**Object-oriented Language and Theory**

**Mini-project Report**

**OOLT.ICT.20192.Group8**

Members: Lai Tien Duc 20176722

Nguyen Viet Hoang 20176762

Nguyen Tri Hung 20176773

**1. Mini-project description**

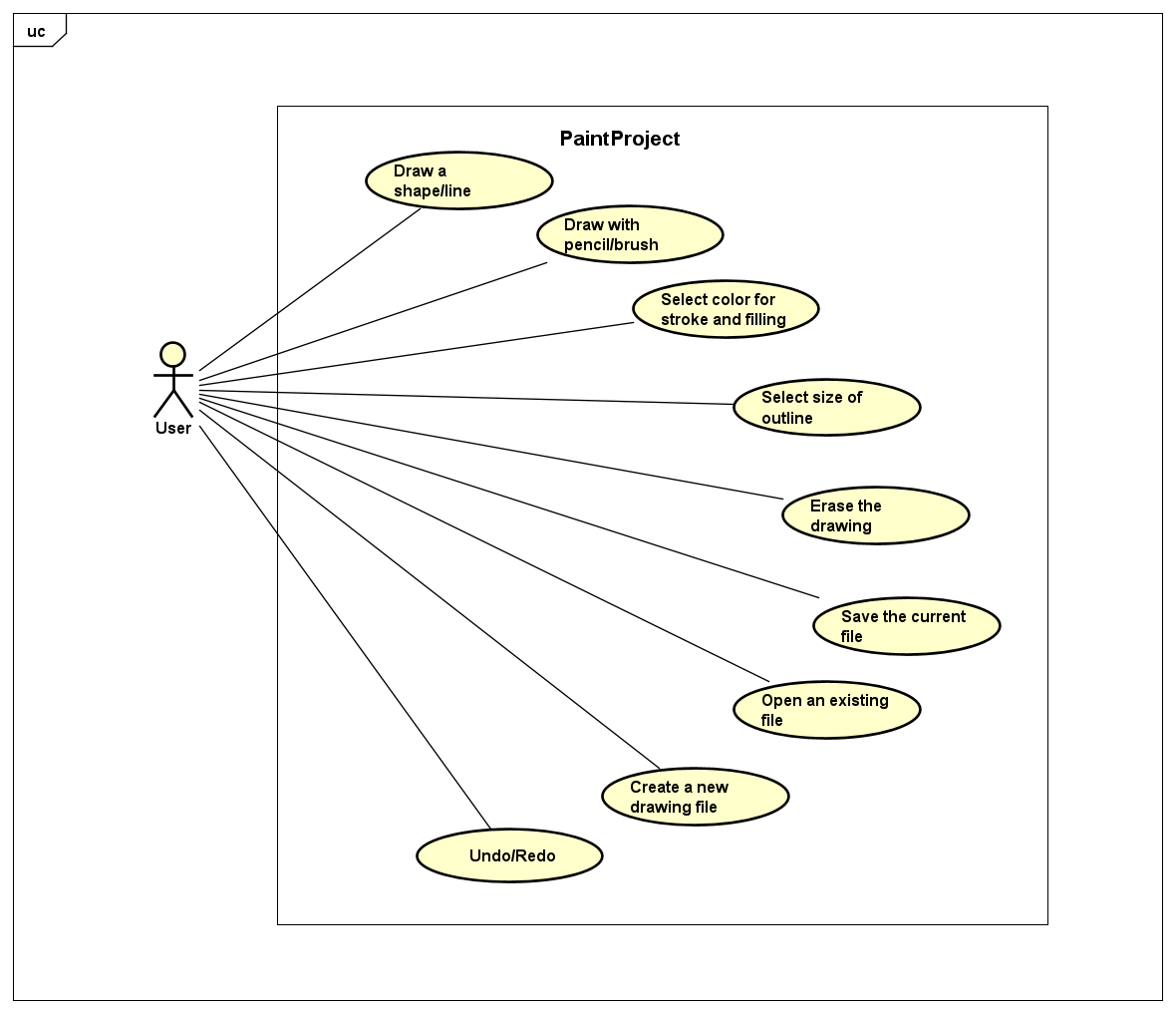
***1.1. Requirements***

* Draw:
* A shape (rectangle, circle, ellipse…): Select a shape, drag it in the drawing space (you can stretch it in whatever size you want within the space), and release the mouse to finish.
* A single straight line: Like shapes, you drag the line and release the mouse to complete.
* A custom line: Choose the “pencil” to customize your drawing.
* A custom spray: Choose the “brush” to spray color to anywhere in your drawing.
* Erase a drawing (with an “eraser”): You can erase some area in the drawing space.

