Abubakr Elmallah
https://app.codingrooms.com/w/Yxexan1LM35u

Periodic Table - Computer Science IA

Criterion A: Planning

The Scenario

My client is my former chemistry teacher at my high school. In chemistry class, students must memorize certain elements of the periodic table, and many people struggled with this last year when I took chemistry. I love chemistry and was really great at it last year as a sophomore, so I think that it is a good idea to use chemistry as my topic.

I talked with my client, and she told me that she was saddened that Quizlet decided to lock the test mode behind a subscription paywall, preventing most students from being able to utilize it. Because of this, we agreed that the best solution would be a free **Periodic Table** program to help her students. The program would show all elements of the periodic table, allowing students to view all the elements of the periodic table, search through the periodic table, and then allow them to test their knowledge via multiple choice and free response.

She was skeptical of the amount of programming knowledge that would be required to take on this task, but after thinking it through, I believe that I have enough skill and knowledge to take on this task.[2]

Rationale

I chose Java as the programming language and Coding Rooms as my Integrated Development Environment because I am familiar with Java because I have used it for the last year. I am not familiar with any other language to the same extent. I chose Coding Rooms as my IDE because it is free and is much better than many other IDEs that I attempted to use like GDB Debugger and Project STEM, while Coding Rooms was perfect because it had a clean, simple, and easy-to-use interface.

Coding Rooms also allows any computer to access it as long as the computer has a browser and an internet connection and has an account (which will take less than 1 minute to make if using their google school accounts), which all students at my high school have. My client and I agreed that most chemistry students would probably study on their computers, so having it optimized for computers would be best.

I will get my data for the attributes of elements of the periodic table from code.org's periodic table data column which allows attributes of the periodic table to be imported as an array. Code.org uses reliable information for their element data so there is no reason for me to worry about the accuracy.

Success Criteria

My client and I agreed on this:

1. View all elements in the periodic table in ascending element order.
2. Choose an element of the periodic table and view its attributes.
3. Search for elements via names or symbols and find the best matches.

4. Allow students to test their element memorization skills via free response and multiple choice.
5. Allow students to test their element memorization skills by answering with names or with symbols.
6. Allow users to choose their own elements that they want to test.
7. Allow users to receive hints about an element during the test to help them.
8. The program can be viewed on all devices, including phones and computers.

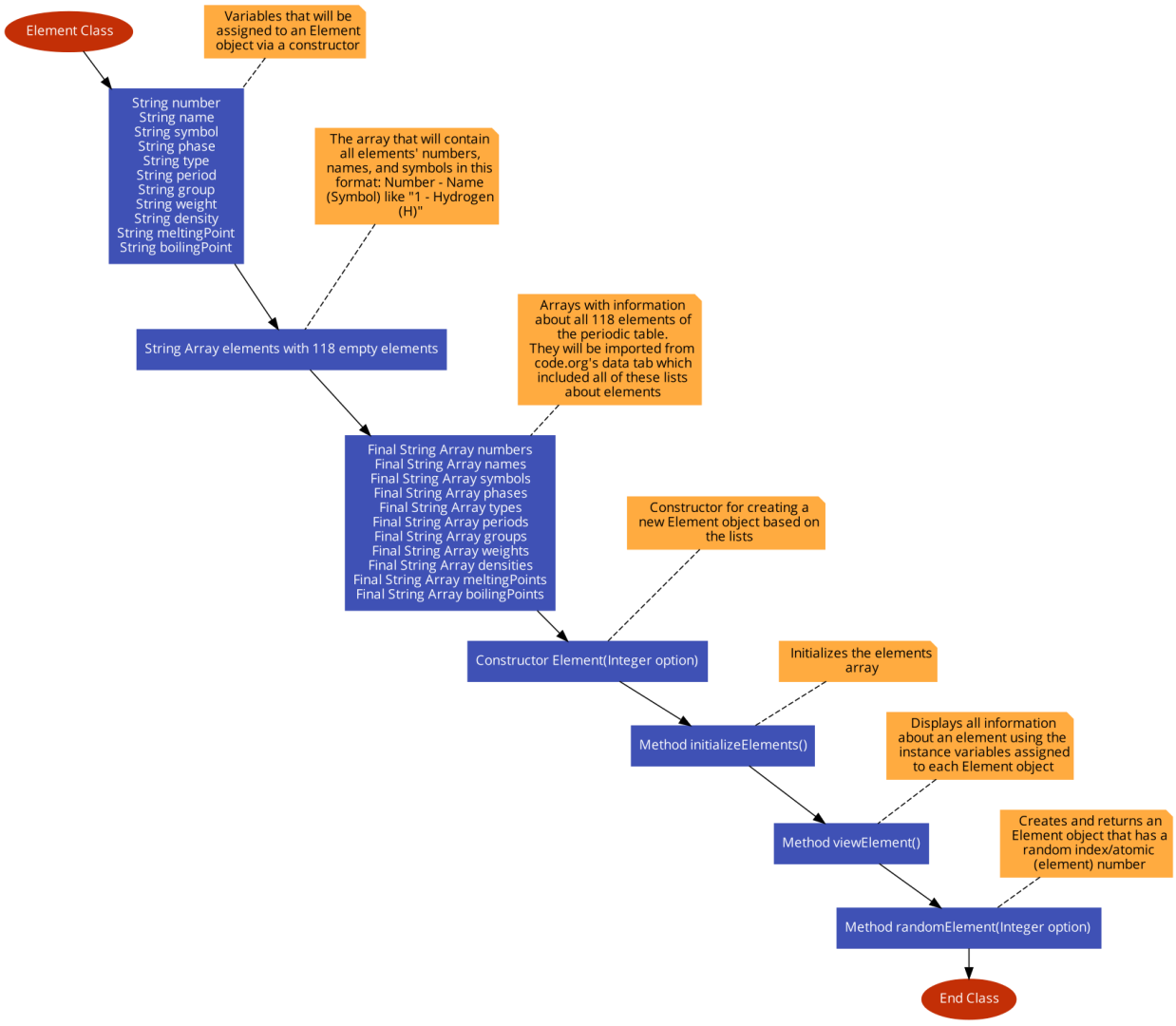
Criterion B: Design

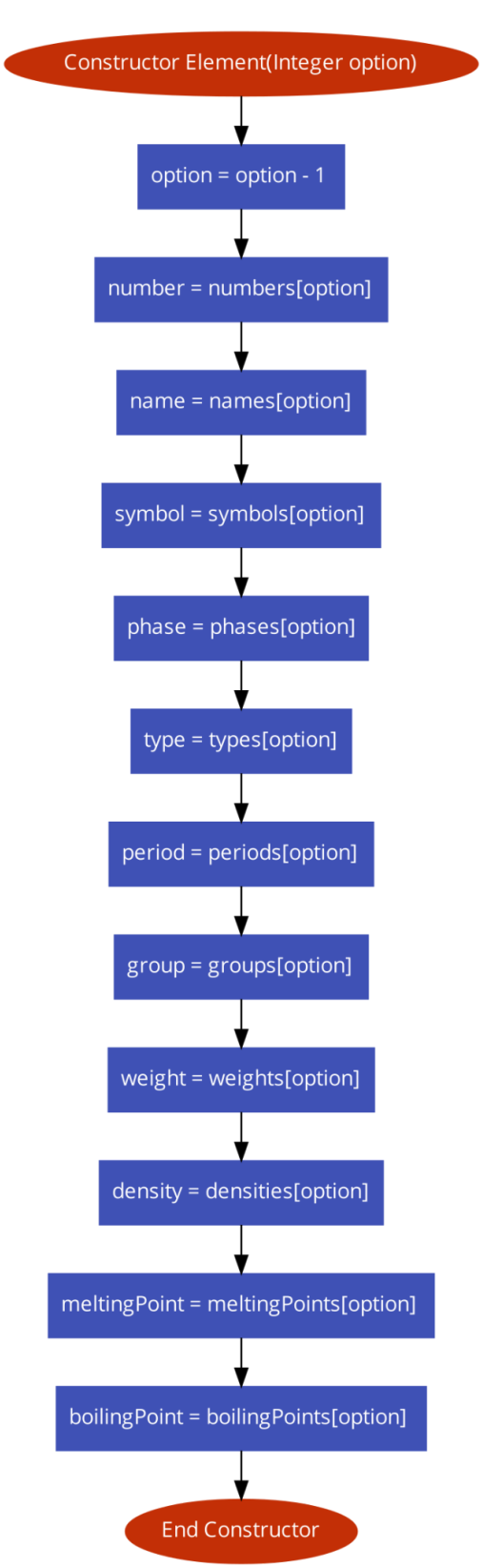
Overview

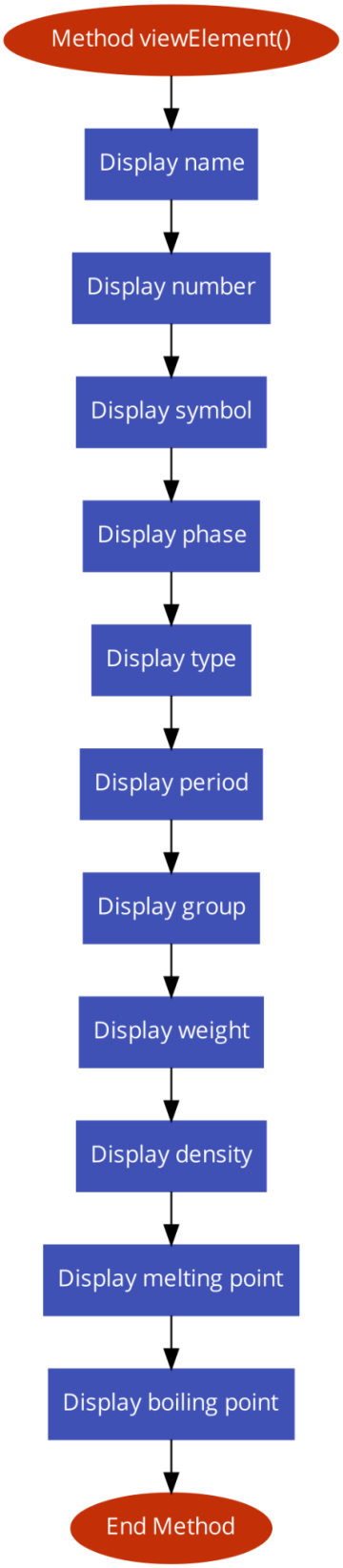
My program does not have an interface and depends on the user interacting with the console. My program will use Java, which is object-oriented, and will have 5 classes. Only one of these classes will revolve around an object which is the Element class that creates an element object with an element's attributes as its instance variables. The other 4 classes will be static classes (all methods and variables are static), with there being a main class that contains methods and variables that all the other classes will use. There other 3 static classes are View, Search, and Test (each one being the option that a student can choose from the home screen) and it has all the methods to do what the student desires.

