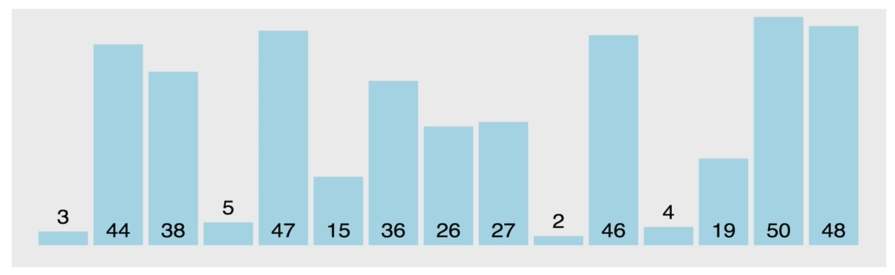
Chapter 7

**Introduction to**

**Data Structures**

**7.1 Data Structures**

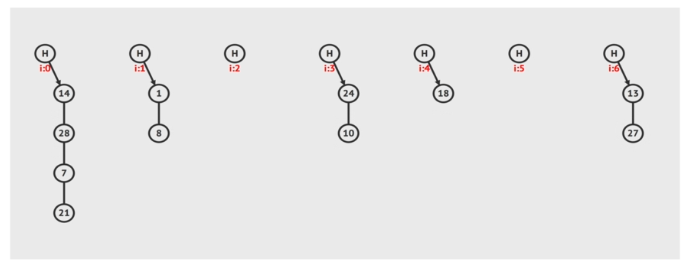
* Why should we care
* What Data structures are
* Why there are many of them
* How they differ
* ES2015
* Working with class in JS
* Following are common Data Structures:
* Binary Search Trees
* Stack
* Queues
* Singly Linked Lists
* Doubly linked Lists
* Undirected Unweighted Graphs
* Binary Heaps
* Directed Graphs
* Hash Tables
* Doubly Linked Lists
* We'll implement each of these Data Structures with their own Class (i.e. we define a Class for each of those DS). Then we use objects from these classes.
* Data structures: Data structures are *collections of values*, the *relationships* among them, and the *functions* or *operations* that can be applied to the data.
* For example: A JS-array is a DS too.
* It can holds data.
* It relates among the data (ORDER in this case).
* It has some operations: like ***push()***, ***pop()*** sorting, reversing functionalities etc
* Our goal: We do the same for our first DS: Singly Linked Lists.
* Firstly, we'll define a class for it
* We define the relation: how to order this kind of data
* We'll define 7-methods in the defined class.
* WHY SO MANY: Different data structures excel at different things. Some are highly specialized, while others (like arrays) are more generally used.
* Our main goal is to learn:
* Where to apply specific DS
* Learn how to best use
* Following are Visual Representation of some common DSs.



