



Topic	Structuring before Coding	
Class Description	Students design a form using p5 dom to allow players to login and log the player names to the database. The gamestate and the player count are also logged. Students use the OOPs programming style to write the code.	
Class	C36	
Class time	45 mins	
Goal	<ul style="list-style-type: none"> • Create a form to log the players' name in the game. • Update playerCount and gameState in the database. • Use OOPs programming style. 	
Resources Required	<ul style="list-style-type: none"> • Teacher Resources <ul style="list-style-type: none"> ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen • Student Resources <ul style="list-style-type: none"> ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen 	
Class structure	Warm Up Teacher-led Activity Student-led Activity Wrap up	5 mins 5 min 25 min 5 mins
CONTEXT <ul style="list-style-type: none"> • Players from different computers should be able to login to the game and their names should be logged in the database. 		
Class Steps	Teacher Action	Student Action
Step 1: Warm Up (5 mins)	Can you recall what we did in the last class?	ESR: - We built the Wrecking ball simulation while reviewing

		<p>the concepts from Angry Bird Game.</p> <ul style="list-style-type: none"> - We controlled a ball so that its movement is synchronized over different browsers on different systems.
	<p>So, you know how to:</p> <ul style="list-style-type: none"> - create a remote database. - read values from a database. - write to a database. <p>Let's quickly review how we did this in the last class.</p>	<p><i>Student reviews the code from the last class on how to:</i></p> <ul style="list-style-type: none"> - <i>create a remote database using firebase console.</i> - <i>how to read values from a database by setting up a listener function.</i> - <i>how to write values to a database.</i>
	<p>I have an exciting quiz question for you! Are you ready to answer this question?</p>  <p>Teacher click on the button on the bottom right corner of your screen to start the In-Class Quiz.</p> <p>A quiz will be visible to both you and the student.</p> <p>Encourage the student to answer the quiz question.</p> <p>The student may choose the wrong option, help the student to think correctly about the question and then answer again.</p>	<p>ESR: Yes</p>

	<p>After the student selects the correct option, the  button will start appearing on your screen.</p> <p>Click the End quiz to close the quiz pop-up and continue the class.</p>	
	<p>Reading from and writing to a database is very important for a multiplayer game.</p> <p>In this class, we will continue to work towards designing a multiplayer car racing game.</p> <p>In our multiplayer game, we need to ask users to log in with their names. Those names need to be registered in the database and new players with these names need to be created.</p> <p>You will be doing this in today's class.</p>	-
Teacher Initiates Screen Share		
<p style="text-align: center;"><u>CHALLENGE</u></p> <ul style="list-style-type: none"> Use p5 dom to create a login form for players to log in. 		
<p>Step 2: Teacher-led Activity (5 min)</p>	<p>We need to create some sort of a form where different users can log in their name and get into the game.</p> <p>Everytime a new user logs in, a new Player should be created.</p>	

