HD Project – Alexander McCutcheon

In relation to the fact whenever I assisted students with programming throughout the semester, it was in person, and out of class hours, and not by using the discussion forum, I opted to do one of the other tasks instead. As I don't really have any Research goals, and as I have used many of java's libraries extensively before, a tutorial on their use seemed the best fit of the available options. I made it a video forgetting that text-based tutorials are thing, hence this is the transcript I used to record the video, and not a guide in and of itself. Each subsection under transcript is a progression in the code within the video, each stage of the code is also attached to the MyLO submission.

Link

The Video can be found here: https://youtu.be/QjeMlgVSiAA
And is attached on MyLO
The code samples are currently only available on MyLO

Transcript

Intro base code and imports

Howdy and welcome to this short guide on how to create a basic GUI for your program
This guide will be focused on creating the rock paper scissors game displaying now.
If you would like to get started with creating GUI programs, hopefully this guide can help you.

Starter code

This guide will be making use of a few standard libraries
A brief explanation for each is on screen now, so feel free to look at those if that interests you
Where I will be starting in this guide, is from a simple base class, with a driver main, and a constructor.
I will be creating the GUI with swing components, and using listeners to get user input
~~From now on there will be temporary comments provided to indicate where any changes occur,~~
~~The code is available for download in the description if that interests you~~

Creating a frame

One of the most important parts of a GUI program, is the top-level container
In swing, there are three such containers, JFrames, JDialogs, and JWindows
They all have their uses, but for the most part you will want to be using JFrames
So, go ahead and create a new JFrame, a common name to use when only using one frame is

"f"
but you can call it whatever you want if that doesn't suit your fancy
If you run the code now, you will not see anything happen
This is because JFrames are initially invisible
so you need to set it to visible by calling frame.setVisible(true)
This is enough to create the most basic Frame in swing
But as you can see, it is very small, and is empty, not very ideal.

~~Extending a frame (CUT FROM VIDEO)~~
~~Another common alternative for working with frames, is extend the JFrame class itself~~
~~This is usually only done if you need to Override something such as its paint method or add additional functionality~~
~~But it isn't a good idea to override a Frames paint, and this is a small program that doesn't add anything to the JFrame class~~
~~So, I will be going with the first approach of creating a frame object.~~

Creating a panel

It is possible to set the Size of a frame simply by calling frame.setSize()
This is fine in some cases, but as you add components to the screen, or if you add something such as a menu
The size will no longer be correct
It is hence better to let the Frames layout choose how big to make the frame, by telling it how much space each component should take up
The command that makes this happen, is frame.pack() but to use it we need some components to pack around
For now, the only component I will be adding is a JPanel, which we will be displaying everything on.
I have instantiated a new JPanel, set its preferred size so the layout manager knows how big it should be,
added it to the frame so we can see it
then packed the frame around it.
If we run it now, it will have a similar effect to frame.setSize(), but it is much more ready for expansion
As you can tell, we still haven't set up anything to display

Overriding Panels paint

An easy way to draw to the screen, is to create an anonymous inner class of the JPanel, and override its paintComponent method

You can do this by adding curly brackets after you instantiate it, this implicitly creates an object of a new nameless class that extends JPanel

In these curly brackets, put a void returning method called paintComponent, with @Override for good measure

A parameter of this method is graphics g, we can use this graphics object to draw our panel.

I start drawing by calling super.paintComponent, which calls the original paintComponent method that we have just overridden,

If you are filling in your entire panel as I am, this isn't needed, but I will show later why it is a good idea to include this regardless

After that, I set the colour to black and fill in the screen with a 300,300 square for a background

So, when I run the program now, you can see a black square, and if I resize the window, you can see the square sticks to the upper right.

