tutorialspoint

SIMPLYEASYLEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About the Tutorial

Ember.js is an open source JavaScript client-side framework used for developing web applications. It uses the MVC(Model-View-Controller) architecture pattern. In Ember.js, *route* is used as model, *handlebar template* as view and *controller* manipulates the data in the model.

This tutorial covers most of the topics required for a basic understanding of EmberJS and to get a feel of how it works.

Audience

This tutorial is designed for software programmers who aspire to learn the basics of EmberJS and its programming concepts in simple and easy ways. This tutorial will give you enough understanding on the components of EmberJS with suitable examples.

Prerequisites

Before proceeding with this tutorial, you should have a basic understanding of HTML, CSS, JavaScript, Document Object Model (DOM) and any text editor. As we are going to develop web-based applications using EmberJS, it will be good if you have a good understanding on how the Internet and the web-based applications work.

Copyright & Disclaimer

Table of Contents

1. EMBERJS – OVERVIEW

What is Ember.js?

Ember.js is an open source, free JavaScript client-side framework used for developing web applications. It allows building client side JavaScript applications by providing a complete solution which contains data management and an application flow.

The original name of Ember.js was *SproutCore MVC framework.* It was developed by *Yehuda Katz* and initially released on in *December 2011.* The stable release of Ember.js is 2.10.0 and this was released on November 28, 2016.

Why Ember.js?

Consider the following points to understand the use of Ember.js:

- Ember.js is an open source JavaScript framework under MIT license.
- It provides the new binding syntax using the *HTMLBars* template engine which is a superset of the *Handerlbars* templating engine.
- It provides the *Glimmer rendering engine* to increase the rendering speed.
- It provides the *Command Line Interface* utility that integrates Ember patterns into development process and focuses easily on the developer productivity.
- It supports *data binding* to create the link between two properties and when one property changes, the other property will get upgraded with the new value.

Features of Ember.js

Following are the some of the most prominent features of Ember.js:

- Ember.js is used for creating reusable and maintainable JavaScript web applications.
- Ember.js has *HTML* and *CSS* at the core of the development model.
- It provides the instance initializers.
- The routes are core features of the Ember.js which are used for managing the URL's.
- Ember.js provides *Ember Inspector* tool for debugging Ember applications.
- Ember.js uses templates that help to automatically update the model, if the content of applications gets changed.

2. EMBERJS – INSTALLATION

It is easy to configure Ember.js in your system. By using the Ember CLI (Command Line Interface) utility, you can create and manage your Ember projects. The Ember CLI deals with different kinds of application asset management such as concatenation, minification and versioning and also provide generators to produce components, routes etc.

To install Ember CLI, you need to have the following dependencies:

- **Git**: It is an open source version control system for tracking the changes made in the files. For more information, check the official website of git. Ember uses Git to manage its dependencies.
    - *Installing Git on Linux*: Install the Git on Linux by using this link – http://git-scm.com/download/linux
    - *Installing Git on Mac*: Install the Git on Mac OS by using this link – https://git-scm.com/download/mac
    - *Installing Git on Linux*: Install the Git on Windows by using this link – https://git-scm.com/download/win
- Node.js and npm: Node.js is an open source, used for developing server side and networking applications. It is written in JavaScript. NPM is a node package manager used for installing, sharing and managing the dependencies in the projects. Ember CLI uses Node.js run time and npm to get the dependencies.
- Bower: It is used for managing the components such as HTML, CSS, JavaScript, image files etc and can be installed by using the npm.
- Watchman: This optional dependency can be used to watch the files or directories and execute some actions when they change.
- PhantomJS: This optional dependency can be used for running browser based unit tests to interact with web page.

Installing Ember CLI

Ember CLI integrates Ember patterns into development process and focuses easily on the developer productivity. It is used for creating Ember apps with Ember.js and Ember data.

