Using JavaScript Objects

CMSC 122

What is an Object?

This is not such an easy question to answer!

- Think of an "object" as any "computational entity."
  - Numbers are objects.
  - Strings are objects.
  - Browsers are objects.
  - HTML Documents are objects!
- Questions that we can ask of any object:
  - What are your properties?
  - What are your methods?
  - What is your "lineage," who are your "parents?"
- For our purposes, we will only focus on the first two: methods & properties.

What are the elements of an Object?

Objects are composed of properties and methods.

- Examples of "properties"—the `length` of an array, the `innerHTML` contained by HTML elements . . . and we'll see many others.
- Examples of "methods"—functions or procedures that allow us to do things with these objects, such as the `getElementById` operator on `document`, which is JavaScript's variable for the HTML DOM object.
- We shall see many examples of both properties and methods shortly.

Distinguishing "primitive" from "first-class" objects

- For various reasons, many languages in current use distinguish "primitive" objects from "first-class" objects.
  - Numbers, such as integers and floating point numbers that we use in JavaScript, are "primitive," meaning that we can treat them as primitive data-types. They cannot be used as "supertypes" or "parent types" of other objects.
  - DOM elements, however, are first-class objects, meaning that we have access to a rich set of properties and methods (functions & procedures) as well as access to their "parent" or supertype data.
- We will focus on first-class object types.

Familiar examples . . .

Consider the following examples that we have seen in this class:

- DOM objects:

```
var output = document.getElementById( "output" );
```

- The variable "output" references an object that has certain "properties," we use one of these properties all the time:

```
output.innerHTML = "The answer is 42.";
```

- None of this makes sense unless the "document" itself was an object that had the "method" (`getElementById`) that allowed us to reference an element by name, such as "output."

Why are Objects a big-deal?

- Objects allow us to group properties and behaviors under a user-specified NAME.
- Once defined, that NAME becomes part of the language—as though it always existed.
- Thus, we build a representation within the language of the problem elements that we need for its solution!
- Others can re-use these objects (NAMEs) in their own code.

Using Objects to represent relationships

Objects are best used to associate common properties and behaviors under a single name.

- A playing card: this might be an object with two properities—a suit and a numeral. We define methods by which we might compare two cards as in a card game.
- A "deck" of Cards—this might be an array of Card objects, or another object itself, that has special "methods," such as shuffle.
- A single Die, which is a six-sided cube that contains "pips," representing the numbers 1 through 6. Rather than representing each "side," we define a "method" on the `Dice` object called `roll()`, that returns a random integer 1 through 6.

How does this look in JavaScript?

As in most languages . . . many ways of saying the same thing.

```
var person = { // using the "literal" syntax ...
    firstName : "Tom",
    lastName : "Reinhardt"
    ...
    };

var person = new Object(); // creates a new empty object
// defines and sets each property, by name
person.firstName = "Tom";
person.lastName = "Reinhardt";
 ...
```

Most references prefer the first to the second syntax.

Constructing many Objects

Often, we need to create thousands of Objects of a particular type. We do this by defining "constructors" for each Object type.

```
 function Person ( fn, ln) {
  // "this" refers to the object being constructed
    this.firstName = fn;
    this.lastName = ln;
}
var TomR = new Person( "Tom", "Reinhardt" );
var BillH = new Person( "Bill", "Henneman" );
...
```

Pay attention to the subtle introduction of a new kind of word, "this" in the example above!

Some fine points . . .

Before continuing . . . , what do we know?

- Objects roughly correspond to "nouns" in natural languages.
- Objects associate properties (attributions) and methods (behaviors) with a Name.
- Objects' properties (and methods) may be accessed through the keyword "this."
- A surprising variety of commonly used data are first-class Objects in the JavaScript language, such as strings, dates, arrays, booleans, and numbers (more on this later).

Using an Object definition to Improve a design

Before introducing additional mechanism, let's consider a well-known problem and explore how Objects might be used to improve our solution.

Revisit the Mascots Game (In-Class Activity that uses only the mechanisms discussed to this point.)

Adding Methods to Object Definitions

- Methods allow us to associate functions and procedures with Objects.
- The syntax for adding methods is similar to that for adding properties.
- We may use many standard Object methods, such as methods for String, Arrays, Dates, and others.

Adding a method to print a Person object

Consider a common use-case:

```
var TomR = new Person( "Tom", "Reinhardt");
window.alert( "Created a new person named " + TomR.firstName +
        " " + TomR.lastName);
```

Instead of "unpacking" the object's properties every time we wish to print any Person object, define a "method" called `toString()` on the Person object and use that!

Writing a method for a class of objects

```
function Person ( ln, fn ) { // added method at bottom
    this.firstName = fn;
    this.lastName = ln;
    this.toString = function () {
       return this.firstName + " " + this.lastName
       }
}
```

Look at the next slide, to see how it is used.

Calling a Method on an Object (instance)

Assuming that you added the method to the constructor on the last slide, create an object and call the method to see what it does:

```
var TomR = new Person( "Tom", "Reinhardt" );
window.alert( "Created a person named " + TomR.toString() );
```

Compare that with

```
// same variable declaration....
window.alert("Created a person named " + TomR.firstName
          + " " + TomR.lastName );
```

Bringing Objects to a Project near to you

- As we'd mentioned earlier: re-visualize the Mascots Game as a series of interactions with Objects.
- Think about what needs to be re-structured in your existing code.
- Think now about the importance of documentation!
- Think now about some of examples mentioned earlier as Objects . . . , such as PlayingCard, Deck, maybe Dice . . . ?