Advanced Object Oriented Programming & Java

Assignment 4 – Interactive Drawing Application of polymorphic Shapes

Dr. Moshe Deutsch

mdeuts@ruppin.ac.il

Submission Instructions:

1. במשימה שאלה אחת. לכל Public Class בשאלה יש לכתוב קובץ java נפרד C.java כאשר C הוא שם ה class שממומש בקובץ זה.

2. הקוד עבור כל ה Classes חייב להיות !!!Well-Commented

3. בנוסף, יש להעתיק את כל התוכניות במלואן לקובץ WORD אחד. בקובץ זה, עבור כל class שמועתק לשם, יש לציין את שם ה class הממומש. בנוסף, בסיום ,יש לצרף כמה דוגמאות מייצגות לפלט ממסך ההרצה. בתחילת הקובץ הנ"ל, יש לציין את שמת המגישים ות.ז. שלהם (**במודגש!**).

4. יש לכתוב הערות לקוד, התומכות ב javadoc **עם קישורים ל Java API Documentation** (ראו פסקה אחרונה בתיאור המשימה בהמשך). יש להריץ Javadoc על כל הקבצים ולנתב את הפלט שלו לתת ספריה בשם Doc. הפעם, אתם **נדרשים להריץ את Javadoc בצורה כזו שיווצרו קישורים ל Java API Documentation מהתיעוד שלכם ושייתועדו גם כל ה private Classes ו private methods בפרוייקט שלכם** (ראו פסקה אחרונה בתיאור המשימה בהמשך).

5. יש לשים את כל ה sources בתת ספריה Src ואת קובץ ה WORD מעל שתי הספריות (Doc ו Src) במבנה הספריות ולכווץ בקובץ ZIP או RAR אחד את כל אלה ולהעלות לאיזור המשימה באתר הקורס, באמצעות ה Moodle.

6. שם קובץ ה-zip יהיה כך: AssigmnentX_id1_id2. כאשר: X מציין את מספר המטלה, id1 מציין את ת.ז של סטודנט 1 ו-id2 מציין את ת.ז של סטודנט 2.

7. כאשר פותחים את הקובץ המכווץ, יש לקבל 3 דברים: Assignment4.docx – קובץ ה WORD (סעיף 3), ספריה Src (המכילה את כל ה Java source files – סעיף 5) וספריה Doc (סעיף 4).

8. תאריך אחרון להגשת המשימה: יום שלישי, תאריך 12/01/2021 – עד סוף היום (קרי עד 23:55 בלילה של תאריך זה). לא ניתן לאחר בהגשה מעבר לתאריך זה המערכת תיסגר באופן אוטומטי להגשות ומי שלא יגיש עד לתאריך זה ייקבל באופן אוטומטי ציון סופי 0 במשימה.

9. ***יש להגיש את המטלה בזוגות!*** בתחילת קובץ ה Word – יש לרשום במודגש את הפרטים המלאים של המגישים (שמות פרטיים, שמות משפחה ות.ז). ***רק סטודנט אחד מהזוג המגיש צריך להעלות את המשימה לאתר ה Moodle***.

Exercise 1 – 100%

Interactive Drawing Application of polymorphic Shapes (of "MyShape" polymorphic hierarchy).

Introduction:

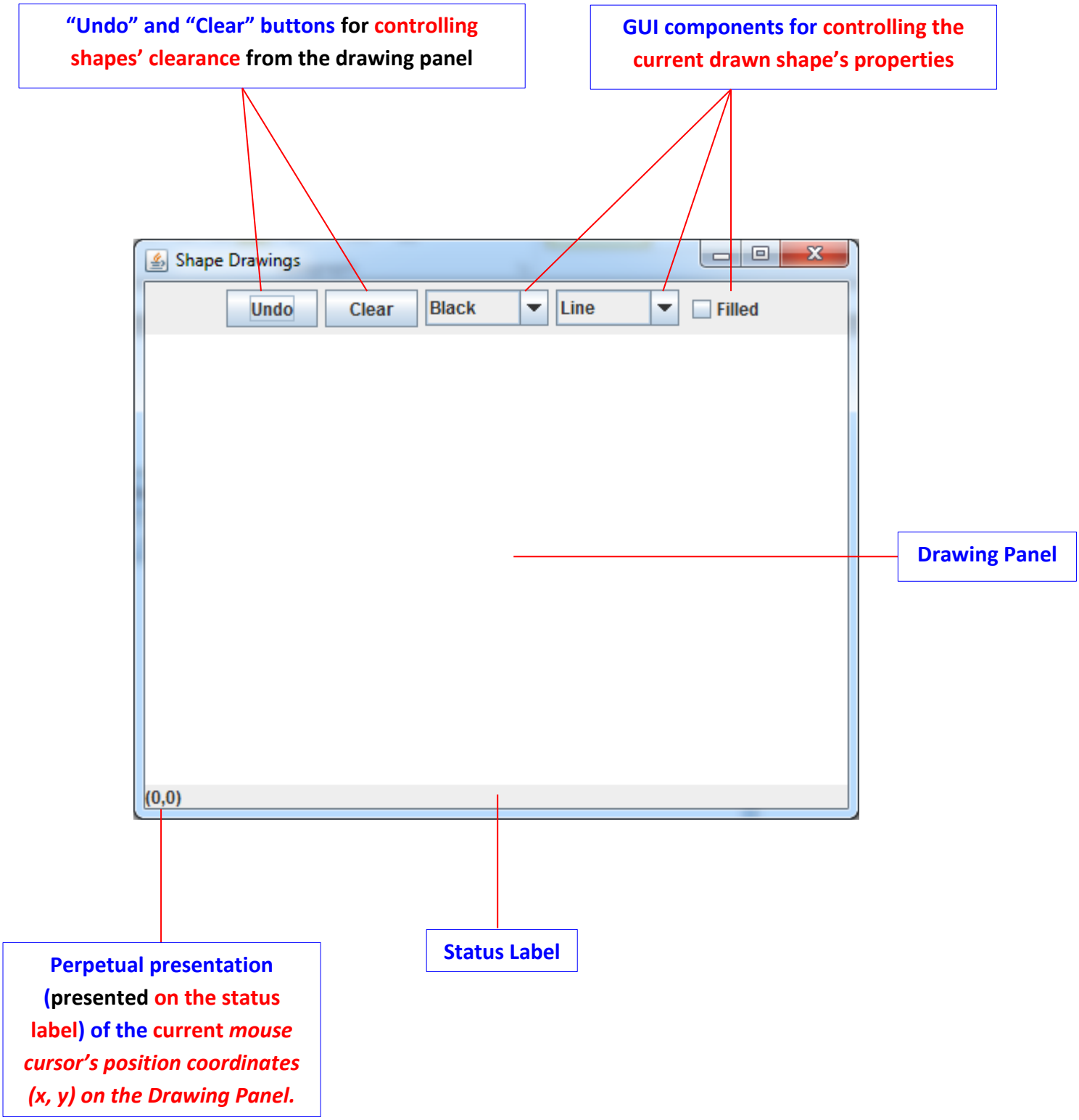
This Assignment is based on "MyShape" polymorphic hierarchy of shapes, you developed in Part II of Assignment 3. You will need to use this hierarchy, as it is, in order to create a smart interactive drawing application, allowing the user to choose:

- The current shape's type (using a combo box GUI component);
- The current shape's color (using a combo box GUI component);
- And whether the shape is filled or not (using a check box GUI component); this option is relevant only for rectangles and ovals and its choice does not affect lines!).

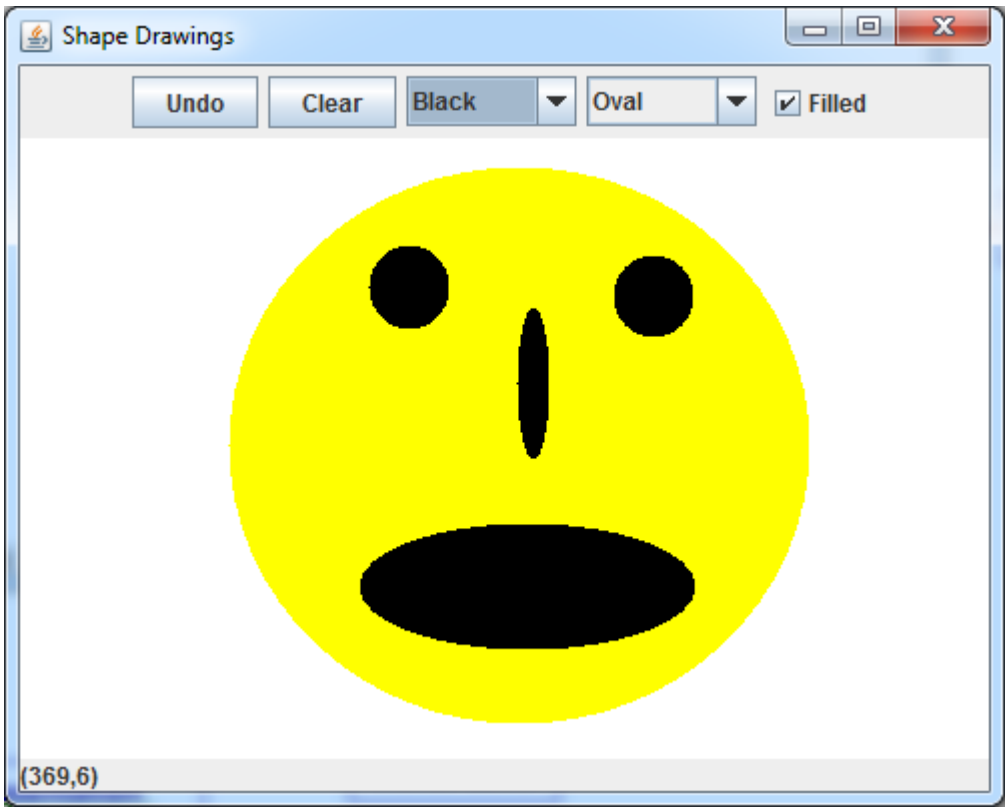
The drawing panel is accumulative: namely, every shape that was drawn on the panel must keep on existing on it until explicitly cleared! There are two types of clearance that can be performed by two respective buttons:

- "Undo" Button – when this button is clicked the last drawn shape should be cleared from the drawing panel, but all the other shapes should remain exactly as they were! Clicking this button again will clear the next last shape… and so on. Clicking this button when the panel has no shapes will have no effect (at least not with regards to GUI visibility!).
- "Clear" Button – when this button is clicked the drawing panel must be cleared of ALL the shapes! The purpose of this button is to clear the panel – to be ready for a new drawing session. Clicking this button when the panel has no shapes will have no effect (at least not with regards to GUI visibility!).

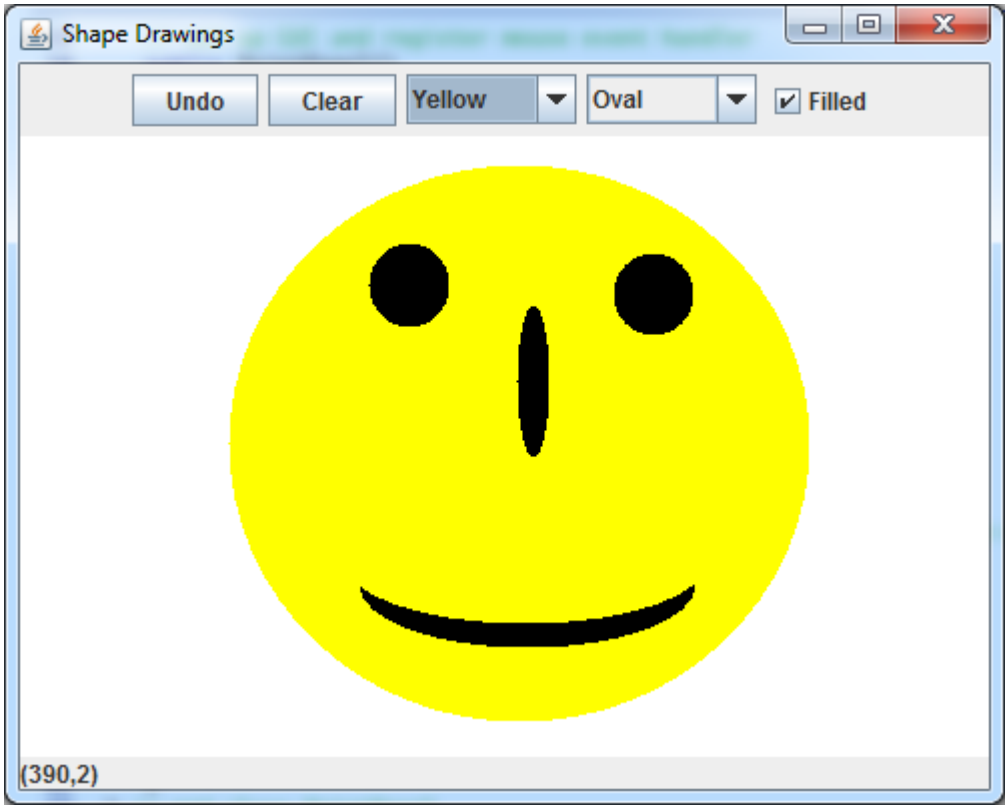
The application's GUI should look as follows:



An example of a drawing session:



And another yellow filled oval to create this:



Recall that, in effect, whenever the drawing panel is "repainted", its paintComponent method is invoked. This suggests that the history of the drawn shapes (for the current drawing session!) must be preserved. Also recall that any of the shapes in "MyShape" hierarchy knows how to "draw" itself polymorphically (according to the data kept in the specific shape object) – this suggests that a polymorphic structure of some sort (say an array of Shapes) should keep the history of the drawn shapes. Thus, whenever paintComponent method of the drawing panel is invoked – this structure should be traversed and get each shape to draw itself polymorphically!