You can install Ember using npm as in the command given below:

```
npm install -g ember-cli
```

To install the beta version, use the following command:

```
npm install -g ember-cli@2.10
```

To check the successful installation of Ember, use the following command:

```
ember -v
```

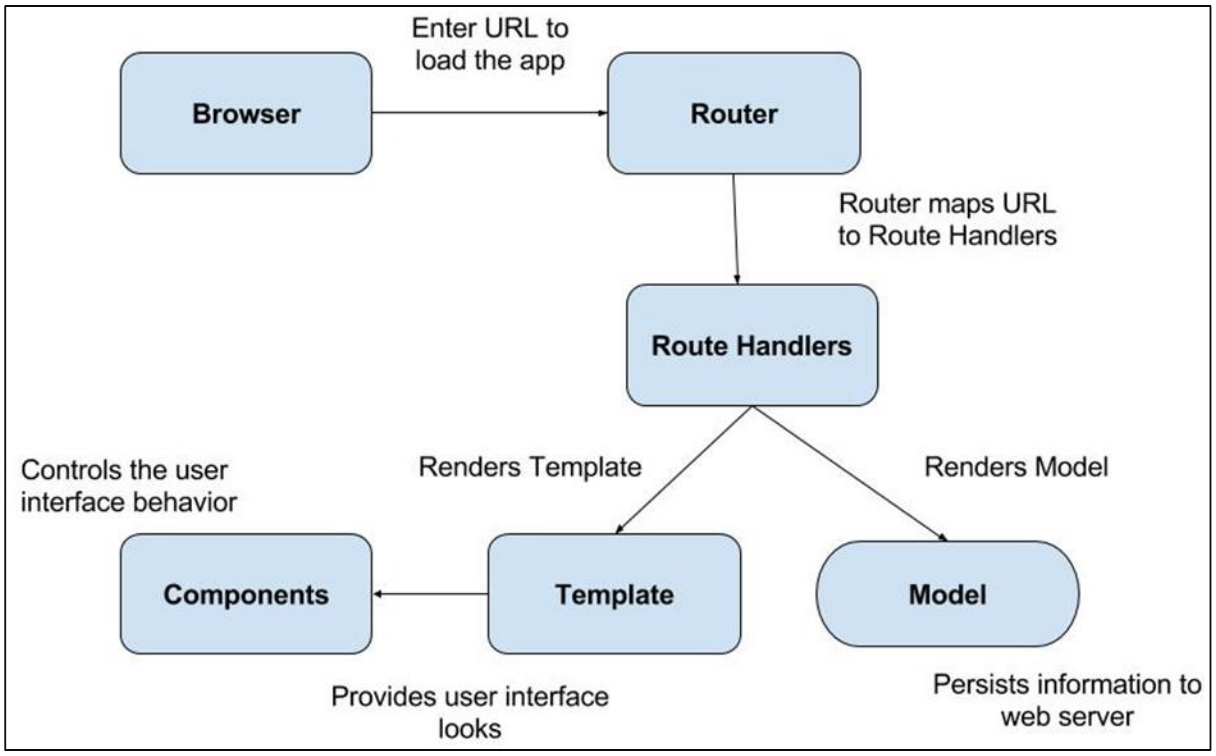
After executing the above command, it will show something like this:

```
ember-cli: 2.10.1

node: 0.12.7

os: win32 ia32
```

3. EMBERJS – CORE CONCEPTS

Ember.js has the following core concepts:

- Router
- Templates
- Models
- Components


Enter URL to load the app
Browser
Router
Router maps URL to Route Handlers
Route Handlers
Controls the user interface behavior
Renders Template
Renders Model
Components
Template
Model
Provides user interface looks
Persists information to web server

Router and Route Handlers

The URL loads the app by entering the URL in the address bar and user will click a link within the app. The Ember uses the router to map the URL to a route handler. The router matches the existing URL to the route which is then used for loading data, displaying the templates and setting up an application state.

The Route handler performs the following actions:

- It provides the template.
- It defines the model that will be accessible to the template.

- If there is no permission for the user to visit a particular part of the app, then the router will redirect to a new route.