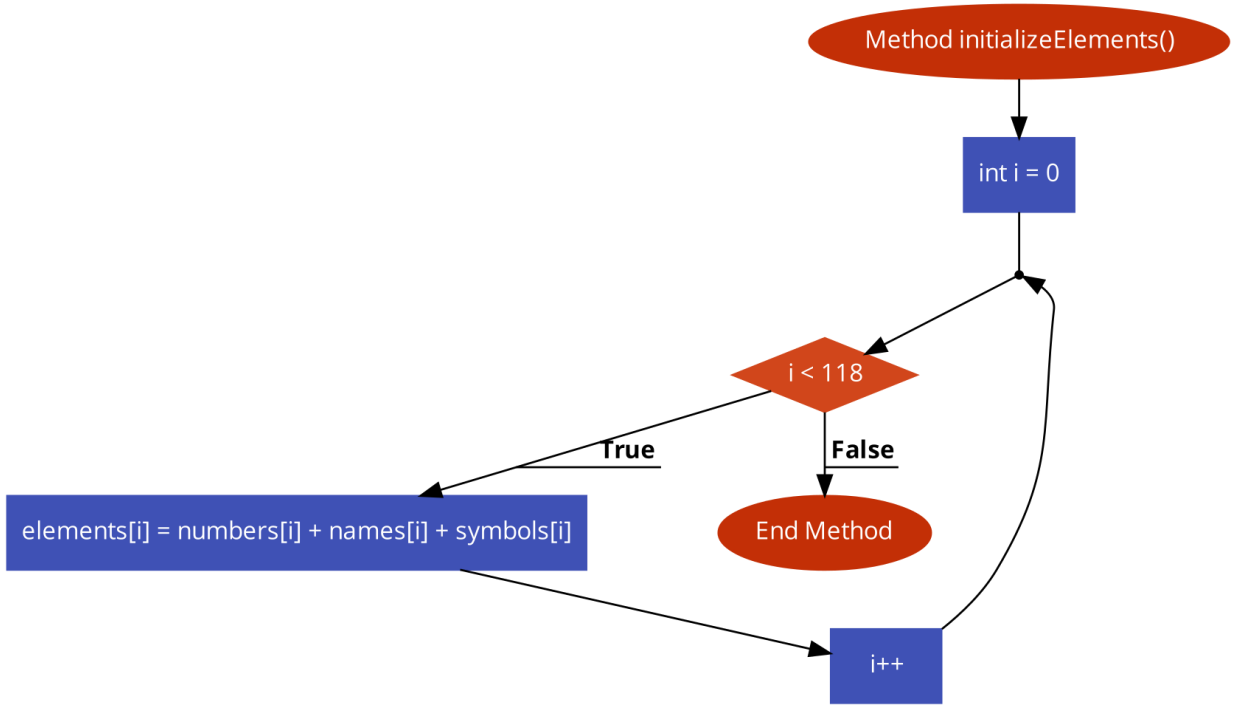
Element Class

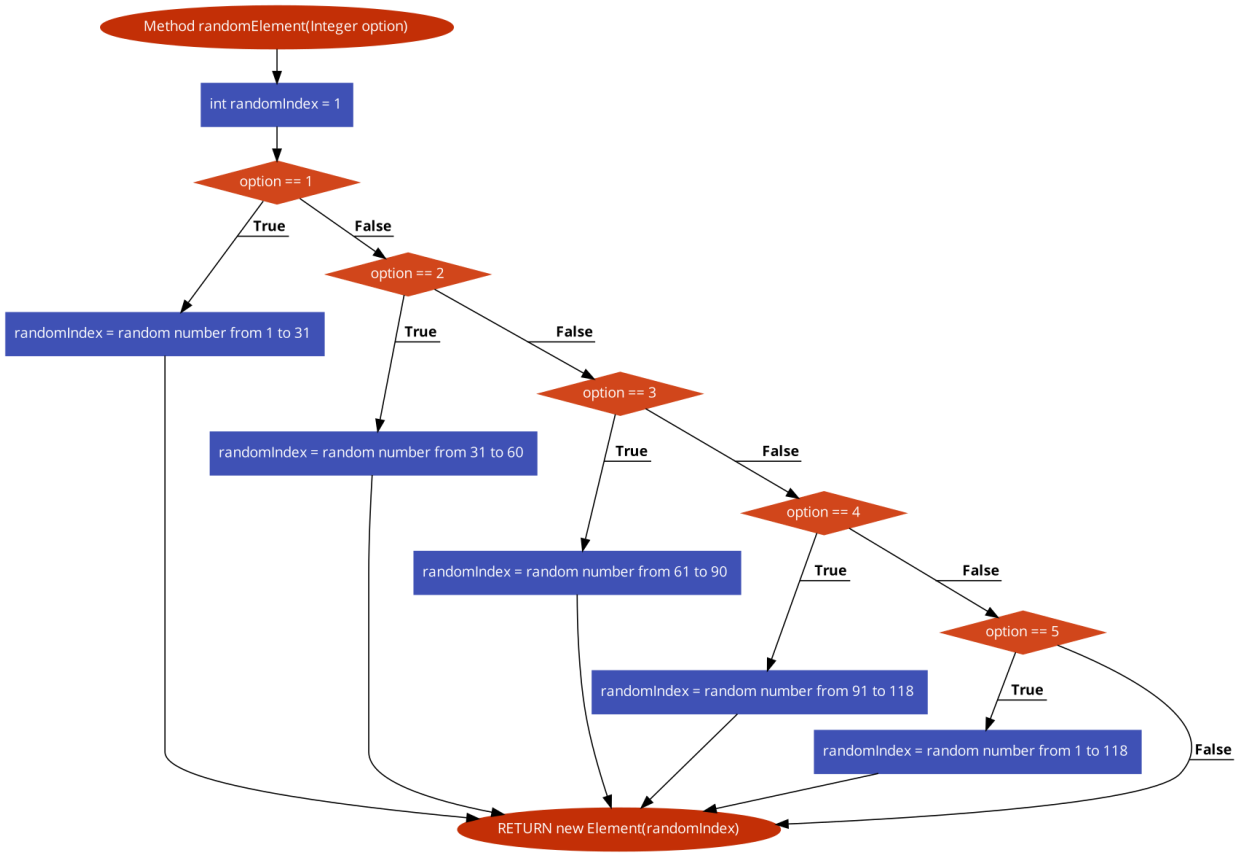
Flowchart




Constructor Element(Integer option)
option = option - 1
number = numbers[option]
name = names[option]
symbol = symbols[option]
phase = phases[option]
type = types[option]
period = periods[option]
group = groups[option]
weight = weights[option]
density = densities[option]
meltingPoint = meltingPoints[option]
boilingPoint = boilingPoints[option]
End Constructor

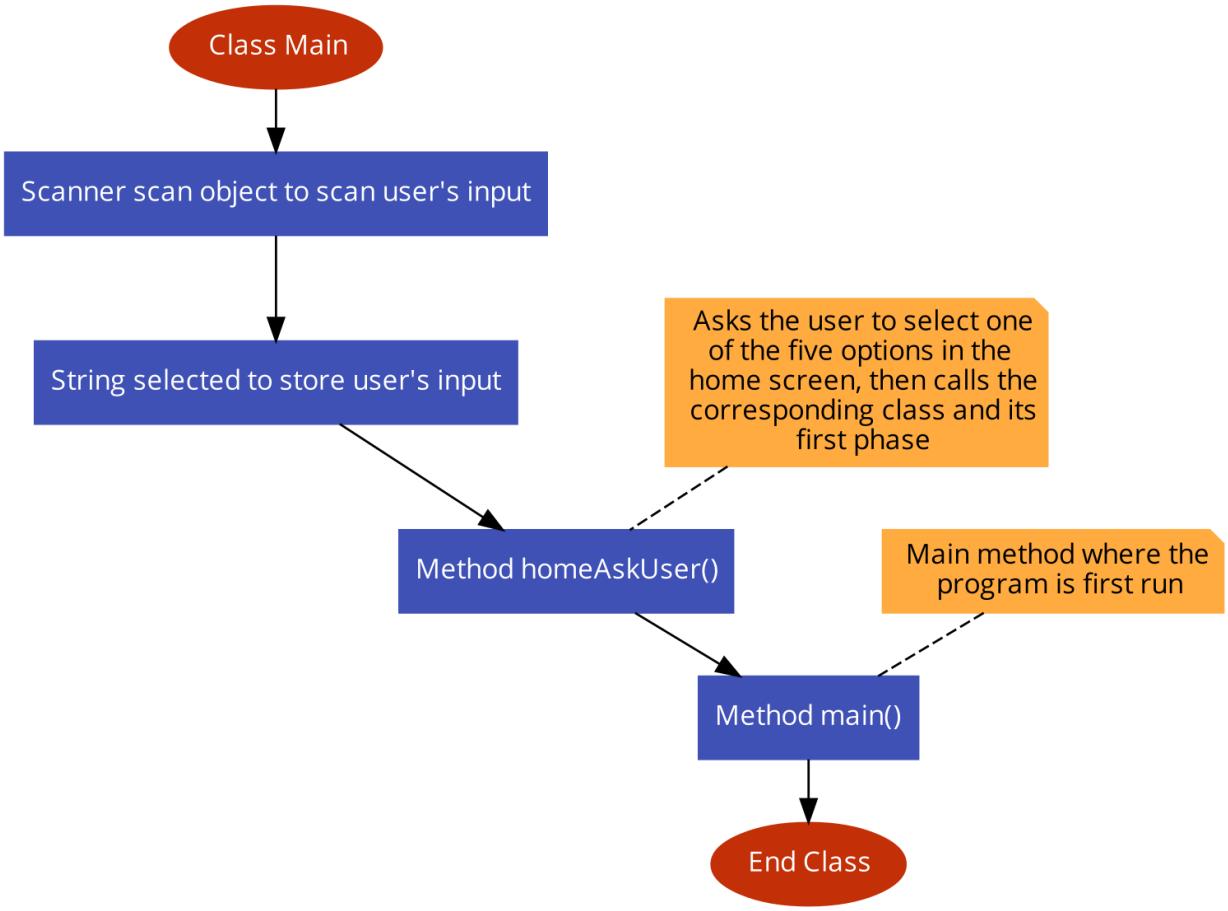

Method viewElement()
Display name
Display number
Display symbol
Display phase
Display type
Display period
Display group
Display weight
Display density
Display melting point
Display boiling point
End Method

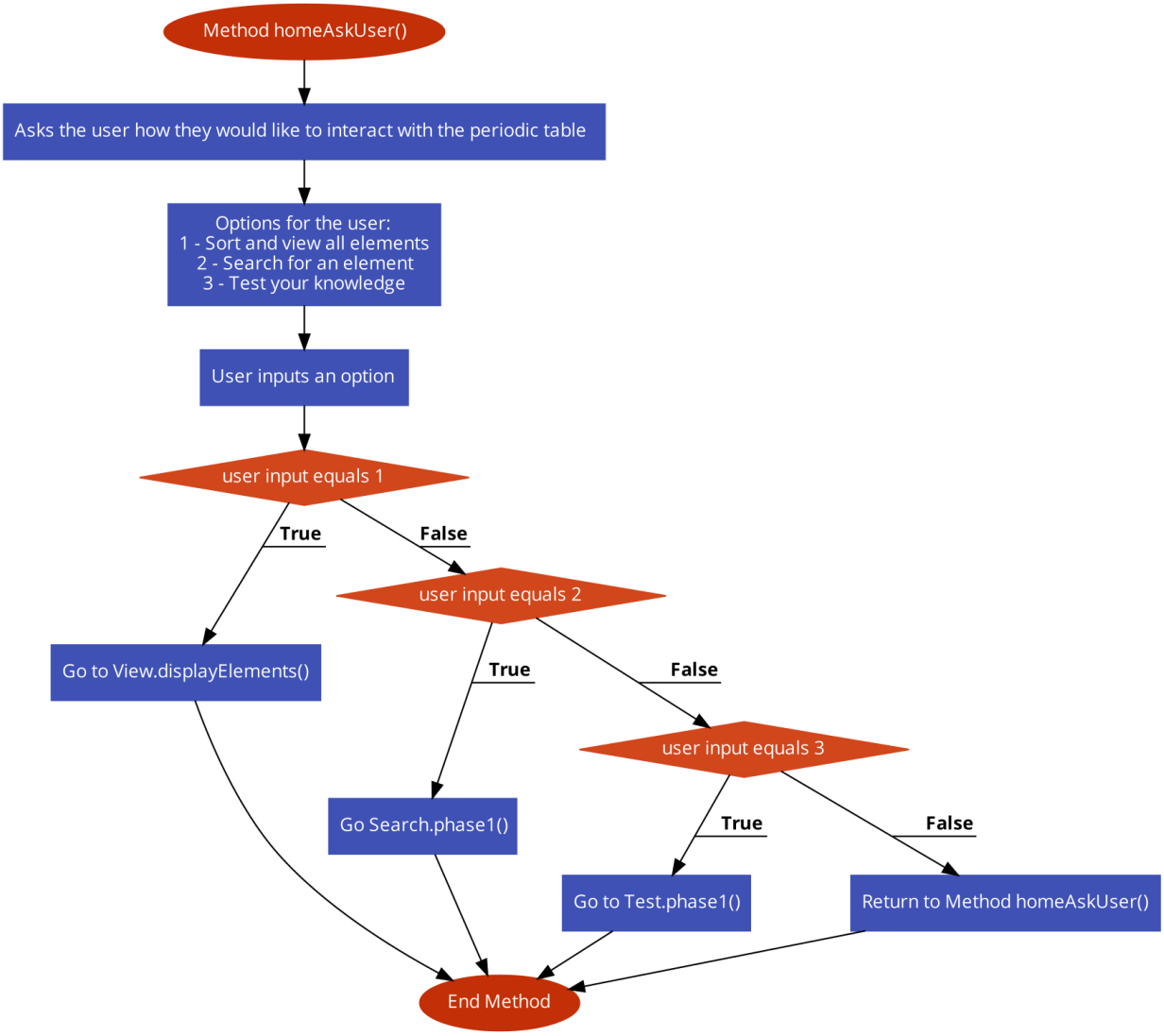

Method initializeElements()
int i = 0
i < 118
True
False
elements[i] = numbers[i] + names[i] + symbols[i]
End Method
i++

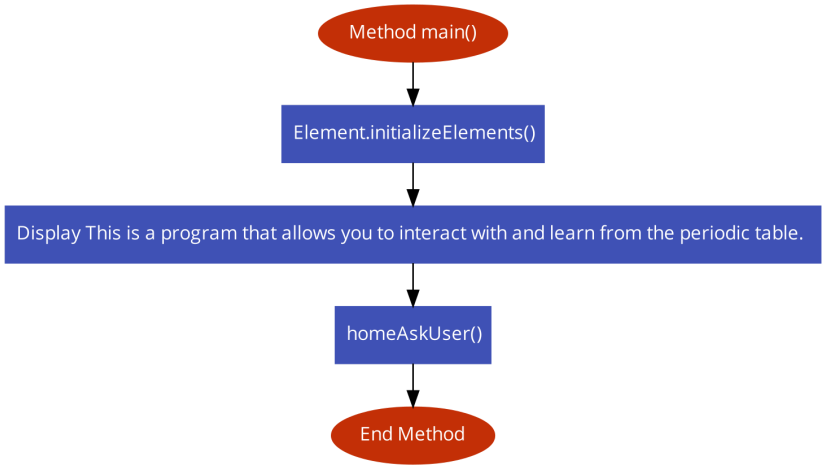

Method randomElement(Integer option)
int randomIndex = 1
option == 1
True
False
option == 2
randomIndex = random number from 1 to 31
True
False
option == 3
randomIndex = random number from 31 to 60
True
False
option == 4
randomIndex = random number from 61 to 90
True
False
option == 5
randomIndex = random number from 91 to 118
True
randomIndex = random number from 1 to 118
False
RETURN new Element(randomIndex)

Main Class

Flowchart




Method homeAskUser()
Asks the user how they would like to interact with the periodic table
Options for the user:
1 - Sort and view all elements
2 - Search for an element
3 - Test your knowledge
User inputs an option
user input equals 1
True
False
user input equals 2
Go to View.displayElements()
True
False
user input equals 3
Go Search.phase1()
True
False
Go to Test.phase1()
Return to Method homeAskUser()
End Method


Method main()
Element.initializeElements()
Display This is a program that allows you to interact with and learn from the periodic table.
homeAskUser()
End Method

Original Interface

Type the number of the option you want:
1 - View all elements
2 - Search for an element
3 - Test your knowledge

Problems

The prompting of the console was not very explicit, so it was very likely that a student would use this and get confused.

New Interface

This is a program that allows you to interact with and learn from the periodic table.

How would you like to interact with the elements of the periodic table?
---
1 - Sort and view all elements
2 - Search for an element
3 - Test your knowledge

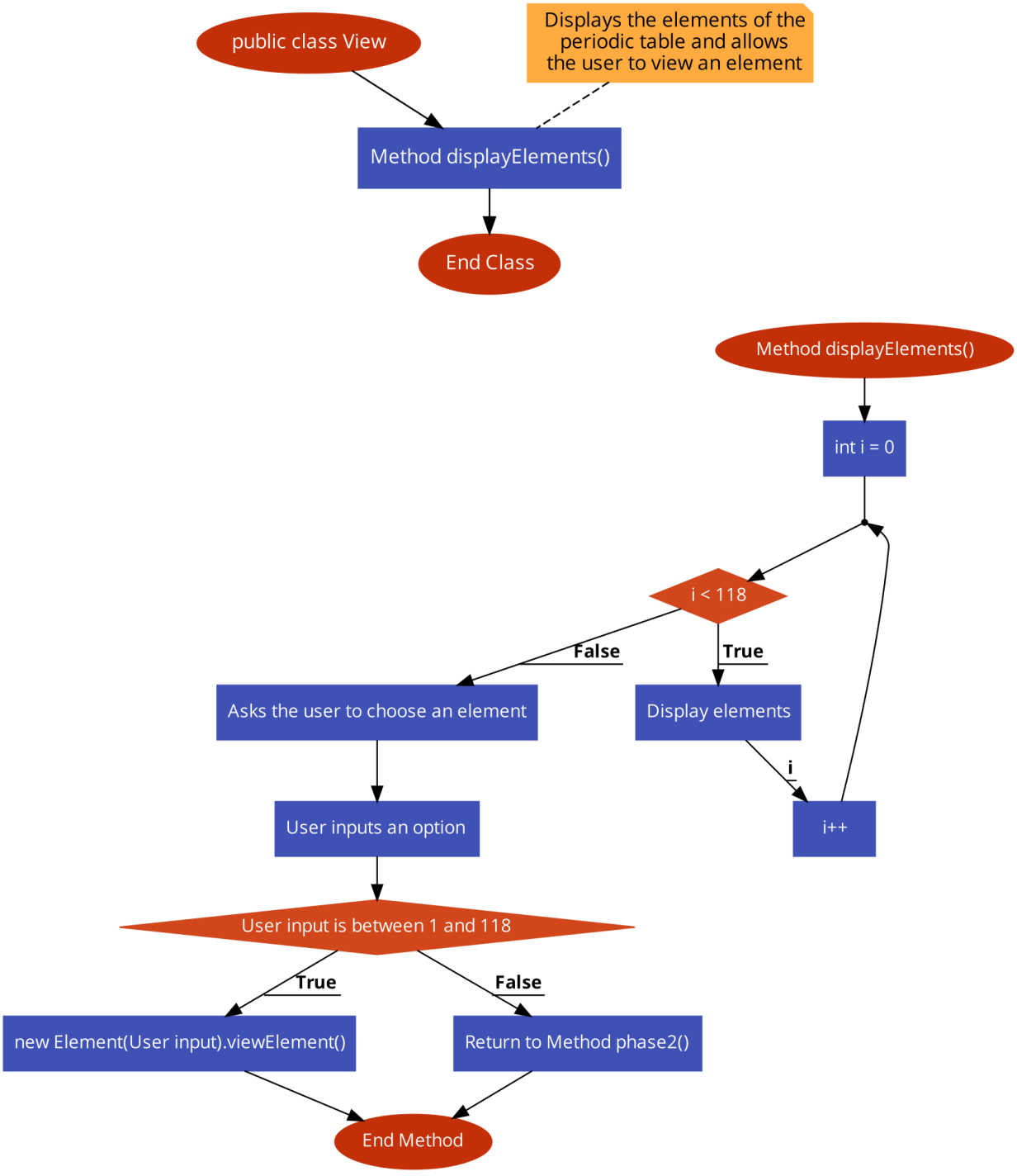
Type and enter the number of the option you want:

Inputs and Outputs

- If the user inputs 1, then the user will go to screen 1, which uses the view class.
- If the user inputs 2, then the user will go to screen 2, which uses the search class.
- If the user inputs 3, then the user will go to screen 3, which uses the test class.
- If anything else is inputted, then the screen will be repeated.