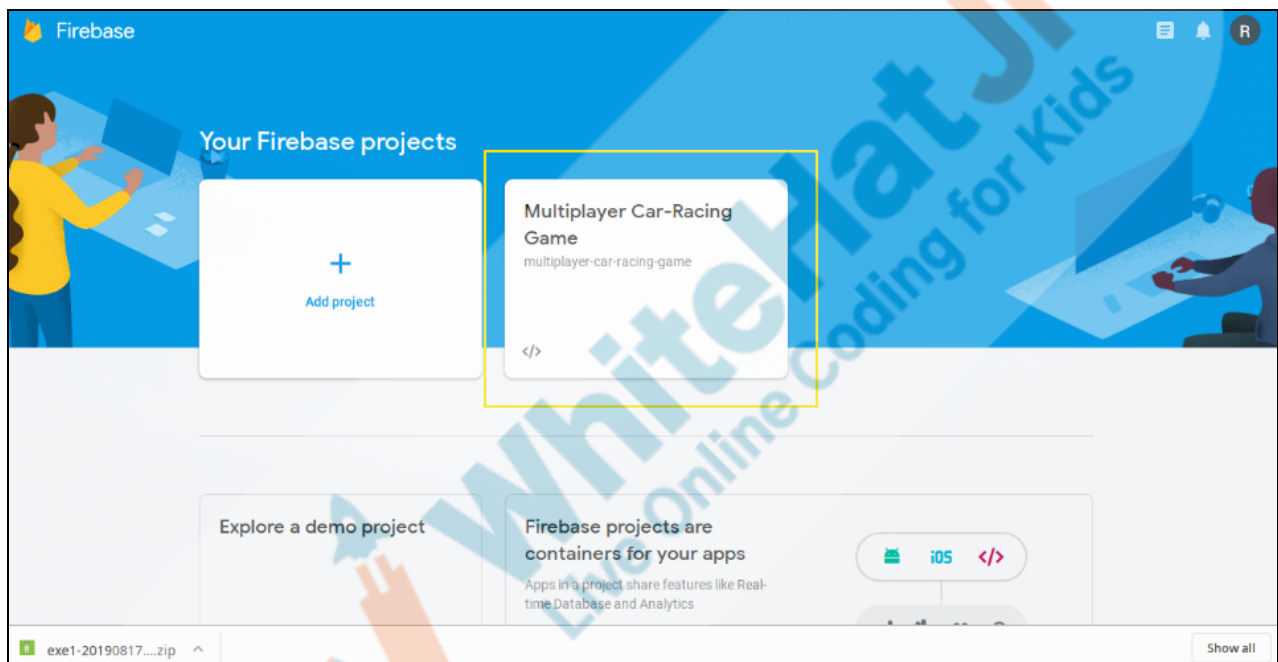
	<p>We also need to keep account of the number of players in the game and the game state.</p> <p>For example, when the game state is 0 (WAIT), we want the players to see the login form where they register their name as players.</p> <p>Let's say we make a 4-player game. When the number of registered players reaches 4, we want the game state to become 1 (PLAY). When the game state changes to 1, we would like the game to start.</p> <p>Any ideas on how to do this?</p>	<p>ESR: varied</p>
	<p>There are a number of ways in which we can go about doing this.</p> <p>We can start writing the code immediately. But good programmers, before writing code, think about how to structure their code.</p> <p>Which programming style are we using in our codes so far?</p>	<p>ESR: OOPs - object oriented style</p>
	<p>For this small part of our game, where are we asking the players to login, what are the different objects that can be in our game? What will their properties and functions be?</p>	<p>ESR: varied</p>
	<p>We need to have at least 3 objects-</p>	<p><i>Student listens and asks questions.</i></p>

