделителя)
* *Литералы*. true (логическое значение «истина»), false (логическое значение «ложь») и null
* *Строка*. Строка – это упорядоченное множество из нуля или более символов юникода, заключённое в двойные кавычки. Символы могут быть указаны с использованием escape-последовательностей, начинающихся с обратной косой черты «\» (поддерживаются варианты \", \\, \/, \t, \n, \r, \f и \b), или записаны шестнадцатеричным кодом в кодировке Unicode в виде \uFFFF.

JSON5 (расширение JSON) Некоторые нововведения

* Поддерживаются как однострочные //, так и многострочные /\* \*/ комментарии
* Записи и списки могут иметь запятую после последнего элемента (удобно при копировании элементов)
* Ключи записей могут быть без кавычек, если они являются валидными идентификаторами ECMAScript 5
* Строки могут заключаться как в одинарные, так и в двойные кавычки
* Числа могут быть в шестнадцатеричном виде, начинаться или заканчиваться десятичной точкой, включать Infinity, -Infinity, NaN и - NaN, начинаться со знака +.

Маршалинг – процесс преобразования информации (данных, двоичного представления объекта), хранящейся в оперативной памяти, в формат, пригодный для хранения или передачи. Маршалинг подобен сериализации, за исключением того, что при маршалинге указывается расположение исполняемого кода. Маршалинг отличается от сериализации тем, что он особым образом применяется к удалённым объектам, для которых не создается копия самого объекта, а лишь указывается удалённый адрес, по которому должна производиться работа с ним.

**21.Потоки. Что такое поток? Что такое многопоточность?**

*Поток выполнения* – наименьшая единица обработки, исполнение которой может быть назначено ядром операционной системы. Каждый поток имеет приоритет планирования и набор структур, в которых система сохраняет контекст потока, когда выполнение потока приостановлено. Контекст потока содержит все сведения, позволяющие потоку безболезненно возобновить выполнение

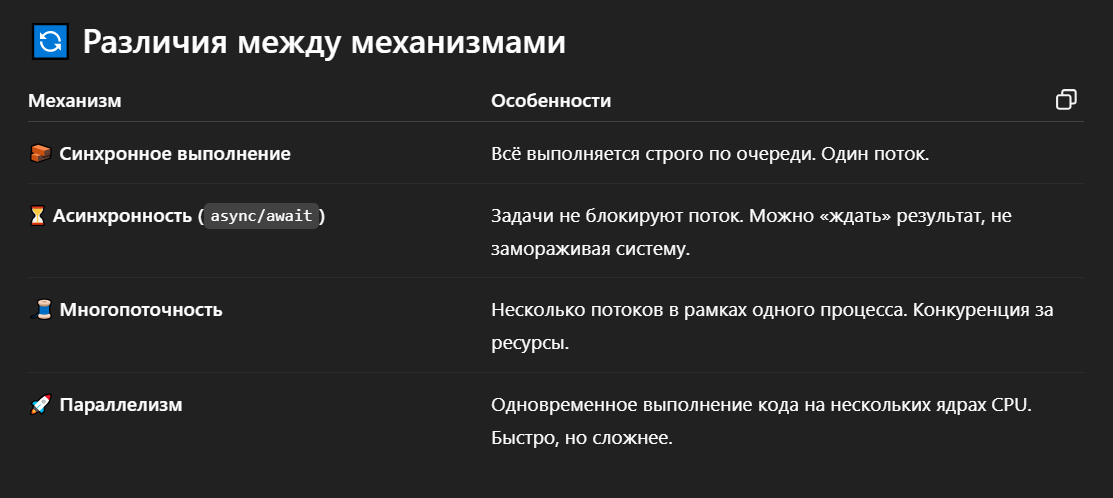
*Многопоточность* – на одном процессоре многопоточность обычно происходит путём временного мультиплексирования (как и в случае многозадачности): процессор переключается между разными потоками выполнения. Это переключение контекста обычно происходит достаточно часто, чтобы пользователь воспринимал выполнение потоков или задач как одновременное.

**22.Параллельное программирование. Чем отличается от других механизмов выполнения методов?**

*Параллельное программирование* – механизм обработки данных, когда большой объем данных бьется на небольшие куски, каждый из которых обрабатывается отдельно, но при этом все эти обработки идут одновременно.