Screen 1 - View Class

Flowchart



Interface

---

1 - Hydrogen (H)
2 - Helium (He)
3 - Lithium (Li)
4 - Beryllium (Be)
5 - Boron (B)
6 - Carbon (C)
7 - Nitrogen (N)
8 - Oxygen (O)
9 - Fluorine (F)
10 - Neon (Ne)
11 - Sodium (Na)
12 - Magnesium (Mg)
13 - Aluminum (Al)
14 - Silicon (Si)
15 - Phosphorus (P)
16 - Sulfur (S)
17 - Chlorine (Cl)
18 - Argon (Ar)
19 - Potassium (K)
20 - Calcium (Ca)
21 - Scandium (Sc)
22 - Titanium (Ti)
23 - Vanadium (V)
24 - Chromium (Cr)
25 - Manganese (Mn)
26 - Iron (Fe)
27 - Cobalt (Co)
28 - Nickel (Ni)
29 - Copper (Cu)
30 - Zinc (Zn)
31 - Gallium (Ga)
32 - Germanium (Ge)
33 - Arsenic (As)
34 - Selenium (Se)
35 - Bromine (Br)
36 - Krypton (Kr)
37 - Rubidium (Rb)
38 - Strontium (Sr)
39 - Yttrium (Y)
40 - Zirconium (Zr)
41 - Niobium (Nb)
42 - Molybdenum (Mo)
43 - Technetium (Tc)
44 - Ruthenium (Ru)
45 - Rhodium (Rh)
46 - Palladium (Pd)
47 - Silver (Ag)
48 - Cadmium (Cd)

49 - Indium (In)
50 - Tin (Sn)
51 - Antimony (Sb)
52 - Tellurium (Te)
53 - Iodine (I)
54 - Xenon (Xe)
55 - Cesium (Cs)
56 - Barium (Ba)
57 - Lanthanum (La)
58 - Cerium (Ce)
59 - Praseodymium (Pr)
60 - Neodymium (Nd)
61 - Promethium (Pm)
62 - Samarium (Sm)
63 - Europium (Eu)
64 - Gadolinium (Gd)
65 - Terbium (Tb)
66 - Dysprosium (Dy)
67 - Holmium (Ho)
68 - Erbium (Er)
69 - Thulium (Tm)
70 - Ytterbium (Yb)
71 - Lutetium (Lu)
72 - Hafnium (Hf)
73 - Tantalum (Ta)
74 - Tungsten (W)
75 - Rhenium (Re)
76 - Osmium (Os)
77 - Iridium (Ir)
78 - Platinum (Pt)
79 - Gold (Au)
80 - Mercury (Hg)
81 - Thallium (Tl)
82 - Lead (Pb)
83 - Bismuth (Bi)
84 - Polonium (Po)
85 - Astatine (At)
86 - Radon (Rn)
87 - Francium (Fr)
88 - Radium (Ra)
89 - Actinium (Ac)
90 - Thorium (Th)
91 - Protactinium (Pa)
92 - Uranium (U)
93 - Neptunium (Np)
94 - Plutonium (Pu)
95 - Americium (Am)
96 - Curium (Cm)
97 - Berkelium (Bk)
98 - Californium (Cf)
99 - Einsteinium (Es)

100 - Fermium (Fm)
101 - Mendelevium (Md)
102 - Nobelium (No)
103 - Lawrencium (Lr)
104 - Rutherfordium (Rf)
105 - Dubnium (Db)
106 - Seaborgium (Sg)
107 - Bogrium (Bh)
108 - Hassium (Hs)
109 - Meitnerium (Mt)
110 - Darmstadtium (Ds)
111 - Roentgenium (Rg)
112 - Copernicium (Cn)
113 - Nihonium (Nh)
114 - Flerovium (Fl)
115 - Moscovium (Mc)
116 - Livermorium (Lv)
117 - Tennessine (Ts)
118 - Oganesson (Og)

Type and enter the element number you want to view:

Inputs and Outputs

- If the user inputs a number between 1 and 118, like for example, 1, then this will be displayed (depending on the user's input):

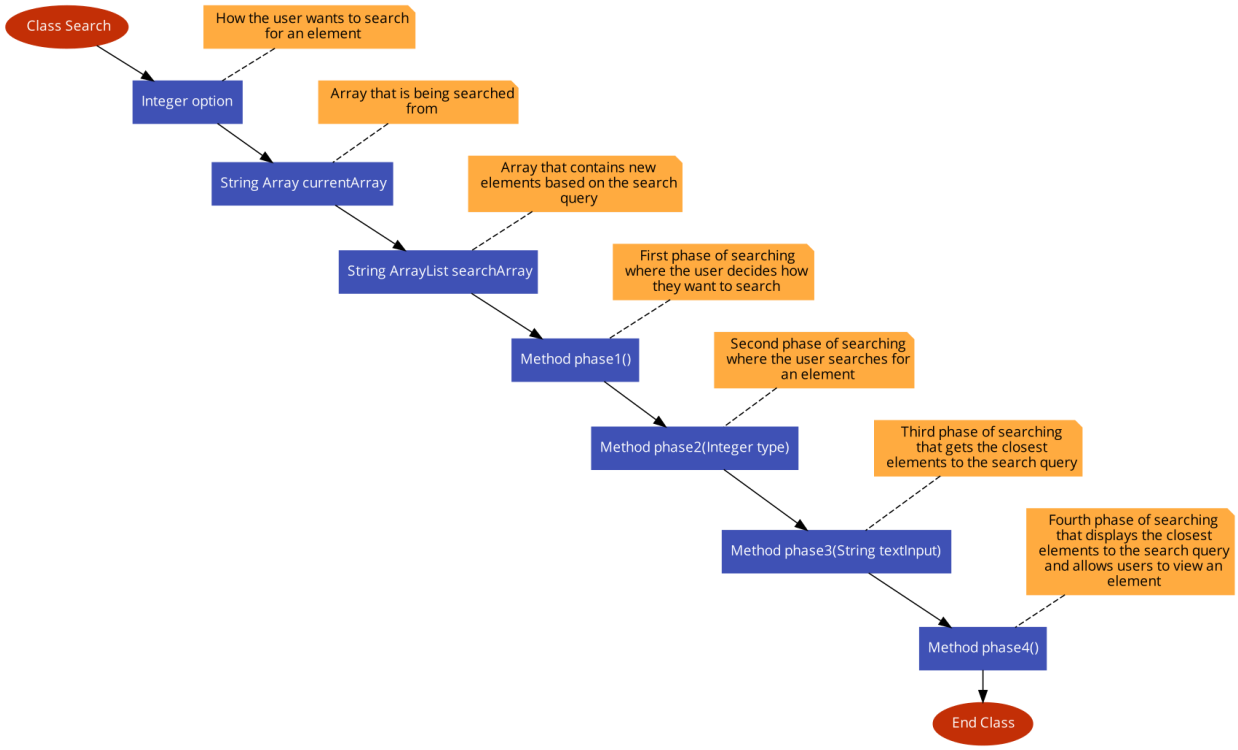
---
1 - Hydrogen (H):
Element Name: Hydrogen
Atomic Number: 1
Element Symbol: H
Phase type: Gas
Element type: Non-Metal
Element Period: 1
Element Group: No definitive group
Weight: 1.00794 amu
Density: 0.00008988 g/L
Melting Point: -434.81°F
Boiling Point: -423.17°F

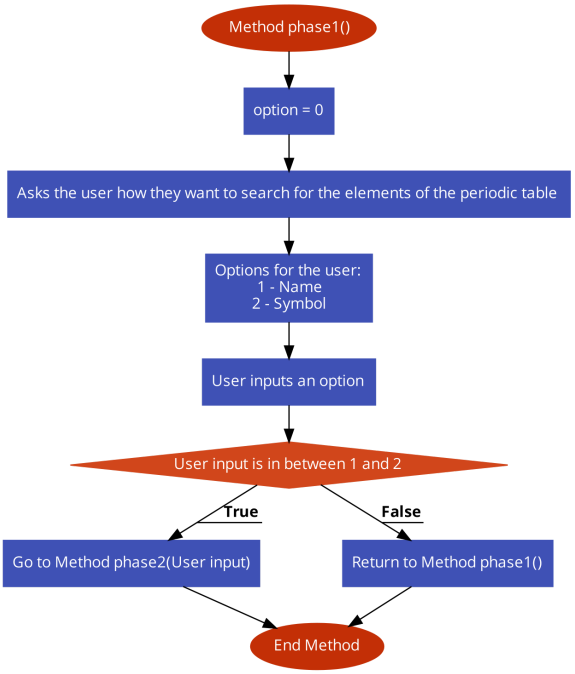
Type and enter anything when you want to go home:

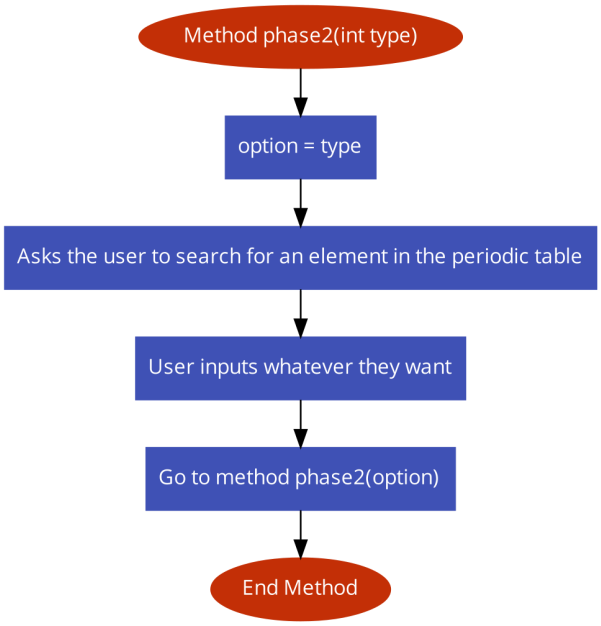
- If anything else is inputted, then the previous screen will be repeated.

