***boxing і unboxing***

Упаковка (boxing) предполагает преобразование объекта значимого типа (например, типа int) к типу object. При упаковке общеязыковая среда CLR обертывает значение в объект типа System.Object и сохраняет его в управляемой куче (хипе). Распаковка (unboxing), наоборот, предполагает преобразование объекта типа object к значимому типу. Упаковка и распаковка ведут к снижению производительности, так как системе надо осуществить необходимые преобразования.

Кроме того, существует другая проблема - проблема безопасности типов.

**ref і out**

Однако в отличие от ref для переменных, которые передаются с ключевым словам out, не требуется инициализация. И кроме того, вызываемый метод должен обязательно присвоить им значение.

**SOLID**

**Принцип единственной ответственности (Single responsibility)**  
*«На каждый объект должна быть возложена одна единственная обязанность»*  
Для этого проверяем, сколько у нас есть причин для изменения класса — если больше одной, то следует разбить данный класс.

**Принцип открытости/закрытости (Open-closed)**  
*«Программные сущности должны быть открыты для расширения, но закрыты для модификации»*  
Для этого представляем наш класс как «чёрный ящик» и смотрим, можем ли в таком случае изменить его поведение.

**Принцип подстановки Барбары Лисков (Liskov substitution)**  
*«Объекты в программе могут быть заменены их наследниками без изменения свойств программы»*  
Для этого проверяем, не усилили ли мы предусловия и не ослабили ли постусловия. Если это произошло — то принцип не соблюдается

**Принцип разделения интерфейса (Interface segregation)**  
*«Много специализированных интерфейсов лучше, чем один универсальный»*  
Проверяем, насколько много интерфейс содержит методов и насколько разные функции накладываются на эти методы, и если необходимо — разбиваем интерфейсы.

**Принцип инверсии зависимостей (Dependency Invertion)**  
*«Зависимости должны строится относительно абстракций, а не деталей»*  
Проверяем, зависят ли классы от каких-то других классов(непосредственно инстанцируют объекты других классов и т.д) и если эта зависимость имеет место, заменяем на зависимость от абстракции.

***OOP (види поліморфізма (статичний, динамічний))***

## **Статический полиморфизм**

Механизм связывания функции с объектом во время компиляции называется ранним связыванием. Он также называется статической привязкой. C # предоставляет два метода для реализации статического полиморфизма. **Перегрузка функций. Перегрузка оператора**

**Динамічний поліморфізм :** **Полиморфизм включения** (наследование) — поддерживается, есть интерфейсы, наследование классов, одиночная и множественная диспетчеризация. Примитивные типы неполиморфны.

***Task замість void?***

Суть в том, что это async+void может привести к сбою системы и обычно должно использоваться только в обработчиках событий на стороне пользовательского интерфейса.

Вызывающий объект асинхронного метода, возвращающего void, не может перехватывать создаваемые методом исключения

***EF(види загрузок) lazy loading і т.д.***

**Eager loading** (жадная загрузка) с помощью метода **Include()** в который передается навигационное свойство. (LEFT JOIN) (если данные уже ранее были загружены в контекст данных или просто ранее были в него добавлены, то можно не использовать метод Include для их получения, так как они уже в контексте) I ThenInclude – загрузка моделей со сложной структурой

**Explicit loading** (явная загрузка) с помощью метода **Load()** Выражение db.Users.Where(p=>p.CompanyId==company.Id).Load() загружает всех пользователей в контекст. Подвыражение Where(p=>p.CompanyId==company.Id) означает, что загружаются только те пользователи, у которых свойство CompanyId соответствует свойству Id ранее полученной компании. После этого нам не надо подгружать связанные данные, так как они уже есть в контексте.

**Lazy loading** (ленивая загрузка)предполагает неявную автоматическую загрузку связанных данных при обращении к навигационному свойству. Однако здесь есть ряд условий: 1)При конфигурации контекста данных вызвать метод **UseLazyLoadingProxies()**

2)Все навигационные свойства должны быть определены как виртуальные (то есть с модификатором **virtual**), при этом сами классы моделей должны быть открыты для наследования

3) в проект через nuget пакет **Microsoft.EntityFrameworkCore.Proxies**.