***1.2. Use-case diagram***

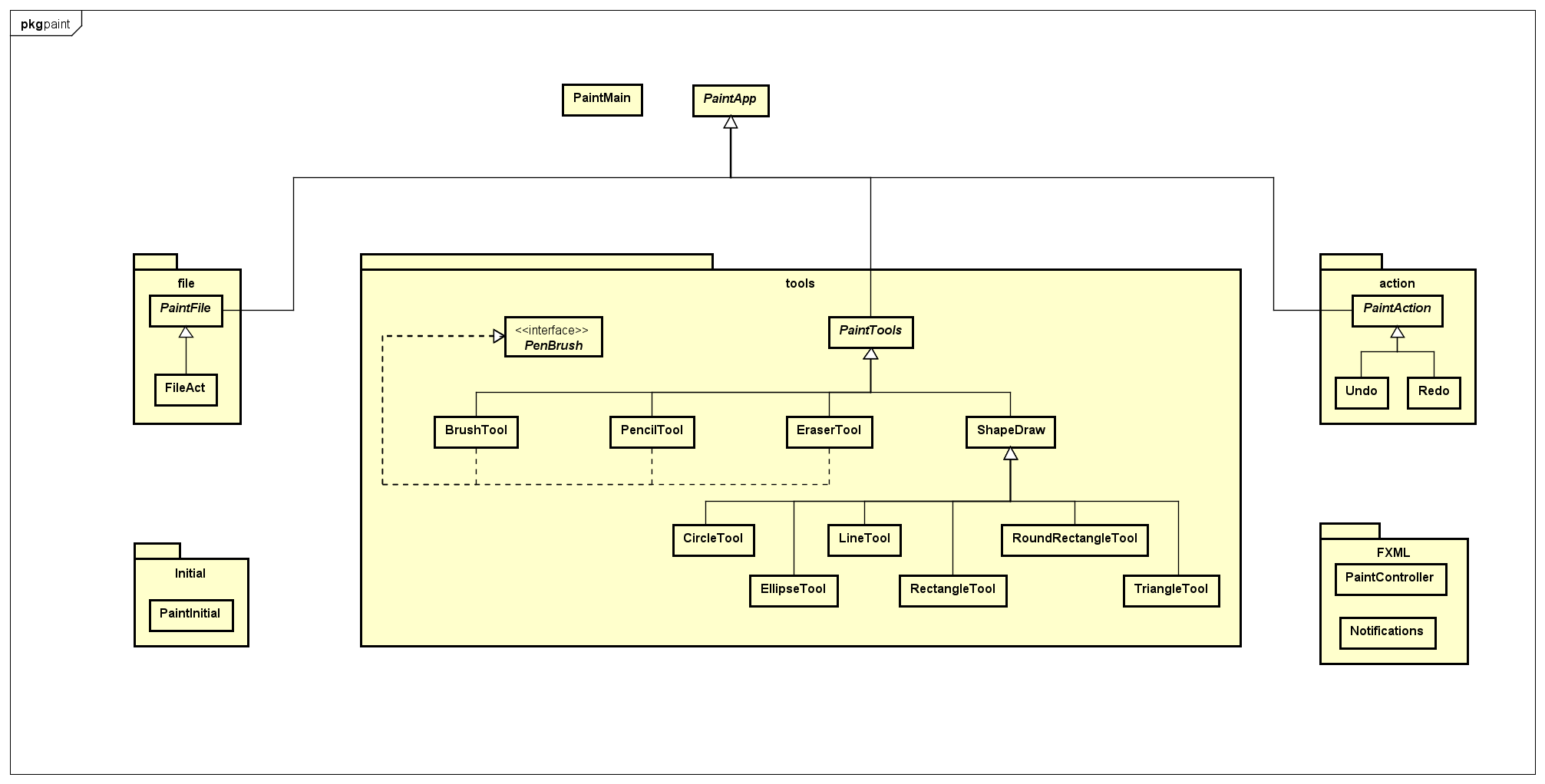
As shown in the figure below, the diagram indicates all of the requirements above. Plus, we have some more features for our Paint application:

