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.

Parallel Arrays

Element Class

First I had to get data for all the elements of the periodic table. I used code.org's periodic table data column[1] 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.

```
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",
```

[1]See Bibliography, Source 1

```
"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",
```

```java
"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"};
```

My constructor used the parallel arrays here:

```
/**
* 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];
}
```

Error Handling

Code

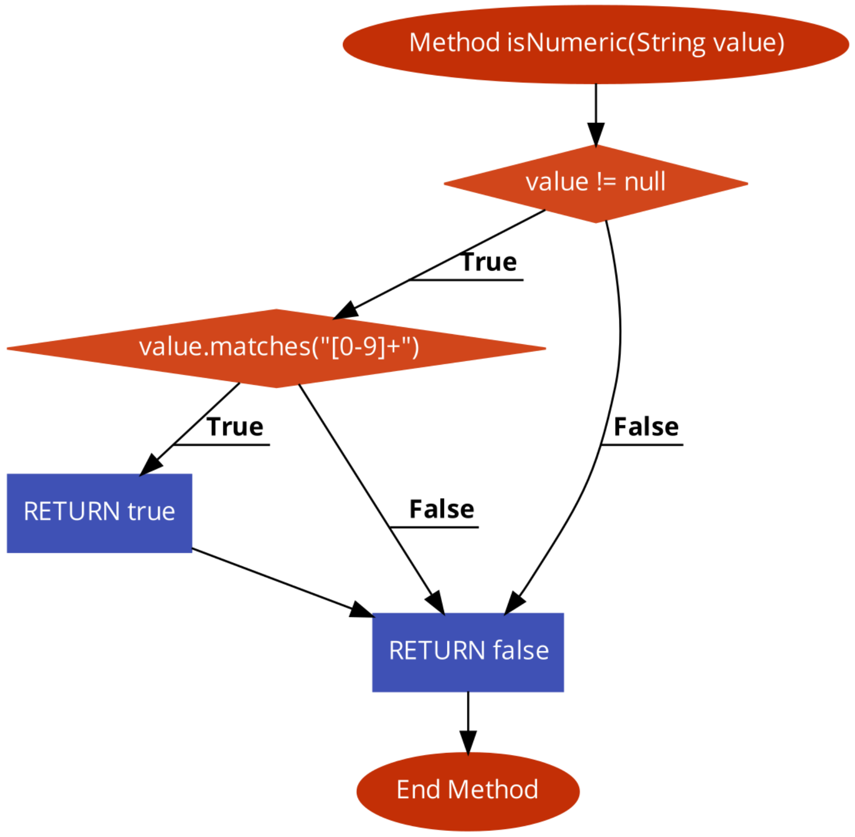
```
/**
* Checks if a given string can be converted to a number.
*
* Credit to MV, Thanoshan. "Java String to Int – How to Convert a String to an Integer."
FreeCodeCamp.org, FreeCodeCamp.org, 23 Nov. 2020,
https://www.freecodecamp.org/news/java-string-to-int-how-to-convert-a-string-to-an-integer/.
&
* @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()[2] method.

[2]See Bibliography, Source 2

Flowchart


Method isNumeric(String value)
value != null
True
value.matches("[0-9]+")
True
False
False
RETURN true
RETURN false
End Method

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

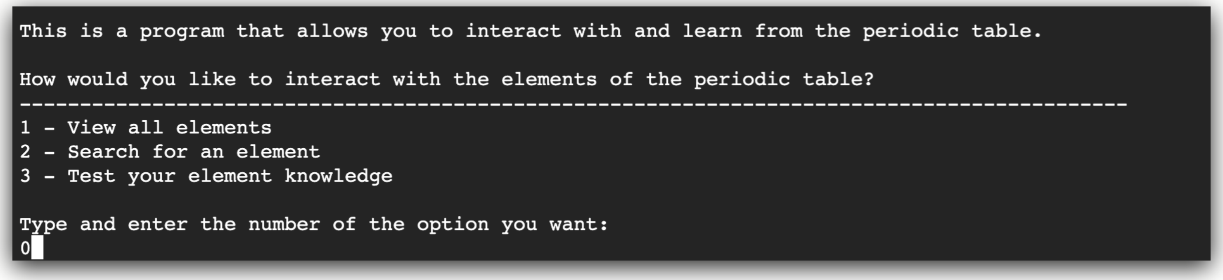
```
/**
* Asks the user to select one of the five options in the home screen, then calls the
corresponding class and its first phase.
*
* @param boolean tryAgain - whether the method was called again due to an invalid user input
*/
public static void userHome(boolean tryAgain) {
   printLine(false);
   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

