Ember 2

Section 5 – Object Model

Ember Objects
Classes and Instances
Computed Properties
Observers / Bindings
Enumerables

Section Objectives

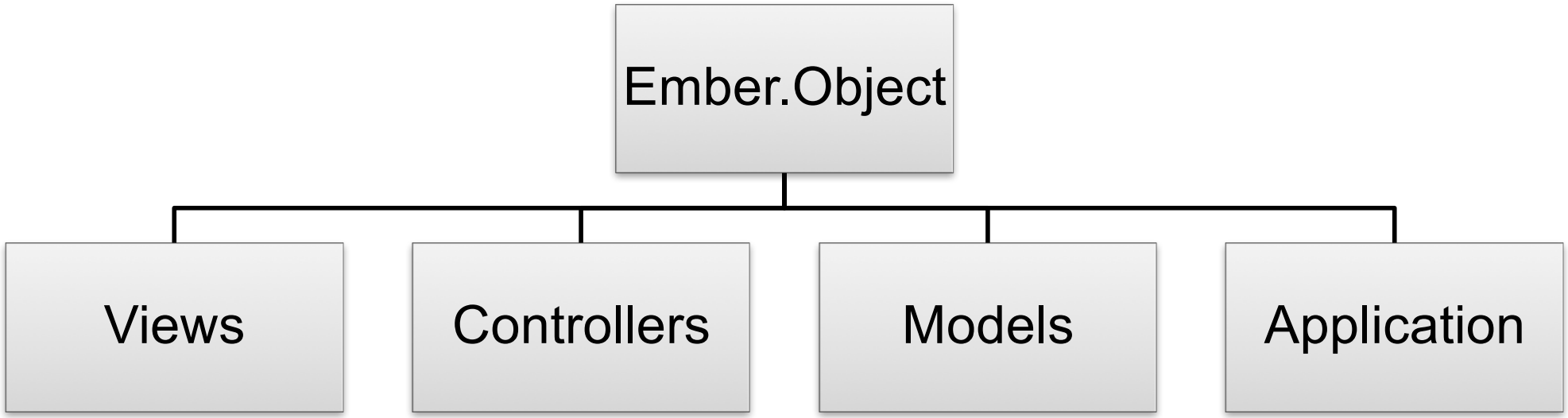
- Understand the Ember Object hierarchy
- Be able to extend current Ember classes and create instances of them
- Be able to create a computed property using a getter, setter method or aggregating data
- Be able to create an Ember Observer
- Be able to use and create Ember Enumerables

Object Model

Ember Objects

Ember Objects
Classes and Instances
Computed Properties
Observers / Bindings
Enumerables

Object Model



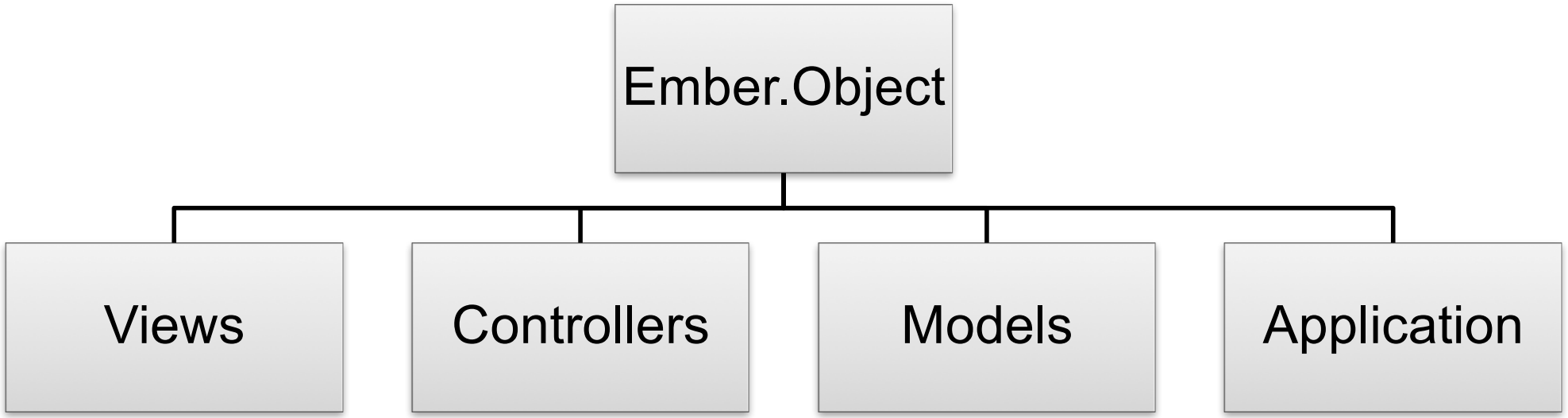
http://emberjs.com/api/classes/Ember.Object.html

