**1.2.1 Abstraction & Loops**

**Introduction**

In this assignment we will build a playable piano and get an introduction to more advanced concepts such as loops, Arrays, and abstract classes.

Concepts covered are:

* Logical operators
* Abstraction
* Array
* while loops
* Keywords: **import, true, false, &&, ||, , !, ^, while**

**Materials**

* Computer with Greenfoot
* Source files - piano.zip

**Activity**

**Part I: Logical Operations & Abstraction**

1. Download piano.zip, create a new subfolder called piano in your Greenfoot project folder, extract the files from the zip into your folder. Open the scenario (update the Java if needed). Make sure to compile if needed.
2. Explore the scenario - there isn’t much to it. Our goal is to create piano keys that appear to be pressed when you click them, make sounds that correspond to a piano, and do this in such a way that we don’t have to **hard-code** each key separately - in other words we need to **abstract** our keys. When we instantiate our keys, we can feed the constructor the key we will press and the name of the sound file it will play, that way each key will be slightly different.

|  |
| --- |
| * Open the Key class and create two new fields: key (a String) and sound (a String). * Create a constructor that takes two String parameters: keyName and soundFile. * Inside the constructor set the keyName to key and soundFile to sound. |

1. We want the key to appear to be pressed when clicked. In order to be efficient, we need to make sure our code is NOT loading an image on top of the same image. For example, if we have an image of a piano key being NOT being pressed we should not be loading the same image.

To do this we need to use logical operators. Logical operators allow us to evaluate multiple boolean conditions at the same time. The logical operators are:

|  |  |  |  |
| --- | --- | --- | --- |
| **Not** | **And** | **Or** | **Xor** |
| **!** | **&&** | **||** | **^** |

|  |  |  |  |
| --- | --- | --- | --- |
| Truth Table | | | |
| **First statement** | **Logical** | **Second statement** | **Result** |
| true | AND | true | true |
| true | AND | false | false |
| false | AND | true | false |
| false | AND | false | false |
| true | OR | true | true |
| true | OR | false | true |
| false | OR | true | true |
| false | OR | false | false |
| true | XOR | true | false |
| true | XOR | false | true |
| false | XOR | true | true |
| false | XOR | false | false |

The NOT operator simply changes false to true and true to false. For example, if I want to see if a variable is not equal to the number 5 I would write:

var != 5;

If I want to make the statement variable, which is set to true, into false I might write something like this:

!statement;

|  |
| --- |
| * Declare a new field (instance variable) called isDown. This will hold **true** if the key is down or **false** if it is not. * In the act() method write two if-statements. Here’s the pseudocode:   if (isDown is **false** and key is pressed)  setImage to “white-key-down.png” and set isDown to **true**  if (isDown is **true** and the key is NOT pressed)  setImage to “white-key.png” and set isDown to **false** |

1. Now to make it play sounds.

|  |
| --- |
| * Create a new method in the Key class called play(). * Have it play sound. We will call it whenever we want play the sound file. Look at the Greenfoot APIs if you need help finding the right methods. * Call this method when the key is pressed. |

1. Notice that the piano keys do not actually play anything yet because they are an abstraction. They use variables to map them to the keyboard and to map sounds.

|  |
| --- |
| * To see how this works, instantiate a Key object by right-clicking it in the class diagram. Give it “g” for keyName and “3a.wav” for soundFile. * Place the key and press ***Run***. Press the key. |

**Part II: While Loops & Arrays**

1. Instead of instantiating and placing every key ourselves we are going to use a **while** **loop** and an **array** to have the computer do it. The first step it to create the method that will instantiate the keys and be looped.

|  |
| --- |
| * Open the source code for the Piano class. * Create a new method called makeKeys() that will create and place keys. We will need to change the y-coordinate each time so that the top of the piano key lines up with the top of the piano itself. Hint: The key image is exactly 280 pixels high x 63 pixels wide. * The method should be somewhat abstract at this point! * The makeKeys() method should be called inside the constructor method of the Piano class. |

