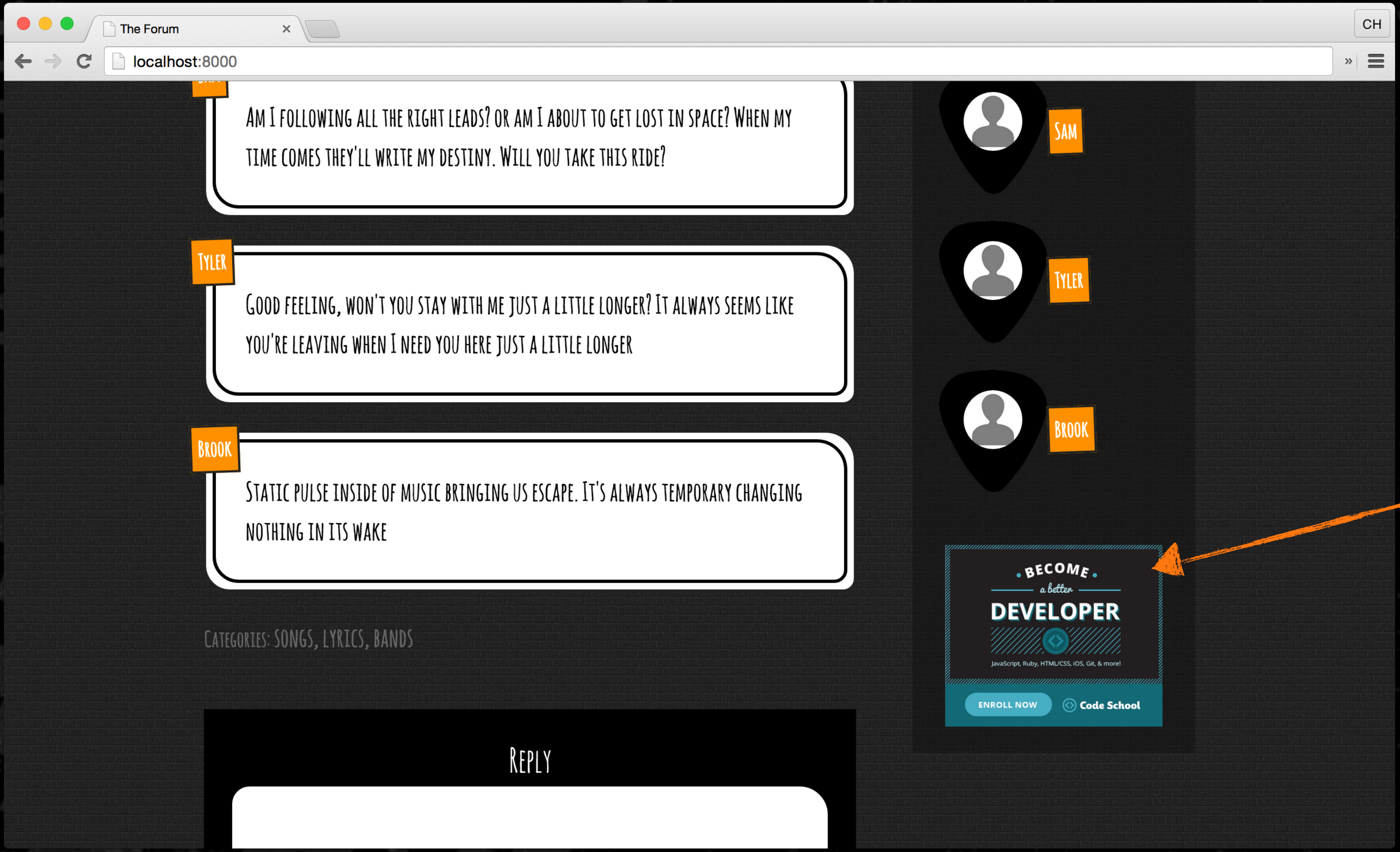
Classes

Level 5 – Section 1

Adding a Sponsor to the Sidebar

We want to add a sponsor widget to the sidebar.



