INTERVIEW QUESTIONS

BASICS

Classes, Objects…

A **class** is a blueprint which you use to create objects.

An **object** is an instance of a class - it's a concrete 'thing' that you made using a specific class

OOP Principles

- **Encapsulation**

Information hiding -> Private properties and public methods to access.

Hiding the details of the object and providing a decent interface for the entities in outer world to interact with that object or entity.

Can validate in these access methods. (Control of properties without outside intervention)

- **Abstraction**

Expose only what's required rather than modeling unnecessary unused details. Outsider can only see restricted content. **Enforce whats required.**

Abstraction means to show only the necessary details to the client of the object… expose only the details which are concern with the user (client) of your object…

Can be used to **hide the implementation.** (Focuses on functionality rather than implementation -> What it should do than how to do that)

- **Inheritance**

Inherit common implementations from parent classes -> avoid redundancy.

IS-A relationship, also known as parent-child relationship

Abstract classes -> With at least one abstract methods. extends
Interfaces -> All methods should be implemented by child. Implements

Can implement from many interfaces, but usually can extend only one abstract class. (Some languages support multiple inheritance)

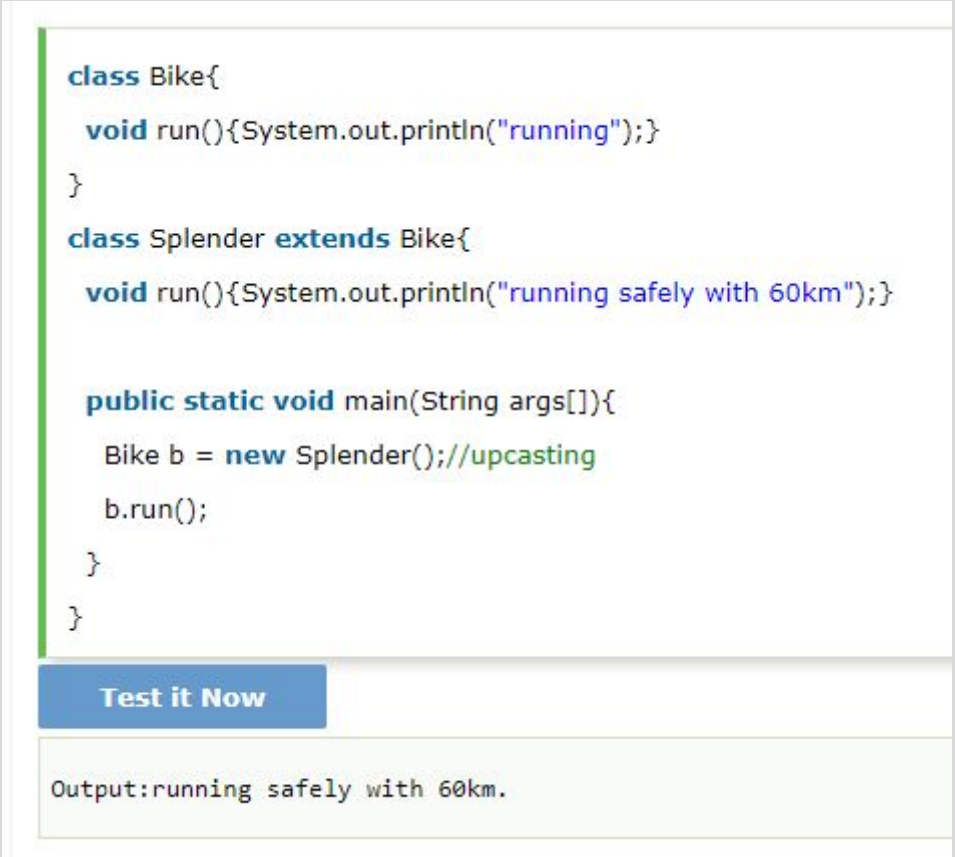
- **Polymorphism**

Take many forms according to context.-> to create generic functionality
Used implementation can be decided on runtime depending on data type.

Method overloading -> static polymorphism
Function overloading vs Operator overloading

Method overriding -> dynamic polymorphism

Runtime polymorphism



```
class Bike{
  void run(){System.out.println("running");}
}
class Splender extends Bike{
  void run(){System.out.println("running safely with 60km");}

  public static void main(String args[]){
    Bike b = new Splender();//upcasting
    b.run();
  }
}
```

Test it Now

```
Output:running safely with 60km.
```

Encapsulation vs Abstraction?
https://stackoverflow.com/questions/742341/difference-between-abstraction-and-encapsulation
https://stackoverflow.com/questions/25029465/whats-the-difference-between-abstraction-and-encapsulation

Interface vs Abstract Class?
https://stackoverflow.com/questions/761194/interface-vs-abstract-class-general-oo

https://stackoverflow.com/questions/1913098/what-is-the-difference-between-an-interface-and-abstract-class
https://beginnersbook.com/2013/05/abstract-class-vs-interface-in-java/

Interface variables? Public static final?
Accessors: get; set;
Access modifiers: public, private, protected (also subclasses)
abstract func vs virtual func (can have implementation)

Static keywords: Static variables? Static classes? Static methods?
static members belong to the class instead of a specific instance.

https://stackoverflow.com/questions/413898/what-does-the-static-keyword-do-in-a-class

https://anampiu.github.io/blog/OOP-principles/
https://www.tutorialspoint.com/csharp/csharp_polymorphism.htm
https://beginnersbook.com/2013/03/oops-in-java-encapsulation-inheritance-polymorphism-abstraction/

SOLID Principles

- **Single-responsibility principle**

A class should have one, and only one, reason to change.

- **Open-closed principle**

You should be able to extend a classes behavior, without modifying it.

- **Liskov substitution principle**

Derived classes must be substitutable for their base classes.

- **Interface segregation principle**

Make fine grained interfaces that are client specific.

- **Dependency Inversion Principle**

Depend on abstractions, not on concretions.

https://scotch.io/bar-talk/s-o-l-i-d-the-first-five-principles-of-object-oriented-design
https://stackoverflow.com/questions/98695/design-principles

Design Patterns

Creational | Behavioral | Structural

Singleton

lazy initialization is the tactic of delaying the creation of an object, the calculation of a value, or some other expensive process until the first time it is needed.

https://stackoverflow.com/questions/11165852/java-singleton-and-synchronization
Just because getInstance() is thread safe, that does not mean any other methods of the class are thread safe. Multiple threads can still perform operations on the singleton object simultaneously.

Singleton Vs Static Classes
singletons can implement interfaces

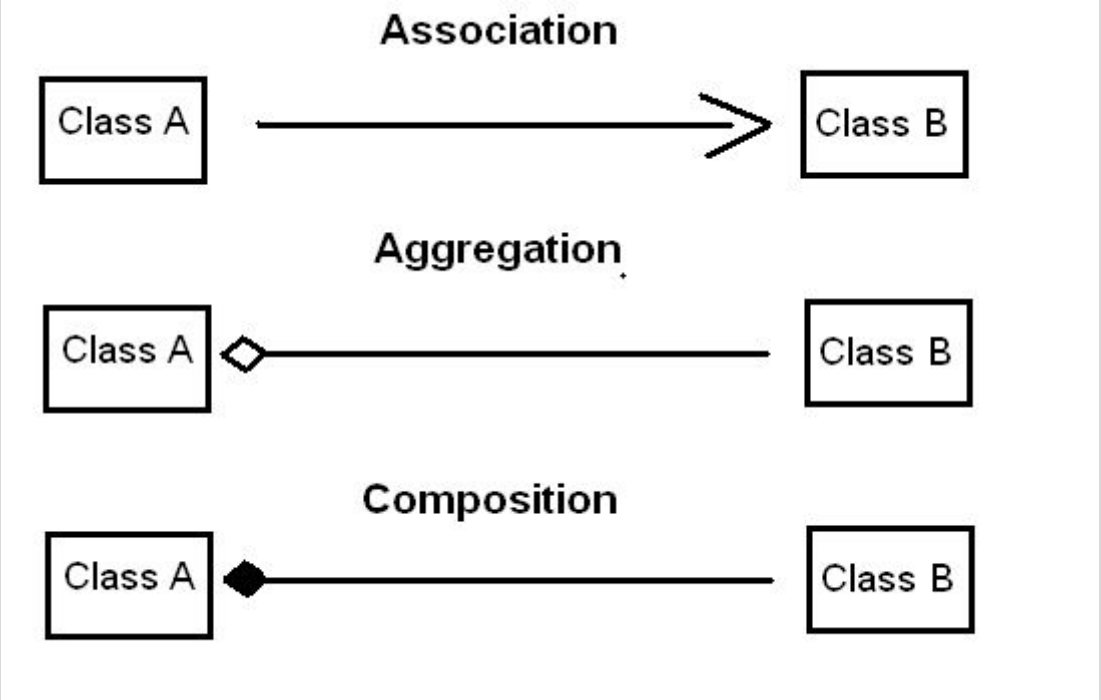
Singleton instance can be passed as a parameter to another method

https://sourcemaking.com/design_patterns
http://www.oodesign.com

https://i.pinimg.com/originals/e4/fb/b0/e4fbb07b8f2858e4f0b82a9f11f9ef21.jpg
http://www.celinio.net/techblog/wp-content/uploads/2009/09/designpatterns2.jpg

UML

Association | Aggregation | Composition
https://www.tutorialspoint.com/object_oriented_analysis_design/index.htm



Association link, states that there is dependency between classes.

