

COMP3021 (Programming Languages Paradigm)

Project Report

Andrew Thenedi
Chan Man Ching Bryan
JooHwan-Lee

Table of Content

Contribution Statement	Page 2
Introduction	Page 3
Requirement and Problem Analysis	Page 3
Design	Page 6
Implementation and Testing	Page 7
Sample Programs	Page 10
Conclusion	Page 11
Appendix	Page 12

Contribution Statement

- Andrew Thenedi [18079969D]
 - Tasks
 - Task 3
 - Task 4
 - Task 5
 - Task 6
 - Task 7
 - Report Sections
 - Design
 - Implementation and Testing
 - Sample Programs (multiple inputs of unique shapes)
 - Demo Video
- Chan Man Ching Bryan [18035914D]
 - Tasks
 - Task 2
 - Task 3
 - Task 6
 - Task 7
 - Report Sections
 - Introduction
 - Requirements & Problem Analysis
 - Design
 - Conclusion
 - Overall organization of project report
 - Appendix
- JooHwan-Lee [18079916D]
 - Tasks
 - Task 2
 - Task 3
 - Task 6
 - Task 7
 - Report Sections
 - Introduction
 - Requirements & Problem Analysis
 - Design
 - Conclusion
 - Overall organization of project report
 - Appendix

Introduction

This project report aims to clarify and explain the design of data types, syntax, and semantics of our own programming language. Also, it discusses and explores the implementation of features and functionalities in the programming language the team has created for the project.

The project report can be mainly divided into 5 components. It first introduces the evaluation of existing programming languages and the main requirements and features of our programming language. For requirement analysis, the team aims to create a simple yet concise programming language for the user to learn and write. For problem analysis, it compares and contrasts the readability, writability, and reliability of existing programming languages, such as Java, C, and Python. After, the project report highlights the main design of our own programming language. For instance, it shows the supported data types and syntax of the language. It also summarizes the semantic by defining major scope rules and typing rules. Moreover, the report explains the implementation and the testing process by exploring the sample codes runned by our programming language. Lastly, the project report will conclude the team's finding of the project and the final version of the programming language implemented.

Please note that all figures mentioned in the project report can be referred to in the Appendix section with the corresponding numerical value.

Requirements & Problem Analysis

At requirement and problem analysis, we do compare the different program languages: Java, C, and Python with 3 criteria: readability, writability, and reliability. We evaluate them in perspective of the drawing tool, examining which features we should apply to our programming language. Then we introduce the main requirement and the feature of our own programming language that we have designed.

Readability

To begin with, the readability of a program is defined as the level of difficulty at which programs can be read and understood. The readability of a programming language is highly affected by its overall simplicity, orthogonality, and syntax design. The overall simplicity of a language is determined by its syntax and basic constructs. Languages that consist of a large number of basic constructs are more difficult to be understood, whereas languages that have a small number of basic constructs are easier to learn. Java and Python are two examples of programming languages that have simple syntax and easy to learn. According to Oracle, Java is easy to learn because of the following reasons: 1) Syntax of Java is based on C++. 2) Java has removed many complicated features such as explicit pointers. 3) Java has Automatic Garbage Collection so programmers are not required to remove unreferenced objects. Python is also a simple programming language because its syntax is concise. Since Python is a high-level and interpreted programming language, it is fairly easy to learn and read. On the other hand, C is a mid-level language, therefore its syntax will be more complicated.

In addition, the readability of a program is highly dependent on orthogonality. Orthogonality is the measurement of how primitive constructs can be combined to create data structures and control the language. An orthogonal language has fewer exceptions rules require. It leads to a higher degree of regularity in the design which makes the language easier to read. Python provides the most flexible grammar in the perspective of orthogonality out of these three programming languages. For instance, Python allows `print("Hello" + "World")` which has the equivalent meaning of `print("Hello World")`. These various combinations of grammar are not

allowed in C and Java, therefore, Python has better orthogonality. Whereas C lacks orthogonality because it has comparatively more rules and exceptions. For example, an array element can be any data type except void or a function. A member of a structure cannot be a structure of the same type. As we can see Python is the most orthogonal programming language among the three selected languages.

The syntax design can be significantly impacted by the possible meaning of the same keywords. For example, the semantics of the reserved word 'static' is dependent on the location of where it is declared. If it is used on a variable inside a function, it refers to that the variable is created at the compile time. However, if 'static' is used outside of all functions, it means that the variable is only visible in that file. Cases like this will decrease the readability of a program, as one keyword may suggest different semantics and cause confusion. On the other hand, Python and Java are not low-level programming languages, therefore it has an English-like syntax. It is fairly easier for programmers to understand and read.

Writability

Writability refers to the degree of complexity in which programs can be created for a chosen problem. It is determined by the expressivity and the readability of a program as well. Expressivity measures the ability of a language to declare an operation with limited expression. For example, C and Java have high expressivity when incrementing variables because the programmer can easily write "x++". Yet, programmers writing in Python are required to write at least "x+=1". As observed, programmers can create the same operation in C and Java more conveniently and with a shorter notation than in Python.

The readability of a language influences its writability since programmers are required to re-read the program during the process of creating it. Hence, if the language syntax is simple and orthogonal, it is easier for the users to create a program as well. For example, languages that have a lot of constructs will tend to lead to misuse features. For example, switch statements and if-else statements can be used to create similar operations, but switch statements can only evaluate string or integer. Therefore, at first, beginners might misuse these constructs and it greatly reduces the writability.

Reliability

Reliability is about reducing the chances of programs having errors. 3 criteria influence reliability, which are Type checking, Exception handling, and Aliasing. Type checking shows how that programming language is good at finding the type error during the compile or execution. As early as the language detects the error, there is less cost to fix them, and finding errors during the compilation is the most desirable situation in the perspective of Type checking. For instance, Python Idle does not show the type error in real-time but the compilation is required to detect the type error, which is decent since it detects an error during the compilation but other platforms of other languages like Visual Studio and IntelliJ are even better as they even detect type error before running the compiler. Exception handling depends on how the language allows programmers to react to diverse run-time errors, as there are more kinds of run-time errors that language can handle, the reliability of the language gets considered higher. Lastly, the Aliasing allowance of multiple variables on the same address of the memory. As more tools in language allow more variables to point to the same memory address, aliasing increases but reliability declines.

As mentioned, Python's type checking works fine. It detects the type of error during the compilation. As the next language Java, it depends on what platform you are using, IntelliJ does not detect the type error until the compilation but another Java platform, Eclipse detects type error without a compiler and warns the user. Lastly, Visual Studio for C also detects type errors before the compilation, thus, we can consider the C having better Type checking than Java and Python as the worst. Then for Exception handling, Python provides an exception handling with "try...except" and "try...finally". It also allows programmers to occur errors or exceptions manually by using "raise" if wanted. Java also allows these exception handlings using "try...catch" and "try... finally" and also allows manual exception occurrence by the user like Python. However, C does not have a specific function or module made for exception handling so we can say C has worse Exception handling than Java and Python. Lastly, as the Aliasing of these languages, Java and Python both do not allow multiple variables pointing to the same memory address. But the C has a technique, the pointer, which allows variables to point to the same specific memory address, thus, C is again the worst for this criterion. Overall, Python takes advantage of Exception handling and Aliasing, Java takes half for Type checking but the full advantage on Exception handling and Aliasing but C only takes Type checking that we may consider Java as the most reliable language out of 3.

Requirements

As the requirement of our program language, we need the sablecc version of 3.2 since our language uses it for development. Also since we use the converter to convert it and run under the environment of Java, therefore, JDK and JRE are required to be installed in the device (Windows: <https://www.guru99.com/install-java.html>, Linux: <https://www.guru99.com/how-to-install-java-on-ubuntu.html>).

Features

Our programming language aims to provide programmers a painting or drawing platform that is simple to use but no obstacle to visually express their creativity. We have sketched some essential features that might be needed to fulfill our aim. By considering all inconveniences of each GUI or painting libraries such as Tkinter of Python and java.awt of Java, requiring users to for now the precise coordinates of axis, we first considered the function of getAxis(). This getAxis() gets the coordinate of the axis by letting users click the screen for n times and return the n points axis as a list that contains n points in the format of [(a, b), (c, d)...]. It will be hard to get the exact axis by the function since you need to click just one point out of 1920x1080 pixels. But still, it will help the guesswork of getting a similar value of axis that the user wanted, then the user just can edit the axis with a small range to achieve the axis user originally desired. As the next function, we considered the group of functions that creates the standard figures. The group contains, makeCircle, makeSquare, makeRect, and makeEclipse... etc. that each function creates a circle, square, rectangle, and eclipse. They follow the common format of parameters: (position, height, width, outline, fill). The position takes the axis as its input which refers to the top-leftmost axis, and the user modulates the location of the figure by changing the input of position. Then the width and height both take positive integers as their input that controls the width and height of the figure. However, makeCircle is an exception that rather than width and height, it takes radius as the input to control its size.