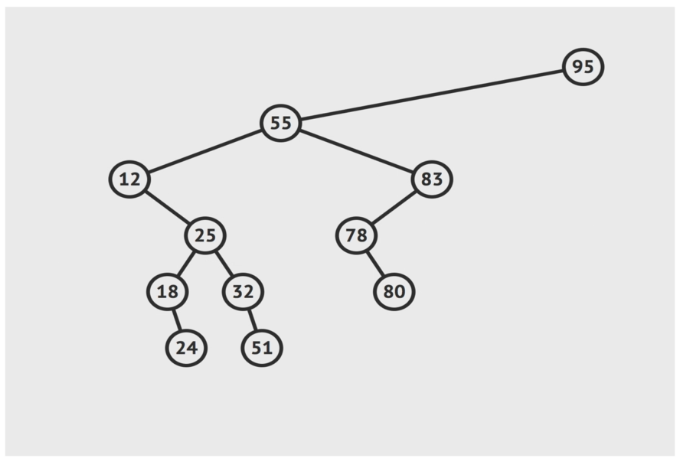
General array



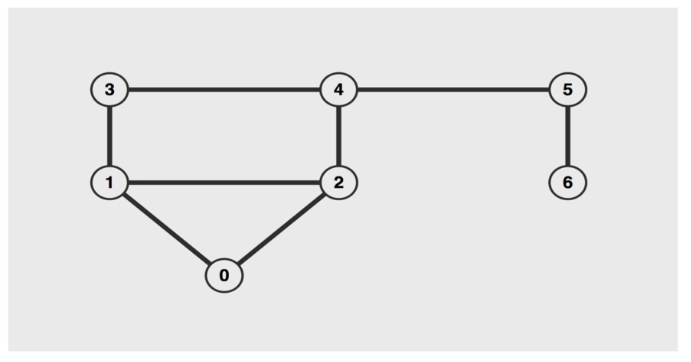
Singly Linked list



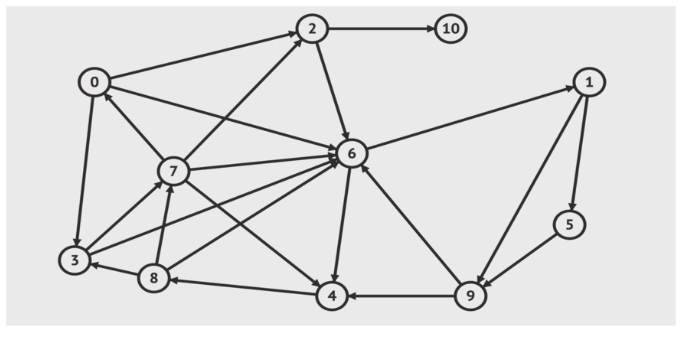
Doubly Linked List



Binary Search Tree



Unweighted un-directed graphs



Unweighted directed graphs

* WHY CARE:
* The more time you spend as a developer, the more likely you'll need to use one of these data structures. Eh: working with different APIs; to store and retrieve data from database.
* You've already worked with many of them unknowingly!
* And of course... Interviews
* Application of different DSs: There is no Best DSs. Each DSs is special for different things. Here are some example of application of different DSs.

1. Working with Map/Location API (GPS app using Google-map): Apply Graph as DS. To find fastest route, shortest distance etc.

* It is difficult to represent these kind of data using a single array (instead of using Graph).

1. Need an ordered list (like an array) with fast inserts/removals at the beginning and end: Apply Singly Linked Lists.
2. Web scrapping in Nested-HTML: Apply Trees.
3. Write a scheduler: Apply Binary Heaps.

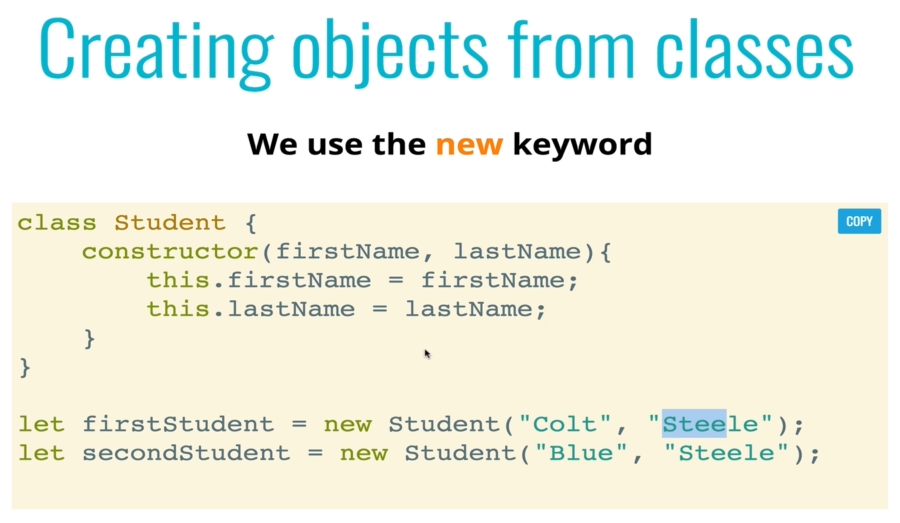
**7.2 Some basic ES2015 Syntax : Class**

Now we just need to learn a bit of ES2015 syntax that well use along the way

* JS doesn't actually support OOP language. However in ***ES2015*** version ***class*** keyword is added and we can implement OOP in JS. So we need to learn these ES2015 syntaxes.
* Objectives:
* Explain what a *class* is (we already familiar with this)
* *Understand* how *JavaScript implements* the idea of *classes*.
* Define terms like *abstraction*, *encapsulation* and *polymorphism* (we also know these)
* Use *ES2015 classes* to implement *data structures*

|  |  |
| --- | --- |
| * How JavaScript implements the idea of classes: JavaScript classes, introduced in ECMAScript 2015, are primarily *syntactical sugar* over JavaScript's existing *prototype-based inheritance*. \ * The class syntax does not introduce a new object-oriented inheritance model to JavaScript. * So JS is not a true OOP language like C++ or Java. * The Syntax:      * The method to create new objects must be called ***constructor***. * The ***class*** keyword creates a constant, so you can not redefine it. Watch out for the syntax as well! * Use PascalCase or camelCase with first capital. | * Here is an Example of How we use Classes to define a DS: |

* Crating Object of a Class: We need to use ***new*** keyword to create an object.