Aggregation link, instance can also be aggregated by other classes in the application (therefore aggregation is also known as shared association)

Composition link, in addition to the part-of relationship between ClassA and ClassB - there's a strong lifecycle dependency between the two, meaning that when ClassA is deleted then ClassB is also deleted as a result

ER Diagram? SQL?

Superkey Candidate Key Primary Key Foreign Key

Expression: Selection | Projection | Natural Join | Union | Product

Practises

TDD vs BDD

Agile vs Waterfall
Agile -> Scrum vs Kanban vs XP?

CS Modules

CS 2022 Data Structures & Algorithms

- Complexity Analysis
- Recursion
- Searching
- Sorting -> Insertion | Bubble | Merge | Heap

	Time Complexity					
	Best	Worst	Avg.	Space	Stable	Comments
Bubble Sort	O(n^2)	O(n^2)	O(n^2)	O(1)	Yes	*For each pair of indices, swap the elements if they are out of order*
Modified Bubble Sort	O(n)	O(n^2)	O(n^2)	O(1)	Yes	*At each Pass check if the Array is already sorted. Best Case-Array Already sorted*
Selection Sort	O(n^2)	O(n^2)	O(n^2)	O(1)	Yes	*Swap happens only when once in a Single pass*
Insertion Sort	O(n)	O(n^2)	O(n^2)	O(1)	Yes	*Very small constant factor even if the complexity is O(n^2).* ***Best Case:*** *Array already sorted* ***Worst Case:*** *sorted in reverse order*
Quick Sort	O(n.lg(n))	O(n^2)	O(n.lg(n))	O(1)	Yes	*Best Case: when pivot divide in 2 equal halves Worst Case: Array already sorted - 1/n-1 partition*
Randomized Quick Sort	O(n.lg(n))	O(n.lg(n))	O(n.lg(n))	O(1)	Yes	*Pivot chosen randomly*
Merge Sort	O(n.lg(n))	O(n.lg(n))	O(n.lg(n))	O(n)	Yes	*Best to sort linked-list (constant extra space). Best for very large number of elements which cannot fit in memory (External sorting)*
Heap Sort	O(n.lg(n))	O(n.lg(n))	O(n.lg(n))	O(1)	No	

- Basic Algorithms -> Divide & Conquer, Greedy, Dynamic
- Data Structures -> Array, Linked List, Queues, Stacks, Sets. Trees, Hash Tables, Graphs

Data Structure Operations

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	O(1)	O(n)	O(n)	O(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	-	O(1)	O(1)	O(1)	-	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	-	O(log(n))	O(log(n))	O(log(n))	-	O(n)	O(n)	O(n)	O(n)
B-Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	-	O(log(n))	O(log(n))	O(log(n))	-	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)

- NP-Completeness

CS 2052 Computer Architecture

- CPU + Memory + I/O
- ALU, Registers

https://docs.google.com/presentation/d/1wG3ItT4NUk66UEDuC5P89lg0UX453DUWtC1C5bjn880/edit#slide=id.p

CS 2012 Principles of OOP

CS 2042 Operating Systems

CS #### ????????

CS2062 OOSD
CS2032 Computer Communication
CS3022 Software Engineering
CS3032 Computer Networks
CS3042 Database Systems
CS3052 Computer Security
CS3062 Theory of Computing

C# and .NET

Data Structures - http://www.vcskicks.com/csharp_data_structures.php
Array, ArrayList, List, LinkedList, Dictionary, HashSet, Stack, Queue

Threading http://www.albahari.com/threading/

```
System.Threading.Thread

Thread th = Thread.CurrentThread;
th.Name = "MainThread";
```

Final keyword

Final variable:	constant
Final method:	cannot override it / inherited but not overridden
Final class:	cannot extends it

Static keyword

Static variable:	common prop of all objects. Allocated at loading
	Common attribute in class not in instances.(memory)
	Common generally accessible by all instances.
Static method:	Belong to class not instance.
	Invoked without creating instance.
	Only access static data
Static Block:	Initialize static data before class loading.

This keyword

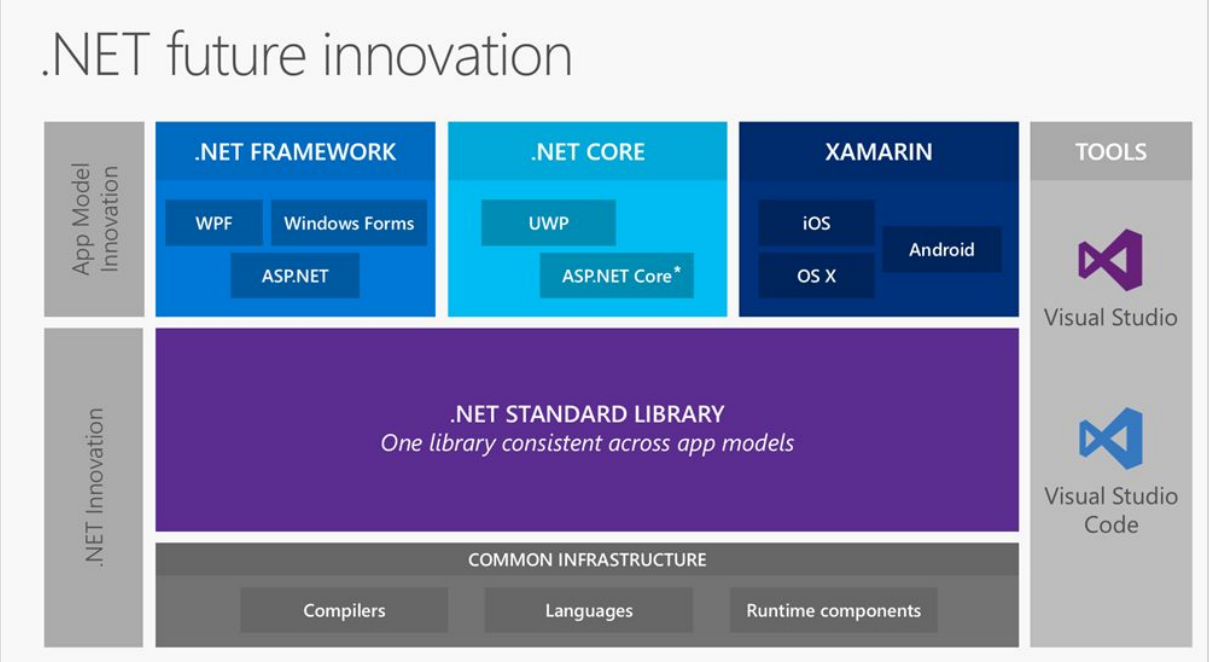
Refers to current object.

```
using System;
using System.Threading;

namespace MultithreadingApplication
{
   class ThreadCreationProgram
   {
      public static void CallToChildThread()
      {
         Console.WriteLine("Child thread starts");
      }

      static void Main(string[] args)
      {
         ThreadStart childref = new ThreadStart(CallToChildThread);
         Console.WriteLine("In Main: Creating the Child thread");
         Thread childThread = new Thread(childref);
         childThread.Start();
         Console.ReadKey();
      }
   }
}
```

