Tuesday: Creating a class

Creating a Class

A goal has much more detail than just a name. It can have a completion date, an ID, number of participants, an owner, etc. So far, we have been creating our goals as pure strings inside our *AppComponent*. We can continue defining these goals as strings in our *AppComponent* but it will get tedious and clumsy as we create more and more goals. To curb this, angular allows us to create a class that will be the blueprint for creating goal objects.

Goal Blueprint Class

While learning Javascript in Prep, we got familiar with Object-Oriented Programming in which we build objects from classes. A **class** is an extensible program-code-template for creating objects by providing initial values for variables and member functions or methods. Let us create a class with the angular CLI to define the blueprint of a goal which is how we will be creating Goal objects. On our terminal, let's execute this command:

```
$ ng generate class Goal
```

We use the `ng generate class <class-name>` command to create a class using the angular CLI. We will create goals that have a name and an ID. Let us define this logic in the goal class we have just created.

src/app/goal.ts

```
export class Goal {
  id: number;
  name: string;
}
```

Inside the class `Goal`, we have defined the `id` and attributed it to the datatype number and `name` to be a string. We have exported the class to make it available for use anywhere else we need it in the app.

Displaying an Array

Let us create several goals using the goal blueprint class.

src/app/app.component.ts

```
import { Component } from '@angular/core';
import { Goal } from './goal';
....
export class AppComponent {
  goals:Goal[] = [
    {id:1, name:'Watch finding Nemo'},
    {id:2,name:'Buy Cookies'},
    {id:3,name:'Get new Phone Case'},
    {id:4,name:'Get Dog Food'},
    {id:5,name:'Solve math homework'},
    {id:6,name:'Plot my world domination plan'},
  ];
}
```

At the top, we have imported the `Goal` blueprint class we have just created. If we do not import it, we'll get errors in our application because the component in which we are trying to use this class does not recognize it yet. The period signs `...` are just to show that we do not change the code before the *AppComponent* class. Inside the *AppComponent* class, we have created an object `goals` and attributed it to the `Goal` blueprint and then defined the array of goals each with its ID and name.

When we check our browser now, the application is not broken but it does not show us the actual goals. It shows us the text [object Object]. This means that the browser can recognize that we want to display goal objects but our HTML code cannot display anything in specific, neither the id nor the name.

To display the goal name for each item in the list, we change our HTML template code to point us to the goal name instead of the goal object.

src/app/app.component.html

```
<div>
  <h1>My Goals</h1>
  <hr>
  <ul>
      <li *ngFor='let goal of goals'>
        {{goal.name}}
      </li>
  </ul>
</div>
```

When we check our browser, we now see the goal names in the list. For practice, display the goal ID along with the goal name.

NgIf Directive

Let's take a look at another directive in Angular known as **NgIf** directive.

src/app/app.component.html

```
<div>
  <h1>My Goals</h1>
  <hr>
  <ul>
      <li *ngFor='let goal of goals'>
        {{goal.name}}
      </li>
  </ul>
  <p *ngIf='goals.length > 5'>Your goals are too many</p>
</div>
```

We have added a <p> tag and defined the directive logic in it. Here, we check if the length of the goals array is greater than five, and when it is, we display the text in the paragraph tag. When we check our browser now, the text is displayed because we have 6 goals in the array. Delete one goal in the *src/app/app.component.ts* file and check your browser to see how this directive works. It simply checks if a certain condition is met and performs an action based on that condition.