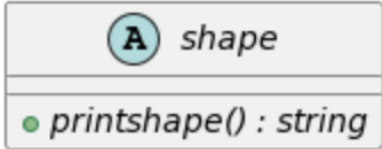
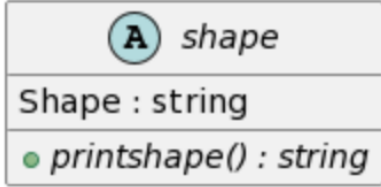
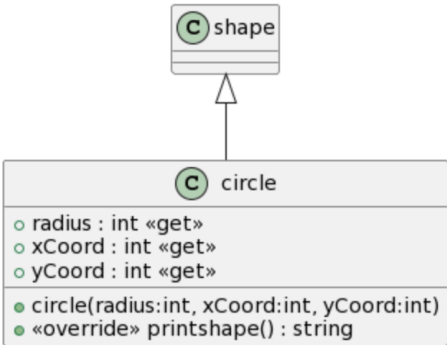
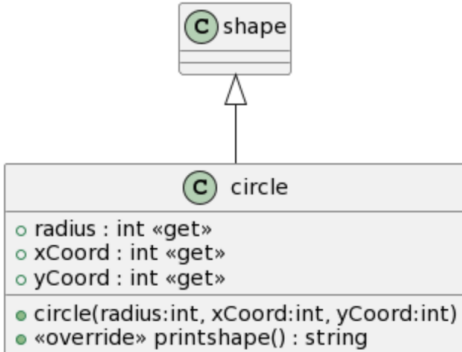
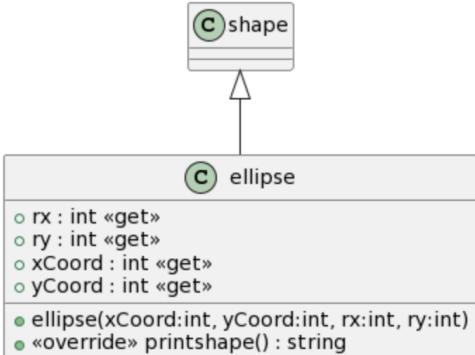
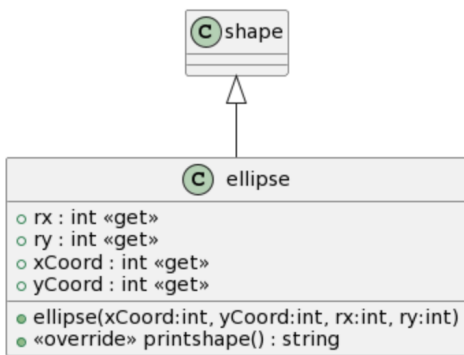
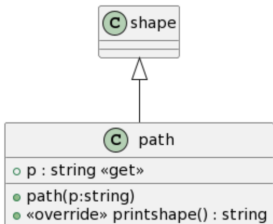
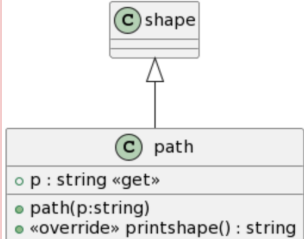
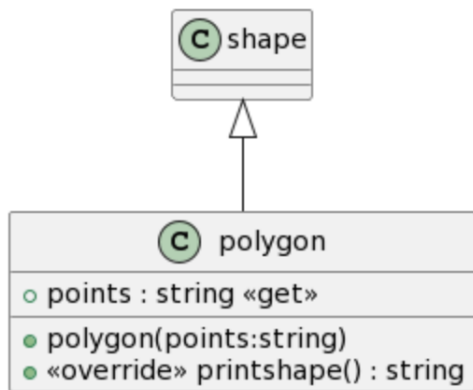


