# Splash!

# Projekt: Raster graphics editor

# Parts from the project documentation

## Table of Contents:

# 1. Project overview

Splash! It is a multiplatform, raster graphics editor inspired by such programs as Photoshop, Gimp and Microsoft Paint. The idea while creating the application was to fill the gap between simple and intuitive Paint and advanced and crude Gimp. The application was written in Java using the Swing library. The main point though, was to learn and use in practice many different design patterns.

Splash! has been designed to offer the following features:

a) Drawing tools:

- pencil
- brush
- line
- broken line
- fill
- foreground color change
- background color change
- rectangle
- triangle
- oval
- spray
- selecting and cutting (Paint-like)
- eraser
- offset (layers)
- pipette
- text
- zoom
- drawing size change

b) Image:

- size change
- scaling
- offset change

c) Layers:

- creating
- moving
- merging
- deleting
- visibility change
- opacity change
- name change

d) Image and layer filters:

- color inversion
- rotation (angle)
- brightness (percentage)

- contrast (percentage)
- blur
- sharpness
- white balance

e) I/O:

- opening and saving files with the following extensions:
- jpg/jpeg
- png
- bmp
- gif
- slh (app specific extension – allows to save current sheet state with layers).

f) Other:

- undo/redo feature for each layer
- menu and other shortcuts
- UI compatible with multiple displays
- multiple sheets
- foreground toolbars

## 2. Design patterns use

1. MVC
2. Observer
3. Strategy
4. Memento
5. Chain of Responsibility
6. Template method
7. Factory (Simple Factory)
8. Prototype
9. Singleton
10. Composite
11. Adapter + Iterator

LEGEND:
Creational patterns
Structural patterns
Behavioral patterns
Architectural patterns

