Create – Applications From Ideas
Written Response Submission Template

Please see Assessment Overview and Performance Task Directions for Student for the task directions and recommended word counts.

Program Purpose and Development

2a)

This program was made in java. The purpose of the program is to provide users with a calculator that allows them to convert numbers between different base systems. These base systems can range from binary to hexadecimal. The video illustrates the use of the program as well as how it functions. It shows that users may input values for the number, current base, and desired base. After this, the video shows that users may press the button to obtain the result. This video also demonstrates the built-in error handling mechanism that displays an error message if invalid inputs are entered.

2b)

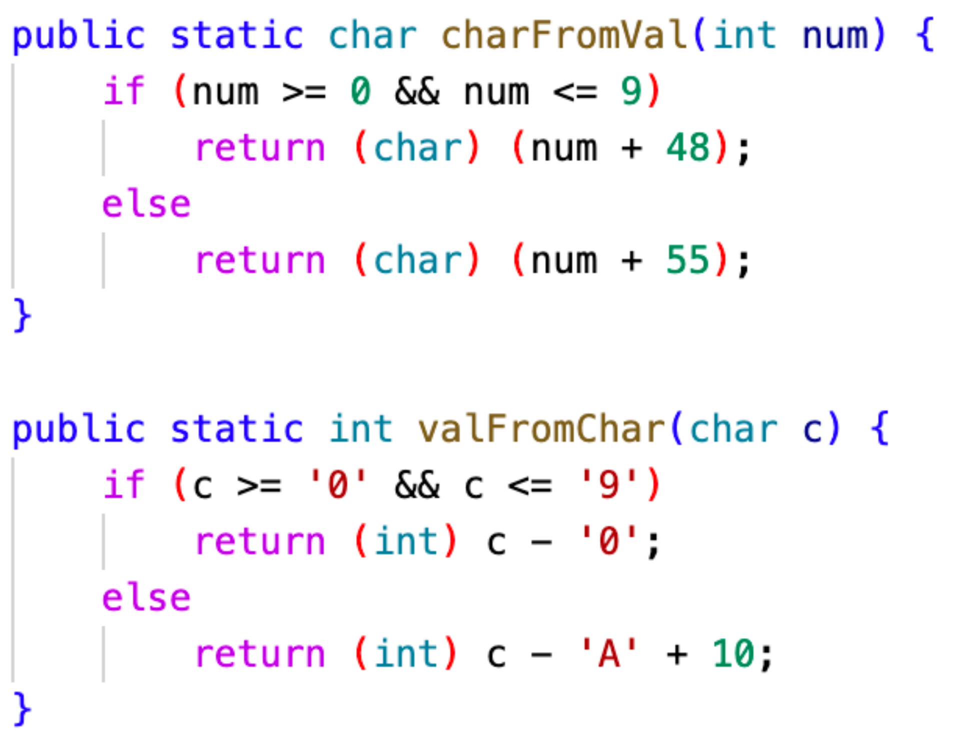
I worked independently throughout the entire development process. It began with brainstorming, in which I realized that most popular calculators would at maximum only allow conversions from binary, decimal, and hexadecimal bases. From here, I focused on the backend of my program that would actually define the methods that would allow for the conversion. The process begain with developing algorithms that could handle conversions between a certain base and decimal and then attempting to generalize this process for the conversion of all bases to decimal. I repeated this process again in order to develop a generalized algorithm for the conversion of a number from decimal to another base. I then began testing my program through a console-based test program that I developed. I tested by providing sample inputs and then evaluating the console outputs. After I had finished the conversion methods, I focused on building the GUI by using the javafx library. I worked by initially setting up the background and placing test labels, text fields, and buttons in a manner while constantly adjusting the locations. I finally implemented the conversion methods in my handler functions to create the final product.

2c)

```java
public static String baseConvert(String input, int base1, int base2) {
    int temp = toDecimal(input, base1);
    String result = fromDecimal(temp, base2);

    return result;
}

public static String fromDecimal(int input, int base) {
    String revResult = "";

    while (input > 0) {
        revResult += charFromVal(input % base);
        input /= base;
    }

    String result = "";

    for (int i = revResult.length() - 1; i >= 0; i--) {
        result += revResult.substring(i, i + 1);
    }

    return result;
}


public static int toDecimal(String input, int base) {
    int power = 1;
    int num = 0;

    for (int i = input.length() - 1; i >= 0; i--) {
        num += valFromChar(input.charAt(i)) * power;
        power *= base;
    }

    return num;
}
```

The main algorithm that is depicted is the baseConvert() method. This method is essentially a wrapper method that first calls the toDecimal() and then the fromDecimal() methods. The toDecimal() and fromDecimal() methods are sub-algorithms to the baseConvert() that use decimal numbers as an intermediate The toDecimal() algorithm operates by evaluating the value of the character regarding its placement in the string. It then multiplies it by the appropriate power of the base. This value then gets added to the integer num and the overall process repeats for all values in the String. The use of Strings is necessary because any base higher than decimal will use upper-case letters to represent higher values. With this decimal value, the parent algorithm baseConvert() then calls the fromDecimal() algorithm in order to convert the obtained decimal value into the desired base. It does so by adding the character equivalent of the remainder that occurs when the decimal value is divided by the desired base to the respective position in the String. However, after this is finished, the result will be reversed, so the String has to be traversed and elements will be added to a new resultant String in a reversed order.

2d)



```
public static char charFromVal(int num) {
    if (num >= 0 && num <= 9)
        return (char) (num + 48);
    else
        return (char) (num + 55);
}

public static int valFromChar(char c) {
    if (c >= '0' && c <= '9')
        return (int) c - '0';
    else
        return (int) c - 'A' + 10;
}
```

This abstractation consists of two methods that I developed in order to manage the complexity of value conversions from integer values to char values and vice versa. These methods are useful because they can be called from anywhere in the program and help prevent the redundant retyping of code, making it more manageable and easier to read. These methods help to simplify the complexity of the code by preventing the need to specify the values of certain characters (such as the letters A through F) each time they appear. By using these methods, the code becomes much more abstracted and thus robust and client-friendly.