Group: Group 8

Group name: Ace and Speed Masters

contributors: Lelethu Conjwayo & Caitlin Elliot

Background

Ekasi Jack is a game that is a more friendlier version of Black jack. The original black jack game was invented during the 1700’s by the French/ Italian and it is a gambling game that was invented during World War I (Rainbow Casino, 2020). Ekasi Jack doesn’t have an exact background history because it is a game that one of our team members grew up playing and was passed down by generations before her.

Motivation and Aim

The primary motivation and aim for the game is to create a more friendly and laid back version of black jack . A lot of people when playing games do not like to be mentally stimulated they want play games to unwind ,without heavy rules, it provides a more relaxing and entertaining environment.

Problem Statement

Similarly to Speed masters a lot of games are too complex to understand , they require focus EKASI-JACK accommodates that aspect by simplifying and makes it more understandable and enjoyable for everyone.

Approach

As mentioned in our project proposal we have maintained storing the card deck in an array lists there were no changes with regards to that.

The game remains multiplier and we decided not to make it a single player game because doing so would have required AI. The point of the game is to be played with various people to make it more fun so a multiplayer format suits the game’s vision.

Unfortunately because our game is only functional on terminal the players will be able too see each other’s cards, which is a limitation that we acknowledge. The game as I stated has not been implemented on the graphical user interface so we did not use any applications on making it more visually appealing.

Initially we had point calculators on determining winners but we made changes to simplify the game not only for the users but for the extent of out programming ability as well. Now the focus is determining the loser rather than the winner s mostly focused on the loser which is the person that will be left with the Jack of spades card at the end of the game.

SDLC

Our game follows the Waterfall Model framework because the game requirements were well defined from the onset and before we moved to another phase we reviewed and decided which changes to make before moving on.

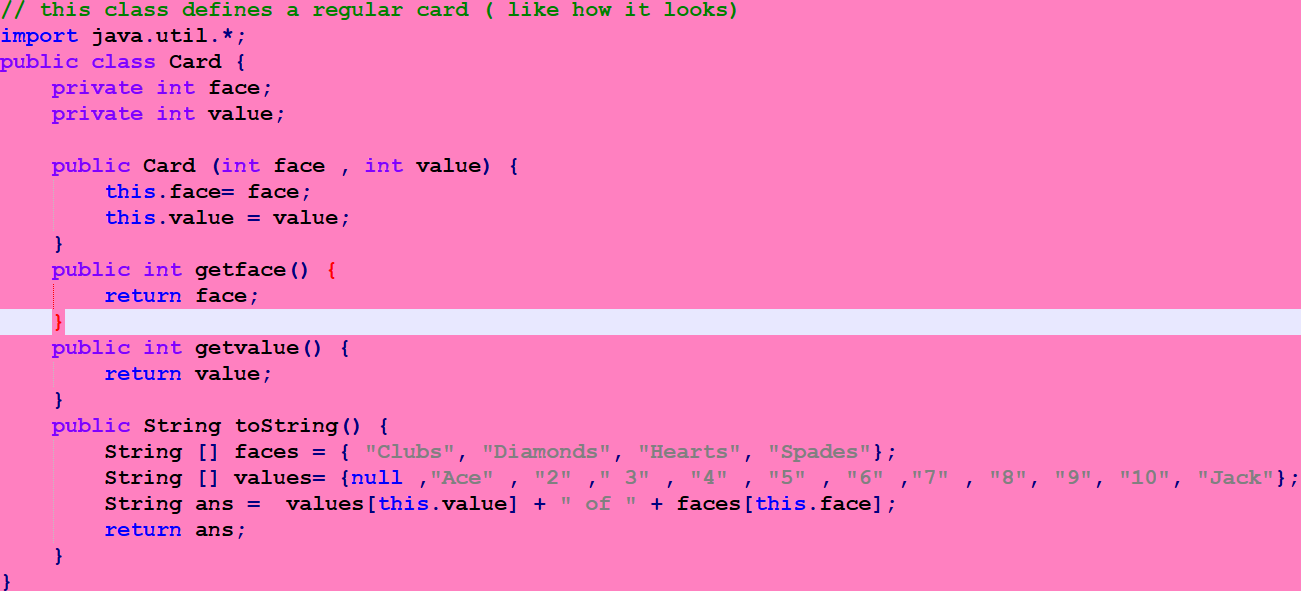
Ekasi-jack is straightforward and it has a well defined scope , the model has allowed us to breakdown the game into small manageable phases and complete each one before moving on.

There were two of us working on the game and this really helped us to know what to focus on and what responsibilities each of us have. For example in the deck class we knew we had to have a deck that represents card objects that has to be shuffled , that had to allow us to remove cards like we would in real life , the next thing we considered was player class etc.