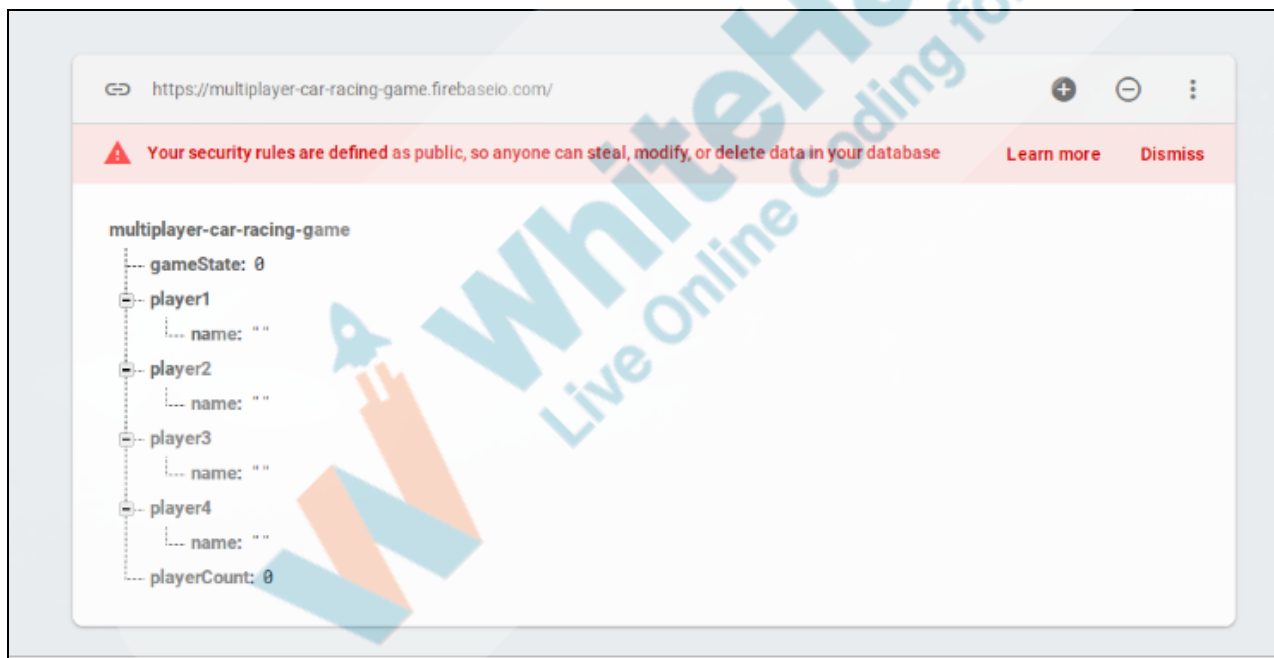
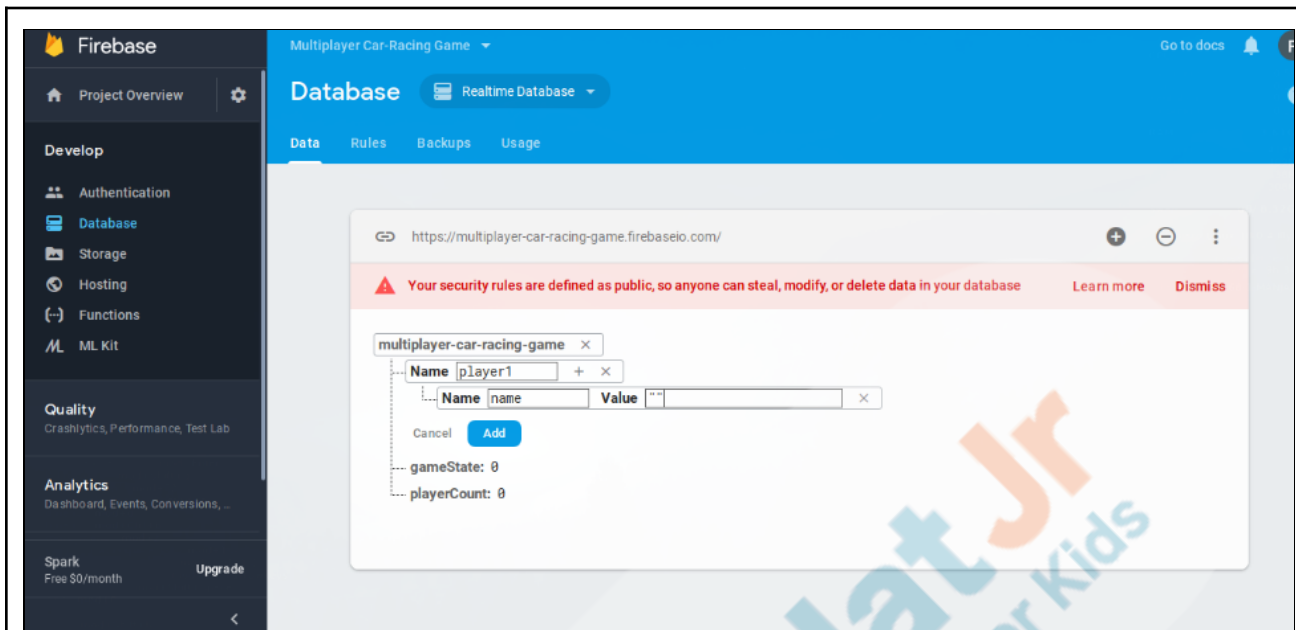
	<ol style="list-style-type: none">1. Form: Form should contain the input box and a button to log in.<ul style="list-style-type: none">• When the button is pressed, the player's name should be registered in the database and a new player should be created.2. Player: A new player object should be created every time a new user logs in. It should contain all the information about the player - name, position in the game etc.<ul style="list-style-type: none">• For now it can just have the name property. It should also be able to read and write player information to the database - for example player count or player name.3. Game Object: Game object should be able to hold the state of the game. It should be able to display form when the game state is 0(WAIT) or the game when the game state is 1(PLAY) or leaderboard when the game state is 2(END).<ul style="list-style-type: none">• For now, we will only consider the case when the game state is 0.	
--	---	--