The outline and fill are optional parameters that all take color as the input. Outline means the color of the figure's outline and fills in the color of the figure's surface, the outline is black

and the fill is white by default. But if users want to create their custom figure, the function `makeCustom` is used. It takes 2 parameters, number of points as positive integers since we do not know how many points are in their custom figures, and another parameter `axis` is the list of axes that has the same amount of axes with the first parameter, the number of points. Then the user might want to create a line rather than the figure, and `makeLine` is ready for usage. It takes a list that contains 2 axes as the first parameter `axis` which becomes each ending point of the line. Then the color is an optional parameter which is default black. Our language also satisfies users wanting to create the curvy line with `makeCurve` that has one more additional parameter from `makeLine`, the degree which is 1 in default. Degree determines the curviness of the line, and as the principle of creating curve lines, we use the sine graph. Assuming the Degree as d , length of the line as s which uses the distance formula between the 2 axes that got input as `axis`, `x_position` as a , and `y_position` as b which both we can get from the parameter of `axis`, formula looks like $y = d(\sin((\pi/s)x - (a * (\pi/s))) + b$. Increasing the d or degree increases the curviness of the line and if the opposite situation is desired, the user can put the degree that is lower than 1 like 0.1 which should not be lower or equal to 0.

Lastly, if the angle between the axes is not 180 degrees, so the line is not horizontal, we can first create the graph horizontally and rotate it after calculating the angle between the axes. As the last function from the sketch, there is `makeText` that allows the user to create the text on the screen which takes `centre_position`, `font`, `color`, and `size` as the parameters. `Centre_position`, the only mandatory parameter takes the single-axis as the input to determine the position of the text box. `Font`, `color`, and `size` are all optional and their default values are “Times New Roman”, 12, and black without any input. As the color system of our programming language, we accepted basic rainbow colors with black, white, pink, and sky blue for simplicity.

Design

Supported Data Types and Syntax

- The input parameters that accept only integer values are: “X_CURRENT”, “Y_CURRENT”, “X_NEXT”, “Y_NEXT”, “WIDTH”, and “HEIGHT”, while the input parameters that accept only string values are: “COLOR” and “SHAPE”. The above-mentioned input parameters are the only possible parameters for our language.
- In addition, our language provides data types of integers and strings. However, while the language lets the users input all possible integers, the language only accepts a string listed on the `grammar_draw.txt`. Such a restriction is done to avoid confusion for the compiler when compiling the input command, as it has considered all possible input parameters in its logic. For instance, if the user is using the “SHAPE” parameter, with the value of “LINE”, as input and compile it. The compiler interprets such input as drawing a line in the canvas.
- For the assignment statement, the language uses the “=” operator, between parameter and value, to assign the value to the parameter. For the end-of-statement interpretation, the language uses the “;” symbol, after value, to interpret that such a statement is finished. Afterward, there are two possible cases:
 - 1) End the program execution
 - Such an action happens if and only if there is not any statement left after “;”.
 - 2) Continue to the next statement
 - Such an action happens if and only if there exists a statement after “;”.

The Semantics of Our Language

- Scope Rules
 - The scope of the input is determined by the value of the parameter “SHAPE”. Depending on the valid value of “SHAPE”, the program expects different combinations of inputs to exist, which determine whether executing such a statement is possible or not. For instance, if the value of the parameter “SHAPE” is “Pentagon”, then the program will interpret such a statement as valid, if and only if the user assigns an appropriate value to three different parameters “COLOR”, “X_CURRENT”, and “Y_CURRENT” in the next input line.
- Typing Rules
 - For all the valid parameters, it is expected that the name for the parameters is in the upper case only. The rule is being implemented to maintain the readability and consistency for a string to be categorized as a parameter.
 - For all the valid values of string, it is expected that the name of the values is either in all upper case, all lower case, or only the first line of the word is in upper case. For instance, the value “DARK GRAY” of parameter ‘COLOR’ can be represented equally by the value “DARK Gray”, “DARK gray”, “Dark GRAY”, and “dark GRAY”.

Implementation and Testing

Set the Drawing Canvas

- In *figure 1* on page 12, once the program is executed, it will create an object of JFrame. It sets a trigger for the closing operation, to denote the end of the execution. The position of the canvas in the screen is initialized as (30,30), which implies to x equals 30 and y equals 30. The size of the canvas on the screen is initialized as the width equals 600 and height equals 600. Set the title as “Group XYZ: Drawing Demo” and enable the canvas to be drawn, according to the user input.

Part 1.

Store the User Input

- Assuming that the input is as shown in *figure 2* on page 12. After the canvas is enabled, the program will store all the parameters and values, given the user input, in a key-value pair format of a map data structure. Regarding each of the values, if the parameter of its value is either “X_CURRENT”, “Y_CURRENT”, “X_NEXT”, “Y_NEXT”, “WIDTH”, or “HEIGHT”, the program will cast the data type of the value from string to integer. Such an action, shown in *figure 3* in page 13, will enable the program to check the validity of the user input.

Test (part 1)

- In *figure 4* on page 13 and *figure 5* on page 14, for each of the parameters, the program will store its total count values in an integer variable. In addition, the program will check whether there exists a value that does not have a parameter in it. If such a case occurs, the program will show the output as “ERROR: Undefined Prerequisite Input” and finish the execution. Moreover, the program will check whether the parameter is valid. If the

parameter is not valid, the program will show the output as “ERROR: Unrecognized Feature” and finish the execution.

- In *figure 6* on page 14, the program will test whether the total counts of the “X_CURRENT” and “Y_CURRENT” axes are the same, and the program will test whether the total counts of “COLOR” and “SHAPE” values are the same. If the total counts of the “X_CURRENT” and “Y_CURRENT” axes are not the same, the program will show the output as “ERROR: Mismatch Size: Coordinate Input” and finish the execution. If the total counts of “COLOR” and “SHAPE” values are not the same, the program will show the output as “ERROR: Mismatch Size: Color AND Shape Input” and finish the execution.
- In *figure 7* on page 15, the program will test for given valid shape input, whether exactly both “X_NEXT” & “Y_NEXT” and “WIDTH” & “HEIGHT” parameters have no value or are not being initialized. If exactly both “X_NEXT” & “Y_NEXT” and “WIDTH” & “HEIGHT” parameters have no value or are not being initialized, the program will show to the output as “ERROR: Undefined Coordinate Input” and finish the execution. Moreover, the program will test whether all the parameters and their values are valid. For instance, if the “SHAPE” value is “LINE”, then, aside from “X_CURRENT” and “Y_CURRENT” parameters and values that are required to exist in all valid shapes, the program also expects “X_NEXT” and “Y_NEXT” to be included in the user input. Assuming that the user has 4 consecutive valid inputs of the “SHAPE” value of “LINE”, the total counts of “Y_NEXT” values have to be equal to the total counts of “SHAPE” value of “LINE”.

Part 2.

Store the User Input

- In *figure 8* on page 15, for each of the keys and values that are stored in two maps (map for integer data type and map for string data type), the program will split such maps into new maps. The number of unique parameter names from user input is equal to the number of new maps. Each of the new map’s values contains the value of the previous map, which such a value is a list, while each of the new map keys contains the indexes of such a list.

Test (part 2)

- In *figure 9* on page 16, if either the total counts of the input values are incomplete or more than required, the program will show the output as “ERROR: Mismatch Size: Coordinate Input” and finish the execution. Note that we do not need to check whether for instance, the total counts of “SHAPE”, which its value requires “X_NEXT” and “Y_NEXT”, is equal to the total counts of “Y_NEXT”, since such a scenario has been covered in *figure 7* previously.

Set the Color of the Shape

- Java provides 13 predefined colors, which those colors have been implemented in the “COLOR” feature, shown in *figure 10* on page 17. If the user inputs an invalid “COLOR” value, the program will show the output as “ERROR: Undefined Color Input” and finish the execution. Else, the program will execute such a valid color in its corresponding shape.

Core Features

- Draw a Line
 - To draw a line, the user needs to provide a valid value of parameters “X_CURRENT”, “Y_CURRENT”, “X_NEXT”, and “Y_NEXT”. The program will then increment the index of “X_CURRENT” and “Y_CURRENT” parameters to execute the next input if any. Since the program has used “X_NEXT” and “Y_NEXT” parameters for a given input, the program will increment the index of those parameters to execute the next input, if any. The corresponding input can be referred to *figure 11* on page 18.
 - Sample Program (1 input) can be referred to *figure 12* on page 18 and *figure 13* on page 19. *Figure 12* illustrates the sample input and *figure 13* is the sample output.
- Draw a Pentagon
 - To draw a pentagon, the user needs to provide a valid value of parameters “X_CURRENT” and “Y_CURRENT”. The program will then increment the index of “X_CURRENT” and “Y_CURRENT” parameters to execute the next input if any. The corresponding input can be referred to *figure 14* on page 20.
 - Sample Program (1 input) can be referred to *figure 15* on page 20 and *figure 16* on page 21. *Figure 15* illustrates the sample input and *figure 16* is the sample output.
- Draw a Circle
 - To draw a circle, the user needs to provide a valid value of parameters “X_CURRENT”, “Y_CURRENT”, “WIDTH”, and “HEIGHT”. The program will then increment the index of “X_CURRENT” and “Y_CURRENT” parameters to execute the next input if any. Since the program has used “WIDTH” and “HEIGHT” parameters for a given input, the program will increment the index of those parameters to execute the next input, if any. The corresponding input can be referred to *figure 17* on page 21.
 - Sample Program (1 input) can be referred to *figure 18* on page 22, and *figure 19* on page 22. *Figure 18* shows the sample input and *figure 19* shows the sample output of drawing the circle.
- Draw a Rectangle
 - To draw a rectangle, the user needs to provide a valid value of parameters “X_CURRENT”, “Y_CURRENT”, “WIDTH”, and “HEIGHT”. The program will then increment the index of “X_CURRENT” and “Y_CURRENT” parameters to execute the next input if any. Since the program has used “WIDTH” and “HEIGHT” parameters for a given input, the program will increment the index of those parameters to execute the next input, if any. The corresponding input can be referred to *figure 20* on page 23.

