1. Поясните назначение паттерна **Repository**.

Паттерн Repository создаёт прослойку между бизнес-логикой и хранилищем данных. Он предоставляет единый интерфейс для выполнения стандартных операций (например, получение, добавление, обновление, удаление данных), скрывая детали, как именно данные извлекаются или сохраняются (будь то база данных, файлы или внешние сервисы). Это позволяет бизнес-логике работать с данными, не заботясь о способе их хранения, а также упрощает тестирование и изменение источника данных без необходимости переписывать основной код.

1. Для чего в паттерне **Repository** применяется интерфейс **IRepository**?

Интерфейс IRepository задаёт набор правил (или контракт), которым должна соответствовать любая реализация репозитория. Это значит, что бизнес-логика будет работать с данными через этот интерфейс, не зависимо от того, как именно устроена работа с данными (например, через базу данных, файлы или API). Благодаря интерфейсу можно легко заменить одну реализацию репозитория на другую, а также использовать моки при тестировании.

«Набор правил» в данном контексте означает, что интерфейс определяет конкретный перечень методов, свойств и поведения, которые должны быть реализованы в классе, который его наследует. Это как контракт: если класс заявляет, что он реализует интерфейс IRepository, он обязуется предоставить реализацию всех объявленных в нём методов, что гарантирует единообразие работы с данными независимо от конкретной реализации.

Интерфейс IRepository задаёт контракт для репозитория, то есть перечень методов и свойств, которые должны быть реализованы. Это позволяет бизнес-логике работать с данными через единый интерфейс, не заботясь о том, как именно эти данные хранятся или извлекаются. Благодаря этому можно легко заменить одну реализацию репозитория на другую (например, для тестирования или изменения способа хранения данных) без изменения остального кода приложения.

1. Для чего применяется интерфейс IDisposable?

Интерфейс **IDisposable** применяется для освобождения ресурсов, которые не управляются сборщиком мусора. То есть, если объект использует ресурсы, такие как файлы, сетевые соединения или соединения с базой данных, его нужно явно "очищать" после использования. Реализуя IDisposable, класс предоставляет метод **Dispose()**, который вызывается для корректного закрытия и освобождения этих ресурсов. Это помогает предотвратить утечки ресурсов и обеспечивает более предсказуемое управление ресурсами в приложении.

4.Поясните работу следующей конструкции языка С#



**Что происходит внутри скобок:**

1. **Объявление переменной:**
   * Repository repository — здесь создаётся переменная с именем repository типа Repository. Это означает, что в будущем эта переменная будет хранить ссылку на объект, который представляет репозиторий.
2. **Вызов метода:**
   * Repository.Create("Celebrities") — это вызов статического метода Create класса Repository.
   * Аргумент "Celebrities" передается в метод. Этот аргумент может указывать, с каким именно набором данных или коллекцией должен работать создаваемый репозиторий.
   * Метод Create обрабатывает этот аргумент и возвращает новый объект типа Repository, настроенный для работы с данными, связанными с "Celebrities".
3. **Присваивание результата:**
   * Результатом вызова метода Repository.Create("Celebrities") является объект, который сразу присваивается переменной repository.
   * Таким образом, переменная repository теперь содержит ссылку на этот новый объект и может быть использована внутри блока using.
4. **Оператор using:**
   * Хотя сам оператор using находится вне круглых скобок, важно понимать, что объявление и инициализация переменной происходит внутри скобок оператора using.
   * Это гарантирует, что после выполнения кода внутри блока using автоматически будет вызван метод Dispose() у объекта repository, что освободит все связанные с ним ресурсы.

**Итог:**

Внутри круглых скобок происходит создание переменной repository типа Repository и её инициализация результатом вызова метода Repository.Create("Celebrities"). Таким образом, вы получаете готовый для использования объект репозитория, который настроен для работы с данными о знаменитостях. Это позволяет безопасно работать с ресурсами, поскольку после завершения блока using объект будет автоматически закрыт и освобожден.

**Удобство использования:**  
Используя статический метод Create, код становится чище и понятнее: вы явно видите, что создается репозиторий, предназначенный для работы с данными о знаменитостях, без лишних деталей реализации.

5.Поясните смысл слов: сериализация и десерилизация.

**Сериализация** — это процесс преобразования объекта или его состояния в формат, пригодный для хранения или передачи (например, в виде JSON, XML или бинарного потока). Это нужно для того, чтобы сохранить объект на диск, передать по сети или использовать в других системах, где нельзя работать напрямую с объектом.

**Десериализация** — это обратный процесс, когда данные, полученные в сериализованном формате, преобразуются обратно в объект с его исходным состоянием и структурой.

Таким образом, сериализация позволяет "сохранить" состояние объекта, а десериализация — "восстановить" его для дальнейшего использования.

6.Расшифруйте аббревиатуру JSON, поясните смысл.

**JSON** расшифровывается как **JavaScript Object Notation** — обозначение объектов в JavaScript. Это легковесный текстовый формат для обмена данными, который:

* **Читаем человеком:** Структура понятна и легко воспринимается, например, через пары «ключ-значение».
* **Универсален:** Используется для передачи данных между сервером и клиентом, а также между различными приложениями.
* **Удобен для парсинга:** Благодаря своей структуре, данные в формате JSON легко обрабатывать программно на большинстве языков программирования.

Таким образом, JSON предоставляет удобный и эффективный способ представления структурированных данных в виде текста.

**JSON** — это формат, предназначенный для структурированного представления данных в виде текста. Его смысл заключается в следующем:

* **Легковесность:** JSON занимает мало места, что делает его удобным для передачи данных по сети.
* **Читаемость:** Его структура на основе пар «ключ-значение» легко воспринимается человеком.
* **Универсальность:** JSON широко поддерживается во многих языках программирования, что позволяет использовать его для обмена данными между различными системами.
* **Простота обработки:** Благодаря стандартному синтаксису его легко парсить (разбирать) и генерировать, что делает его удобным для работы в веб-приложениях и API.

Таким образом, JSON используется для передачи, хранения и обмена данными между сервером и клиентом, обеспечивая простой и эффективный способ взаимодействия между различными компонентами системы.

**7ю.** Кто такой Эдсгер Дейкстра?☺

Эдсгер Дейкстра — один из самых влиятельных учёных в области информатики. Он известен своими фундаментальными работами по алгоритмам и структурированному программированию. Вот несколько ключевых моментов о нём:

* **Алгоритмы:** Дейкстра разработал алгоритм поиска кратчайшего пути в графе, который широко используется в сетевых протоколах и навигационных системах.
* **Структурированное программирование:** Он был одним из сторонников отказа от безусловных переходов (оператора goto), продвигая более понятный и надёжный стиль программирования.
* **Философия программирования:** Дейкстра много размышлял о том, как правильно писать программы, чтобы они были простыми, читаемыми и безопасными.
* **Награды и признание:** За его вклад в науку он получил множество престижных наград, включая премию Тьюринга.

Таким образом, Эдсгер Дейкстра оказал огромное влияние на развитие теоретической и практической информатики, и его идеи продолжают влиять на современные подходы к разработке программного обеспечения.