	<p>Now that we know how the basic structure of our program will be, it will become fairly easy to write our code! Without this structure, writing code can appear complex.</p> <p>With this guide in mind, why don't you start with the coding exercise. I will be guiding you in the exercise.</p>	<i>Student shares the screen, fires up the editor and prepares to code.</i>
Teacher Stops Screen Share		
	<p>Now it's your turn. Please share your screen with me.</p>	
<ul style="list-style-type: none"> • Ask Student to press ESC key to come back to panel • Guide Student to start Screen Share • Teacher gets into Fullscreen 		
<p style="text-align: center;">ACTIVITY</p> <ul style="list-style-type: none"> • Use p5 dom to create a login form for players to log in. 		
Step 3: Student-Led Activity (25 min)	<p><i>Guide the student to open the previous class project and use it as a boilerplate. The student can clear the sketch.js file.</i></p>	<p><i>Student opens the code from the previous class and clears the sketch.js file.</i></p> <p><i>Alternatively, the student can clone Student Activity 1.</i></p>
	<p>Let's modify our database.</p> <p>Guide the student to login to the console.firebase.google.com and modify the previous database to create a new database structure equivalent to the following:</p> <pre>{ gameState: 0,</pre>	<p><i>The student logs in to the firebase console and creates the database structure.</i></p>

```
playerCount: 0,  
player1: {name: ""},  
player2: {name: ""},  
....  
}
```

The firebase config files will remain the same since we are not changing the database.





Let's create a new folder in our directory called js. This will contain the blueprint of all the 3 objects in our game - Game, Form and Player.

Let's create files for these and include them in the index.html file.

Student creates a js folder inside the current working directory.

The student then creates Game.js, Form.js and Player.js - where they will be creating the blueprints

for these objects.
The student includes these files in the index.html.

```

File Edit Selection View Go Debug Terminal Help
EXPLORER JS Form.js JS Player.js JS Game.js index.html x
OPEN EDITORS
  JS Form.js js
  JS Player.js js
  JS Game.js js
  X index.html
  images
  js
    JS Form.js
    JS Game.js
    JS Player.js
  index.html
  p5.play.js
  sketch.js
  style.css
OUTLINE
  html
  body
  script
  head

index.html
  15 <script>
  16 // Your web app's Firebase configuration
  17 var firebaseConfig = {
  18   apiKey: "AIzaSyBYV9KwLjd-8zIRsYSLGiv2zBX4MhKNAo8",
  19   authDomain: "multiplayer-car-racing-game.firebaseio.com",
  20   databaseURL: "https://multiplayer-car-racing-game.firebaseio.com",
  21   projectId: "multiplayer-car-racing-game",
  22   storageBucket: "",
  23   messagingSenderId: "936147099930",
  24   appId: "1:936147099930:web:dba47c5bb648f4ef"
  25 };
  26 // Initialize Firebase
  27 firebase.initializeApp(firebaseConfig);
  28 </script>
  29
  30 <script src="js/Player.js"></script>
  31 <script src="js/Form.js"></script>
  32 <script src="js/Game.js"></script>
  33 <link rel="stylesheet" type="text/css" href="style.css"/>
  34 </head>
  35 <body>
  36   <script src="sketch.js"></script>
  37 </body>
  38 </html>
  39
  
```

Let's start with the sketch.js file and include all the global variables will be needing.

Guide the student to create the global variables used in the program, create a canvas and connect to the firebase database.

The student writes code in the sketch.js file as shown in the picture below.

```

js sketch.js ▶ ⚙️ setup
1  var canvas, backgroundImage;
2
3  var gameState = 0;
4  var playerCount;
5
6  var database;
7
8  var form, player, game;
9
10
11 function setup(){
12   canvas = createCanvas(400,400);
13   database = firebase.database();
14
15 }
16
17
18 function draw(){
19 }
20

```

Let's write the Game class first.

Note: Help the student write the code and then go through it to make sure the student understands the code.

Our Game object should be able to read the gameState and update the gameState. It should also be able to start itself and display the game on the screen depending on the gameState.

Constructor of a class is used to give properties to an object when it is created. For now, we can keep the constructor empty.

Let's write functions inside the Game Class to getState and update the state.

- getState() will simply read the game state from the database.
- update(state) will update the

The student writes code for creating the Game class as shown in the image below.

gameState in the database to a value passed to it inside the parentheses.

-> `databaseReference.on()` creates a listener which keeps listening to the gameState from the database.

When the gameState is changed in the database, the function passed as an argument to it is executed.

Note: Here the function is directly written inside the `.on()` listener.