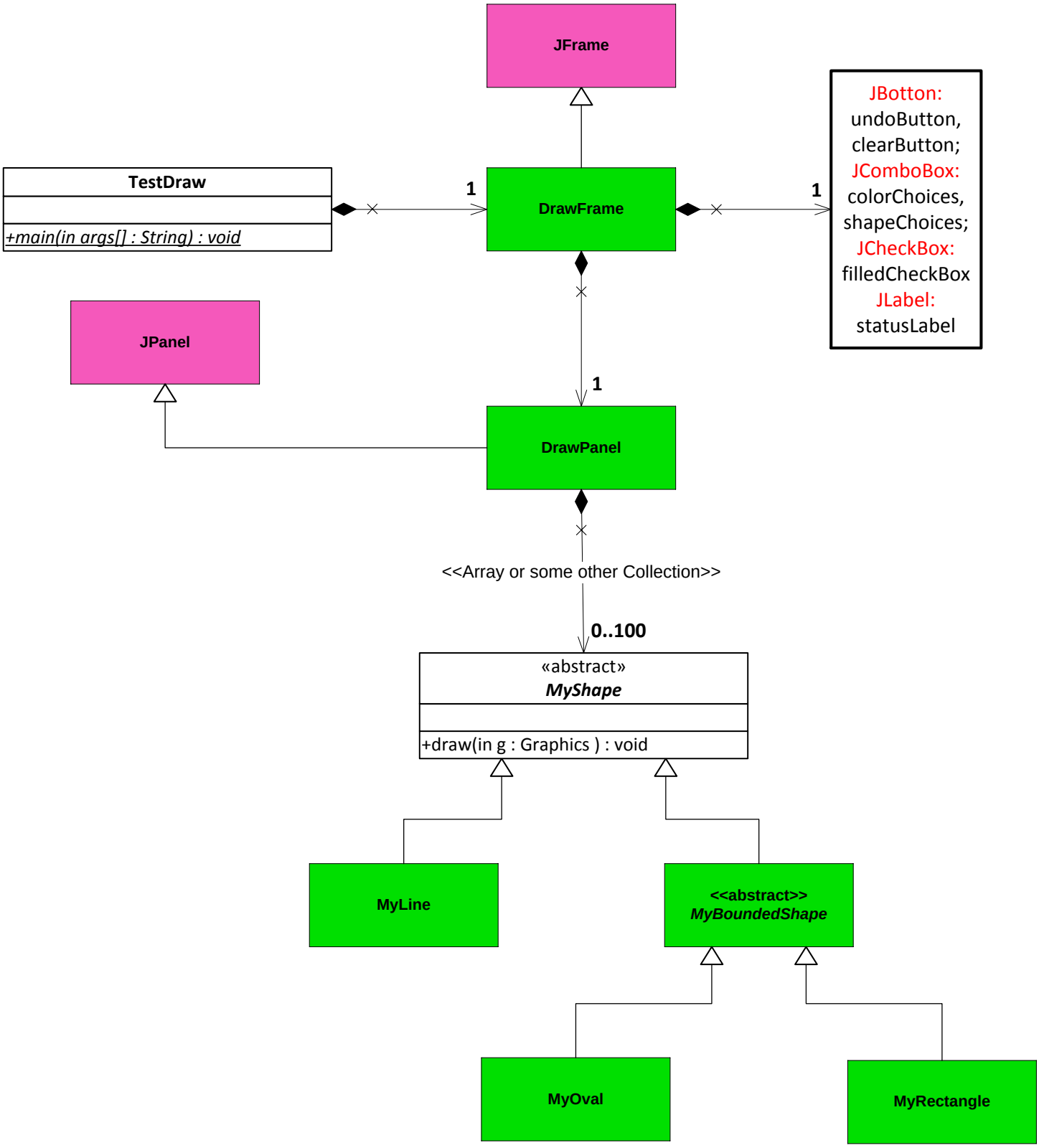
The maximum number of shapes, allowed to be drawn on the panel is 100. If this maximum is reached, any next attempt to draw any shape should be denied and a warning message should be issued, using a Message Dialog Box.

