University of Maryland College Park
Department of Computer Science
CMSC335 Spring 2022
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.
- **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 exam pages. Also, do not bend any pages, as this will interfere with the scanning process.**
- Provide answers in the rectangular areas. Make a note if you continue a problem on another page(s).
- **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.**
- You don't need to use meaningful variable names; however, we expect good indentation.
- **You must stop writing once the time is up; otherwise, you will lose significant credit.**

Grader Use Only

Problem #1 (Miscellaneous)	34	
Problem #2 (Array Methods)	30	
Problem #3 (Custom Type Definition)	50	
Problem #4 (Class Declaration using "class")	50	
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 so we can produce the following output:

First: Mike, Second: Tom

```
let names = ["Mike", "Tom", "John"];

let [                                                    ] = names;

document.writeln(`First: ${first}, Second: ${second}`);
```

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

Name: photoApp, Developed: 2020

```
let app = {name: "photoApp", developed: 2020, location: "CP"};

let {                                                    } = app;

document.writeln(`Name: ${name}, Developed: ${developed}`);
```

3. (4 pts) The **repairCar** function has the following prototype: **function repairCar(carTag, speakersBrand, howMany, installer)** Write a function call that will use the spread operator and the array **["JCAudio", 3]** to initialize the **speakersBrand** and **howMany** parameters. You can assume the **carTag** and **installer** parameters are "XX123", and "Hector", respectively.

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

   a. **city** property with a value of "Bethesda"
   b. **pop** property with a value of 6500
   c. **train** property with a value of true

5. (6 pts) Using the => operator, initialize the variable **process** with a function that takes two parameters **a** and **b**. The second parameter has a default value of 5. The function will use document.writeln() to print the product of **a** and **b,** and console.log() to print the sum of **a** and **b**. For example, process(2, 7) will print to the webpage 14 and 9 to the console; process(3) will print 15 to the webpage and 8 to the console.

let process =

6. (10 pts) Define an **Error** type called **DataRangeError**. The following is an example of using your error type.

```
try {
   let value = Number(prompt("Enter value between 2 and 100"));
   if (value < 2 || value > 100) {
      throw new DataRangeError("Out of range", value);
   } else {
      document.writeln(`Square root ${Math.sqrt(value)}`);
   }
} catch (error) {
    alert(error.message + ", " + error.value);
}
```

Problem #2 (Array Methods)

A **projects** array keeps track of programming projects for a class. The following is an example of some entries the array could have:

```
const projects = [
   {name: "Btree", language: "Java", lines: 200},
   {name: "LinkedList", language: "C", lines: 150},
   {name: "Viewer", language: "PHP", lines: 300},
   {name: "Chess", language: "Java", lines: 100},
   {name: "Cards", language: "JavaScript", lines: 300},
   {name: "Simulator", language: "PHP", lines: 500}
];
```

To answer the following questions, you may not use any for/while/do-while loops and only the following functions (otherwise, you will not get credit): filter, forEach, some, find, join, findIndex. Also you may only use => functions. Your code should work with different data (not just the entries shown above).

1. (6 pts) Complete the following statement, so each project's **name** and **language** is printed on separate lines, using document.writeln.

projects.

2. (8 pts) Complete the following statement so the **language** of projects with less than 200 **lines** is printed on separate lines using document.writeln(). The same language can appear multiple times.

projects.

3. (8 pts) Complete the following statement so **hasAtLeastAJavaProjectWith100** is initialized to true if there is a least one Java project with 100 lines of code, and false otherwise.

const hasAtLeastAJavaProjectWith100 = projects.

4. (8 pts) Complete the following statement so **findProject** is initialized with the name of a project that uses PHP as the language and has 500 lines. If no such project exists, the findProject will be initialized with undefined.

const findProject = projects.