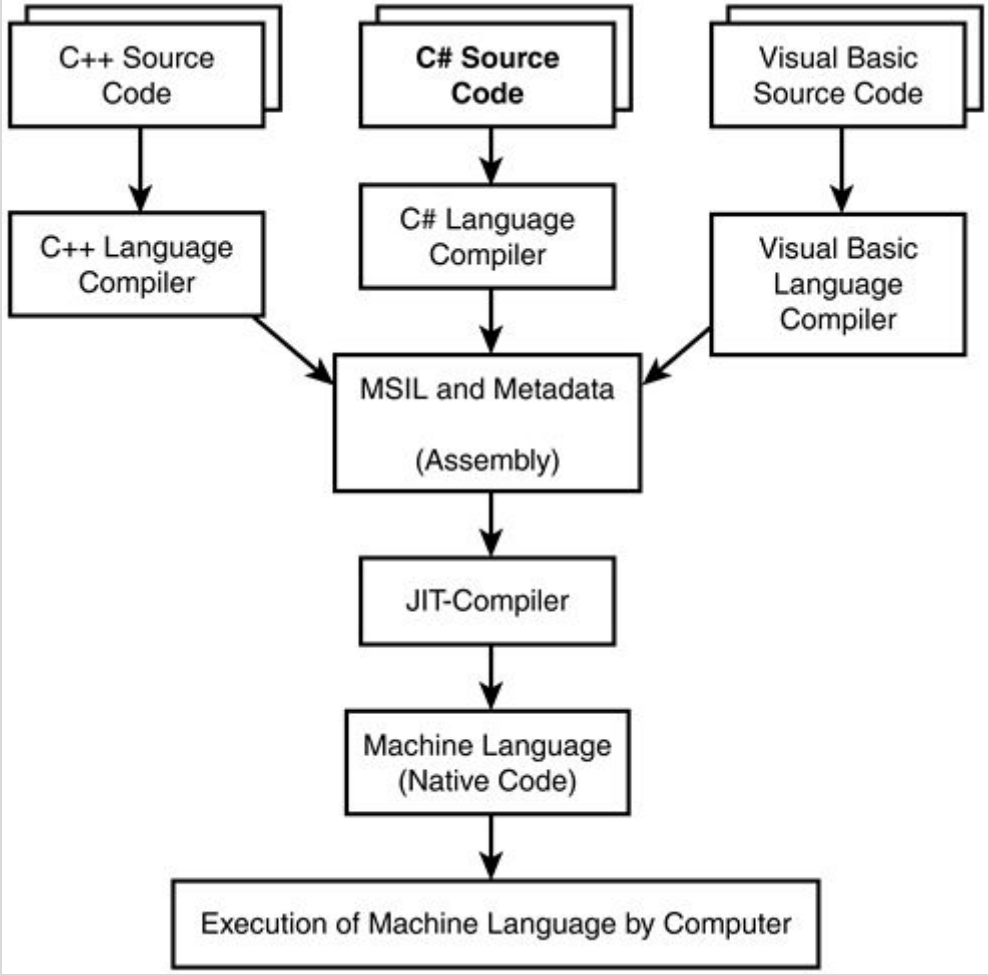
Generics

```
public class MyGenericArray<T>
{
   private T[] array;
   public MyGenericArray(int size)
   {
      array = new T[size + 1];
   }

   public T getItem(int index)
   {
      return array[index];
   }

   public void setItem(int index, T value)
   {
      array[index] = value;
   }
}
```

Architecture? .NET Standard 2.0 | .NET Core | ASP.NET?



Compiler Process? JIT | IL



C# Coding Best Practises Variable naming conventions?

Dates

LINQ

C# 7.0 | .NET 4.7 | .NET/ASP.NET Core 2.0

http://a4academics.com/interview-questions/52-dot-net-interview-questions/417-c-oops-interview-questions-and-answers
Struct vs class
https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/objects
IQueryable vs IEnumerable

Array vs ArrayList/List<T>

Array:	Fixed length	1+ Dimensions	
List:	Add/Remove data	Resizing array is expensive	more fun.

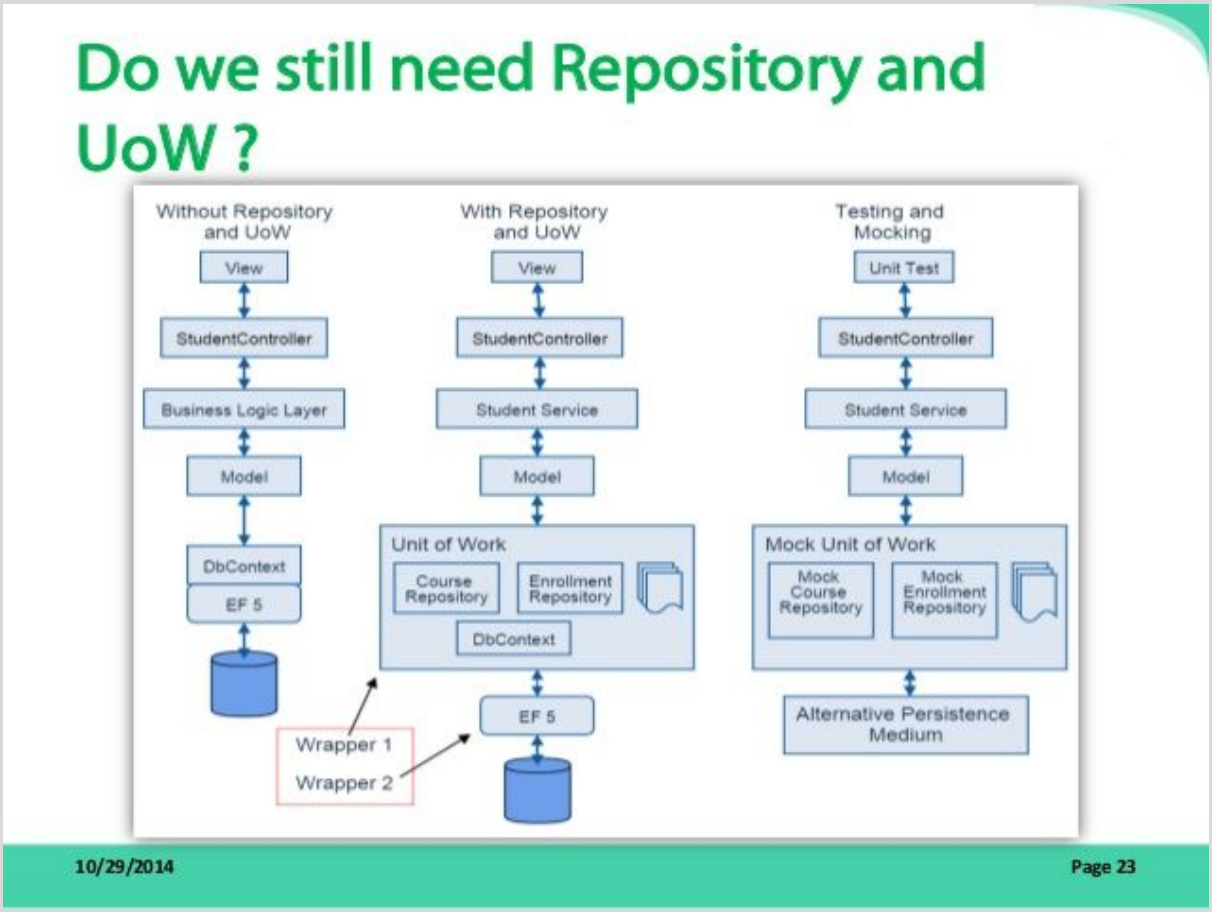
ArrayList vs List<T>

ArrayList:	Without Generic	Object ref.
List:	Generic	Value

https://stackoverflow.com/questions/32020000/what-is-the-difference-between-an-array-arraylist-and-a-list

DDD & CQRS

Repo + UoW ->



DDD ->

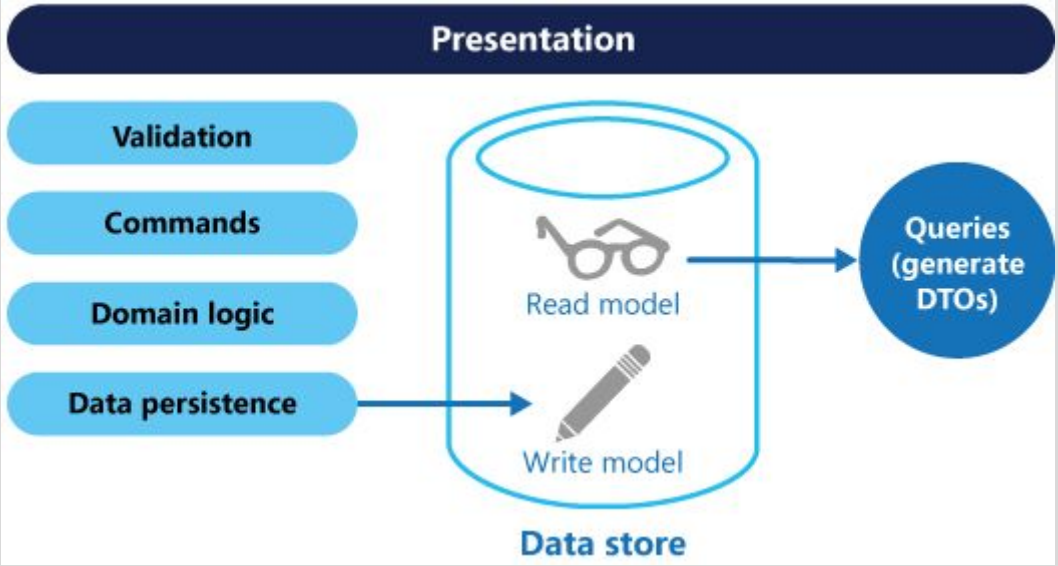
```
Domain Services : Account.Transfer(Acc1, Acc2, Amount);

Not   Account.Deduct()   Account.Increase()
```

CQRS ->
Every method should either be a command that performs an action, or a query that returns data to the caller, but not both.