* Select a numeric size (from 1 to 100, default is 5) for pencil, brush, shape outline and eraser.
* Select a stroke color (default is black) for pencil, brush and shapes; and a filling color (default is transparent) for shapes only.
* Open an existing file (PNG file recommended).
* Save the current file to PNG.
* Create a new file (a blank drawing space).
* Undo a drawing (back to the previous state of the drawing).
* Redo a drawing (return to the pre-undo state).



**2. Design ideas**

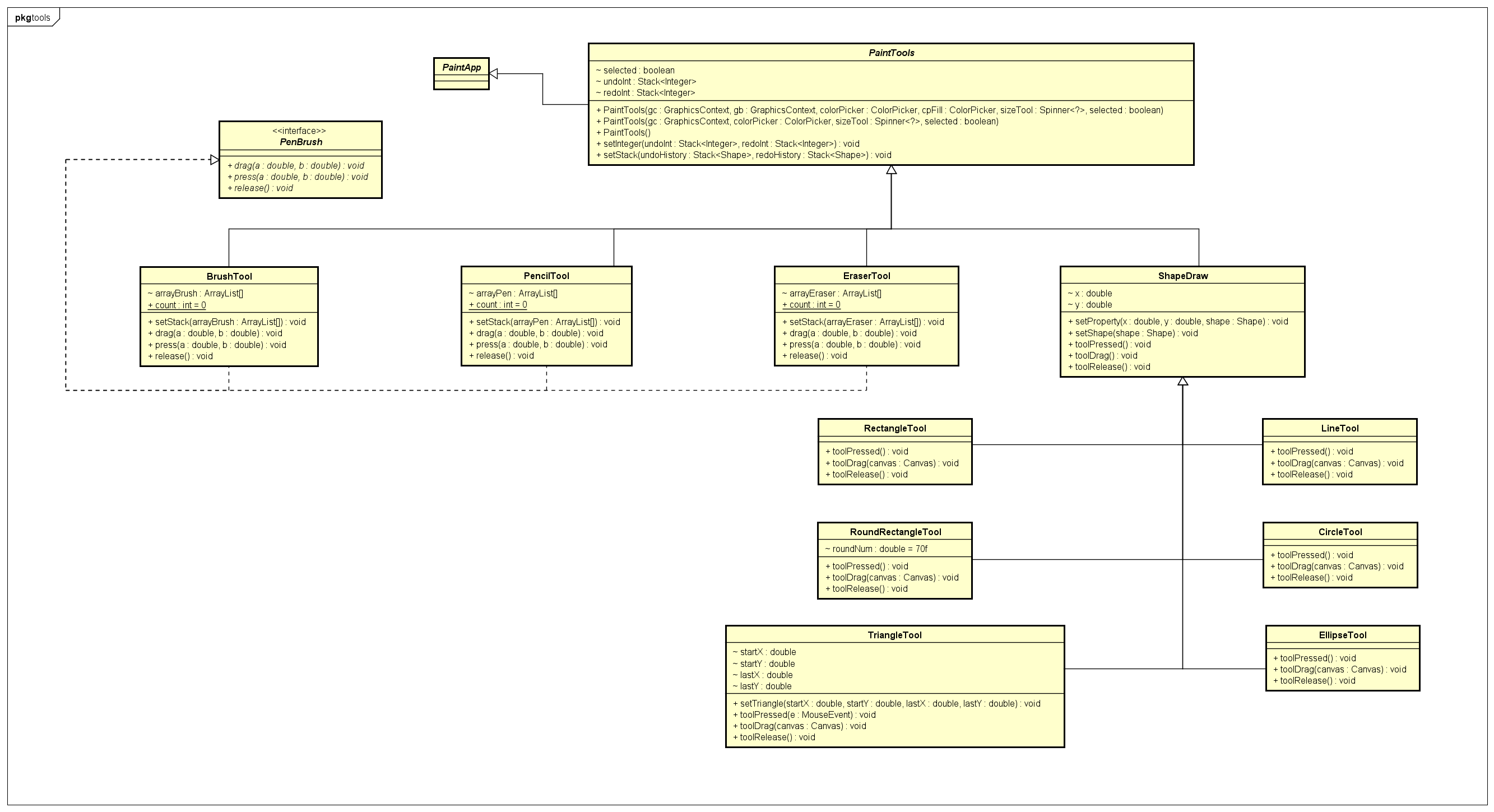
***2.1. General overview***

Class diagram of overall mini-project

The core of this mini-project is the “paint.tools” package with classes for making the graphics. 