



University of Maryland College Park

Department of Computer Science

CMSC335 Spring 2023

Exam #2

FIRSTNAME, LASTNAME (PRINT IN UPPERCASE):

STUDENT ID (e.g., 123456789):

Instructions

- This exam is a closed-book, closed-notes exam with a duration of 75 minutes and 200 total points.
- You may lose credit if you do not follow the instructions below.
- **At this point, you must write your name and id at the top of this page and add your directory id (e.g., terps) at the end of odd-numbered pages.**
- Please use a pencil to answer the exam.
- **Do not remove the exam's staple or bend any of the pages, as doing so will interfere with the scanning process.**
- Provide answers in the rectangular areas. If you continue a problem on another page(s), make a note. **For multiple-choice questions, please fill in the bubble (do not circle).**
- For multiple-choice questions, you can assume only one answer unless stated otherwise.
- **Your code must be efficient and as short as possible.**
- **Provide your answer within the provided box.**
- You don't need to use meaningful variable names; however, we expect good indentation.
- You must stop writing once the time is up.

Grader Use Only

Problem #1 (Miscellaneous)	30	
Problem #2 (Array Methods)	38	
Problem #3 (Custom Type Definition)	48	
Problem #4 (Class Declaration using "class")	48	
Problem #5 (Diagram)	10	
Problem #6 (Form/JS)	26	
Total	200	

Problem #1 (Miscellaneous)

1. (4 pts) Complete the second let declaration below using array destructuring to produce the average of the first and last array elements.

Average: 90

```
let scores = [80, 35, 45, 100];
```

```
let  = scores;
```

```
document.writeln("Average: " + (first + last) / 2);
```

2. (4 pts) Complete the second let declaration below using object destructuring to produce the following output:

Model: turbo, Revolutions: 10000

```
let engine = {model: "turbo", year: 4000, revolutions: 10000};
```

```
let  = engine;
```

```
document.writeln(`Model: ${model}, Revolutions: ${revolutions}`);
```

3. (4 pts) Using the spread operator, complete the assignment below so we can generate the following output:

John,Kelly,40,50

```
let names = ["John", "Kelly"];
```

```
let ages = [40, 50];
```

```
let merged = 
```

```
document.writeln(merged);
```

4. (6 pts) Write the **JSON** (not a JavaScript object) representation of an object that has the following properties:

- a. **gate** property with a value of "21A"
- b. **nonstop** property with a value of true
- c. **duration** property with a value of 1.5

5. (5 pts) Using the => operator, initialize the **perimeter** variable with a function that computes a rectangle's perimeter. The perimeter is the sum of the length and width. The default values for length and width are four and five, respectively.

```
let perimeter = 
```

6. (7 pts) Complete the definition of the following **Error** type using the JavaScript class approach. **Using any other approach will receive no credit.**

```
class DataRangeError
```

Problem #2 (Array Methods)

A **files** array keeps track of files. The following is an example of some entries the array could have:

```
const files = [
  {name: "BobsResume", type: "docx", size: 100},
  {name: "RachelsResume", type: "docx", size: 300},
  {name: "Assignment1", type: "txt", size: 40},
  {name: "Project2", type: "txt", size: 100},
  {name: "Compiler", type: "docx", size: 700},
  {name: "Quiz1", type: "md", size: 40},
  {name: "Final", type: "md", size: 500}
];
```

To answer the following questions, you may only use the following functions: **filter, forEach, some, find, join, findIndex, reduce, and arrow (=>) functions**. You may **NOT** use iteration statements or conditionals (unless specified otherwise). Your code should work with different data (not just the entries above).

1. (6 pts) Complete the following statement so each file's **name** and **type** are printed on separate lines, using `document.writeln()`.

files.

2. (8 pts) Complete the following statement so the **name** of files with a size of 100 is printed on separate lines using `document.writeln()`.

files.

3. (8 pts) Complete the following statement so **hasAtLeastATxtFileWith40** is initialized to true if there is at least one file with a "txt" type and a size of 40.

const hasAtLeastATxtFileWith40 = files.

4. (8 pts) Complete the following statement so **findMDFile** is initialized with the name of a file with an "md" type and a size of 500. If no such file exists, **findMDFile** will be undefined.

const findMDFile = files.

5. (8 pts) Complete the following assignment so the file with the largest size is assigned to the **largestFile** variable. For example, for the above array, printing `largestFile.name` will display "Compiler". **Using conditionals (e.g., if) is OK.**

const largestFile = files.

DirectoryId (e.g., terps):

Problem #3 (Custom Type Definition)

Write **JavaScript** that defines two classes (**Bed** and **WaterBed**) using the "Default Pattern for Custom Type Definition" presented in lecture. **If you use E6 class definitions (similar to what you have in Java, where we use class and extends), you will not receive any credit for this problem.**

1. Bed

- Define a custom type with two instance variables named **model** and **weight** (not private).
- Define a constructor that has two parameters: **model** and **weight**, and initializes the appropriate instance variables.
- Define a method named **setModel** that will update the **model** instance variable with the specified parameter. The parameter has a default value of "BASIC".
- Define a method called **info** that **returns** a string with the **model** and **weight** (see the example below for format information).
- Your implementation must be efficient (i.e., do not create unnecessary objects).**

The following is an example of using the custom types you need to define.

