HIRING QUESTIONS

Are you aware that the internship is not remote?	**Yes**
Until when are you registered as a student?	February 2027
In case you are finishing your bachelor's study, are you planning to enroll in a masters after?	Yes. But also subject to the situation as at the time of my graduation
In what timeframe are you available for the internship	**From** 2nd September, 2024 As long as possible
Can you work the entire period or just part of the specified period?	Yes, I can work and am willing to work.
If you are not available during the full period, what would be the preferable period?	No idea yet
Are you a European citizen?	No
Are you able to work full-time?	I can only work part-time as a non-EU citizen (although this can be extended a bit) during the school period and full-time during the holidays

Exercise One

The GitHub link to the task: https://github.com/StephenNnamani/Infineon_Exercises.git

**** Please also go through the README.md file for a bit of my documentation*

Exercise Two

a) What is a Design Pattern?

- It is a general way of using reusable blueprints (solutions) to solve recurring problems in a given context of Software design. Note, it's not a code you can copy from anywhere but a pattern you can replicate.
- It also helps maintain good communication between programmers. They can easily understand the structure of the solution once you tell them the design pattern used. They would know which class does what and in which part of the program something is to be found.
- **Analogy:** Design patterns are not algorithms although they both describe a solution. An algorithm is like a navigator with both a map of a city and instructions on the fastest possible ways to reach a destination. A *Design Pattern is like just a map of the city without instructions.*
- It also helps in reducing unnecessary errors and speeding up the process of solving a problem that already has a similar solution (Design Pattern).
- They are mostly classified into three main categories: **Creational, Behavioral, and Structural.** But, others are also a blend of two or more categories and may not be listed in any of the main three categories.

b) What is MVC and what are the use cases?

MVC which means *Model View Controller* is a design. The MVC design pattern is used to create loosely coupled applications where the model, view, and controller are independent of each other in terms of responsibility. The model or business logic is responsible for interacting with the database by receiving and processing data, and persisting data in the database. The controller is an intermediate between the model (data) and the display (view). The view receives requests from the user (either to input or output data), and then the view sends the action to the controller which interacts with

the business logic (model) to retrieve or persist data (this job of retrieving and persisting data is done by the business logic not controller).

Use Cases:

Building A Student Result page

- The business logic (model) helps to store the students as objects with their details e.g. names, list of the subjects offered by each student object, their exam scores, project scores, and assignment scores.
- The controller interacts with the business logic (model) to get the details, and in some cases calculates the total score, and then sends the data to the appropriate view.
- The view receives these data from the controller and displays it to the user in many possible formats e.g. chart, listed scores, etc.

Building A Hospital Management Application

- The business logic (model) helps to store the patients, nurses, and doctors as objects with their details e.g. names, the relationship between the objects (which doctor is in charge of which patient), their health history, which drugs had been administered before and the dosage given, and the observation from both doctors and nurses, etc.
- The controller interacts with the business logic (model) to get the details, and in some cases calculates the effects of a particular drug on the number of patients who got treated with it, then sends the data to the appropriate view.
- The view receives these data from the controller and displays it to the Doctors for proper decision-making and sometimes to the patients in many possible formats e.g., chart, listed scores, etc.

c) Three other design patterns

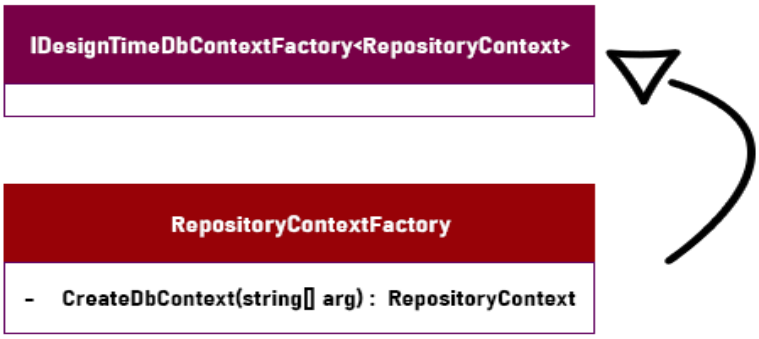
✓ **Factory method:** I used this to create the DBContext in my CompanyEmployee project on my github.

✓ **Dependency injection:** In the same project, I used dependency injection in the service classes of both CompanyService and EmployeeService where I created a

private read-only instance attribute of IRepositoryManager without instantiating it with the new keyword