Command query responsibility segregation (CQRS) applies the CQS principle by using separate Query and Command objects to retrieve and modify data, respectively

it is safely usable only in single-threaded applications.

https://en.wikipedia.org/wiki/Command–query_separation
https://docs.microsoft.com/en-us/azure/architecture/patterns/cqrs



- Often reading data is much more frequent than writing.
- Reading data we typically retrieve a larger amount of data or lists of data compared to writing that should affect one aggregate only.
- Reads from a user perspective has to be more performant than writes. User tends to find it easier to accept a slower response when data is changed.

YouTube

CQRS and Event Sourcing in C# and .NET
6. Introduction to CQRS - Event Sourcing, Distributed Systems & CQRS

JavaScript

Basics / Advanced

```
Data Types
      Number, String, Boolean, Function, Object, Undefined

      boolean, null, undefined, number, string, and object *
```

https://developer.mozilla.org/en-US/
https://javascript.info
https://johnresig.com/apps/learn/
http://htmldog.com/guides/javascript/advanced/

https://blog.sessionstack.com/how-does-javascript-actually-work-part-1-b0bacc073cf
https://medium.com/dailyjs/understanding-v8s-bytecode-317d46c94775

Object Oriented JS

Prototypal Inheritance
Instances inherit directly from other objects vs class inheritance?

```
var obj1 = {};
var obj2 = new Object();
var obj3 = Object.create(null);
```

```
var car1 = {
    color: 'red',
    make:'Toyota',
    model:'Sedan',
    getInfo: function (){
        console.log( this );
    }
};
```

Object.prototype is on the top of the prototype chain.
Object.values Object.entries

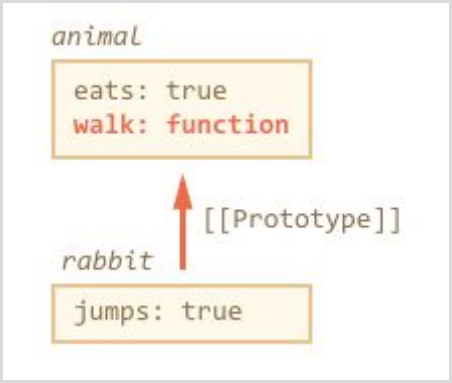
```
/* Create a object by function */

var Car = function(color, make, model, getInfo ) {
    this.color='';
    this.make='';
    this.model='';
    this.getInfo= function( time ){
            console.log( this );
    };
};

var car1 = new Car('red','Toyota','Sedan');
```

```
var Car = function(color, make, model ) {
    this.color='';
    this.make='';
    this.model='';
};
Car.prototype = {
    getInfo : function( ){
            console.log( this );
    }
};
```

```
/* Child class or sub-class */
function Dog()  {};

/* Inheritance */
Dog.prototype = new Pet();
Dog.prototype.species = "Dog";

var dog1 = Object.create(new Dog());
dog1.setName ( "Polly");
```

Class in ES6 is a specific function

```
let animal = {
  eats: true
};
let rabbit = {
  jumps: true
};

rabbit.__proto__ = animal;
```



No matter where the method is found: in an object or its prototype. In a method call, this is always the object before the dot.

https://medium.com/@harryho2/an-introduction-of-oo-javascript-31f1c6ab7058
https://javascript.info/prototype-inheritance
https://codeburst.io/javascripts-new-keyword-explained-as-simply-as-possible-fec0d87b2741

RxJS ReactiveX Angular2: Observables

Functional JS

first class func lambdas

Closures

```
function init() {
    var name = "Mozilla"; // name is a local variable created by init
    function displayName() { // displayName() is the inner function, a closure
        alert (name); // displayName() uses variable declared in the parent function
    }
    displayName();
}
init();
```

https://medium.com/javascript-scene/10-interview-questions-every-javascript-developer-should-know-6fa6bdf5ad95

ES6

Transpiler ES6/TypeScript -> Vanilla JS (ES5) Babel

const and let

```
{
  var x = 100
  let y = 200
  const z = 300
  console.log('x in block scope is', x)
  console.log('y in block scope is', y)
  console.log('z in block scope is', z)
}
```

Array helper func

forEach | map | filter | find | every | some | reduce

Arrow func

```
const sum = (acc, value) => acc + value
const product = (acc, value) => acc * value
```

```
const sum = (acc, value) => {
  const result = acc + value
  console.log(acc, ' plus ', value, ' is ', result)
  return result
}
```

Classes

```
class Point {
    constructor(x, y) {
        this.x = x
        this.y = y
    }

    toString() {
        return '[X=' + this.x + ', Y=' + this.y + ']'
    }
}

class ColorPoint extends Point {
    static default() {
        return new ColorPoint(0, 0, 'black')
    }

    constructor(x, y, color) {
        super(x, y)
        this.color = color
    }

    toString() {
        return '[X=' + this.x + ', Y=' + this.y + ', color=' + this.color + ']'
    }
}

console.log('The first point is ' + new Point(2, 10))
console.log('The second point is ' + new ColorPoint(2, 10, 'green'))
```

Spread Operators

```
var defaultColors = ['red', 'blue', 'green']
var userDefinedColors = ['yellow', 'orange']

var mergedColors = [...defaultColors, ...userDefinedColors]
```

Object Destructuring

```
function printBasicInfo({firstName, secondName, profession}) {
        console.log(firstName + ' ' + secondName + ' - ' + profession)
}

var person = {
  firstName: 'John',
  secondName: 'Smith',
  age: 33,
  children: 3,
  profession: 'teacher'
}

printBasicInfo(person)
```

Promises

```
var p = new Promise(function(resolve, reject) {

    // Do an async task async task and then...

    if(/* good condition */) {
        resolve('Success!');
    }
    else {
        reject('Failure!');
    }
});

p.then(function() {
    /* do something with the result */
}).catch(function() {
    /* error :( */
})
```

The new Promise() constructor should only be used for legacy async tasks, like usage of setTimeout or XMLHttpRequest.

https://davidwalsh.name/promises
https://ponyfoo.com/articles/es6-promises-in-depth

Generators

Fundamental about your functions: once the function starts running, it will always run to completion before any other JS code can run.

With ES6 generators, we have a different kind of function, which may be paused in the middle, one or many times, and resumed later, allowing other code to run during these paused periods.

(Note: Web Workers are a mechanism where you can spin up a whole separate thread for a part of a JS program to run in, totally in parallel to your main JS program thread.)

```
function *foo() {
    var x = 1 + (yield "foo");
    console.log(x);
}
```

The yield "foo" expression will send the "foo" string value out when pausing the generator function at that point, and whenever (if ever) the generator is restarted, whatever value is sent in will be the result of that expression, which will then get added to 1 and assigned to the x variable.

https://ponyfoo.com/articles/es6-generators-in-depth
https://ponyfoo.com/articles/es6
https://davidwalsh.name/es6-generators

ES7/ES8

Async/Await

```
async function add1(x) {
  const a = await resolveAfter2Seconds(20);
  const b = await resolveAfter2Seconds(30);
  return x + a + b;
}
```

call back func -> promises -> async/await

https://medium.com/@reasoncode/javascript-es8-introducing-async-await-functions-7a471ec7de8a
http://rossboucher.com/await/#/

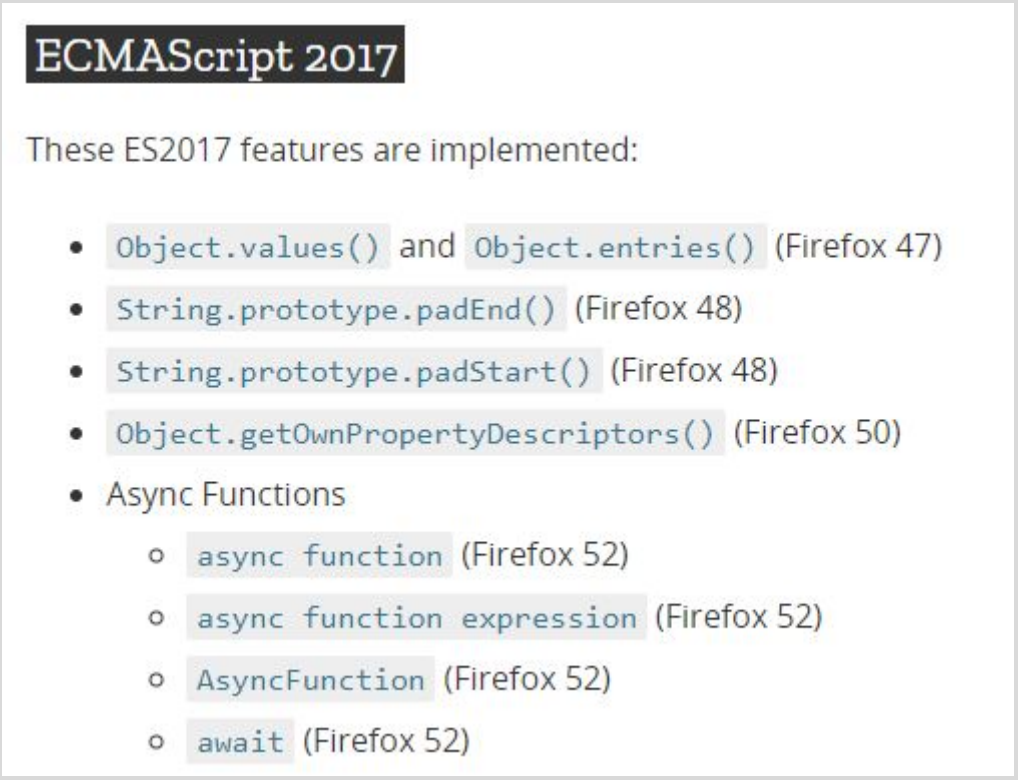
https://blog.pragmatists.com/top-10-es6-features-by-example-80ac878794bb
https://derickbailey.com/2017/06/06/3-features-of-es7-and-beyond-that-you-should-be-using-now/

https://ponyfoo.com/articles/understanding-javascript-async-await
https://medium.com/@reasoncode/javascript-es8-introducing-async-await-functions-7a471ec7de8a

