Adacta

1. Are you familiar with object-oriented concepts?:
    - Yes, these are:
        - Abstraction
        - Encapsulation
        - Inheritance
        - Polymorphism
2. What is the difference between a class and an object?
   - A class is used as a blueprint for an object. It describes what properties and methods the object will contain. Object is a data structure which can contain states (Properties) and behaviour (Methods).
3. What are properties and methods?
   - Properties and methods are members of a class. Properties are variables inside of a class, which can be read or changed with the use of methods getter and setter. Methods are code blocks inside of a class which when called will perform a desired function such as returning a type, printing out something etc. Every instance of an object has its own properties and methods.
4. Visibility modifiers ?

   A warning! I know that a lot of languages have other modifiers so I am only using the most typical ones.
    - With visibility modifiers you can determine which class can use a specific method, variable, class or interface. In most OOP language we know 4 types of accessors (default, public, private, protected)
    - default it is set when we don't give it any accessor (int example;) all default members are visible to all classes
    - Public, any other class can access it (public int example;).
    - Private can only be accessed inside its class.
    - Protected, can only be accessed inside its class or a class that is derived (Inherited) from that class
5. Are you familiar with inheritance? Abstract classes and interfaces. What is the difference? When would you use one over the other? Why?
    - Abstract classes: Is a class that usually can't be instantiated but can only be used to derive other classes from it. Abstract class can have properties, implemented methods and abstract methods. Abstract methods, don't have a body but demand that all classes derived from an abstract class to implement their own version of the required method. We use Abstract classes when we need to implement
    - Interface automatically assumes that all methods inside of it are abstract, it can't have its own implemented methods or properties.
    - Abstract class is used when we want our classes to also use abstract classes properties and implemented methods. Interface is used when we need abstract methods that can be shared between classes.
6. What are ways to write object oriented JavaScript? For example, explain how inheritance works.:

There are 4 ways to write object oriented JavaScript:

- Encapsulation restricts access to most of the objects components, only showing the behaviour of the code.
- Abstraction hides implementation parts and shows only the functional details to the users.
- Inheritance (read bellow)
- Polymorphism allows us to use a single method on multiple objects.

Inheritance:

- Inheritance in OOP means basing a class (sub-class) on another class (super class). With inheritance the sub-class will retain the super classes methods and properties on top of having his own unique properties. In Javascript we can use the prototype object to use with what is called prototypical inheritance;. We can also use the Extend keyword like more "classic" object oriented languages (Java, C# etc.). While the prototype and Extend have the same result (inheriting the methods and properties), the way we implement them is different. Extend is used on classes, while prototype is used on functions.

7. Explain how .this works in javascript in JS. Illustrate by examples some situations where .this can be used
   - .this is an object that is executing the current function, when a function is inside of a class (method) the .this keyword is referencing the current object (owner). If the function is outside of a class then .this is referencing a global object in node and window object in browsers.
   - Some of the possible uses:
     - Accessing the values of a class object
     - Console logging in the window object
     - Used together with call() method we can use this to bind an object to a function. When doing this we change the property value, similar to a class .this

       Example:

       ```
       let setValue= function() {
         this.value= value;
       }
       let user= {
       value: 60;
       }

       setValue.call(user);
       //Now we can change the users value
       ```

     - Using bind(), similar as call() method, with the difference that call() calls the method directly while bind returns a brand new function to invoke.
8. Describe what are the differences between arrays, lists and associative arrays (hash tables).
   In which scenarios would you use each of them and why?

Arrays:

- In OOP arrays are used when we need to store multiple single type data in a container with a predetermined length. To use one you need to determine how many elements it will store (length) and what data type you want it to store.

Lists:

- Lists store multiple single type elements but we do not need too determine its length since the list adapts the length for us. We use it when we have intent to delete and add multiple elements during our operation since its much easier and less memory intense than to do it on an array.

Associative arrays:

- Uses a collection of a value and an unique key to store it in, to access the value we need to call the key to get it. Associative arrays are best used when we need a fast access to a specific value and when we need too check if a certain key exists or not.

PROGRAMMING

3. Is your implementation safe for use in a multithreaded environment?

- No

4. Why/Why not?

- There is no guarantee when each thread is gonna start executing itself. If multiple threads are using the same array this can lead to problems.

5. How would you make this structure thread-safe?

- By using mutex lock, which would help us force the threads too wait for the current thread to finish executing itself.

6. What is the time complexity (big O) of linked list.

- Singly linked list will usually have a time complexity of 0(n) since we will almost always need to iterate through the list. The 2 exceptions are if we need too access the first element (Head) or last element (Tail) which will be n(1).

1. How would you approach designing a simple registry of persons for a country (only natural persons)
   - I would use a relational tabel which I would build in the following way.

     The row:

     Unique-key | First Name | Last Name | Birthdate | City_residence
2. How would you store the data, what data model would you use?
   - In a database, I would use relational mode because its way of storing data (Tables: Rows and Columns) fits our need the best. We could also use our City_residence to enable as a traversal through other tables to for instance find how many people live in a certain city.
3. What architecture would you use?
   - Microservices
     - Because all programs are independent from each other (they communicate through API-s) the scalability and further development would be much easier in the future.

- With changing and adding of additional services our chances of getting problems would be smaller.
- Switching technology would be much easier.

4. Which clients (UI) would you support and how?

- I would use React, since I have the most experience with this library. I would create a single page website on the internet with an application form where we would be able to enter and submit all the required data, the website would also have a display function for all the users and a "filter by name" option and a "sort by creation date" and "sort by last name" options.

5. Which technologies would you use to implement the registry and why those technologies.

- Front-end: ReactJs, I have the most experience with it, and despite its simplicity its one of the best libraries in JS, with its usage of components, mobile support etc.
- Back-end: Ruby on Rails, I have the most experience with Ruby language, with an reliable, easy to understand and interesting syntax. It is easy and fast to set up.
- Database: SQL