You can of course change yours to be a different colour, or different size whatever you want

Common frame functions

JFrames are fairly customizable, you can remove the frame borders, give it a custom icon and all sorts of things

The Oracle Javadoc's for JFrame is probably the best place to go if you want to customize it in a specific way

For my rock paper scissors game, I want to give it a suitable title,

make it so you can't resize it effectively locking away the non-black area,

and make it so other windows can't be on top of it

These last two are often fun for small test applications like this, but do think about your end user when deciding on these things

When I run my program now, I see my title in the title bar at the top, I can no longer resize it, and if I try to bring another window up over it, I can't, wonderful

Add buttons to the screen

A black box isn't very interesting, and it would be nice to have a way to get input from the user

A quick and easy solution to this would be to use JButtons for each possible input

I have created three such buttons here, the text given as a parameter will be written on the button

This is a quick and easy way of telling the user which button which is
In my constructor, I am calling a new method called setUpButtons, this is to avoid cluttering it too much,
for each button I set its bounds, which is its location on the screen, followed by its width and height
then I add it to the panel
I've set them up so they should be lined up along the bottom of the screen, taking up the entire base
But when I run it currently, the buttons appear at the top of the screen and are not the size I set them
This is because currently the panels layout manager is deciding where components should be

removes layout manager

We could import and set the layout manager to something else that suits our needs
But a more simple solution is just to remove the layout manager altogether
you can do this by setting the layout manager of the panel to null
now when I run the game, the buttons appear where, and what size, I set them to

Add listeners to buttons

Currently the buttons don't actually do anything, you can click them, but nothing else happens as a result
In order to get the event when a button is pressed, you need to use action listeners
To use action listeners, you need to do three things
1. implement ActionListener so your program can receive ActionEvents
2. Add the actionPerformed method which will be called when an event occurs
3. Add an action listener to each button you want to be able to press
Now, when any of these three buttons are pressed actionPerformed will be called
I've temporarily placed code to quit the program here,
so, when I run the program now, all three buttons will quit the program

Adds rock paper scissors functionality

Now that we've got user input up and running, it’s time to implement the basics of rock paper scissors
I've created an enum called Choice that contains what the user and computer can pick
I've also created variables to store a choice for each the player and computer
and variable called rand, that I will be using to generate the computers choice randomly

For now, I've simply added text to the screen displaying the computer and user's choice
Finally let's look at actionPerformed
AS I mentioned before, every button calls this method,
which means we need some way to establish which button was pressed if we want to do anything useful
One of the easiest ways to do this, is e.getSource,
the ActionEvent parameter that is a part of actionPerformed stores various information about the event
including which component called it
We can use this with a chain of if statements for each button to implement our decision
I've got it set to set the user's choice to the button type, then end the turn,
I have end turn in each of the if statements rather than at the end in case we add more buttons
For now, this end turn method just gives the computer a random choice
This should be enough to have basic rock paper scissors functionality display
However, if I run it, you can see that pressing the buttons still appears to do nothing
This is because components are only painted when they need to be by default, such as when the frame initially opens
when you resize a frame, or if you minimize then restore the frame.
So, if I minimize then restore, you can see that changes are occurring
but this is no good for our game to leave it like this

Add repaint so you can see changes and importance of super

Currently, we are relying on the system to know when to repaint the screen,
but even though we know that the screen needs to be repainted after a button press, the system doesn't know that, so it doesn't get repainted
So, we need to tell the panel to repaint ourselves,
to do that we call it repaint method at the end of actionPerformed
So, when I run the program now, we can see that each button press actually has effect, and the screen is repainted appropriately
now that repaint is implemented, the importance of super can be demonstrated
If we comment out super with the code as it is now, nothing will happen
But let's say instead of white on black, we left the background how it was, and draw black on it
we can achieve this effect by commenting out these two lines
So, when we run it, all is as we expect, until we press a button
now the old text is remaining even after the repaint, this can cause a lot of issues as you can

tell
So, it is a good idea to call super to avoid doing this by accident, especially when starting out
I will now restore the code to how it was before proceeding

Calculate winner and display

It's nice that we can see what each person chooses for their turn, but knowing who has won a game would be nice
For extra visual effect, it would also be nice if the colour of the text changed based on who won
To achieve this, I am adding two new variables, the fontColor, and verdict, which will store the colour, and display text respectively
To draw these to the screen is simple
All I am doing is setting the colour to this new variable
then displaying the verdict right above the buttons and somewhat centered based on its length horizontally
This is quick attempt at centering and will look bad with longer strings, but is good enough for this use
I have put the actual win logic in the endTurn method as that is the logical place to do so
I start by checking if the computer and player have chosen the same Choice, in which case it is a tie, and should be displayed in white
Next, I check if the player has won by or'ing the three possible win combinations, if so, the player wins, and should be displayed in yellow
Otherwise by process of elimination, the computer must have won, and should be displayed in red
So, when I run the game now, I have a nice display at the bottom saying who has won in a relevant colour