Explanation Movie on YouTube is divided into two movies: Part I and Part II:

- Click here to watch YouTube Explanation Movie Part I
- Click here to watch YouTube Explanation Movie Part II

System Design:

The UML Class Diagram for the public classes in this project is as follows:


JFrame
JBotton:
undoButton,
clearButton;
JComboBox:
colorChoices,
shapeChoices;
JCheckBox:
filledCheckBox
JLabel:
statusLabel
TestDraw
+main(in args[] : String) : void
1
DrawFrame
1
JPanel
1
DrawPanel
<<Array or some other Collection>>
0..100
«abstract»
MyShape
+draw(in g : Graphics) : void
MyLine
<<abstract>>
MyBoundedShape
MyOval
MyRectangle

Guidelines:

In this assignment, you will NOT receive specific instructions, as to which data members and methods you need to implement in each class! Rather, you will be given a guideline overview of the intended functionality, behavior and GUI appearance (as well as GUI intensions). These guidelines are accompanied with a number of **Demonstration Videos** (added in a supplement directory below Assignment4 submission object in the course area – there are 7 Video files compressed in a single file named "DemonstrationVideos.rar"), each demonstrates a particular usage of the system.

All these guidelines shall provide you with a clear intentional idea of:

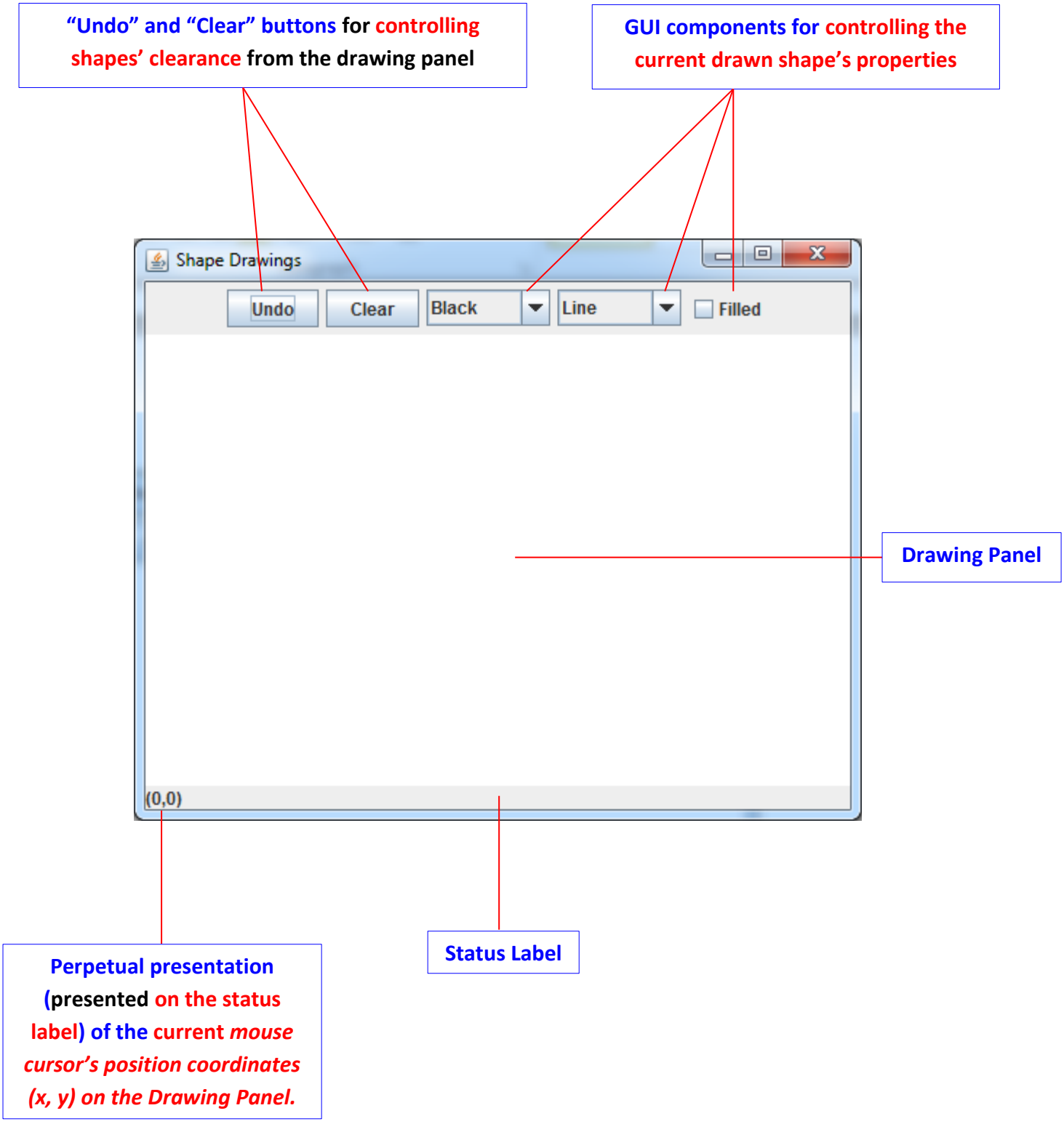
- How the system can be used;
- What are the system limitations;
- What are the extreme cases you should be aware of (and protect from!);
- The interaction of the various classes, what information is passed between classes (and in which direction!) and the functional responsibility of each class.