**Generic Repository -** представляет паттерн, задача которого заключается в управлении доступом к источнику данных (для инкапсулирования логики работы с источниками данных). Паттерн **Unit of Work** позволяет упростить работу с различными репозиториями и дает уверенность, что все репозитории будут использовать один и тот же контекст данных.

***DESIGN PATTERNS(ГРУПИ, ЗНАТИ ХОЧ БИ ПО ОДНОМУ ПАТЕРНУ З ГРУПИ І ВИКОРИСТАТИ)***

**Породжувальні.** Відповідають за зручне та безпечне створення нових об'єктів або навіть цілих сімейств об'єктів.

**Одинак** — це породжувальний патерн проектування, який гарантує, що клас має лише один екземпляр, та надає глобальну точку доступу до нього.**Коли в програмі повинен бути єдиний екземпляр якого-небудь класу, доступний усім клієнтам (наприклад, спільний доступ до бази даних з різних частин програми).**

**Структурні**. Відповідають за побудову зручних в підтримці ієрархій класів.

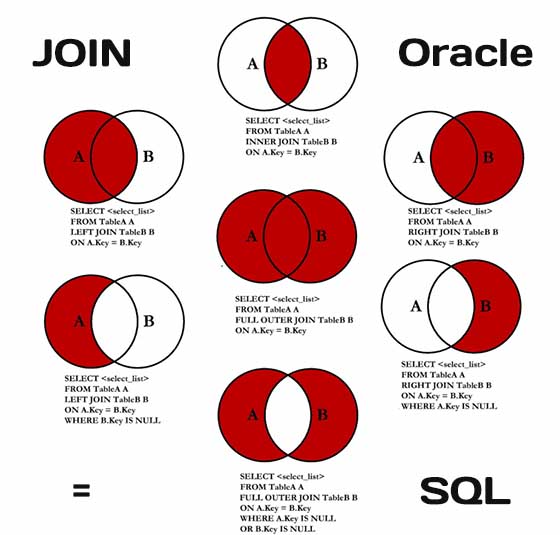
**Міст** — це структурний патерн проектування, який розділяє один або кілька класів на дві окремі ієрархії — абстракцію та реалізацію, дозволяючи змінювати код в одній гілці класів, незалежно від іншої. **Якщо клас потрібно розширювати в двох незалежних площинах**

**Поведінкові**.Вирішують завдання ефективної та безпечної взаємодії між об'єктами програми.

**Стратегія** — це поведінковий патерн проектування, який визначає сімейство схожих алгоритмів і розміщує кожен з них у власному класі. Після цього алгоритми можна заміняти один на інший прямо під час виконання програми. **Якщо вам потрібно використовувати різні варіації якого-небудь алгоритму всередині одного об’єкта. (прокладати шлях різними видами транспорту)**

**Посередник** — це поведінковий патерн проектування, що дає змогу зменшити зв’язаність великої кількості класів між собою, завдяки переміщенню цих зв’язків до одного класу-посередника.

***joins (SQL)***

 ***індексація***

Индекс-это не что иное, как структура данных, в которой **хранятся значения для определенного столбца** в таблице. Индекс создается в столбце таблицы.

Пример: У нас есть таблица базы данных под названием User с тремя столбцами – Name , Age и Address . Предположим, что таблица User содержит тысячи строк.

Теперь предположим, что мы хотим выполнить запрос, чтобы найти все сведения о любых пользователях с именем 'John'. Если мы выполним следующий запрос:

SELECT \* FROM User

WHERE Name = 'John'

Программное обеспечение базы данных буквально должно было бы просмотреть каждую строку в таблице User , чтобы увидеть, является ли Name для этой строки "Джон". Это займет много времени.

Вот где нам помогает index : **индекс используется для ускорения поисковых запросов, существенно сокращая количество записей/строк в таблице, которую необходимо изучить** .