- Sample Program (1 input) can be shown with *figure 21* on page 23 and *figure 22* on page 24. *Figure 21* shows the input to generate the figure shape of the rectangle, and *figure 22* shows how the output was obtained.
- Draw a House
 - To draw a house, the user needs to provide a valid value of parameters “X_CURRENT”, “Y_CURRENT”, “WIDTH”, and “HEIGHT”. The program will then increment the index of “X_CURRENT” and “Y_CURRENT” parameters to execute the next input if any. Since the program has used “WIDTH” and “HEIGHT” parameters for a given input, the program will increment the index of those parameters to execute the next input, if any. The corresponding input can be referred to *figure 23* on page 24.
 - Sample Program (1 input) can be shown with *figure 24* on page 25 and *figure 25* on page 25. *Figure 24* is the sample input and *figure 25* as is the sample output.
- Draw a Car
 - To draw a car, the user needs to provide a valid value of parameters “X_CURRENT” and “Y_CURRENT”. The program will then increment the index of “X_CURRENT” and “Y_CURRENT” parameters to execute the next input if any. The corresponding input can be referred to *figure 26* on page 26.
 - Sample Program (1 input) is shown with *figure 27* on page 26 as the input for drawing a car with our programming language and *figure 28* on page 27 as the result of input in *figure 27*.
- Draw a Smiley Face
 - To draw a smiley face, the user needs to provide a valid value of parameters “X_CURRENT” and “Y_CURRENT”. The program will then increment the index of “X_CURRENT” and “Y_CURRENT” parameters to execute the next input if any. The corresponding input can be referred to *figure 29* on page 27.
 - Sample Program (1 input), *figure 30* on page 28 shows how we need to input in our language to draw a smiley face and *figure 31* on page 28 shows what the input would generate.

Test (part 3)

- If the input value of parameter “SHAPE” is undefined, the program will show the output as “ERROR: Undefined Shape Input” and finish the execution. The corresponding input can be referred to *figure 32* on page 29.

Sample Programs (multiple inputs of unique shapes)

2 Inputs

As the sample program of 2 inputs, we have generated a smiley face and the house at the same portrait, the *figure 33* on page 29 shows what values were input handed into our compiler and the *figure 34* on page 30 showing one smiley face and one house in the same screen as the output.

8 Inputs

As the sample program having 8 inputs of different figures, we have generated 1 line, 2 smiley faces, 1 house, 1 pentagon, 1 rectangle, 1 circle and lastly a car. *Figure 35* on page 31 shows what we have used as the input to generate them and *figure 36* on page 32 shows the output how our compiler has created the output like.

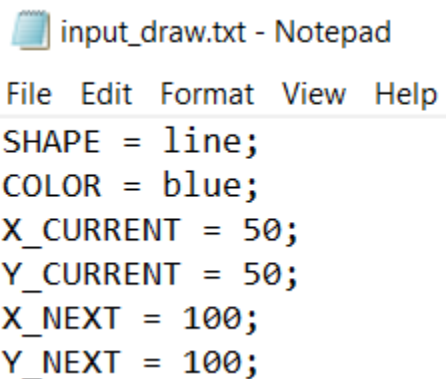
Conclusion

As the conclusion, our team was successful in developing our own programming language, specialized in the purpose of drawing tools. It supports both traditional inputs like string or integer and custom inputs like "HEIGHT" and "COLOR" that determine the features like size of the shape or input that determines the shape like "LINE" and inputs that determine the coordinates of the figure like "X_CURRENT". Also our input can be general that both "Dark GRAY" and "dark GRAY" will be recognized as the standard color "DARK GRAY", showing decent performance for handling errors and exceptions by preparing situations of them occurring. Our language with the compiler has fulfilled all requirements that the project description has asked for our language, and were successful in drawing all following figures. Moreover, our programming language is straightforward and easy to use. We just have to define at which coordination that our figure wants to locate at, input the height and width, and lastly decide which color and shape that the figure wants to be like. It also allows you to read the text file as the input into the compiler rather than asking the user to manually type every line.

Appendix

```
416  → → → // .creating .object .of .JFrame (Window .popup) . LF
417  → → → JFrame .window . = .new . JFrame () ; . LF
418  .. LF
419  → → → // .setting .closing .operation . LF
420  → → → window .setDefaultCloseOperation (JFrame .EXIT_ON_CLOSE) ;
421  .. LF
422  → → → window .getContentPane () .setBackground (Color .WHITE) ; LF
423  → → → // .setting .size .of .the .pop .window . LF
424  → → → window .setBounds (30 , .30 , .600 , .600) ; . LF
425  → → → LF
426  → → → window .setTitle ("Group .XYZ : . Drawing . Demo") ; LF
427  .. LF
428  → → → // .setting .canvas .for .draw . LF
429  → → → window .getContentPane () .add (new . MyCanvas () ) ; . LF
430  → → → LF
431  .. LF
432  → → → // .set .visibility . LF
433  → → → window .setVisible (true) ; . LF
```

Figure 1. Initialize the canvas.



```
input_draw.txt - Notepad
File Edit Format View Help
SHAPE = line;
COLOR = blue;
X_CURRENT = 50;
Y_CURRENT = 50;
X_NEXT = 100;
Y_NEXT = 100;
```

Figure 2. Input for the “Draw a Line” feature.


```

104 for (String statement : map_integers.keySet()) {
105     if (!map_integers.get(statement.toUpperCase()).isEmpty()) {
106         if (statement.toUpperCase().equals("X_CURRENT")) {
107             SIZE_X_CURRENT = map_integers.get(statement.toUpperCase()).size();
108         }
109         else if (statement.toUpperCase().equals("Y_CURRENT")) {
110             SIZE_Y_CURRENT = map_integers.get(statement.toUpperCase()).size();
111         }
112         else if (statement.toUpperCase().equals("X_NEXT")) {
113             SIZE_X_NEXT = map_integers.get(statement.toUpperCase()).size();
114         }
115         else if (statement.toUpperCase().equals("Y_NEXT")) {
116             SIZE_Y_NEXT = map_integers.get(statement.toUpperCase()).size();
117         }
118         else if (statement.toUpperCase().equals("WIDTH")) {
119             SIZE_WIDTH = map_integers.get(statement.toUpperCase()).size();
120         }
121         else if (statement.toUpperCase().equals("HEIGHT")) {
122             SIZE_HEIGHT = map_integers.get(statement.toUpperCase()).size();
123         }
124         else {
125             System.out.println("ERROR: Unrecognized Feature");
126             return;
127         }
128     }
129     else {
130         System.out.println("ERROR: Undefined Prerequisite Input");
131         return;
132     }
133 }

```

Figure 5. Test for the parameters name of integer data type, whether it is a valid name.

```

144 if (SIZE_X_CURRENT != SIZE_Y_CURRENT) {
145     System.out.println("ERROR: Mismatch Size: Coordinate Input");
146     return;
147 }
148 if (SIZE_COLOR != SIZE_SHAPE) {
149     System.out.println("ERROR: Mismatch Size: Color AND Shape Input");
150     return;
151 }

```

Figure 6. Test on whether the total counts of coordinate inputs of X are equal to the total counts of coordinate inputs of Y, and whether the total counts of values of color are equal to the total counts of values of shape.

```

160 → → → → → for. (String.shape :: map_strings.get("SHAPE")) {
161 → → → → → if. (shape.toUpperCase().equals("LINE")) {
162 → → → → → {
163 → → → → → → SIZE_SHAPE_1++;
164 → → → → → }
165 → → → → → else if. (shape.toUpperCase().equals("CIRCLE")) {
166 → → → → → → || shape.toUpperCase().equals("RECTANGLE")
167 → → → → → → || shape.toUpperCase().equals("HOUSE")) {
168 → → → → → {
169 → → → → → → SIZE_SHAPE_2++;
170 → → → → → }
171 → → → → → else if. (shape.toUpperCase().equals("CAR")) {
172 → → → → → → || shape.toUpperCase().equals("SMILEY")
173 → → → → → → || shape.toUpperCase().equals("PENTAGON")) {
174 → → → → → {
175 → → → → → → exist_shape_only_pos.=.1;
176 → → → → → }
177 → → → → → }
178 → → → → → if. (exist_shape_only_pos==.0) {
179 → → → → → if. ((SIZE_X_NEXT==.0 && SIZE_Y_NEXT==.0)
180 → → → → → && (SIZE_WIDTH==.0 && SIZE_HEIGHT==.0)) {
181 → → → → → {
182 → → → → → → System.out.println("ERROR: Undefined.Coordinate.Input");
183 → → → → → → return;
184 → → → → → }
185 → → → → → }
186 → → → → → if. (SIZE_SHAPE_1 != SIZE_Y_NEXT || SIZE_SHAPE_2 != SIZE_HEIGHT) {
187 → → → → → {
188 → → → → → → System.out.println("ERROR: Mismatch.Size.Coordinate.Input");
189 → → → → → → return;
190 → → → → → }