Templates

Templates are powerful UI for the end-users. Ember template provides user interface look of an application which uses the syntax of Handlebars templates. It builds the front-end application, which is like the regular HTML. It also supports the regular expression and dynamically updates the expression.

Model

The route handlers render the model that persists information to the web server. It manipulates the data stored in the database. The model is the simple class that extends the functionality of the Ember Data. Ember Data is a library that is tightly coupled with Ember.js to manipulate with the data stored in the database.

Components

The component controls the user interface behavior which includes two parts:

- a template which is written in JavaScript
- a source file which is written in JavaScript that provides behavior of the components.

4. EMBERJS – CREATING AND RUNNING APPLICATION

You can easily configure the Ember.js in your system. The installation of Ember.js is explained in the EmberJS Installation chapter.

Creating Application

Let us create one simple app using Ember.js. First create one folder where you create your applications. For instance, if you have created the "emberjs-app" folder, then navigate to this folder as:

```
$ cd ~/emberjs-app
```

Inside the "emberjs=app" folder, create a new project by using the *new* command:

```
$ ember new demo-app
```

When you create a project, *new* command provides the following directory structure with files and directories:

```
|-- app
|-- bower_components
|-- config
|-- dist
|-- node_modules
|-- public
|-- tests
|-- tmp
|-- vendor


bower.json
ember-cli-build.js
package.json
README.md
testem.js
```

- **app**: It specifies the folders and files of models, routes, components, templates and styles.
-

- **bower_components / bower.json**: It is used for managing the components such as HTML, CSS, JavaScript, image files etc and can be installed by using the npm. The *bower_components* directory contains all the Bower components and *bower.json* contains the list of dependencies which are installed by Ember, Ember CLI Shims and QUnit.
-
- **config**: It contains the *environment.js* directory which is used for configuring the settings of an application.
-
- **dist**: It includes the output files which are deployed when building the app.
-
- **node_modules / package.json**: NPM is a node package manager for Node.js which is used for installing, sharing and managing the dependencies in the projects. The *package.json* file includes the current npm dependencies of an application and the listed packages get installed in the **node_modules** directory.
-
- **public**: It includes assets like images, fonts, etc.
-
- **vendor**: It is a directory in which the front-end dependencies such as JavaScript, CSS are not controlled by Bower go.
-
- **tests / testem.js**: The automated tests are stored under the tests folder and the test runner *testem* of Ember CLI's is arranged in *testem.js*.
-
- **tmp**: It contains the temporary files of Ember CLI.
-
- **ember-cli-build.js**: It specifies how to build the app by using the Ember CLI.

Running Application

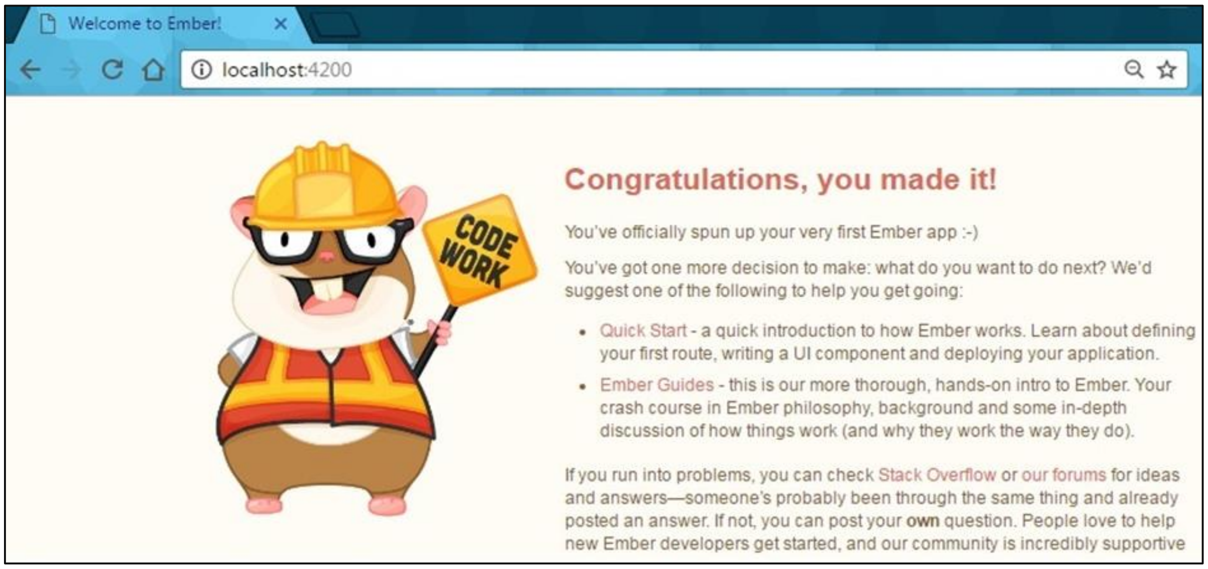
To run the application, navigate to the newly created project directory:

```
$ cd demo-app
```