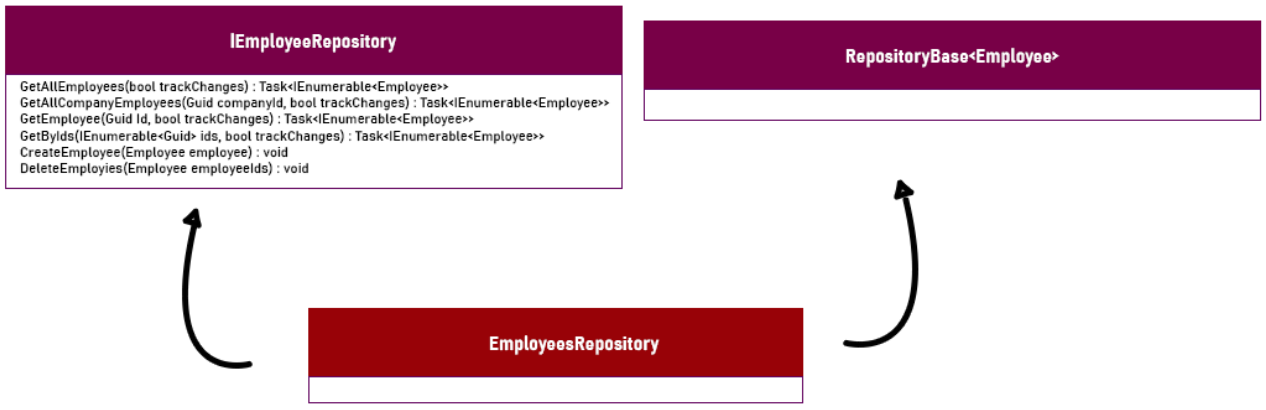
- ✓ **Repository pattern:** I used this pattern in the same projects where I created for example an interface (IRepositoryBase that predefined some methods using generics<T> which any class that implements it can use)

Factory Method



Dependency Injection

Repository Pattern



Exercise Three

B) Are you able to directly create a new instance of *ObjectA*? Please explain your answer.

Yes, we can create a new instance of Class A because it is neither a STATIC nor ABSTRACT class

C) Given an instance of ObjectC, are you able to call the method PrintMessage defined in ObjectB? Please explain your answer.

Yes, we can call the method PrintMessage() because it is a child class of ObjectB. Infact, we can get the property Name, and also the method in ObjectA because in inheritance, a class acquires all the properties, attributes and methods of all the classes it inherited from (But here I was refering to the grandparent of ObjectC).

D) **Inheritance:** is a fundamental concept of object-oriented programming (OOP) that allows a class to inherit properties and methods from another class. This promotes code reusability and establishes a hierarchical relationship between classes. Here ObjectA is the parent class of ObjectB and ObjectD which are the child classes. ObjectC is a child class of ObjectB and ObjectC has all the properties and methods from ObjectA and ObjectB it inherited from.

Method: A method (Actionable block of code) is a block of code that performs a specific task, defined within a class or an object, and can be called upon to execute that task when needed.

Instantiation: Instantiation is the process of creating a specific instance of an object from a class in object-oriented programming. This involves allocating memory for the object and initializing its attributes.

Access Modifiers: Access modifiers in C# define the visibility and accessibility of classes, methods, and other members. They determine where a particular piece of code can be accessed from.

Exercise Four

A) Important Structures or Methods to Cope with Expansions

1. Modular Design:

   Separation of Concerns: Divide the system into distinct sections where each module handles a specific aspect of functionality. For example, separate modules for instrument control, test sequence execution, and data interface management.

   Plug-in Architecture: Use a plug-in architecture where new instruments or interfaces can be added as plug-ins without modifying the core framework.

2. Design Patterns:

   Factory Pattern: Create objects without specifying the exact class of object that will be created. This is useful for creating different instrument controls.

   Repository Pattern: Define a family of repository implementations, encapsulate each one, and make them interchangeable. This allows the repository implementation to vary independently from clients that use it

3. Interfaces and Abstraction:

   Define interfaces for instrument controls, data interfaces, and test sequences. This allows different implementations to be swapped in and out.

   Use abstract classes or interfaces to define common behavior and ensure a consistent API.

4. Configuration Management:

   Use configuration files (e.g., JSON, XML) to manage settings and parameters for different instruments and interfaces. This allows easy updates and additions without changing the codebase.

5. Dependency Injection:

- Implement dependency injection to manage dependencies between different modules. This enhances testability and flexibility.

B) Methods to Make It Easier for Test Engineers to Compose Test Sequences

1. Template-Based Approach:

Provide predefined templates for common test sequences. Engineers can use these templates as starting points and customize them as needed.

2. Unit Testing:

Enable unit testing capabilities (e.g., using frameworks like NUnit, MSTest) to allow engineers to write tests for their code. This can provide flexibility and power to the test validation process.

3. Validation and Debugging Tools:

Integrate tools for validating and debugging test sequences within the framework. This helps engineers identify and fix issues early in the development process.

C) Techniques to Develop the Program

1. Programming Language: **C#** is Suitable for Windows-based applications with a rich set of libraries for GUI development (e.g., WPF, WinForms) and strong support for object-oriented programming.

2. Frameworks: .NET Framework/Core

3. Tools: **IDE:** Visual Studio (for C#)

4. Version Control: Git

4. Approach:

Agile Development: Use Agile methodologies to iteratively develop and improve the framework. This allows for frequent reassessment and adaptation to new requirements.

Test-Driven Development (TDD): Write tests before the actual implementation to ensure robust and reliable code.

`