Object Model

Classes and Instances

Ember Objects
Classes and Instances
Computed Properties
Observers / Bindings
Enumerables

Object Model




```
let Person =
Ember.Object.extend(
   {
      species: “homo sapiens”,
      category: “Mammal”
   }
);
```

Class

```
let joe =
Ember.Object.create(
   {
      id: ‘333-55-777’,
      name: “Joe Rizzo”
   }
);
```

Instance

Classes

- ES6 class

```
class Person {
    constructor(name) {
        this.name = name;
    }

    say(something) {
        console.log(something);
    }
}
```

- Ember class

```
const Person = Ember.Object.extend({
    name: “”,
    say(something) {
        console.log(something);
    }
});
```

Instances

- ES6

```
var joe = new Person("Joe");
```

- Ember

```
var joe = Person.create({
    name: "Joe"
});
```

Initializing

- Shared Across Instances

```
const Person = Ember.Object.extend({
    vehicles: [‘MINI’, ‘VW’, ‘Porsche’]
}); // if another instance adds to it, all instances
       pick it up
```

- Unique per Instance

```
const Person = Ember.Object.extend({
    init() {
        this.set(‘vehicles’, [‘MINI’, ‘VW’, ‘Porsche’]);
    }
}); // if another instance adds to it, only that
       instance gets the addition
```

Accessing and Mutating Object Properties

- SET

```
const Person = Ember.Object.extend({
    name: ‘’
});

let joe = Person.create({name: ’Joe’});
joe.set(‘name’, ‘Joe’);
```

- GET

```
const Person = Ember.Object.extend({
    name: ‘’
});

let shoaib = Person.create({name: ‘Shoaib’});
shoaib.get(‘name’); // Shoaib
```

Object Model

Computed Properties

Ember Objects
Classes and Instances
Computed Properties
Observers / Bindings
Enumerables

Computed Properties

- Properties backed with functions and associated Observables

```
var Person = Ember.Object.extend({
    first_name: “”,
    last_name:  “”,
    full_name:  Ember.computed(‘first_name’, ‘last_name’, function() {
        return `${this.get(‘first_name’)} ${this.get(‘last_name’)}`;
    })
});
```

Chained Computed Properties

```
var Person = Ember.Object.extend({
    first_name: “”,
    last_name:  “”,
    prefix:     “”,
    full_name:  Ember.computed(‘first_name’, ‘last_name’, function() {
        return `${this.get(‘first_name’} ${this.get(‘last_name’}`;
    },
    description: Ember.computed(‘full_name’, ‘prefix’, function() {
        return `${this.get(‘prefix’)} ${this.get(‘full_name’)}`;
    })
});
```

Dynamic Updating

- Computed property functions are not executed until they are requested

- Once requested, the result is cached until there is a change to one of its dependent properties; then it is invalidated

Setting Computed Properties

- Must provide a get()

```
var Person = Ember.Object.extend({
    first_name: “”,
    last_name:  “”,
    prefix:     “”,
    full_name:  Ember.computed(‘first_name’, ‘last_name’, {
        get(key) {
            return `${this.get(‘first_name’)} ${this.get(‘last_name’)}`;
        },
        set(key, value) {
             let [firstName, lastName] = value.split(/\s+/);
             this.set(‘first_name’, firstName);
             this.set(‘last_name’, lastName);
             return value;
        }
    },
    description: Ember.computed(‘full_name’, ‘prefix’, function() {
        return `${this.get(‘prefix’)} ${this.get(‘full_name’)}`;
    })
});
```

