Appendix

Conversations:

Criterion A:

First Conversation (January 21, 2023):

Me: Hey, Mrs. X, how are you doing today?

Client: I'm doing okay, how are you doing? How can I help you today?

Me: I'm doing good thanks for asking! Remember last year when we had to memorize the elements and a lot of people struggled with it and failed? I barely got an A on that hah.

Client: Yeah, I remember that, a lot of people got Cs and Ds on the quiz.

Me: Yeah I know, so that's why I was wondering if I could create a program to maybe help students, what do you think? I need a client and a problem for my IB Computer Science IA.

Client: Well a lot of students already use Quizlet, but I mean there is no more test option unless they pay 25 bucks a year, which is really stupid. You remember last year when I gave those out but a lot of people complained about the paywall, right? It was frustrating. Anyway, what did you have in mind?

Me: Well I was thinking of maybe creating a program where you can view information about the elements and then students can take a test to see if they got it.

Client: Ooh that sounds good, will it be multiple choice or free response?

Me: Uhhh, I actually never thought that far ahead, because I just wanted to see if it was something you were interested in, but I mean, I can have both options, it shouldn't be too hard to code. At least, I think so hah. Is this something you're interested in, though? Would you want to be my client?

Client: Yeah I definitely think, or at least hope, that this helps my students next year. Is there anything you want me to do?

Me: Well, I just need to have conversations with you and know what you would specifically want the program to do. I'll let you think about it for a while and I'll think some more about what to do, then we can meet up again and talk about it, does that sound good?

Client: Yeah, that sounds good, have a good day X.

Me: You too, Mrs. X, thank you very much!

Second Conversation (February 6, 2023):

Me: Hey, Mrs. X, so I came up with some ideas, do you have some time to talk about it?

Client: Yeah, I have some time. So before you go, I wanted to ask a couple of things. Where are you going to get the information for the elements? Because it's not just going to be the number, name, and symbol of the elements, right?

Me: Yeah so that's a good question, I am getting all of my data from code.org's periodic table data, and they get their periodic table data from Jefferson's Lab, which is a renowned and trustworthy source. However, I do plan on checking all the data on my own via other sources. I haven't done any research yet because I haven't imported the data, but I will try to use at least two different sources and see if it matches code.org's data.

Client: Okay, yeah that sounds good, if you want, I can take a look at it too and make sure that it's all accurate.

Me: Yeah, if you would like to, but that will take a lot of time on your hand and I don't think it's necessary hah.

Client: Well if you need any help with that, just let me know.

Me: Yeah, of course, thank you. So I know you wanted a test and an option to view all the elements, but I thought of some of my own things if you want to hear about it.

Client: Yeah sure go ahead.

Me: So I wanted to add an option to search for elements as well as like a random element feature just for fun for the kids.

Client: Yeah, that's a good idea, but wouldn't that take a long time to make? I don't know much about programming but that sounds complex.

Me: It'll be a simple search function, nothing too complicated, but it'll hopefully get the job done.

Client: What about something like searching, but through categories, and then the students can choose to see like different categories or something? I don't think I explained myself that well, but do you know what I'm talking about? I feel like that would help students find certain elements based on certain attributes.

Me: You mean like a filter option?

Client: Yeah that's the word hah, that would be great. Do you think you could do that?

Me: Yeah that shouldn't be too complicated. At least I hope so hah. Is there anything else you want?

Client: I think you got all the important parts, there's nothing else that I would really want.

Me: Okay so now that we got that out of the way, I want to talk about what I will use to create the program. In Computer Science A, we're programming in Java, so I was thinking about using that. And as for the IDE, I was thinking of using either Coding Rooms or Online GDB Debugger. What do you think?

Client: Hah well like I said, I know nothing about programming so I didn't understand a thing you said, but I just want you to use whatever will be best for my students.

Me: Okay, so I'll use Java, and I'll find an IDE to use it in, and uhh I think that's it. Thank you very much, Mrs. X, I'll come back later to finalize some of the things.

Client: Thank you, Y, have a good day!

Third Conversation (February 10, 2023):

Me: Hey, Mrs. X, so I just wanted to finalize everything that you want for the program. This will probably be the last time I talk to you for probably a couple of weeks.

Client: Hey, X, yeah sure, what do you need?

Me: So I need to come up with a success criteria, kind of like a checklist to make sure that this product meets all of your needs by the end.

Client: All right, let me hear them.

Me: So we want to be able to view all the elements, choose and view an element, search for an element with either name or symbol, and have a test mode, having multiple options. Is there anything else that I'm missing?

Client: Will the program allow students to test for certain elements? Like they can choose the elements?

Me: Oh yeah, I can add that too, anything else?

Client: I would assume that most students would view this on their Chromebooks, would this be supported on Chromebooks?

Me: And also, I just thought of this like yesterday, but what about like some sort of a hint system? Like to help students if they get stuck? Would you be interested in that?

Client: I don't think that'll be too helpful, but I mean, if it can help some kids, sure why not. I mean, actually, I can kind of see some students using that.

Me: Okay yeah, sounds good, I'll add that too. So I did some research and I decided to choose Coding Rooms as my IDE, and anyone with a computer and an internet connection can view the program.

Client: What about on phones?

Me: Hmm, I haven't tested that, but I can add that to the criteria and try my best to see if that will work. Is that all?

Client: I think that's it. I appreciate your help so much, have a good day X.

Me: You too, Mrs. X, thank you!

Criterion E Conversation (March 17, 2023):

Me: Hey, Mrs. X, so I finished the product, yayyy.

Client: Wow, well congratulations, X! I'm assuming you want me to take a look at it?

Me: Yeah that would be great, I would appreciate it so much, I just need to enter this link into your Chrome.

I set things up on her computer and explain and show her everything

Client: Wow. That was quite a lot. That is impressive, I do appreciate this so much, thank you. And all of this shows that you truly have a love for chemistry. I know you said that you couldn't take AP chem this year because of schedule conflicts, but do you think you could take it next year? I would love to see you in my class again, and I can really see that you care about chemistry.

Me: Yeah thank you very much, I appreciate it. Honestly, I wanted to take AP chem so badly this year, but yeah it just didn't work out. As for next year, I mean, I am not actually 100% sure I want to even take it hah because I heard about all those horror stories about chem from my friends, but you know, ultimately, it will be based on whether it works with my IB schedule or not.

Client: Yeah I understand hah.

Me: So anyway, what did you think?

Client: Honestly, I think it will be a great resource for my students. I did have some thoughts about the program, though, like when viewing all the elements, it would be nice to have a better interface so that students don't have to scroll up to see all of them. I thought it was a little annoying to view all of that. Also, electron configurations! Those are very very important for when new students start learning about the periodic table, that would be really nice to add!

Me: I completely understand, Mrs. X, I agree that a more user-friendly interface would be helpful. The problem is that Coding Rooms is kind of… stupid, and it doesn't really allow for that, so I would have to use a different website or program to do that. But it is one of the major improvements that I'm considering for the program.

Client: Yeah, I understand. Also, I remember struggling to memorize the polyionic compounds as well as strong bases and acids, so I think adding a feature for molecules, ions, acids, and bases could be really helpful for them. And I wanted to wait until you were done showing me everything, but I found it weird that certain elements that seemed unrelated to the search query would search up. Like that first time, I think you searched up lithium, or wait no, yeah no, it was lithium, yeah, and nitrogen and fluorine showed up near the end, that was weird.

Me: Wow I can't believe I forgot about all that. You see, this is why it's not a good idea for me to take it next year, I already forgot everything hah. But yeah, that's a great suggestion. I'm glad you

mentioned it. Lastly, just so you know, I tried to test this out on mobile, and… it works, but it's just kind of… not good. As for the search problem, well the thing is that fluorine for example, it was chosen because there's an "l" in lithium and fluorine, and it was one of the first element with that. It is a little weird, I admit, but the alternative would be to only have like 3 or so matches.

Client: Ah, I see… well I mean, it doesn't seem to be a problem, though, so I think it's fine.

Me: Yeah. And also, I forgot to mention this, but Coding Rooms is not mobile-friendly, so is that a concern for you or your students?

Client: While that is kind of a bummer, I wouldn't worry about it too much as I don't think it's really a problem. Like I said the first time we talked, most of my students use Chromebooks to study, or at least I assume so, so they'll be able to access the program on their laptops. But it would've been nice to have a mobile-friendly version in case they need to access it on the go.