Driver

```
let model = "ExtraSoft", weight = 500;
const bed1 = new Bed(model, weight);
document.writeln(`Bed: ${bed1.info()}<br>`);
bed1.setModel("Premium");
document.writeln(`Bed after setModel: ${bed1.info()}<br>`);
document.writeln("=====<br>");
let model2 = "ocean", weight2 = 1000, gallons = 74;
const waterBed = new WaterBed(model2, weight2, gallons);
document.writeln(`WaterBed: ${waterBed.info()}<br>`);
waterBed.setModel("megaOcean");
document.writeln(`WaterBed after setModel: ${waterBed.info()}<br>`);
document.writeln(`WaterBed gallons: ${waterBed.getGallons()}<br>`);
let bed3 = new Bed("last", 999);
bed3.setModel();
document.writeln(`last bed ${bed3.info()}`);
```

Output

```
Bed: Model: ExtraSoft, Weight: 500
Bed after setModel: Model: Premium, Weight: 500
=====
WaterBed: Model: ocean, Weight: 1000
WaterBed after setModel: Model: megaOcean, Weight: 1000
WaterBed gallons: 74
last bed Model: BASIC, Weight: 999
```

2. WaterBed

- Define an **WaterBed** custom type that "extends" the **Bed** custom type. The type has an instance variable named **gallons**; this instance variable is not private.
- Define a constructor that has **model**, **weight**, and **gallons** as parameters. The constructor will initialize the corresponding instance variables.
- Define a method named **getGallons** that returns the gallons.
- Your implementation must be efficient (i.e., do not create unnecessary objects).**

If you use E6 class definitions (similar to what you have in Java, where we use class and extends), you will not receive any credit for this problem.

DirectoryId (e.g., terps):

Problem #4 (Class Declaration using "class")

Write **JavaScript** that defines two classes (**Door** and **ElectricDoor**) using E6 class definitions (using **class**, **extends**, and **super**, etc. as in Java). **You will not get any credit if you use the "Default Pattern for Custom Type Definition" presented in class.**

1. Door

Define a **Door** class with the specifications below. A door is associated with a **make** and an **area**.

- A **private** static field named **totalDoors** initialized to 0.
- Two **private instance** variables called **make**, and **area**. You must use the approach described in the lecture to make them private.
- Define a constructor that has two parameters: **make** and **area**. The constructor will initialize the corresponding instance variables, and increase the **totalDoors** static variable.
- Define a **non-static** method called **info()** that **prints** (using `document.writeln`) the **make**, and **area**. See the sample driver for format information.
- Define the equivalent of the `toString()` Java method. The method will return a string with the **make** and **area** values separated by a comma. The driver we provided has an example of using this method (look for **toString() output**).
- Define a setter method (using **set**) that has as parameter a make value and will update the **make** instance variable only if the parameter value is NOT null (otherwise, no change will occur). See the driver for an example of how we can use this method.
- Define a getter method (using **get**) that returns the **make** value (see driver for an example of how we can use it).
- Define a static method called **getTotalDoors()** that returns the total number of door objects created.

2. ElectricDoor

The **ElectricDoor** class extends the **Door** class, and it is associated with a voltage value. Define the **ElectricDoor** class with the specifications below.

- A **private** instance variable named **voltage**. You must use the approach described in the lecture to make it private.
- Define a constructor with three parameters: **make**, **area**, and **voltage**. The constructor will call the base class constructor and initialize the **voltage** instance variable with the corresponding parameter.
- Define a **non-static** method called **info()** that calls the base class **info()** method and then prints the **voltage** value using `document.writeln`. See the sample driver for format information.

The following is an example of using the classes you need to define.

<u>Driver</u>	<u>Output</u>
<pre>let make = "DoorDepot", area = 6; const door = new Door(make, area); door.info(); document.writeln(`
Door object: \${door}
`); door.make = "TerpDepot"; document.writeln(`Door object: toString() output: \${door}
`); document.writeln(`Door make: \${door.make}`); let eMake = "ElectricDist", eArea = 40, voltage = 220; let electricDoor = new ElectricDoor(eMake, eArea, voltage); document.writeln(`
Electric Door object:
`); electricDoor.info(); document.writeln(`
Number of Doors:`); document.writeln(Door.getTotalDoors());</pre>	<pre>Make: DoorDepot, Area: 6 Door object: DoorDepot, 6 Door object: toString() output: TerpDepot, 6 Door make: TerpDepot Electric Door object: Make: ElectricDist, Area: 40 ,Voltage: 220 Number of Doors: 2</pre>

PROVIDE YOUR CODE ON THE NEXT PAGES

Page for Door Class

DirectoryId (e.g., terps):

Problem #5 (Diagram)

The **Plane** function is defined as follows:

```
function Plane(gate) {  
  this.gate = gate;  
}
```

Draw a diagram illustrating the objects and the relationships among the objects present after the following two **Plane** objects are created. Please make sure you label prototype objects as such (e.g., `Plane.prototype`). In your diagram, we expect to see the **prototype** and **__proto__** properties (and the objects they refer to). Add the **gate** property to the appropriate objects.

```
let local = new Plane("MD");  
let abroad = new Plane("Italy");
```



DirectoryId (e.g., terps):

Problem #6 (Forms/JS)

Under the comments "You must implement/complete" provide code that will complete the functionality of a form that computes the square of the value provided in the text field. The function **computeSquare()** is called when a button labeled "Print Square" is clicked on. This function will display the square result inside the `<div></div>` provided. The text field has a default value of 5. You can add additional functions if you think it is necessary. The following is an example where the user entered 5 and clicked the "Print Square" button.

Value:

Result
25

```
<!--You must implement/complete -->
```

```
Value: <input type="text"
<input type="button" value="Print Square"
```

```
<br><strong>Result</strong><br>
<div id="display"></div>
```

```
<script>
```

```
function computeSquare() {
    // You must implement/complete
```

```
</script>
```

EXTRA PAGE IN CASE IT IS NEEDED (DO NOT REMOVE)

DirectoryId (e.g., terps):

EXTRA PAGE IN CASE IT IS NEEDED (DO NOT REMOVE)

LAST PAGE