```

Figure 7. Test whether all the parameters and their values are valid.

```

204 → → → → → for. (String.statement :: map_integers.keySet()) {
205 → → → → → for. (int.j = .0; j < map_integers.get(statement).size(); j++) {
206 → → → → → if. (statement.toUpperCase().equals("X_CURRENT")) {
207 → → → → → → x_current.put(new Integer(j), map_integers.get(statement.toUpperCase()).get(j));
208 → → → → → }
209 → → → → → else if. (statement.toUpperCase().equals("Y_CURRENT")) {
210 → → → → → → y_current.put(new Integer(j), map_integers.get(statement.toUpperCase()).get(j));
211 → → → → → }
212 → → → → → else if. (statement.toUpperCase().equals("X_NEXT")) {
213 → → → → → → x_next.put(new Integer(j), map_integers.get(statement.toUpperCase()).get(j));
214 → → → → → }
215 → → → → → else if. (statement.toUpperCase().equals("Y_NEXT")) {
216 → → → → → → y_next.put(new Integer(j), map_integers.get(statement.toUpperCase()).get(j));
217 → → → → → }
218 → → → → → else if. (statement.toUpperCase().equals("WIDTH")) {
219 → → → → → → width.put(new Integer(j), map_integers.get(statement.toUpperCase()).get(j));
220 → → → → → }
221 → → → → → else if. (statement.toUpperCase().equals("HEIGHT")) {
222 → → → → → → height.put(new Integer(j), map_integers.get(statement.toUpperCase()).get(j));
223 → → → → → }
224 → → → → → }
225 → → → → → }
226 → → → → → }
227 → → → → → for. (String.statement :: map_strings.keySet()) {
228 → → → → → for. (int.j = .0; j < map_strings.get(statement).size(); j++) {
229 → → → → → if. (statement.toUpperCase().equals("COLOR")) {
230 → → → → → → color.put(new Integer(j), map_strings.get(statement.toUpperCase()).get(j));
231 → → → → → }
232 → → → → → else if. (statement.toUpperCase().equals("SHAPE")) {
233 → → → → → → shape.put(new Integer(j), map_strings.get(statement.toUpperCase()).get(j));
234 → → → → → }
235 → → → → → }
236 → → → → → }

```

Figure 8. Store input assignment statements, according to their parameter name.


```

249 → → → if (shape.get(idx).toUpperCase().equals("LINE")) {LF
250 → → → {LF
251 → → → if (x_next.size() != y_next.size() || y_next.size() > x_current.size())LF
252 → → → {LF
253 → → → System.out.println("ERROR: Mismatch.Size: Coordinate.Input");LF
254 → → → break;LF
255 → → → }LF
256 → → → }LF
257 → → → else if (shape.get(idx).toUpperCase().equals("CIRCLE")) {LF
258 → → → || shape.get(idx).toUpperCase().equals("RECTANGLE") {LF
259 → → → || shape.get(idx).toUpperCase().equals("HOUSE") {LF
260 → → → || shape.get(idx).toUpperCase().equals("CAR") {LF
261 → → → || shape.get(idx).toUpperCase().equals("SMILEY") {LF
262 → → → || shape.get(idx).toUpperCase().equals("PENTAGON") {LF
263 → → → {LF
264 → → → if (width.size() != height.size() || height.size() > x_current.size()) {LF
265 → → → {LF
266 → → → System.out.println("ERROR: Mismatch.Size: Coordinate.Input");LF
267 → → → break;LF
268 → → → }LF
269 → → → }LF

```

Figure 9. Test whether for given values of shape, the corresponding input values are either incomplete or more than required.

```

270      switch(color.get(idx).toUpperCase()) {
271      {
272          case "RED":
273              g2d.setColor(Color.RED);
274              break;
275          case "GREEN":
276              g2d.setColor(Color.GREEN);
277              break;
278          case "BLUE":
279              g2d.setColor(Color.BLUE);
280              break;
281          case "YELLOW":
282              g2d.setColor(Color.YELLOW);
283              break;
284          case "BLACK":
285              g2d.setColor(Color.BLACK);
286              break;
287          case "CYAN":
288              g2d.setColor(Color.CYAN);
289              break;
290          case "DARK_GRAY":
291              g2d.setColor(Color.DARK_GRAY);
292              break;
293          case "LIGHT_GRAY":
294              g2d.setColor(Color.LIGHT_GRAY);
295              break;
296          case "MAGENTA":
297              g2d.setColor(Color.MAGENTA);
298              break;
299          case "ORANGE":
300              g2d.setColor(Color.ORANGE);
301              break;
302          case "PINK":
303              g2d.setColor(Color.PINK);
304              break;
305          case "WHITE":
306              g2d.setColor(Color.WHITE);
307              break;
308          default:
309              System.out.println("ERROR: Undefined Color Input");
310              return;
311      }

```

Figure 10. Given 13 possible colors, color the shape based on the user input color value.

```

317 case "LINE":
318     g2d.drawLine
319     (
320         x_current.get(idx),
321         y_current.get(idx),
322         x_next.get(idx_next),
323         y_next.get(idx_next)
324     );
325     idx_next++;
326     break;

```

Figure 11. Draw a line shape, according to the user input coordinates.

input_draw.txt - Notepad

File Edit Format View Help

```

SHAPE = line;
COLOR = DARK Gray;
X_CURRENT = 200;
Y_CURRENT = 50;
X_NEXT = 300;
Y_NEXT = 150;

```

Figure 12. Exactly 1 Input for the “Draw a Line” feature.