Me: Yeah, it kind of sucks, it's one of the limitations of Coding Rooms. Thank you so much for your feedback and suggestions. Can I ask for some of the success criteria and see what you think? We already got through almost all of them, though they were out of order.

Client: Yeah, sure, go for it!

Me: So what did you think about the test feature? Did you like the multiple choice and free response and the fact that you could answer with both symbols and names?

Client: Yeah, I thought that that was pretty good, but I feel like a real screen, like an interface would have been better, kind of like when viewing all the elements, but I understand that that is the ILE limitation, or whatever that thing is called.

Me: IDE?

Client: Yeah that's the one hah, remember, I'm not that familiar with technology hah.

Me: Hah, yeah I understand. And unfortunately, I tried adding an option where the students could choose the individual elements on the test, but I found it quite tedious to manually do this, so I scrapped it altogether. Is that fine?

Client: Well, all the elements we test on, except for maybe a couple, fall into the 1 through 20 element range, so it's all accounted for. I don't really mind that it's not there.

Me: Oh okay, that's good haha. Is there anything else that you would like to add?

Client: No, I think that's it, thank you very much X, I will make sure to pass this on to my new students next year when they have to start learning. Make sure to keep in touch with me, I would love to know how you're doing in school. And if you ever need help with anything, let me know.

Me: Thank you very much for your generosity Mrs. X, have a nice day!

Code:

Main Class:

```
1. import java.util.Scanner;
2. import java.util.ArrayList;
3. 
4. public class Main {
5. 
6.     // The Scanner object that takes user inputs.
7.     public static Scanner scan = new Scanner(System.in);
8. 
9.     // String that stores the user's selection.
10.    public static String selected;
11. 
12.    // Integer version of the user's selection.
13.    public static int intSelected;
14. 
15.    // Boolean that holds whether or not the program is being run for the first time.
16.    public static boolean first = true;
17. 
18.    /**
19.     * Asks the user to select one of the five options in the home screen, then calls the
    corresponding class and its first phase.
20.     *
21.     * @param boolean tryAgain - whether the method was called again due to an invalid user
    input
22.     */
23.    public static void userHome(boolean tryAgain) {
24.        printLine(false);
25.        if(first) {
26.            System.out.println("This is a program that allows you to interact with and learn
    from the periodic table.\n");
27.            first = false;
28.        }
29.        System.out.println("How would you like to interact with the elements of the periodic
    table?");
30. 
    System.out.println("------------------------------------------------------------------------
    -------------------");
31.        System.out.println("1 - View all elements");
32.        System.out.println("2 - Search for an element");
```

```
33.         System.out.println("3 - Test your element knowledge");
34.
35.         System.out.println("\n" + (tryAgain ? "You did not choose a valid option. Please try
    again.\n" : "") + "Type and enter the number of the option you want:");
36.
37.         selected = scan.nextLine();
38.
39.         if(selected.equals("1")) {View.displayElements(false);}
40.         else if(selected.equals("2")) {Search.userChooseSearch(false);}
41.         else if(selected.equals("3")) {Test.userChooseOption1(false);}
42.         else {
43.             userHome(true);
44.         }
45.     }
46.
47.     /**
48.      * Checks if a given string can be converted to a number.
49.      *
50.      * Credit to MV, Thanoshan. "Java String to Int - How to Convert a String to an Integer."
    FreeCodeCamp.org, FreeCodeCamp.org, 23 Nov. 2020,
    https://www.freecodecamp.org/news/java-string-to-int-how-to-convert-a-string-to-an-integer/.
51.      &
52.      * @param String value - the string to be checked.
53.      *
54.      * @return boolean indicating whether the string can be converted to a number or not (true
    if can be converted, false if can't be converted).
55.      */
56.     public static boolean isNumeric(String str){
57.         return str != null && str.matches("[0-9]+");
58.     }
59.
60.     /**
61.      * Prints many empty lines and then prints a border.
62.      */
63.     public static void printLine(boolean border) {
64.         for(int i = 0; i < 50; i++) {
65.             System.out.println();
66.         }
67.         if(border)
    System.out.println("--------------------------------------------------------------------------------
    ------------------");
68.     }
```

```
69.
70.    /**
71.    * Main method where the program is first run
72.    */
73.    public static void main(String[] args) {
74.        Element.initializeElements();
75.        userHome(false);
76.    }
77.
78. }
```