# 3. Patterns usage description and UML diagrams

## 1. MVC (Model View Controller)

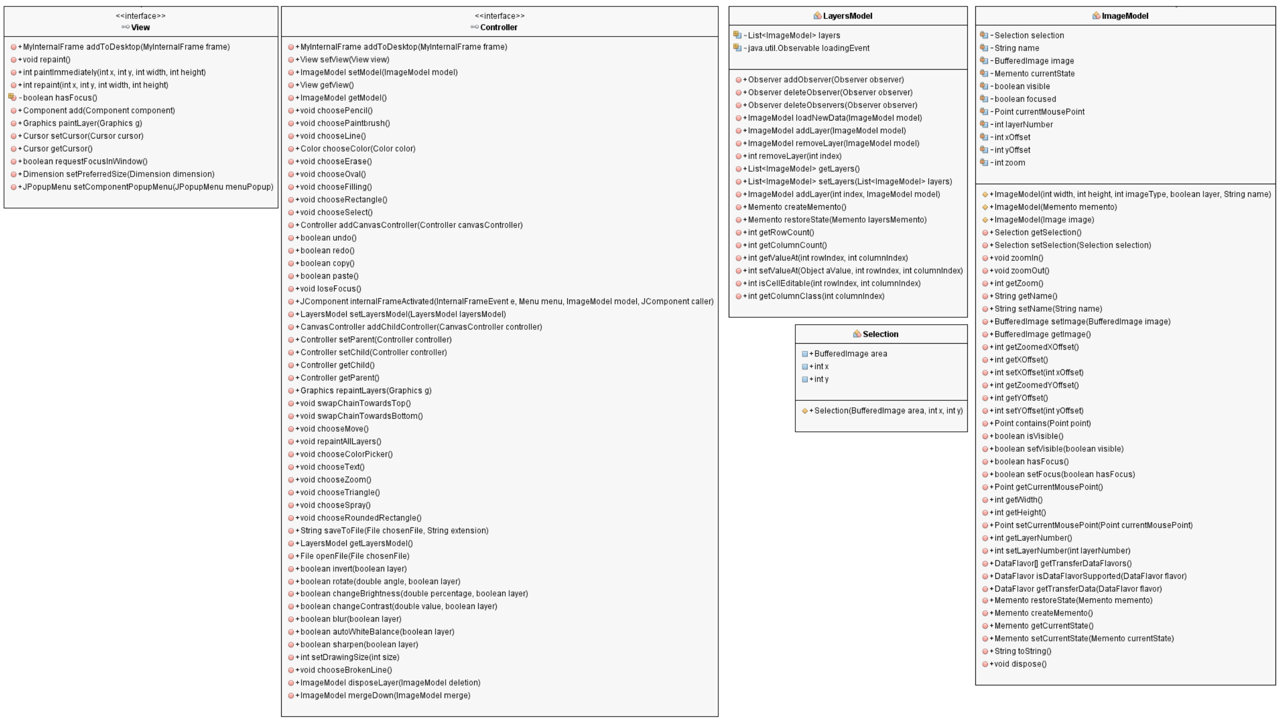
a) Model

I. ImageModel – represents the data of a single layer with BufferedImage and other minor elements.

II. LayersModel – represents several layers (ImageModel).

b) View – this layer consists of views inheriting from the Swing library classes and implementing the View interface. Here are, among others: MainView (main view), InfoPanel (displays the current cursor position), LayersPanel (displays layers of current sheet), Menu, ToolPanel, etc.

c) Controller – his layer includes classes implementing the Controller interface, which in turn inherits from typical Swing interfaces such as MouseMotionListener or MouseListener. These objects are usually added as listeners in views. Two classes of controllers were distinguished: MainController (main controller) and CanvasController (controller responsible for handling a single layer).



```
<<interface>>
View
+MyInternalFrame addToDesktop(MyInternalFrame frame)
+void repaint()
+int paintImmediately(int x, int y, int width, int height)
+int repaint(int x, int y, int width, int height)
~boolean hasFocus()
+Component add(Component component)
+Graphics paintLayer(Graphics g)
+Cursor setCursor(Cursor cursor)
+Cursor getCursor()
+boolean requestFocusInWindow()
+Dimension setPreferredSize(Dimension dimension)
+JPopupMenu setComponentPopupMenu(JPopupMenu menuPopup)
```

```
<<interface>>
Controller
+MyInternalFrame addToDesktop(MyInternalFrame frame)
+View setView(View view)
+ImageModel setModel(ImageModel model)
+View getView()
+ImageModel getModel()
+void choosePencil()
+void choosePaintbrush()
+void chooseLine()
+Color chooseColor(Color color)
+void chooseErase()
+void chooseOval()
+void chooseFilling()
+void chooseRectangle()
+void chooseSelect()
+Controller addCanvasController(Controller canvasController)
+boolean undo()
+boolean redo()
+boolean copy()
+boolean paste()
+void loseFocus()
+JComponent internalFrameActivated(InternalFrameEvent e, Menu menu, ImageModel model, JComponent caller)
+LayersModel setLayersModel(LayersModel layersModel)
+CanvasController addChildController(CanvasController controller)
+Controller setParent(Controller controller)
+Controller setChild(Controller controller)
+Controller getChild()
+Controller getParent()
+Graphics repaintLayers(Graphics g)
+void swapChainTowardsTop()
+void swapChainTowardsBottom()
+void chooseMove()
+void repaintAllLayers()
+void chooseColorPicker()
+void chooseText()
+void chooseZoom()
+void chooseTriangle()
+void chooseSpray()
+void chooseRoundedRectangle()
+String saveToFile(File chosenFile, String extension)
+LayersModel getLayersModel()
+File openFile(File chosenFile)
+boolean invert(boolean layer)
+boolean rotate(double angle, boolean layer)
+boolean changeBrightness(double percentage, boolean layer)
+boolean changeContrast(double value, boolean layer)
+boolean blur(boolean layer)
+boolean autoWhiteBalance(boolean layer)
+boolean sharpen(boolean layer)
+int setDrawingSize(int size)
+void chooseBrokenLine()
+ImageModel disposeLayer(ImageModel deletion)
+ImageModel mergeDown(ImageModel merge)
```