-> `databaseReference.update()` will update the database reference. Here `"/"` refers to the main database inside which gameState is created.

We can also create a `start()` function which will start the game and display on the screen depending on the state of the game.

For now, when the game State is 0, we want a form and a player object to be created. We want to display the form and get the playerCount.

We will write code to create these objects even though the blueprint isn't defined yet. This is called writing code using abstraction.

We will be writing code for these classes and creating these objects after this.

```

js ▶ JS Game.js ▶ Game ▶ getState
1  class Game {
2      constructor(){}
3
4      getState(){
5          var gameStateRef = database.ref('gameState');
6          gameStateRef.on("value",function(data){
7              gameState = data.val();
8          })
9
10     }
11
12     update(state){
13         database.ref('').update({
14             gameState: state
15         });
16     }
17
18     start(){
19         if(gameState === 0){
20             player = new Player();
21             player.getCount();
22             form = new Form()
23             form.display();
24         }
25     }
26 }
27

```

Let's write the Form Class now.

HTML is used to create any content like a form on a page. HTML is similar to markdown in some ways.

An HTML contains elements which define the structure of a page. A simple html page contains:

- head: where all the scripts and stylesheets for the page is added.
- body: where all the content of the page is added.

The body of an HTML page can contain several different types of elements:

- h1, h2, h3: display headings of

Student listens and asks questions.

	<p>different sizes.</p> <ul style="list-style-type: none"> - input: to collect input from the user. - button: to display a button. <p>This model of an HTML page is called Document object Model (or DOM).</p> <p>We will be using the p5 Dom library to create the form.</p> <p>You can look at the reference of the P5 dom library on how it is used. (Teacher Activity 2)</p>	
	<p>We will keep the constructor in the Form class empty.</p> <p>Let's write a display() function which displays the form.</p> <p><i>(Teacher asks students to refer to the p5 dom reference while writing code.)</i></p> <p>We create a title for our game "Car Racing Game":</p> <ul style="list-style-type: none"> - we create an h2 element. - we change the html content inside the element. - we position the title on the canvas. <p>Similarly, we create the input and the button element. We position the input and the button element.</p>	<p><i>The student writes the code in the display function for Form.</i></p>

```

js ▶ JS ▶ Form.js ▶ Form ▶ display
1  class Form {
2      constructor() {
3
4      }
5
6      display(){
7          var title = createElement('h2')
8          title.html("Car Racing Game");
9          title.position(130, 0);
10
11         var input = createInput("Name");
12         var button = createButton('Play');
13         var greeting = createElement('h3');
14
15         input.position(130, 160);
16         button.position(250, 200);
17
18
19
20     }
21 }
22

```

We want to greet the player when the player writes their name and logs in.

We also want to update the playerCount and the player name in the database.

button.mousePressed() can be used to trigger an action when a mouse button is pressed. It expects a function as an argument.

Let's write the code to display a greeting and update the database when the button is pressed.

Student writes the `button.mousePressed()` function and the function inside it as an argument.

When the button is pressed, student writes code to-

- hide the input and the buttons.

- increase the playerCount.
- update the playerCount and the player name in the database.

- create an h2 element and use it to greet the player when the player has logged in.

Note that `player.update()` or `player.updateCount()` are not defined yet - but the

student can use it as an abstraction.

```
js ▶ JS Form.js ▶ Form ▶ display
1  class Form {
2    constructor() {
3
4    }
5
6    display() {
7      var title = createElement('h2')
8      title.html("Car Racing Game");
9      title.position(130, 0);
10
11      var input = createInput("Name");
12      var button = createButton('Play');
13      var greeting = createElement('h3');
14
15      input.position(130, 160);
16      button.position(250, 200);
17
18      button.mousePressed();
19
20    }
21  }
22
```

```

js ▶ JS Form.js ▶ Form ▶ display ▶ button.mousePressed() callback
1  class Form {
2    constructor() {
3
4    }
5
6    display(){
7      var title = createElement('h2')
8      title.html("Car Racing Game");
9      title.position(130, 0);
10
11     var input = createInput("Name");
12     var button = createButton('Play');
13     var greeting = createElement('h3');
14
15     input.position(130, 160);
16     button.position(250, 200);
17
18     button.mousePressed(function(){
19       input.hide();
20       button.hide();
21
22       var name = input.value();
23
24       playerCount++;
25       player.update(name)
26       player.updateCount(playerCount);
27
28       greeting.html("Hello " + name )
29       greeting.position(130, 160)
30     });
31   }
32 }
33

```

Finally, let's write the code for the Player Class.