Element Class:

```
1. import java.util.Arrays;
2.
3. public class Element {
4.
5.     // Variables assigned to an Element object via a constructor
6.     public String number;
7.     public String name;
8.     public String symbol;
9.     public String phase;
10.    public String type;
11.    public String period;
12.    public String group;
13.    public String weight;
14.    public String density;
15.    public String meltingPoint;
16.    public String boilingPoint;
17.
18.    // Array that contains all elements' numbers, names, and symbols in this format: Number -
   Name (Symbol) like "1 - Hydrogen (H)"
19.    public static String[] elements = new String[118];
20.
21.    // Arrays with information about all 118 elements of the periodic table.
22.    // Imported from code.org's data tab which included all of these lists about elements in
   JavaScript (which I converted to arrays in Java). Code.org allows anyone to use this
   information.
23.    public static final String[] numbers = {"1", "2", "3", "4", "5", "6", "7", "8", "9", "10",
   "11", "12", "13", "14", "15", "16", "17", "18", "19", "20", "21", "22", "23", "24", "25",
   "26", "27", "28", "29", "30", "31", "32", "33", "34", "35", "36", "37", "38", "39", "40",
   "41", "42", "43", "44", "45", "46", "47", "48", "49", "50", "51", "52", "53", "54", "55",
   "56", "57", "58", "59", "60", "61", "62", "63", "64", "65", "66", "67", "68", "69", "70",
   "71", "72", "73", "74", "75", "76", "77", "78", "79", "80", "81", "82", "83", "84", "85",
   "86", "87", "88", "89", "90", "91", "92", "93", "94", "95", "96", "97", "98", "99", "100",
   "101", "102", "103", "104", "105", "106", "107", "108", "109", "110", "111", "112", "113",
   "114", "115", "116", "117", "118"};
24.
25.    public static final String[] names = {"Hydrogen", "Helium", "Lithium", "Beryllium",
   "Boron", "Carbon", "Nitrogen", "Oxygen", "Fluorine", "Neon", "Sodium", "Magnesium",
   "Aluminum", "Silicon", "Phosphorus", "Sulfur", "Chlorine", "Argon", "Potassium", "Calcium",
   "Scandium", "Titanium", "Vanadium", "Chromium", "Manganese", "Iron", "Cobalt", "Nickel",
   "Copper", "Zinc", "Gallium", "Germanium", "Arsenic", "Selenium", "Bromine", "Krypton",
   "Rubidium", "Strontium", "Yttrium", "Zirconium", "Niobium", "Molybdenum", "Technetium",
```

```
"Ruthenium", "Rhodium", "Palladium", "Silver", "Cadmium", "Indium", "Tin", "Antimony",
"Tellurium", "Iodine", "Xenon", "Cesium", "Barium", "Lanthanum", "Cerium", "Praseodymium",
"Neodymium", "Promethium", "Samarium", "Europium", "Gadolinium", "Terbium", "Dysprosium",
"Holmium", "Erbium", "Thulium", "Ytterbium", "Lutetium", "Hafnium", "Tantalum", "Tungsten",
"Rhenium", "Osmium", "Iridium", "Platinum", "Gold", "Mercury", "Thallium", "Lead", "Bismuth",
"Polonium", "Astatine", "Radon", "Francium", "Radium", "Actinium", "Thorium", "Protactinium",
"Uranium", "Neptunium", "Plutonium", "Americium", "Curium", "Berkelium", "Californium",
"Einsteinium", "Fermium", "Mendelevium", "Nobelium", "Lawrencium", "Rutherfordium",
"Dubnium", "Seaborgium", "Bogrium", "Hassium", "Meitnerium", "Darmstadtium", "Roentgenium",
"Copernicium", "Nihonium", "Flerovium", "Moscovium", "Livermorium", "Tennessine",
"Oganesson"};
26.
27.    public static final String[] symbols = {"H", "He", "Li", "Be", "B", "C", "N", "O", "F",
"Ne", "Na", "Mg", "Al", "Si", "P", "S", "Cl", "Ar", "K", "Ca", "Sc", "Ti", "V", "Cr", "Mn",
"Fe", "Co", "Ni", "Cu", "Zn", "Ga", "Ge", "As", "Se", "Br", "Kr", "Rb", "Sr", "Y", "Zr",
"Nb", "Mo", "Tc", "Ru", "Rh", "Pd", "Ag", "Cd", "In", "Sn", "Sb", "Te", "I", "Xe", "Cs",
"Ba", "La", "Ce", "Pr", "Nd", "Pm", "Sm", "Eu", "Gd", "Tb", "Dy", "Ho", "Er", "Tm", "Yb",
"Lu", "Hf", "Ta", "W", "Re", "Os", "Ir", "Pt", "Au", "Hg", "Tl", "Pb", "Bi", "Po", "At",
"Rn", "Fr", "Ra", "Ac", "Th", "Pa", "U", "Np", "Pu", "Am", "Cm", "Bk", "Cf", "Es", "Fm",
"Md", "No", "Lr", "Rf", "Db", "Sg", "Bh", "Hs", "Mt", "Ds", "Rg", "Cn", "Nh", "Fl", "Mc",
"Lv", "Ts", "Og"};
28.
29.    public static final String[] phases = {"Gas", "Gas", "Solid", "Solid", "Solid", "Solid",
"Gas", "Gas", "Gas", "Gas", "Solid", "Solid", "Solid", "Solid", "Solid", "Solid", "Gas",
"Gas", "Solid", "Solid", "Solid", "Solid", "Solid", "Solid", "Solid", "Solid", "Solid",
"Solid", "Solid", "Solid", "Solid", "Solid", "Solid", "Liquid", "Gas", "Solid",
"Solid", "Solid", "Solid", "Solid", "Solid", "Solid", "Solid", "Solid", "Solid", "Solid",
"Solid", "Solid", "Solid", "Solid", "Solid", "Solid", "Gas", "Solid", "Solid", "Solid",
"Solid", "Solid", "Solid", "Solid", "Solid", "Solid", "Solid", "Solid", "Solid", "Solid",
"Solid", "Solid", "Solid", "Solid", "Solid", "Solid", "Solid", "Solid", "Solid", "Solid",
"Solid", "Solid", "Liquid", "Solid", "Solid", "Solid", "Solid", "Solid", "Gas", "Solid",
"Solid", "Solid", "Solid", "Solid", "Solid", "Solid", "Solid", "Solid", "Solid", "Solid",
"Solid", "Solid", "Solid", "Solid", "Solid", "Solid", "Solid", "Solid", "Solid", "Solid",
"Solid", "Solid", "Expected to be Solid", "Expected to be Solid", "Expected to be Solid",
"Expected to be Solid", "Expected to be Solid", "Expected to be Solid", "Expected to be
Solid", "Expected to be Solid", "Expected to be Solid"};
30.
31.    public static final String[] types = {"Non-Metal", "Non-Metal", "Metal", "Metal",
"Semi-Metal", "Non-Metal", "Non-Metal", "Non-Metal", "Non-Metal", "Non-Metal", "Metal",
"Metal", "Metal", "Semi-Metal", "Non-Metal", "Non-Metal", "Non-Metal", "Non-Metal", "Metal",
"Metal", "Metal", "Metal", "Metal", "Metal", "Metal", "Metal", "Metal", "Metal", "Metal",
"Metal", "Metal", "Semi-Metal", "Semi-Metal", "Non-Metal", "Non-Metal", "Non-Metal", "Metal",
```

```
"Metal", "Metal", "Metal", "Metal", "Metal", "Metal", "Metal", "Metal", "Metal", "Metal",
"Metal", "Metal", "Metal", "Semi-Metal", "Semi-Metal", "Non-Metal", "Non-Metal", "Metal",
"Metal", "Metal", "Metal", "Metal", "Metal", "Metal", "Metal", "Metal", "Metal", "Metal",
"Metal", "Metal", "Metal", "Metal", "Metal", "Metal", "Metal", "Metal", "Metal", "Metal",
"Metal", "Metal", "Metal", "Metal", "Metal", "Metal", "Metal", "Metal", "Metal",
"Semi-Metal", "Non-Metal", "Metal", "Metal", "Metal", "Metal", "Metal", "Metal", "Metal",
"Metal", "Metal", "Metal", "Metal", "Metal", "Metal", "Metal", "Metal", "Metal", "Metal",
"Metal", "Metal", "Metal", "Metal", "Metal", "Metal", "Metal", "Metal", "Metal", "Metal",
"Metal", "Metal", "Metal", "Unknown", "Non-Metal"};
32.
33.   public static final String[] periods = {"1", "1", "2", "2", "2", "2", "2", "2", "2", "2",
"3", "3", "3", "3", "3", "3", "3", "3", "4", "4", "4", "4", "4", "4", "4", "4", "4", "4",
"4", "4", "4", "4", "4", "4", "4", "4", "5", "5", "5", "5", "5", "5", "5", "5", "5", "5",
"5", "5", "5", "5", "5", "5", "5", "5", "6", "6", "6", "6", "6", "6", "6", "6", "6", "6",
"6", "6", "6", "6", "6", "6", "6", "6", "6", "6", "6", "6", "6", "6", "6", "6", "6", "6",
"6", "6", "6", "6", "7", "7", "7", "7", "7", "7", "7", "7", "7", "7", "7", "7", "7", "7",
"7", "7", "7", "7", "7", "7", "7", "7", "7", "7", "7", "7", "7", "7", "7", "7", "7", "7"};
34.
35.   public static final String[] groups = {"No definitive group", "Noble Gas", "Alkali Metal",
"Alkaline Earth Metal", "Metalloid", "Non-Metal", "Non-Metal", "Non-Metal", "Halogen", "Noble
Gas", "Alkali Metal", "Alkaline Earth Metal", "Poor Metal", "Metalloid", "Non-Metal",
"Non-Metal", "Halogen", "Noble Gas", "Alkali Metal", "Alkaline Earth Metal", "Transition
Metal", "Transition Metal", "Transition Metal", "Transition Metal", "Transition Metal",
"Transition Metal", "Transition Metal", "Transition Metal", "Transition Metal", "Transition
Metal", "Poor Metal", "Poor Metal", "Metalloid", "Non-Metal", "Halogen", "Noble Gas", "Alkali
Metal", "Alkaline Earth Metal", "Transition Metal", "Transition Metal", "Transition Metal",
"Transition Metal", "Transition Metal", "Transition Metal", "Transition Metal", "Transition
Metal", "Transition Metal", "Transition Metal", "Poor Metal", "Poor Metal", "Poor Metal",
"Metalloid", "Halogen", "Noble Gas", "Alkali Metal", "Alkaline Earth Metal", "Rare Earth
Metal", "Rare Earth Metal", "Rare Earth Metal", "Rare Earth Metal", "Rare Earth Metal", "Rare
Earth Metal", "Rare Earth Metal", "Rare Earth Metal", "Rare Earth Metal", "Rare Earth Metal",
"Rare Earth Metal", "Rare Earth Metal", "Rare Earth Metal", "Rare Earth Metal", "Rare Earth
Metal", "Transition Metal", "Transition Metal", "Transition Metal", "Transition Metal",
"Transition Metal", "Transition Metal", "Transition Metal", "Transition Metal", "Transition
Metal", "Poor Metal", "Poor Metal", "Poor Metal", "Poor Metal", "Halogen", "Noble Gas",
"Alkali Metal", "Alkaline Earth Metal", "Actinide Metal", "Actinide Metal", "Actinide Metal",
"Actinide Metal", "Actinide Metal", "Actinide Metal", "Actinide Metal", "Actinide Metal",