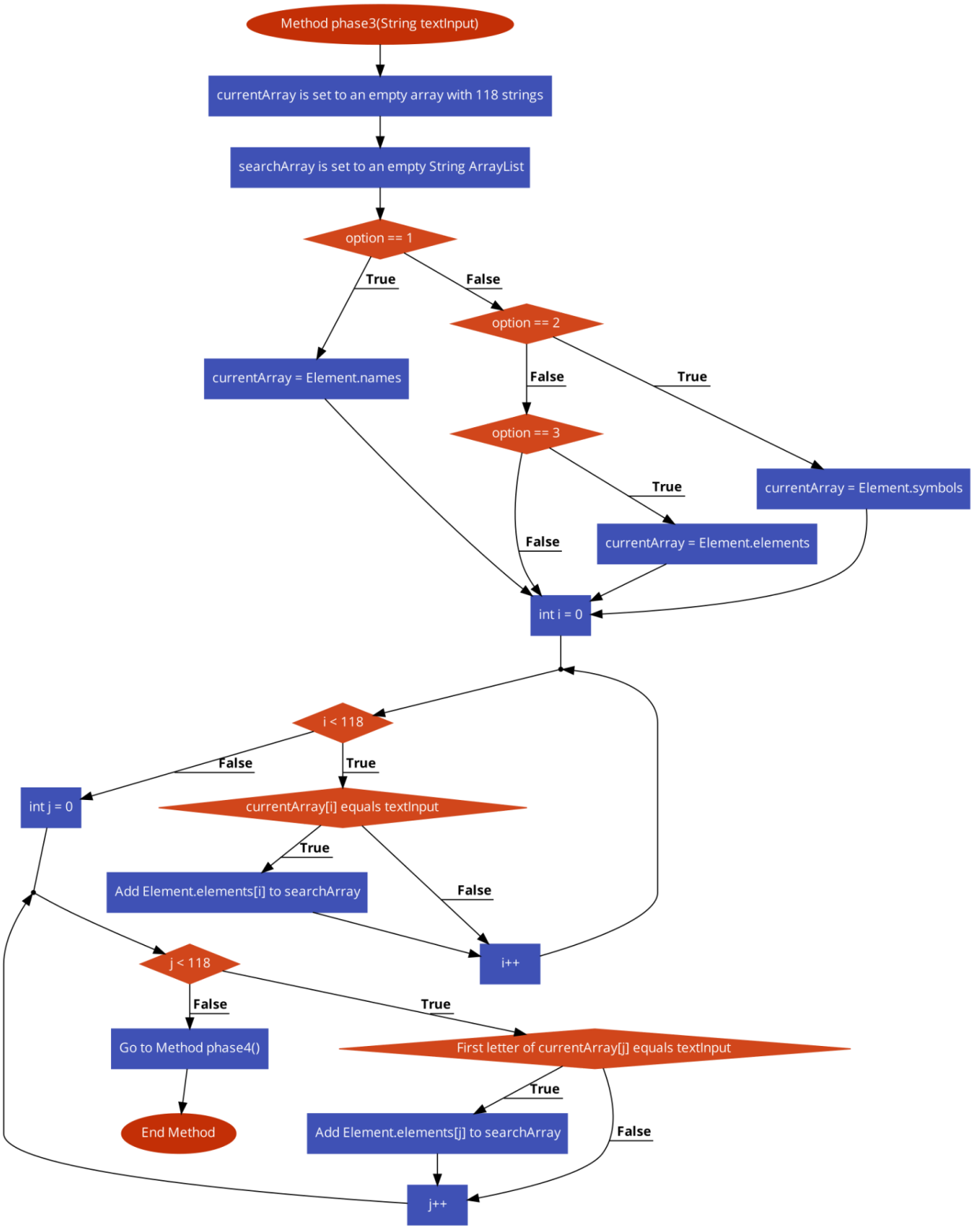
Screen 2 - Search Class

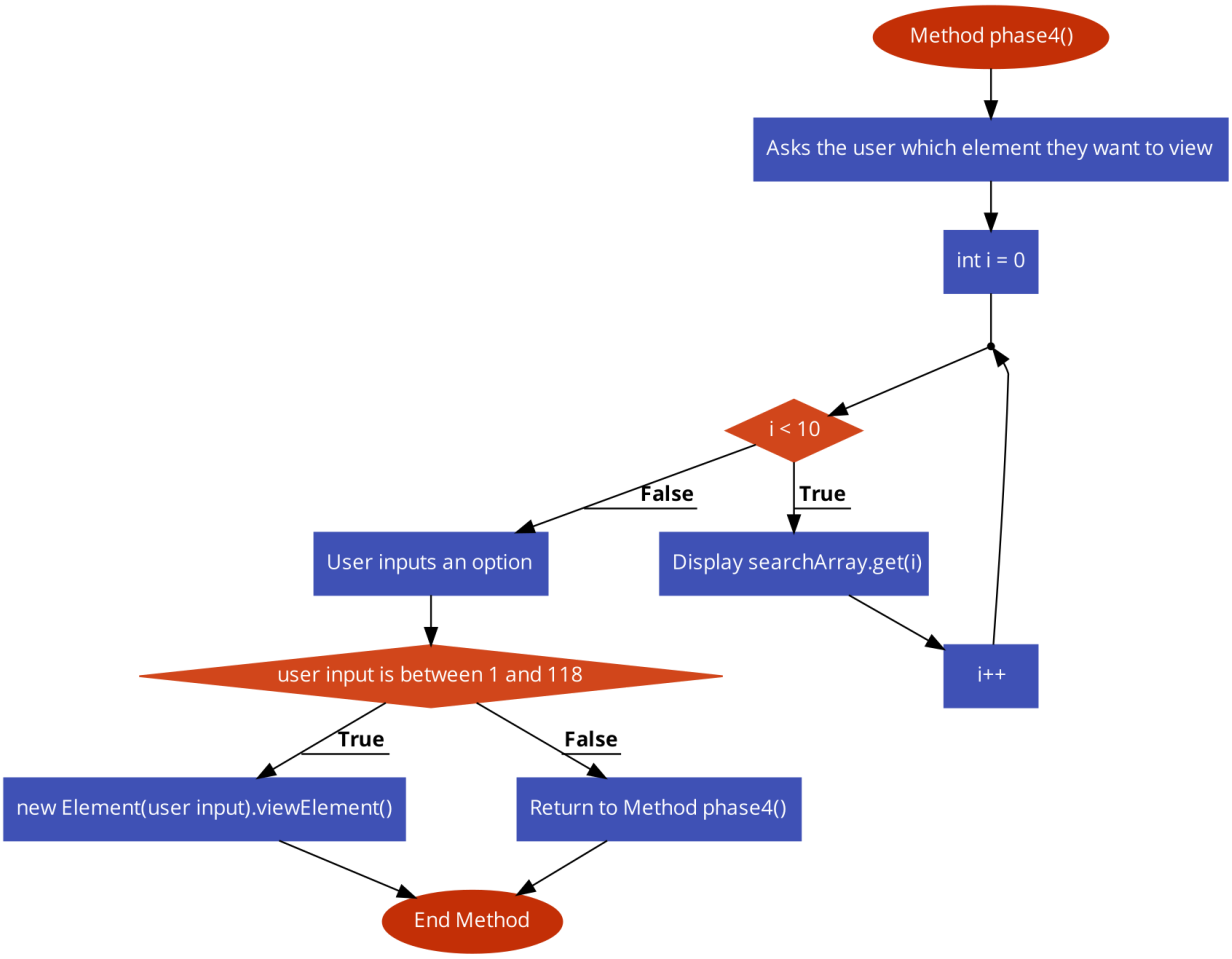
Flowchart








Method phase3(String textInput)
currentArray is set to an empty array with 118 strings
searchArray is set to an empty String ArrayList
option == 1
True
False
option == 2
currentArray = Element.names
False
True
option == 3
currentArray = Element.symbols
True
False
currentArray = Element.elements
int i = 0
i < 118
False
True
int j = 0
currentArray[i] equals textInput
True
Add Element.elements[i] to searchArray
False
i++
j < 118
False
True
Go to Method phase4()
First letter of currentArray[j] equals textInput
True
End Method
Add Element.elements[j] to searchArray
False
j++



Original Interface

Type whatever you want to search:

Problems

The original search function would search for just names, however, I thought that maybe students would only want to search for either names or symbols.

New Interface

How would you like to search for the elements of the periodic table?

---

1 - Name
2 - Symbol

Type and enter the option you want or type and enter -1 to go back:

Inputs and Outputs

- If the user inputs 1 or 2, then this will be displayed:

Type and enter whatever you want to search for:

- Anything that the user inputs will be searched for, and at most 11 elements will be displayed. If the user, for example, inputs Hydro, this will be displayed:

---
1 - Hydrogen (H)
2 - Helium (He)
67 - Holmium (Ho)
72 - Hafnium (Hf)
108 - Hassium (Hs)
3 - Lithium (Li)
4 - Beryllium (Be)
5 - Boron (B)
6 - Carbon (C)
7 - Nitrogen (N)
8 - Oxygen (O)

- If the user inputs a number between 1 and 118, like for example, 1, then this will be displayed (depending on the user's input):

---
1 - Hydrogen (H):
Element Name: Hydrogen
Atomic Number: 1
Element Symbol: H
Phase type: Gas
Element type: Non-Metal
Element Period: 1
Element Group: No definitive group
Weight: 1.00794 amu
Density: 0.00008988 g/L
Melting Point: -434.81°F
Boiling Point: -423.17°F

Type and enter anything when you want to go home:

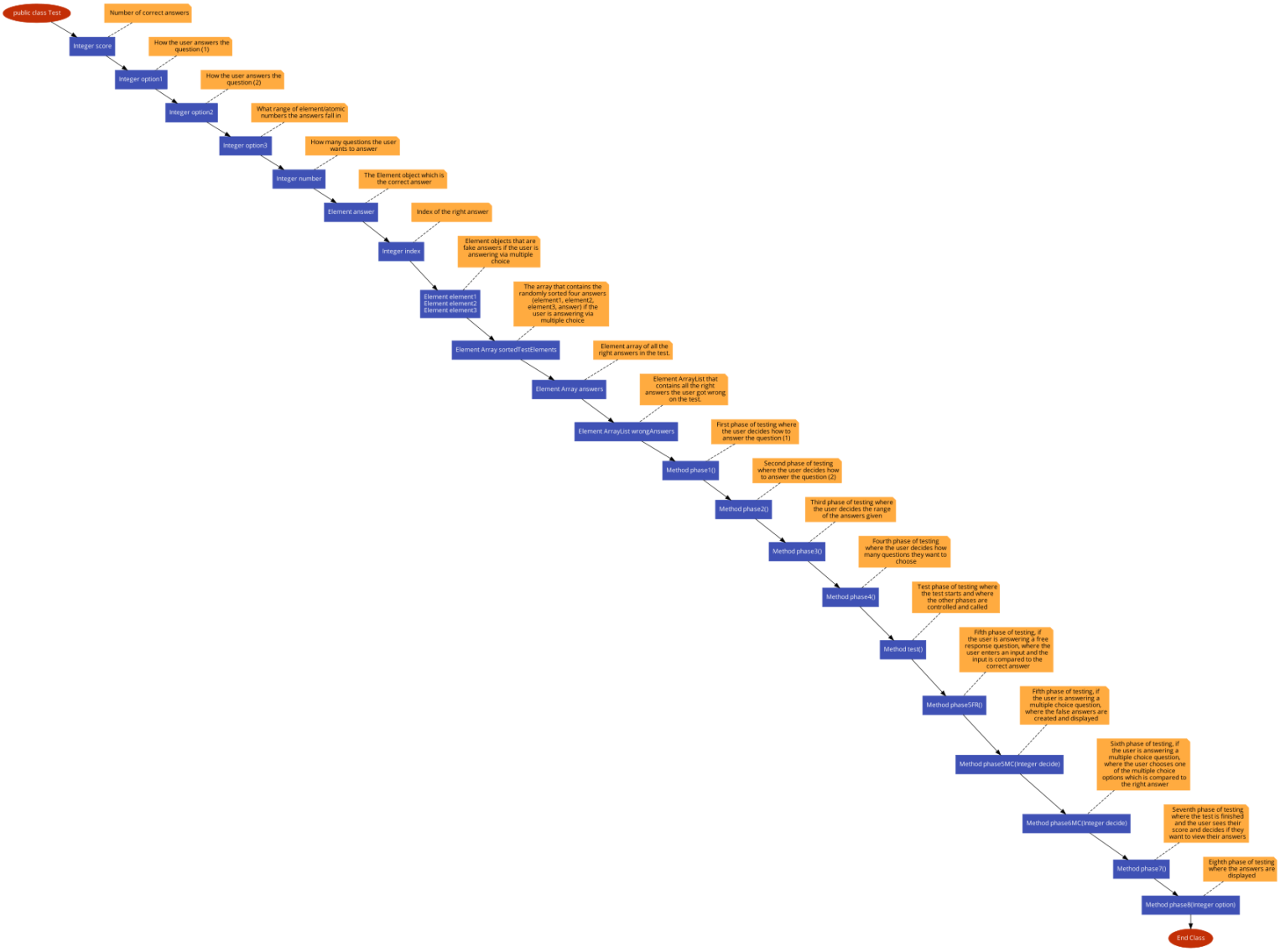
- If the user inputs anything, then the user will go back to the first screen.

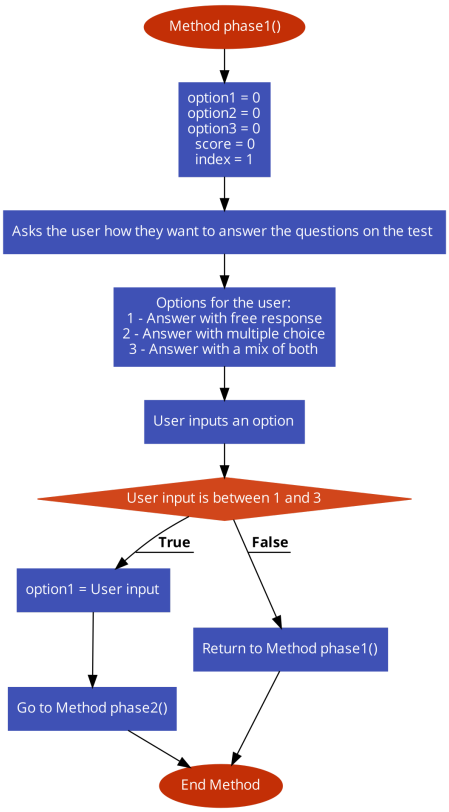
- If anything else is inputted, then the previous screen will be repeated.

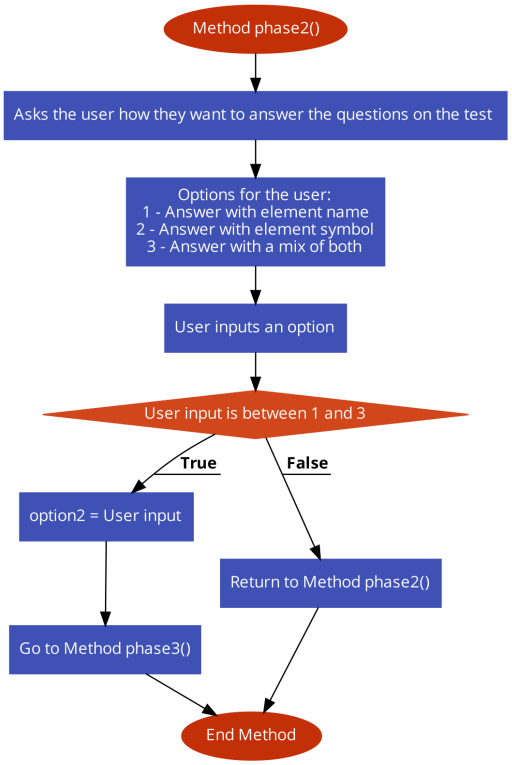
- If anything else is inputted, then the previous screen will be repeated.

