Eric Roberts and Jerry Cain
CS 106J

Handout #49
June 2, 2017

Adventure Workshop

Adventure Workshop + Using Interactors

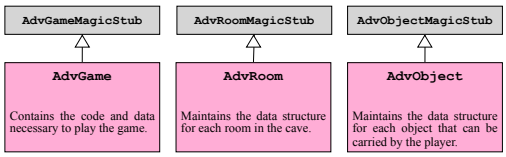
Eric Roberts and Jerry Cain
CS 106J
June 2, 2017

The Motivation for this Session

- We have seen a number of you in office hours this week, and our sense is that there is more confusion than we have seen in past years. Adventure is an old, highly successful assignment, but the JavaScript implementation is new and clearly requires more clarification than the handout provides.
- This class focuses on the following questions, which seem to be the most problematic:
  - What is the role of the magic stubs and where do they end up?
  - What is an object anyway?
  - Why are the fields stored explicitly rather than in the closure?
  - What do the methods in the starter files return?
  - How do I get data from a file into an object?
- We'll answer these questions in class today and try to address any others that you have.

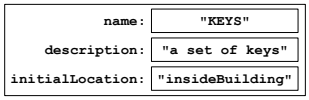
The Role of the Magic Stubs

- The purpose of the magic stubs is to ensure that Adventure works from the moment you start to code it and that it continues to work throughout the development process.
- Your goal is to make the stubs disappear.
- In your final version, there should be no calls to any of the stub methods. Instead, the factory methods should create empty objects and then add the necessary fields to them.



What Is an Object Anyway?

- Much of the confusion we have seen is simply that many of you are confused about what an object is.
- That confusion is exacerbated by the fact that JavaScript uses the same data structure to cover three distinct usage patterns:
  - Objects as ***aggregates*** or collections of data
  - Objects as ***maps*** that link keys and values
  - Objects as ***encapsulated values*** with hidden internal fields
- In thinking about the Adventure assignment, it is probably most valuable to focus on objects as aggregates. For example, each `AdvObject` is an aggregate with three fields:



Why Are the Fields Explicit?

- In class, we went to great lengths to show how JavaScript makes it possible to hide private data in the closure. We then seem to abandon that idea in Adventure.
- The reason behind this decision is that private fields stored in a closure are inaccessible to every other function, which means that the magic stub can't see them either, even though it needs access to those fields to do its job.
- In the next generation of CS 106J, we can fix this problem by changing the set of methods that the class exports so that the stub—just like any other function—can do everything it needs to do through methods.
- For now, you should keep the factory methods as they are and have the functions `readRoomsFile` and `readObjectsFile` assign values directly to individual fields.

What Do the Starter File Methods Return?

- The starter files define several methods that are implemented using functions in the `AdvMagicStub`. The comments tell you what those methods return, but here is a quick summary:
  - The `AdvGame` factory method returns an object that contains everything you need to play the game. The only method that is specified is `play`, but you will want helper methods as well.
  - The `readSynonyms` file returns a map from alternative forms of the words used in the game to their primary forms.
  - The `AdvRoom` factory method returns an empty `AdvRoom` object that defines the necessary methods.
  - The `readRoomsFile` function reads in all the rooms and returns a map from room names to `AdvRoom` objects. The map also includes a `"START"` key that indicates the first room.
  - The `AdvObject` factory method returns an empty `AdvObject`.
  - The `readObjectsFile` function reads in all the objects and returns a map from object names to `AdvObject`s.

How Do I Read Files?

- The only functions in **AdvRoom.js** and **AdvObject.js** that are at all long are **readRoomsFile** and **readObjectsFile**.
- These functions have to read a data file into an internal structure, which in both cases is a map from names to objects of the relevant type.
- These functions must include a loop that reads each entry in the file, where an entry is all the data pertaining to one room or one object. Your implementation must call the appropriate factory method (**AdvRoom** or **AdvObject**) for each entry.
- These functions are much easier to write if you use the **shift** method to read each line from an array of lines than if you try to use array indexing. The best model is the **TMCourse.js** file as it was presented in class, which is reprised on the next two slides.

Code for the TMCourse Class

```
/*
 * File: TMCourse.java
 * -------------------
 * This class defines the data structure for a course for use with
 * the TeachingMachine program.
 */

/* Constants */

const MARKER = "-----";

/*
 * Creates a new course for the teaching machine by reading the
 * data from the specified file, which consists of questions and
 * their accepted answers.
 */

function TMCourse(filename) {
   var lines = File.readLines(filename);
   if (lines === undefined) return null;
   var nLines = lines.length;
   var title = lines.shift();
   var questions = { };
```

Code for the TMCourse Class

```
   var line = lines.shift();
   while (line !== undefined) {
      var qnum = parseInt(line);
      var text = [ ];
      while (line !== undefined && line !== MARKER) {
         text.push(line);
         line = lines.shift();
      }
      var question = TMQuestion(text);
      line = lines.shift();
      while (line !== undefined && line !== "") {
         var colon = line.indexOf(":");
         var response = line.substring(0, colon).toLowerCase().trim();
         var nextQuestion = parseInt(line.substring(colon + 1).trim());
         question.addAnswer(response, nextQuestion);
         line = lines.shift();
      }
      questions[qnum] = question;
      line = lines.shift();
   }
   return {
      getTitle: function() { return title; },
      getQuestion: function(qnum) { return questions[qnum]; }
   };
}
```