"Actinide Metal", "Actinide Metal", "Actinide Metal", "Actinide Metal", "Actinide Metal",
"Actinide Metal", "Actinide Metal", "Superheavy Element", "Superheavy Element", "Superheavy
Element", "Superheavy Element", "Superheavy Element", "Superheavy Element", "Superheavy
Element", "Superheavy Element", "Superheavy Element", "Superheavy Element", "Superheavy
Element", "Superheavy Element", "Superheavy Element", "Superheavy Element", "Superheavy
```

```
Element"};
36.
37.    public static final String[] weights = {"1.00794", "4.002602", "6.941", "9.1021831",
"10.811", "12.0107", "14.00674", "15.9994", "18.9984032", "20.1797", "22.9897693", "24.305",
"26.9815385", "28.0855", "30.973762", "32.066", "35.4527", "39.948", "39.0983", "40.078",
"44.955908", "47.867", "50.9415", "51.9961", "54.938044", "55.845", "58.933194", "58.6934",
"63.546", "65.38", "69.723", "72.63", "74.921595", "78.971", "79.904", "83.798", "85.4678",
"87.62", "88.90584", "91.224", "92.90637", "95.95", "98", "101.07", "102.9055", "106.42",
"107.8682", "112.414", "114.818", "118.71", "121.76", "127.6", "126.90447", "131.293",
"132.905452", "137.327", "138.90547", "140.116", "140.90766", "144.242", "145", "150.36",
"151.964", "157.25", "158.92535", "162.5", "164.93033", "167.259", "168.93422", "173.045",
"174.9668", "178.49", "180.94788", "183.84", "186.207", "190.23", "192.217", "195.084",
"196.966569", "200.592", "204.3833", "207.2", "208.9804", "209", "210", "222", "223", "226",
"227", "232.0377", "231.03588", "238.02891", "237", "244", "243", "247", "247", "251", "252",
"257", "258", "259", "262", "263", "268", "271", "270", "270", "278", "281", "281", "285",
"286", "289", "289", "293", "294", "294"};
38.
39.    public static final String[] densities = {"0.00008988", "0.0001785", "0.534", "1.85",
"2.37", "2.267", "0.0012506", "0.001429", "0.001696", "0.0008999", "0.97", "1.74", "2.7",
"2.3296", "1.82", "2.067", "0.003214", "0.0017837", "0.89", "1.54", "2.99", "4.5", "6",
"7.15", "7.3", "7.874", "8.86", "8.912", "8.933", "7.134", "5.91", "5.323", "5.776", "4.809",
"3.11", "0.003733", "1.53", "2.64", "4.47", "6.52", "8.57", "10.2", "11", "12.1", "12.4",
"12", "10.501", "8.69", "7.31", "7.287", "6.685", "6.232", "4.93", "0.005887", "1.93",
"3.62", "6.15", "6.77", "6.77", "7.01", "7.26", "7.52", "5.24", "7.9", "8.23", "8.55", "8.8",
"9.07", "9.32", "7.9", "9.84", "13.3", "16.4", "19.3", "20.8", "22.57", "22.42", "21.46",
"19.282", "13.5336", "11.8", "11.342", "9.807", "9.31", "7", "0.00973", "Unknown", "5",
"10.07", "11.72", "15.37", "18.95", "20.25", "19.84", "13.79", "13.51", "14", "Unknown",
"Unknown", "Unknown", "Unknown", "Unknown", "Unknown", "Unknown", "Unknown", "Unknown",
"Unknown", "Unknown", "Unknown", "Unknown", "Unknown", "Unknown", "Unknown", "Unknown",
"Unknown", "Unknown", "Unknown", "Unknown"};
40.
41.    public static final String[] meltingPoints = {"-434.81", "-458", "356.9", "2349", "3767",
"6422", "-346", "-361.82", "-363.32", "-415.46", "208.04", "1202", "1220.581", "2577",
"111.47", "239.38", "-150.7", "-308.83", "146.08", "1548", "2806", "3034", "3470", "3465",
"2275", "2800", "2723", "2651", "1984.32", "787.15", "85.57", "1720.85", "1503", "428.9",
"19", "-251.25", "102.76", "1431", "2772", "3371", "4491", "4753", "3915", "4233", "3567",
"2830.8", "1763.2", "609.93", "313.88", "449.47", "1167.13", "841.12", "236.7", "-169.22",
"83.19", "1341", "1684", "1468", "1708", "1870", "1908", "1965", "1512", "2395", "2473",
"2574", "2685", "2784", "2813", "1506", "3025", "4051", "5463", "6192", "5767", "5491",
"4435", "3215.1", "1947.52", "-37.89", "579", "621.43", "520.52", "489", "576", "-96", "81",
"1292", "1924", "3182", "2862", "2075", "1191", "1184", "2149", "2453", "1922", "1652",
"1580", "2781", "1521", "1520", "2961", "Unknown", "Unknown", "Unknown", "Unknown",
```

```
    "Unknown", "Unknown", "Unknown", "Unknown", "Unknown", "Unknown", "Unknown", "Unknown",
    "Unknown", "Unknown", "Unknown"};
42.
43.    public static final String[] boilingPoints = {"-423.17", "-452.07", "2448", "4480",
    "7232", "6917", "-320.44", "-297.31", "-306.62", "-410.94", "1621", "1994", "4566", "5909",
    "536.9", "832.28", "-29.27", "-302.53", "1398", "2703", "5137", "5949", "6165", "4840",
    "3742", "5182", "5301", "5275", "4644", "1655", "3999", "5131", "1137", "1265", "137.8",
    "-243.8", "1270", "2520", "6053", "7968", "8571", "8382", "7709", "7502", "6683", "5365",
    "3924", "1413", "3762", "4715", "2889", "1810", "364", "-162.62", "1240", "3447", "6267",
    "6195", "6368", "5565", "5432", "3261", "2784", "5923", "5846", "4653", "4892", "5194",
    "3542", "2185", "6156", "8317", "9856", "10031", "10105", "9054", "8002", "6917", "5173",
    "674.11", "2683", "3180", "2847", "1764", "Unknown", "-79.1", "Unknown", "2084", "5788",
    "8650", "Unknown", "7468", "7065", "5842", "3652", "5600", "Unknown", "Unknown", "Unknown",
    "Unknown", "Unknown", "Unknown", "Unknown", "Unknown", "Unknown", "Unknown", "Unknown",
    "Unknown", "Unknown", "Unknown", "Unknown", "Unknown", "Unknown", "Unknown", "Unknown",
    "Unknown", "Unknown", "Unknown"};
44.
45.    /**
46.     * Constructor for creating a new Element object based on the lists.
47.     *
48.     * @param int option - the element number of the Element object.
49.     */
50.    public Element(int option) {
51.        option--;
52.        number = numbers[option];
53.        name = names[option];
54.        symbol = symbols[option];
55.        phase = phases[option];
56.        type = types[option];
57.        period = periods[option];
58.        group = groups[option];
59.        weight = weights[option];
60.        density = densities[option];
61.        meltingPoint = meltingPoints[option];
62.        boilingPoint = boilingPoints[option];
63.    }
64.
65.    /**
66.     * Initializes the elements array (the main array that contains element names, numbers,
    and symbols).
67.     */
68.    public static void initializeElements() {
```

```
69.         for(int i = 0; i < 118; i++) {
70.             elements[i] = numbers[i] + " - " + names[i] + " (" + symbols[i] + ")";
71.         }
72.     }
73.
74.     /**
75.      * Displays all information about an element using the instance variables assigned to each
    Element object.
76.      */
77.     public void viewElement() {
78.         Main.printLine(true);
79.         System.out.println(elements[Integer.parseInt(number) - 1] + ":");
80.         System.out.println("Element Name: " + name);
81.         System.out.println("Atomic Number: " + number);
82.         System.out.println("Element Symbol: " + symbol);
83.         System.out.println("Phase type: " + phase);
84.         System.out.println("Element type: " + type);
85.         System.out.println("Element Period: " + period);
86.         System.out.println("Element Group: " + group);
87.         System.out.println("Weight: " + weight + " amu");
88.         System.out.println("Density: " + (density.equals("Unknown") ? "Unknown" : density + "
    g/L"));
89.         System.out.println("Melting Point: " + (meltingPoint.equals("Unknown") ? "Unknown" :
    meltingPoint + "°F"));
90.         System.out.println("Boiling Point: " + (boilingPoint.equals("Unknown") ? "Unknown" :
    boilingPoint + "°F"));
91.     }
92.
93.     public boolean equals(Element other) {
94.         return this.number == other.number;
95.     }
96.
97.     public String toString() {
98.         return elements[Integer.parseInt(number) - 1];
99.     }
100.
101.      /**
102.       * Creates an Element object that has a random index/atomic (element) depending on the
    user's input.
103.       *
104.       * @param int option - Range of element numbers that can be randomly chosen.
105.              1 - Random Element from 1 to 30.
```

```
106.              2 - Random Element from 31 to 60.
107.              3 - Random Element from 61 to 90.
108.              4 - Random Element from 91 to 118.
109.              5 - Random Element with any atomic number.
110.       *
111.       * @return Element object that has a random index/atomic (element) number depending on
   the parameter option.
112.       */
113.      public static Element randomElement(int option) {
114.          int randomIndex = 1;
115.          if(option == 1) {
116.              randomIndex = (int) (Math.random() * 30) + 1;
117.          } else if(option == 2) {
118.              randomIndex = (int) (Math.random() * 30) + 31;
119.          } else if(option == 3) {
120.              randomIndex = (int) (Math.random() * 30) + 61;
121.          } else if(option == 4) {
122.              randomIndex = (int) (Math.random() * 28) + 91;
123.          } else if(option == 5) {
124.              randomIndex = (int) (Math.random() * 118) + 1;
125.          }
126.
127.          return new Element(randomIndex);
128.      }
129.
130.   }
131.
```