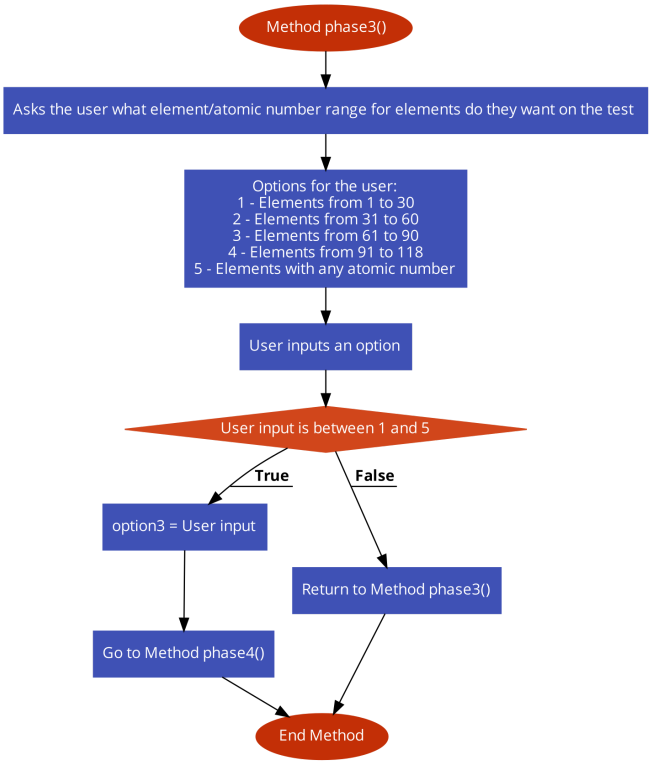
Screen 3 - Test Class

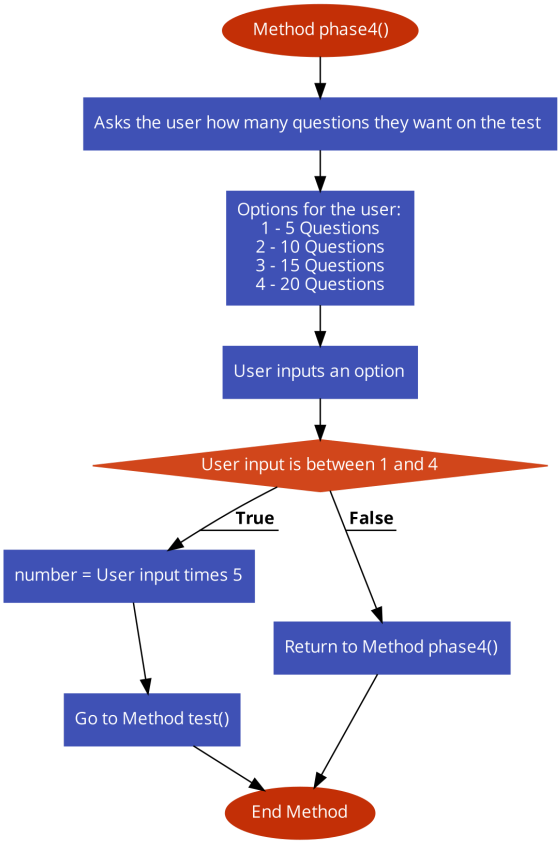
Flowchart

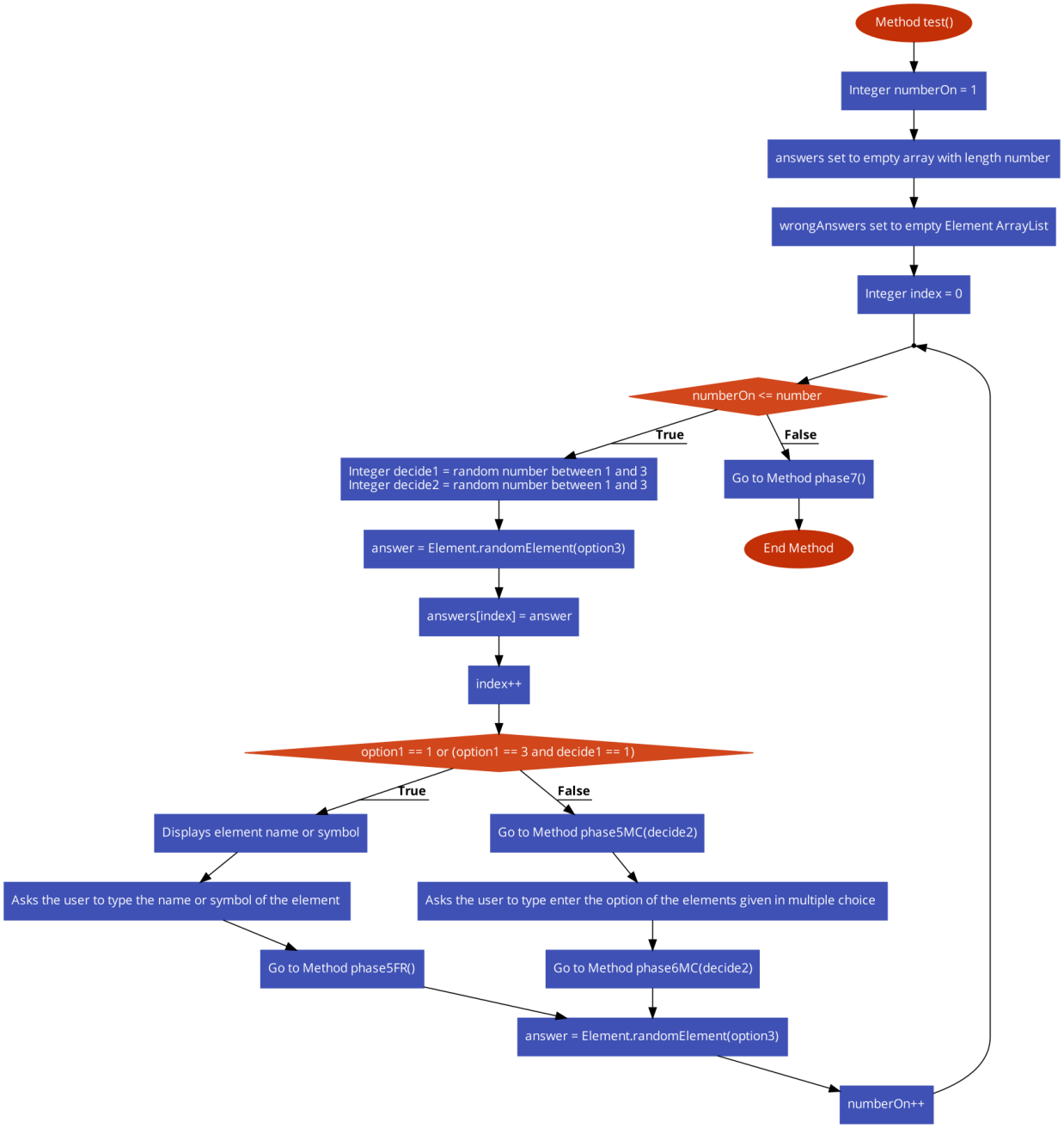


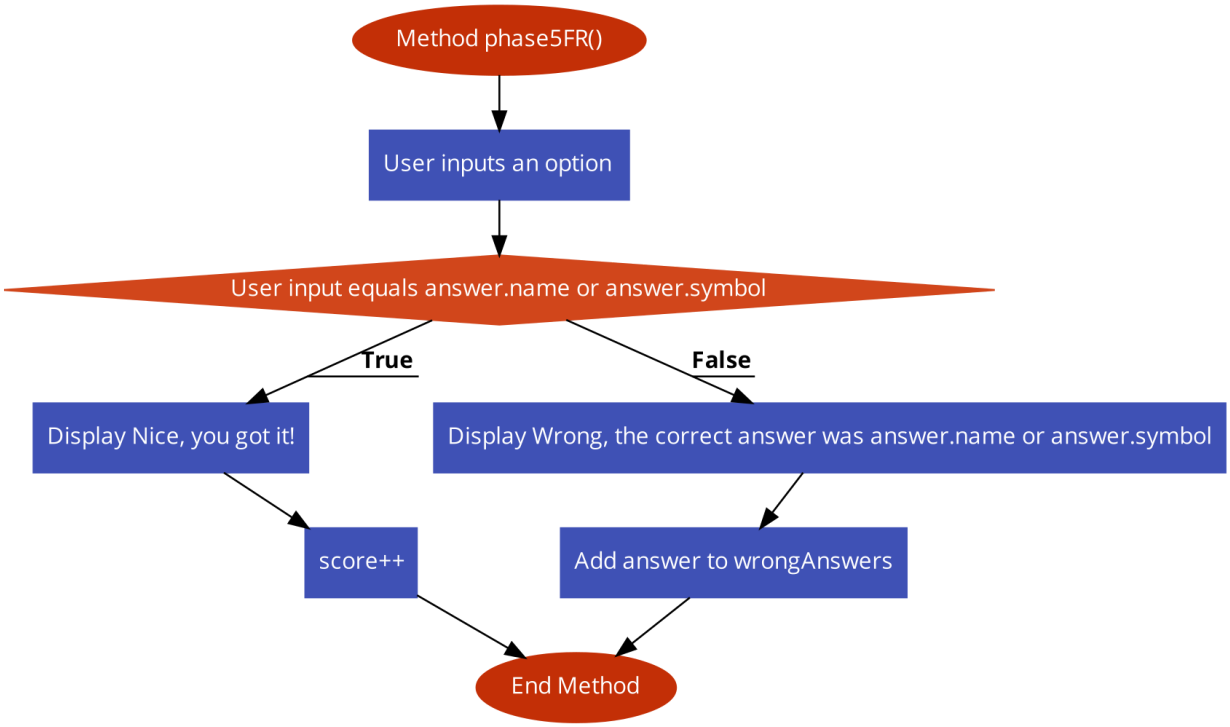

Method phase1()
option1 = 0
option2 = 0
option3 = 0
score = 0
index = 1
Asks the user how they want to answer the questions on the test
Options for the user:
1 - Answer with free response
2 - Answer with multiple choice
3 - Answer with a mix of both
User inputs an option
User input is between 1 and 3
True
False
option1 = User input
Return to Method phase1()
Go to Method phase2()
End Method

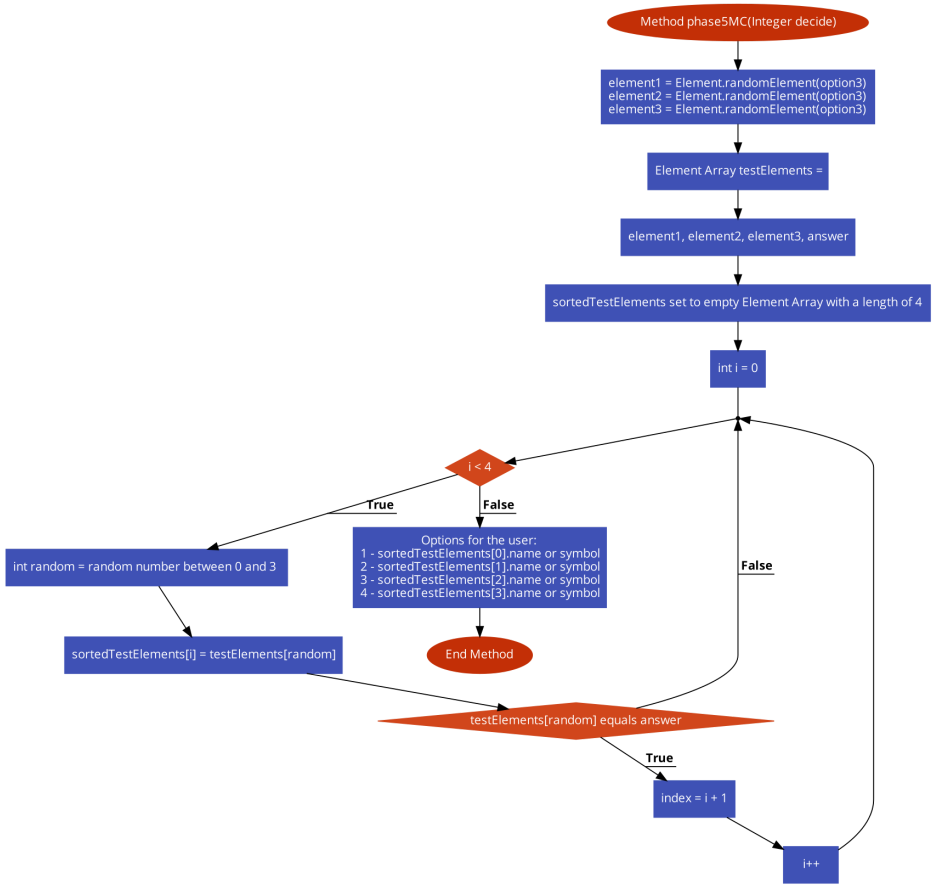

Method phase2()
Asks the user how they want to answer the questions on the test
Options for the user:
1 - Answer with element name
2 - Answer with element symbol
3 - Answer with a mix of both
User inputs an option
User input is between 1 and 3
True
False
option2 = User input
Return to Method phase2()
Go to Method phase3()
End Method

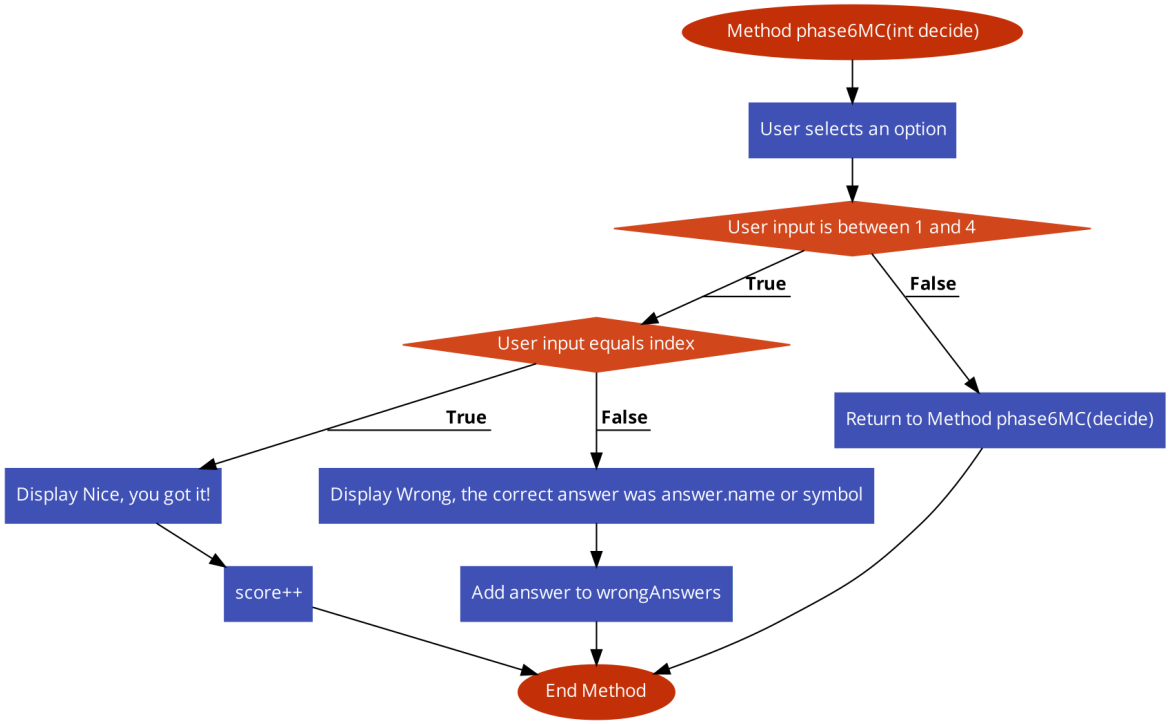

Method phase3()
Asks the user what element/atomic number range for elements do they want on the test
Options for the user:
1 - Elements from 1 to 30
2 - Elements from 31 to 60
3 - Elements from 61 to 90
4 - Elements from 91 to 118
5 - Elements with any atomic number
User inputs an option
User input is between 1 and 5
True
False
option3 = User input
Return to Method phase3()
Go to Method phase4()
End Method

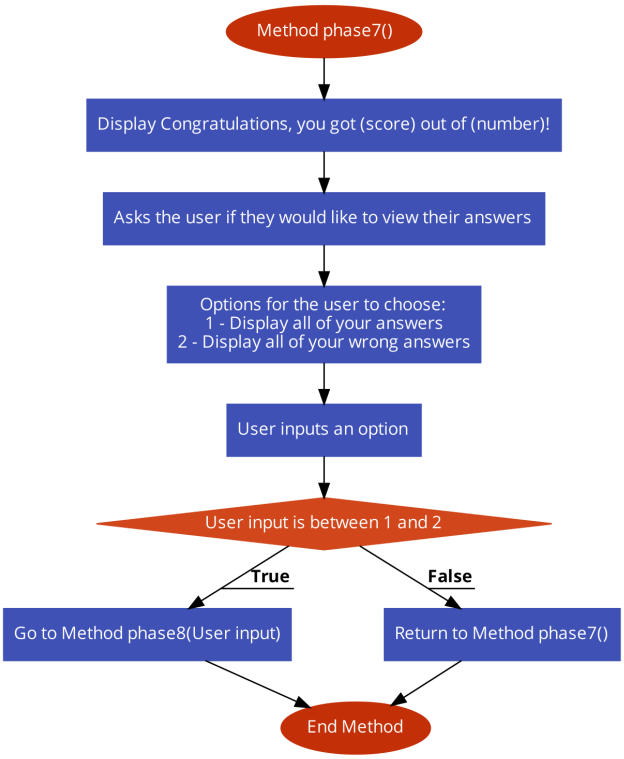

Method phase4()
Asks the user how many questions they want on the test
Options for the user:
1 - 5 Questions
2 - 10 Questions
3 - 15 Questions
4 - 20 Questions
User inputs an option
User input is between 1 and 4
True
False
number = User input times 5
Return to Method phase4()
Go to Method test()
End Method