We have created the new project and it is ready to run with the command given below:

```
$ ember server
```

Now open the browser and navigate to http://localhost:4200/. You will get the Ember Welcome page as shown in the image below:



Congratulations, you made it!

You've officially spun up your very first Ember app :-)

You've got one more decision to make: what do you want to do next? We'd suggest one of the following to help you get going:

- Quick Start - a quick introduction to how Ember works. Learn about defining your first route, writing a UI component and deploying your application.
- Ember Guides - this is our more thorough, hands-on intro to Ember. Your crash course in Ember philosophy, background and some in-depth discussion of how things work (and why they work the way they do).

If you run into problems, you can check Stack Overflow or our forums for ideas and answers—someone's probably been through the same thing and already posted an answer. If not, you can post your **own** question. People love to help new Ember developers get started, and our community is incredibly supportive

5. EMBERJS – OBJECT MODEL

In Ember.js, all objects are derived from the *Ember.Object*. Object-oriented analysis and design technique is called **object modeling**. The *Ember.Object* supports features such as mixins and constructor methods by using the class system. Ember uses the Ember.Enumerable interface to extend the JavaScript *Array* prototype to give the observation changes for arrays and also uses the formatting and localization methods to extend the *String* prototype.

The following table lists down the different types of object model in Ember.js along with their description:

S.NO.	Types & Description
1	Classes and Instances Class is a template or blue print, that has a collection of variables and functions, whereas instances are related to the object of that class. You can create new Ember class by using the Ember.Object's *extend()* method.
2	Reopening Classes and Instances This is nothing but updating the class implementation without redefining it.
3	Computed Properties A computed property declares functions as properties and Ember.js automatically calls the computed properties when needed and combines one or more properties in one variable.
4	Computed Properties and Aggregate Data The computed property accesses all items in an array to determine its value.
5	Observers The observer observes the property such as computed properties and updates the text of the computed property.
6	Bindings The binding is a powerful feature of Ember.js which helps to create a link between two properties and if one of the properties gets changed, the other one is updated automatically.

EmberJS – Classes and Instances

Class is a template or blue print, that has a collection of variables and functions, where as instances are related to the object of that class. Creating and extending the Ember class on Ember.Object is the main property of the Ember object model.

Defining Classes

You can create new Ember class by using the Ember.Object's *extend()* method:

```
const Demo = Ember.Object.extend({
   //code here
});
```

The above code creates new Ember class called "Demo" which inherits the properties from initializers, computed properties, etc. After creating the class, you need to create instance of it by using the *create()* method as shown below:

```
const state = Demo.create();
```

Using the above instance "state", access the properties by using the *set* and *get* accessor methods.

```
console.log(state.get('stateOn'));
```

You can change the "stateon" property by using the set method as shown below:

```
state.set('stateOn', true);
```

Initializing Instance

You can initialize the new instance by invoking the *init()* method. When declaring objects in the class, you need to initialize each instance with the *init()* method.