*Место применения*

* Обработка больших файлов
* Сложные вычисления
* Вычисления по большому объему данных

****

**23.Параллельное программирование. Какие подходы есть?**

*Параллелизм данных:*

* Одна операция применяется сразу к нескольким элементам массива данных. Различные фрагменты такого массива обрабатываются на векторном процессоре или на разных процессорах (ядрах) параллельной вычислительной системы
* Обработкой данных управляет одна программа
* Пространство имен является глобальным
* Параллельные операции над элементами массива выполняются одновременно на всех доступных данной программе процессорах.

*Параллелизм задач:*

* Вычислительная задача разбивается на несколько относительно самостоятельных подзадач. Каждая подзадача выполняется на своем процессоре (ориентация на архитектуру MIMD)
* Для каждой подзадачи пишется своя собственная программа на обычном языке программирования
* Подзадачи должны обмениваться результатами своей работы, получать исходные данные. Обмен осуществляется вызовом процедур специализированной библиотеки. Программист может контролировать распределение данных между различными процессорами и различными подзадачами, а также обмен данными.

**24.Зачем нужно синхронизировать потоки? Какие инструменты синхронизации поток есть?**

Синхронизация позволяет избежать ошибок согласованности памяти, вызванные из-за непоследовательного доступа к общей памяти: чаще всего при работе разные потоки работают с одним ресурсом, это может быть файл с данными, список данных и т.п. Либо могут использовать данные для обмена между собой. В таких случаях возникают проблемы правильной организации взаимодействия потоков так, чтобы они не мешали работе друг друга. Один поток должен знать об изменениях, внесенных в ресурс другим потоком.

*Механизмы синхронизации потоков:*

* Lock (определяет блок кода, внутри которого весь код блокируется и становится недоступным для других потоков до завершения работы текущего потока)
* Monitor (пока поток владеет блокировкой объекта, другой поток не может получить эту блокировку.)
* AutoResetEvent (Поток ждёт, пока не получит сигнал от другого потока. После получения сигнала автоматически сбрасывается в неустановленное состояние, чтобы другой поток не прошёл случайно)
* Mutex (как и Monitor, предоставляет монопольный доступ к общему ресурсу. Аналогично Monitor, Mutex реализует привязку потока, и поток, который получил мьютекс, должен освободить его по завершению работы с блокирующим кодом. В отличие от Monitor, класс Mutex может использоваться для межпроцессной синхронизации)
* Семафоры (Разрешает ограниченное количество потоков заходить в критическую секцию одновременно)

**25.Какие есть потокобезопасные коллекции?**

*Потокобезопасные коллекции* предназначены для работы с коллекциями в многопоточном приложении. Несколько потоков могут безопасно и эффективно добавлять и удалять элементы из таких коллекций, не требуя при этом дополнительной синхронизации в пользовательском коде.