According to these guidelines, you should carefully plan your solution and implement it, so that it meets all the guidelines' requirements.

Functional overview of the various classes

a. TestDraw class is simply the class that contains the main method that creates the DrawFrame object (which is the main window of the application) and sets its properties, according to most of the examples we have seen in chapter 8 (its default close operation, its initial size and its visibility).

b. DrawFrame class constitutes the main window of the application (hence it extends JFrame API class). This class is responsible for creating all the GUI elements and organize these in the introduction section above. A remainder of how it should look like (See also demonstration videos: 1 to 4 to get a closer look of how it should look like):



The Draw DrawFrame class is responsible to **define an event handler for listening and handling the events of the 5 GUI components that are added to the top part of the window** (the mouse events for the drawing should be handled exclusively by the object of class DrawPanel, which is the smart drawing panel!).

Look carefully at the UML class diagram (System Design section above). Few important guidelines according to this diagram:

- All the GUI components are encapsulated within the DrawFrame class – i.e. they are private members of this class (this includes the object of class DrawPanel, which is the smart drawing panel!);
- The information flow is only from the DrawFrame to the DrawPanel (and NOT the opposite!). Therefore, the DrawFrame should provide to the DrawPanel the information the DrawPanel needs via its constructor (e.g. a reference to the statusLabel, so that DrawPanel can save it internally and update it directly – with the (x, y) coordinates of the mouse position on the DrawPanel);
- DrawFrame is **NOT allowed(!!!)** to provide to DrawPanel any reference to any of the 5 GUI components (added in the top part of the DrawFrame). Since the DrawPanel is a smart object – its class needs to provide a well-defined interface (i.e. methods!!!), so that the DrawFrame can invoke on the DrawPanel object to update it with all the various GUI events. For example, on the event of selecting a color in colorChoices JComboBox, DrawFrame should invoke a DrawPanel object method named "setDrawingColor" – sending it the Color object chosen in the item selection event of the colorChoices JComboBox. The "setDrawingColor" in DrawPanel should simply save this information in its state machine, so that the next Shape drawing will use the newly selected color;
    - In conclusion, the communication between DrawFrame and DrawPanel is performed only via the DrawPanel public methods, which DrawFrame can invoke on its contained DrawPanel object – this requires you to plan carefully, which methods the DrawPanel should

expose, in order to allow DrawFrame to inform it, regarding the various GUI events (of the 5 GUI components added in the top part of the DrawFrame).

- DrawFrame class also contains private data members to support the data presented by the GUI components and reflected by them. For example, you will need an array of Strings of 13 color names for initializing the colorCoices JComboBox and another array of the same size, containing the corresponding Color objects, so that when a choice is made in the JComboBox – its selected index will match the Color objects array, from which you can immediately get the corresponding Color object (see the JList example on Ch. 8, slide 96). See also demonstration video 1 to see exactly what information you need in each of the 2 JComboBox objects of DrawFrame.

c. DrawPanel is the smart drawing panel. As you can see in the UML Diagram, it extends the API JPanel, so it has capabilities of JPanel (to draw Shapes, using the Graphics object methods – sent as a parameter to "paintComponent" method; recall, this method is invoked automatically in 4 circumstances – see Ch. 8, slide 134), but it also implements the entire state-machine for drawing the shapes. In fact, most of the sophistication of this project will be in this class!

Recall that the communication between DrawFrame and DrawPanel is performed only via the DrawPanel public methods, which DrawFrame can invoke on its contained DrawPanel object – this requires you to plan carefully, which methods the DrawPanel should expose, in order to allow DrawFrame to inform it, regarding the various GUI events (of the 5 GUI components added in the top part of the DrawFrame) – so that DrawPanel can properly update its state machine, upon GUI events in DrawFrame.