AM I FOLLOWING ALL THE RIGHT LEADS? OR AM I ABOUT TO GET LOST IN SPACE? WHEN MY TIME COMES THEY'LL WRITE MY DESTINY. WILL YOU TAKE THIS RIDE?

TYLER

GOOD FEELING, WON'T YOU STAY WITH ME JUST A LITTLE LONGER? IT ALWAYS SEEMS LIKE YOU'RE LEAVING WHEN I NEED YOU HERE JUST A LITTLE LONGER

BROOK

STATIC PULSE INSIDE OF MUSIC BRINGING US ESCAPE. IT'S ALWAYS TEMPORARY CHANGING NOTHING IN ITS WAKE

CATEGORIES: SONGS, LYRICS, BANDS

REPLY

SAM

TYLER

BROOK

BECOME a better DEVELOPER

JavaScript, Ruby, HTML/CSS, iOS, Git, & more!

ENROLL NOW

Code School

Sponsor widget

Using a Function Approach

A common approach to encapsulation in JavaScript is using a **constructor function**.

```
function SponsorWidget(name, description, url){
  this.name         = name;
  this.description  = description;
  this.url          = url;
}
```

Constructor functions are invoked with the new operator

```
SponsorWidget.prototype.render = function(){
  //...
};
```

Too verbose!

Invoking the *SponsorWidget* function looks like this:

```
let sponsorWidget = new SponsorWidget(name, description, url);
sponsorWidget.render();
```

ES2015

Using the New Class Syntax

To define a class, we use the *class* keyword followed by the name of the class. The body of a class is the part between curly braces.

```
class SponsorWidget {

  render(){
    //...
  }
}
```

instance method definitions in classes look just like the method initializer shorthand in objects!

Initializing Values in the Constructor Function

The *constructor* method is a special method for **creating and initializing** an object.

```
class SponsorWidget {

  constructor(name, description, url){
    this.name        = name;
    this.description = description;
    this.url         = url;
  }

  render(){
    //...
  }
}
```

Runs every time a new instance is created with the new operator

Assigning to instance variables makes them accessible by other instance methods

Still use it just like before

```
let sponsorWidget = new SponsorWidget(name, description, url);
sponsorWidget.render();
```

Accessing Class Instance Variables

Instance variables set on the *constructor* method can be accessed from all other instance methods in the class.

```
class SponsorWidget {

  constructor(name, description, url){
    //...
    this.url = url;
  }

  render(){
    let link = this._buildLink(this.url);
    //...
  }

  _buildLink(url){
    //...
  }
}
```

Don't forget to use this to access instance properties and methods

Can access previously assigned instance variables

Prefixing a method with an underscore is a convention for indicating that it should not be invoked from the public API

Creating an Instance From a Class

The class syntax is not introducing a new object model to JavaScript. It's just **syntactical sugar** over the existing **prototype-based** inheritance.