ECMAScript 2016

These ES2016 features are implemented:

- `Array.prototype.includes()` (Firefox 43)
- `TypedArray.prototype.includes()` (Firefox 43)
- Generators and generator methods are no longer constructable (Firefox 43)
- Proxy enumerate handler removed (Firefox 47)
- Exponentiation operator (Firefox 52)
- Rest parameter destructuring (Firefox 52)



React & Redux

WHY React? **React vs Angularjs vs Vue**

WHY Redux? **Redux vs Flux** **Redux vs Mobx**

SPA vs PWA? **MV*?**

```
class HelloMessage extends React.Component {
  render() {
    return (
      <div>
        Hello {this.props.name}
      </div>
    );
  }
}

ReactDOM.render(
  <HelloMessage name="Jane" />,
  mountNode
);
```

```
class Timer extends React.Component {
  constructor(props) {
    super(props);
    this.state = { seconds: 0 };
  }

  tick() {
    this.setState((prevState) => ({
      seconds: prevState.seconds + 1
    }));
  }

  componentDidMount() {
    this.interval = setInterval(() => this.tick(), 1000);
  }
```

JSX? class? Constructor? super(props)? this.props this.setState? render() componentDidMount()

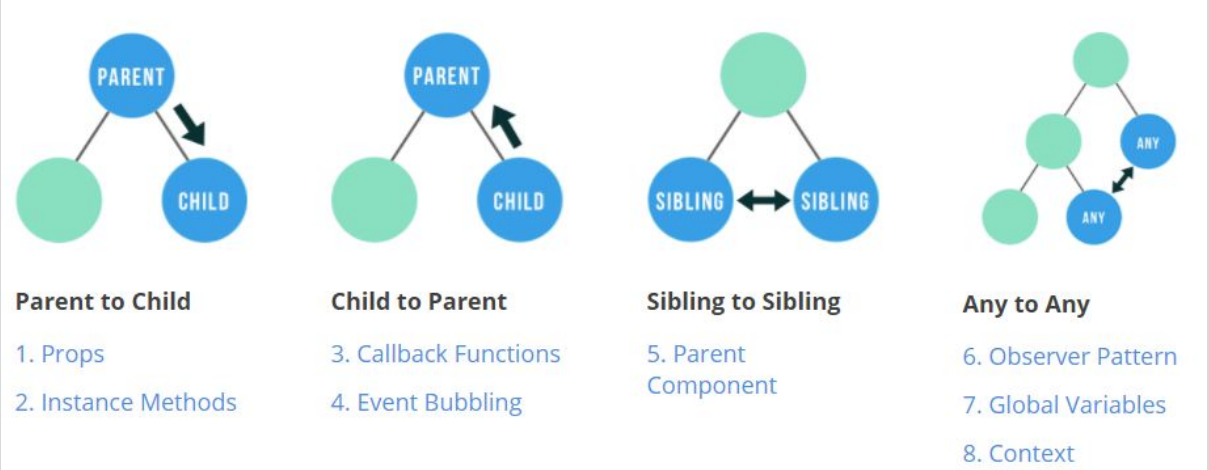
https://medium.com/@harryho2/angular-vs-react-vs-vue-f470f5b74bf6
https://medium.com/unicorn-supplies/angular-vs-react-vs-vue-a-2017-comparison-c5c52d620176

AngularJS TypeScript and React ES6 -> Type Safety? Static Types? : Flow?
Angular 2 to put 'JS' into HTML. React puts 'HTML' into JS.(JSX)
ES6 Syntax import {Component} from 'react'; babel?

Virtual DOM
https://medium.com/@gethylgeorge/how-virtual-dom-and-diffing-works-in-react-6fc805f9f84e
https://stackoverflow.com/questions/24698620/dirty-checking-on-angular

Libraries vs Framework
react, react-dom, react-router, redux-thunk, redux-saga, redux

```
Server side rendering          SEO

State Management & Data Binding
      Redux ->     1. Single Source of Truth |
                   2. State ReadOnly |
                   3. Changes with pure func |
                   Redux state -> immutable JS object          comparison ===

      MVC -> Two way data binding          Angular RxJS?

      React component -> props, state, render(), lifecycle methods
      Redux container -> mapStateToProps, mapDispatchToProps

**Redux Application**     http://redux.js.org/docs/advanced/ExampleRedditAPI.html

Redux vs Flux
      Flux is a fancy name for the observer pattern modified
      With Redux middleware, actions can also be functions and promises
      Flux it is a convention to have multiple stores per application
      Instead of a dispatcher it uses pure functions to alter the state
      Redux is influenced by functional programming (FP) principles
      https://edgecoders.com/the-difference-between-flux-and-redux-71d31b118c1
      https://www.robinwieruch.de/redux-mobx-confusion/

Without Redux
      http://andrewhfarmer.com/component-communication/
```

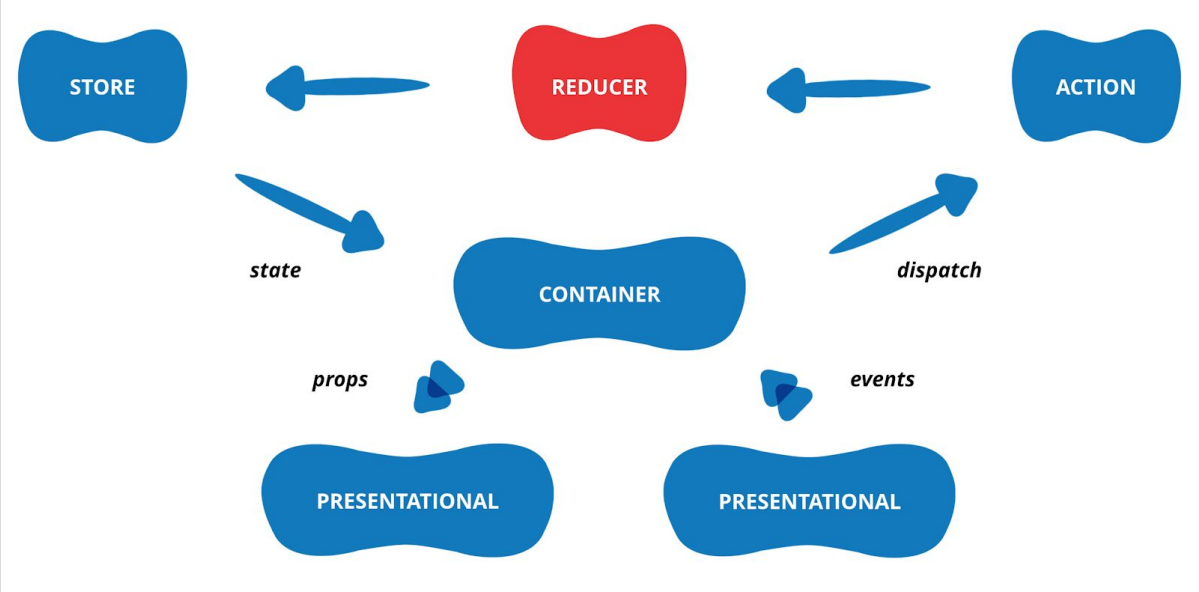


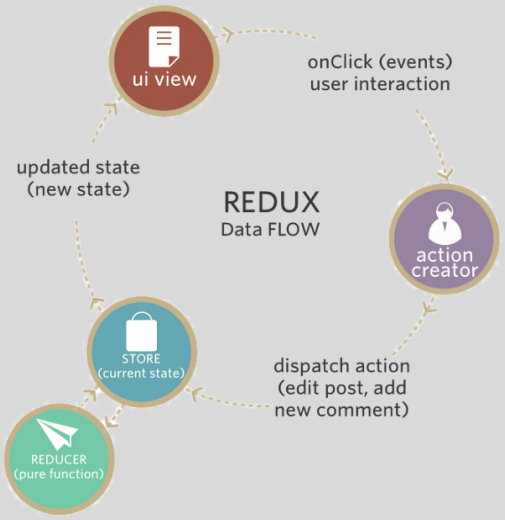
Redux

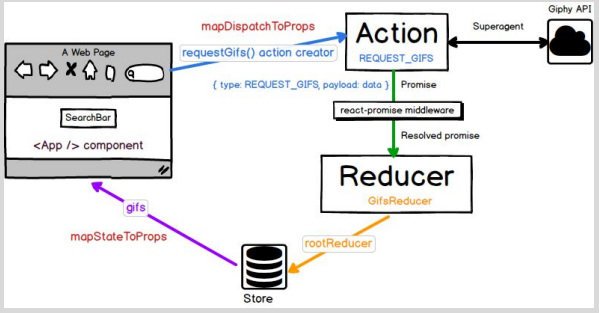
If you want to change state, you have to fire off an action.
State (the store) only has a getter, not setters.