Guidelines regarding the data members of DrawPanel:

- **The maximum number of shapes, allowed to be drawn on the panel is 100**. If this maximum is reached, any next attempt to draw any shape should be denied and a warning message should be issued, using a Message Dialog Box. See example for this in demonstration video 6.
- You must use the "MyShape" hierarchy of shapes, you developed in Part II of Assignment 3. You will need to **use this hierarchy, as it is!** That is, every shape has the capability of drawing itself polymorphically (according to the data kept in the specific concrete shape object).
- This means you must create the "Current Shape", according to the current selections ("Shape Type", "Current Color", and whether it should be "Filled Shape") – only at the creation point, it is treated specifically (because it needs to invoke the specific constructor!). But **from this point onwards, ALL shapes must be treated polymorphically!!!** (all the necessary updates – after the point of creation can, and should!, be done polymorphically via a reference to MyShape object). This means, both the "Current Shape" that is currently drawn, and the collection of shapes that were previously drawn (in the current drawing session) must be treated polymorphic ally!!!
- As soon as the "Current Shape" is drawn, it should be kept in a Collection object of objects of type MyShape. This can be a simple array or a collection class such as ArrayList, or even the Generic infrastructure you have written in Assignment 2 (but if you use it, you will need to provide a way to traverse all the elements from outside it!).

Guidelines regarding the functionality of DrawPanel:

- You should design the public methods of DrawPanel so that they support the ability to get the various state changes from

the outside, and update the state variables accordingly (for example, update the "Shape Type" or the "Current Color", etc). The current drawing will always get its properties from the up-to-dated state variables!

- DrawPanel **background color should be white!**
- DrawPanel class is the **ONLY one who is responsible for listening and handling the NECESSARY mouse events, in order to support the drawing functionality**. See demonstration videos 1 to 5, which will give you an idea of which mouse events you need to support.
- In general, the current drawing will start – **only as a result of the LEFT MOUSE BUTTON Pressed!!!** As long as the left mouse button is pressed and **the mouse is dragged**, the current shape should be constantly **re-drawn**, until the **left mouse button is released** (see this **re-drawing feeling on mouse dragged** in demonstration videos 1 to 5 and demonstrating video 7). Only when the **left mouse button is released**, the current shape should be re-drawn with its final X2 and Y2 coordinates and its MyShape object should be kept in the MyShape array (or other collection).
    - The way to implement this is to create the "Current Shape" object with the x, y mouse coordinates, from the mouse press event (both x1 & x2 are initialized with the x mouse coordinate and both y1 & y2 are initialized with the y mouse coordinate). Then as dragging/release events occur – update only the X2 and Y2 coordinates of the current shape on those events and then simply get the drawing panel re-painted to redraw the current shape (how do we repaint a JPanel)?
    - Given that every shape (current shape and the ones in the array/collection) can draw itself – then all the drawing work can be done in one place, that is DrawPanel **paintComponent method** (overridden on

the one inherited from JPanel). There – you need to invoke the **paintComponent** of the superclass (to get the JPanel cleared) and draw all shapes in the array/collection (if any!) and then draw the "Current Shape" (if exists!).

- The **array/collection of Myshape objects should be managed, in concept, as a stack**. Why? In order to support the functionality of "Undo". You should provide a public method in DrawPanel to be invoked by the "Undo" button handler in DrawFrame; all it should do is to remove the last Shape from the array/collection (where now, the shape before the last – becomes the last… and so on…). If you implement it with array, you do not need to remove any element! You can simply hold an integer instance variable that will always hold the up-to-dated "Shape Count", where you treat the elements in the array up to this "Shape Count" (and NOT up to the array's length!) – "Undo" simply means decrementing this "Shape Count" and get the drawing panel re-painted! **"Undo" does not influence any of the other state variables**
- "Clear" simply means clearing the array/collection of shapes and get the drawing panel re-painted! This will get the drawing panel cleared. Again, if you implement it with an array and "Shape Count" – all you need to do is to zero this instance variable and get the drawing panel re-painted! **"Clear" does not influence any of the other state variables**
- When the mouse is cursor is on the DrawPanel area, you **must ensure that its up-to-dated (x, y) coordinates are always updated on the left side of the statusLabel** (see the above screen capture and see demonstrating video 5 that demonstrates the desired behavior in this case).

- When marking this assignment, a **major emphasis will be put on modularity and encapsulation!** Avoid repetitions of code!