* Example: Following is the simple example of a class.

**class** Student {

**constructor**(firstName, lastName, year){

**this.**firstName=firstName;

**this.**lastName=lastName;

**this.**grade=year;

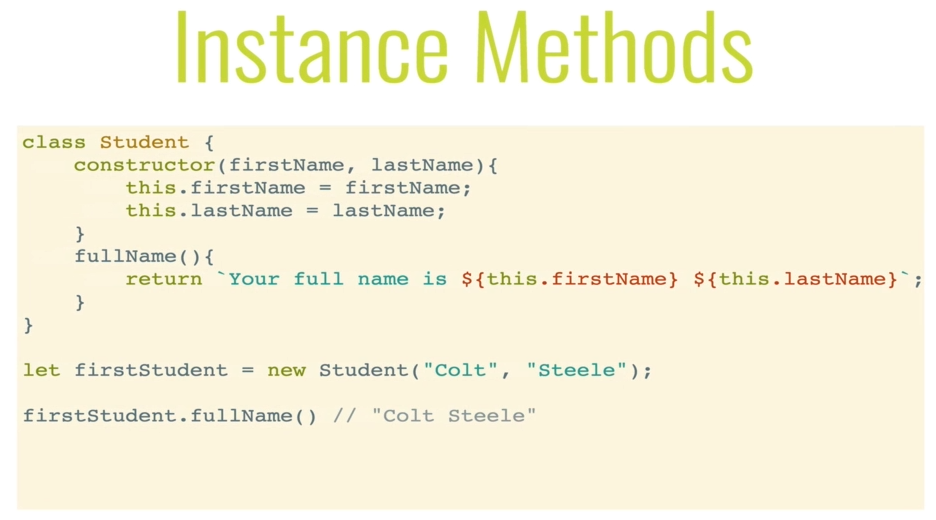
}

}

**let** firstStudent= **new Student**("Colt","Steele",1);

**let** secondStudent= **new Student**("Blue","Steele",2);

**7.3 ES2015 Syntax : Instance-methods**



* Instance Methods: We'll use this kind of method most of the time. Generally these methods acts on ***individual instance*** of a ***Class***. For example: ***numArry.push()*** here ***push()*** is a instance method.
* Example: In following example, **fullName**()is one of the instance method. Notice the ***variables*** are declared inside ***constructor***, to be used in *instance methods*. (similar to Python)
* How to call: We access the method by "." operator.

**let** secondStudent= **new Student**("Blue","Steele",2);

secondStudent**.fullName**()

**class** Student {

**constructor**(firstName, lastName, year){

**this.**firstName=firstName;

**this.**lastName=lastName;

**this.**grade=year;

**this.**tardies=0;

**this.**scores=[];

}

**fullName**(){

**return** `Your full name is ${**this.**firstName} ${**this.**lastName}`;

}

**markLate**(){

**this.**tardies+=1;

**if**(**this.**tardies **>=** 3){

**return** "YOU ARE EXPELLED!!!!"

}

**return** `${**this.**firstName} ${**this.**lastName} has been late ${**this.**tardies} times`;

}

**addScore**(score){

**this.**scores**.push**(score);

**return this.**scores

}

**calculateAverage**(){

**let** sum= **this.**scores**.reduce**(**function**(a,b){**return** a+b;})

**return** sum/**this.**scores**.**length;

}

}

**let** firstStudent= **new Student**("Colt","Steele",1);

**let** secondStudent= **new Student**("Blue","Steele",2);

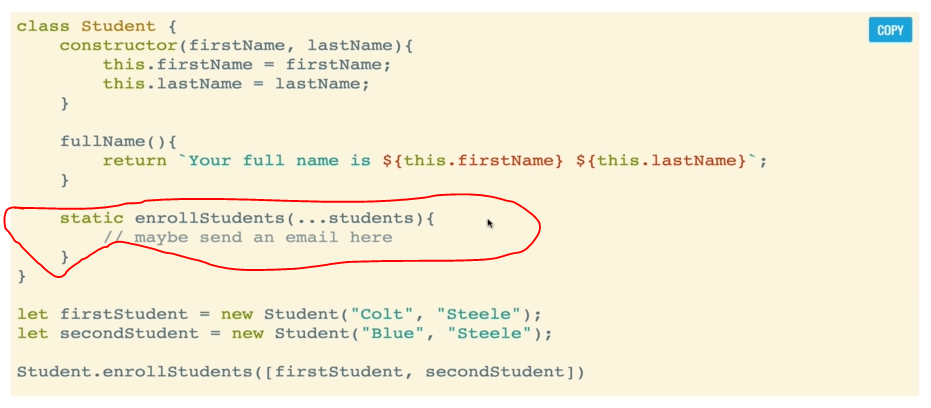
firstStudent**.fullName**()

// *'Your full name is Colt Steele'*

* When we implement our own DSs, for example in case of Singly linked list we'll define: ***reverse(), remove(), push(), pop(), shift(), unshift(), get(), insert()*** etc as instance methods.