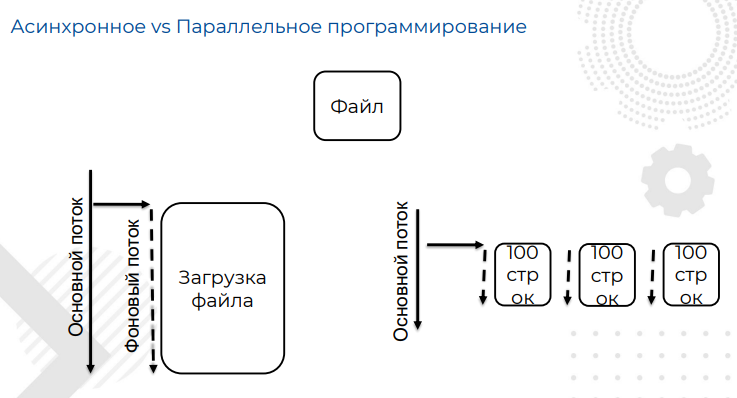
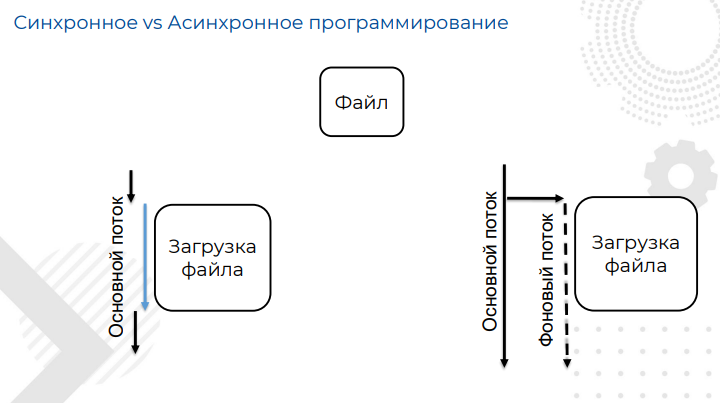
* *ConcurrentBag* – потокобезопасна неупорядоченная коллекция объектов
* Является потокобезопасной реализацией контейнера, оптимизированной для сценариев, в которых один и тот же поток будет создавать и потреблять данные, хранящиеся в контейнере
* TryPeek(T) – пытается вернуть объект из ConcurrentBag без его удаления
* TryTake(T) – пытается удалить и вернуть объект из коллекции ConcurrentBag
* *ConcurrentQueue* – предоставляет потокобезопасную коллекцию, обслуживаемую по принципу «первым поступил – первым обслужен» (FIFO)
* TryDequeue(T) – пытается удалить и вернуть объект, находящийся в начале параллельной очереди
* TryPeek(T) – пытается вернуть объект из начала ConcurrentQueue без его удаления
* *ConcurrentStack* – предоставляет потокобезопасную коллекцию, обслуживаемую по принципу «последним поступил – первым обслужен» (LIFO)
* Push – вставляет элемент в начало стека
* TryPop – удаляет элемент из верхней части стека или возвращает false, если элемент не может быть удален
* TryPeek – возвращает элемент, расположенный в верхней части, стека но не удаляет его
* TryPopRange и PushRange – обеспечивают эффективную принудительную отправку и извлечение нескольких элементов в одной операции.
* *ConcurrentDictionary* – Представляет потокобезопасную коллекцию пар "ключ-значение", доступ к которой могут одновременно получать несколько потоков.
* TryAdd – добавляет указанную пару "ключ-значение", если ключ в настоящее время не существует в словаре
* TryUpdate – проверяет, имеет ли ключ указанное значение, и, если это так, обновляет ключ новым значением
* AddOrUpdate – использует передаваемые методы, чтобы добавить запись, если ключа нет, или чтобы обновить, если ключ есть
* GetOrAdd – добавляет пару "ключ-значение" в коллекцию, если ключ еще не существует. Возвращает новое значение или существующее значение, если ключ существует.

**26.Асинхронное программирование. Чем отличается от других механизмов выполнения методов?**

Механизм выполнения ряда методов в неблокирующем режиме, что позволяет основным методам программы работать, не ожидая завершения работы других методов.

Асинхронная концепция программирования заключается в том, что результат выполнения функции доступен не сразу же, а через некоторое время. Т.е. метод можно вызвать заранее, зная, что выполнятся он будет долго. А пока он выполняется, совершить ряд иных действий.

(в синхронном методы выполняются последовательно)



**27.Что такое конечный автомат, как он работает?**

*Конечный автомат (finite-state machine, FSM)* – это математическая абстракция, модель, которая может находиться только в одном из конечного числа состояний в каждый конкретный момент времени. Автомат умеет переходить из одного состояния в другое в ответ на данные, которые подаются на вход; изменение состояния называется переходом.

*Как работает:* на каждое использование ключевого слова await компилятор генерирует дополнительное состояние для конечного автомата. Конечный автомат переходит через состояния с помощью метода MoveNext(), по сути, – большого переключателя.

(гпт)

Конечный автомат (КА) — это абстрактная модель, у которой есть:

* набор состояний
* начальное состояние
* возможные переходы между состояниями в ответ на события или действия.

Когда пишем async/await в языке вроде C#, компилятор автоматически преобразует эту функцию во внутренний конечный автомат, чтобы:

* приостанавливать выполнение, когда встречает await,
* возобновлять выполнение, когда асинхронная операция завершится.