In assignment four I used the command design pattern. This design pattern was harder to wrap my head around than the memento design pattern that I used in assignment three. The reason the command pattern was more tricky for me was because of how many more classes there were compared to memento. With the command pattern for example to add the clear canvas functionality I needed to create a whole new command class for the clear canvas function and write methods in my canvas class to be called by the user class. Whereas with memento that same function could be added with a simple method in my caretaker class. Another unique feature of command patterns is that you have to make a new canvas and a new user in your main method for it to properly work. This was another thing that I needed to learn more about. Finally someone told me that it was easier to think of the canvas as a television and the user as a remote and then it finally made sense. The memento pattern really helped me feel more comfortable with classes and objects. Both patterns really helped my main program file to stay very clean and simple. With both of them I used a list or a stack to save what was in the canvas before a clear() was done. They are very similar if you think about it, the command pattern is just a list of commands that are stored and the memento pattern is a list of mementos. What I've found the most interesting is how similar some of the classes are from these two projects. For example my caretaker class in my assignment three is very similar to my canvas class in my assignment four. The caretaker and canvas classes from these two assignments are where the majority of my functions are for these programs. These classes are where I can print to the console, save the shapes to an svg file, and undo/redo. I think the memento design pattern worked best for this assignment because it is the easiest out of the two to add more functionality to. In memento if I wanted to add another feature I could simply just write a method in my caretaker class and call it when I need to in my main program. I also think the memento design pattern is just easier to make changes to and edit on. When I was editing and changing things in the command design pattern, soon enough you change something in one file and next thing you know you have twenty three errors in two other files. It was very overwhelming. Overall this has been a very interesting experience to see how many ways the same thing could be designed. Learning about these design patterns has helped me to think about the way I code and think about how I could have cleaner and easier to read code. I have learned a lot about these two design patterns and I am curious to learn more design patterns.

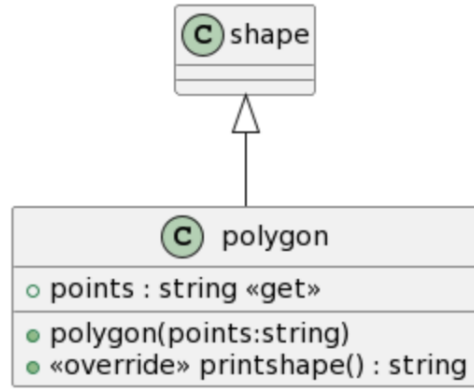
UML Diagrams:

Memento UML	Command UML
<p>Shape class</p>  <pre> classDiagram class shape { +A +printshape() string } </pre>	<p>Shape class</p>  <pre> classDiagram class shape { +A +Shape : string +printshape() string } </pre>
<p>Circle class</p>  <pre> classDiagram class shape { +C } class circle { +C +radius : int +xCoord : int +yCoord : int +circle(radius:int, xCoord:int, yCoord:int) +<override> printshape() string } shape < -- circle </pre>	<p>Circle class</p>  <pre> classDiagram class shape { +C } class circle { +C +radius : int +xCoord : int +yCoord : int +circle(radius:int, xCoord:int, yCoord:int) +<override> printshape() string } shape < -- circle </pre>
<p>Ellipse class</p>  <pre> classDiagram class shape { +C } class ellipse { +C +rx : int +ry : int +xCoord : int +yCoord : int +ellipse(xCoord:int, yCoord:int, rx:int, ry:int) +<override> printshape() string } shape < -- ellipse </pre>	<p>Ellipse class</p>  <pre> classDiagram class shape { +C } class ellipse { +C +rx : int +ry : int +xCoord : int +yCoord : int +ellipse(xCoord:int, yCoord:int, rx:int, ry:int) +<override> printshape() string } shape < -- ellipse </pre>
<p>Path class</p>  <pre> classDiagram class shape { +C } class path { +C +p : string +path(p:string) +<override> printshape() string } shape < -- path </pre>	<p>Path class</p>  <pre> classDiagram class shape { +C } class path { +C +p : string +path(p:string) +<override> printshape() string } shape < -- path </pre>