We need to write a function `getCount()` to get the `playerCount` and `updateCount()` to update the `playerCount` in the database.

We also need to update the player name in the database. For this, we need to create new entries in the database.

We can do this using string concatenation. If the `playerCount` is 1,

Student writes code for `getCount()`, `updateCount()` and `update(name)` as in the image below.

we create a database entry for player1 and we set the name for it and so on.

```
js ▶ JS Player.js ▶ Player
1  class Player {
2    constructor(){}
3
4    getCount(){
5      var playerCountRef = database.ref('playerCount');
6      playerCountRef.on("value",function(data){
7        playerCount = data.val();
8      })
9    }
10
11    updateCount(count){
12      database.ref('/').update({
13        playerCount: count
14      });
15    }
16
17    update(name){
18      var playerIndex = "player" + playerCount;
19      database.ref(playerIndex).set({
20        name:name
21      });
22    }
23  }
24
```

Finally, let's add some code in our sketch.js file to create a new Game object, get the gameState and then start the game.

The student adds the code to create a new Game object, get the game State and start the game as in the image below.

```

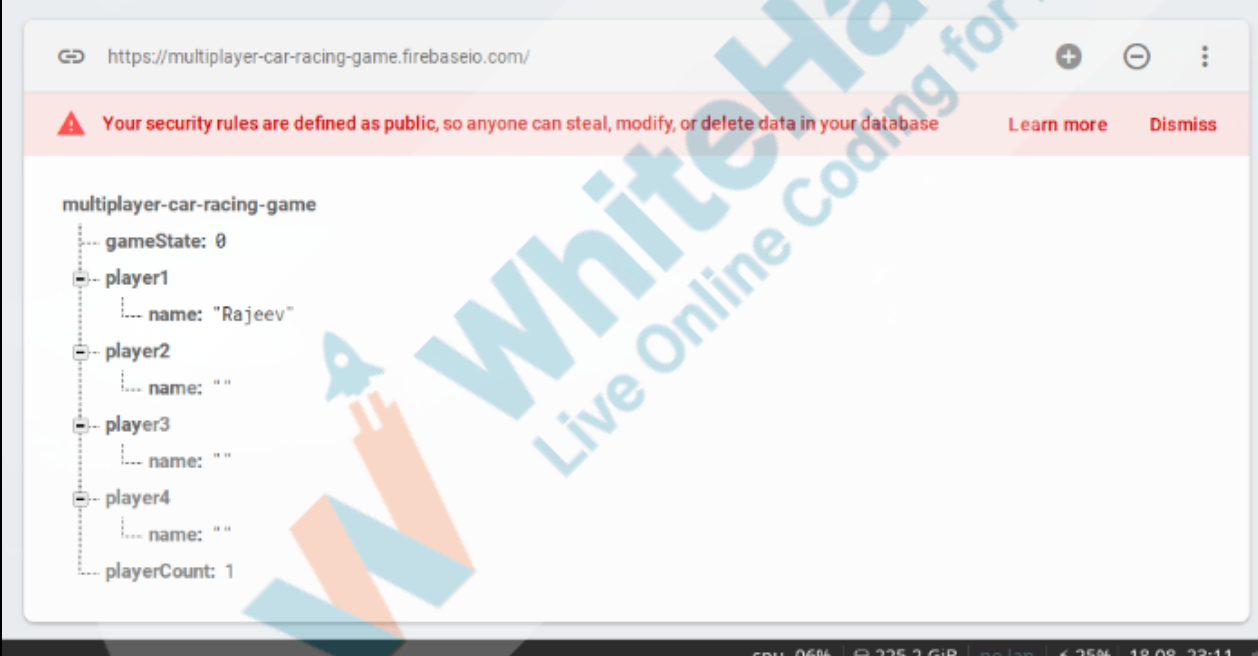
Js sketch.js ▶ setup
1  var canvas, backgroundImage;
2
3  var gameState = 0;
4  var playerCount;
5
6  var database;
7
8  var form, player, game;
9
10
11 function setup(){
12   canvas = createCanvas(400,400);
13   database = firebase.database();
14   game = new Game();
15   game.getState();
16   game.start();
17 }
18
19
20 function draw(){
21 }
22




```

Let's test our code.