**28.Как используются async/await?**

* *async* – ключевое слово, модификатор метода. Указывает, что данный метод может выполнятся асинхронно. В методах с таким модификатором позволяется использовать оператор await.
* *await* – унарный оператор. В качестве операнда выступает объект класса Task. Указывает, что во время работы необходимо дождаться окончания выполнения операции, заданной в операнде.
* При вызове метода с async/await он начинает работу как обычный, синхронный метод. Дойдя до первого обращения к оператору await метод переходит в асинхронный режим работы.
* Оператор await нельзя использовать в методах, у которых в сигнатуре отсутствует модификатор async. Однако, модификатор async может присутствовать в сигнатуре метода и при этом в теле метода не будут фигурировать операторы await.
* В качестве возвращаемого типа для асинхронного метода используется один из следующих:
* Task<T> – если асинхронный метод должен вернуть значение, где T – тип возвращаемого значения. ▪
* ValueTask – если асинхронный метод должен вернуть значение, но, в отличии от Task будет возвращаться экземпляр структуры, а не объекта
* ValueTask<T> – если асинхронный метод должен вернуть значение, но, в отличии от Task будет возвращаться экземпляр структуры, а не объекта
* Если метод асинхронен по причине использования в нем другого асинхронного метода (например, при чтении/записи в файл, выполнения запроса к БД, взаимодействия с удаленном сервисом) и при этом в теле метода имеется единственная конструкция вызова другого асинхронного метода, которым завершается работа исходного метода, то конструкцию async/await можно опустить.
* Также, если метод был объявлен как асинхронный (например, в интерфейсе), но, по каким-то причинам в логике метода нет вызова иных асинхронных методов или запуска асинхронных задач (например, это заглушка в тестах), то можно использовать специальные методы для возвращения завершенной задачи:
* Task.CompletedTask. Свойство. Возвращает задачу, которая уже завершилась успешно.
* Task.FromResult. Метод. Создает задачу Task , которая успешно завершается с указанным результатом.

**29.Task. Зачем нужен класс Task? Как выполнять методы асинхронно с его помощью?**

Класс представляет отдельную операцию, которая не возвращает значение и обычно выполняется асинхронно.

Task-объекты – это один из центральных компонентов асинхронной модели на основе задач. Работа, выполняемая Task‐объектом, обычно выполняется асинхронно в потоке пула потоков, а не синхронно в основном потоке приложения.

Зачем нужен класс Task:

* Асинхронное выполнение: Task позволяет запускать операции асинхронно, не блокируя основной поток приложения.
* Управление потоками: Операции, выполняемые объектом Task, обычно выполняются в потоках из пула потоков, что позволяет эффективно использовать ресурсы системы.
* Композиция задач: Task позволяет легко объединять несколько асинхронных операций, создавая сложные асинхронные рабочие процессы.

Как выполнять методы асинхронно с помощью Task:

* Создание задачи: Задача может быть создана с использованием метода Task.Run, который запускает метод в пуле потоков.
* Использование async и await: В асинхронных методах можно использовать ключевые слова async и await для упрощения работы с задачами. Это позволяет приостановить выполнение метода до завершения асинхронной операции и продолжить выполнение после её завершения.
* Ожидание завершения задачи: Можно ожидать завершения задачи с помощью await, что позволяет продолжить выполнение кода после завершения задачи.
* Продолжение задач: Можно задать продолжение для задачи, которое выполнится после завершения основной задачи.
* Возврат значения: Если задача должна вернуть значение, можно использовать Task<T>, где T — это тип возвращаемого значения. Это позволяет задаче вернуть результат после завершения

**30.Task. Как получить результат выполнения асинхронной операции?**

Свойство Result:

Позволяет получить результат выполнения задачи.

Блокирует поток до завершения операции.

В случае ошибки выбрасывает исключение AggregateException.

Метод GetAwaiter().GetResult():

Используется для ожидания завершения асинхронной операции.

Предпочтительнее, чем Result, так как напрямую распространяет исключения.

**31.Локализация. Зачем нужна локализация? Каков подход к локализации приложения?**

Локализация приложения — это процесс адаптации приложения для работы с различными языками и региональными параметрами.

Разработка подобного приложения, включает следующие шаги:

* Глобализация
* анализ локализуемости
* локализация

### Подход к локализации приложения:

1. Использование ресурсов: Хранение всех локализуемых строк и других ресурсов (например, изображений, звуков) в отдельных файлах ресурсов для каждого языка.

2. Поддержка Unicode: Убедиться, что приложение поддерживает Unicode для корректного отображения символов всех языков.

3. Адаптация пользовательского интерфейса: Учет различий в длине строк, направлении текста (слева направо или справа налево) и других региональных особенностей.