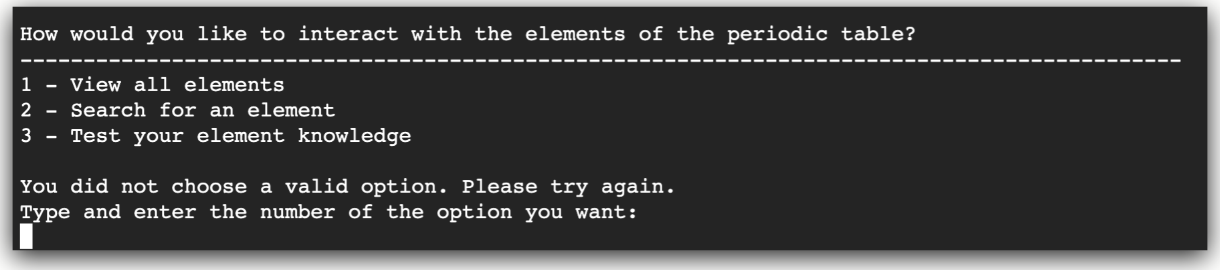
Whenever this method is called from another method, tryAgain is set to false, but if the method is called from this method, tryAgain is set to true. The method will continue calling itself if the base case is not met, as long as the user inputs something that is not accepted.

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



```
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 - View all elements
2 - Search for an element
3 - Test your element knowledge

Type and enter the number of the option you want:
0
```

Then, the console will repeat the method:

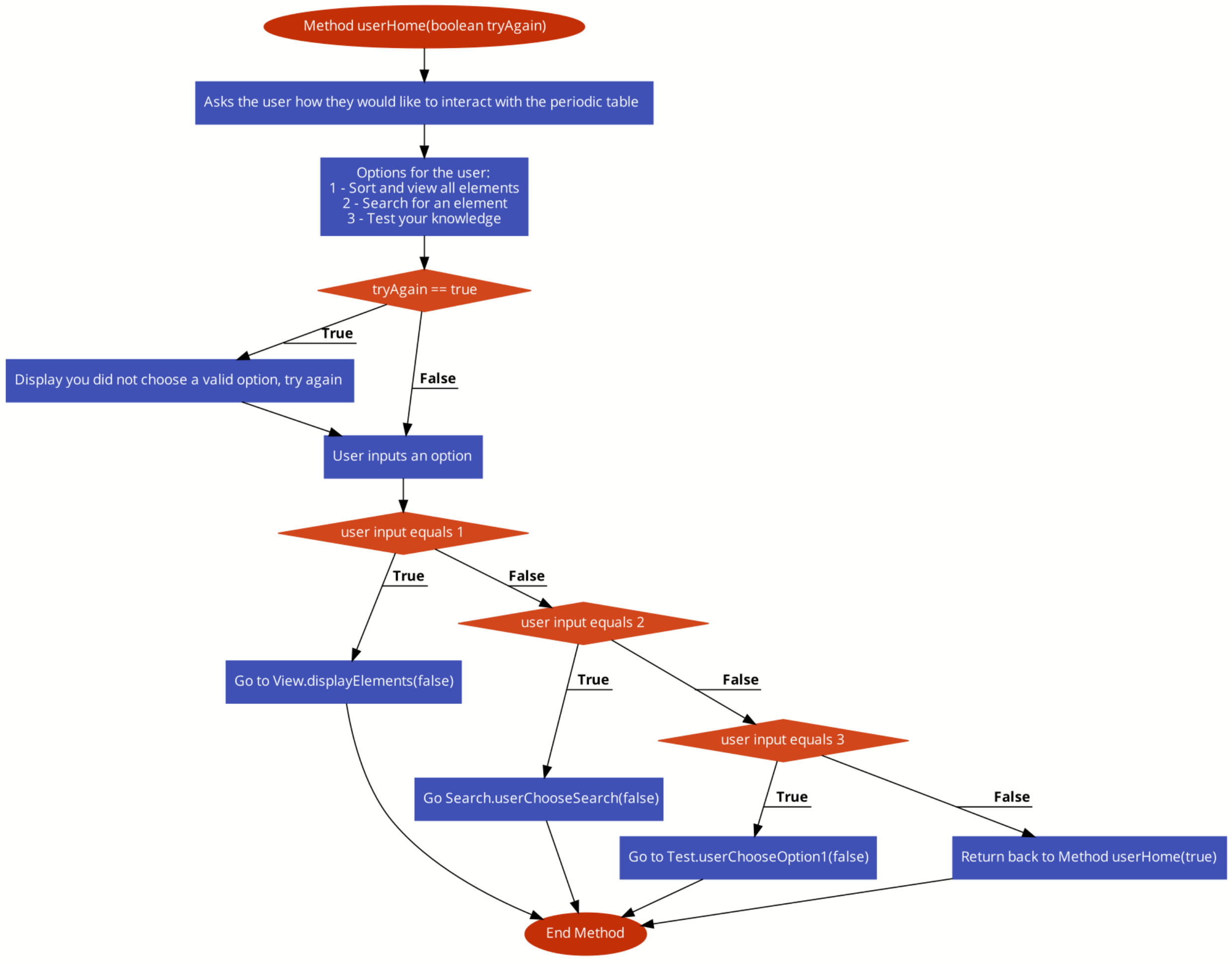


```
How would you like to interact with the elements of the periodic table?
-----------------------------------------------------------------------------------------------
1 - View all elements
2 - Search for an element
3 - Test your element knowledge

You did not choose a valid option. Please try again.
Type and enter the number of the option you want:
```

