Jerry Cain
CS 106AJ

Handout #48S
November 28th and 29th, 2018

Solution for Section #8

Portions of this handout by Kat Gregory

Problem 1: Discussion of Different Flavors of JavaScript Objects

Under the hood, all three “flavors” of JavaScript objects are backed by the same data structure and thus look the same. However, we choose to construct and use them in different ways. Some of the defining characteristics are listed below.

Objects as Aggregates:

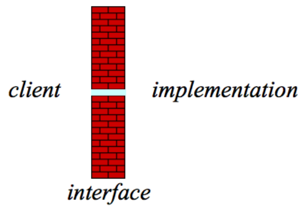
- Each object represents a collection of independent data items that are part of a single, unified structure.
- Data items within an object can be of different types.
- Data items within an object are identified by name (rather than by sequence number, like in an array).
- To access a data item, use dot-notation (**`obj.field`**) or square-bracket notation (**`obj["field"]`**).
- Aggregates can be created with JSON or via Factory functions.
- Examples from lecture: Hogwarts students, employees at Scrooge & Marley, points in a cartesian coordinate system.

Objects as Maps:

- Each map object associates pairs of data values.
- In particular, a simple data value called a *key* (usually a string) is associated with a value (which is usually larger and more complex).
- To access a value, use the square-bracket notation: **`map[key]`**.
- All values within a map are usually of the same type.
- Examples from lecture: airport code dictionary, symbol table, state codes

Objects in OOP:

- Client: Code that uses a library.
- Implentation: The code for the library itself.
- Interface: The point where the client and implementation meet, which allows the client to communicate with the implementation in select ways and acts as a barrier against all other interaction.
- Information hiding: The priniciple that each level of abstraction should hide as much complexity as possible from the layers that depend on it.
- Integration: Objects should combine the internal representation of the data with the operations that



implement the necessary behavior into an integrated whole.

- Encapsulation: Insofar as possible, objects should restrict the client's access to the internal state, mediating all communication across the interface boundary through the use of methods.
- Inheritance: Objects should be able to inherit behavior from other, more general object classes.
- Examples from lecture: Stanford Graphics Library, the encapsulated **`Point`** class, the **`Rational`** class.

Problem 2: Roman Numerals

```
function romanToDecimal(symbolToValueMap, numeral) {
   let result = 0;
   for (let i = 0; i < numeral.length; i++) {
      let currentSymbol = numeral[i];
      let currentValue = symbolToValueMap[currentSymbol];
      if (currentValue === undefined) {
        return -1; // Not a proper Roman numeral symbol
      }

      if (i < numeral.length - 1) {
         let nextSymbol = numeral[i + 1];
         let nextValue = symbolToValueMap[nextSymbol];

         if (currentValue < nextValue) {
            result += nextValue - currentValue;
            i++; // Skip next letter
         } else {
            result += currentValue;
         }
      } else { // on last symbol? no next symbol!
         result += currentValue;
      }
   }

   return result;
}
```

Problem 3: Morse Code (next page)

```
function main() {
    let letterToCodeMap = createLetterToCodeMap();
    let codeToLetterMap = createCodeToLetterMap(letterToCodeMap);
    console.log("Morse code translator");
    let consoleCallback = function(phrase) {
       if (phrase === "") {
          return; // End when user enters blank line
       }
       if (phrase[0] === "-" || phrase[0] === ".") {
          console.log(fromMorseCode(codeToLetterMap, phrase));
       } else {
          console.log(toMorseCode(letterToCodeMap,
                                              phrase.toLowerCase()));
       }
       console.requestInput("> ", consoleCallback);
    };
    console.requestInput("> ", consoleCallback);
}

function createLetterToCodeMap() {
    return {
       'a': '.-',    'b': '-...',  'c': '-.-.', 'd': '-..',  'e': '.',    'f': '..-.',
       'g': '--.',   'h': '....',  'i': '..',   'j': '.---', 'k': '-.-',  'l': '.-..',
       'm': '--',    'n': '-.',    ‘o': '---',  'p': '.--.', 'q': '--.-', 'r': '.-.',
       's': '...',   't': '-',     'u': '..-',  'v': '...-', 'w': '.--',  'x': '-..-',
       'y': '-.--',  'z': '--..'
    };
}

function createCodeToLetterMap(letterToCodeMap) {
    let codeToLetterMap = {};
    for (let letter in letterToCodeMap) {
       let code = letterToCodeMap[letter];
       codeToLetterMap[code] = letter;
    }
    return codeToLetterMap;
}

function fromMorseCode(letterToCodeMap, phrase) {
    let codes = phrase.split(" ");
    let result = "";
    for (let i = 0; i < codes.length; i++) {
       let code = codes[i];
       let letter = letterToCodeMap[code];
       if (letter !== undefined) {
          result += letter;
       }
    }
    return result.toUpperCase().trim();
}

function toMorseCode(letterToCodeMap, phrase) {
    let result = "";
    for (let i = 0; i < phrase.length; i++) {
       let letter = phrase[i];
       let code = letterToCodeMap[letter];
       if (code !== undefined) result += code + " ";
    }
    return result.trim();
}
```