[跳过内容](https://foothillcollege.instructure.com/courses/22864/assignments/717476" \l "content)

[控制面板](https://foothillcollege.instructure.com/)

* Leiyu Chen

帐户

[控制面板](https://foothillcollege.instructure.com/)

* 课程

[日历](https://foothillcollege.instructure.com/calendar)

[收件箱](https://foothillcollege.instructure.com/conversations)

* 历史记录

[Studio](https://foothillcollege.instructure.com/accounts/1/external_tools/10056?launch_type=global_navigation)

[帮助](http://help.instructure.com/)

[Student Support Portal](https://foothillcollege.instructure.com/courses/12004)

* [W23 CS F001B 01W Interm Software Design In Java](https://foothillcollege.instructure.com/courses/22864)
* [作业](https://foothillcollege.instructure.com/courses/22864/assignments)
* [⭐️ Assignment 4 - GUI Cards](https://foothillcollege.instructure.com/courses/22864/assignments/717476)

Immersive Reader

窗体顶端

Search

窗体底端

*Winter 2023*

* [主页](https://foothillcollege.instructure.com/courses/22864)
* [单元](https://foothillcollege.instructure.com/courses/22864/modules)
* [Simple Syllabus](https://foothillcollege.instructure.com/courses/22864/external_tools/20958)
* [评分](https://foothillcollege.instructure.com/courses/22864/grades)
* [讨论](https://foothillcollege.instructure.com/courses/22864/discussion_topics)
* [**作业**](https://foothillcollege.instructure.com/courses/22864/assignments)
* [Foothill Tutoring](https://foothillcollege.instructure.com/courses/22864/external_tools/12353)
* [Foothill Zoom](https://foothillcollege.instructure.com/courses/22864/external_tools/13208)

⭐️ Assignment 4 - GUI Cards

开始任务

* **截止** 2月8日 由 14:00 编辑

* **得分** 20

* **提交** 一个文本输入框或一份上传文件

* **文件类型** txt

* **可用** 1月25日 8:00 至 2月11日 14:00

**Due Wednesday @ 2:00 PM**

Goals

* In this assignment, we will be designing a program using JavaFX

 Instructions

Hand in only one program, please.

**A Scene of Card Images**

Understand the Classes and Problem

We wish to move our CardIdentity and Card classes from the realm of console apps to that of GUI apps.  We'll do this in stages.

1. Read and display Card pictures - Read .gif image files as Images, and attach them to an ImageView that we can display on a Pane.
2. Encapsulate the Card Images in a class GUICard - Once we debug imagery for cards, above, we can move it into its own class, GUICard.
3. Create a CardTable class - This Scene class will embody the Pane and Layout(s) needed for our application. This is where all the cards and controls will be placed.

The first phase (item 1) will allow you to debug the problem of reading the .gif files and displaying them on a Pane without any excess logic or class complexity.  The second phase (items 2 and 3) will let you turn what you did in the first phase and what you read in the chapter into a multi-class project.

The three bullets will be done in two phases.  The main, public class of each program must be named Foothill so I can easily run it, and other classes must not be public.  You do not submit any runs - I will run your programs.

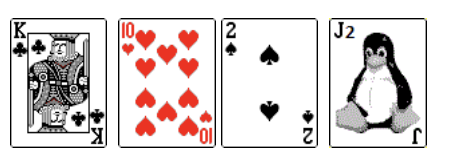
Download the Card .gifs

The GNU Free Software Foundation provides code and images for playing-cards that are public domain.  I prepared a **.zip** file which contains a **.gif f**or every card with some minor name changes.  Download it here:

[Card Images Download](https://foothillcollege.instructure.com/courses/22864/files/6864453?wrap=1).

After you unzip them, you will have a single folder called images that contains all the .gif files.  Move that folder to your [java workspace]/[project name] directory.  If your project is named Assignment\_4, and your Eclipse workspace is named workspace, then move images folder to workspace/Assignment\_4.  Since your program considers Assignment\_4 to be the root directory, all .gif files can be referenced from your program using names like "images/3S.gif" for, say, 3 of spades.