The student runs the code using the 200 OK web server.

- *The student opens the link in different browsers.*
- *Student adds the player Names and observes the changes in the firebase database.*

<div style="border: 1px solid black; padding: 10px; text-align: center;"> <h3>Car Racing Game</h3> <div style="margin-top: 20px;"> <input style="width: 100px;" type="text" value="Name"/> </div> <div style="margin-top: 10px;"> <input type="button" value="Play"/> </div> </div>	<div style="border: 1px solid black; padding: 10px; text-align: center;"> <h3>Car Racing Game</h3> <div style="margin-top: 20px;"> <p>Hello Rajeev</p> </div> </div>	
		
	Help the student debug any errors.	-
Teacher Guides Student to Stop Screen Share		
<p style="text-align: center;"><u>FEEDBACK</u></p> <ul style="list-style-type: none"> Encourage the student to make reflection notes in markdown format. Complement the student for her/his effort in the class. Review the content of the lesson. 		

<p>Step 4: Wrap-Up (5 mins)</p>	<p>Do you think structuring our code before writing helped us write code better?</p> <p>How did it help us?</p>	<p>ESR: Yes!</p> <p>ESR: varied</p>
	<p>In general, a good programmer spends quality time thinking about the structure their code should follow. Programmers also spend time restructuring their code after writing so that it is more readable and understandable by others.</p> <p>How do you think it helps the programmers?</p> <p>This habit of structuring code helps them prevent future bugs and errors in their code. It also helps them write new code on top of the previously written code in an easy manner.</p>	<p>ESR: varied</p>
	<p>You get a hats off.</p> <p>We have just created a form to register our players and their names in the game.</p> <p>We need to do a number of things more - we need to stop the player addition after 4 players, we need to change the game state to play, we need to create a car racing game between all the 4 players. We will be writing new code for all this in the coming classes.</p>	<p><i>Make sure you have given at least 2 Hats Off during the class for:</i></p> <div data-bbox="1019 1266 1312 1367"> <p>Creatively Solved Activities  +10</p> </div> <div data-bbox="1019 1415 1312 1516"> <p>Great Question  +10</p> </div> <div data-bbox="1019 1564 1312 1665"> <p>Strong Concentration  +10</p> </div>

<p>Project Overview</p>	<p>Note: This is a tiered project with multiple tasks. All students must do the main task. The main task is very similar to the projects that are already live. Each tiered project has two or more additional tasks which are optional.</p> <p>VIRTUAL PET</p> <p>Goal of the Project:</p> <p>Today you created a form for players to log in, added input for name and button to Play. You also created playerCount and gameState in the database. You learned to update gameState and player count to database.</p> <p>In this project you will apply the concepts learned in class and create game design and buttons to feed and add food. Also add and remove milk bottles as per the buttons clicked.</p> <p>Story:</p> <p>Shreya really wants a pet. But nobody else in her family wants to bring a pet into the home.</p> <p>Can you create a virtual pet for Shreya? Shreya can easily track the food stock she has and the time she feeds the dog. She should also be able to add food to food stock when it is finished.</p>	<p><i>Students engage with the teacher over the project.</i></p>
--------------------------------	---	--

	<p>I am very excited to see your project solution and I know you will do really well.</p> <p>Bye Bye!</p>	
<div>Teacher Clicks</div> <div>✕ End Class</div>		
Additional Activities	<p><i>Encourage the student to write reflection notes in their reflection journal using markdown.</i></p> <p>Use these as guiding questions:</p> <ul style="list-style-type: none"> • What happened today? <ul style="list-style-type: none"> - Describe what happened - Code I wrote • How did I feel after the class? • What have I learned about programming and developing games? • What aspects of the class helped me? What did I find difficult? 	<p><i>Student uses the markdown editor to write her/his reflection as a reflection journal.</i></p>

Activity	Activity Name	Links
Teacher Activity 1	Previous class code	https://github.com/whitehatjr/synchronousBallMovement
Teacher Activity 2	p5 dom reference	https://p5js.org/reference/#group-DOM
Teacher Activity 3	Final reference code	https://github.com/whitehatjr/carRacingStage0.5
Student Activity 1	Previous class code	https://github.com/whitehatjr/synchronousBallMovement
Student Activity 2	p5 dom reference	https://p5js.org/reference/#libraries/p5.dom