Example

The following example uses the above mentioned properties and displays an alert message when an Ember object is initialized:

```
import Ember from 'ember';   //import ember module
export default function() {
    //new ember object
    const Demo = Ember.Object.extend({
        init(){
            alert('The default property of stateOn is : ' + this.get('stateOn'));
        },
        stateOn: false
    });

    const state = Demo.create();   //new instance from object with create() method
    state.set('stateOn', true);
    console.log(state.get('stateOn'));
}
```

Now open the *app.js* file and add the following line on top of the file:

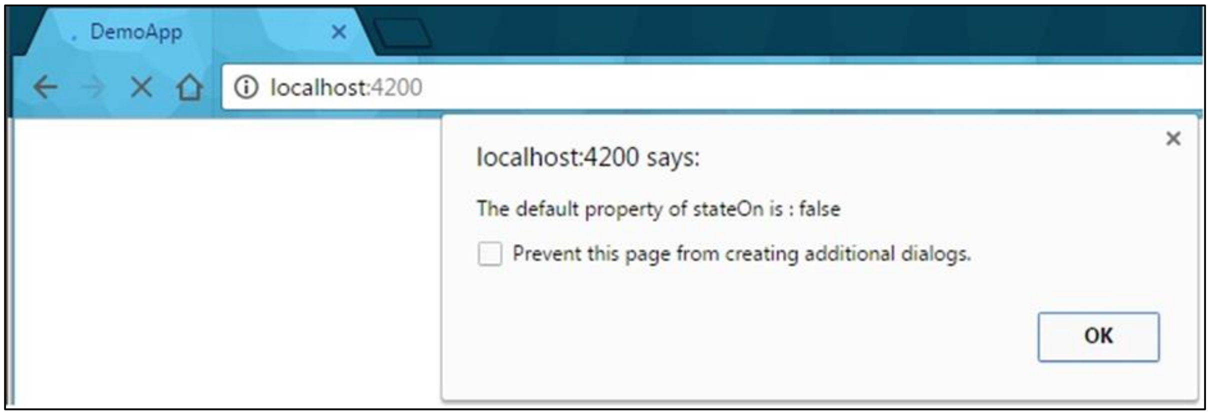
```
import classinstance from './classinstance';
```

Where, classinstance is a name of the file specified as "classinstance.js" and created under the "app" folder. Now, call the inherited "classinstance" at the bottom, before the export. This executes the classinstance function which is created in the *classinstance.js* file:

```
classinstance();
```

Output

Run the ember server and you will receive the following output:



localhost:4200 says:

The default property of stateOn is : false

Prevent this page from creating additional dialogs.

EmberJS – Classes and Instances

This is nothing but updating the class implementation without redefining it and reopening the class by specifying new properties in it. This is possible by using the following methods:

- **reopen()**: It adds properties and methods to *instances*.
- **reopenClass()**: It adds properties and methods to the *classes*.

Example

The following example uses the methods mentioned above and specifies the new properties or methods in it:

```
import Ember from 'ember';
export default function() {
   // reopen() method for instances
   var Person = Ember.Object.extend({
      firstName: null,
      lastName:  null,
   });

   // adding new variable to the Person class
   Person.reopen({
      middleName: 'Smith',
   });
   document.write('Middle Name: '+Person.create().get('middleName'));
   document.write("<br>");

   // reopenClass() method for classes
   Person.reopenClass({
      //creating new function for class Person
      openClass: function() {
         return Person.create({isMan: true});
      }
   });
   document.write('isMan: '+Person.openClass().get('isMan'));
}
```

Now open the *app.js* file and add the following line at the top of the file:

```
import reopenclass from './reopenclass';
```

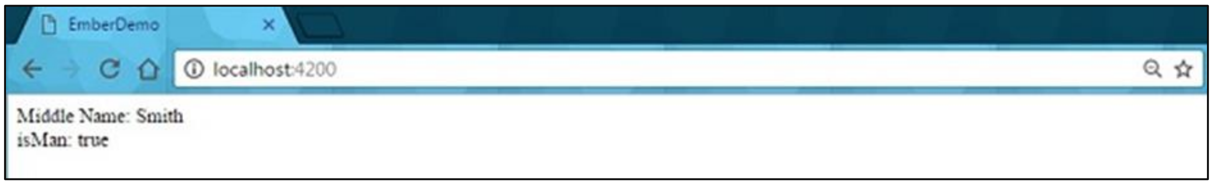
Where, reopenclass is a name of the file specified as "reopenclass.js" and created under the "app" folder.