In addition to the 52 standard cards, there are four jokers and a card-back image which can be used to display dealer or other player cards that you don't want the user to see yet.



(The card on the far right is ***joker 2***, not a jack.)

Phase 1: Reading and Displaying the .gif Files

In section 14.9 of our book, you learned how to instantiate an Image object to represent any .gif, .jpg or other image file on your disk and then place that Image on a ImageView.  In Phase 1, we simply create an array of 57 ImageViews, attach the 57 .gif files to them, and display the labels, unstructured, in a single Pane.  Here is a possible start() that you can use as a starting point. You don't have to use my main() but yours should be no longer or more complicated.  Look at your images folder to see the names of each .gif file:  you have to be able to construct their names in a loop (-20 points if you list all 52 cards named literally because this is array logic).  Where is the card-back image stored?  Find it in the images folder.  \*\*If you have issues, try adding a "file:" in front, so it became "file:images/" and that should help.\*\*

import javafx.application.Application;  
import javafx.scene.Scene;  
import javafx.scene.layout.\*;  
import javafx.geometry.Insets;  
import javafx.stage.Stage;  
import javafx.scene.image.\*;  
import javafx.scene.control.Label;  
     
public class Foothill extends Application  
{  
  // static for the 57 images and their corresponding labels  
   // normally we would not have a separate label for each card, but  
   // if we want to display all at once using labels, we need to.  
     
  static final int NUM\_CARD\_IMAGES = 57; // 52 + 4 jokers + 1 back-of-card image  
   static Pane pane = new FlowPane();  
   static Image[] image = new Image[NUM\_CARD\_IMAGES];  
   static ImageView[] views = new ImageView[NUM\_CARD\_IMAGES];  
  
   // for assisting with conversions:  
   static String cardlValsConvertAssist = "23456789TJQKAX";  
   static String suitValsConvertAssist = "CDHS";  
     
 public static void main(String[] args)   
 {  
   launch(args);  
 }  
  
   public void start(Stage primaryStage)  
   {  
   // prepare the image array  
      loadCardImages();  
        
      // Create the scene and place it in the stage  
   Scene scene = new Scene(pane, 800, 600);  
      primaryStage.setTitle("Card Room");  
      primaryStage.setScene(scene);  
        
      // show everything to the user  
      primaryStage.show();  
   }  
  
   static void loadCardImages()  
   {  
      String imageFileName;  
      int intSuit, intVal;  
  
      for (intSuit = 0; intSuit < 4; intSuit++)  
         for (intVal = 0; intVal < 14; intVal++ )  
         {  
      
            // card image files stored in Foothill/images folder with names like  
            // "AC.gif", "3H.gif","XD.gif", etc.  
  
// This is all you need to figure out for Phase 1.    
  
         }  
  
      imageFileName = "images/BK.gif";  
      image[image.length - 1] = new Image(imageFileName);  
      views[image.length - 1] = new ImageView(image[image.length - 1]);  
      pane.getChildren().add(views[image.length - 1]);  
        
  
   }  
     
   // turns 0 - 13 into 'A', '2', '3', ... 'Q', 'K', 'X'  
   static char turnIntIntoCardValueChar(int k)  
   {  
     
      if ( k < 0 || k > 13)  
         return '?';   
      return cardlValsConvertAssist.charAt(k);  
   }  
     
   // turns 0 - 3 into 'C', 'D', 'H', 'S'  
   static char turnIntIntoCardSuitChar(int k)  
   {  
      if ( k < 0 || k > 3)  
         return '?';   
      return suitValsConvertAssist.charAt(k);  
   }  
    
}



This is for your testing purposes only, do NOT hand in the main for this phase.

Phase 2: Encapsulating Layout and Images into the GUICard Classes

We will now bring back our Card class from Assignment #2, to work with a GUI presentation.