4. Тестирование: Проведение тщательного тестирования локализованного приложения для выявления и исправления ошибок, связанных с локализацией.

**32.Глобализация. Каковы особенности обработки строк?**

независимость кода от региональных параметров и языковых настроек

Глобализация приложений подразумевает создание кода, который может работать с различными языками и региональными настройками.

Особенности обработки строк в глобализированных приложениях включают:

1. Использование Unicode:

- Строки рекомендуется хранить в формате Unicode, так как он поддерживает почти все символы, используемые в языках по всему миру.

- В .NET строки в формате Unicode состоят из объектов Char, каждый из которых представляет собой единицу кода UTF-16.

2. Избегание кодовых страниц:

- Кодовые страницы, такие как Windows-1252, могут использоваться для представления кодировок, но их использование может привести к проблемам, так как **интерпретация символов зависит от конкретной кодовой страницы.**

- Рекомендуется избегать использования кодовых страниц в глобализированных приложениях.

3. Обработка строк целиком:

- При возможности строки следует обрабатывать целиком, а не как последовательности отдельных символов. Это важно для корректной сортировки и поиска подстрок, особенно в языках с нелатинскими символами.

4. Проверка на равенство:

- Сравнения на равенство должны быть порядковыми (**посимвольно**, без учета языка и локали), если они не связаны с лингвистическими особенностями. Это важно для таких операций, как проверка пароля или существования файла.

- При поиске подстрок также следует использовать порядковые сравнения, если цель не связана с языковыми особенностями.

5. Сортировка:

* Строки, отображаемые в пользовательском интерфейсе, должны сортироваться с учетом языка и региональных параметров.
* По умолчанию строки будут отсортированы с помощью соглашений сортировки текущего языка и региональных параметров
* В .NET сортировка строк зависит от операционной системы и использует таблицы, содержащие данные по весам сортировки и нормализации строк.

6. Объединение строк:

- Следует избегать использования составных строк, полученных во время выполнения программы, так как они могут быть трудны для локализации из-за грамматических особенностей исходного языка.

7. Файлы ресурсов:

- Использование файлов ресурсов для хранения строк и других ресурсов, отображаемых в пользовательском интерфейсе, имеет ряд преимуществ:

- Все строки находятся в одном месте, что упрощает их изменение для конкретного языка.

- Исключается дублирование строк, что снижает вероятность ошибок при изменении строк.

- Позволяет включать нестроковые ресурсы, такие как изображения или двоичные данные, что упрощает их извлечение и управление.

**33.Глобализация. Каковы особенности обработки дат и чисел?**

Способ обработки значений даты и времени зависит от того, *отображаются* ли они в интерфейсе пользователя или *сохраняются*.

* Отображение
* Как правило, когда даты и время отображаются в пользовательском интерфейсе, необходимо использовать соглашения о форматировании, принятые в языке и региональных параметрах пользователя.
* Сохранение:
* Никогда не следует сохранять данные даты и времени в формате, который может зависеть от языка и региональных параметров (в частности, в строку).
* Для точного сохранения даты и времени рекомендуется использовать строки стандартного формата, такие как формат "o" (цикл обработки) или преобразовывать время в формат UTC и сохранять его в виде строки с использованием форматов "r" (RFC1123) или "u" (универсальное сортируемое).

- При десериализации данных в целевой системе необходимо учитывать часовой пояс и преобразовывать время в местное.

3. Всемирное координированное время (UTC):

- UTC используется как стандарт для регулирования времени по всему миру.

- Часовые пояса выражаются как смещения от UTC, что позволяет унифицировать обработку времени в различных регионах.

4. Арифметические операции:

- Арифметические операции с датами и временем не учитывают часовые пояса и правила коррекции часовых поясов, что может привести к неточным результатам.

- Для точных результатов необходимо преобразовывать время в формат UTC перед выполнением операций и обратно в местное время после их завершения.

5. Вывод:

- При выводе названий дней недели или месяцев необходимо учитывать языковые и региональные настройки, используя специфичные инструменты, такие как строки даты и времени пользовательского формата или свойства объекта `DateTimeFormatInfo`.

### Обработка чисел:

1. Отображение в пользовательском интерфейсе:

- Числа, особенно денежные значения, должны отображаться в соответствии с языковыми и региональными настройками пользователя.