1. We will now put the code to build the piano keys into a **while** loop. A loop will repeat the code placed into it while the conditional is **true**. Loop syntax looks like this:

declare a variable to control the loop

**while** ( conditional )

{

statement;

statement;

…

increment control variable

}

It is very common for programmers to use a variable to control the number of **iterations** of a loop (stepper).

|  |
| --- |
| * Create a while loop inside the makeKeys() method.   + Create an integer to use as a control variable, call it i and set it to 0.   + Have the loop iterate 12 times.   + Copy the other makeKeys() code into the loop.   + After the code make sure you increment i by one each iteration.   + Add code to the loop to make certain the keys are placed on the piano next to each other but lined up at the top. Hint: use the incrementing variable to help adjust the x-coordinate each time a key is placed.   + Adjust the code so that the keys are offset correctly. Greenfoot uses the center of an object for placement. Hint: you may need a fixed offset of 54 pixels. * Optional challenge: Hard-coding numbers into a method is generally not the best way to solve a problem in programming - what if we want to change our keys to a higher-quality image later? Try making the loop place the keys using built-in methods such as getHeight() or getWidth() and without numbers (except the control variable i will use numbers of course). |

1. The loop also needs to give the new keys different values for keyName and soundFile when they are created so that they don’t all react to the same keyboard key and play the same note. To do this we will use an object called an **array**. In order to use Arrays in the program we need to import them. Before the Piano class opens we need to add a statement to import Arrays.

|  |
| --- |
| * Add this statement to the top of the Piano before the class opens: import java.util.Arrays; |

1. The syntax to declare an Array looks like this:

DataType[elements] arrayName;

Assigning an Array looks like this:

arrayName = {element1, element2, … elementn}

But usually we declare AND assign arrays at the same time. This is what we will do in this project:

DataType[] arrayName = {element1, element2, … elementn}

Each element in the Array object has a position number (called an **index**) that is indicated like this:

arrayName[1] (for element1)

|  |
| --- |
| * Declare and assign two private fields:   + An Array called whiteKeys[] to hold key names: “a”, “s”, “d”, “f”,“g”,“h”, “j”,“k”,“l”,“;”,“‘“,“\\”   + An Array called whiteNotes[] to hold the names of sound files: “3c” ,“3d”,“3e”,“3f”,“3g”,“3a”,“3b”,“4c”,“4d”,“4e”,“4f”, “4g” * Modify your loop so that it uses the index of the Arrays you just created to construct new keys. Use the control variable i as an index to step through your Arrays. Hint: you will need to add “.wav” to each whiteNotes[] item when using it as an argument. * Replace the 12 in the conditional with whiteKeys.length. This will return the number of items in whiteKeys[]. Each Array has a public length variable! * Now change the code so that the white keys play sounds that are one octave lower (use 2c instead of 3c for the first key and move up from there). |

1. Now to add the black keys:

|  |
| --- |
| * Use abstraction to modify the Key class so that it can show either white keys or black keys. Hint: you will need to use abstraction like we did to generate white keys. Also, the image files are hard-coded and this is not good practice...abstract it! * Add two more Arrays to the Piano class that store sounds and keyboard keys for the black keys. * Add another loop in the makeKeys() method of the Piano class that create and place the black keys. This is tricky because black keys are not as evenly spaced as white keys. Here is a link that has pictures of how piano keys are placed:   <https://2012books.lardbucket.org/books/music-theory/s07-03-the-keyboard-as-a-visual-tool.html> |

**Conclusion**

Answer the following questions:

1. Test your logical operations:
   1. Practice your logic: <http://codingbat.com/prob/p187868>
   2. <http://codingbat.com/prob/p181646>
   3. <http://codingbat.com/prob/p140449>
   4. <http://codingbat.com/prob/p182873>