The second part creates a separate **GUICard** class that manages the reading and building of the card **images**. As a result, much of the machinery and statics that we debugged in the first phase of the main, **Foothill** class, will be moved into the new class.

GUICard Class

This class is the benefactor of most of the GUI machinery we tested in ***Phase 1***. It will read the image files and store them in the Image and ImageView arrays. Rather than a 1-D array of ***Phase 1*** (if you followed my earlier outline), this will be a 2-D array to facilitate addressing the value and suit of a **Card** in order get its **Image**. While simple in principle (just read the **Images** and store them in an array for client use), the details are subtle. We have to be able to convert from **chars** and **suits** to **ints**, and back again, in order to find the **Image** for any given **Card** object. The overview of the class data and methods, shown below, will suggest the right approach and should take the mystery out of this class.

Static Members

   private static Image[][] imageCards = new Image[14][4];   // 14 = A thru K (+ joker)  
   private static ImageView[][] imageCardViews = new ImageView[14][4];  
   private static Image imageBack;  
   private static ImageView imageCardBack;  
   private static boolean imagesLoaded = false;

The 52 (+ 4 jokers optional) **Images** will be read and stored into the**imageCards[][]** array.  The ***card-back*** image in the **imageBack** member.  Both of those then need to go to the **ImageView**arrays. None of these data need to be stored more than once, so this is a class without instance data.  This class is used is to produce an image when the client needs one.

Helper Static Arrays

private static String cardlValsConvertAssist = "23456789TJQKAX";

private static String suitValsConvertAssist = "CDHS";

private static Card.Suit suitConvertAssist[] =

{

Card.Suit.clubs,

Card.Suit.diamonds,

Card.Suit.hearts,

Card.Suit.spades

};

The first two Strings are the ones we used in phase 1, but moved into this class.  The third is a new array used to convert ints into actual enum Suits, as you'll see.

Static Methods

To begin, we need a method that generates the image array from files:

* **static void loadCardImages()** - the code for this was fundamentally done in ***Phase 1***.  The difference here is that we are storing the**Image**s in a 2-D array.  So you have to use a nested ***for-loop***, suits and values, to generate the 2-D index for each **Image** in the array. Another suggestion:  I'd like you to not require the client to call this method.  Think about where you would need to call it and how can you avoid having the method reload the images after it has already loaded them once.  The hint is in the **static boolean imagesLoaded = false;**, above.  ***Hint:****Call this method any time you might need an****Image****, but make sure that it loads the entire array the first time it is called, and does nothing any later time.*

The use of **loadCardImages()** should be clearer after you take a look at the primary public method offered by this class:

* **static public Image getImage(Card card)**- This method takes a **Card** object from the client, and returns the **Image** for that card.  It would be used when the client needs to instantiate or change an **imageView**.  Here is what the **getImage()** method definition might look like:
* static public Image getImage(Card card)
* {
* loadCardImages(); // will not load twice, so no worries.
* return imageCards[valueAsInt(card)][suitAsInt(card)];

}

I just realized I wrote the entire method for you.  Well, there are three method calls packed into this definition, and those, I did not write.  Now you see where **loadCardImages()** is called, and you see why we don't want it to do anything except **return** if the cards are already loaded.  There is another method that returns the ***card-back*** image:

* **static public Image getBackCardImage()** - this one is even simpler than **getImage()**.

The above methods three comprise the essential part of the **GUICard** class.  Everything else is support for these three, so you can work off my implied suggestions, or you can build the class from scratch as you wish.  Just make sure you are efficient.