Как создать индекс:

CREATE INDEX name\_index

ON User (Name)

index состоит из **значений столбцов(например , John) из одной таблицы**, и эти значения хранятся в **структуре данных** .

Таким образом, теперь база данных будет использовать индекс для поиска сотрудников по имени Джон , потому что индекс, предположительно, будет отсортирован в алфавитном порядке по имени пользователя. И, поскольку он отсортирован, это означает поиск имени это намного быстрее, потому что все имена, начинающиеся с “J”, будут правильными рядом друг с другом в индексе!

**транзакція**

SQL Transaction - это unit of work, применяемая к базе данных (например, DB2, Oracle, MySQL и т.д.).

Это последовательность операций, выполненных на базе данных в логическом порядке.

SQL-запросы транзакции будут либо фиксироваться, либо откатываться назад.

Транзакция включает в себя два результата: либо Успех, либо Неудача.

Реализация паттерна Unit of Work следит за всеми действиями приложения, которые могут изменить БД в рамках одного бизнес-действия. Когда бизнес-действие завершается, Unit of Work выявляет все изменения и вносит их в БД.

## Свойства транзакции SQL

Обычно транзакции имеют четыре стандартных свойства и обозначаются как 'ACID'.

Atomicity (Атомарность) операции прерываются в момент сбоя, что обеспечивает успешное завершение операций в других частях прилоэения.

Consistency (Согласованность) - после успешной фиксации, она гарантирует, что состояние базы данных будет изменено.

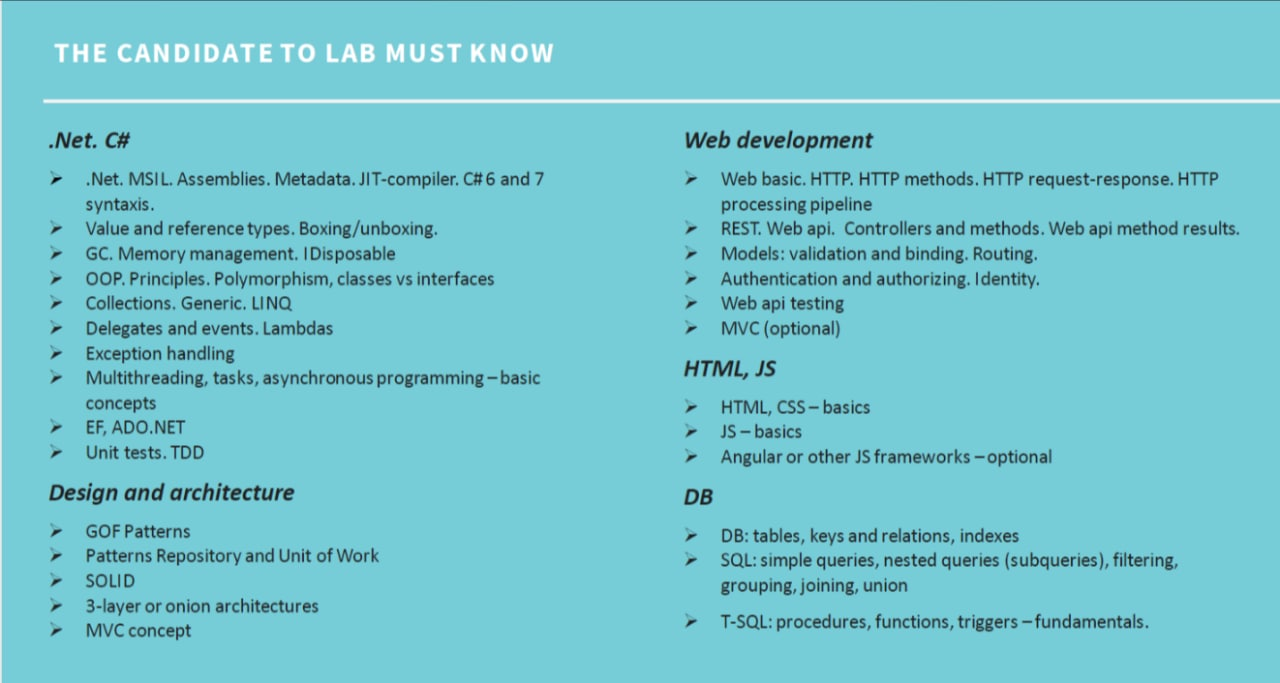
Isolation (Изоляция) - транзакции будут осуществляться независимо и прозрачно.