Next call the inherited "reopenclass" at the bottom, before the export. It executes the reopenclass function which is created in the *reopenclass.js* file:

```
reopenclass();
```

Output

Run the ember server and you will receive the following output:



```
Middle Name: Smith
isMan: true
```

EmberJS – Classes and Instances

This is nothing but updating the class implementation without redefining it and reopening the class by specifying new properties in it. This is possible by using the following methods:

- **reopen()**: It adds properties and methods to *instances*.
- **reopenClass()**: It adds properties and methods to the *classes*.

Example

The example given below uses the methods mentioned above and specifies the new properties or methods in it:

```
import Ember from 'ember';
export default function() {
   // reopen() method for instances
   var Person = Ember.Object.extend({
      firstName: null,
      lastName:  null,
   });

   // adding new variable to the Person class
   Person.reopen({
      middleName: 'Smith',
   });
   document.write('Middle Name: '+Person.create().get('middleName'));
   document.write("<br>");
```

```
    // reopenClass() method for classes
    Person.reopenClass({
       //creating new function for class Person
       openClass: function() {
          return Person.create({isMan: true});
       }
    });
    document.write('isMan: '+Person.openClass().get('isMan'));
 }
```

Now open the *app.js* file and add the following line at the top of the file:

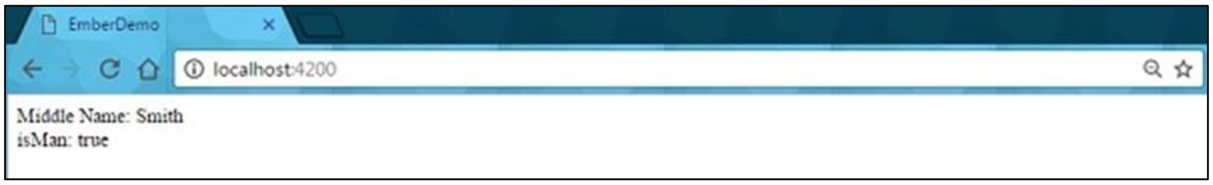
```
import reopenclass from './reopenclass';
```

Where, reopenclass is a name of the file specified as "reopenclass.js" and created under the "app" folder. Now, call the inherited "reopenclass" at the bottom, before the export. It executes the reopenclass function which is created in the *reopenclass.js* file:

```
reopenclass();
```

Output

Run the ember server and you will receive the following output:



```
Middle Name: Smith
isMan: true
```

EmberJS – Computed Properties

A computed property declares functions as properties and Ember.js automatically calls the computed properties when needed and combines one or more properties in one variable.

The following table lists down the properties of the computed property:

S.NO.	Properties & Description
1	Chaining Computed Properties The chaining computed propertiy is used to *aggregate* with one or more predefined computed properties.

2	Dynamic Updating Dynamically updates the computed property when they are called.
3	Setting Computed Properties Helps set up the computed properties by using the *setter and getter* methods.

Example

The following example adds the computed property to Ember.object and shows how to display the data:

```
import Ember from 'ember';
export default function () {
   var Car = Ember.Object.extend({
      //The values for below variables will be supplied by 'create' method
      CarName: null,
      CarModel: null,
      carDetails: Ember.computed('CarName', 'CarModel', function () {
         //returns values to the computed property function 'carDetails'
         return ' Car Name: ' + this.get('CarName') + '<br>' + ' Car Model: ' + 
this.get('CarModel');
      })
   });

   var mycar = Car.create({
      //initializing the values of Car variables
      CarName: "Alto",
      CarModel: "800",
   });
   //Displaying the information of the car
   document.write("<h2>Details of the car: <br></h2>");
   document.write(mycar.get('carDetails'));
}
```