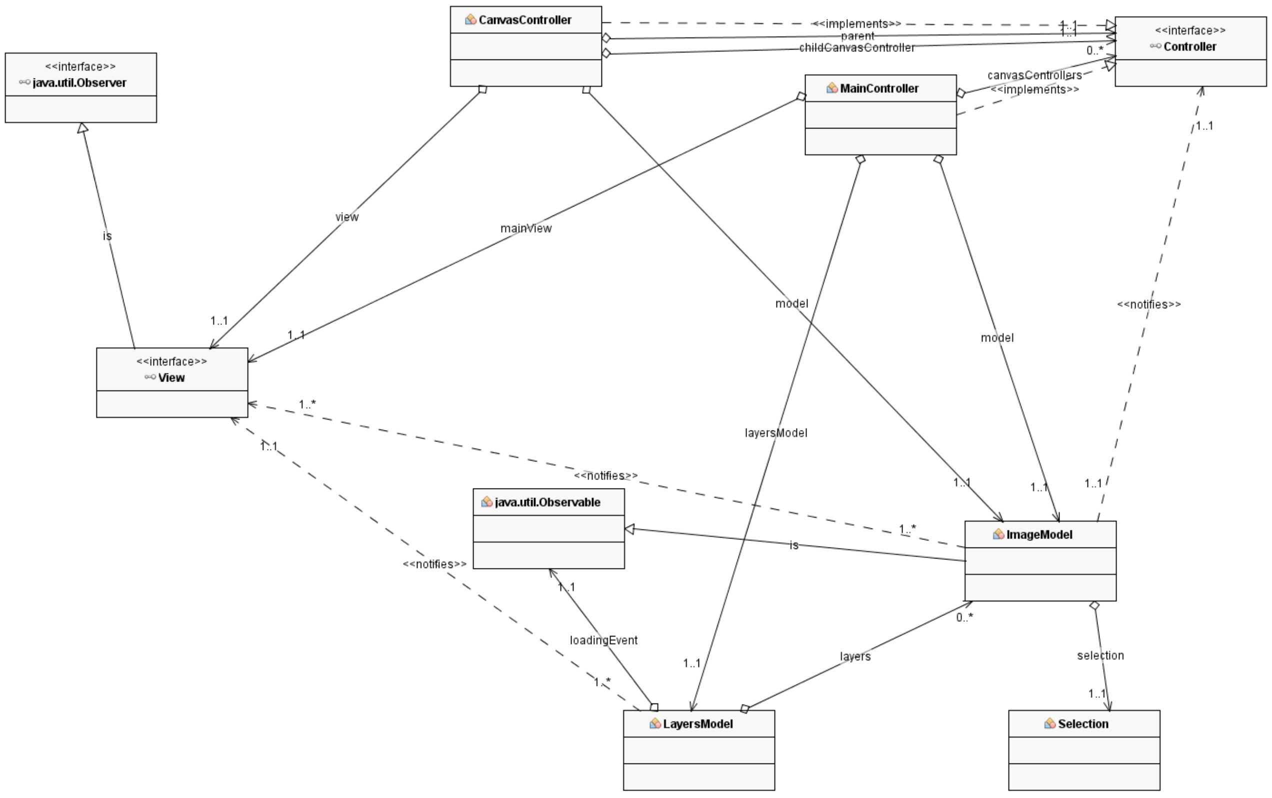
```
LayersModel
~List<ImageModel> layers
~java.util.Observable loadingEvent
+Observer addObserver(Observer observer)
+Observer deleteObserver(Observer observer)
+Observer deleteObservers(Observer observer)
+ImageModel loadNewData(ImageModel model)
+ImageModel addLayer(ImageModel model)
+ImageModel removeLayer(ImageModel model)
+int removeLayer(int index)
+List<ImageModel> getLayers()
+List<ImageModel> setLayers(List<ImageModel> layers)
+ImageModel addLayer(int index, ImageModel model)
+Memento createMemento()
+Memento restoreState(Memento layersMemento)
+int getRowCount()
+int getColumnCount()
+int getValueAt(int rowIndex, int columnIndex)
+int setValueAt(Object aValue, int rowIndex, int columnIndex)
+int isCellEditable(int rowIndex, int columnIndex)
+int getColumnClass(int columnIndex)
```

```
Selection
+BufferedImage area
+int x
+int y
+Selection(BufferedImage area, int x, int y)
```

```
ImageModel
-Selection selection
-String name
-BufferedImage image
-Memento currentState
-boolean visible
-boolean focused
-Point currentMousePoint
-int layerNumber
-int xOffset
-int yOffset
-int zoom
+ImageModel(int width, int height, int imageType, boolean layer, String name)
+ImageModel(Memento memento)
+ImageModel(Image image)
+Selection getSelection()
+Selection setSelection(Selection selection)
+void zoomIn()
+void zoomOut()
+int getZoom()
+String getName()
+String setName(String name)
+BufferedImage setImage(BufferedImage image)
+BufferedImage getImage()
+int getZoomedXOffset()
+int getXOffset()
+int setXOffset(int xOffset)
+int getZoomedYOffset()
+int getYOffset()
+int setYOffset(int yOffset)
+Point contains(Point point)
+boolean isVisible()
+boolean setVisible(boolean visible)
+boolean hasFocus()
+boolean setFocus(boolean hasFocus)
+Point getCurrentMousePoint()
+int getWidth()
+int getHeight()
+Point setCurrentMousePoint(Point currentMousePoint)
+int getLayerNumber()
+int setLayerNumber(int layerNumber)
+DataFlavor[] getTransferDataFlavors()
+DataFlavor isDataFlavorSupported(DataFlavor flavor)
+DataFlavor getTransferData(DataFlavor flavor)
+Memento restoreState(Memento memento)
+Memento createMemento()
+Memento getCurrentState()
+Memento setCurrentState(Memento currentState)
+String toString()
+void dispose()
```