Developer tooling hard to do with Flux
The primary building block of Redux state management is the reducer function.

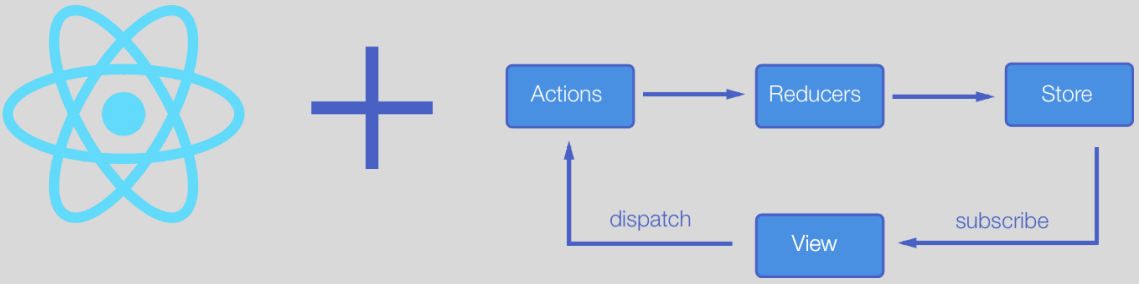
https://medium.com/javascript-scene/10-tips-for-better-redux-architecture-69250425af44





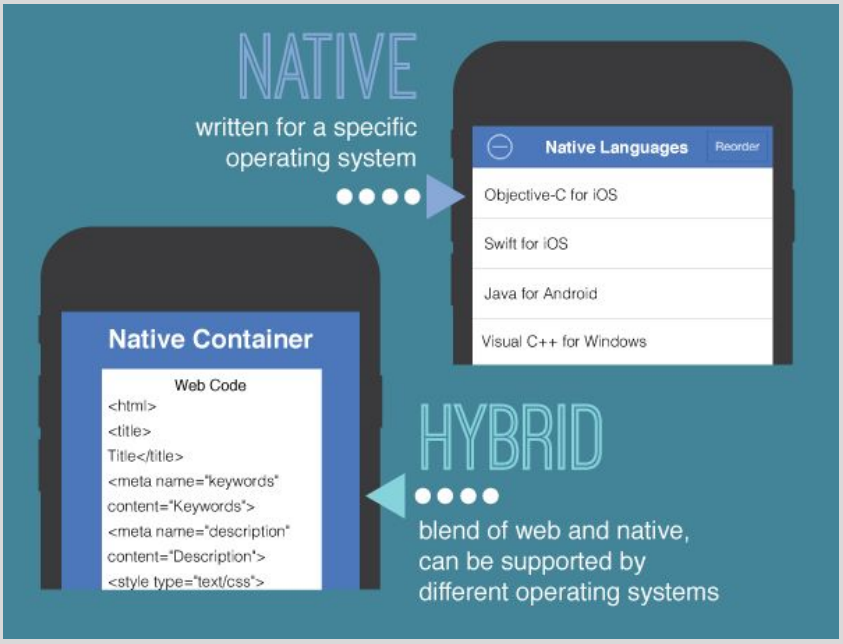


Redux



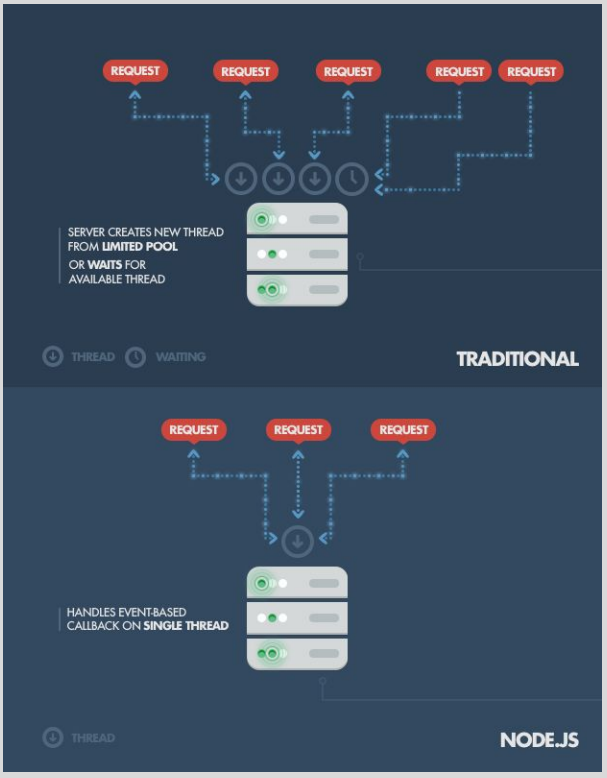
React Native

```
import React, { Component } from 'react';
import { Text, View } from 'react-native';

Hybrid -> Ionic
Native -> React Native
```



Developing in React Native is primarily done with Javascript, which means that most of the code you need to get started can be shared across platforms. However, where hybrid apps render using HTML and CSS, React Native will render using native components.

NodeJS



NPM NVM? Express? Socket.io? Mongodb? Redis?
ALL IN JAVASCRIPT

REAL TIME APPLICATION

I/O Operations Not CPU-heavy jobs Not CRUD?

```
const http = require('http');

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World\n');
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

HTML5

LocalStorage

WebSockets

SPA PWA

JS Q.

Performance: **AngularJS vs React vs Vue NodeJS vs C#/Java/Python**

undefined vs not defined typeof eval this isNaN

== and === parseInt(,) str.reverse() strict new

https://www.thatjsdude.com/interview/js2.html
https://github.com/nishant8BITS/123-Essential-JavaScript-Interview-Question
https://www.toptal.com/javascript/interview-questions

```
var Employee = function (name, company, salary) {
    this.name = name || "";       //Public attribute default value is null
    this.company = company || ""; //Public attribute default value is null
    this.salary = salary || 5000; //Public attribute default value is null

    // Private method
    var increaseSalary = function () {
        this.salary = this.salary + 1000;
    };

    // Public method
    this.dispalyIncreasedSalary = function() {
        increaseSlary();
        console.log(this.salary);
    };
};

