Jerry Cain
CS 106AJ

Handout #42S
November 14-15, 2018

Solution for Section #7

Portions of this handout by Eric Roberts

Solution 1: Maps

```
/*
 * This main function makes a roadmap given a roads.txt file.
 * It then checks whether pairs of cities are reached in two steps
 * to test the implementation of isReachableInTwo.
 */
function Maps() {
   let callback = function(text) {
      let lines = JSFileChooser.convertToLineArray(text);
      let roadmap = {};
      populateRoadmap(roadmap, lines);
      if (isReachableInTwo("Los Angeles", "San Diego", roadmap))
         console.log("pass");
      if (isReachableInTwo("San Francisco", "Reno", roadmap))
         console.log("pass");
      if (!isReachableInTwo("San Francisco", "Salt Lake City", roadmap))
         console.log("pass");
   };
   JSFileChooser.chooseTextFile(callback);
}

/*
 * This function takes a roadmap object and an array of lines as
 * inputs. It parses the lines and populates the roadmap so that it
 * represents what cities each city is connected to.
 */
function populateRoadmap(roadmap, lines) {
   let delimiter = " - ";
   while(lines.length > 0) {
      let line = lines.shift();
      let cutoff = line.indexOf(delimiter);
      let cityOne = line.substring(0, cutoff);
      let cityTwo = line.substring(cutoff + delimiter.length);
      if (roadmap[cityOne] === undefined) {
         roadmap[cityOne] = [];
      }
      if (roadmap[cityTwo] === undefined) {
         roadmap[cityTwo] = [];
      }
      roadmap[cityOne].push(cityTwo);
      roadmap[cityTwo].push(cityOne);
   }
   // Do we need to return roadmap here? Why or why not?
   // Answer: No, roadmap is an object, thus passed by reference.
}
```

```
/*
 * This function takes two cities and a roadmap and returns
 * a boolean of whether the two cities can be reached within
 * two roads.
 */
function isReachableInTwo(currentCity, destinationCity, roadmap) {
   if (currentCity === destinationCity) return true;
   let firstLevelNeighbors = roadmap[currentCity];
   for (let i = 0; i < firstLevelNeighbors.length; i++) {
      let firstNeighbor = firstLevelNeighbors[i];
      if (firstNeighbor === destinationCity) return true;
      let secondLevelNeighbors = roadmap[firstNeighbor];
      for (let j = 0; j < secondLevelNeighbors.length; j++) {
         let secondNeighbor = secondLevelNeighbors[j];
         if (secondNeighbor === destinationCity) {
            return true;
         }
      }
   }
   return false;
}
```

Solution 2: Potions

```
/*
 * Makes a potion collection from the potionsFile.
 * Then, allows the user to enter the name of a potion.
 * The program outputs the ingredients for that potion.
 */
function Potions() {
   let fileCallback = function(text) {
      let lines = JSFileChooser.convertToLineArray(text);
      let potionCollection = PotionCollection(lines);
      let consoleCallback = function(name) {
         let potion = potionCollection.getPotion(name);
         if (potion !== null) {
            let ingredients = potion.getIngredients();
            for (let i = 0; i < ingredients.length; i++) {
               console.log(ingredients[i]);
            }
         }
         console.requestInput("Enter name of the potion: ",
                              consoleCallback);
      };
      console.requestInput("Enter name of the potion: ",
                           consoleCallback);
   }
   JSFileChooser.chooseTextFile(fileCallback);
}
```

```
/*
 * The Potion class takes the name of the potion and
 * returns an object that contains the functions to get
 * the names and ingredients of the potion, as well as
 * to add an ingredient.
 */
function Potion(name) {
   let ingredientList = [];
   return {
      getName: function() {
         return name;
      },
      getIngredients: function() {
         return ingredientList;
      },
      addIngredient: function(ingredient) {
         ingredientList.push(ingredient);
      }
   };
}

/*
 * The PotionCollection class an array of lines of a potions file.
 * It adds all the potions in the file into a PotionCollection
 * and returns an object that contains the functions to get
 * a potion or get all potion names.
 */
function PotionCollection(lines) {
   if (lines === undefined) return null;
   let collection = {};

   // Add potions from file
   while (lines.length > 0) {
      let name = lines.shift();
      let potion = Potion(name);
      while (lines.length > 0) {
         let line = lines.shift();
         if (line === "") break;
         potion.addIngredient(line);
      }
      collection[name] = potion;
   }

   return {
      getPotion: function(name) {
         if (collection[name] === undefined) return null;
         else return collection[name];
      },
      getPotionNames: function(name) {
         let keys = [];
         for (let key in collection) {
            keys.push(key);
         }
         return keys;
         // Ignore the order in which they appeared in file.
      }
   };
}
```