These last four are in support of the ones above

* **static char turnIntIntoCardValueChar(int k)**
* **static char turnIntIntoCardSuitChar(int k)**
* **static Card.Suit turnIntIntoSuit(int k)**
* **static int valueAsInt(Card card)**
* **static int suitAsInt(Card card)**

Main Foothill Client

Static Methods

This class now only has the **start()** and one other static helper method:

* **Card generateRandomCard()** - This method uses Java's Math.random() utility as a helper, and from it, generates a random card.  We don't care if it generates the same card more than once - that's fine.  There are some public helpers, now in the GUICard class, that will also be useful -- invoke them as needed to make this a short method:  **turnIntIntoSuit()** and **turnIntIntoCardValueChar()**.

start() for Phase 2

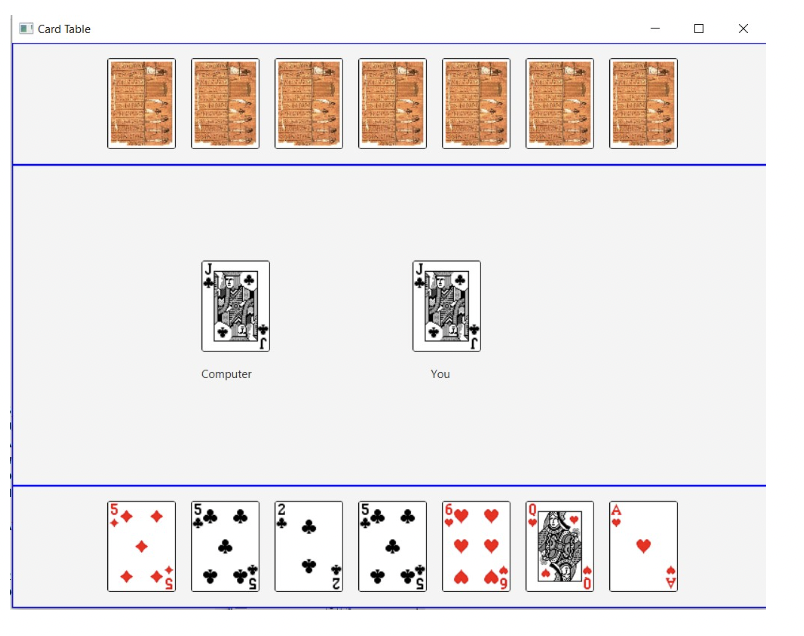
Use two **HBox**es, one for each hand (player-bottom and computer-top).  Then use a **FlowPane**, as a middle "playing" area.  The client (below) will generate the human's cards at random and those will be *visible* in the bottom **HBox**, while the computer's cards will be chosen (again, by the client) to be all ***back-of-card*** images in the top **HBox**.  The middle **FlowPane**will display cards that are "played" by the computer and human during the conflict.  Let's assume that each player plays one card per round, so for a 2-person game (computer + human) there will be exactly two cards played in the central region*per round of battle*.  My client chose a ***jack***for the two central cards, just so we would have something to see in the playing region.  No game is being played in this assignment, so the cards to be displayed in the center are immaterial.

The main, **Foothill**, class needs to define the specific **Image and ImageView**arrays that will go into each of **CardTable**'s **panes**. You will need **NUM\_CARDS\_PER\_HAND** **Image and ImageView**arrays for the player and the computer (each), even though the computer only uses one **Image** (***back-of-card***).  **Hint**: create the HBox with a spacing of (15), HBox(15); then use paneName.setPadding(new Insets(15, 100, 15, 100)); to get the right spacing for the cards.

We also want two **Image and ImageView**arrays for the central **pane**(these are the two cards played by computer and human, each turn). But, we also need to some **text** L**abels**below each of the two center images to we know who played which card (**"Computer"** or **"You").** Since we want the text directly below the image, one way to do this is to use the FlowPane with enough spacing to make it wrap around to the correct location below that describe the images. **Hint**: to adjust, use

     FlowPane playedPane = new FlowPane(150,15);  
     playedPane.setPadding(new Insets(100, 200, 100, 200));  
      playedPane.setAlignment(Pos.CENTER);  
      playedPane.setStyle("-fx-border-color: blue");