DirectoryId (e.g., terps):

Problem #3 (Custom Type Definition)

Write **JavaScript** that defines two classes (**Photo** and **DigitalPhoto**) 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. **Photo**
   a. Define a custom type with two instance variables named **date** and **description** (not private).
   b. Define a constructor that has two parameters: **date** and **description,** and initializes the appropriate instance variables.
   c. Define a method named **setDate** that will update the **date** instance variable if the parameter is different than null; otherwise, the date will be set to "NONE."
   d. Define a method called **details** that returns a string with the **date** and **description** (see the example below for format information).
   e. **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 description = "Beach trip", date = "12/01/2010", bytes = 40000;
const beachPhoto = new Photo(date, description);
document.writeln("<br>=== Photo ===<br>");
document.writeln(beachPhoto.details() + "<br>");
beachPhoto.setDate("05/03/2017");
document.writeln("After setDate<br>");
document.writeln(beachPhoto.details() + "<br>");

document.writeln("<br>=== Digital Photo ===<br>");
const mountainDigitalPhoto = new DigitalPhoto("01/01/2000", "Mountain Trip",
50000);
document.writeln(mountainDigitalPhoto.details() + "<br>");
document.writeln("Bytes: " + mountainDigitalPhoto.getBytes() + "<br>");
```

Output

```
=== Photo ===
Date: 12/01/2010, Description:
Beach trip
After setDate
Date: 05/03/2017, Description:
Beach trip

=== Digital Photo ===
Date: 01/01/2000, Description:
Mountain Trip
Bytes: 50000
```

2. **DigitalPhoto**

a. Define an **DigitalPhoto** custom type that "extends" the **Photo** custom type. The type has an instance variable named **bytes**; this instance variable is not private.
b. Define a constructor that has **date**, **description**, and **bytes** as parameters. The constructor will initialize the corresponding instance variables.
c. Define a method named **getBytes** that returns the bytes.
d. **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 (**Game** and **VideoGame**) 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. **Game**

   Define a **Game** class with the specifications below. A game is associated with a **name**, a number of **players** and a **score**.

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

2. **VideoGame**

   The **VideoGame** class extends the **Game** class, and it is associated with a video card. Define the **VideoGame** class with the specifications below.

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

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

Driver

```
document.writeln("** Game **<br>");
let name = "Chess", players = 2;
let chessGame = new Game(name, players);
chessGame.info();
document.writeln(`<br>toString() output: ${chessGame}`);
chessGame.score = 100;
document.writeln(`<br>After score change ${chessGame.score}`);

document.writeln("<br><br>** VideoGame **<br>");
let videoGameName = "MarioJr", videoGamePlayers = 1, videoCard = "NFast";
let videoGame = new VideoGame(videoGameName, videoGamePlayers, videoCard);
videoGame.score = 50;
videoGame.info();
document.writeln(`<br>Total games: ${Game.getTotalGames()}<br>`);
```

Output

```
** Game **
Game Name: Chess, Players: 2,
Score: 0
toString() output: Chess, 0
After score change 100

** VideoGame **
Game Name: MarioJr, Players: 1,
Score: 50 , VideoCard: NFast
Total games: 2
```

PROVIDE YOUR CODE ON THE NEXT PAGES

Page for Game Class

DirectoryId (e.g., terps):

Page for VideoGame Class

Problem #5 (Diagram)

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

```
function Road(surface) {
   this.surface = surface;
}
```

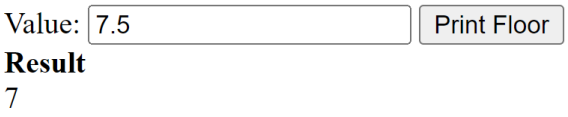
Draw a diagram illustrating the objects and the relationships among the objects present after the following two **Road** objects are created. Please make sure you label prototype objects as such (e.g., Road.prototype). In your diagram, we expect to see the **prototype** and **__proto__** properties (and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

```
let roadOne = new Road("asphalt");
let roadTwo = new Road("concrete");
```

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 floor of the value provided in the text field. The function **computeFloor()** is called when a button labeled "Print Floor" is clicked on. This function will display the floor result inside the we provided. The text field has a default value of 4.5. Use Math.floor() to compute the floor value. You can add additional functions if you think it is necessary. The following is an example where the user entered 7.5 and clicked the "Print Floor" button.




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

Value: <input type="text"
<input type="button" value="Print Floor"

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


<script>



    function computeFloor() {
      // You must implement/complete









  </script>
```

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

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

LAST PAGE