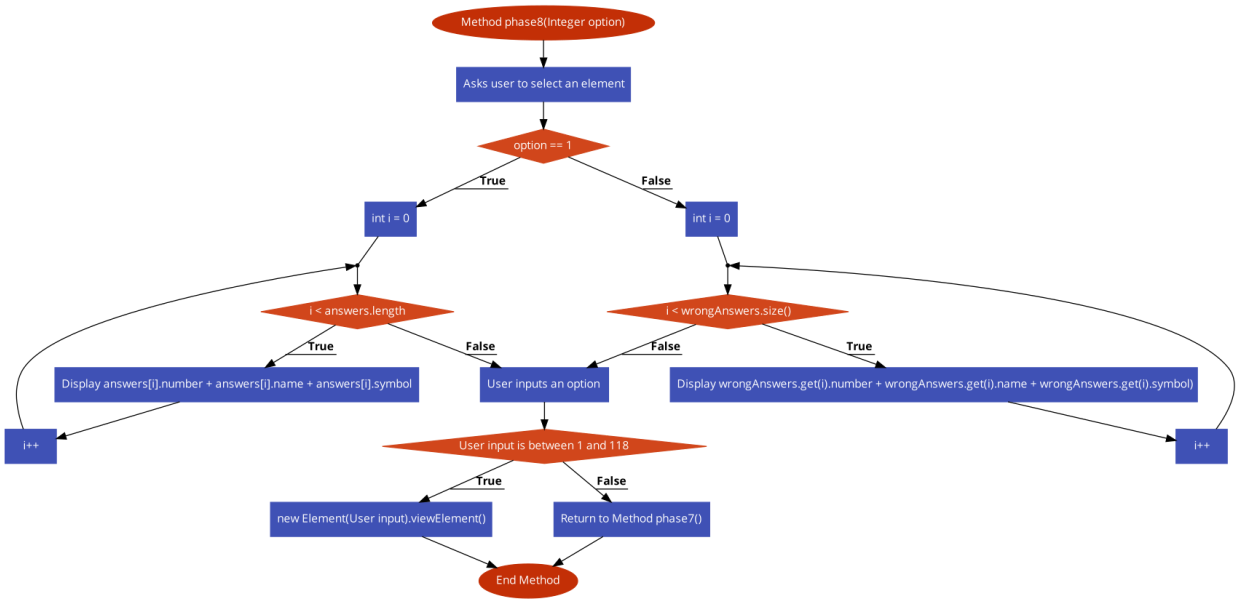

Method test()
Integer numberOn = 1
answers set to empty array with length number
wrongAnswers set to empty Element ArrayList
Integer index = 0
numberOn <= number
True
False
Integer decide1 = random number between 1 and 3
Integer decide2 = random number between 1 and 3
Go to Method phase7()
End Method
answer = Element.randomElement(option3)
answers[index] = answer
index++
option1 == 1 or (option1 == 3 and decide1 == 1)
True
False
Displays element name or symbol
Go to Method phase5MC(decide2)
Asks the user to type the name or symbol of the element
Asks the user to type enter the option of the elements given in multiple choice
Go to Method phase5FR()
Go to Method phase6MC(decide2)
answer = Element.randomElement(option3)
numberOn++


Method phase5FR()
User inputs an option
User input equals answer.name or answer.symbol
True
False
Display Nice, you got it!
Display Wrong, the correct answer was answer.name or answer.symbol
score++
Add answer to wrongAnswers
End Method


Method phase5MC(Integer decide)
element1 = Element.randomElement(option3)
element2 = Element.randomElement(option3)
element3 = Element.randomElement(option3)
Element Array testElements =
element1, element2, element3, answer
sortedTestElements set to empty Element Array with a length of 4
int i = 0
i < 4
True
False
int random = random number between 0 and 3
Options for the user:
1 - sortedTestElements[0].name or symbol
2 - sortedTestElements[1].name or symbol
3 - sortedTestElements[2].name or symbol
4 - sortedTestElements[3].name or symbol
sortedTestElements[i] = testElements[random]
End Method
testElements[random] equals answer
False
True
index = i + 1
i++


Method phase6MC(int decide)
User selects an option
User input is between 1 and 4
True
False
User input equals index
True
False
Return to Method phase6MC(decide)
Display Nice, you got it!
Display Wrong, the correct answer was answer.name or symbol
score++
Add answer to wrongAnswers
End Method


Method phase7()
Display Congratulations, you got (score) out of (number)!
Asks the user if they would like to view their answers
Options for the user to choose:
1 - Display all of your answers
2 - Display all of your wrong answers
User inputs an option
User input is between 1 and 2
True
False
Go to Method phase8(User input)
Return to Method phase7()
End Method



Original Interface

Type the option of the number of questions you want to answer:
1 - 5 Questions
2 - 10 Questions
3 - 15 Questions
4 - 20 Questions
5 - Custom

Problems

The test function was way too simple for the main feature of the program, so I added more customizability to help students memorize elements of the periodic table.

New Interface

How would you like to answer the questions on the test?
---
1 - Answer with free response
2 - Answer with multiple choice
3 - Answer with a mix of both

Type and enter the number of the option you want:

Inputs and Outputs

- If the user inputs 1, 2, or 3, then this will be displayed:

How would you like to answer the questions on the test?
---
1 - Answer with element name
2 - Answer with element symbol
3 - Answer with a mix of both

Type and enter the number of the option you want:

- If the user inputs 1, 2, or 3, then this will be displayed:

What atomic number range for elements do you want to be on the test?
---
1 - Elements from 1 to 30
2 - Elements from 31 to 60
3 - Elements from 61 to 90
4 - Elements from 91 to 118
5 - Elements with any atomic number

Type and enter the number of the option you want:

- If the user inputs 1, 2, 3, 4, or 5, then this will be displayed:

How many questions do you want on the test?
---
1 - 5 Questions
2 - 10 Questions
3 - 15 Questions
4 - 20 Questions

Type and enter the number of the option you want:

- If the user inputs 1, 2, 3, or 4, then the test will begin. Depending on what the user chose previously, the test will act differently. For example, if the user inputted 1, 1, 5, and 1, then this will be displayed 5 times with different random elements:

Type and enter the name of the element "Ni" (or type and enter 0 to receive a hint):

- Another example would be if the user inputted 2, 1, 5, and 1, then this will be displayed 5 times with different random elements:

---
1 - Germanium
2 - Dubnium
3 - Lawrencium
4 - Copernicium

Type and enter the option of the element "Lr" (or type and enter 0 to receive a hint):

- In the multiple choice part, if anything other than 0, 1, 2, 3, or 4 is inputted, then the previous screen will be displayed.
- If the number, for multiple choice, or the name/symbol, for free response, is correct, then something like this will be displayed:

Nice, you got it!

- If the number, for multiple choice, or the name/symbol, for free response, is wrong, then something like this will be displayed:

Wrong, the correct answer was Barium.

- After the test, the user will see their score then something like this will be displayed:

Congratulations, you got 4 out of 5! Would you like to view your answers?
---
1 - Display all of your answers
2 - Display all of your wrong answers

Type and enter the number of the option you want:

- If the user inputs 1, then something like this will be displayed (depending on the user's results on the test):

These were all the elements asked for in the test:
---
85 - Astatine (At)
103 - Lawrencium (Lr)
39 - Yttrium (Y)
47 - Silver (Ag)
81 - Thallium (Tl)

Type and enter the element number you want to view:

- If the user inputs 2, then something like this will be displayed (depending on the user's results on the test):

These were all the elements you got wrong on the test:
---
81 - Thallium (Tl)

Type and enter the element number you want to view:

- If anything else is inputted, then the previous screen will be repeated.

        - If anything else is inputted, then the previous screen will be repeated.
    - If anything else is inputted, then the previous screen will be repeated.
- If anything else is inputted, then the previous screen will be repeated.

Criterion C: Development

Complex Techniques

In order of least complex to most complex

Technique	Class	Summary
Parallel Arrays	Element	Imported data from code.org and used the arrays for the instance variables and constructors.
Error Handling	Main	Borrowed a method from Free Code Camp to check whether the user input's string can be converted to an integer.
Recursion	All	Used to repeat a prompt an infinite amount of times every time the user improperly inputs something.
2D Array	Test	Two rows for all the right answers on the test, the first row having all the right answers, and the second one having all the right answers the user got wrong.
Nested Loops	Search	Used to check every 2 substrings of the user input to see if it matches with an element.

1. Parallel Arrays - Element Class

Code

```
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"};

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"};

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"};

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", "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"};

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"};

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"};

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"};

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"};

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"};

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"};

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"};


/**
* Constructor for creating a new Element object based on the lists.
*
* @param int option - the element number of the Element object.
*/
public Element(int option) {
```

```
    option--;
    number = numbers[option];
    name = names[option];
    symbol = symbols[option];
    phase = phases[option];
    type = types[option];
    period = periods[option];
    group = groups[option];
    weight = weights[option];
    density = densities[option];
    meltingPoint = meltingPoints[option];
    boilingPoint = boilingPoints[option];
}
```

Explanation

First I had to get data for all the elements of the periodic table. I used code.org's periodic table data column and I converted them all from JavaScript to Java. The arrays imported were of different types and that would not work because there were string values ("Unknown") in some of the primitive arrays. So to fix this, I converted these arrays to strings, but I also decided to convert the other number arrays to strings because no calculations were done using the arrays. Each index responds to the same element, so I used these arrays as parallel arrays.

2. Error Handling - Main Class

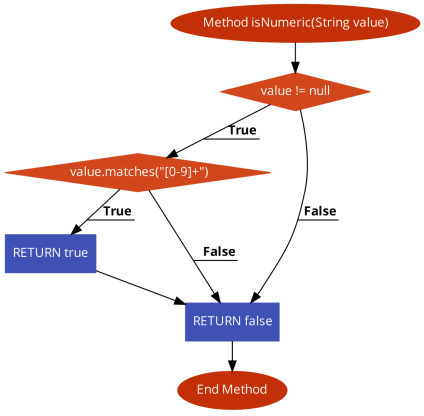
Code

```
/**
* Checks if a given string can be converted to a number.
*
* 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/.
&
* @param String value - the string to be checked.
*
* @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).
*/
public static boolean isNumeric(String str){
   return str != null && str.matches("[0-9]+");
}
```

Explanation

The user will only input integers when prompted, but to ensure that no errors occur, my code will take the user's input as a string so that anything that the user types will be accepted. Strings need to be converted to integers to see whether a user input falls within a certain number range. So to prevent an error, before converting, I need to check to see if it can be converted, so after some research, I discovered the matches() method.

Flowchart



3. Recursion - All Classes

All methods in my program that have “user” in the identifier uses recursion. This is because my code only accepts certain inputs, so if the user improperly inputs something, recursion will be used to get the program to prompt the user again. However, when the method is called again, it will have a different parameter. There are over 10 methods that use recursion in my program, and they are all there to prevent errors when there is improper user input. The code below is an example from the main class.

Code

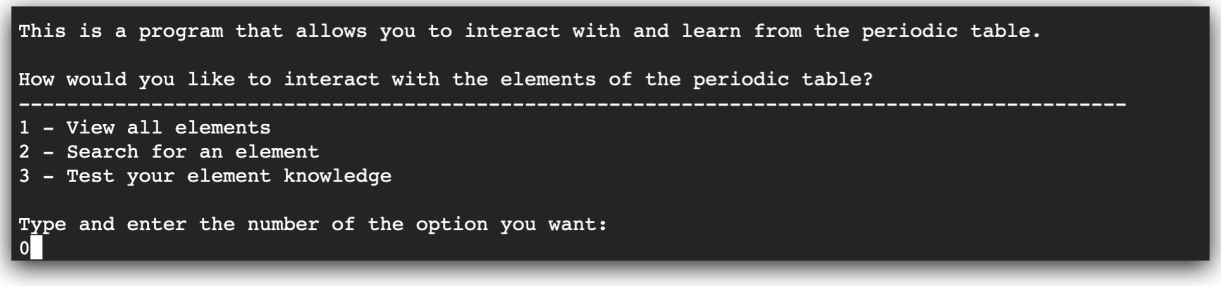
```
/**
* Asks the user to select one of the five options in the home screen, then calls the
corresponding class and its first phase.
*
* @param boolean tryAgain - whether the method was called again due to an invalid user input
*/
public static void userHome(boolean tryAgain) {
   printLine(false);
   if(first) {
       System.out.println("This is a program that allows you to interact with and learn from
the periodic table.\n");
       first = false;
   }
   System.out.println("How would you like to interact with the elements of the periodic
table?");

System.out.println("------------------------------------------------------------------------
--------------------");
   System.out.println("1 - View all elements");
   System.out.println("2 - Search for an element");
   System.out.println("3 - Test your element knowledge");

   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:");

   selected = scan.nextLine();

   if(selected.equals("1")) {View.displayElements(false);}
   else if(selected.equals("2")) {Search.userChooseSearch(false);}
   else if(selected.equals("3")) {Test.userChooseOption1(false);}
   else {
       userHome(true);
   }
```

```
}
```

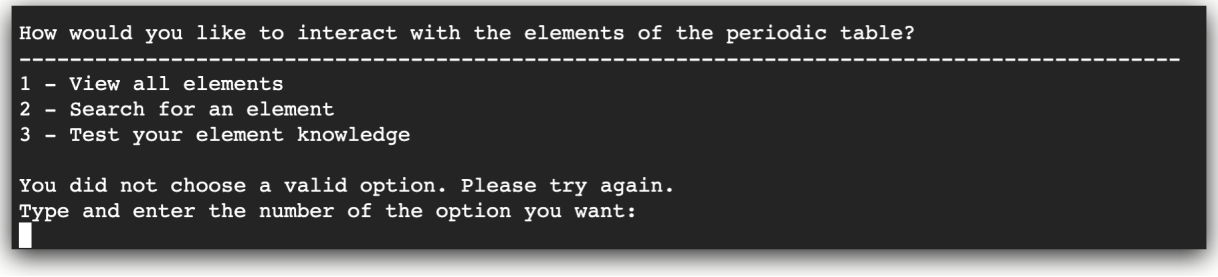
Explanation

Whenever this method is called from another method, tryAgain is set to false, but if the method is called from this method, tryAgain is set to true. The method will continue calling itself if the base case is not met, as long as the user inputs something that is not accepted.