Computed Property Macros

- Must provide a get()

```
var Person = Ember.Object.extend({
    first_name: “”,
    last_name:  “”,
    prefix:     “”,
    full_name:  Ember.computed(‘first_name’, ‘last_name’, {
        get(key) {
            return `${this.get(‘first_name’} ${this.get(‘last_name’}`;
        },
        set(key, value) {
             let [firstName, lastName] = value.split(/\s+/);
             this.set(‘first_name’, firstName);
              this.set(‘last_name’, lastName);
              return value;
        }
    },
    isIronMan: Ember.computed.equal(‘full_name’, ‘Tony Stark’)
});
```

http://emberjs.com/api/classes/Ember.computed.html

Object Model

Observers / Bindings

Ember Objects
Classes and Instances
Computed Properties
Observers / Bindings
Enumerables

Observers

- Ember supports observing any property, including computed properties
- Observers are useful to perform some action after a binding synchronization as occurred
- Typically over used by new Ember developers when computed properties are sufficient
- Observers are synchronous which can lead to sync'ing bugs
- Let's take a look

Observers

- Synchronous behavior

```
var Person = Ember.Object.extend({
    first_name: “”,
    last_name:  “”,
    full_name:  Ember.computed(‘first_name’, ‘last_name’, function() {
        return `${this.get(‘first_name’} ${this.get(‘last_name’}`;
    }),
    lastNameChanged: Ember.observer(‘last_name’, function() {
       // because he observer depends on last name and so does
       // full name, since the observer is synchronous full name
       // will be the old value
        console.log(this.get(‘full_name’));
    })
});
```

- What would happen if the observer depended on multiple properties?

Run Loop

- To get around observers being synchronous and ensuring they run when bound properties are updated use:

Ember.run.once()

```
var Person = Ember.Object.extend({
    first_name: “”,
    last_name:  “”,
    full_name:  Ember.computed(‘first_name’, ‘last_name’, function() {
        return `${this.get(‘first_name’} ${this.get(‘last_name’}`;
    },
    lastNameChanged: Ember.observer(‘last_name’, function() {
        Ember.run.once(this, ‘displayFullName’);
    }
    displayFullName() {
        console.log(this.get(‘full_name’);
    }
});
```

Object Initialization

- Observers never fire until **after** the initialization of an object is complete

```
var Person = Ember.Object.extend({
    first_name: “”,
    last_name:  “”,
    full_name:  Ember.computed(‘first_name’, ‘last_name’, function() {
        return `${this.get(‘first_name’} ${this.get(‘last_name’}`;
    },
    init() {
        this.set(‘last_name’, ‘Smith’);
    },
    lastNameChanged: Ember.on(‘init’, Ember.observer(‘last_name’,
        function() {
            // do something with last name
        }
    )
});
```

Unconsumed Computed Properties

- Because computed properties only process when accessed, any observer that is dependent on a computed property will not be processed unless the computed property has been accessed

- To ensure that a computed property has been accessed, initialize it in your init(), and then have an observer Ember.on('init', Ember.observer(property, callback))

Observers Outside of Class

```
var Person = Ember.Object.extend({
    first_name: “”,
    last_name:  “”,
    full_name:  Ember.computed(‘first_name’, ‘last_name’, function() {
        return `${this.get(‘first_name’} ${this.get(‘last_name’}`;
    },
    init() {
        this.set(‘last_name’, ‘Smith’);
    },
    lastNameChanged: Ember.on(‘init’, Ember.observer(‘last_name’,
        function() {
            // do something with last name
        }
    )
});
```

```
var david = Person.create();
david.addObserver(‘full_name’, function() {
    // deal with change from Person class
};
```

Bindings

- Bindings are most often used within the Ember framework

- Computed properties will suffice for most problems you will face when needing to create your own bindings

Two-way binding

```
let NoteComponent = Ember.Component.extend({
    note: null,
    selected: Ember.computed.alias('note.selected');
});
```

One-way binding

```
let NoteComponent = Ember.Component.extend({
    note: null,
    selected: Ember.computed.oneWay('note.selected');
});
```