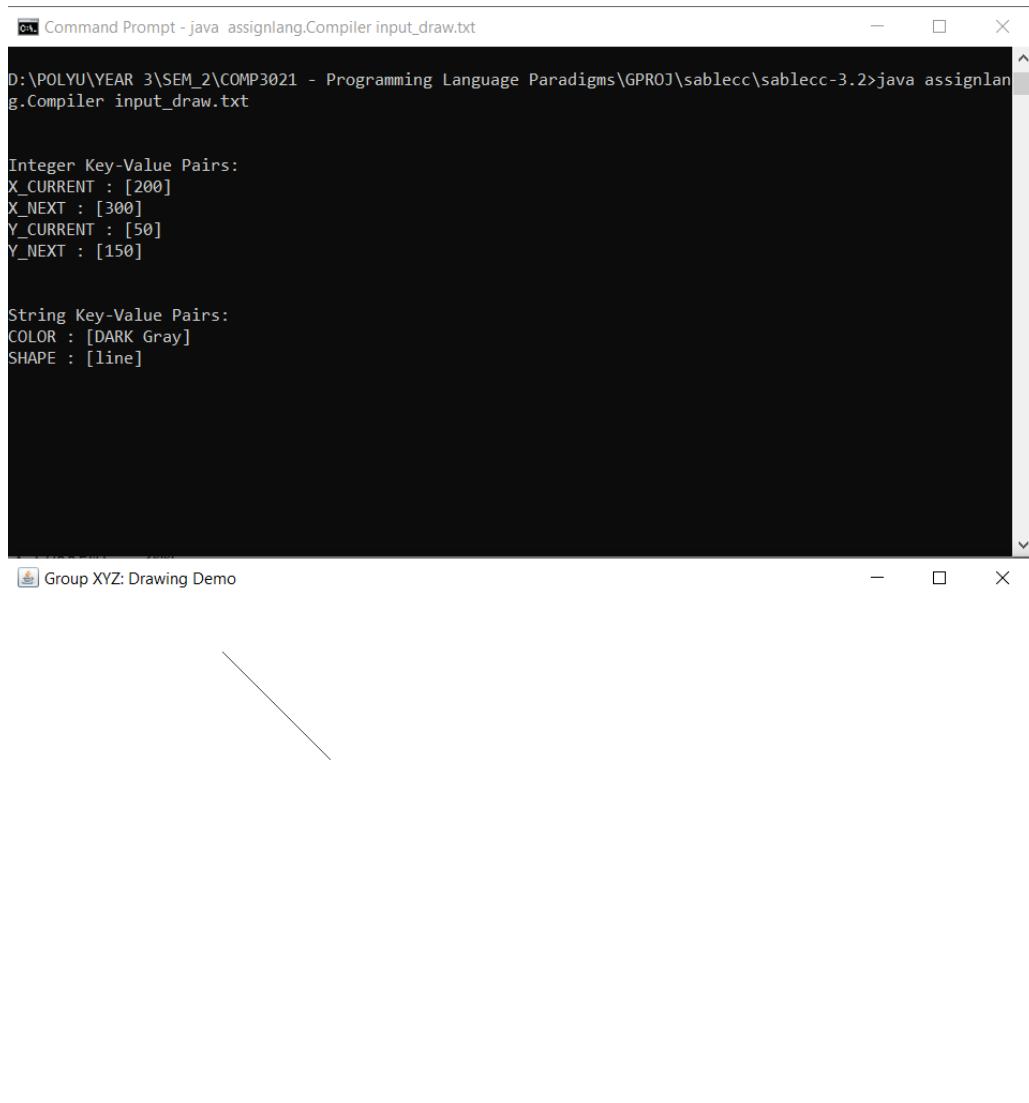


Figure 13. Exactly 1 Output for the “Draw a Line” feature.

```

327 case."PENTAGON":
328     angle_rad_double=0;
329     xcur_double.=x_current.get(idx);
330     ycur_double.=y_current.get(idx);
331     for (int i=0;i<5;i++) {
332         angle_rad_double+=2*Math.PI/5;
333         xnext_double=xcur_double+100*Math.cos(angle_rad_double);
334         ynext_double=ycur_double+100*Math.sin(angle_rad_double);
335         g2d.drawLine(
336             (int)xcur_double,
337             (int)ycur_double,
338             (int)xnext_double,
339             (int)ynext_double);
340         xcur_double=xnext_double;
341         ycur_double=ynext_double;
342     }
343     break;
344 }
345 }
346 }

```

Figure 14. Draw a pentagon shape, according to the user input coordinates.

input_draw.txt - Notepad

File Edit Format View Help

```

SHAPE = Pentagon;
COLOR = green;
X_CURRENT = 250;
Y_CURRENT = 250;

```

Figure 15. Exactly 1 Input for the “Draw a Pentagon” feature.

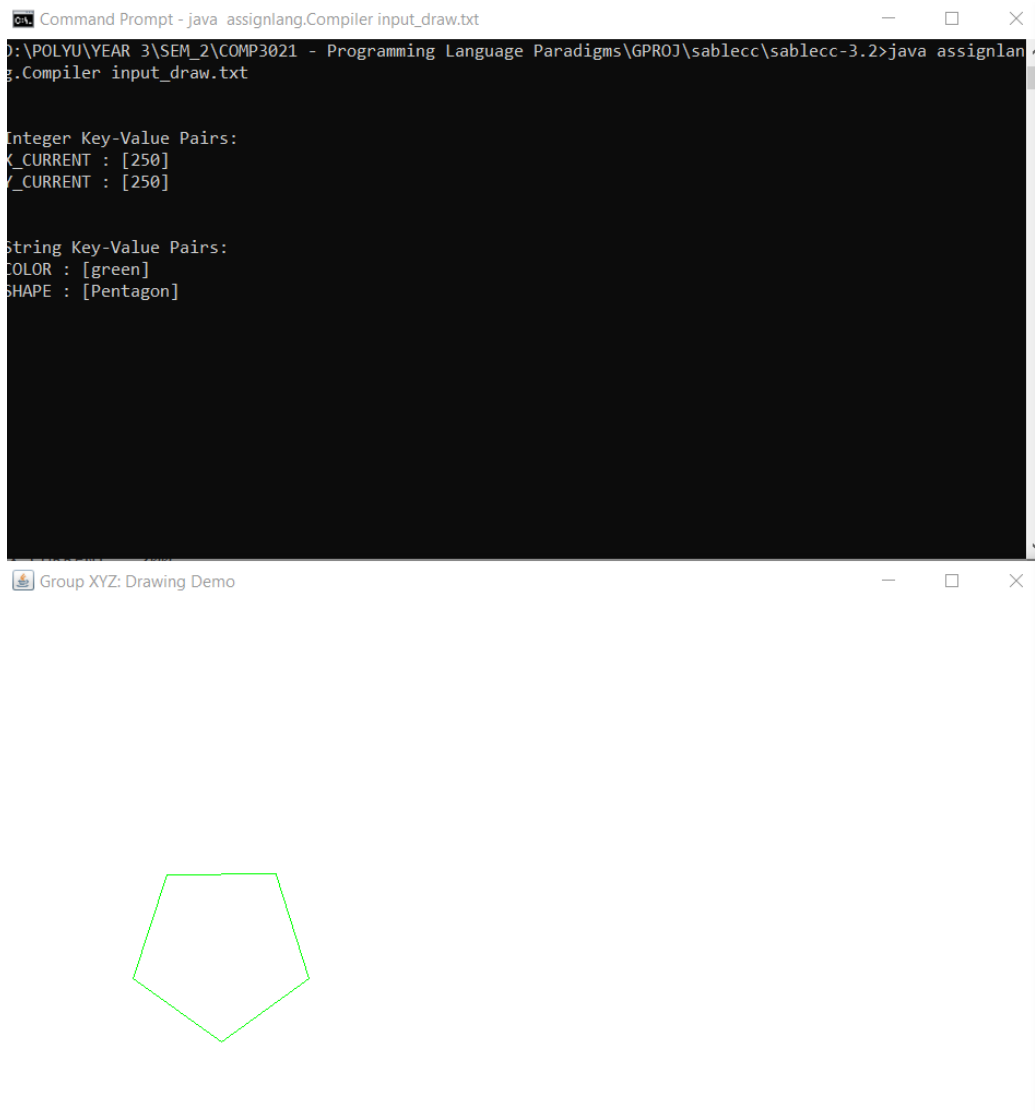


Figure 16. Exactly 1 Output for the “Draw a Pentagon” feature.

```

348  → → → → → case . "CIRCLE": 00
349  → → → → → g2d.drawOval 00
350  → → → → → ( 00
351  → → → → → x_current.get (idx) , 00
352  → → → → → y_current.get (idx) , 00
353  → → → → → width.get (idx_width_height) ,
354  → → → → → height.get (idx_width_height)
355  → → → → → ) ; 00
356  → → → → → idx_width_height++ ; 00
357  → → → → → break ; 00

```

Figure 17. Draw a circle shape, according to the user input coordinates and size.


 input_draw.txt - Notepad
File Edit Format View Help
SHAPE = Circle;
COLOR = Orange;
X_CURRENT = 300;
Y_CURRENT = 300;
WIDTH = 100;
HEIGHT = 100;

Figure 18. Exactly 1 Input for the “Draw a Circle” feature.