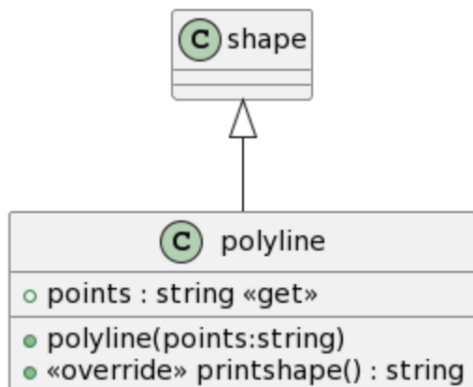
Polygon class



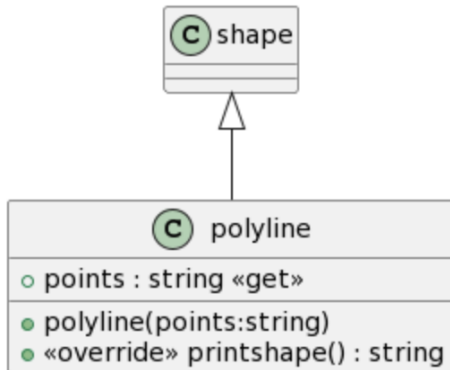
Polygon class



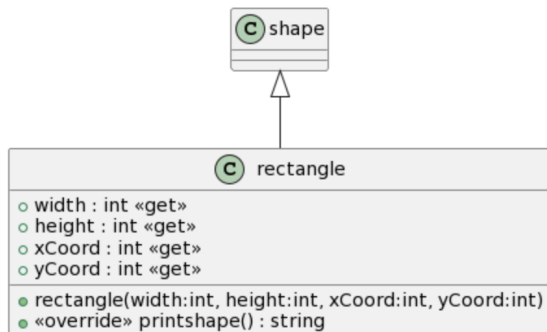
Polyline class



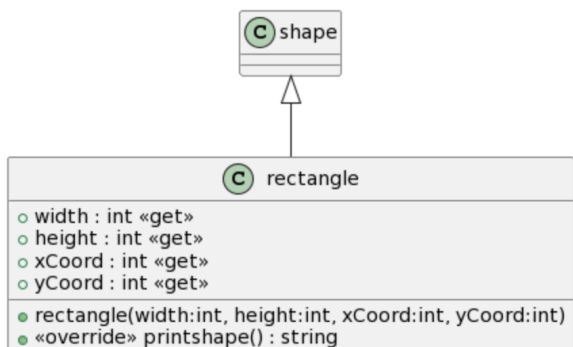
Polyline class



Rectangle class

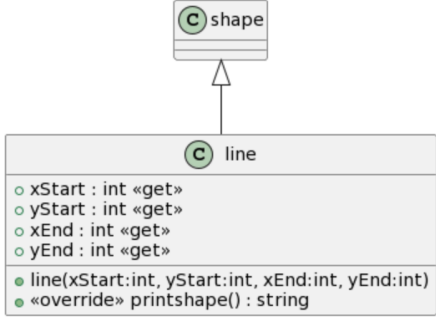
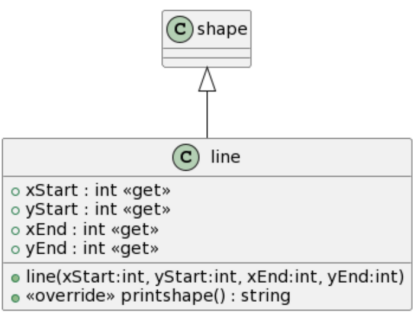
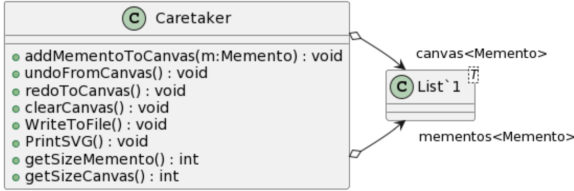
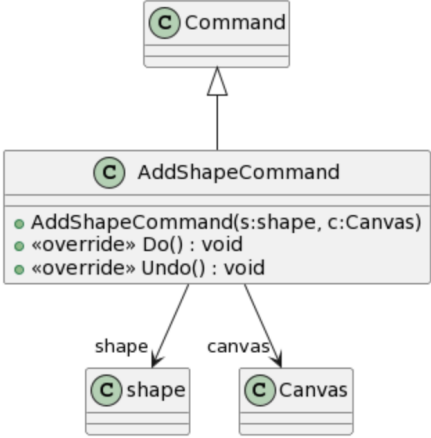
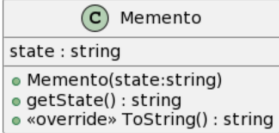
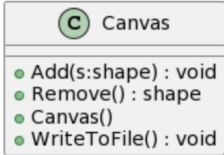
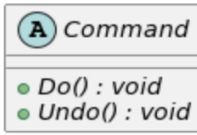


