Task 1 README

For this task, I used two separate classes – the HW1\_Applet class is used for the canvas on which we draw, and PopupWindow is used for the menu.

In the PopupWindow class, we have the following components: a draw button and two checkbox groups, one for choosing the shape you want to draw and one for choosing its color.

Each checkbox implements an itemListener, which calls the repaint(); method for the canvas, in order to properly show which checkboxes are selected using drawString within the paint(); method. The if(canvas.getShowed()) call returns a boolean value if the menu is shown.

The draw button implements an actionListener which calls the setPressed(); method for the canvas, a method that sets the Boolean variable “pressed” to true, indicating that an object is ready to be drawn.

The HW1\_Applet class has the following attributes: two Boolean variables, pressed (for when the draw button is pressed) and showed (which becomes *true* when the menu is visible), the PopupWindow w which is by default null (it is initialized in the mouseListener.mouseClicked(MouseEvent me) method), a Random variable r which is used in randomizing the values of the width and length of each shape. The integer variables xcord and ycord are the values of mousePressed(MouseEvent me).getX() and mousePressed(MouseEvent me).getY() and are used alongside width and length in the computation of xc and yc (the coordinates of the center of a shape).

In the init() method we only add the mouseListener to the applet.

In the paint(Graphics g) method we always check if the menu is shown. If this is true, then two Checkbox variables, checkShape and checkColor are initialized in order to determine the shape (either Rectangle, Round Rectangle or Ellipse) and color (Red, Blue or Green) of the object we want to draw. If the draw button is pressed, the pressed variable should be *true.* If this is correct, then the length, width and center of the shape are computed. Then, for each shape option (Rectangle, Round Rectangle or Ellipse), we decide its color based on the selected checkbox.

Finally, mousePressed(mouseEvent me) is the method that computes the coordinates of the mouse’s position, then calls the repaint(); method ONLY IF the menu is visible and the draw button was pressed.

The only defect of the first task is that I cannot change the checkbox options for color and shape after pressing the draw button, but before clicking the canvas to draw an object.

Task 2 README

For this task, I used five separate classes and an interface – the HW1\_Applet2 class is used for the canvas on which we draw, and PopupWindow is used for the menu. MyRectangle, MyRoundRectangle and MyEllipse are used for drawing shapes .They implement the interface MyObject.

The PopupWindow is similar to the previous exercise, but with 2 major differences: the addition of an Edit button and a new checkbox group for edit operations.

MyRectangle, MyRoundRectangle and MyEllipse all have identical constructors. They all receive the xcenter, ycenter, width, length and color values.

The interface MyObject has 11 methods:

* public void paint(Graphics g); is the method used to paint each object inside a canvas. It has 4 possible ways to paint an object:
  + - * Hollow, with its default color
      * Hollow, pink (meaning it is selected for editing)
      * Filled, with its default color
      * Filled, pink (meaning it is selected for editing).
* public int getXC(); returns the X coordinate of the shape’s center.
* public int getYC(); returns the Y coordinate of the shape’s center.
* public int getW(); returns the shape’s width.
* public int getL(); returns the shape’s length.
* Public void setSelected(Boolean a); sets the value of the “selected” attribute to either *true* or *false*, for when a shape is selected or deselected.
* public boolean isSelected(); returns the values of the “selected” attribute.
* public void setFilled(); sets the value of the “filled” attribute to *true* for a shape. This attribute cannot be reverted to being *false* (by design), so the method does not need a boolean parameter.
* public void setColor(Color c); sets the color of a shape to a new color. This method is used mainly during the fill operation.
* public void setXC(int xtr); sets the X coordinate of the shape’s center to a new value. This method is mainly used in the translate operation.
* public void setYC(int ytr); sets the Y coordinate of the shape’s center to a new value. This method is mainly used in the translate operation.

The HW1\_Applet2 class is similar to the one in the previous task, with a few differences in how shapes are stored. It can no longer only draw a single shape. Instead, it stores each shape as an object inside an ArrayList<MyObject> called “objects”, such that any operation made inside the ArrayList can be made on all objects, regardless of the class they belong to. Also, every time an object is added, a counter called objNr increases by 1.