Now open the *app.js* file and add the following line at the top of the file:

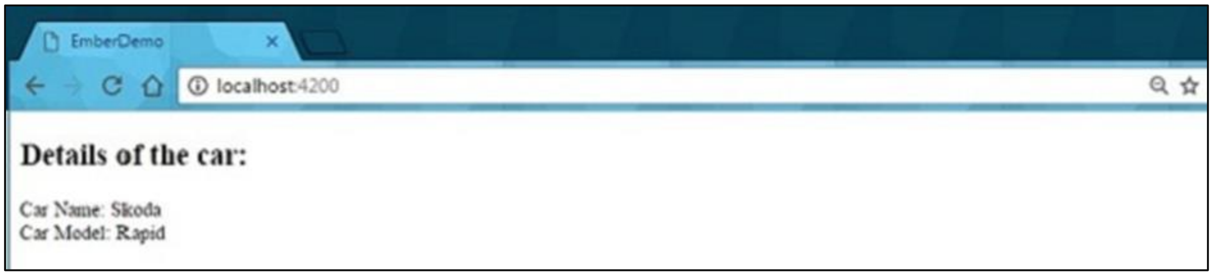
```
import computedproperties from './computedproperties';
```

Where, **computedproperties** is a name of the file specified as "computedproperties.js" and created under the "app" folder. Now, call the inherited "computedproperties" at the bottom, before the export. It executes the computedproperties function which is created in the *computedproperties.js* file:

```
computedproperties();
```

Output

Run the ember server and you will receive the following output:



Details of the car:

Car Name: Skoda
Car Model: Rapid

EmberJS – Object Model Chaining Computed Properties

The chaining computed property is used to *aggregate with one or more predefined computed properties* under a single **property**.

Syntax

```
var ClassName = Ember.Object.extend({
   NameOfComputedProperty1: Ember.computed(function() {
      return VariableName;
   }),

   NameOfComputedProperty2: Ember.computed(function() {
      return VariableName;
   });
});
```

Example

The following example shows how to use the computed properties as values to create new computed properties:

```
import Ember from 'ember';
export default function () {
   var Person = Ember.Object.extend({
      firstName: null,
      lastName: null,
      age: null,
      mobno: null,
      // Defining the Details1 and Details2 computed property function
      Details1: Ember.computed('firstName', 'lastName', function () {
         return this.get('firstName') + ' ' + this.get('lastName');
      }),

      Details2: Ember.computed('age', 'mobno', function () {
         return 'Name: ' + this.get('Details1') + '<br>' + ' Age: ' +
this.get('age') + '<br>' + ' Mob No: ' + this.get('mobno');
      }),
   });

   var person_details = Person.create({
      //initializing the values for variables
      firstName: 'Jhon',
      lastName: 'Smith',
      age: 26,
      mobno: '1234512345'
   });
   document.write("<h2>Details of the Person: <br></h2>");
   //displaying the values by get() method
   document.write(person_details.get('Details2'));
}
```

Now open the *app.js* file and add the following line at the top of the file:

```
import chainingcomputedproperties from './chainingcomputedproperties';
```

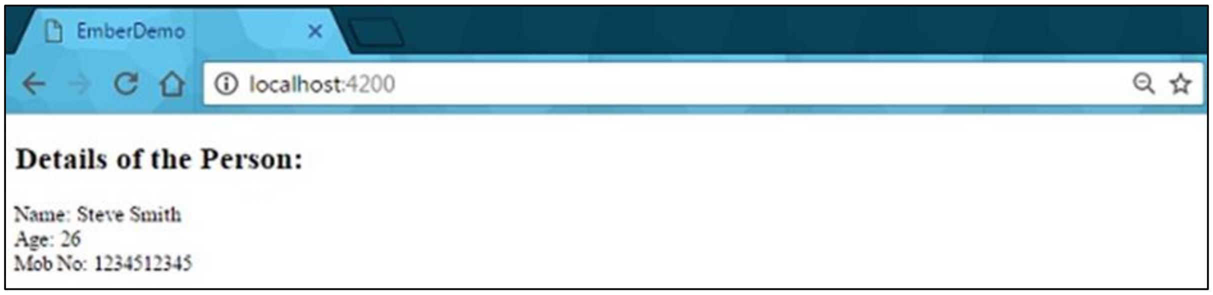
Where, **chainingcomputedproperties** is a name of the file specified as "chainingcomputedproperties.js" and created under the "app" folder.