2 packages, “paint.file” and “paint.action”, also play important roles to fulfill the functions of our Paint. Those packages all have an abstract class of their own, which extends from the abstract class “PaintApp” (containing the attribute gc, gb of type GraphicContext – used for set properties of a drawing).

The package “paint.FXML” contributes to the GUI, which is really important for users, while “paint.Initial” contains the class that creates the first look of that GUI.

***2.2. Each package details:***

 *2.2.1. Package “paint.tools”:*

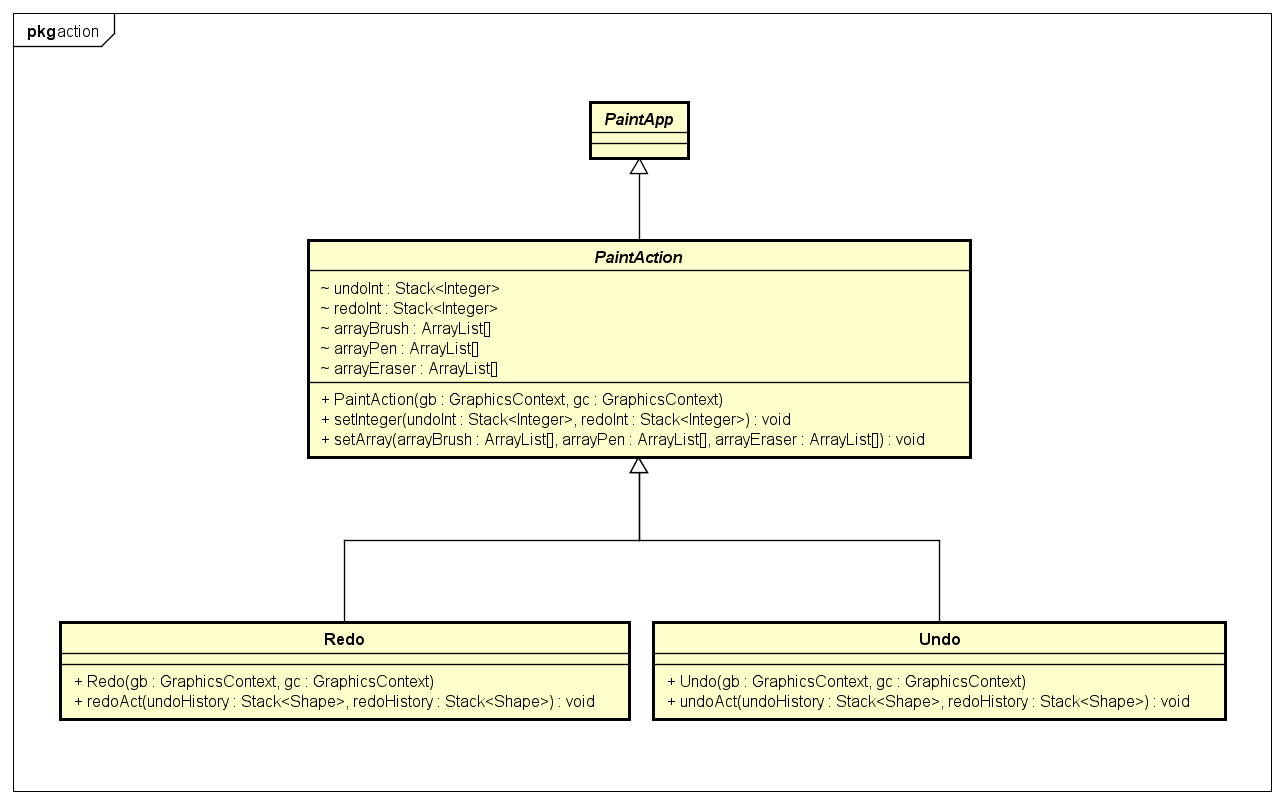
Package “paint.tools” begins with an abstract class “PaintTools”, which extends from “PaintApp” and contains the necessary attributes (of type GraphicContext, ColorPicker…) and constructors for other classes to make the functions of Paint. There are 4 classes extending “PaintTools”: PencilTool, BrushTool, EraserTool and ShapeDraw.