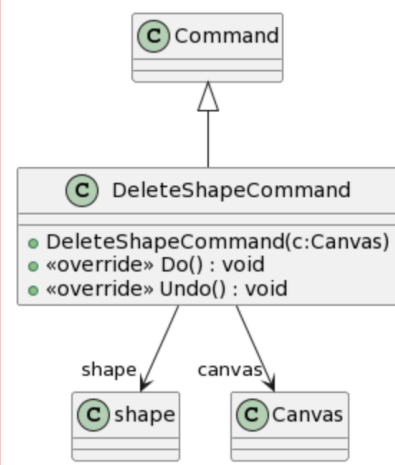
Rectangle class



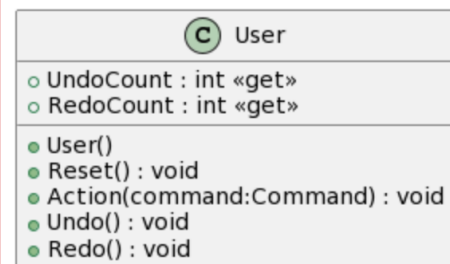
Line class

Line class

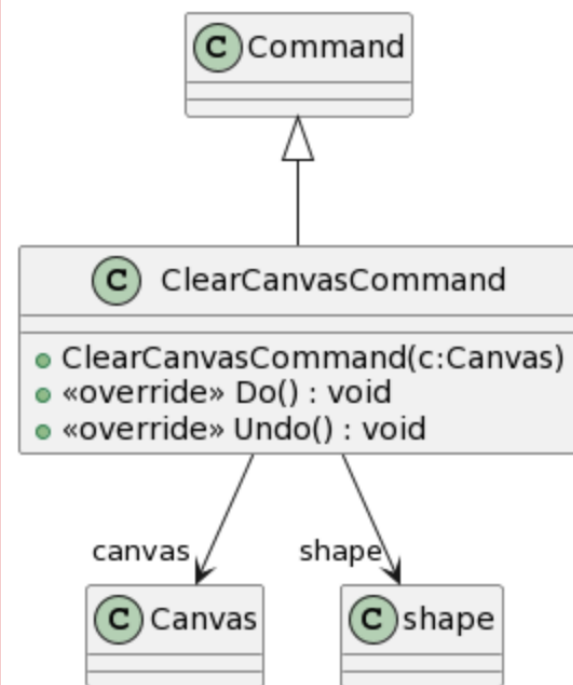
 <pre> classDiagram class Shape { <<abstract>> } class Line { xStart : int «get» yStart : int «get» xEnd : int «get» yEnd : int «get» line(xStart:int, yStart:int, xEnd:int, yEnd:int) «override» printshape() : string } Shape < -- Line </pre>	 <pre> classDiagram class Shape { <<abstract>> } class Line { xStart : int «get» yStart : int «get» xEnd : int «get» yEnd : int «get» line(xStart:int, yStart:int, xEnd:int, yEnd:int) «override» printshape() : string } Shape < -- Line </pre>
<p>Things that memento has that command doesn't</p>	<p>Things that command has that memento doesn't</p>
<p>Caretaker class</p>  <pre> classDiagram class Caretaker { addMementoToCanvas(m:Memento) : void undoFromCanvas() : void redoToCanvas() : void clearCanvas() : void WriteToFile() : void PrintSVG() : void getSizeMemento() : int getSizeCanvas() : int } Caretaker --> "1" List : canvas<Memento> Caretaker --> "mementos" List : mementos<Memento> </pre>	<p>AddShapeCommand class</p>  <pre> classDiagram class Command { <<abstract>> } class AddShapeCommand { AddShapeCommand(s:shape, c:Canvas) «override» Do() : void «override» Undo() : void } Command < -- AddShapeCommand AddShapeCommand --> shape AddShapeCommand --> Canvas </pre>
<p>Memento class</p>  <pre> classDiagram class Memento { state : string Memento(state:string) getState() : string «override» ToString() : string } </pre>	<p>Canvas class</p>  <pre> classDiagram class Canvas { Add(s:shape) : void Remove() : shape Canvas() WriteToFile() : void } </pre>
	<p>Command class</p>  <pre> classDiagram class Command { Do() : void Undo() : void } </pre>
	<p>DeleteShapeCommand class</p>



User class



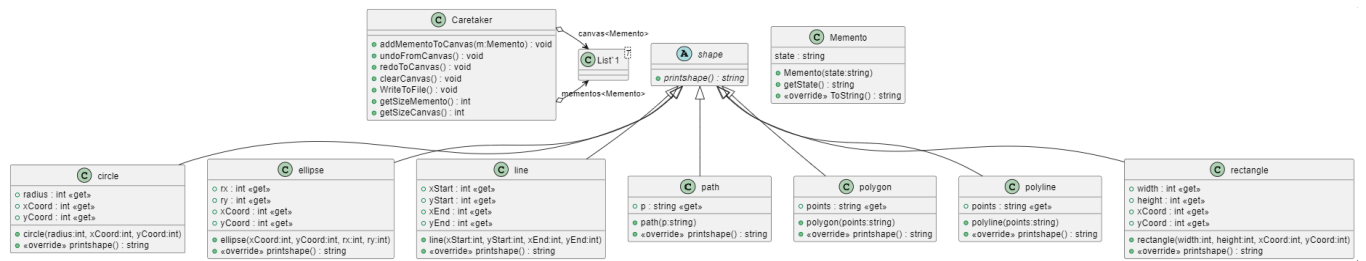
ClearCanvasCommand Class



Jessica Krapfl 20250742

CS264 Assignment 4

Memento UML diagram



Command UML diagram