Durability (Надежность) - в случае сбоя системы, это гарантирует, что результат зафиксированной транзакции будет сохранен.

**хранимие процедури** (stored procedure) То есть по сути хранимые процедуры представляют набор инструкций, которые выполняются как единое целое. Тем самым хранимые процедуры позволяют упростить комплексные операции и вынести их в единый объект

***трігери***

Триггеры по своей сути представляют обработчики событий. Они выполняются при наступлении какого-либо простого действия в SQL. Такими действиями обычно являются: удаление, вставка и обновление данных.



***CLR****("Common Language Runtime", "общеязыковая исполняющая среда") - это компонент .NET Framework, основной задачей которого является управление интерпретацией и исполнением кода IL. CLR отвечает за изоляцию памяти приложений, проверку типов, безопасность кода, преобразование IL в машинный код.*

***IL****(Intermediate Language) - код, содержащий набор инструкций, не зависящих от платформы. Иными словами, после компиляции исходного кода он преобразуется не в код для какой-то определенной платформы, а в промежуточный код на языке IL.*

***CLS****("Common Language Specification", общеязыковая спецификация) - это набор правил, следуя которым разработчики достигают бесконфликтной работы во всех языках .NET.*

### *****Сборка******(assembly) - один или несколько файлов, содержащий логический набор функциональности (код и другие данные, связанные с кодом). Бывают статические сборки, хранящиеся на диске, и динамические, которые создаются во время выполнения программы. Сборка - это базовый блок приложения, все ресурсы, относящиеся к ней, доступны или только внутри этого блока, или экспортируются наружу. При выполнении сборка задает область видимости имен и следит за ее соблюдением.*

### *Namespace (пространство имен) является логическим соглашением, используемым во время разработки, в то время как assembly (сборка) устанавливает область видимости имени в процессе выполнения.*

***CAS****- это защитная технология, позволяющая установить ограничение на исполнение управляемого кода. Таким образом можно определить разрешения и установить права доступа к ресурсам компьютера.*

**System.Array.CopyTo() и System.Array.Clone()?**

*Первая операция осуществляет глубокое копирование массива, а вторая – поверхностное.*

### Что обозначает ключевое слово “virtual” для метода или свойства?

*То, что метод или свойство может быть перекрыто.*

### Чем перекрытый метод отличается от перегруженного метода?

*При перекрывании метода мы изменяем его поведение в классе наследнике. Перегрузка метода просто приводит к использованию другого метода с тем же именем внутри класса.*

### Можно ли объявить перекрытый метод статическим, если перекрываемый метод не является статическим?

*Нет. Сигнатура виртуального метода должна остаться постоянной, кроме замены ключевого слова virtual на ключевое слово override.*

**REST архітектура**

В общем случае REST является очень простым интерфейсом управления информацией без использования каких-то дополнительных внутренних прослоек. Каждая единица информации однозначно определяется глобальным идентификатором, таким как URL. Каждая URL в свою очередь имеет строго заданный формат.  
Наиболее распространенный протокол конечно же HTTP. Так вот, для HTTP действие над данными задается с помощью методов:   
GET (получить),   
PUT (заменить),   
POST (добавить),

DELETE (удалить).

Таким образом, действия CRUD (Create-Read-Updtae-Delete) могут выполняться как со всеми 4-мя методами, так и только с помощью GET и POST.  
  
**H**yper**T**ext **T**ransfer **P**rotocol, протокол передачі [гіпертекстових документів](https://uk.wikipedia.org/wiki/%D0%93%D1%96%D0%BF%D0%B5%D1%80%D1%82%D0%B5%D0%BA%D1%81%D1%82" \o ")