2. Сохранение:

- Числовые данные не должны сохраняться в формате, зависящем от языка и региональных параметров.

- Для сохранения числовых данных можно использовать строковое представление числа с использованием строки пользовательского формата или инвариантного формата, не зависящего от языка и региональных параметров.

3. Сериализация значений валюты:

- Значения валют должны сериализоваться вместе с информацией о языке, чтобы обеспечить корректную десериализацию независимо от текущего языка и региональных параметров системы.

**34.В чем заключается анализ локалезуемости?**

Анализ локализуемости заключается в проверке приложения на предмет его готовности к локализации. Этот процесс включает несколько ключевых этапов:

1. Реализация рекомендаций по глобализации:

- Проверяется, были ли соблюдены все рекомендации по глобализации на этапах проектирования и разработки приложения. Если да, то анализ локализуемости становится формальным этапом контроля качества. В противном случае необходимо изучить и внедрить рекомендации по глобализации, а также исправить проблемы в исходном коде, которые могут помешать локализации.

2. Правильно ли обрабатываются компоненты, чувствительные к языку и региональным параметрам?

- Проверяется, как приложение обрабатывает компоненты, которые могут значительно различаться в зависимости от языка и региональных параметров. Это включает такие области, как:

- Адреса

- Номера телефонов

- Размеры бумаги

- Единицы измерения длины, веса, площади, объема и температуры

- В большинстве случаев для корректной обработки этих областей требуется написание специального кода.

3. Тестирование приложения с данными на других языках:

- Перед локализацией приложение тестируется с использованием данных на других языках в международных версиях операционной системы. Это помогает выявить различные проблемы, такие как:

- Неправильная десериализация данных в разных версиях операционной системы.

- Числовые данные, которые не соответствуют правилам текущих языка и региональных параметров.

- Неправильное отображение даты и времени.

- Проблемы с поиском ресурсов из-за отсутствия языка и региональных параметров по умолчанию.

- Строки, отображаемые в нестандартном порядке для определенных языков и региональных параметров.

- Непредвиденные результаты при сравнении строк или проверке на равенство.

**35.Ресурсы. Как используется ресурсы в проектах. Зачем нужен менеджер ресурсов?**

Ресурсы в проектах используются для хранения неисполняемых данных, которые логически связаны с приложением. Они могут включать в себя строки, изображения, объекты и другие данные, которые отображаются в приложении, например, сообщения об ошибках или элементы пользовательского интерфейса.

### Создание и использование ресурсов:

- Форматы файлов: Ресурсы часто создаются в виде текстовых (txt) или XML-файлов (.resx), которые затем компилируются в двоичные файлы ресурсов с помощью генераторов файлов ресурсов.

- Локализация: Ресурсы могут быть локализованы для конкретных языков и региональных параметров, что позволяет создавать переведенные версии приложений. Для этого назначаются нейтральные или резервные языки и региональные параметры, которые используются в случае отсутствия подходящих локализованных ресурсов.

### Развертывание ресурсов:

- Вспомогательные сборки: Локализованные ресурсы развертываются во вспомогательных сборках, каждая из которых содержит ресурсы для одного языка и региональных параметров. Это позволяет обновлять или добавлять поддержку новых языков без изменения основной сборки приложения.

### Извлечение ресурсов:

- Менеджер ресурсов: Во время выполнения приложение загружает соответствующие локализованные ресурсы для каждого потока на основе языка и региональных параметров. Основной код приложения обращается к менеджеру ресурсов для получения ресурса по имени. Менеджер ресурсов находит локализованную версию ресурса и передает её в основной код.

### Зачем нужен менеджер ресурсов:

Менеджер ресурсов играет ключевую роль в управлении ресурсами приложения. Он отвечает за:

1. Загрузку ресурсов: Загрузку локализованных ресурсов в зависимости от языка и региональных параметров пользователя.

2. Управление доступом: Обеспечение доступа к ресурсам по имени, что позволяет основному коду приложения легко получать необходимые данные.

3. Обработку локализации: Управление различными версиями ресурсов для разных языков и регионов, обеспечивая корректное отображение данных в зависимости от настроек пользователя.