View Class:

```
1. public class View {
2. 
3.     /**
4.      * Displays the elements of the periodic table and allows the user to view an element.
5.      *
6.      * @param boolean tryAgain - whether the method was called again due to an invalid user
   input.
7.      */
8.     public static void displayElements(boolean tryAgain) {
9.         Main.printLine(true);
10.        for(int i = 0; i < 118; i++) {
11.            System.out.println(Element.elements[i]);
12.        }
13.        System.out.println("\n" + (tryAgain ? "You did not choose a valid option. Please try
   again.\n" : "") + "Type and enter the element number you want to view, or type and enter -1
   or 0 to go home:");
14. 
15.        Main.selected = Main.scan.nextLine();
16.        Main.intSelected = Main.isNumeric(Main.selected) ? Integer.parseInt(Main.selected) :
   -10;
17. 
18.        if(Main.intSelected >= 1 && Main.intSelected <= 118) {
19.            new Element(Main.intSelected).viewElement();
20.            userDone();
21.        } else if(Main.selected.equals("-1")) {
22.            Main.userHome(false);
23.        } else if(Main.selected.equals("0")) {
24.            Main.userHome(false);
25.        } else {
26.            displayElements(true);
27.        }
28.    }
29. 
30.    /**
31.     * Allows the user to go back to the home screen or allows them to view all elements
   again.
32.     */
33.    private static void userDone() {
34.        System.out.println("\nType and enter anything when you want to go home or type and
   enter -1 to go back:");
35.        Main.selected = Main.scan.nextLine();
```

```
36.
37.         if(Main.selected.equals("-1")) {
38.             displayElements(false);
39.         } else {
40.             Main.userHome(false);
41.         }
42.     }
43.
44. }
```