Next call the inherited "chainingcomputedproperties" at the bottom, before the export. It executes the chainingcomputedproperties function which is created in the *chainingcomputedproperties.js* file:

```
chainingcomputedproperties();
```

Output

Run the ember server and you will receive the following output:



Details of the Person:

Name: Steve Smith
Age: 26
Mob No: 1234512345

EmberJS – Object Model Dynamic Updating

Computed properties detect the changes made on the properties and dynamically update the computed property when they are called by using the *set()* method.

Syntax

```
ClassName.set('VariableName', 'UpdatedValue');
```

Example

The following example shows dynamically updated value when changes are made to the properties:

```
import Ember from 'ember';
export default function () {
   var Person = Ember.Object.extend({
      firstName: null,
      lastName: null,
      age: null,
      mobno: null,
      //Defining the Details1 and Details2 computed property function
      Details1: Ember.computed('firstName', 'lastName', function () {
         return this.get('firstName') + ' ' + this.get('lastName');
      }),

      Details2: Ember.computed('age', 'mobno', function () {
         return 'Name: ' + this.get('Details1') + '<br>' + ' Age: ' +
this.get('age') + '<br>' + ' Mob No: ' + this.get('mobno');
      }),
   });

   //initializing the Person details
   var person_details = Person.create({
      //Dynamically Updating the properties
      firstName: 'Jhon',
      lastName: 'Smith',
      age: 26,
      mobno: '1234512345'
   });

   // updating the value for 'firstName' using set() method
   person_details.set('firstName', 'Steve');
   document.write("<h2>Details of the Person: <br></h2>");
   document.write(person_details.get('Details2'));
}
```

Now open the *app.js* file and add below line at top of the file:

```
import dynamicupdating from './dynamicupdating';
```

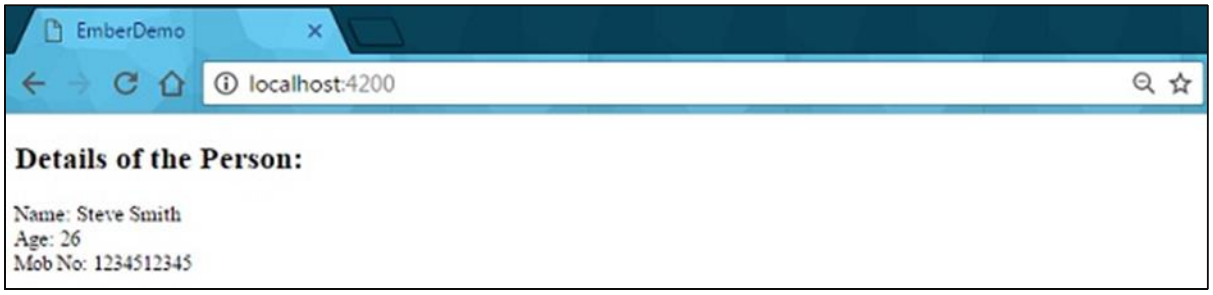
Where, dynamicupdating is a name of the file specified as "dynamicupdating.js" and created under the "app" folder.

Next call the inherited "dynamicupdating" at the bottom, before the export. It executes the dynamicupdating function which is created in the *dynamicupdating.js* file:

```
dynamicupdating();
```

Output

Run the ember server and you will receive the following output:



Details of the Person:

Name: Steve Smith
Age: 26
Mob No: 1234512345