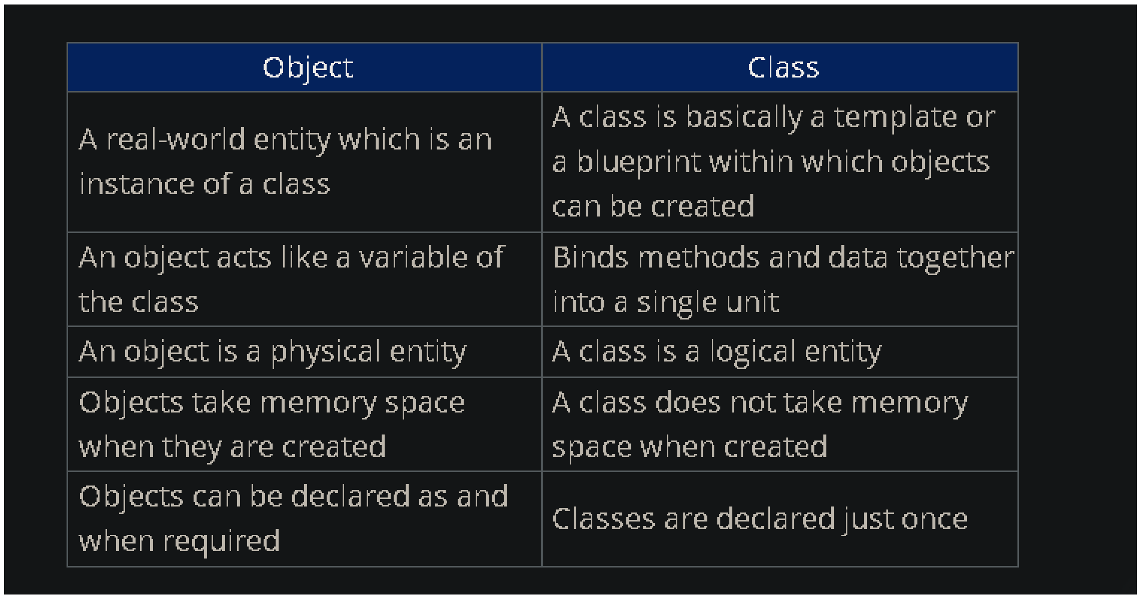
Interview Intership Checklist

Archive	☐
Resources	CS
Tags	

Things that I gotta know:

- OOP
  - type of programming based on objects rather then just functions and procedures
  - classes and objects
    - objects are entities that have different states or attributes, and behaviors
    - classes are the prototype/blueprint for those objects



Object	Class
A real-world entity which is an instance of a class	A class is basically a template or a blueprint within which objects can be created
An object acts like a variable of the class	Binds methods and data together into a single unit
An object is a physical entity	A class is a logical entity
Objects take memory space when they are created	A class does not take memory space when created
Objects can be declared as and when required	Classes are declared just once

  - Encapsulation
    - biding data and code together
    - setters and getters

            - access modifiers(public, private, protected)
        - Data Abstractisation
            - hides the implementation details
            - abstract classes
                - abstract methods
                    - method declared but not defined
                - virtual/pure virtual functions
            - interfaces
                -
        - Inheritence
            - allows classes to inherit common properties from other classes
            - mutiple(not in java), multilevel, hybrid, single
        - Polymorphism
            - ablity to exist in multiple forms, for ex, multiple definition can be given to a single interface
            - static : early binding at compile time, method overloading
            - dynamic: late binding at runtime, method overriding
        - exceptions
            - try/catch block
            - finally always runs
- SOLID
    - design principles for oop
        - single responsability
            - applied to classes, microservices, software components
            - each class can only do one thing and be responsible for only one function

        - open closed
            - closed for modifications
            - open for extension
            - modifying causes code-related issues
        - liskov substitution
            - you can replace the parent class in a software program with its derived class subtype instances and it does not modify the prevailing efficiency or accuracy of the software program
        - interface segregation principle
            - separate client-specific interfaces
            - no client should be forced to depend on interfaces they don’t use
        - dependency inversion
            - high level modules should be easily reusable and unaffected by changes in low-level modules
            - low and high-level modules ought to depend on abstractions
            - abstractions should not depend on details, details should depend on abstractions
    - benefits
        - efficient software design
        - makes possible to change specific software areas without adversely impacting the functionality of the rest
        - write code that is logical, easy to read, convenient to maintain and possible to extend.
- Design Patterns
    - creational patterns
        - singleton
            - solves two problems at the same time

            - lets you ensure that a class has only one instance, while providing global access point to this instance
            - solution
                - make the default constructor prive
                - create a static creation method that acts as a constructor, this method calls the private constructor and saves it in a static field, all the following calls to this method return the cached object
        - factory
            - provides an interface for creating objects in a superclass, but allows subclasses to alter the type of objects that will be created
        - abstract factory
            - produce families of related objects without specifying their concrete classes
    - behavioural patterns
        - iterator
            - lets you traverse elements of a collection without exposing its underlying representation(list, stack, tree, etc)
        - observer
            - a subscription mechanism to notify multiple objects about any events that happen to the object they’re observing
- Java

MAP Lecture Notes

- C++
- JavaScript
    - hoisting
    -

- CSS
- HTML
- Data Structures
    - array/dynamic array
    - linked lists: sll, dll, xor, skip
    - heap
    - set
    - binary tree, bst/avl - treemap(red-black tree in java.util)
    - hashtable: separate/coalecent chaining, open addressing(linear probing, quadratic probing), double hashing, cuckoo hashing
    - map
    - matrix: sparred - triples,compressed line/column
    - stack
    - queue/priority queue - binary heap
    - graphs
- Algorithms
    - sorting algorithms (Bubble, Insertion, Heap, Quick, Merge, Counting, Radix,Bucket)
    - Binary Search
    - Greedy algorihms
    - divide and conquer
    - Dyamic Programming
    - Graphs(BFS, DFS)
    - Backtracking
- Agile/Scrum
- Java Spring

- React
    - hooks
        - only work at the beggining of the component, except with custom hooks

    ```
    //useState
    const [count, setCount] = useState(0)
    //useEffect
    //3 states: Mount, Cycle of react, unmount
    ```

    - based on components , functions, the return value is the html
    - jsx, combined html with js
- SQL

    Databases Lecture Notes