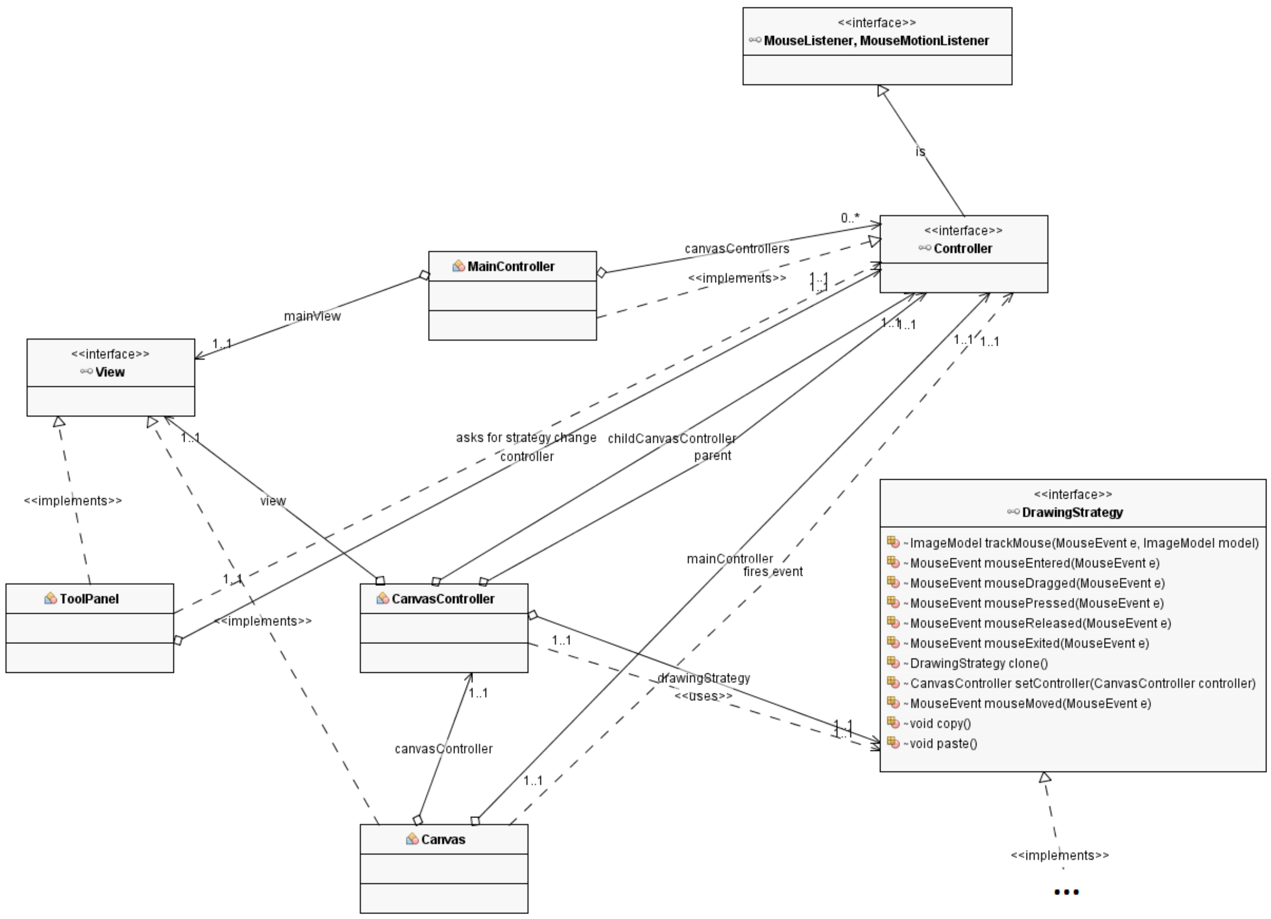
*Picture 1: Pattern – MVC. Interfaces and classes*