**7.4 ES2015 Syntax : Class/static-methods**

The ***static*** keyword defines a static method for a class. Static methods are called without instantiating their class and cannot be called through a class instance. Static methods are often used to create utility functions for an application.



**class** Student {

**constructor**(firstName, lastName, year){

**this.**firstName=firstName;

**this.**lastName=lastName;

**this.**grade=year;

**this.**tardies=0;

**this.**scores=[];

}

**fullName**(){

**return** `Your full name is ${**this.**firstName} ${**this.**lastName}`;

}

**markLate**(){

**this.**tardies+=1;

**if**(**this.**tardies **>=** 3){

**return** "YOU ARE EXPELLED!!!!"

}

**return** `${**this.**firstName} ${**this.**lastName} has been late ${**this.**tardies} times`;

}

**addScore**(score){

**this.**scores**.push**(score);

**return this.**scores

}

**calculateAverage**(){

**let** sum= **this.**scores**.reduce**(**function**(a,b){**return** a+b;})

**return** sum/**this.**scores**.**length;

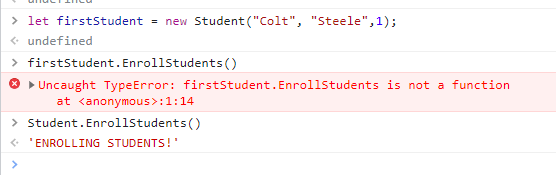
}

**static** **EnrollStudents**(){

**return** "ENROLLING STUDENTS!"

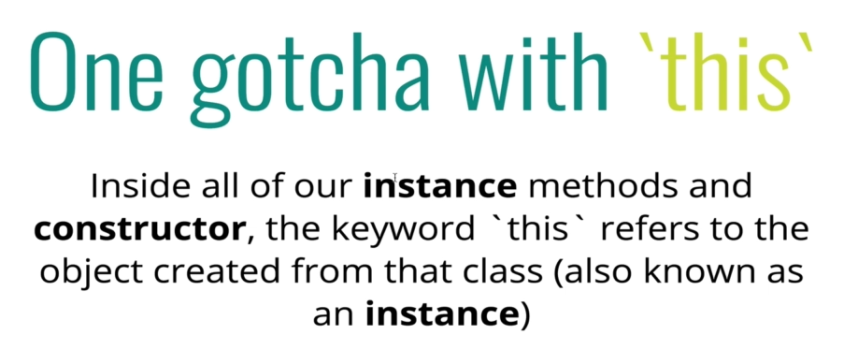
}

}



* Notice **EnrollStudents** cannot be called using **firstStudent** object of **Student** class.
* **Student** class used to call **EnrollStudents**.
* Following demonstrates the usual use of Static/Class-methods.

|  |  |
| --- | --- |
| * Notice that ***this*** keyword is not used inside the Static/Class-method **distance**(a, b). * Because it has nothing to do with object creation, it just working with the object. * In **distance**(a, b), ***a*** and ***b*** are the ***objects***, not a variable of the class. | **class** Point {  **constructor**(x, y){  **this.**x=x;  **this.**y=y;  }  **static** **distance**(a, b){  **const** dx=a**.**x-b**.**x;  **const** dy=a**.**y-b**.**y;  **return** Math**.hypot**(dx,dy);  }  }  **const** p1= **new Point**(5,5);  **const** p2= **new Point**(10,10);  console**.log**(Point**.distance**(p1, p2));  // *7.0710678118654755* |

****

* Inside all of our instance methods and constructor, the keyword **'this'** refers to the object created from that class (also known as an instance).
* In C++ this is a pointer that is *automatically passed to any member function* when it is called, and it is a *pointer to the object* that generates the call (this works *implicitly*).
* Recap
* Classes are blueprints that when created make objects known as instances
* Classes are created with the ***new*** keyword
* The *constructor* function is a special function that *gets run* when the class is *instantiated*
* Instance methods can be added to classes similar to methods in objects
* *Class methods* can be added using the *static* keyword