The three classes, “PencilTool”, “BrushTool” and “Eraser Tool”, define three fundamental and familiar tools in traditional MS Paint, pencil (drawing custom line), brush (the color spray) and eraser. They both implement the “PenBrush” interface, which set the methods to use the tools above: drag(), press() and release().

The class “ShapeDraw” define some methods for its child classes to extend for their own uses. toolPressed() sets the graphic context (line width, stroke color, filling color) at the starting position, while toolDrag() makes the temporary graphic context at the current-cursor position (if that context is not in the expected shape, it will be overwritten). The toolRelease() method not only completes the drawing shape, but also add it to the “undoHistory” stack for the use of Undo/Redo classes afterwards. The last method is setProperty() for the shape, used in the FXML controller class.

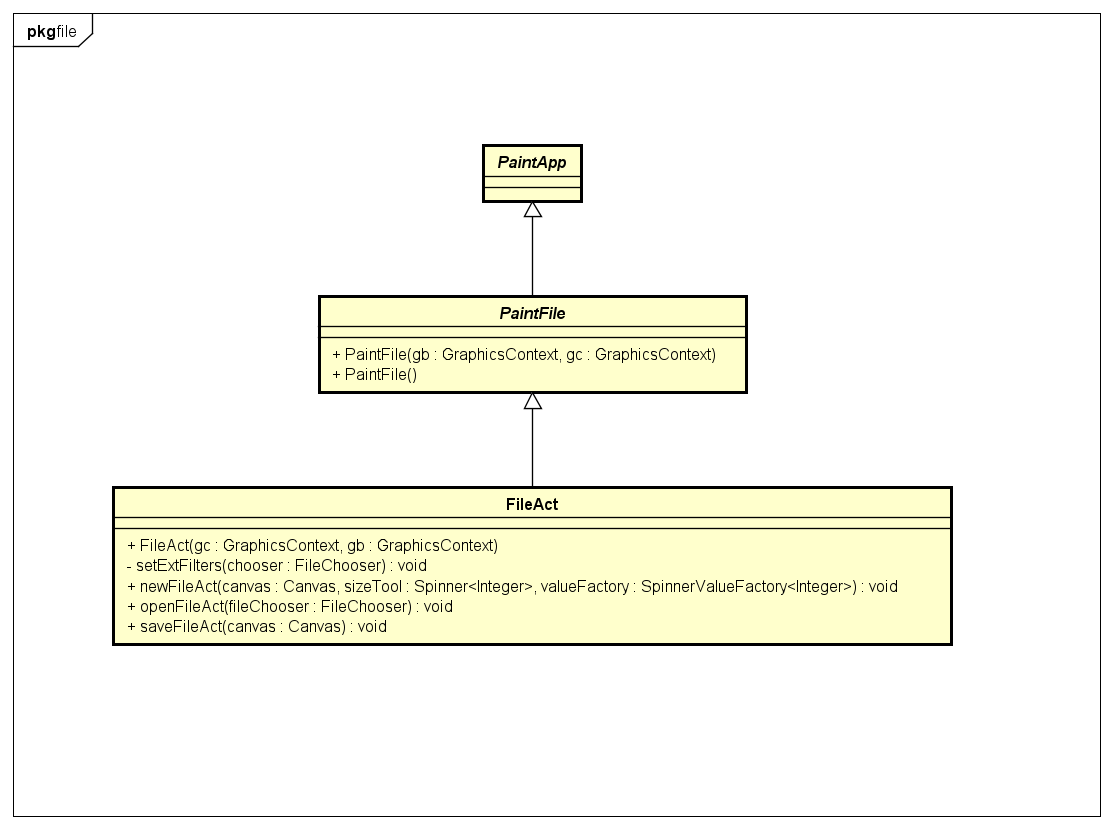
The child classes of “ShapeDraw” are LineTool, CircleTool, EllipseTool, TriangleTool, RectangleTool and RoundRectangleTool. While the other 5 classes implement just the methods from their parent, “TriangleTool” defines one more method setTriangle(), used for assigning the co-ordinates of the drawing triangle.