2. Observer – the observed object (model) notifies the views about changes.



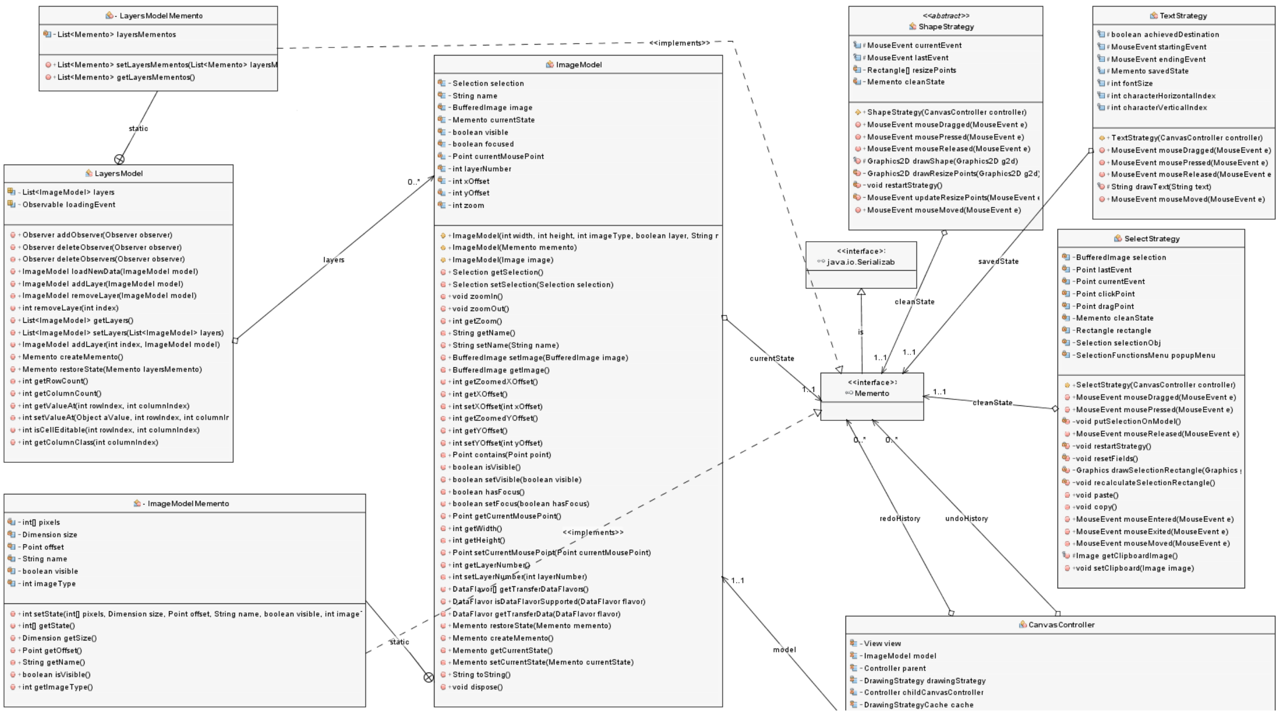
*Picture 2: Pattern – Observer. Communication choice in the MVC pattern*

3. Strategy – this pattern in the project is used for various forms of the drawing algorithm selected from the tool panel such as: pencil, brush, drawing figures, selection, etc.