Win counter

Another nice functional addition before we return to graphics, would be a win loss counter
To implement this, you can simply add a variable for each the players wins, and the computers wins
Then display these variables to the screen, I have chosen to the left and right of the verdict in the relevant colours
now all that’s left to do is, where you decided who won, increment the relevant winner's score, doing nothing for ties of course

Adds images to the screen

Now we've got a functionally complete version of rock paper scissors, it’s time to make it a bit prettier

One of the most obvious improvements we can make, is adding images for the users and the computers hand

this part requires that you have in your project folder, 6 images, one for each hand state

To store these 6 images, I am adding 2 new arrays, userImages, and computerImages which will contain BufferedImages

To draw an image, it is much the same as drawString, you provide the image object you want to draw, its coordinates, and dimensions

there is however one more required parameter, the ImageObserver, which although we don't need to use, we do still need to provide it

The easiest way to deal with this is to simply give it null, so we don't have to deal with image observers.

For now, I just want to get anything displaying, so I will simply be drawing the first image in the array for now

In the constructor, I have added yet another method call, this time to setUpImages, as it is also a lengthy process

In setUpImages you will notice there is a try catch block, even though this is a small program, and dealing with error handling isn't

really that useful to us, because we are working with the IO libraries, we need to use them.

I start by creating a length 3 array for the player images, then setting each one in the order rock paper scissors.

The imageIO.read method requires a file object not a string, so we need to create a new File object,

The string we give this file, is the filename of each rock paper or scissors image,

Repeat this for the computer's array, and all should be ready

If you run the program now, you should see the first images of each array displaying,

But they don’t yet change with the choices being made.

Uses ordinal to select correct image

Selecting the correct image might seem like it will take a few lines of code at first

but I have been very deliberate in my ordering thus far,

and you will notice that the enum's order is ROCK PAPER SCISSORS, and I put the images in the array in the same order ROCK PAPER SCISSORS

Enums have a nice inbuilt method, .ordinal() that returns the index of an enum constant as if

it were in an array
So, Choice.ROCK's ordinal is 0, Choice.PAPER's ordinal is 1, and Choice.SCISSORS ordinal is 2
This means we can just give the relevant choices ordinal as an index when drawing
which is exactly what I have done here
This means that when we play the game now choices are represented properly

Remove debug text, add a menu

As we now have the images to show what choices are made, the white text at the top of the screen is no longer needed
so, I have removed it
For the most part our game is done, but it would be nice to have a way to reset the score without quitting and reloading
One way to do this would just be to add another button to the screen, but this is also a good opportunity to add a menu bar to the frame
so, I will do that instead
A menu bar consists of three main parts, the bar itself, the menus that are directly on the bar, and buttons that appear under each
In my case, I'd like to add a File menu, with a reset and quit button
So, I have created a JMenubar, a File JMenu, a reset JMenuItem, and a quit JMenuItem
Much like JButtons, the text in the parameter is what will be written on each menu or menuItem
once again, I am putting the setup code for this menu in its own method
To actually make the menu appear on the frame, you need to set it as the frames JMenuBar
Then you need to add all menus you want on the bar onto it
and add all menuitems to appear under that menu, to the menu
This is all you need to do to set up a menu
So, when I run the program now, a menu bar with file on it is now present, and reset and quit buttons are under that
But much like when we added the buttons, these don’t currently do anything

Add listeners to the menu

If you want to detect when these menuitems are pressed, you need to add actionlisters to them just as you did the buttons
I have added the lines to do so to the setUpMenu method
Now when you press the menuItems, actionPerformed will be called

So, it would be a good idea to extend actionPerformed to act for these items
Quit is simple, it is the same line I used for testing if buttons worked
But for reset, I am setting all variables that can change, back to what they are initially
This is a quick and easy way to reset a small game like this.
It would probably be a good idea to put this process into its own method
But to be consistent, and as it's not too bad I am going to leave it how it is
So now, when I run the game, I can play for a while, then hit reset to put everything back to normal
then I can use the quit button under file instead of the X to quit if I wanted to
This marks the completion of this program

Postscript

If you found this little guide of use, well that’s all that really matters isn't it
thanks for watching