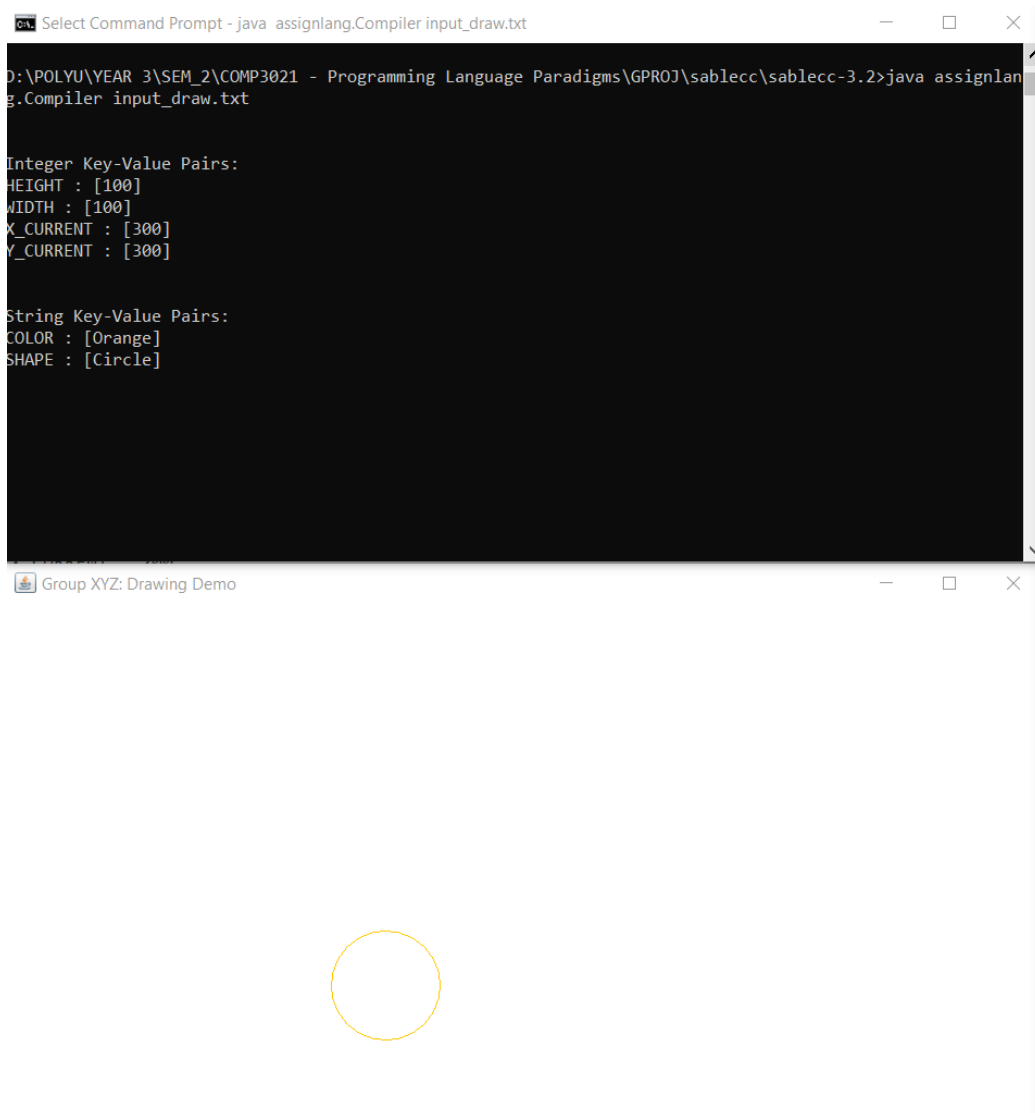


Figure 19. Exactly 1 Output for the “Draw a Circle” feature.

```

358  → → → → → case."RECTANGLE":LF
359  → → → → → g2d.drawRectLF
360  → → → → → (LF
361  → → → → →     x_current.get(idx),LF
362  → → → → →     y_current.get(idx),LF
363  → → → → →     width.get(idx_width_height),
364  → → → → →     height.get(idx_width_height)
365  → → → → → );LF
366  → → → → → idx_width_height++;LF
367  → → → → → break;LF

```

Figure 20. Draw a rectangle shape, according to the user input coordinates and size.

input_draw.txt - Notepad

File Edit Format View Help

```

SHAPE = rectangle;
COLOR = light gray;
X_CURRENT = 200;
Y_CURRENT = 200;
WIDTH = 200;
HEIGHT = 200;

```

Figure 21. Exactly 1 Input for the “Draw a Rectangle” feature.

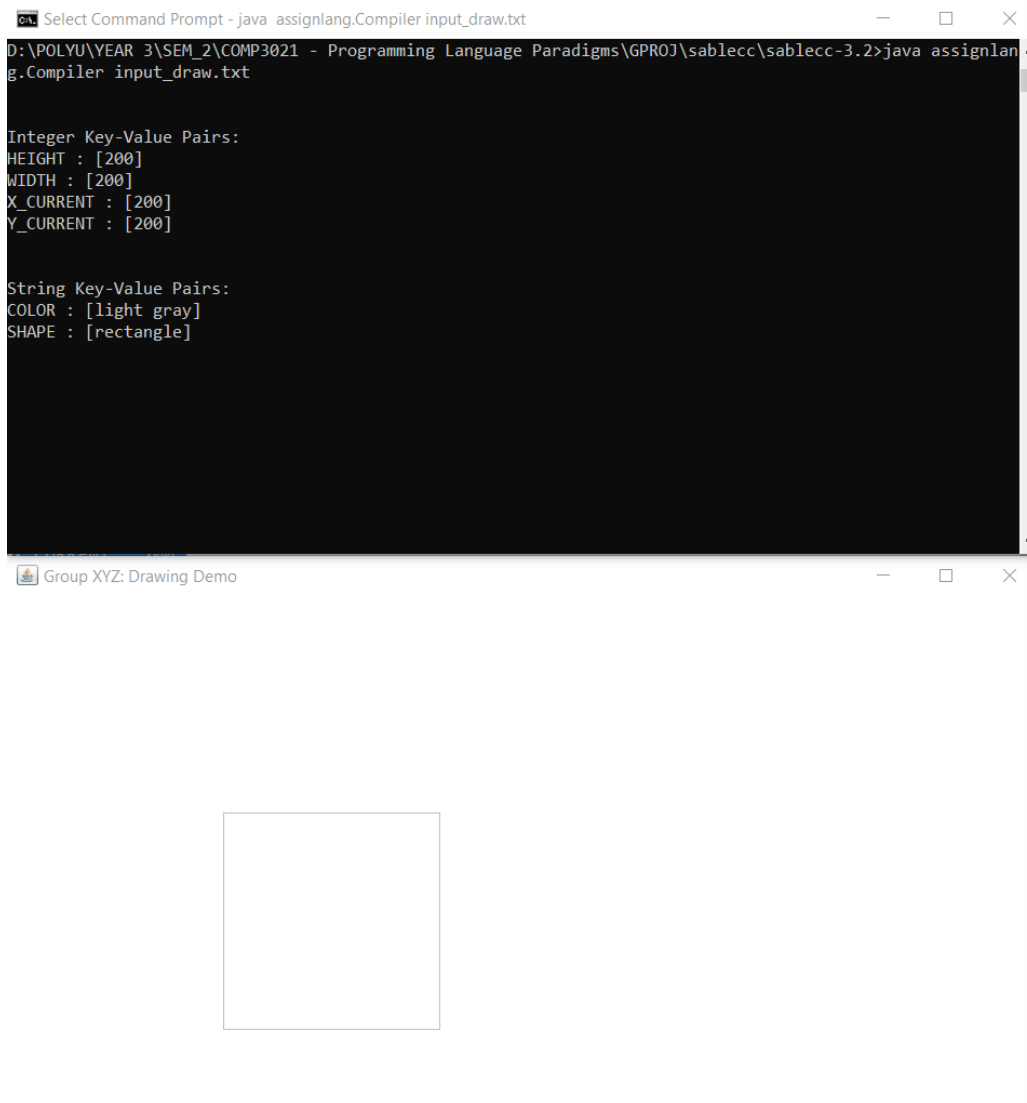


Figure 22. Exactly 1 Output for the “Draw a Rectangle” feature.

```

368 case "HOUSE":
369     xcur_int = x_current.get(idx);
370     ycur_int = y_current.get(idx);
371     w_int = width.get(idx_width_height);
372     h_int = height.get(idx_width_height);
373
374     g2d.drawRect(xcur_int, ycur_int, w_int, h_int);
375     // Draw front of house.
376     g2d.drawRect(xcur_int - (w_int/20), ycur_int, w_int + (w_int/10), h_int/3); // Roof with some overhang.
377     g2d.drawRect(xcur_int + (2 * w_int/3), ycur_int + h_int/2, w_int/3, h_int/2); // Draw the door.
378     g2d.drawRect(xcur_int + (w_int/4), ycur_int + h_int/2, w_int/8, h_int/8); // Draw the window.
379     g2d.drawRect(xcur_int + (3 * w_int/4), ycur_int - h_int/8, w_int/8, h_int/8); // Draw the chimney.
380     idx_width_height++;
381     break;

```

Figure 23. Draw a house shape, according to the user input coordinates and size.


 input_draw.txt - Notepad
File Edit Format View Help
SHAPE = House;
COLOR = BLUE;
X_CURRENT = 50;
Y_CURRENT = 100;
WIDTH = 100;
HEIGHT = 80;|

Figure 24. Exactly 1 Input for the “Draw a House” feature.