For example, if I enter 0 in the home screen:



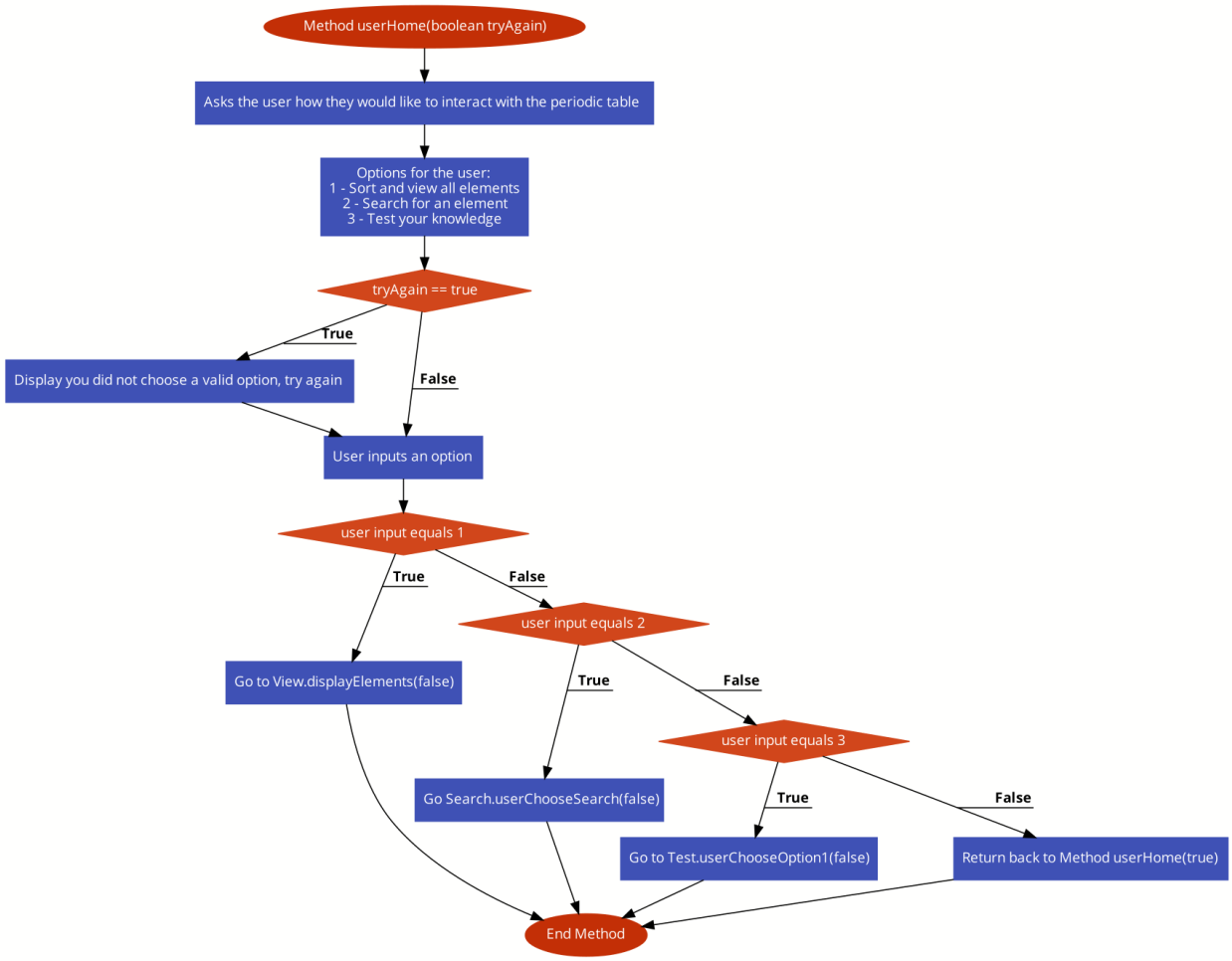
Then, the console will repeat the method:



Debugging and Justification

I knew from the design stage that I would need to use recursion and recall the method if there was improper user input, but at first, there was no boolean tryAgain and it would simply just call itself. However, I decided to add tryAgain because I felt that it did not make things clear if the user inputted something wrong.

Flowchart


Method userHome(boolean tryAgain)
Asks the user how they would like to interact with the periodic table
Options for the user:
1 - Sort and view all elements
2 - Search for an element
3 - Test your knowledge
tryAgain == true
True
False
Display you did not choose a valid option, try again
User inputs an option
user input equals 1
True
False
user input equals 2
Go to View.displayElements(false)
True
False
user input equals 3
Go Search.userChooseSearch(false)
True
False
Go to Test.userChooseOption1(false)
Return back to Method userHome(true)
End Method

4. 2D Array - Test Class

Code

```
// 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.
private static Element[][] answers;

/**
* Test phase of testing where the test starts and where the other phases are controlled and called.
*/
private static void test() {
   answers = new Element[2][number];
   index = 0;

   Main.printLine(false);
   while(numberOn <= number && !done) {
       hintPhase = 0;

       do {
           answer = Element.randomElement(option3);
       } while(isPresent(answers[0], answer));

       answers[0][index] = answer;

       // If option1 is 3, then it randomly becomes either 1 or 2.
       int decide1 = (int) (Math.random() * 3) + 1;

       // If option2 is 3, then it randomly becomes either 1 or 2.
       int decide2 = (int) (Math.random() * 3) + 1;

       if(option2 == 1 || (option2 == 3 && decide2 == 1)) {
           lookingFor = "name";
           lookingForVar = answer.name;
           given = "symbol";
           givenVar = answer.symbol;
       } else {
           lookingFor = "symbol";
           lookingForVar = answer.symbol;
           given = "name";
           givenVar = answer.name;
```

```
        }

        if(option1 == 1 || (option1 == 3 && decide1 == 1)) {
            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:");

System.out.println("----------------------------------------------------------------------------
--------------------");
            userAnswersFR();
        } else {
            createMC(decide2);
        }

        if(done) return;

        answer = Element.randomElement(option3);
        index++;
        numberOn++;
    }
    userChooseDisplay(false);
}

/**
* 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.
*/
private static void userAnswersFR() {
    Main.selected = Main.scan.nextLine();

    if(Main.selected.equals("-1")) {
        done = true;
        userChooseDisplay(false);
    } else if(Main.selected.equals("0")) {
        hint(1, 0);
    } else if(!Main.selected.matches("[a-zA-Z]+")) {
        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:");
        userAnswersFR();
    } else {
        if(Main.selected.equalsIgnoreCase(lookingForVar)) {
```

```
            System.out.println("\nNice, you got it!");
            score++;
        } else {
            System.out.println("\nWrong, the correct answer was " + lookingForVar + ".");
            isWrongAnswer = true;
            answers[1][index] = answer;
        }
    }
}

/**
* 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.
*
* @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.
*/
private static void userAnswersMC(int decide) {
    Main.selected = Main.scan.nextLine();
    Main.intSelected = Main.isNumeric(Main.selected) ? Integer.parseInt(Main.selected) : -10;

    if(Main.selected.equals("-1")) {
        done = true;
        userChooseDisplay(false);
    } else if(Main.selected.equals("0")) {
        hint(2, decide);
    } else if(Main.intSelected >= 1 && Main.intSelected <= 4) {
        if(Main.intSelected == rightIndex) {
            System.out.println("\nNice, you got it!");
            score++;
        } else {
            System.out.println("\nWrong, the correct answer was " + lookingForVar + ".");
            answers[1][index] = answer;
            isWrongAnswer = true;
        }
    } else {

System.out.println("\n--------------------------------------------------------------------------
----------------------");
        if(option2 == 1 || (option2 == 3 && decide == 1)) {
            System.out.println("1 - " + sortedTestElements[0].name);
            System.out.println("2 - " + sortedTestElements[1].name);
```

```
            System.out.println("3 - " + sortedTestElements[2].name);
            System.out.println("4 - " + sortedTestElements[3].name);
        } else {
            System.out.println("1 - " + sortedTestElements[0].symbol);
            System.out.println("2 - " + sortedTestElements[1].symbol);
            System.out.println("3 - " + sortedTestElements[2].symbol);
            System.out.println("4 - " + sortedTestElements[3].symbol);
        }
        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:");
        userAnswersMC(decide);
    }
}

/**
* Seventh phase of testing where the answers are displayed.
*
* @param boolean tryAgain - whether the method was called again due to an invalid user input.
*/
private static void displayElements(int option, boolean tryAgain) {
    if(option == 1) {
        Main.printLine(false);
        System.out.println("These were all the elements asked for in the test:");

System.out.println("----------------------------------------------------------------------------
--------------------");
        for(int i = 0; i < answers[0].length; i++) {
            if(answers[0][i] != null) {
                System.out.println(answers[0][i]);
            }
        }
    } else {
        Main.printLine(false);
        System.out.println("These were all the elements you got wrong on the test:");

System.out.println("----------------------------------------------------------------------------
--------------------");
        for(int i = 0; i < answers[1].length; i++) {
            if(answers[1][i] != null) {
                System.out.println(answers[1][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:");

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

    if(Main.intSelected >= 1 && Main.intSelected <= 118) {
        new Element(Main.intSelected).viewElement();
        userDone(option);
    } else if(Main.selected.equals("-1")) {
        userChooseDisplay(false);
    } else if(Main.selected.equals("0")) {
        Main.userHome(false);
    } else {
        displayElements(option, true);
    }
}
```

Explanation

There is a 2D array named answers that has 2 rows and a “number” amount of columns, and number is set to whatever number of questions the user wants. The first row has all the right answers on the test, and the second row has all the right answers that the user got wrong on the test.

The first method that uses answers is the test() which initializes it to what I previously mentioned, and the first row is used to check to see if the new element is in it or not to make sure that there are no duplicate answers in the test.

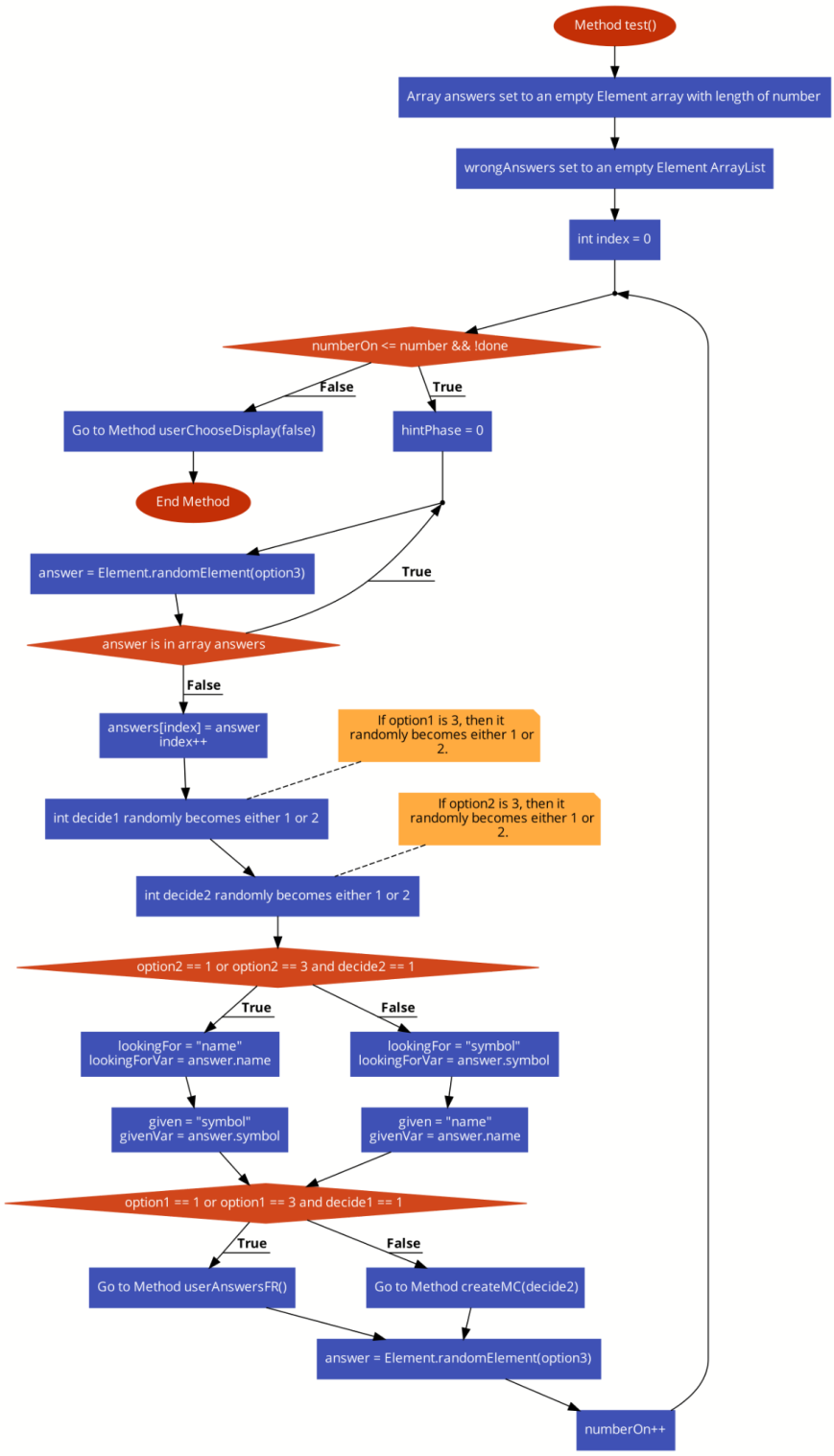
The second and third methods that use answers are userAnswersFR() and userAnswersMC(), and they both just check to see if the user got the answer right. If the user got the answer right, then nothing happens, and the value stays as null. However, if the user got the answer wrong, the second row of answers with the index (which is the next index) will get the value of the first column, which is the right answer.

The last method displayElements() simply traverses the array (uses the first row if the user wants to display all of the answers, and uses the second row if the user wants to see the wrong ones), and displays all the elements, and for the second row, it makes sure that it does not display null.

Debugging and Justification

At first, I had two parallel arrays that would do this job, but then I realized that it would be much better to just have one 2D array since it does the same thing anyway and take up less space.