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

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();
```

```java
    } 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.

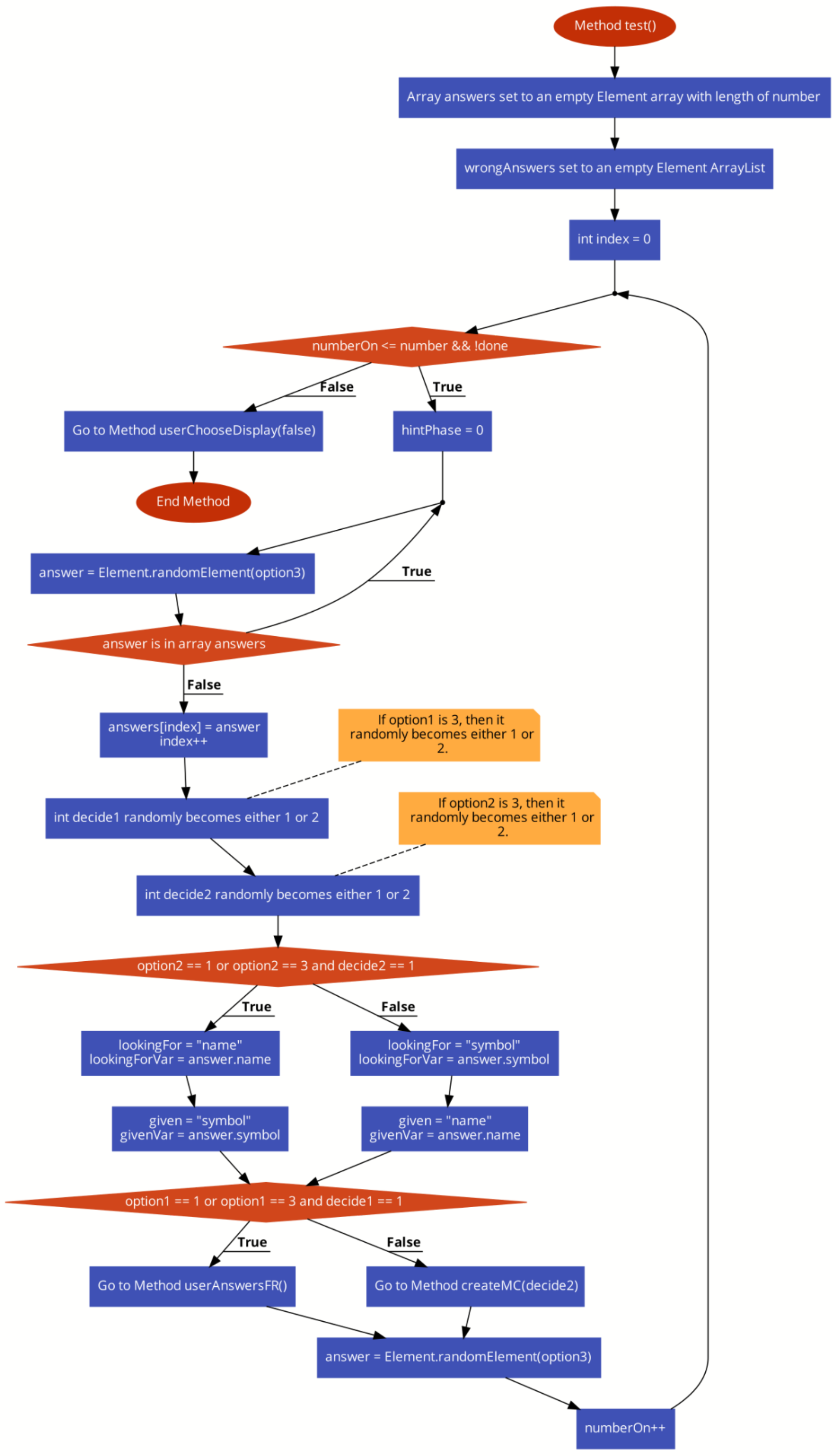
The second and third methods that use answers are userAnswersFR() and userAnswersMC(), and they both just check to see if the user got the answer right. If the user got the answer right, then nothing happens, and the value stays as null. However, if the user got the answer wrong, the second row of answers 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++

