Hawk-Eye Innovations Technical Assessment

Design Choices

I leveraged OOP design principles when creating the games for the CLI, by adding a Game interface, which ensures that every class that implements it must have a startGame() method. This meant that the CLI menu is easily extendable, and new games will work with little work. I also created a multi-class solution to implement the Card system, with an overarching interface with regular cards and joker card implementations. I decided this was the best solution, as the logic for the Joker was different, since it isn't part of a suit and has different Image resources. Additionally, it meant that when I created the deck I could add both types of cards to Collections.

When I was designing the CLI version, I was actively thinking about what logic could be reused by the CLI and GUI versions of the game, and I moved that logic into a helper class that both versions utilise. This cut down on repeated code and consequently made the code shorter.

I wanted every game to be able to be played as many times as the user wants, so added options to play again on the CLI and GUI versions of the games. The GUI version also has the option to switch games, if the user wants to do so.

When I was researching solutions to implement the cards on the GUI, I thought that adding each card individually was too time-consuming and not an elegant solution, so instead I used a texture map where all the cards are on a single image and then you can programmatically cut out the card you need in that moment. However, I couldn't find an image with jokers included, so I added those separately, which again fed into the support of separating the Joker and regular class files.

I added smooth animations to both GUI games, to give the user useful feedback on what the game was doing behind the scenes. In the Higher or Lower game, the cards flip and in Blackjack, the new cards fly from the deck, and go to either the player or the dealer, to show they are being drawn .

Areas for Improvement

The GUI library I used, JavaFX, looks quite dated and is hard to install, so it wouldn't be suitable for deploying my games. In the future, I would choose an easier library or a different programming language such as HTML/JavaScript or Python to create the games.

The texture map for my cards is quite low resolution, so when I scale up the cards, they can look quite blurry. Next time, I would find a higher resolution image, to prevent this dip in quality, as it makes the whole game look worse. I could've kept the original size of the cards, but I thought they were too small for the game.

In terms of testing, I only tested with human tests trying to play the game, but in the future I could've used a testing framework like Junit to test each parts of my individual game logic, to ensure that it worked when integrating into the rest of the program.