// Create Employee class object
var emp1 = new Employee("John","Pluto",3000);
```

Database

SQL -> MS SQL Server/MySQL

CREATE TABLE … primary key(UserID) DESCRIBE tabel_1

INSERT INTO … UPDATE… SET… WHERE… DELETE DROP

SELECT … FROM … BY … SELECT DISTINCT

GROUPBY? HAVING? ORDERBY?

JOINS -> LEFT/RIGHT JOIN INNER/FULL OUTER JOIN

https://medium.com/towards-data-science/sql-cheat-sheet-for-interviews-6e5981fa797b

Views? Update View?

Functions? SELECT AVG(age)… AVG, MIN, MAX, SUM, COUNT

ACID Properties *** -> Transactions Indexing?

Normalization? 1NF 2NF 3NF BCNF

NoSQL -> MongoDB/Redis

Why NoSQL? CAP theorem?

Programming Languages

C/C++ vs GO vs Rust?
Java (JVM) vs C# (.NET)
Garbage Collector?
Compiled vs Interpreted
Stack (static) vs Heap (dynamic)
C++ memory management: new, delete, memcpy
C++ Destructors

https://github.com/kamranahmedse/developer-roadmap
https://hackr.io

Others

Docker GraphQL Serverless, Azure func -> Microservices
Assembly PYTHON SPARK AWS GO
Repo/UoW + IoC/DI -> DDD CQRS

DISTRIBUTED SYSTEMS Concurrent -> MPI HPC -> Charm++ | Legion
BigData -> Spark Cloud -> AWS | Azure

DATA SCIENCE Machine Learning Data Mining / IR
Big Data

GraphQL

GO

https://hackernoon.com/the-beauty-of-go-98057e3f0a7d
https://tour.golang.org/welcome/1

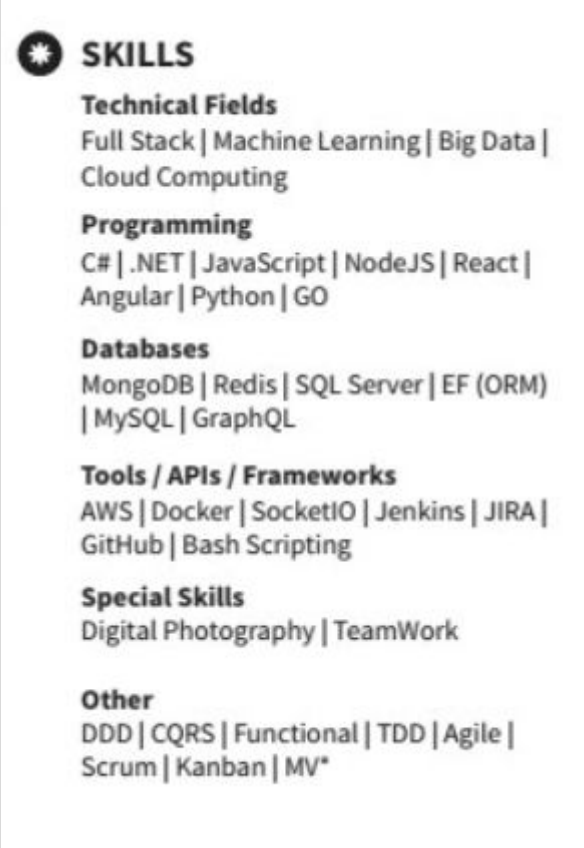
CONCURRENCY -> goroutines, channels
Alternative to C++ GC

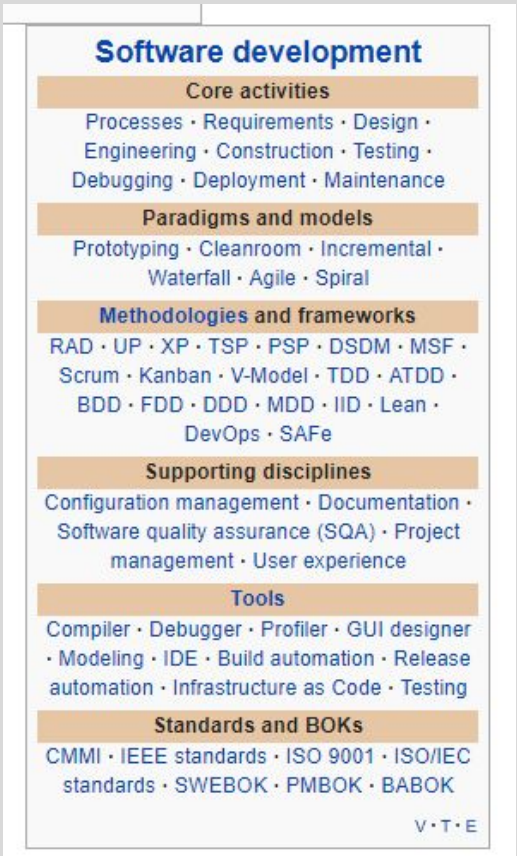
Statically Typed

Docker

Docker is a tool designed to make it easier to create, deploy, and run applications by using containers. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package.

Spark





Interview Q.

SYSCO LABS

Threading in single core? Runnable? Process vs Thread?
Sessions and Cookies?
SQL Queries?
Insertion Sort? Quick Sort?

Prototypes in JS? Object Destructing in ES6? MVC? MVC vs Flux?
Callback Hell? Transpilers? MongoDB? Authentication vs Authorizatiotion?

CODEGEN

Components of OS? Context Switch? Collections in Java?
Array vs ArrayList?

EYEPAX

Java vs C++ in OOP Coupling & Cohesion?

ABOUT COMPANY

ABOUT MYSELF STRENGTHS AND WEAKNESS 5 YEARS PLAN

FINAL YEAR PROJECT

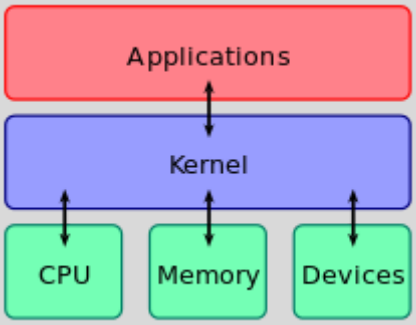
IMPORTANT

medium.com/@chathuranga94/important-b8ecdc99714f

OTHER???

KERNEL (OS)

It handles the rest of start-up as well as
input/output requests from software,
translating them into data-processing instructions for central processing unit.
It handles memory and peripherals like keyboards, monitors, printers.



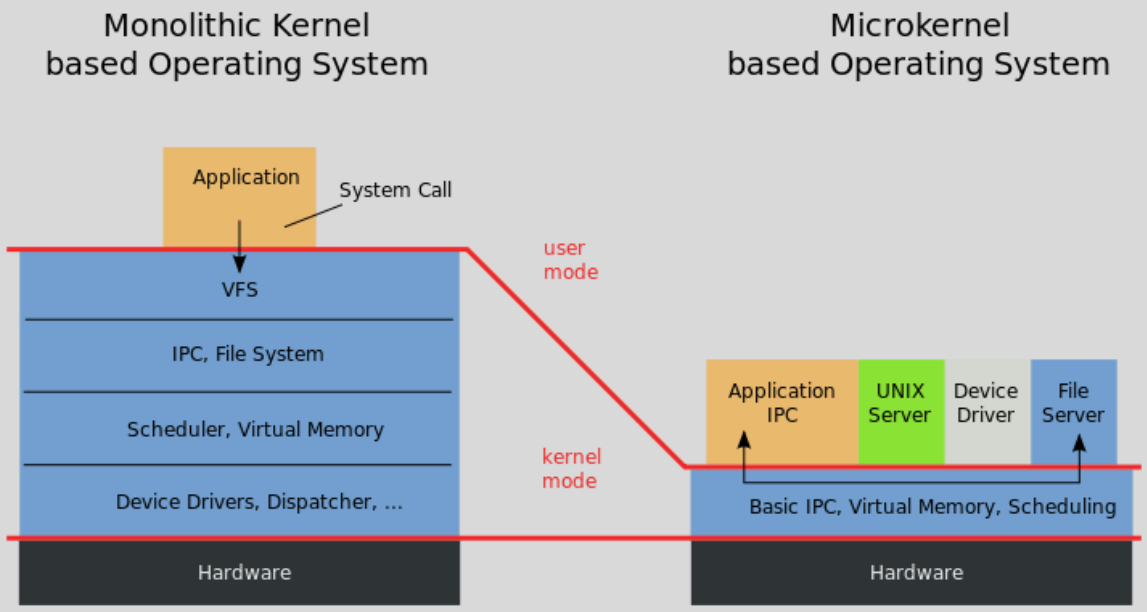
Running processes and handling interrupts, in kernel space.
Writing text in a text editor, running programs in a GUI in user space.

Microkernel

The microkernel approach is to define a very simple abstraction over the hardware, with a set of primitives or system calls to implement minimal OS services such as thread management, address spaces and interprocess communication.

All other services, those normally provided by the kernel such as networking, are implemented in user-space programs referred to as servers.

A microkernel is a piece of software or even code that contains the near-minimum amount of functions and features required to implement an operating system.



HTTP vs HTTPS?
RESTful API?
REST vs SOAP? SOA?
http://adrianmejia.com/blog/2016/03/23/how-to-scale-a-nodejs-app-based-on-number-of-users/
By default, node limits itself to 1.76 GB on 64 bit machines.

ACID in MongoDB? https://dzone.com/articles/how-acid-mongodb

- **Atomicity** requires that each transaction is executed in its entirety, or fail without any change being applied.
- **Consistency** requires that the database only passes from a valid state to the next one, without intermediate points.
- **Isolation** requires that if transactions are executed concurrently, the result is equivalent to their serial execution. A transaction cannot see the partial result of the application of another one.
- **Durability** means that the the result of a committed transaction is permanent, even if the database crashes immediately or in the event of a power loss.

CAP Theorem?

Garbage Collectors? Not used again & out of scope Heap
Managed memory is cleaned up by a Garbage Collecto

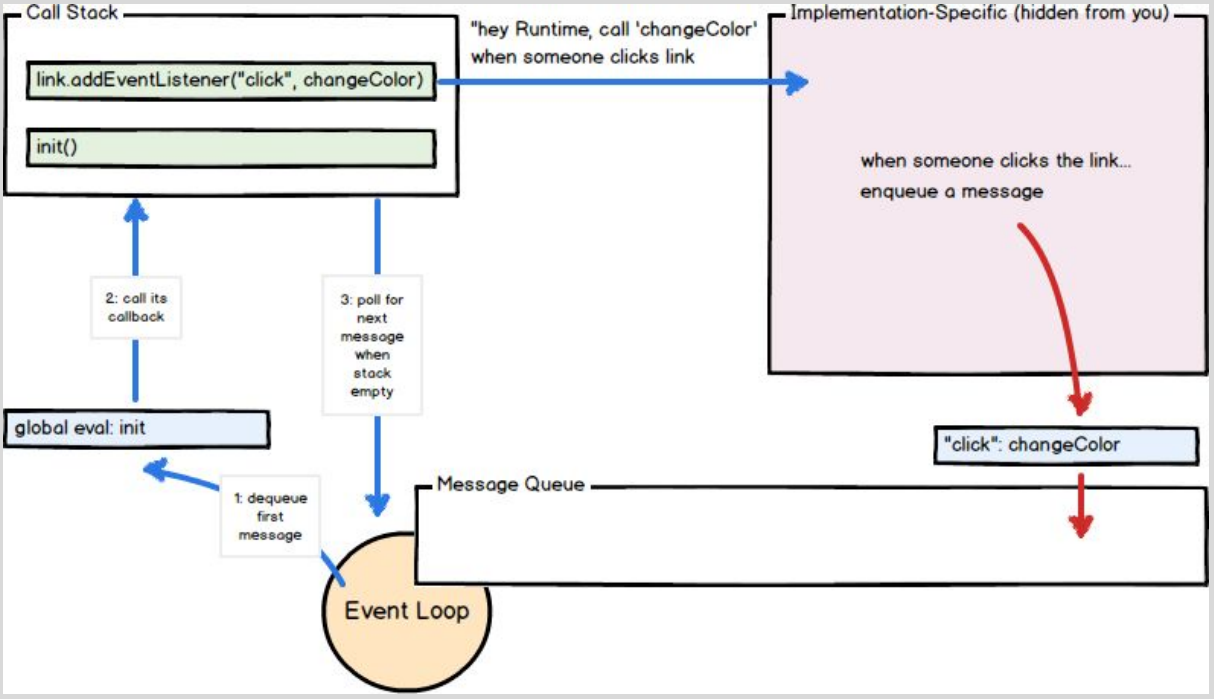
Recursively reverse a string

JavaScript Event Loop
https://blog.carbonfive.com/2013/10/27/the-javascript-event-loop-explained/

Non-blocking I/O
The single thread of execution asks the runtime to perform an operation, providing a callback function and then moves on to do something else.