Search Class:

```
1. import java.util.ArrayList;
2. 
3. public class Search {
4. 
5.     // How the user wants to search for an element.
6.     private static int option;
7. 
8.     // Array that is being searched from.
9.     private static String[] currentArray;
10. 
11.    // Array that contains new elements based on the search query.
12.    private static ArrayList<String> searchArray;
13. 
14.    /**
15.     * First phase of searching where the user decides how they want to search.
16.     *
17.     * @param boolean tryAgain - whether the method was called again due to an invalid user
    input.
18.     */
19.    public static void userChooseSearch(boolean tryAgain) {
20.        option = 0;
21. 
22.        Main.printLine(false);
23.        System.out.println("How would you like to search for the elements of the periodic
    table?");
24. 
    System.out.println("------------------------------------------------------------------------
    ------------------");
25.        System.out.println("1 - Name");
26.        System.out.println("2 - Symbol");
27.        System.out.println("\n" + (tryAgain ? "You did not choose a valid option. Please try
    again.\n" : "") + "Type and enter the option you want or type and enter -1 or 0 to go
    home:");
28. 
29.        Main.selected = Main.scan.nextLine();
30.        Main.intSelected = Main.isNumeric(Main.selected) ? Integer.parseInt(Main.selected) :
    -10;
31. 
32.        if(Main.intSelected >= 1 && Main.intSelected <= 2) {
33.            userSearch(Main.intSelected, false);
34.        } else if(Main.selected.equals("0")) {
```

```
35.             Main.userHome(false);
36.         } else if(Main.selected.equals("-1")) {
37.             Main.userHome(false);
38.         } else {
39.             userChooseSearch(true);
40.         }
41.     }
42.
43.     /**
44.      * Second phase of searching where the user searches for an element.
45.      *
46.      * @param int type - how the user wants to search for an element.
47.             1 - Name
48.             2 - Symbol
49.      * @param boolean tryAgain - whether the method was called again due to an invalid user
    input.
50.      */
51.     private static void userSearch(int type, boolean tryAgain) {
52.         option = type;
53.         Main.printLine(false);
54.         System.out.println("\n" + (tryAgain ? "You did not choose a valid option. Please try
    again.\n" : "") + "Type and enter whatever you want to search for, type and enter -1 to go
    back, or type and enter 0 to go home:");
55.
    System.out.println("------------------------------------------------------------------------
    ------------------");
56.
57.         Main.selected = Main.scan.nextLine();
58.
59.         if(Main.selected.equals("-1")) {
60.             userChooseSearch(false);
61.         } else if(Main.selected.equals("0")) {
62.             Main.userHome(false);
63.         } else if(!Main.selected.matches("[a-zA-Z]+")) {
64.             userSearch(option, true);
65.         } else {
66.             search(Main.selected);
67.         }
68.     }
69.
70. /**
71. * Third phase of searching that gets the closest elements to the search query.
```

```
72. *
73. * @param String textInput - user input.
74. */
75. private static void search(String textInput) {
76.    currentArray = Element.names;
77.    searchArray = new ArrayList<String>();
78.
79.    if(option == 2) currentArray = Element.symbols;
80.
81.    // Check for an exact match
82.    for(int i = 0; i < 118; i++) {
83.        if(currentArray[i].equalsIgnoreCase(textInput)) {
84.            searchArray.add(Element.elements[i]);
85.            break;
86.        }
87.    }
88.
89.    // Check for the first two letters
90.    if(textInput.length() >= 2) {
91.        for(int i = 0; i < 118; i++) {
92.            if(currentArray[i].toUpperCase().startsWith(textInput.toUpperCase().substring(0,
    2)) && !searchArray.contains(Element.elements[i])) {
93.                searchArray.add(Element.elements[i]);
94.            }
95.        }
96.    }
97.
98.    // Check for the first letter
99.    for(int i = 0; i < 118; i++) {
100.         if(currentArray[i].toUpperCase().startsWith(textInput.toUpperCase().substring(0,
    1)) && !searchArray.contains(Element.elements[i])) {
101.             searchArray.add(Element.elements[i]);
102.         }
103.     }
104.
105.     // Check for two letters
106.     if(searchArray.size() < 10) {
107.         if(textInput.length() > 2) {
108.             for(int i = 0; i < 118; i++) {
109.                 for(int j = 0; j < (textInput.length() < currentArray[i].length() ?
    textInput.length() - 2 : currentArray[i].length() - 2); j++) {
110.                     if(searchArray.size() > 9) {
```

```
                            displayElements(false);
                            return;
                        }
                        if(currentArray[i].substring(j, j +
    2).equalsIgnoreCase(textInput.substring(j, j + 2)) &&
    !searchArray.contains(Element.elements[i])) {
                            searchArray.add(Element.elements[i]);
                        }
                    }
                }
            }

        displayElements(false);
    }

        /**
         * Fourth phase of searching that displays the closest elements to the search query
    and allows users to view an element.
         *
         * @param boolean tryAgain - whether the method was called again due to an invalid
    user input.
         */
        private static void displayElements(boolean tryAgain) {
            Main.printLine(true);
            if(searchArray.size() > 0) {
                for(int i = 0; i < searchArray.size(); i++) {
                    System.out.println(searchArray.get(i));
                }
                System.out.println("\n" + (tryAgain ? "You did not choose a valid option.
    Please try again.\n" : "") + "Type and enter the element number you want to view, type and
    enter -1 to go back, or type and enter 0 to go home:");
            } else {
                System.out.println((tryAgain ? "You did not choose a valid option. Please try
    again.\n" : "") + "No elements found. Type and enter -1 to go back or type and enter 0 to go
    home:");
            }

            Main.selected = Main.scan.nextLine();
            Main.intSelected = Main.isNumeric(Main.selected) ? Integer.parseInt(Main.selected)
    : -10;