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.

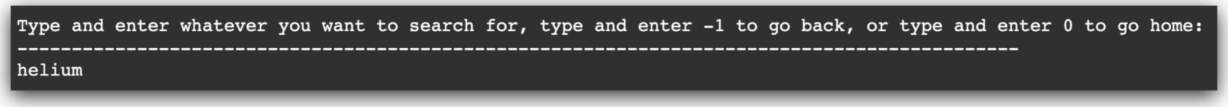
The ArrayList searchArray contains all the best matches and the array currentArray will be searched using the user's input. The first part checks for an exact match in currentArray, and if a match is found, it will add the item of the array elements with the index of the match to searchArray. The second 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:

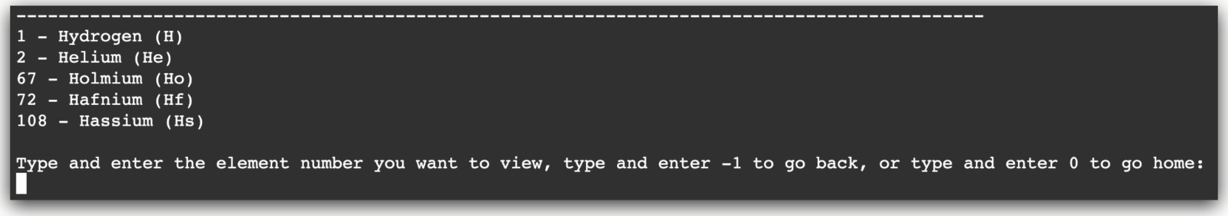
```
for(int i = 0; i < 118; i++) {
   if(currentArray[i].equalsIgnoreCase(textInput)) {
       searchArray.add(Element.elements[i]);
   }

   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
```