**36.Рефлексия. Что такое рефлексия? Какие основные элементы и классы в рефлексии?**

Рефлексия — это процесс, позволяющий программе исследовать и манипулировать метаданными типов во время выполнения. Она предоставляет возможность динамически получать информацию о типах, создавать экземпляры типов, вызывать методы и получать доступ к полям и свойствам объектов. Рефлексия также позволяет работать с атрибутами, которые могут быть применены к типам и их членам.

Основные классы, используемые в рефлексии, включают:

1. Assembly: Предоставляет доступ к метаданным сборки и позволяет загружать сборки, исследовать содержащиеся в них модули и типы.

2. Module: Описывает модуль в сборке, предоставляя доступ к типам, содержащимся в модуле.

3. Type: Основной класс для работы с типами. Он предоставляет методы для исследования типов, создания их экземпляров, вызова методов и доступа к полям и свойствам.

4. CustomAttributeData: Позволяет получать информацию о пользовательских атрибутах, примененных к типам и их членам.

5. PropertyInfo: Предоставляет доступ к свойствам типа, позволяя получать и устанавливать их значения.

6. FieldInfo: Позволяет получать информацию о полях типа и манипулировать их значениями.

7. EventInfo: Предоставляет доступ к событиям типа, позволяя подписываться и отписываться от событий.

8. ConstructorInfo: Позволяет получать информацию о конструкторах типа и создавать новые экземпляры типа.

9. MethodInfo: Предоставляет доступ к методам типа, позволяя вызывать их динамически.

10. ParameterInfo: Позволяет получать информацию о параметрах методов и конструкторов.

**37.Разбор типа. Как получить доступ к типу? Как получить доступ к элементам типа?**

Для получения доступа к типу и его элементам в .NET используется рефлексия. Это позволяет динамически исследовать и взаимодействовать с типами и их членами во время выполнения. Вот как можно получить доступ к типу и его элементам:

### Получение доступа к типу

1. Использование класса `Type`:

- Основной способ получения информации о типе — это использование класса `Type`. Объект `Type` можно получить, например, с помощью метода `GetType()` или статического метода `Type.GetType()`.

2. Методы для получения информации о членах типа:

- `GetMembers()`: Возвращает массив объектов `MemberInfo`, представляющих все открытые члены текущего типа.

- `GetNestedTypes()`: Возвращает массив объектов `Type`, представляющих открытые типы, вложенные в текущий объект `Type`.

### Получение доступа к элементам типа

1. Методы и конструкторы:

- `MethodInfo`: Используется для получения информации о методах, таких как имя, возвращаемый тип, параметры и модификаторы доступа. Методы `GetMethods()` или `GetMethod()` класса `Type` используются для вызова определенного метода.

- `ConstructorInfo`: Используется для получения информации о конструкторах, таких как имя, параметры и модификаторы доступа. Методы `GetConstructors()` или `GetConstructor()` класса `Type` используются для вызова конкретного конструктора.

2. Поля:

- `FieldInfo`: Используется для получения информации о полях, таких как имя, тип данных, модификаторы доступа и сведения о реализации. Методы `GetFields()` или `GetField()` класса `Type` используются для доступа к полям.

3. Свойства:

- `PropertyInfo`: Используется для получения информации о свойствах, таких как имя, тип данных, объявляющий тип и состояние доступности. Методы `GetProperties()` или `GetProperty()` класса `Type` используются для доступа к свойствам.

4. События:

- `EventInfo`: Используется для получения информации о событиях, таких как имя, тип данных обработчика событий и пользовательские атрибуты. Методы `GetEvents()` или `GetEvent()` класса `Type` используются для доступа к событиям.

5. Параметры:

- `ParameterInfo`: Используется для получения информации о параметрах методов, таких как имя, тип данных и вид параметра (входной или выходной).

**38.Создание объект. Как создается объект через рефлексию?**

Создание объекта через рефлексию в .NET осуществляется с помощью класса `Activator`, который предоставляет методы для создания экземпляров типов.

Основные методы включают:

1. CreateInstance(Type): Создает экземпляр указанного типа с использованием конструктора без параметров.

2. CreateInstance с параметрами: Создает экземпляр типа с использованием конструктора, который соответствует указанным параметрам, включая `BindingFlags`, `Binder`, аргументы и информацию о культуре.

3. CreateInstance(String, String): Создает экземпляр типа с заданным именем, используя именованную сборку и конструктор без параметров.