```

```
144.          if(Main.intSelected >= 1 && Main.intSelected <= 118) {
145.              new Element(Main.intSelected).viewElement();
146.              userDone(option);
147.          } else if(Main.selected.equals("-1")) {
148.              userSearch(option, false);
149.          } else if(Main.selected.equals("0")) {
150.              Main.userHome(false);
151.          } else {
152.              displayElements(true);
153.          }
154.      }
155.
156.      /**
157.       * Allows the user to go back to the home screen or allows them to view the searched
    list again.
158.       */
159.      private static void userDone(int option) {
160.          System.out.println("\nType and enter anything when you want to go home or type and
    enter -1 to go back:");
161.          Main.selected = Main.scan.nextLine();
162.
163.          if(Main.selected.equals("-1")) {
164.              displayElements(false);
165.          } else {
166.              Main.userHome(false);
167.          }
168.      }
169.
170.  }
```

Test Class:

```
1. public class Test {
2.
3.    // Number of correct answers.
4.    private static int score;
5.
6.    // Whether the test is finished.
7.    private static boolean done;
8.
9.    // How the user answers the question (1).
10.           // 1 - Free Response
11.           // 2 - Multiple Choice
12.           // 3 - Mix of Both
13.   private static int option1;
14.
15.   // How the user answers the question (2).
16.           // 1 - Name
17.           // 2 - Symbol
18.           // 3 - Mix of Both
19.   private static int option2;
20.
21.   // What range of element numbers the answers fall in.
22.           // 1 - (1 to 30)
23.           // 2 - (31 to 60)
24.           // 3 - (61 to 90)
25.           // 4 - (91 to 118)
26.           // 5 - Any atomic number
27.   private static int option3;
28.
29.   // How many questions the user wants to answer.
30.           // 1 - 5 Questions
31.           // 2 - 10 Questions
32.           // 3 - 15 Questions
33.           // 4 - 20 Questions
34.           // 5 - Custom (user input)
35.   private static int number;
36.
37.  // The question number that the user is on during the test.
38.   private static int numberOn;
39.
40.   // Either "name" or "symbol" depending on option2.
41.   private static String given;
```

```
42.
43.    // The name or symbol of the element depending on option2.
44.    private static String givenVar;
45.
46.    // The opposite of given (if given is name, lookingFor is symbol and vice versa).
47.    private static String lookingFor;
48.
49.    // The opposite of givenVar (if givenVar is the element's name, lookingForVar is the
   element's symbol and vice versa).
50.    private static String lookingForVar;
51.
52.    // The Element object which is the correct answer.
53.    private static Element answer;
54.
55.    // Index of the next right answer.
56.    private static int index;
57.
58.    // Index of the answer for multiple choice questions.
59.    private static int rightIndex;
60.
61.    // Element objects that are fake answers if the user is answering via multiple choice.
62.    private static Element element1;
63.    private static Element element2;
64.    private static Element element3;
65.
66.    // The array that contains the randomly sorted four answers (element1, element2, element3,
   answer) if the user is answering via multiple choice.
67.    private static Element[] sortedTestElements;
68.
69.    // Element 2D array where in one row, it contains all the answers in the test, and another
   row has all the answers that the user got wrong on the test.
70.    private static Element[][] answers;
71.
72.    // Boolean variable that holds whether or not the user got an answer wrong on the test.
73.    private static boolean isWrongAnswer;
74.
75.    // What hint phase the user is in (how many hints they have received).
76.    private static int hintPhase;
77.
78.    /**
79.     * This method checks if a given value is present in an array of Element objects.
80.     *
```

```
81.     * @param Element[] array - array of Element objects to check the value in.
82.     * @param Element value - The value to be checked for in the array.
83.     *
84.     * @return boolean indicating whether the value was found in the array or not (true if
    found, false if not).
85.     */
86.    public static boolean isPresent(Element[] array, Element value) {
87.        if(value == null) {return false;}
88.        for(int i = 0; i < array.length; i++) {
89.            if(array[i] != null && value.equals(array[i])) {
90.                return true;
91.            }
92.        }
93.        return false;
94.    }
95.
96.    /**
97.     * First phase of testing where the user decides how to answer the question (multiple
    choice, free response, or a mix of both).
98.     *
99.     * @param boolean tryAgain - whether the method was called again due to an invalid user
    input.
100.       */
101.      public static void userChooseOption1(boolean tryAgain) {
102.          number = 0;
103.          option1 = 0;
104.          option2 = 0;
105.          option3 = 0;
106.          index = 1;
107.
108.          Main.printLine(false);
109.          System.out.println("How would you like to answer the questions on the test?");
110.
    System.out.println("------------------------------------------------------------------------
    -------------------");
111.          System.out.println("1 - Answer with free response");
112.          System.out.println("2 - Answer with multiple choice");
113.          System.out.println("3 - Answer with a mix of both");
114.          System.out.println("\n" + (tryAgain ? "You did not choose a valid option. Please
    try again.\n" : "") + "Type and enter the number of the option you want or type and enter -1
    or 0 to go home:");
115.
```

```
116.          Main.selected = Main.scan.nextLine();
117.          Main.intSelected = Main.isNumeric(Main.selected) ? Integer.parseInt(Main.selected)
    : -10;
118.
119.          if(Main.intSelected >= 1 && Main.intSelected <= 3) {
120.              option1 = Main.intSelected;
121.              userChooseOption2(false);
122.          } else if(Main.selected.equals("-1")) {
123.              Main.userHome(false);
124.          } else if(Main.selected.equals("0")) {
125.              Main.userHome(false);
126.          } else {
127.              userChooseOption1(true);
128.          }
129.      }
130.
131.      /**
132.       * Second phase of testing where the user decides how to answer the question (name,
    symbol, or a mix of both).
133.       *
134.       * @param boolean tryAgain - whether the method was called again due to an invalid
    user input.
135.       */
136.      private static void userChooseOption2(boolean tryAgain) {
137.          Main.printLine(false);
138.          System.out.println("How would you like to answer the questions on the test?");
139.
    System.out.println("--------------------------------------------------------------------------
    --------------------");
140.          System.out.println("1 - Answer with element name");
141.          System.out.println("2 - Answer with element symbol");
142.          System.out.println("3 - Answer with a mix of both");
143.          System.out.println("\n" + (tryAgain ? "You did not choose a valid option. Please
    try again.\n" : "") + "Type and enter the number of the option you want, type and enter -1 to
    go back, or type and enter 0 to go home:");
144.
145.          Main.selected = Main.scan.nextLine();
146.          Main.intSelected = Main.isNumeric(Main.selected) ? Integer.parseInt(Main.selected)
    : -10;
147.
148.          if(Main.intSelected >= 1 && Main.intSelected <= 3) {
149.              option2 = Main.intSelected;
```

```
150.             userChooseOption3(false);
151.         } else if(Main.selected.equals("-1")) {
152.             userChooseOption1(false);
153.         } else if(Main.selected.equals("0")) {
154.             Main.userHome(false);
155.         } else {
156.             userChooseOption2(true);
157.         }
158.     }
159.
160.     /**
161.      * Third phase of testing where the user decides the range of the answers given (1 -
    30, 31 - 60, 61 - 90, 91 - 118, any atomic number).
162.      *
163.      * @param boolean tryAgain - whether the method was called again due to an invalid
    user input.
164.      */
165.     private static void userChooseOption3(boolean tryAgain) {
166.         Main.printLine(false);
167.         System.out.println("What atomic number range for elements do you want to be on the
    test?");
168.
    System.out.println("--------------------------------------------------------------------------
    --------------------");
169.         System.out.println("1 - Elements from 1 to 30");
170.         System.out.println("2 - Elements from 31 to 60");
171.         System.out.println("3 - Elements from 61 to 90");
172.         System.out.println("4 - Elements from 91 to 118");
173.         System.out.println("5 - Elements with any atomic number");
174.         System.out.println("\n" + (tryAgain ? "You did not choose a valid option. Please
    try again.\n" : "") + "Type and enter the number of the option you want, type and enter -1 to
    go back, or type and enter 0 to go home:");
175.
176.         Main.selected = Main.scan.nextLine();
177.         Main.intSelected = Main.isNumeric(Main.selected) ? Integer.parseInt(Main.selected)
    : -10;
178.
179.         if(Main.intSelected >= 1 && Main.intSelected <= 5) {
180.             option3 = Main.intSelected;
181.             userChooseQuestions(false);
182.         } else if(Main.selected.equals("-1")) {
183.             userChooseOption2(false);
```

```
184.          }  else if(Main.selected.equals("0")) {
185.              Main.userHome(false);
186.          }else {
187.              userChooseOption3(true);
188.          }
189.      }
190.
191.      /**
192.       * Fourth phase of testing where the user decides how many questions they want to
    choose (5, 10, 15, 20, custom).
193.       *
194.       * @param boolean tryAgain - whether the method was called again due to an invalid
    user input.
195.       */
196.      private static void userChooseQuestions(boolean tryAgain) {
197.          Main.printLine(false);
198.          System.out.println("How many questions do you want on the test?");
199.
    System.out.println("--------------------------------------------------------------------------
    --------------------");
200.          System.out.println("1 - 5 Questions");
201.          System.out.println("2 - 10 Questions");
202.          System.out.println("3 - 15 Questions");
203.          System.out.println("4 - 20 Questions");
204.          System.out.println("5 - Custom");
205.          System.out.println("\n" + (tryAgain ? "You did not choose a valid option. Please
    try again.\n" : "") + "Type and enter the number of the option you want, type and enter -1 to
    go back, or type and enter 0 to go home:");
206.
207.          Main.selected = Main.scan.nextLine();
208.          Main.intSelected = Main.isNumeric(Main.selected) ? Integer.parseInt(Main.selected)
    : -10;
209.
210.          if(Main.intSelected >= 1 && Main.intSelected <= 4) {
211.              number = Main.intSelected * 5;
212.              test();
213.          } else if(Main.intSelected == 5) {
214.              userChooseCustomQuestions(false);
215.          } else if(Main.selected.equals("-1")) {
216.              userChooseOption3(false);
217.          } else if(Main.selected.equals("0")) {
218.              Main.userHome(false);
```

```
219.          } else {
220.              userChooseQuestions(true);
221.          }
222.      }
223.
224.      /**
225.       * Fourth phase of testing where the user decides how many questions they want to
   choose if number is equal to 5 meaning that they choose a custom number of questions.
226.       *
227.       * @param boolean tryAgain - whether the method was called again due to an invalid
   user input.
228.       */
229.      private static void userChooseCustomQuestions(boolean tryAgain) {
230.          Main.printLine(false);
231.          System.out.println("\n" + (tryAgain ? "You did not choose a valid option. Please
   try again.\n" : "") + "Type and enter the number of questions you want to answer, type and
   enter -1 to go back, or type and enter 0 to go home:");
232.
   System.out.println("------------------------------------------------------------------------
   --------------------");
233.
234.          Main.selected = Main.scan.nextLine();
235.          Main.intSelected = Main.isNumeric(Main.selected) ? Integer.parseInt(Main.selected)
   : -10;
236.
237.          if(option3 == 5 && Main.intSelected >= 1 && Main.intSelected <= 118) {
238.              number = Main.intSelected;
239.              test();
240.          } else if(option3 != 5 && Main.intSelected >= 1 && Main.intSelected <= 30) {
241.              number = Main.intSelected;
242.              test();
243.          } else if(Main.selected.equals("-1")) {
244.              userChooseQuestions(false);
245.          } else if(Main.selected.equals("0")) {
246.              Main.userHome(false);
247.          } else {
248.              userChooseCustomQuestions(true);
249.          }
250.      }
251.
252.      /**
253.       * Shuffles an element array in random order.
```

```
254.       *
255.       * @param Element[] array - the array that is to be shuffled.
256.       *
257.       * @return Element[] array that is shuffled.
258.       */
259.      private static Element[] shuffleArray(Element[] array) {
260.          Element[] shuffledArray = new Element[array.length];
261.
262.          int i = 0;
263.          while(i < array.length) {
264.              int random = (int) (Math.random() * array.length);
265.              if(!isPresent(shuffledArray, array[random])) {
266.                  shuffledArray[i] = array[random];
267.                  i++;
268.              }
269.          }
270.
271.          return shuffledArray;
272.      }
273.
274.      /**
275.       * Test phase of testing where the test starts and where the other phases are
    controlled and called.
276.       *
277.       * @param boolean testingAgain - whether or not the user is testing again.
278.       */
279.      private static void test() {
280.          answers = new Element[2][number];
281.          index = 0;
282.          score = 0;
283.          numberOn = 1;
284.          done = false;
285.          isWrongAnswer = false;
286.
287.          Main.printLine(false);
288.          while(numberOn <= number && !done) {