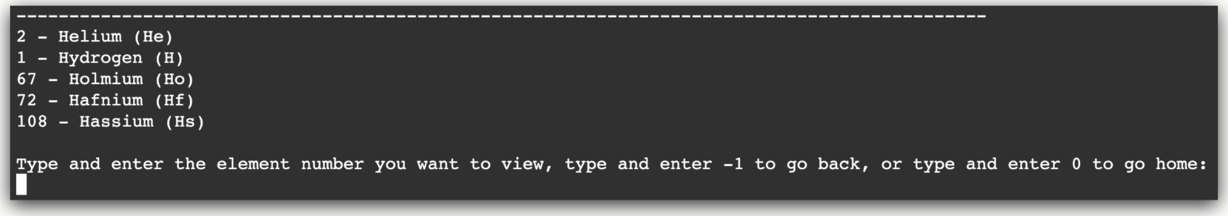
The program will display:



```
---------------------------------------------------------------------------------------------
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.

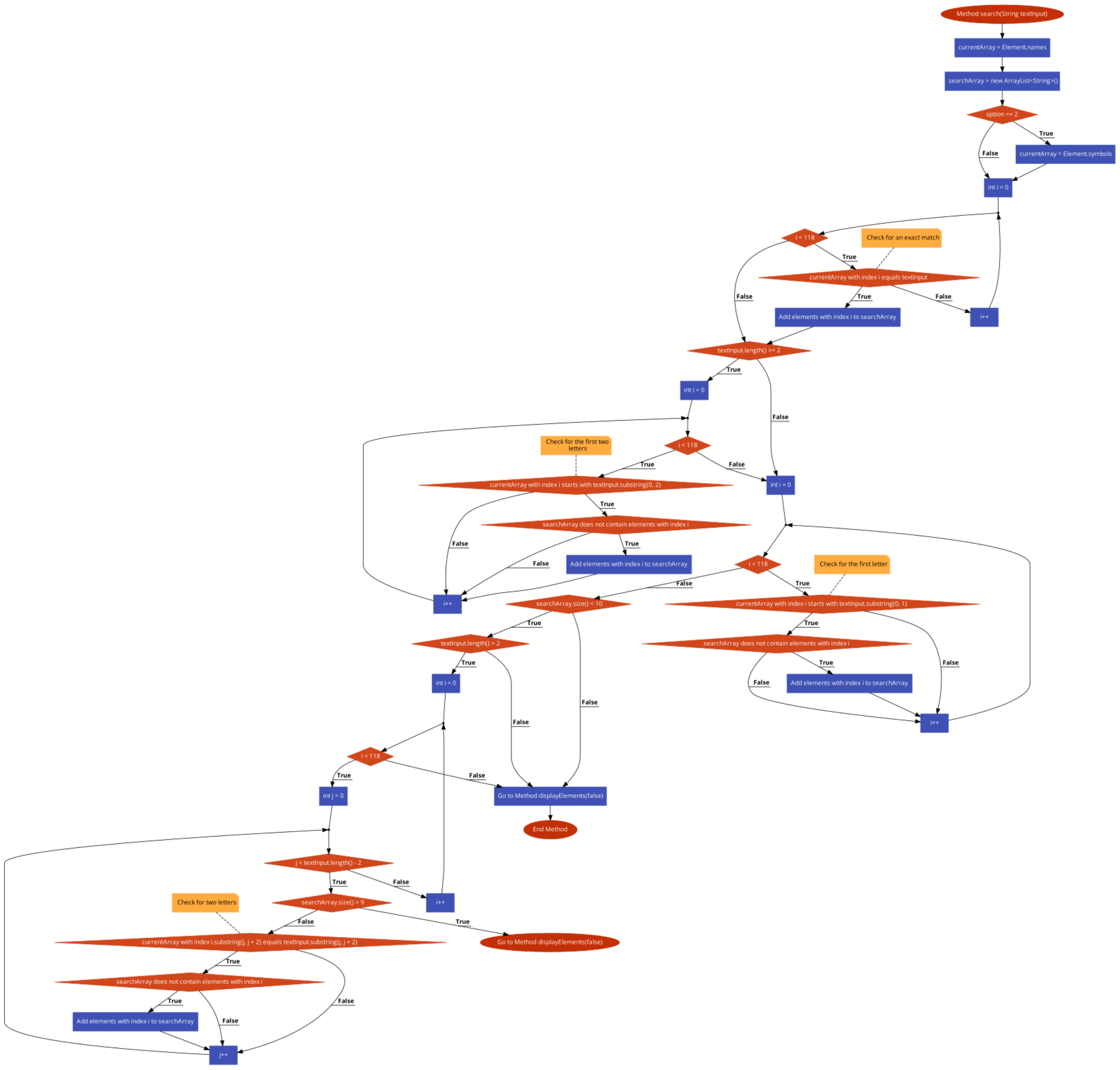
However, with multiple for loops, if I searched up “helium” when searching by name, the program will display this:



```
-------------------------------------------------------------------------------------------
2 - Helium (He)
1 - Hydrogen (H)
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:
```

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
False
True
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
False
True
False
Add elements with index i to searchArray
i < 118
Check for the first letter
False
True
i++
searchArray.size() < 10
currentArray with index i starts with textInput.substring(0, 1)
True
True
textInput.length() > 2
searchArray does not contain elements with index i
True
True
False
int i = 0
False
Add elements with index i to searchArray
False
False
i++
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
False
Add elements with index i to searchArray
j++

Word Count: 993

Bibliography

"Code.Org. Learn Computer Science - Code.Org, https://studio.code.org/data_docs/periodic-table-elements/ Accessed 20 Feb. 2023.

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/.