4. CreateInstance с дополнительными параметрами: Создает экземпляр типа с заданным именем, используя именованную сборку и конструктор, который соответствует заданным параметрам, включая флаги привязки и информацию о культуре.

Для создания универсальных объектов необходимо:

1. Получить объект `Type` универсального типа.

2. Создать массив объектов `Type` для параметров универсального типа.

3. Использовать метод `MakeGenericType` для создания нового типа с конкретными параметрами.

4. Использовать метод `CreateInstance` для получения нового экземпляра этого типа.

Класс `ObjectHandle` используется для передачи объектов между доменами приложений без загрузки метаданных. Метод `Unwrap` возвращает свернутый объект.

**39.Работа с объектом. Как считать данные из объекта типа через рефлексию? Как заполнить объекта типа через рефлексию?**

### Чтение данных из объекта:

1. Поля:

- Используйте метод `FieldInfo.GetValue`, чтобы получить значение поля объекта. Этот метод принимает объект, из которого нужно получить значение поля.

2. Свойства:

- Используйте метод `PropertyInfo.GetValue`, чтобы получить значение свойства объекта. Этот метод также принимает объект, из которого нужно получить значение свойства.

3. События:

- Для событий нет прямого метода получения значения, так как события представляют собой механизм уведомления. Однако можно получить информацию о событиях, используя `EventInfo`.

### Заполнение объекта данными:

1. Поля:

- Используйте метод `FieldInfo.SetValue`, чтобы установить значение поля объекта. Этот метод принимает объект, значение которого нужно установить, и новое значение.

2. Свойства:

- Используйте метод `PropertyInfo.SetValue`, чтобы установить значение свойства объекта. Этот метод принимает объект, значение которого нужно установить, и новое значение. Для индексированных свойств может потребоваться дополнительный параметр для указания индекса.

3. События:

- Используйте метод `EventInfo.AddEventHandler`, чтобы добавить обработчик событий к источнику события. Этот метод принимает источник события и делегат, который будет вызван при возникновении события.

### Вызов методов:

1. InvokeMember:

- Метод `Type.InvokeMember` позволяет вызывать методы, устанавливать или получать значения полей и свойств. Он принимает имя члена, флаги привязки, объект, для которого вызывается член, и массив аргументов.

2. MethodInfo.Invoke:

- Метод `MethodInfo.Invoke` используется для вызова метода или конструктора, представленного текущим экземпляром. Он принимает объект, для которого вызывается метод, и список аргументов.

**40.Атрибуты. Назначение атрибутов. Как описываются? Как применяются?**

Атрибуты в .NET используются для добавления метаданных к элементам программирования, таким как типы, поля, методы и свойства. Атрибуты помогают управлять сериализацией данных, усиливать безопасность, ограничивать оптимизацию JIT-компилятором и многое другое.

**Назначение атрибутов**

* Сериализация данных: Атрибуты могут определять, как данные должны быть сериализованы.
* Безопасность: Они могут задавать характеристики, используемые для усиления безопасности.
* Оптимизация и отладка: Атрибуты могут ограничивать оптимизацию JIT-компилятором, облегчая отладку кода.
* Метаданные: Они могут содержать информацию, такую как имя файла или автор кода.
* Управление видимостью: Атрибуты могут управлять видимостью элементов управления и членов при разработке форм.

**Описание атрибутов**

Атрибуты описываются как классы, которые наследуют от System.Attribute. Они могут содержать методы для хранения и извлечения данных. Основные этапы разработки настраиваемых атрибутов включают:

* Применение атрибута AttributeUsageAttribute: Этот атрибут управляет тем, как настраиваемый атрибут может быть использован.
* Объявление класса атрибута: Класс атрибута должен быть объявлен как открытый и, по соглашению, его имя должно заканчиваться словом Attribute.
* Объявление конструкторов: Конструкторы используются для инициализации атрибутов.
* Объявление свойств: Свойства позволяют определять именованные параметры и предоставлять доступ к значениям, хранящимся в атрибуте.

**Применение атрибутов**

Определение или использование атрибута: Можно определить новый атрибут или использовать существующий.

Применение атрибута: Атрибут применяется к элементу кода, помещая его перед элементом. В C# атрибут заключается в квадратные скобки.

Указание параметров: Можно указать позиционные и именованные параметры атрибута.

Применение атрибутов — это просто **указание их над классами, методами или свойствами**:

