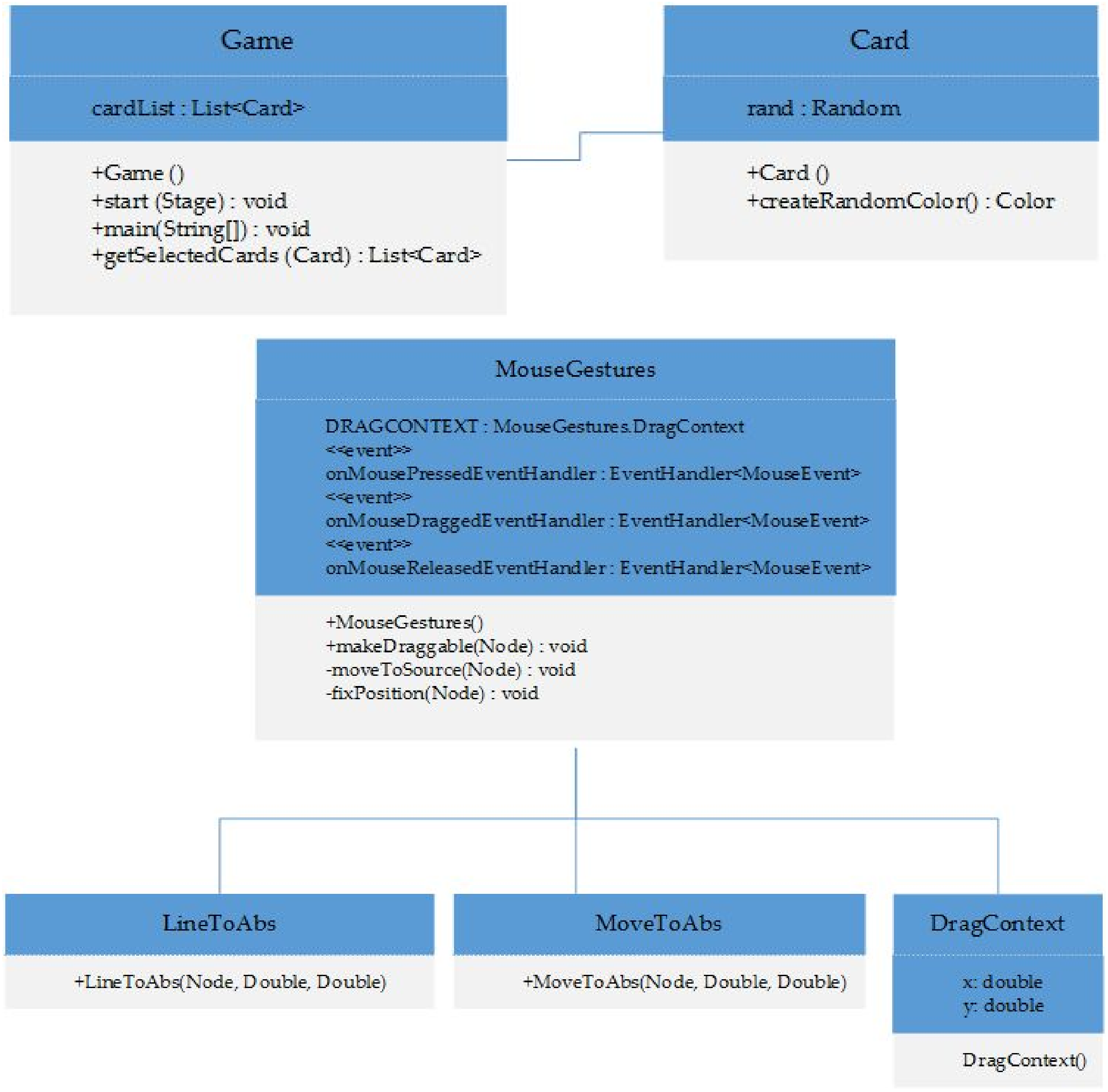
Tyler Odom
Nyenty Ayuk-Enow
Amier Chery
Coogan Koerts
Osman Khan

Technical Design


Game
cardList : List<Card>
+Game ()
+start (Stage) : void
+main(String[]) : void
+getSelectedCards (Card) : List<Card>
Card
rand : Random
+Card ()
+createRandomColor() : Color
MouseGestures
DRAGCONTEXT : MouseGestures.DragContext
<<event>>
onMousePressedEventHandler : EventHandler<MouseEvent>
<<event>>
onMouseDraggedEventHandler : EventHandler<MouseEvent>
<<event>>
onMouseReleasedEventHandler : EventHandler<MouseEvent>
+MouseGestures()
+makeDraggable(Node) : void
-moveToSource(Node) : void
-fixPosition(Node) : void
LineToAbs
+LineToAbs(Node, Double, Double)
MoveToAbs
+MoveToAbs(Node, Double, Double)
DragContext
x: double
y: double
DragContext()

Supporting Text Specification:

Game class

The “Game” class creates and initializes an Arraylist of Card objects called “cardList”. It also has a constructor to create a game object of itself.

The first method is the “start” method which creates the layout, 3 menu options, a space for the menu options. It also creates multiple card objects from the card class constructor. The “start” method then creates 10 cards and adds them to the “cardList” list. The method finished by setting up the rest of the GUI.

The second method is the main method which simply launches the application. The main method sets everything in motion

The third method is the “getSelectCards” method which takes a Card object as a parameter and returns a list of selected card objects. The methods starts by creating an arraylist of Card objects and creates an indexing int variable to get the location of the current card. The method then uses a for loop to find the designated card and returns that card.

Card class

The “Card” class is used to create card objects. The class creates a static Random variable in order to create colored cards to distinguish for use in the prototype.

The “Card” constructor creates a card object that is set to a standard height and width as well as giving the object as random color to help distinguish the cards from each other. The constructor calls the “createRandomColor” method which will be described next. The constructor finishes by filling the object with the randomly selected color.

The “createRandomColor” method will return a color and takes no parameters. It uses the “Random” variable to create values to get various values (up to 200) for red, green and blue and will return a corresponding color.

MouseGestures Class

The “MouseGestures” class is designed to help with the overall functionality of mouse movements and the placement of cards on the various areas of the board. The class creates a final variable called “dragContext” of the DragContext class that creates an x and y variable to pinpoint the exact location of the mouse on the application.

The first mouse event is “onMousePressedEventHandler” which gets the current location of the cursor on the application when the mouse button is pressed on the actual mouse itself.

The second mouse event is “onMouseDraggedEventHandler” which tracks the current location of the cursor on the application as the cursor is being dragged while the mouse button is being pressed. This event also selects a card that is being clicked on and enables that card to be dragged.

The third mouse event is “onMouseReleasedEvenetHandler” which tracks the current location of the mouse and will place a card in a particular location if the mouse button is released. If a card is not placed on an appropriate location then it will return to the place from which it came.

The first method is “makeDraggable” which has a void return type and takes a node object as a parameter. This method is used in all 3 mouse events in order to make the cards able to be moved.

The second method is “moveToSource” which has a void return type and a node as a parameter. All this method does is to help the event handlers to move the card back to its source location if the destination location is not appropriate. When a card is clicked, the initial location is recorded and if the target location is not suitable the card object is automatically returned to its source location.

The third method is “fixPosition” has a void return type and a node object as a parameter. All this method does is fix the current object (most likely a card object) in a specified location.

The first static class is “LineToAbs” which has a method of the same name which takes a node object and 2 double values and is used to make a line towards a location in order for another method to use later.

The second static class is “MoveToAbs” which also has a method of the same name which take a node object and 2 double values and moves and element along a created path.

Database Table Descriptions:

Our database for our Baker’s Dozen card game consists of three tables: PLAYER, RULES, and SYSTEM. Each of these three tables are unique to defining the overall structure of the game and are solely responsible for the overall architecture and design flow.

The first table, PLAYER, is a table that is responsible for holding all of the attributes of the PLAYER class, as well as any actions they can have and the relationships they have to the other tables. PLAYER has options such as MOVE(), CLEAR(), TRYAGAIN(), or any other moves that are directly controlled through input of the game’s user. Without the player table, there would be no interaction between the game and the player.

The second table, RULES, is the table that holds all of the rule attributes of the Baker’s Dozen card game. Attributes such as CARDSTACK(), and GAMEOVER() are set in place to make sure the PLAYER table does not input any commands that are against the rules of the game. Without the RULES table, there would be no rules to the game and the player would not have a clear objective on how to complete the game. The rule attributes also prevent the player from cheating or using any forbidden modifications to the game.

And lastly, the SYSTEM table is the one that connects directly to the system of the game. These are options such as CLOSE(), RESET(), and START(). The system table shows the program when to start, when to close, reset, etc. The system table is the most important table of the three, because it ties into the hierarchy of the game and is responsible for bringing the PLAYER, RULES, and SYSTEM tables together to provide the user with the Baker's Dozen card game experience.