Binding events to collections

In the previous step I showed you that `fetch` accepts two callbacks: `success` and `error` and that we might use the first one to render our view after some data is loaded. However, this is not the best idea: we want to re-render the view every time the collection gets reloaded, a new item is being added or an existing one being removed. Of course, we might manually call render after each of these actions takes place, but that it really a bad idea.

Events can be bound not only to the views - collections can listen to events as well.

Open browser's console by pressing `F12` and instantiate a new collection:

```
var events = new Organizer.Events();
```

Attach an `add` event that gets fired whenever new item is added to the collection:

```
events.on('add', function(item) {console.log(item)});
```

Add some data:

```
events.add({title: 'added'});
```

Bind an event to a specific model

You can also listen for change events. For example, use the following code to be notified when title of any model inside the collection has changed:

```
events.on('change:title', function() {console.log('title in the collection has changed')});
```

We can access any model of the collection and change its title:

```
events.models[0].set('title', 'changed');
```

You can bind an event to a specific model if you wish:

```
events.models[0].on('change', function() {console.log('the model has changed');});
events.models[0].set('title', 'changed again');
```

You can listen to other events like `remove` or `reset`:

```
events.on('reset', function() { console.log('Collection was reset!'); } );
events.reset();
```

If you want an event handler to run only once, use the once method:

```
events.once('add', function() {console.log('something was added');});
events.add({title: 'added'});
events.add({title: 'added'});
```

Re-render the list view

Now let's refactor our `initialize` function. The idea is to re-render our list view whenever a collection gets reset or updated in some way. Attach collection to the global object

```
Organizer.EventsList = new Organizer.Events();
Organizer.EventsList.fetch();
var eventsList = new Organizer.EventsListView({collection: Organizer.EventsList});
```

Remove this line

```
eventsList.render();
```

and this declaration:

```
var eventsList =
```

Every view, model and collection may accept a function literal called `initialize` which gets called when it is instantiated. So this is the place where we can bind our events:

```
initialize: function() {
  this.collection.on('reset', this.render, this);
  this.collection.on('add', this.render, this);
  this.collection.on('remove', this.render, this);
},
```

Open the console and run

```
Organizer.EventsList.add({title: 'added'});
```

The view should be re-rendered.

These lines may be re-written using `listenTo` method. `listenTo` is similar to on, however that makes an object to listen for an event on another object:

```
this.listenTo(this.collection, 'reset', this.render);
```

If you need to stop listening to all events, use `stopListening` method:

```
this.stopListening();
```

If you wish to stop listening to events on specific object, pass that object as an argument:

```
this.stopListening(this.collection);
```