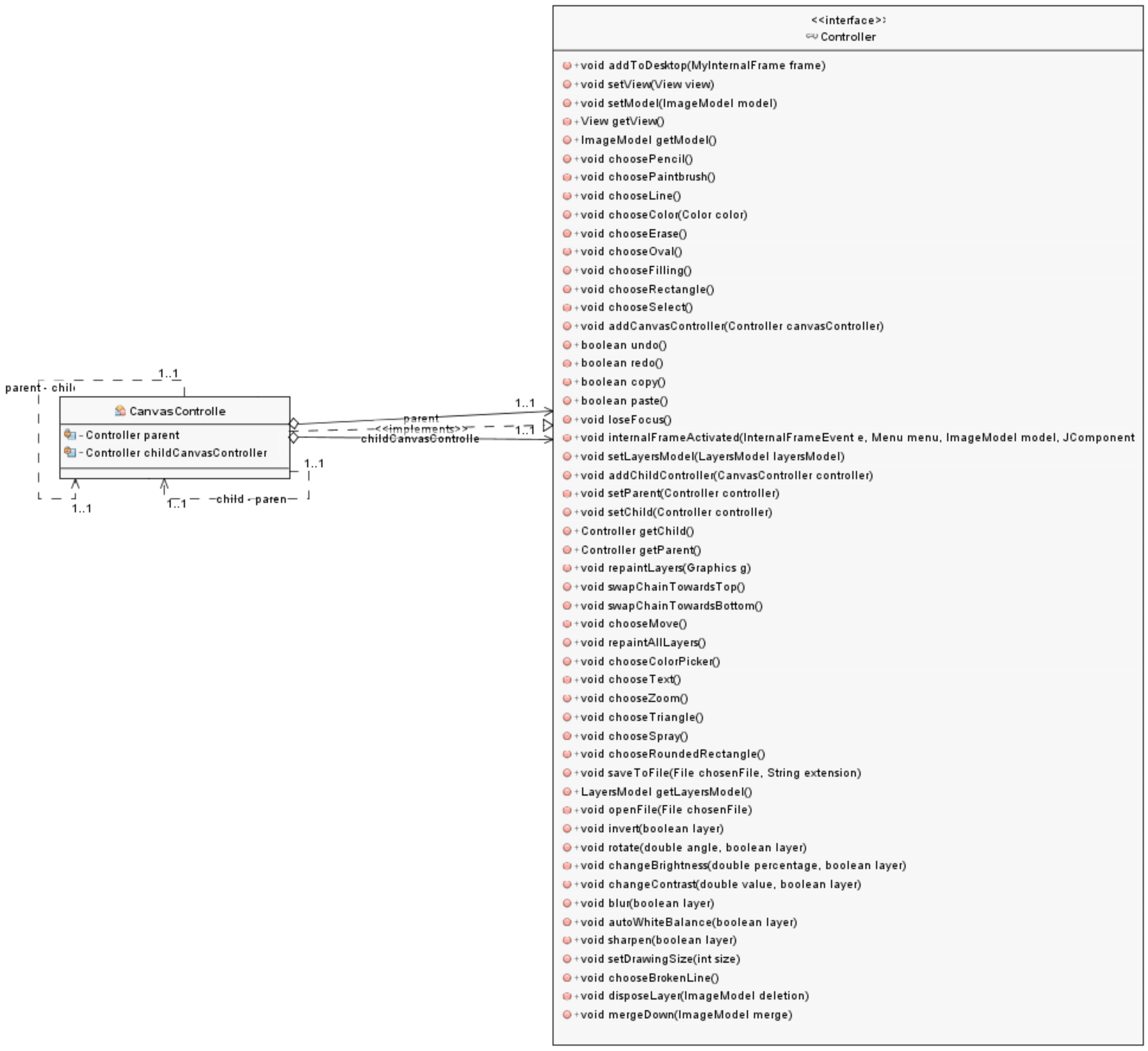
*Picture 3: Pattern – Strategy. ToolPanel informs of the Strategy change caused by the user (drawing tool selection), Canvas sends events based on which context (CanvasController) uses the correct Strategy. For strategy class hierarchy – see the Prototype pattern*

4. Memento – deals with saving the state of the model and makes it available to the outside without violating the rules of hermetization, which allows to implement the mechanism "undo / redo" and save the state of the application.



*Picture 4: Wzorzec Memento. Caretaker - Controller; Originator - LayersModel, ImageModel; Memento – narrow interface; Private methods of inner static classes act as a wide interface for the Originator. Static classess are not connected with static chain during serialization process*

5. Chain of responsibility – the undo and redo, copying and pasting commands follow the path from the main controller, through the controllers for which the views have focus, ending with the controller for which the model is active, where the action is performed.