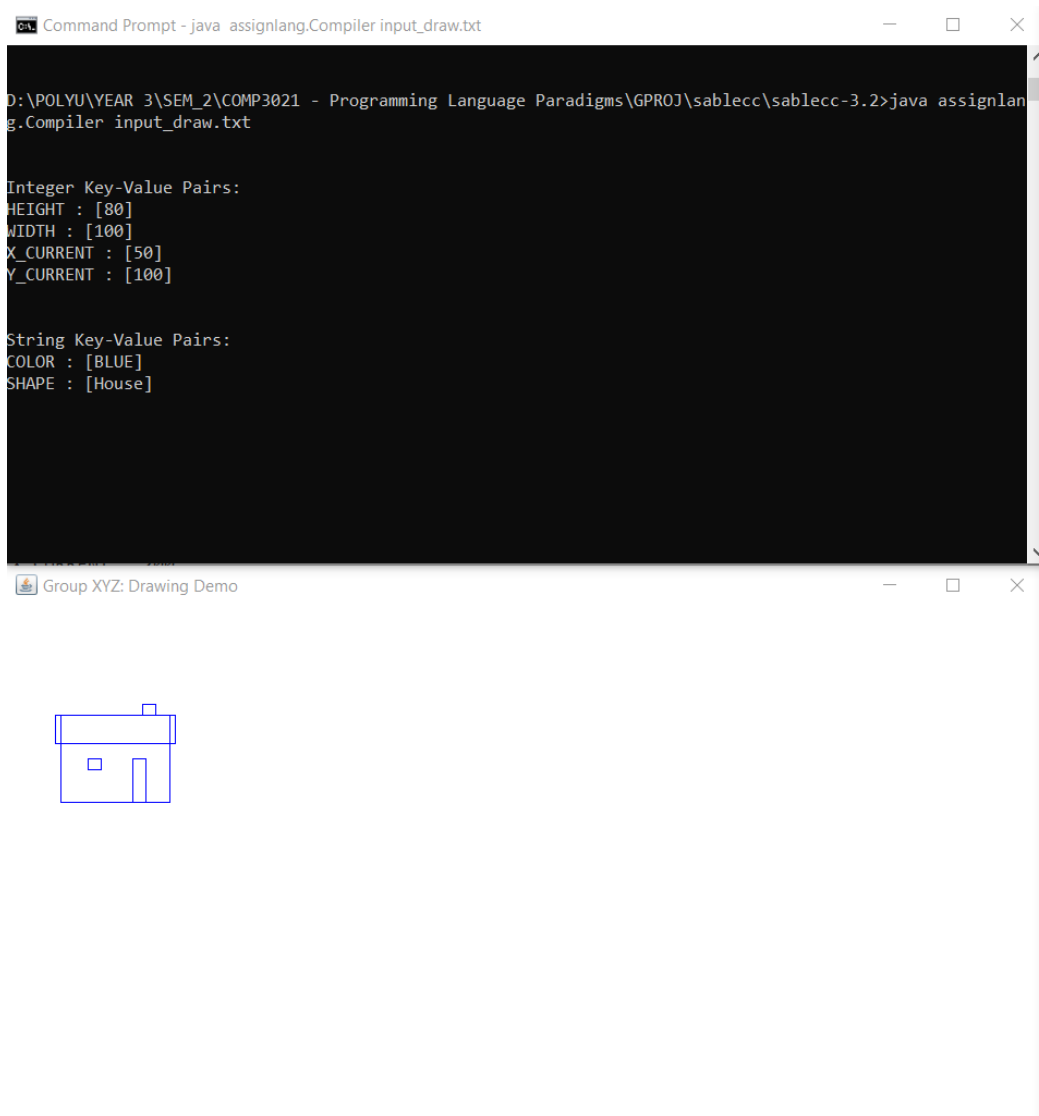


Figure 25. Exactly 1 Output for the “Draw a House” feature.

```

382         case "CAR":
383             xcur_int = x_current.get(idx)
384             ycur_int = y_current.get(idx)
385             g2d.drawRect(xcur_int, ycur_int + 10, 60, 10)
386             g2d.drawOval(xcur_int + 10, ycur_int + 20, 10, 10)
387             g2d.drawOval(xcur_int + 40, ycur_int + 20, 10, 10)
388             g2d.drawLine(xcur_int + 10, ycur_int + 10, xcur_int + 20, ycur_int)
389             g2d.drawLine(xcur_int + 20, ycur_int, xcur_int + 40, ycur_int)
390             g2d.drawLine(xcur_int + 40, ycur_int, xcur_int + 50, ycur_int + 10)
391         break;
392

```

Figure 26. Draw a car shape, according to the user input coordinates.

input_draw.txt - Notepad

File Edit Format View Help

```

SHAPE = Car;
COLOR = Cyan;
X_CURRENT = 200;
Y_CURRENT = 400;

```

Figure 27. Exactly 1 Input for the “Draw a Car” feature.

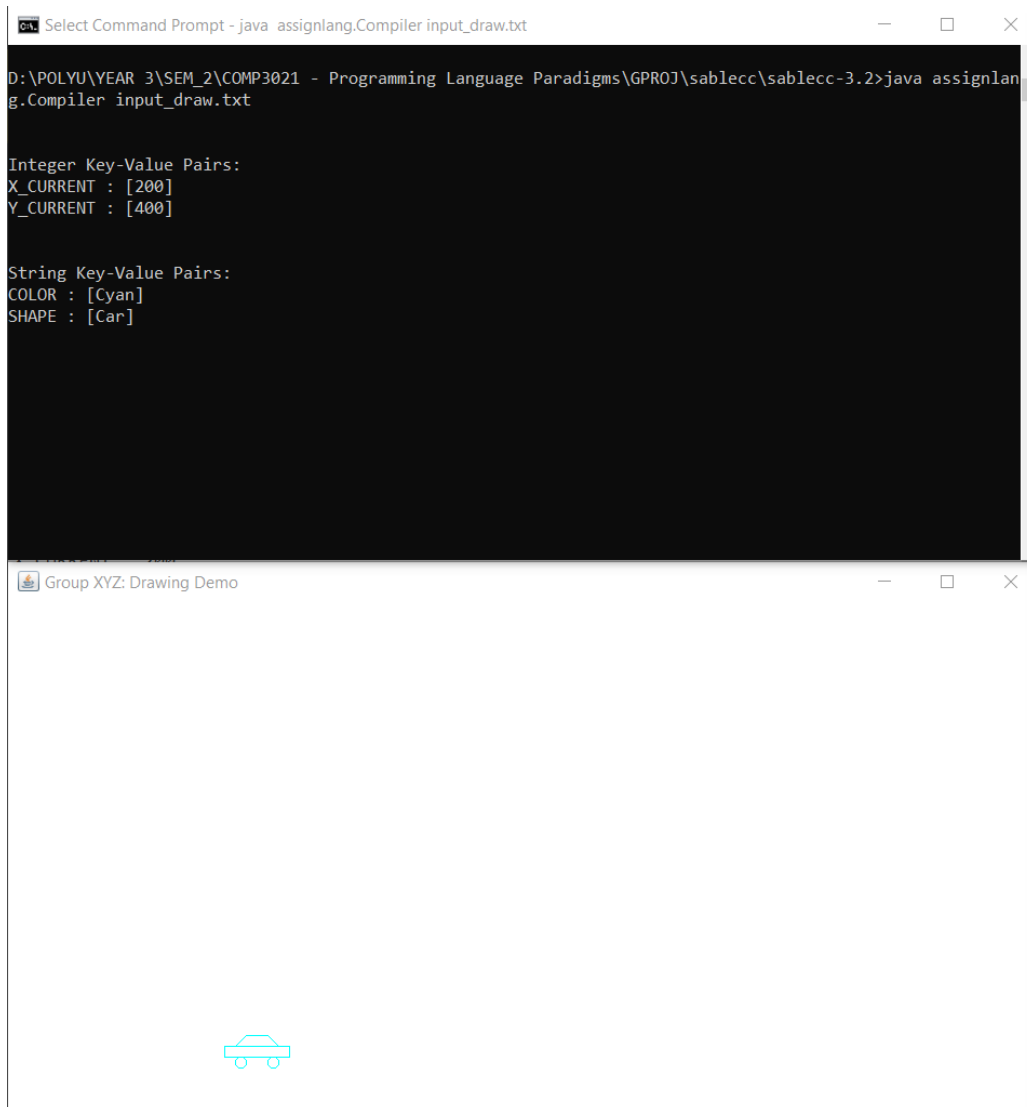


Figure 28. Exactly 1 Output for the “Draw a Car” feature.

```

393  case "SMILEY":
394      xcur_int = x_current.get(idx);
395      ycur_int = y_current.get(idx);
396      //
397      int rad = 3 * w_int;
398      //
399      // Ovals for eyes
400      g2d.drawOval(xcur_int, ycur_int, 150, 150);
401      g2d.drawOval(xcur_int+40, ycur_int+40, 15, 15);
402      g2d.drawOval(xcur_int+90, ycur_int+40, 15, 15);
403      //
404      // Arc for the smile
405      g2d.drawArc(xcur_int+50, ycur_int+110, 50, 20, 180, 180);
406      break;

```

Figure 29. Draw a smiley face shape, according to the user input coordinates.

```
input_draw.txt - Notepad
File Edit Format View Help
SHAPE = smiley;
COLOR = magenta;
X_CURRENT = 200;
Y_CURRENT = 60;
```

Figure 30. Exactly 1 Input for the “Draw a Smiley Face” feature.

