Project Documentation  
Emoji Maker /   
Face Maker

Author: Mike LeBlanc

Date: February 13, 2024

Table of Contents

[Project Requirements 3](#_Toc158809797)

[Design Plans 4](#_Toc158809798)

[Command Design Pattern 4](#_Toc158809799)

[Principals 4](#_Toc158809800)

[Implementation 5](#_Toc158809801)

[ICommand 5](#_Toc158809802)

[Class Diagram 5](#_Toc158809803)

[Testing 6](#_Toc158809804)

# Project Requirements

Use “HelpNDoc” to create a help file:

1. Include Windows "Help" for your latest face-maker assignment.
2. Identify all the controls where help is required.
   1. Short-cut keys
   2. Buttons
   3. Menus
3. Create a Windows Help file.
   1. Include a “Home Page” to provide overall help with the app. And a “hot key” page to show what keys do what.
   2. Include a way to navigate between help topics.
4. Integrate your Windows file in your face-maker apps.
   1. Create a “help” menu pull down for your app.

# Design Plans

## Command Design Pattern

The command pattern is a design pattern that enables developers to encapsulate requests or operations into separate objects, making it possible to modify the requests dynamically and execute them without the sender or receiver knowing anything about each other.

We execute this by connecting our Command Handler class to the function we expect to happen, when this command is called, the function can fire.

We take this a step further by adding a key binding that when pressed, calls the command handler.

This facilitates the steps needed to allow the same function to be called by both the menu selection and the hotkey.

## Principals

Through the course of this application, we have implemented various coding principals including Single Responsibility and Open-closed principals. Apart from the transparency needed to show differences between internal assignments, the code-behind from the XAML has a Single Responsibility, as well as the Faces class and Command Handler class.

These classes are Open-closed because they are open for extension while being closed to modification. This forces other programmers to extend the functionality of what it does rather than change the core methods.

# Implementation

## ICommand

A screen shot of a computer program

Description automatically generated

With this ICommand class, we can accept a hotkey command and map it correctly to the desired outcome. We are then able to provide the user with clear and concise instructions on what that hotkey is, and what the hotkey does.

## Class Diagram

A screenshot of a computer screen

Description automatically generated

Our class diagram shows the simplicity of our class structure and overall application. This aids with ease of maintenance and bug fixing as there are not many other classes to trace errors back through.

# Testing

A yellow circle with white border

Description automatically generated

Figure 1: Application state upon start.

A yellow circle with black text

Description automatically generated

Figure 2: Menu icons with associated hotkeys shown.

A cartoon face with red hair

Description automatically generated

Figure 3: Random Face hotkey pressed. CTRL+R

A screenshot of a computer

Description automatically generated

Figure 4: Help menu option brings up Help Window