Flowchart


Method test()
Array answers set to an empty Element array with length of number
wrongAnswers set to an empty Element ArrayList
int index = 0
numberOn <= number && !done
False
True
Go to Method userChooseDisplay(false)
hintPhase = 0
End Method
answer = Element.randomElement(option3)
True
answer is in array answers
False
answers[index] = answer
index++
If option1 is 3, then it randomly becomes either 1 or 2.
int decide1 randomly becomes either 1 or 2
If option2 is 3, then it randomly becomes either 1 or 2.
int decide2 randomly becomes either 1 or 2
option2 == 1 or option2 == 3 and decide2 == 1
True
False
lookingFor = "name"
lookingForVar = answer.name
lookingFor = "symbol"
lookingForVar = answer.symbol
given = "symbol"
givenVar = answer.symbol
given = "name"
givenVar = answer.name
option1 == 1 or option1 == 3 and decide1 == 1
True
False
Go to Method userAnswersFR()
Go to Method createMC(decide2)
answer = Element.randomElement(option3)
numberOn++

5. Nested Loops - Search Class

Code

```
/**
* Third phase of searching that gets the closest elements to the search query.
*
* @param String textInput - user input.
*/
private static void search(String textInput) {
   currentArray = Element.names;
   searchArray = new ArrayList<String>();

   if(option == 2) currentArray = Element.symbols;

   // Check for an exact match
   for(int i = 0; i < 118; i++) {
       if(currentArray[i].equalsIgnoreCase(textInput)) {
           searchArray.add(Element.elements[i]);
           break;
       }
   }

   // Check for the first two letters
   if(textInput.length() >= 2) {
       for(int i = 0; i < 118; i++) {
           if(currentArray[i].toUpperCase().startsWith(textInput.toUpperCase().substring(0,
2)) && !searchArray.contains(Element.elements[i])) {
               searchArray.add(Element.elements[i]);
           }
       }
   }

   // Check for the first letter
   for(int i = 0; i < 118; i++) {
       if(currentArray[i].toUpperCase().startsWith(textInput.toUpperCase().substring(0, 1))
&& !searchArray.contains(Element.elements[i])) {
           searchArray.add(Element.elements[i]);
       }
   }

   // Check for two letters
```

```
    if(searchArray.size() < 10) {
        if(textInput.length() > 2) {
            for(int i = 0; i < 118; i++) {
                for(int j = 0; j < (textInput.length() < currentArray[i].length() ?
textInput.length() - 2 : currentArray[i].length() - 2); j++) {
                    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);
}
```

Explanation

The user is asked how they would like to search the elements, and then the user searches for whatever they want, as long as the input only consists of letters. Then this method comes after, finding the best search matches via the user input.

The ArrayList searchArray contains all the best matches and the array currentArray will be searched using the user's input. The first part checks for an exact match in currentArray, and if a match is found, it will add the item of the array elements with the index of the match to searchArray. The second part checks for a match of the first two letters, but first, it ensures that the user input's length is greater than or equal to 2. The third part does the exact same thing as the second part, but instead of the first two letters, it only checks the first letter of the user input. The fourth part checks for any matches of two letters of the user input, it needs a nested for loop. The first for loop iterates through all 118 elements, using a second for loop that goes through until j equals either textInput's length minus 2 or currentArray with index i's length minus 2. To prevent an out-of-bounds exception error, I have to make sure that it stops 2 before the shortest length.

Debugging and Justification

The method has multiple for loops, and while it makes things less efficient, it allows the best matches to come up first. I originally had it all under one for loop like this:

```
for(int i = 0; i < 118; i++) {
   if(currentArray[i].equalsIgnoreCase(textInput)) {
       searchArray.add(Element.elements[i]);
   }

   if(textInput.length() >= 2 &&
currentArray[i].toUpperCase().startsWith(textInput.toUpperCase().substring(0, 2)) &&
!searchArray.contains(Element.elements[i])) {
       searchArray.add(Element.elements[i]);
   }

   if(textInput.length() >= 2 &&
currentArray[i].toUpperCase().startsWith(textInput.toUpperCase().substring(0, 1)) &&
!searchArray.contains(Element.elements[i])) {
       searchArray.add(Element.elements[i]);
   }
}
```

But the problem was that if I searched “helium” when searching by name, as shown below:

```
Type and enter whatever you want to search for, type and enter -1 to go back, or type and enter 0 to go home:
--------------------------------------------------------------------------------
helium
```

The program will display:

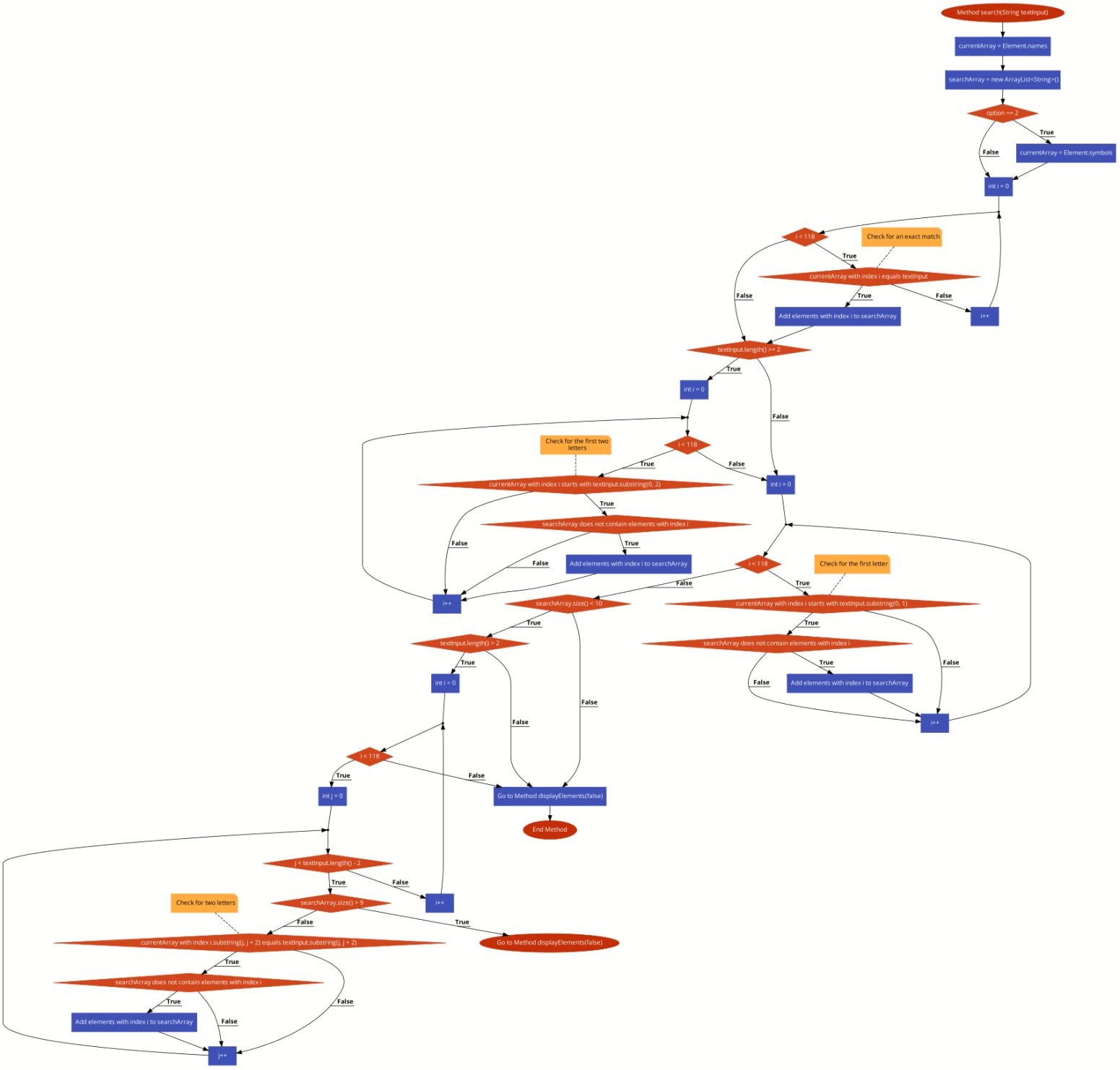
```
--------------------------------------------------------------------------------
1 - Hydrogen (H)
2 - Helium (He)
67 - Holmium (Ho)
72 - Hafnium (Hf)
108 - Hassium (Hs)

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:
```

The best match is obviously helium since it’s an exact match, but because hydrogen’s index is 0 and helium’s index is 1, with one for loop, hydrogen would match one of the conditional statements first, so it would be added to searchArray first.

However, with multiple for loops, if I searched up “helium” when searching by name, the program will display this:

In my opinion, it is better to lose some efficiency (even though the difference is not even noticeable with arrays this small) to get better matches.

Flowchart


Method search(String textInput)
currentArray = Element.names
searchArray = new ArrayList<String>()
option == 2
True
False
currentArray = Element.symbols
int i = 0
i < 118
Check for an exact match
True
currentArray with index i equals textInput
True
False
False
Add elements with index i to searchArray
i++
textInput.length() >= 2
True
int i = 0
False
Check for the first two letters
i < 118
True
False
currentArray with index i starts with textInput.substring(0, 2)
int i = 0
True
searchArray does not contain elements with index i
True
False
False
Add elements with index i to searchArray
i < 118
Check for the first letter
False
i++
searchArray.size() < 10
True
currentArray with index i starts with textInput.substring(0, 1)
True
textInput.length() > 2
True
searchArray does not contain elements with index i
True
False
int i = 0
True
False
Add elements with index i to searchArray
False
i++
False
i < 118
True
False
int j = 0
Go to Method displayElements(false)
End Method
j < textInput.length() - 2
True
False
Check for two letters
searchArray.size() > 9
i++
False
True
currentArray with index i.substring(j, j + 2) equals textInput.substring(j, j + 2)
Go to Method displayElements(false)
True
searchArray does not contain elements with index i
True
False
Add elements with index i to searchArray
False
j++

Criterion E: Evaluation

Evaluation of the product:

After showing the product to my client, we agreed on this:

1. View all elements in the periodic table in ascending element order.
    - ✅The client was satisfied that she was able to view all the elements but she thought that having to scroll up to see all the elements was quite annoying.
2. Choose an element of the periodic table and view its attributes.
    - ✅The client was fully pleased with this and had no issues, but said that maybe extra information would have been helpful.
3. Search for elements via names or symbols and find the best matches.
    - ✅The client was happy with this, but she wished that users could search for both names and symbols, but she did not mind.
4. Allow students to test their element memorization skills via free response and multiple choice.
    - ✅The client was extremely pleased with this and she said that this was perfect for her students.
5. Allow students to test their element memorization skills by answering with names or with symbols.
    - ✅The client thought that this was added perfectly because students could test their knowledge both ways, allowing them to truly memorize the elements.
6. Allow users to choose their own elements that they want to test.
    - ❌It would have been helpful if the user could choose elements to use on the test, but it would have been extremely annoying for the student if they were choosing more than 5 elements. The client did not mind because all of the elements that she tested her students were the first 20 elements.
7. Allow users to receive hints about an element during the test to help them.
    - ✅The client was pleased with this but she said that it wasn't that helpful, but happy that it was there.
8. The program can be viewed on all devices, including phones and computers.
    - ❌Unfortunately, Coding Rooms does not have a mobile-friendly version. The program can be used on any computer that has a browser, internet connection, and an account. This is a limitation of Coding Rooms as the website is not well supported on mobile, but the client did not mind because she believed that most students would study on their Chromebooks.

Recommendations for Further Development:

- The client said that adding additional information to an element could have been much more helpful like being able to see when an element was discovered, who discovered it, an element's electronic configuration, electronegativity, electron affinity, and radius.
- The client said that for searching, while most people would search for only names or symbols, some might want to search for both, or even for other things like groups and types.
- The client talked about how when viewing all the elements, it takes a long time, so it would be annoying to scroll up. There are other problems relating to the console, so using a real interface would have been better.

Appendix

Main Class Code

```
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 Code

```
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"};

    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"};

    /**
     * Constructor for creating a new Element object based on the lists.
     *
     * @param int option - the element number of the Element object.
     */
    public Element(int option) {
        option--;
        number = numbers[option];
        name = names[option];
        symbol = symbols[option];
        phase = phases[option];
        type = types[option];
        period = periods[option];
        group = groups[option];
        weight = weights[option];
        density = densities[option];
        meltingPoint = meltingPoints[option];
        boilingPoint = boilingPoints[option];
    }

    /**
     * Initializes the elements array (the main array that contains element names, numbers,
and symbols).
     */
    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.     /**
94.      * Displays all information about an element using the instance variables assigned to each
    Element object.
95.      *
96.      * @param Element other - the element that is being compared.
97.      *
98.      * @return boolean whether or not the other element is the same as this element.
99.      */
100.      public boolean equals(Element other) {
101.          return this.number == other.number;
102.      }
103.
104.      /**
105.       * Returns a string representation of the element.
```

```
106.       *
107.       * @return String representation of the element.
108.       */
109.      public String toString() {
110.          return this.number + " - " + this.name + " (" + this.symbol + ")";
111.      }
112.
113.      /**
114.       * Creates an Element object that has a random index/atomic (element) depending on the
   user's input.
115.       *
116.       * @param int option - Range of element numbers that can be randomly chosen.
117.              1 - Random Element from 1 to 30.
118.              2 - Random Element from 31 to 60.
119.              3 - Random Element from 61 to 90.
120.              4 - Random Element from 91 to 118.
121.              5 - Random Element with any atomic number.
122.       *
123.       * @return Element object that has a random index/atomic (element) number depending on
   the parameter option.
124.       */
125.      public static Element randomElement(int option) {
126.          int randomIndex = 1;
127.          if(option == 1) {
128.              randomIndex = (int) (Math.random() * 30) + 1;
129.          } else if(option == 2) {
130.              randomIndex = (int) (Math.random() * 30) + 31;
131.          } else if(option == 3) {
132.              randomIndex = (int) (Math.random() * 30) + 61;
133.          } else if(option == 4) {
134.              randomIndex = (int) (Math.random() * 28) + 91;
135.          } else if(option == 5) {
136.              randomIndex = (int) (Math.random() * 118) + 1;
137.          }
138.
139.          return new Element(randomIndex);
140.      }
141.
142.  }
```

View Class Code

```
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 Code

```
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) {
```

```
111.                         displayElements(false);
112.                         return;
113.                     }
114.                     if(currentArray[i].substring(j, j +
    2).equalsIgnoreCase(textInput.substring(j, j + 2)) &&
    !searchArray.contains(Element.elements[i])) {
115.                         searchArray.add(Element.elements[i]);
116.                     }
117.                 }
118.             }
119.         }
120.     }
121.
122.     displayElements(false);
123.   }
124.
125.     /**
126.      * Fourth phase of searching that displays the closest elements to the search query
    and allows users to view an element.
127.      *
128.      * @param boolean tryAgain - whether the method was called again due to an invalid
    user input.
129.      */
130.     private static void displayElements(boolean tryAgain) {
131.         Main.printLine(true);
132.         if(searchArray.size() > 0) {
133.             for(int i = 0; i < searchArray.size(); i++) {
134.                 System.out.println(searchArray.get(i));
135.             }
136.             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:");
137.         } else {
138.             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:");
139.         }
140.
141.         Main.selected = Main.scan.nextLine();
142.         Main.intSelected = Main.isNumeric(Main.selected) ? Integer.parseInt(Main.selected)
    : -10;
143.
```

```
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 Code

```
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.  }
```