The mousePressed(MouseEvent me) method has undergone dramatic changes. Not only does it contain the controller for drawing objects, it also controls the selection and editing of shapes. Through the variables editPressed an editPressed2 it controls the order in which selection and editing work. The sequence is the following (implying that there is at least one shape drawn on the canvas):

Press the Edit button -> Click on the shapes you want to select -> the shape is highlighted in pink -> press the Edit button a second time -> click on the selected shapes again

Note that it does not matter when the operation checkbox is modified. It can be modified at any moment in time.

The three methods that are called by mousePressed(MouseEvent me) are selectShapes(int xsel, int ysel), fillShapes() and deleteShapes().

The selectShapes(int xsel, int ysel) method receives xsel and ysel as its parameters and returns an ArrayList<MyObject> data structure. xsel and ysel are computed in mousePressed if the edit button has been pressed only once. Inside the selectShapes(int xsel, int ysel), the integer nr copies objNr such that objNr will not be modified by the method, causing permanent damage to further selections. Also, a new, empty ArrayList<MyObject> called “sel” is initialized. The method then goes through the ArrayList<MyObject> objects using a for instruction. It copies each object to a new object “o” and allocates the values of the width, length, X and Y coordinates of the upper-left corner of the object to 4 variables, wobj, lobj, xobj, yobj. If xsel and ysel are within the bounds of the object, the method setSelected(*true*); is called for the object o. Then, o is added to sel and is removed from the original ArrayList of objects.

The method then returns the ArrayList<MyObject> sel. Note that we also have an ArrayList of the same type written as an attribute of the class, called selected, which takes the values of the selectShapes(int xsel, int ysel) method inside the mousePressed(MouseEvent me) method. The temporary ArrayList sel is used to avoid causing damage to the main one.

If the return value of selectedShapes(int xsel, int ysel) is null, then the applet will display a message saying “You must select an object”. This resets the edit button so that the user may properly select an object.

Inside the paint(); method, there are two for instructions. One iterates the “objects” ArrayList. The other iterates the “selected” ArrayList, for when an object is selected and needs to be painted in pink to signal that it is selected. Inside both instructions, we call the method o.paint(g); to paint the objects on the canvas.

The fillShapes() method uses an iterator to go through the selected ArrayList. At each iteration, the method checks which checkbox is selected in the Colors group, calls the methods setFilled() and setSelected(*false*) (finishing an operation always deselects it) for the current object, adds it to the original ArrayList of objects and removes it from the iterator.

The deleteShapes() method uses an iterator to go through the selected ArrayList. At each iteration, it deselects the object (using setSelected(*false*)) and removes it from the iterator, while also decreasing the value of objNr. This means that the objects in the selected ArrayList are not added back to the original ArrayList of objects. They are lost forever.

Note that at the end of fillShapes() and deleteShapes(), editPressed and editPressed2 are set to *false*. This means that the cycle has been reset and, in order to select and edit objects once again, we have to repeat the process written above.

The mouseReleased (MouseEvent me) method works alongside mousePressed(MouseEvent me) in the editing process, but only for the translate operation. It checks if the translate operation checkbox is selected, and if so, calls the translateShapes(int xtr, int ytr) method.

The translateShapes(int xtr, int ytr) method uses an iterator to go through the selected ArrayList. At each iteration it makes a copy of the iterator’s next() method, then sets the X and Y coordinates of the center of the object to xtr-o.getW/2 and ytr-o.getL()/2 respectively. Afterward, the object is deselected (using setSelected(*false*)), added back to the original ArrayList of objects and then it is removed from the iterator.

There are a few problems I have run across in the making of this project.

* The selectShapes(int xsel, int ysel) method does not work with intersections. It only selects objects if the position of the click was a common point between two or more objects.
* The translateShapes(int xtr, int ytr) method works for all selected objects, but with one defect; it centers them all around the same point.
* I cannot change the checkbox options for color, shape and operation after pressing the draw button, but before clicking the canvas to draw an object.