Visuals

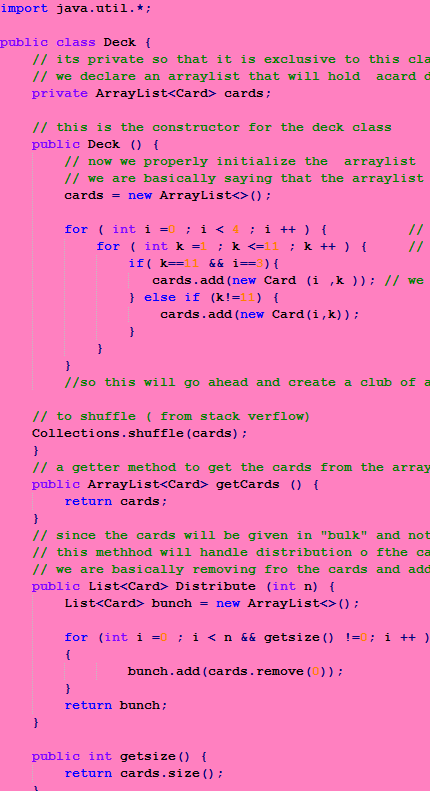
The card class

* The card class provides a representation of a card object
* The card object has two attributes , the face(suit) and the value(rank)



The deck class

* This class represents the deck of cards that the game will be using
* It allows for shuffling
* Distributing cards to the players]



The player class

* This class represents the ‘players’ component of the game
* It controls the management of the player names
* Allows the players to interact with each other

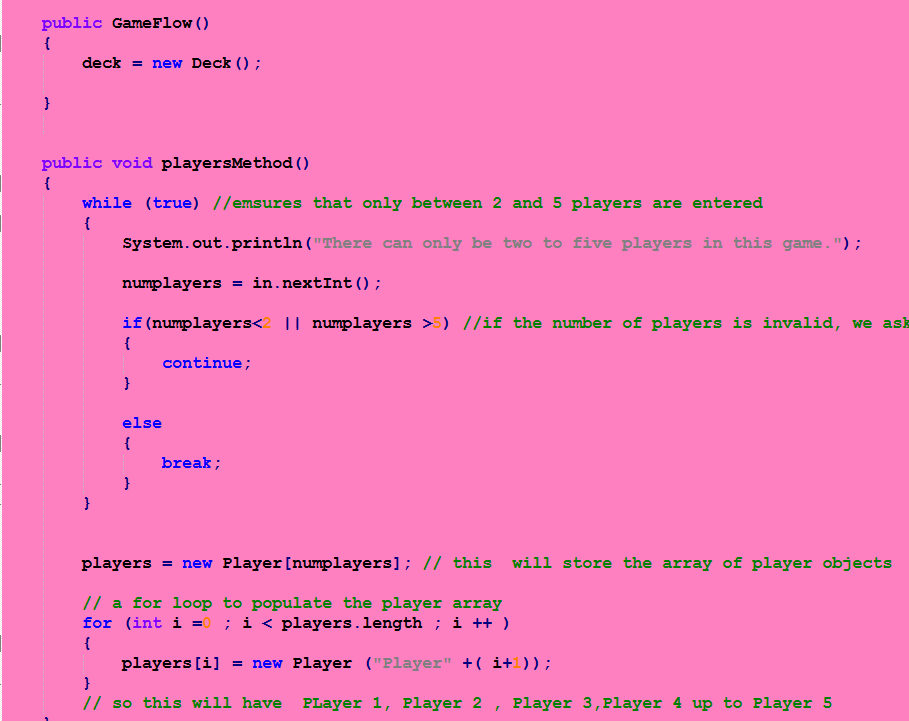


///add hand class snippet and update the uml diagram for changes u made

**@caitlin**

Game flow class

* This is the main logic of the game
* Takes user input to assess how many players are in the game this is important because it affects the way cards are distributed an dhow they are distributed
* We used an array to store player objects because it allows us to manage multiple instances of the class easily
* By doing that it made it easy to access and manipulate individual players



UML DIAGRAMS FOR EACH O FTHE CLASSES

|  |
| --- |
| Card |
| -face : int  - value: int |
| + Card(face: int , value : int):void +getface(): int  +getvalue():int  +toString():String |

|  |
| --- |
| Deck |
| -cards:ArrayList<Card> |
| + Deck():void  +getCards():ArrayList<Card>  + Distribute(n int) : List<Card>  +getsize():int  +toString:String |

|  |
| --- |
| Player |
| -name :String  -hand:ArrayList<Card> |
| +Player(name:String)  +getName():String  +getHand():ArrayList<Card> +addcards(card:Card)  +removecards(card Card)  +getSize():int  +matchingcards():void  +passcards(anotherplayer : Player)void  +Loser (): Boolean  +nocards():Boolean |



|  |
| --- |
| Hand |
| -hand :ArrayList<Card> -changingCard :Card  -name:String  -handSize:int  -player :Player |
| +Hand(hand:ArrayList<Card>)  +add (card Card):void  +Remove(card Card):void  +swap(card Card) : void  +isEmpty():Boolean  +getName():String |

|  |
| --- |
| GameFlow |
| -deck:Deck  -players:Player[]  -numplayers:int |
| +GameFlow()  +playersMethod():void  +getPlayers():Player[ ]  +handingCardMethod():void  +Play():void |

//main method

|  |
| --- |
| Program |
| +main(args:String [ ] ):void |