289.              hintPhase = 0;
290.
291.              do {
292.                  answer = Element.randomElement(option3);
293.              } while(isPresent(answers[0], answer));
294.
```

```
295.             answers[0][index] = answer;
296.
297.             // If option1 is 3, then it randomly becomes either 1 or 2.
298.             int decide1 = (int) (Math.random() * 3) + 1;
299.
300.             // If option2 is 3, then it randomly becomes either 1 or 2.
301.             int decide2 = (int) (Math.random() * 3) + 1;
302.
303.             if(option2 == 1 || (option2 == 3 && decide2 == 1)) {
304.                 lookingFor = "name";
305.                 lookingForVar = answer.name;
306.                 given = "symbol";
307.                 givenVar = answer.symbol;
308.             } else {
309.                 lookingFor = "symbol";
310.                 lookingForVar = answer.symbol;
311.                 given = "name";
312.                 givenVar = answer.name;
313.             }
314.
315.             if(option1 == 1 || (option1 == 3 && decide1 == 1)) {
316.                 System.out.println("\nType and enter the " + lookingFor + " of the element
    " + '"' + givenVar + '"' + " (type and enter 0 to receive a hint) or type and enter -1 to end
    the test:");
317.
    System.out.println("----------------------------------------------------------------------
    --------------------");
318.                 userAnswersFR();
319.             } else {
320.                 createMC(decide2);
321.             }
322.
323.             if(done) return;
324.
325.             answer = Element.randomElement(option3);
326.             index++;
327.             numberOn++;
328.         }
329.         userChooseDisplay(false);
330.     }
331.
332.     /**
```

```
333.       * Option to receive a hint about an element if the user inputs 0.
334.       */
335.      private static void hint(int option, int decide) {
336.          hintPhase++;
337.          System.out.println();
338.          if(hintPhase == 1) {
339.              System.out.println("The element's phase is " + answer.phase);
340.          } else if(hintPhase == 2) {
341.              System.out.println("The element's type is " + answer.type);
342.          } else if(hintPhase == 3) {
343.              System.out.println("The element is in period " + answer.period);
344.          } else if(hintPhase == 4) {
345.              System.out.println("The element's group is " + answer.group);
346.          } else {
347.              System.out.println("You have no hints left.");
348.          }
349.
350.          if(option == 1) {
351.              System.out.println("\nType and enter the " + lookingFor + " of the element " +
    '"' + givenVar + '"' + (hintPhase < 4 ? " (type and enter 0 to receive another hint)" : "") +
    " or enter -1 to end the test:");
352.              userAnswersFR();
353.          } else {
354.
    System.out.println("----------------------------------------------------------------------------
    --------------------");
355.              if(option2 == 1 || (option2 == 3 && decide == 1)) {
356.                  System.out.println("1 - " + sortedTestElements[0].name);
357.                  System.out.println("2 - " + sortedTestElements[1].name);
358.                  System.out.println("3 - " + sortedTestElements[2].name);
359.                  System.out.println("4 - " + sortedTestElements[3].name);
360.              } else {
361.                  System.out.println("1 - " + sortedTestElements[0].symbol);
362.                  System.out.println("2 - " + sortedTestElements[1].symbol);
363.                  System.out.println("3 - " + sortedTestElements[2].symbol);
364.                  System.out.println("4 - " + sortedTestElements[3].symbol);
365.              }
366.              System.out.println("\nType and enter the option of the element " + '"' +
    givenVar + '"' + (hintPhase < 4 ? " (type and enter 0 to receive another hint)" : "") + " or
    type and enter -1 to end the test:");
367.              userAnswersMC(decide);
368.          }
```

```
369.      }
370.
371.      /**
372.       * Fifth phase of testing, if the user is answering a free response question, where
   the user enters an input and the input is compared to the correct answer.
373.       */
374.      private static void userAnswersFR() {
375.          Main.selected = Main.scan.nextLine();
376.
377.          if(Main.selected.equals("-1")) {
378.              done = true;
379.              userChooseDisplay(false);
380.          } else if(Main.selected.equals("0")) {
381.              hint(1, 0);
382.          } else if(!Main.selected.matches("[a-zA-Z]+")) {
383.              System.out.println("\nYou did not choose a valid option. Please try
   again.\nType and enter the " + lookingFor + " of the element " + '"' + givenVar + '"' + "
   (type and enter 0 to receive a hint) or type and enter -1 to end the test:");
384.              userAnswersFR();
385.          } else {
386.              if(Main.selected.equalsIgnoreCase(lookingForVar)) {
387.                  System.out.println("\nNice, you got it!");
388.                  score++;
389.              } else {
390.                  System.out.println("\nWrong, the correct answer was " + lookingForVar +
   ".");
391.                  isWrongAnswer = true;
392.                  answers[1][index] = answer;
393.              }
394.          }
395.      }
396.
397.      /**
398.       * Creates and displays fake answers if the user is answering a multiple choice
   question.
399.       *
400.       * @param int decide - randomly chosen integer that is only used if option2 equals 3
   to randomly decide if the user answers with a name or symbol.
401.       */
402.      private static void createMC(int decide) {
403.          rightIndex = 1;
404.          do {
```

```
405.              element1 = Element.randomElement(option3);
406.              element2 = Element.randomElement(option3);
407.              element3 = Element.randomElement(option3);
408.          } while(element2.equals(element1) || element3.equals(element1) ||
    element2.equals(element3) || answer.equals(element1) || answer.equals(element2) ||
    answer.equals(element3));
409.
410.          Element[] testElements = {element1, element2, element3, answer};
411.          sortedTestElements = shuffleArray(testElements);
412.
413.          for(int i = 0; i < 4; i++) {
414.              if(sortedTestElements[i].equals(answer)) {
415.                  rightIndex = i + 1;
416.              }
417.          }
418.
419.
    System.out.println("\n-------------------------------------------------------------------------
    ----------------------");
420.          if(option2 == 1 || (option2 == 3 && decide == 1)) {
421.              System.out.println("1 - " + sortedTestElements[0].name);
422.              System.out.println("2 - " + sortedTestElements[1].name);
423.              System.out.println("3 - " + sortedTestElements[2].name);
424.              System.out.println("4 - " + sortedTestElements[3].name);
425.          } else {
426.              System.out.println("1 - " + sortedTestElements[0].symbol);
427.              System.out.println("2 - " + sortedTestElements[1].symbol);
428.              System.out.println("3 - " + sortedTestElements[2].symbol);
429.              System.out.println("4 - " + sortedTestElements[3].symbol);
430.          }
431.
432.          System.out.println("\nType and enter the option of the element " + '"' + givenVar
    + '"' + " (type and enter 0 to receive a hint) or type and enter -1 to end the test:");
433.          userAnswersMC(decide);
434.      }
435.
436.      /**
437.       * Fifth phase of testing, if the user is answering a multiple choice question, where
    the user chooses one of the multiple choice options which is compared to the right answer.
438.       *
439.       * @param int decide - randomly chosen integer that is only used if option2 equals 3
    to randomly decide if the user answers with a name or symbol.
```

```
440.       */
441.      private static void userAnswersMC(int decide) {
442.          Main.selected = Main.scan.nextLine();
443.          Main.intSelected = Main.isNumeric(Main.selected) ? Integer.parseInt(Main.selected)
    : -10;
444.
445.          if(Main.selected.equals("-1")) {
446.              done = true;
447.              userChooseDisplay(false);
448.          } else if(Main.selected.equals("0")) {
449.              hint(2, decide);
450.          } else if(Main.intSelected >= 1 && Main.intSelected <= 4) {
451.              if(Main.intSelected == rightIndex) {
452.                  System.out.println("\nNice, you got it!");
453.                  score++;
454.              } else {
455.                  System.out.println("\nWrong, the correct answer was " + lookingForVar + "
    (" + rightIndex + ").");
456.                  answers[1][index] = answer;
457.                  isWrongAnswer = true;
458.              }
459.          } else {
460.
    System.out.println("\n--------------------------------------------------------------------------
    ----------------------");
461.              if(option2 == 1 || (option2 == 3 && decide == 1)) {
462.                  System.out.println("1 - " + sortedTestElements[0].name);
463.                  System.out.println("2 - " + sortedTestElements[1].name);
464.                  System.out.println("3 - " + sortedTestElements[2].name);
465.                  System.out.println("4 - " + sortedTestElements[3].name);
466.              } else {
467.                  System.out.println("1 - " + sortedTestElements[0].symbol);
468.                  System.out.println("2 - " + sortedTestElements[1].symbol);
469.                  System.out.println("3 - " + sortedTestElements[2].symbol);
470.                  System.out.println("4 - " + sortedTestElements[3].symbol);
471.              }
472.              System.out.println("\nYou did not choose a valid option. Please try
    again.\nType and enter the option of the element " + '"' + givenVar + '"' + " or type and
    enter -1 to end the test:");
473.              userAnswersMC(decide);
474.          }
475.      }
```

```
476.
477.      /**
478.       * Sixth phase of testing where the test is finished and the user sees their score and
   decides if they want to view their answers.
479.       *
480.       * @param boolean tryAgain - whether the method was called again due to an invalid
   user input.
481.       */
482.      private static void userChooseDisplay(boolean tryAgain) {
483.          Main.printLine(false);
484.
485.          System.out.println("Congratulations, you got " + score + " out of " + (numberOn -
   1) + "! Would you like to view your answers?");
486.
   System.out.println("------------------------------------------------------------------------
   --------------------");
487.          System.out.println("1 - Display all of your answers");
488.          if(isWrongAnswer) System.out.println("2 - Display all of your wrong answers");
489.          System.out.println("\n" + (tryAgain ? "You did not choose a valid option. Please
   try again.\n" : "") + "Type and enter the number of the option you want or type and enter -1
   or 0 to go home:");
490.
491.          Main.selected = Main.scan.nextLine();
492.          Main.intSelected = Main.isNumeric(Main.selected) ? Integer.parseInt(Main.selected)
   : -10;
493.
494.          if(Main.intSelected == 1 || (isWrongAnswer && Main.intSelected == 2)) {
495.              displayElements(Main.intSelected, false);
496.          } else if(Main.selected.equals("-1")) {
497.              Main.userHome(false);
498.          } else if(Main.selected.equals("0")) {
499.              Main.userHome(false);
500.          } else {
501.              userChooseDisplay(true);
502.          }
503.      }
504.
505.      /**
506.       * Seventh phase of testing where the answers are displayed.
507.       *
508.       * @param boolean tryAgain - whether the method was called again due to an invalid
   user input.
```

```
509.       */
510.      private static void displayElements(int option, boolean tryAgain) {
511.          if(option == 1) {
512.              Main.printLine(false);
513.              System.out.println("These were all the elements asked for in the test:");
514.
    System.out.println("------------------------------------------------------------------------
    --------------------");
515.              for(int i = 0; i < answers[0].length; i++) {
516.                  if(answers[0][i] != null) {
517.                      System.out.println(answers[0][i]);
518.                  }
519.              }
520.          } else {
521.              Main.printLine(false);
522.              System.out.println("These were all the elements you got wrong on the test:");
523.
    System.out.println("------------------------------------------------------------------------
    --------------------");
524.              for(int i = 0; i < answers[1].length; i++) {
525.                  if(answers[1][i] != null) {
526.                      System.out.println(answers[1][i]);
527.                  }
528.              }
529.          }
530.          System.out.println("\n" + (tryAgain ? "You did not choose a valid option. Please
    try again.\n" : "") + "Type and enter the element number you want to view, type and enter -1
    to go back, or type and enter 0 to go home:");
531.
532.          Main.selected = Main.scan.nextLine();
533.          Main.intSelected = Main.isNumeric(Main.selected) ? Integer.parseInt(Main.selected)
    : -10;
534.
535.          if(Main.intSelected >= 1 && Main.intSelected <= 118) {
536.              new Element(Main.intSelected).viewElement();
537.              userDone(option);
538.          } else if(Main.selected.equals("-1")) {
539.              userChooseDisplay(false);
540.          } else if(Main.selected.equals("0")) {
541.              Main.userHome(false);
542.          } else {
543.              displayElements(option, true);
```

```
544.          }
545.      }
546.
547.      /**
548.       * Allows the user to go back to the home screen or allows them to view their answers
   on the test again.
549.       */
550.      private static void userDone(int option) {
551.          System.out.println("\nType and enter anything when you want to go home or type and
   enter -1 to go back:");
552.          Main.selected = Main.scan.nextLine();
553.
554.          if(Main.selected.equals("-1")) {
555.              displayElements(option, false);
556.          } else {
557.              Main.userHome(false);
558.          }
559.      }
560.
561.  }
```