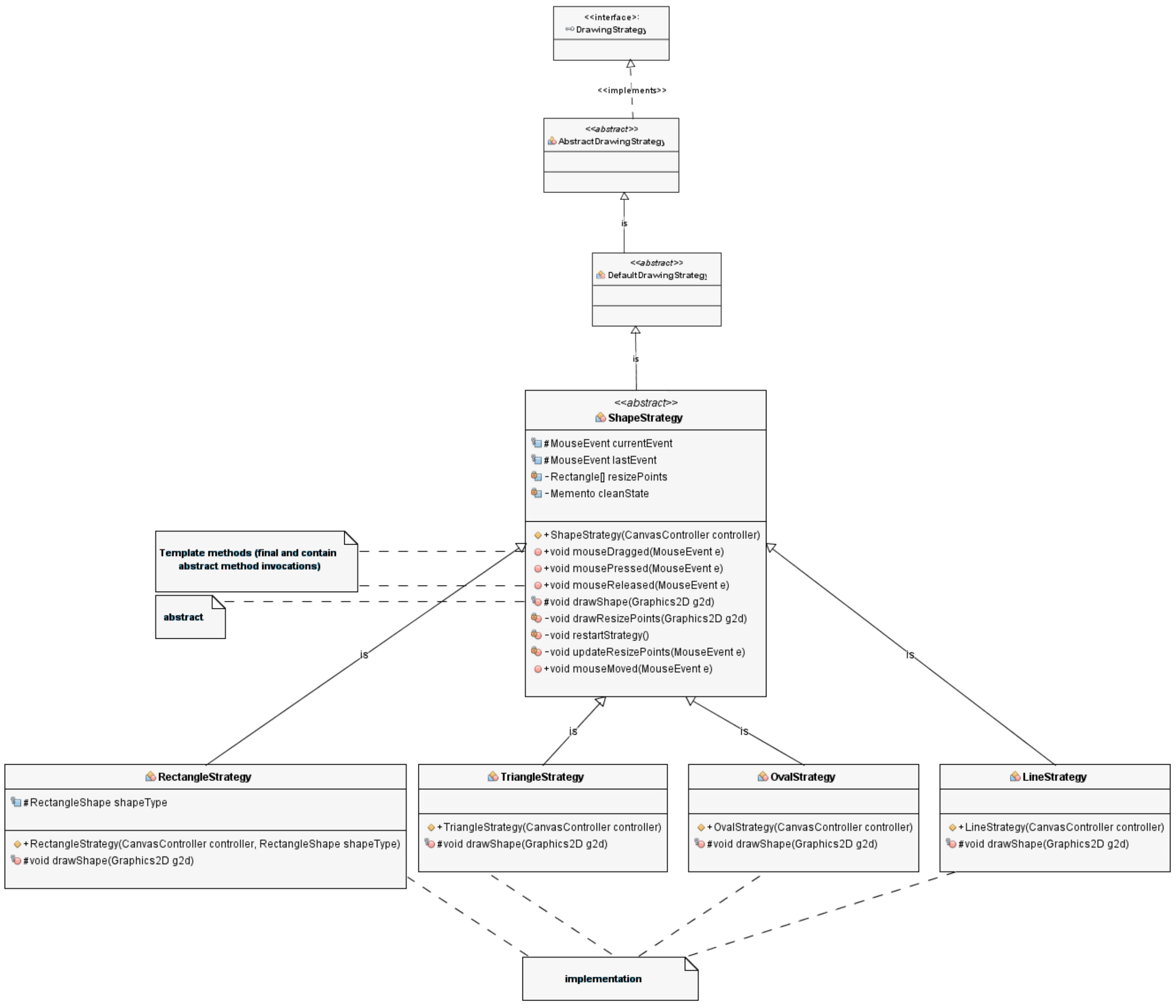


```
- Controller parent
- Controller childCanvasController
```

```
+void addToDesktop(MyInternalFrame frame)
+void setView(View view)
+void setModel(ImageModel model)
+View getView()
+ImageModel getModel()
+void choosePencil()
+void choosePaintbrush()
+void chooseLine()
+void chooseColor(Color color)
+void chooseErase()
+void chooseOval()
+void chooseFilling()
+void chooseRectangle()
+void chooseSelect()
+void addCanvasController(Controller canvasController)
+boolean undo()
+boolean redo()
+boolean copy()
+boolean paste()
+void loseFocus()
+void internalFrameActivated(InternalFrameEvent e, Menu menu, ImageModel model, JComponent
+void setLayersModel(LayersModel layersModel)
+void addChildController(CanvasController controller)
+void setParent(Controller controller)
+void setChild(Controller controller)
+Controller getChild()
+Controller getParent()
+void repaintLayers(Graphics g)
+void swapChainTowardsTop()
+void swapChainTowardsBottom()
+void chooseMove()
+void repaintAllLayers()
+void chooseColorPicker()
+void chooseText()
+void chooseZoom()
+void chooseTriangle()
+void chooseSpray()
+void chooseRoundedRectangle()
+void saveToFile(File chosenFile, String extension)
+LayersModel getLayersModel()
+void openFile(File chosenFile)
+void invert(boolean layer)
+void rotate(double angle, boolean layer)
+void changeBrightness(double percentage, boolean layer)
+void changeContrast(double value, boolean layer)
+void blur(boolean layer)
+void autoWhiteBalance(boolean layer)
+void sharpen(boolean layer)
+void setDrawingSize(int size)
+void chooseBrokenLine()
+void disposeLayer(ImageModel deletion)
+void mergeDown(ImageModel merge)
```

*Picture 5: Pattern - Chain of responsibility. Each object contains a reference to parent and child. Most methods are called in the chain on the focused model.*