Object Model

Enumerables

Ember Objects
Classes and Instances
Computed Properties
Observers / Bindings
Enumerables

Enumberables

- Ember.Enumerable API
  - Mixin that is applied to the Ember.Array class on page load

- Ember.ArrayProxy
  - wraps any other object that implements Ember.Array

JavaScript Method	Observable Equivalent
pop	popObject
push	pushObject
reverse	reverseObject
shift	shiftObject
unshift	unshiftObject

Common Functions and Properties

- **forEach**

```
let names = ['Joe', 'Shoaib', 'James'];

names.forEach((item, index) => {
    console.log(`Name Index ${index+1}: ${item}`);
});
```

- **firstObject / lastObject properties**

```
let names = ['Joe', 'Shoaib', 'James'];

names.get('firstObject'); // Joe
names.get('lastObject'); // James
```

- **map**

```
let names = ['Joe', 'Shoaib', 'James'];

names.map(item => `Mr. ${item}`);
// ['Mr. Joe', 'Mr. Shoaib', 'Mr. James']
```

Common Functions and Properties

- **mapBy**

```
let arizona= Ember.Object.create({
    capital: 'Phoenix'
});
let california = Ember.Object.create({
    capital: 'Sacramento'
}):
let states = [arizona, california];

states.mapBy('capital'); // ['Phoenix', 'Sacramento']
```

- **filter**

```
let arr = [1, 2, 3, 4, 5];

arr.filter((item, index, self) => item < 4); // [1, 2, 3]
```

Common Functions and Properties

- **filterBy**

```
let Note = Ember.Object.extend({
    title: '',
    selected: false,
});

let notes = [
    Note.create({ title: 'Groceries', selected: true });
    Note.create({ title: 'TODO', selected: false });
];

notes.filterBy('selected', true);
// returns only first note
```

Common Functions and Properties

- **every**

```
let Note = Ember.Object.extend({
    title: '',
    selected: false,
});

let notes = [
    Note.create({ title: 'Groceries', selected: true });
    Note.create({ title: 'TODO', selected: false });
];

notes.every((note, index, self) => note.get('selected');
// returns false
```

- **any**

```
notes.any((note, index, self) => note.get('selected');
// returns true
```

Common Functions and Properties

- **isEvery**

```
let Note = Ember.Object.extend({
    title: '',
    selected: false,
});

let notes = [
    Note.create({ title: 'Groceries', selected: true });
    Note.create({ title: 'TODO', selected: false });
];

notes.isEvery('selected', true); // false
```

- **isAny**

```
notes.isAny('selected', true); // true
```

Custom Enumerables

1. You must have a length property. This should change whenever the size of your collection changes. If you use with an Ember.Object subclass, be sure to change the length property using ***set()***.

2. You must implement ***nextObject()***.

3. Apply the Ember.Enumerable mixin to your class.

```
const MySkipEnumMixin = Mixin.create(Enumerable, {
    length: null,

    nextObject(index) {
        return objectAt(this, ++index);
    }
});
```

Lab 3 – Computed Properties

1. In the ‘note-row’ component, we have a two-way binding of ‘note.selected’ to the ‘selected’ property. In the selectNote(), we set ‘note.selected’ to true. Is this necessary? Set another available property based on your understand of two-way bindings.
2. In the ‘application’ controller, we set the model’s ’singleNote’ property in the addNote(). Remove this and make the ’singleNote’ property a computed property in the ‘application’ route. We were kind of breaking single responsibility methods here.
3. There is a bug with our ’numSelected’ property corresponding to the master view title when we ’Cancel’ the selection process; it continues to show the number we selected if we go back into it. Fix this.

Lab 4 – Enumerations

1. In the ‘application’ controller’s editMode(), we check to see if we are going out of ‘editMode’ and reset all selected notes to false. Instead of using the ‘forEach’ enumerator, use the ‘map’ enumerator to set the ’edit’ property to false.

Section Review

- Understand the Ember Object hierarchy
- Be able to extend current Ember classes and create instances of them
- Be able to create a computed property using a getter, setter method or aggregating data
- Be able to create an Ember Observer
- Be able to use and create Ember Enumerables

End of Section

- This slide is purposely devoid of any information