Syntactic Sugar

```
class SponsorWidget {
  //...
}
```

Prototype Object Model

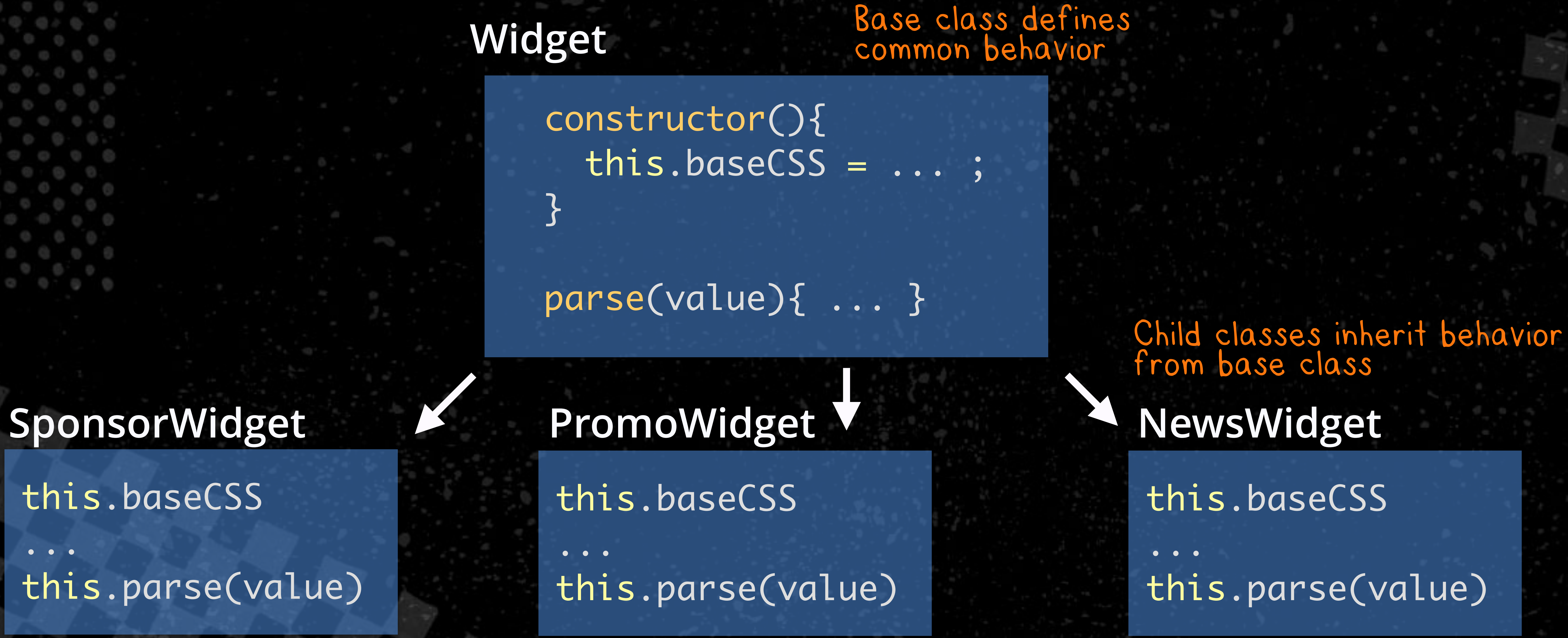
```
function SponsorWidget(name, description, url){
  //...
}
```

Instances are created the same way

```
let sponsorWidget = new SponsorWidget(name, description, url);
sponsorWidget.render();
```

Class Inheritance

We can use class inheritance to reduce code repetition. Child classes **inherit** and **specialize** behavior defined in parent classes.


Widget
Base class defines common behavior
Child classes inherit behavior from base class
SponsorWidget
PromoWidget
NewsWidget

```
constructor(){
  this.baseCSS = ... ;
}

parse(value){ ... }
```

```
this.baseCSS
...
this.parse(value)
```

```
this.baseCSS
...
this.parse(value)
```

```
this.baseCSS
...
this.parse(value)
```

Using extends to Inherit From Base Class

The *extends* keyword is used to create a class that **inherits methods and properties** from another class. The *super* method runs the constructor function from the parent class.

Parent Class

```
class Widget {

  constructor(){
    this.baseCSS = "site-widget";
  }

  parse(value){
    //...
  }
}
```

runs parent's setup code

Child Class

```
class SponsorWidget extends Widget {

  constructor(name, description, url){
    super();

    //...
  }

  render(){
    let parsedName = this.parse(this.name);
    let css = this._buildCSS(this.baseCSS);
    //...
  }
}
```

inherits methods

inherits properties

Overriding Inherited Methods

Child classes can invoke methods from their **parent** classes via the *super* object.

Parent Class

```
class Widget {

  constructor(){
    this.baseCSS = "site-widget";
  }

  parse(value){
    //...
  }
}
```

Child Class

```
class SponsorWidget extends Widget {

  constructor(name, description, url){
    super();

    //...
  }

  parse(){
    let parsedName = super.parse(this.name);
    return `Sponsor: ${parsedName}`;
  }

  render(){
    //...
  }
}
```

Calls the parent version of the parse() method