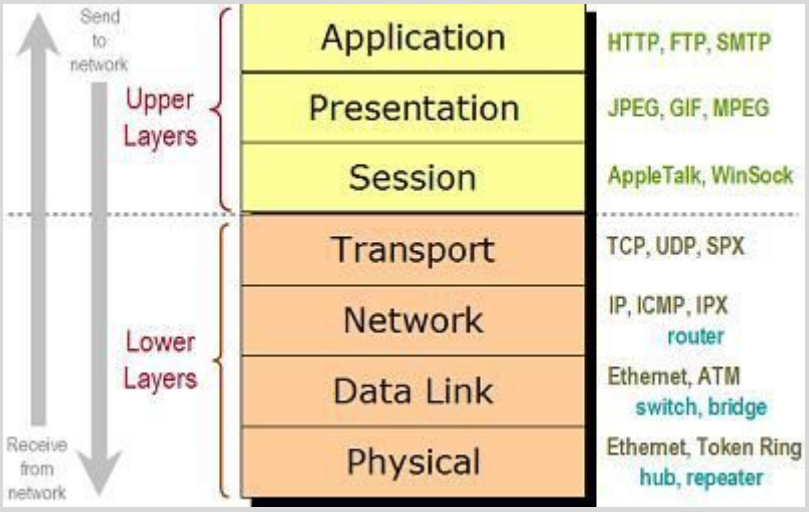
The Event Loop
The decoupling of the caller from the response allows for the JavaScript runtime to do other things while waiting for your asynchronous operation to complete and their callbacks to fire.

JavaScript runtimes contain a message queue which stores a list of messages to be processed and their associated callback functions.



Abstract func vs virtual func.

OSI MODEL



Hashtable is synchronized, whereas HashMap is not.

List vs ArrayList

List<T> is a generic class. It supports storing values of a specific type without casting to or from object. List<T> implements the generic IEnumerable<T> interface and can be used easily in LINQ.
ArrayList simply stores object references.

Reference vs Value
Call by value and call by reference (also known as pass-by-value and pass-by-reference). These methods are different ways of passing (or calling) data to functions.

Protected
The protected keyword is a member access modifier. A protected member is accessible within its class and by derived class instances

Vector C#
Vector<T> is an immutable structure that represents a single vector of a specified numeric type. The count of a Vector<T> instance is fixed, but its upper limit is CPU-register dependent.

thread.wait() vs thread.sleep()
wait() releases the lock while sleep() doesn't releas any lock while waiting.

Virtual vs simple inheritance
https://stackoverflow.com/questions/18787977/difference-between-virtual-and-simple-inheritance-in-c

== checks for memory reference in objects value not type in JS

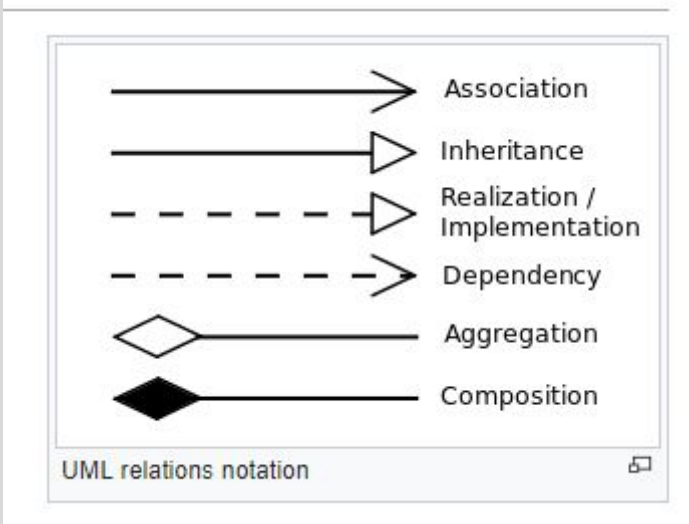
.equals content of objects

SOAP vs REST
Web services are of two kinds: Simple Object Access Protocol (SOAP) and Representational State Transfer (REST).

Promises and Observables
RxJS Observables vs Promises

MVC vs MVVM
ViewModel isn't a controller. It instead acts as a binder that binds data between the view and model. MVVM format with data-binding is designed specifically to allow the view and model to communicate directly with each other.
MVC format is specifically designed to create a separation of concerns between the model and view

UML Class Diagram



- **Association** is a relationship where all objects have their own lifecycle and there is no owner.

  Let's take an example of Teacher and Student. Multiple students can associate with single teacher and single student can associate with multiple teachers, but there is no ownership between the objects and both have their own lifecycle. Both can be created and deleted independently.
- **Aggregation** is a specialised form of Association where all objects have their own lifecycle, but there is ownership and child objects can not belong to another parent object.

  Let's take an example of Department and teacher. A single teacher can not belong to multiple departments, but if we delete the department, the teacher object will *not* be destroyed. We can think about it as a "has-a" relationship.
- **Composition** is again specialised form of Aggregation and we can call this as a "death" relationship. It is a strong type of Aggregation. Child object does not have its lifecycle and if parent object is deleted, all child objects will also be deleted.

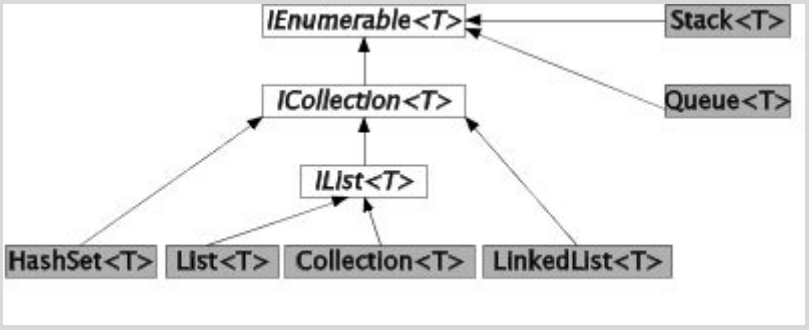
  Let's take again an example of relationship between House and Rooms. House can contain multiple rooms - there is no independent life of room and any room can not belong to two different houses. If we delete the house - room will automatically be deleted.

  Let's take another example relationship between Questions and Options. Single questions can have multiple options and option can not belong to multiple questions. If we delete the questions, options will automatically be deleted.

Sorting Algo

Algorithm	Best-case	Worst-case	Average-case	Space Complexity	Stable?
Merge Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$	Yes
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	Yes
Bubble Sort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	Yes
Quicksort	$O(n \log n)$	$O(n^2)$	$O(n \log n)$	$\log n$ best, n avg	Usually not*
Heapsort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(1)$	No

Collection Hierarchy C#



CREATIONAL	Singleton	Factory	Prototype
BEHAVIORAL	Chain of Responsibility		Observable
STRUCTURAL	Adapter		Flyweight

SQL	ACID	Consistency?	
NoSQL	BASE	Availability?	CAP?