*2.2.2. Package “paint.action”:*



This package contains an abstract class “PaintAction”, which extends “PaintApp” from outer-package “paint”, and its two children “Redo” and “Undo”. Those two define operations to come back to the previous state (undo a drawing), or return to the pre-undo state (redo a drawing) of the drawing space. The operations use stacks, which contain the drawings, and the push(), pop() methods to execute undo or redo.

*2.2.3. Package “paint.file”:*



The package also has an abstract class “PaintFile” extending from “PaintApp”, and the child class “FileAct”. The “FileAct” class defines 3 operations to open, save and create a new paint file (here we use PNG file). newFileAct() creates a new blank drawing canvas, while saveFileAct() renders the current canvas to a PNG file and saves it in the computer, and openFileAct() loads the selected PNG file to redraw to the drawing area.

**3. Member assignments**

***3.1. Tien Duc:***

Link reference source code: <https://gist.github.com/abdelaziz321/e9932bd15e4b263c3dae08644c61600c>

1. Idea
   * Using mouse event (press, drag, release) to draw on canvas
   * Redo/Undo using stack to store value of shape and draw again on canvas
   * Using fill color to color the shape
2. Copy with modify
   * SetOnMousePressed:
     + Copy method for pencil, rectangle, circle, line
     + Add new shape (ellipse, triangle, roundRectangle, bursh)
     + Modify eraser method: using method setStroke()
   * SetOnMouseDragged:
     + Copy method for pencil, rectangle, circle, line
     + Add new shape (ellipse, triangle, roundRectangle, bursh)
     + Add canvas ‘overlay’ under canvas ‘canvas’. Purpose: when mouse is dragging, it draws on both canvas but clearing on canvas ‘overlay’ imediately. Therefore, user know the size of the shape
   * SetOnMouseRelease:
     + Copy method for pencil, rectangle, circle, line
     + Add new shape (ellipse, triangle, roundRectangle, bursh)
   * Redo/Undo:
     + Copy method for rectangle, circle, line
     + Add new shape (ellipse, triangle, roundRectangle)
   * File:
     + Copy open, save method
     + Add new file method
     + Clear redo/undo stacks when creating a new file
3. Modify
   * Each tool has 3 method: setOnMousePressed(), setOnMouseDragged(), setOnMouseRelease()
   * Add abstract class: PaintTools, ShapeDraw, PaintApp, PaintFile, PaintAction
   * Add interface: PenBrush

***3.2. Tri Hung:***

a) Design GUI using SceneBuilder:

* + 3 sections: Tools, Canvas & Palette, each section lies in an Anchor Pane
  + Tools section includes:
    - Pencil: Draws patterns as the cursor goes
    - Brush: Draws like a pencil but with dots
    - Line: Draws a straight line, in any direction
    - Rectangle: Draws a rectangle
    - Circle: Draw a circle
    - Ellipse: Draw an ellipse/oval
    - Rounded Rectangle: Draws a rectangle with rounded angle
    - Triangle: Draw a triangle
    - Eraser: Erases anything that it moves through
  + Canvas includes:
    - Overlay: The canvas that lies underneath, used to display the effect when user move their mouse while drawing shapes (Line, Rectangle, Circle, Ellipse, Rounded Rectangle, Triangle)
    - Canvas: The canvas on the surface, displays:
      * the result after the user release their mouse when they draw shapes
      * The stroke of the pencil and the brush
  + Palette includes:
    - Application’s symbol – dark green circle with the letters: ‘S I M P L E P A I N T'. When clicked, displays application information.
    - Color picker - chooses color for the stroke of pencil, brush and the border of shapes
    - Size picker: Changes stroke size of the pencil and brush, the shapes’ borders’ thickness.
    - Fill: Fill color inside the shapes. Default: transparent
    - Undo: Undo drawing shapes
    - Redo: Redo drawing shapes
  + Reference: <https://o7planning.org/en/11009/javafx>

b) Modifying the main controller - PaintController.java:

* + Add actions to open alert windows (or so called dialog boxes) from class Notification.java when user clicks on these MenuItemss from the MenuBar:
    - New File: Notifications.ShowNewFileAlert()
    - Exit: Notifications.ShowExitAlert()
    - About: Notifications.ShowAboutAlert()

Then perform the suitable operation: Save File, Exit or simply do nothing based on the return value of each fucntion.

* + Add a function labelClick() to display information when the user clicks in the application’s symbol (dark green circle with the letters ‘S I M P L E P A I N T'). This is done by using adding a button at the front of that circle and make that button transparent, making it seem like the user click the symbol instead of the button.
  + Changing the icons of the tools (changed in the PaintController.java) for better appearance (Old icons are from Hoang’s). Icons are downloaded from: <https://www.flaticon.com/> and modified by myself.
  + Changing the several icons of the cursor: pencil, brush, eraser to fit the new icon (changed in the PaintController.java), the rest remain as Hoang’s. Icons are downloaded from: <https://www.flaticon.com/> and modified by myself.

c) Notifications:

* + There are 3 functions that are defined within the class Notifications. They are used to inform the user or ask for their confirmation:
    - ShowAboutAlert() returns void : informs the user about the app’s information
    - ShowExitAlert() returns an int: 1, 2 or 3 respectively for button OK, Save & Cancel : Asks the user for confirmation of exiting with/without saving current painting
    - ShowNewFileAlert() returns an int: 1, 2 or 3 respectively for button OK, Save & Cancel: Asks the user for confirmation of creating new file with/without saving current painting
  + Reference: <https://code.makery.ch/blog/javafx-dialogs-official/> (Copy with modification for ShowExitAlert() and ShowNewFileAlert())

d) Resizing the canvas so it fits the size of the AnchorPane that contains it (This function is disabled because it brings an unwanted bug causing the painting to be deleted partially when minimizing the drawing window):

* + References:
    - <https://www.thetopsites.net/article/53247942.shtml> (Used the idea),
    - <https://stackoverflow.com/questions/52472046/alerts-in-javafx-do-not-close-when-x-button-is-pressed> (Used the idea)

e) Adding favicon to the app:

* + Method used: Stage.getIcons.add(new Image(<Image link goes here>));
  + Icon is downloaded from: <https://www.flaticon.com/> without any modification

***3.3. Viet Hoang:***

* Color: Using class ColorPicker (from package javafx.scene.control) to have the color tables for stroke and filling. Set black for stroke (“colorPicker” attribute) and transparent for filling (“cpFill” attribute) as default.
* Size Tool: Using class SpinnerValueFactory (from package javafx.scene.control) to set the range (1 – 100) of the size of stroke. Set size 5 as default. The size tool is editable.
* Add images to buttons: Get the image files as streams, then create new “Image” attributes with the streams, image size to make “ImageView” after that. Finally, set graphic for the corresponding buttons.
* Cursor display: For each operation, the cursor is set into different appearances. For example, when drawing a line or a shape, it is a “crosshair” cursor. When hovering the button, it appears as a “hand”. When using a pencil, brush or eraser, it has the corresponding look, using “ImageCursor”.
* Shortcut key: For some “MenuItem” attributes in “MenuBar”, set shortcut key combinations. For example, “Ctrl+S” for save files, “Ctrl+Z” for undo…