Further GUI Issues – Emphasizing some of the above points

Further Elaboration points with the reference to the Demonstration Videos:

a. In Demonstration Video 1, you can see the exact GUI appearance and layout of all GUI components. In addition, you can see exactly what textual information you need to present in each one of the 2 JComboBox objects of DrawFrame.

b. Demonstrating Videos 2 – 4 demonstrate several points:

   i. Drawing on the DrawPanel **begins only as a result of the LEFT MOUSE BUTTON Pressed!!!**

   ii. Notice the **effect of re-drawing feeling on mouse dragged event** in DrawPanel.

   iii. Notice that the **state of the filledCheckBox influences only on the drawing of Ovals and Rectangles**, but does not influence on the drawing of Lines (at all!!!).

   iv. Notice the behavior resulting from clicking "Undo" button several times. Moreover, note that clicking "Undo" when the DrawPanel is empty, should not have any effect on the DrawPanel functionality; in fact, DrawPanel must protect from this extreme case – ensure your system behaves correctly in this case!

   v. Notice the behavior resulting from clicking "Clear" button. Likewise, clicking "Clear" when the DrawPanel is empty, should not have any effect on the DrawPanel functionality.

vi. Notice that neither "Undo", nor "Clear" influence the state of the other GUI components. Namely, clearing the drawing panel or undoing its last drawing does not affect the state data of the drawing (the last user selections remain as they are!)

c. Demonstrating Video 5 demonstrates the behavior of the mouse event handler in updating the up-to-dated mouse coordinates on the left side of the statusLabel:

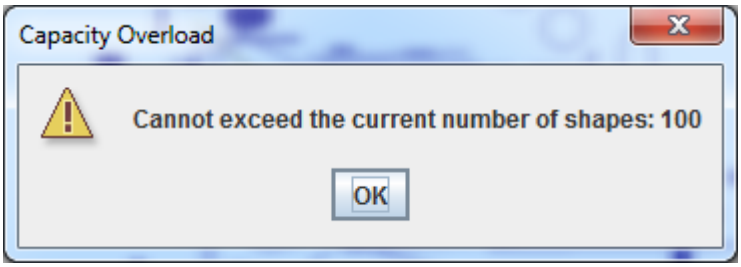
i. In general, all mouse events are listened and handled only in DrawPanel. Pay attention that when the mouse goes out of the DrawPanel area no mouse events occur (drawing is not possible and the statusLabel is not updated!), because the event listener/handler is attached only to the DrawPanel!

ii. When the mouse is cursor is on the DrawPanel area, you **must ensure that its up-to-dated (x, y) coordinates are always updated on the left side of the statusLabel**. Notice that statusLabel update is performed when the mouse is simply ***moved*** on the draw panel. Moreover, the statusLabel update is also performed when a drawing is being performed (i.e. when the mouse is ***dragged***).

iii. Again, notice the **effect of re-drawing feeling on mouse dragged event** in DrawPanel – this causes many interesting effects like swapping the shape's direction, resizing the shape or shrinking it – all this is possible, as long as a "Current Shape" drawing takes place; namely, a drawing was started with a **left mouse button pressed** and the **mouse is dragged – with the left button still pressed!** Recall, the final "Current Shape" is **committed** (and saved to the array/collection) **only when the left mouse button is released!**

d. Demonstrating Video 6 demonstrates several points regarding the maximum capacity of shapes allowed on the DrawPanel.

i. **The maximum number of shapes, allowed to be drawn on the panel is 100**. If this maximum is reached, **any next attempt to draw any shape** (i.e. it is examined when the mouse is pressed!) should be denied and a warning message should be issued, using a **Message Dialog Box**:



ii. Notice the use of "Undo" after the maximum capacity is reached: you can only add a number of new shapes, which corresponds to the number of "Undo" clicks!

iii. Notice the use of "Clear" after the maximum capacity is reached: this means **clearing the drawing panel and starting a new drawing session**, so the **user can draw up to 100 shapes in this new session!**

e. Demonstrating Video 7 demonstrates an example of extreme case you should protect from:

i. When a **left mouse button press occurs, but no dragging took place** – such a "Current Shape" **MUST BE DISCARDED!!! And MUST NOT BE SAVED TO THE ARRAY/COLLECTION**. **Otherwise, the user can press the left mouse button 100 times and get to the maximum capacity of the DrawPanel!**

ii. When a "Current Shape" drawing starts, but committed at the same point of its starting (this case is identical, in concept, to the previous case of just "pressing" the left mouse button!) – such a "Current Shape" **MUST BE DISCARDED!!! And MUST NOT BE SAVED TO THE ARRAY/COLLECTION**. This case is demonstrated in Demonstrating Video 7.

Enhanced Javadoc Comments:

In this assignment, you will require to comment your code, in a similar "spirit" to the comments provided in the "Calculator" Case Study in Ch. 8. Namely, you will need to also use the {@link Classname}, {@link #methodName} and {@link Classname#Methodname}, in order to link your comments in the program to the Java API Documentation (***at least 5 link examples from each of the 3 link-types above!***).

When you run the Javadoc, you should do it in a similar way to what is described in Ch. 8, slide 256. That is, generating the documentation for the ***private classes/methods (and above!), as well as linking the documentation to the online Java API Documentation***.

Good Luck!