Exercise 2 - TypeScript Refresher: Classes

Objective

To use Classes to represent our training company within a simple web application.

Overview

In this exercise you will extend a static website to contain information about various courses available within a training organisation.

Instructions

1. A starter template is already in place for you. If needs be, run npm install to download the required modules.
2. We have a simple webpage which we will use to display the various courses available at QA. Run the gulp serve task to ensure everything is ready to go. You should see a webpage with a navbar, header image and a footer. We will add the content.
3. We're looking to focus on TypeScript for this exercise so let's get the basis of our application in place. Our training organisation runs many courses, all of which have the following important information attached to them:

    a. Title
    b. ID
    c. Price
    d. Description

    Each time a course is scheduled to run, we call it an 'event' where an event has the following details:

    a. Course
    b. Instructor
    c. Date
    d. Location
    e. Capacity
    f. Delegates

    Each instructor has the following data attached to them:

    a. Firstname
    b. Lastname

   c. Courses they teach

Take some time to consider and draw the relationships between these real world 'things' making sensible assumptions about data types.

4. Now that you have an idea of how you think this system will look, let's have a go at coding it.
5. Create three classes
   a. Course
   b. Event
   c. Instructor

   Set up each class with their appropriate properties as described above.
6. Instantiate a new course, instructor and event with the following data

Course

Description	Lorem text
id	QATYPESCRIPT
price	399999
title	Programming with TypeScript

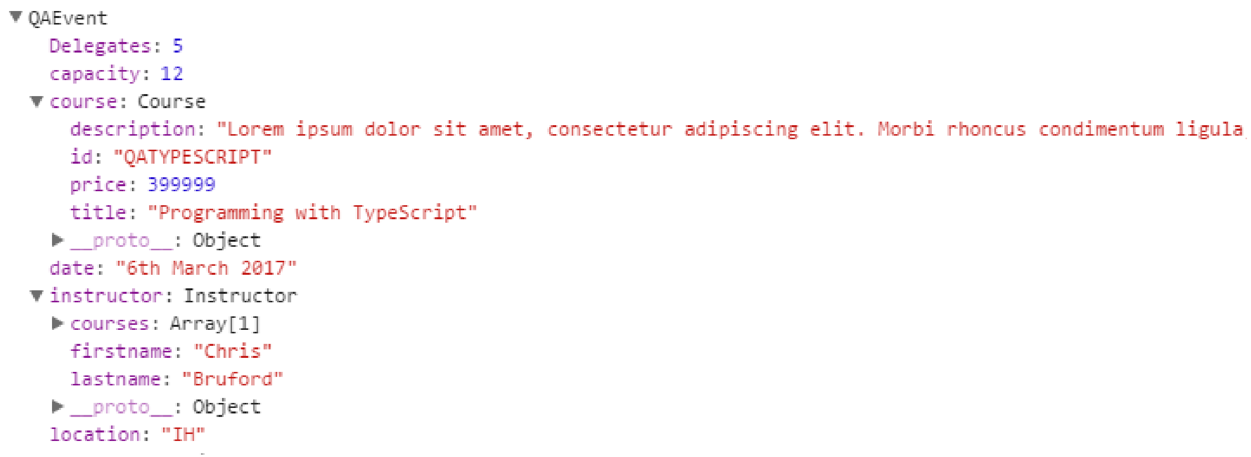
Event

delegates	5
capacity	12
course	Course
date	6th March 2017
instructor	Instructor
location	IH

Instructor

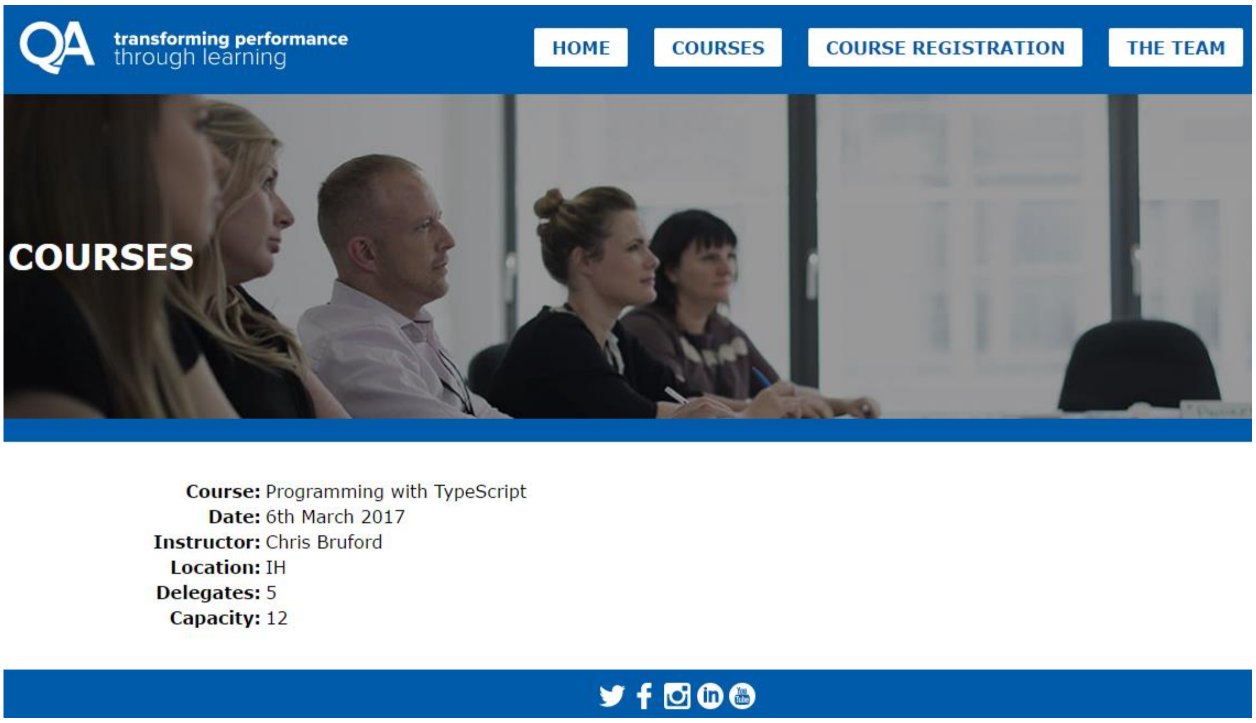
courses	Course
firstname	Chris
lastname	Bruford