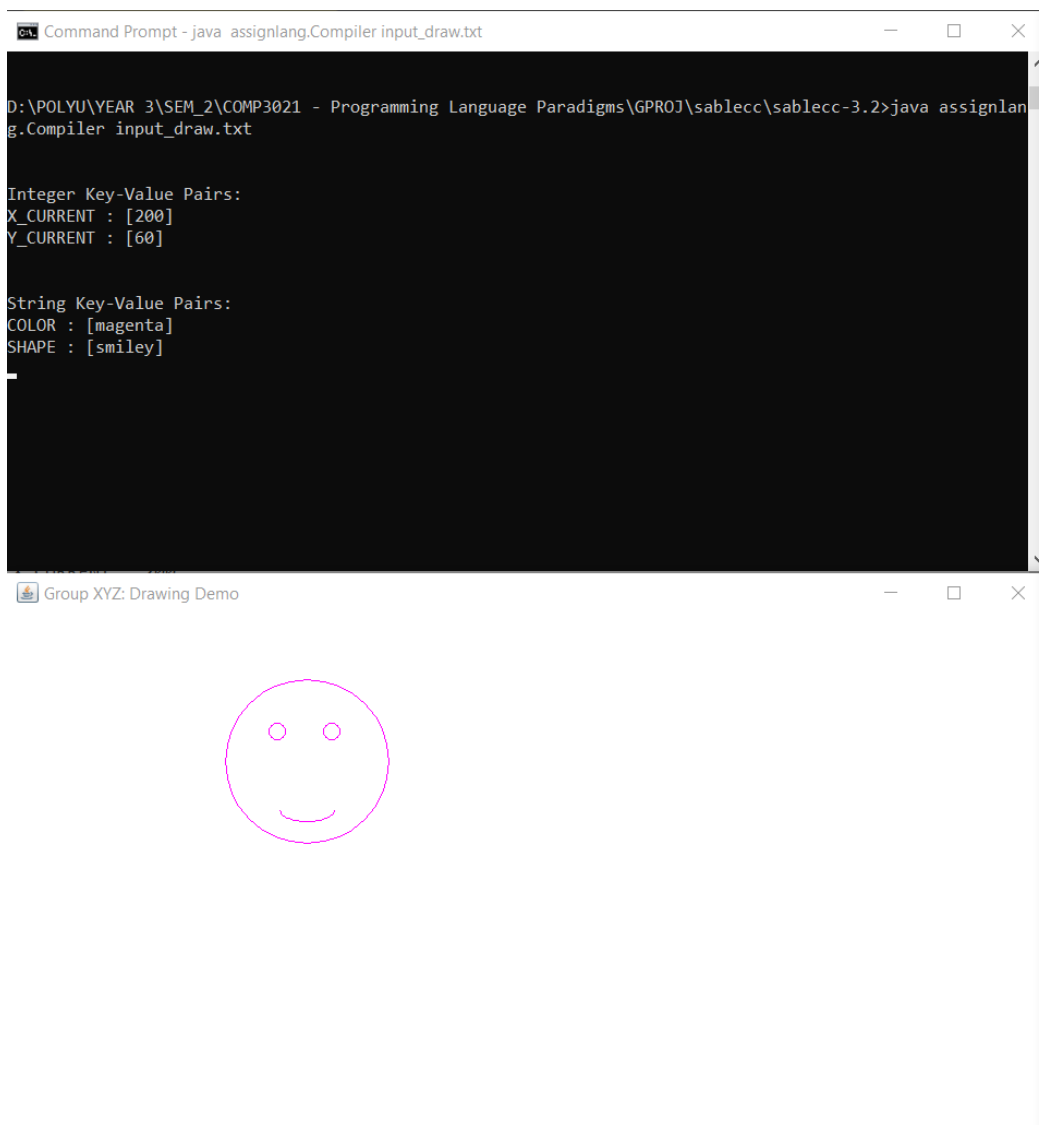


Figure 31. Exactly 1 Output for the “Draw a Smiley Face” feature.

```

407 → → → → → → → → default: 0;
408 → → → → → → → → System.out.println("ERROR: Undefined Shape Input");
409 → → → → → → → → return; 0;

```

Figure 32. Handle Unknown Shape Input.

input_draw.txt - Notepad

File Edit Format View Help

```

SHAPE = Circle;
COLOR = Orange;
X_CURRENT = 300;
Y_CURRENT = 300;
WIDTH = 100;
HEIGHT = 100;

SHAPE = House;
COLOR = BLUE;
X_CURRENT = 50;|
Y_CURRENT = 100;
WIDTH = 100;
HEIGHT = 80;

```

Figure 33. Exactly 2 Inputs for “Draw a Circle” and “Draw a House” features.

```
Select Command Prompt - java assignlang.Compiler input_draw.txt
D:\POLYU\YEAR 3\SEM_2\COMP3021 - Programming Language Paradigms\GPROJ\sablecc\sablecc-3.2>java assignlang.Compiler input_draw.txt

Integer Key-Value Pairs:
HEIGHT : [100, 80]
WIDTH : [100, 100]
X_CURRENT : [300, 50]
Y_CURRENT : [300, 100]

String Key-Value Pairs:
COLOR : [Orange, BLUE]
SHAPE : [Circle, House]
```

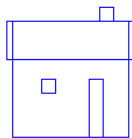


Figure 34. Exactly 2 Outputs for “Draw a Circle” and “Draw a House” features.

input_draw.txt - Notepad

File Edit Format View Help

```
SHAPE = line;
COLOR = DARK Gray;
X_CURRENT = 200;
Y_CURRENT = 50;
X_NEXT = 300;
Y_NEXT = 150;

SHAPE = smiley;
COLOR = Red;
X_CURRENT = 100;
Y_CURRENT = 30;
|
SHAPE = Circle;
COLOR = Orange;
X_CURRENT = 300;
Y_CURRENT = 300;
WIDTH = 100;
HEIGHT = 100;

SHAPE = Pentagon;
COLOR = green;
X_CURRENT = 250;
Y_CURRENT = 250;

SHAPE = rectangle;
COLOR = light gray;
X_CURRENT = 200;
Y_CURRENT = 200;
WIDTH = 200;
HEIGHT = 200;

SHAPE = House;
COLOR = BLUE;
X_CURRENT = 50;
Y_CURRENT = 100;
WIDTH = 100;
HEIGHT = 80;

SHAPE = Car;
COLOR = Cyan;
X_CURRENT = 200;
Y_CURRENT = 400;

SHAPE = smiley;
COLOR = magenta;
X_CURRENT = 200;
Y_CURRENT = 60;
```

Figure 35. Exactly 8 Inputs for multiple features.

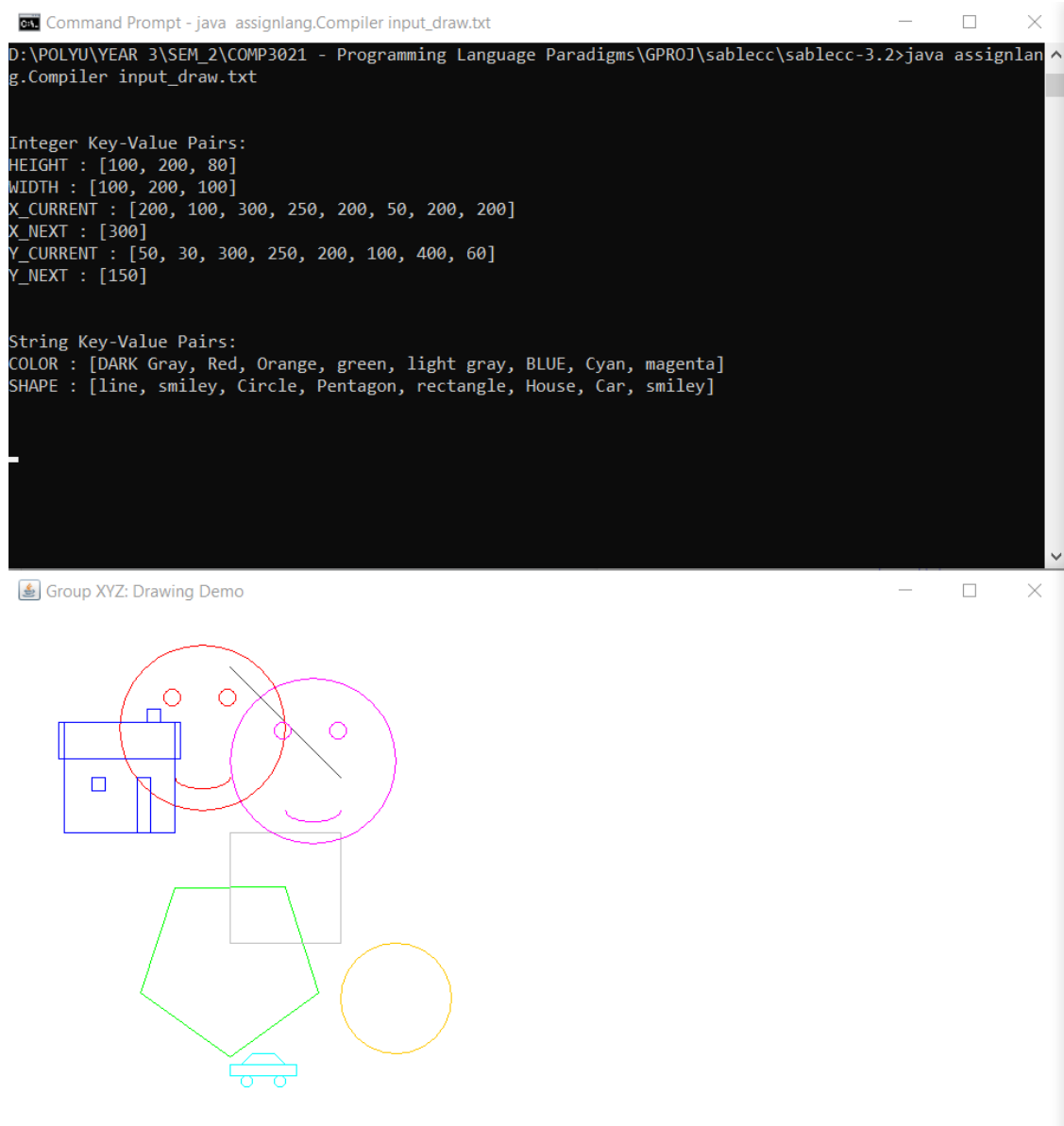


Figure 36. Exactly 8 Outputs for multiple features.