The net result should be cards we can see (our hand) in the lower **pane**, cards that we can't see -- except for the card backs -- in the upper **pane**(the computer's hand) and a central playing region which would represent two cards, one each played by the user and the computer. These two cards depend on what game we are playing, the rules, and the goal.  Based on these two cards played, either we or the computer win that round and then we go on to the next round. For this assignment, we don't worry about strategy or rules or winning -- we just want to see two cards in the central **pane**so we know they are correctly positioned for later program development.  Here's a partial picture of a basic solution:



Your job is to simply produce this output using the classes and methods suggested.  Here is an idea for a **main()** that you can use to get started:

import javafx.application.Application;  
import javafx.scene.Scene;  
import javafx.scene.layout.\*;  
import javafx.geometry.Insets;  
import javafx.stage.Stage;  
import javafx.scene.image.\*;  
import javafx.scene.control.Label;  
  
public class Foothill extends Application   
{  
  final int NUM\_CARDS\_PER\_HAND = 7;  
   final int NUM\_PLAYERS = 2;  
   Image[] humanImages = new Image[NUM\_CARDS\_PER\_HAND];  
   ImageView[] humanViews = new ImageView[NUM\_CARDS\_PER\_HAND];  
   Image[] computerImages = new Image[NUM\_CARDS\_PER\_HAND];  
   ImageView[] computerViews = new ImageView[NUM\_CARDS\_PER\_HAND];  
   Image[] playedImages = new Image[NUM\_CARDS\_PER\_HAND];  
   ImageView[] playedViews = new ImageView[NUM\_CARDS\_PER\_HAND];  
   Label[] playLabelText = new Label[NUM\_PLAYERS];  
  
 public static void main(String[] args)   
 {  
   launch(args);  
 }  
  
   public void start(Stage primaryStage)  
   {  
      // Create the scene and place it in the stage  
      BorderPane pane = new BorderPane();      
      Scene scene = new Scene(pane, 800, 600);  
      primaryStage.setTitle("Card Table");  
      primaryStage.setScene(scene);  
  
      // CREATE IMAGES ----------------------------------------------------

code goes here ...

// ADD IMAGEVIEWS TO PANES -----------------------------------------

code goes here ...

// and two random cards in the play region (simulating a computer/hum ply)

code goes here ...

    pane.setTop(computerPane);  
     pane.setCenter(playedPane);   
     pane.setBottom(humanPane);   
  
     // show everything to the user  
     primaryStage.show();      
 }

**\*\*\*Hand in only Phase 2 and be sure to include the CardIdentity and Card classes with your submission\*\*\***

 Submission and Grading

This assignment will be submitted via file upload in Canvas.

I will grade the submission according to the rubric posted here.

To view the rubric, select the gear icon. If you're using a mobile device, the rubrics can be found in the course navigation menu.

The grading rubric can be found right below this assignment.

 Helpful Canvas Guides

The following Canvas Guides will help you successfully submit this assignment and review your feedback:

[STEM Center for tutoring help](https://foothill.edu/stemcenter/)

[How do I upload a file as an assignment submission in Canvas?  (Links to an external site.)](https://community.canvaslms.com/t5/Student-Guide/How-do-I-upload-a-file-as-an-assignment-submission-in-Canvas/ta-p/274)

[How do I view grades in a current course?](https://community.canvaslms.com/docs/DOC-9540-4212724430)

评分标准说明

**Assignment (1)**

| Assignment (1) | |
| --- | --- |
| **标准** | **等级** |
| 此标准已链接至学习结果Followed the spec  Program fits the description completely, all classes and methods have correct functionality |  |
| 此标准已链接至学习结果Logic and Syntax  Program has no issues with logic or syntax errors |  |
| 此标准已链接至学习结果Output  Output is according to spec |  |
| 此标准已链接至学习结果Style issues  All style issues are correct |  |
|  | |

[上一页](https://foothillcollege.instructure.com/courses/22864/modules/items/2018202)[下一页](https://foothillcollege.instructure.com/courses/22864/modules/items/2018205)