7. Log the event object to the console and inspect it, all being well it should look something like this



```
▼ QAEvent
    Delegates: 5
    capacity: 12
  ▼ course: Course
      description: "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi rhoncus condimentum ligula
      id: "QATYPESCRIPT"
      price: 399999
      title: "Programming with TypeScript"
    ▶ __proto__: Object
    date: "6th March 2017"
  ▼ instructor: Instructor
    ▶ courses: Array[1]
      firstname: "Chris"
      lastname: "Bruford"
    ▶ __proto__: Object
    location: "IH"
```

8. Let's get this displayed on the website. Add a div with a data-qa-courses attribute and then select this element in your TS application, assigning it to the variable "container"
9. Add the event you've instantiated to an array of such event objects and assign it to the variable "events"
10. Write a loop which traverses your events array and for each event creates a table which will display the details of the event, and then appends it to the container. It should look something like this:



Course: Programming with TypeScript
Date: 6th March 2017
Instructor: Chris Bruford
Location: IH
Delegates: 5
Capacity: 12

11. Above the loop code you just wrote, create a new event and push it onto your events array (you can reuse the Instructor and Course objects if you wish)
12. Check that the webpage now displays 2 tables.
13. So far, we've (implicitly) created public properties on our Classes, and we're not using constructors - causing us a lot of repetition when it comes to instantiating these objects. Let's update all three classes to use constructors which take arguments for all the properties we need to set. For example, I've done Instructor for you below (you can shorten this significantly by explicitly setting access modifiers)

```
class Instructor {
  firstname: string;
  lastname: string;
  courses: Array<Course>;
  constructor(
    firstname: string,
    lastname: string,
    courses:Array<Course>
  ) {
      this.firstname = firstname;
      this.lastname = lastname;
      this.courses = courses;
  }
}
```

14. Let's refactor the code which creates the table so that it is encapsulated within a function which accepts an array of such event objects.

Implementing Inheritance

15. Modern training organisations run many different types of "events" which will differ in the data and methods they expose, and so there are numerous ways of categorising them in more useful ways. For now, we're going to consider that there are two types of training events available to our customers: classroom based learning or remote attendance, both of which will be subclasses of a QAEvent base class.
16. Start by creating an abstract class called "QAEvent" and changing our existing event class to "CBLEvent"
17. Within the abstract class define the constructor with parameters for course, instructor, date, capacity and delegates. Note we do not want location. It's also unlikely we would want to provide direct access to these properties for other parts of the application to change on a whim, nor do we wish to commit ourselves to this implementation for all time, so let's make these protected properties.
18. We now need to amend our CBLEvent class. Firstly, adjust its declaration so that it inherits from our new abstract QAEvent.
19. Your IDE is probably telling you that CBLEvent now needs to include a super call in its constructor, so add that with the appropriate parameters, and then amend the current constructor parameters so they are also protected.
20. Our createPricingTables is now broken because we are trying to access a series of protected properties. Let's fix that. Start by adding an underscore before each property name in the class: the age-old convention to describe private variables.
21. We now need to add getters for each of the properties. All of our events will have a course, instructor, data, capacity and delegates - so add the getters for these to the abstract base class, and the final property getter for 'location' within the appropriate class.
22. We still have a problem in that our location property isn't recognised as it is not a valid property on a QAEvent object. We have a few choices of how to address this, including creating separate functions to handle different types of QAEvents - but we'll look into this later. For now, let's simply hack our way out of dodge and delete the location from being referenced in the createPricingTables function at all.
23. We now need to set up the class for our remote attendance events, so create a class called RAEvent which extends QAEvent. It will need to mirror the constructor as our earlier CBLEvent does. In addition, we will add a boolean property to CBLEvent called "_vm" which will be a flag to let us know if this course is run off Virtual Machines or not. Don't forget to create a getter for it too.
24. Instantiate a RAEvent object with some dummy data and add it to the array of QAEvent objects being passed to createPricingTables.

25. Compile your ts file and run in the browser, all 3 courses should be displayed.