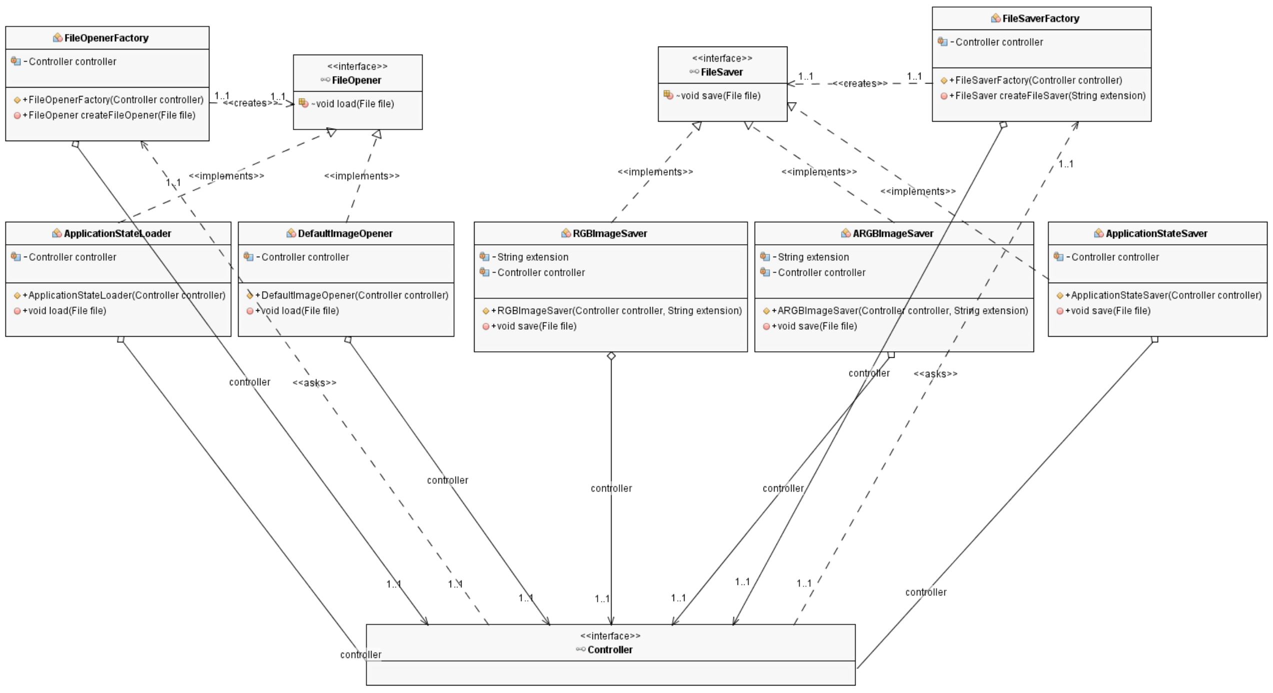
6. Template method – the drawing scheme of the figures is quite similar, it consists of, among others, drawing small squares at the ends of the figures (to resize by dragging) and handling drawing. In this algorithm, the variable element is drawing the figure. Using the template template method in derived classes only the implementation of the figure drawing method is given, which allows to easily and quickly expand the strategy by successive figures.



<<interface>>
DrawingStrategy
<<implements>>
<<abstract>>
AbstractDrawingStrategy
is
<<abstract>>
DefaultDrawingStrategy
is
<<abstract>>
ShapeStrategy
#MouseEvent currentEvent
#MouseEvent lastEvent
-Rectangle[] resizePoints
-Memento cleanState
+ShapeStrategy(CanvasController controller)
+void mouseDragged(MouseEvent e)
+void mousePressed(MouseEvent e)
+void mouseReleased(MouseEvent e)
#void drawShape(Graphics2D g2d)
-void drawResizePoints(Graphics2D g2d)
-void restartStrategy()
-void updateResizePoints(MouseEvent e)
+void mouseMoved(MouseEvent e)
Template methods (final and contain abstract method invocations)
abstract
is
is
is
is
RectangleStrategy
#RectangleShape shapeType
+RectangleStrategy(CanvasController controller, RectangleShape shapeType)
#void drawShape(Graphics2D g2d)
TriangleStrategy
+TriangleStrategy(CanvasController controller)
#void drawShape(Graphics2D g2d)
OvalStrategy
+OvalStrategy(CanvasController controller)
#void drawShape(Graphics2D g2d)
LineStrategy
+LineStrategy(CanvasController controller)
#void drawShape(Graphics2D g2d)
implementation

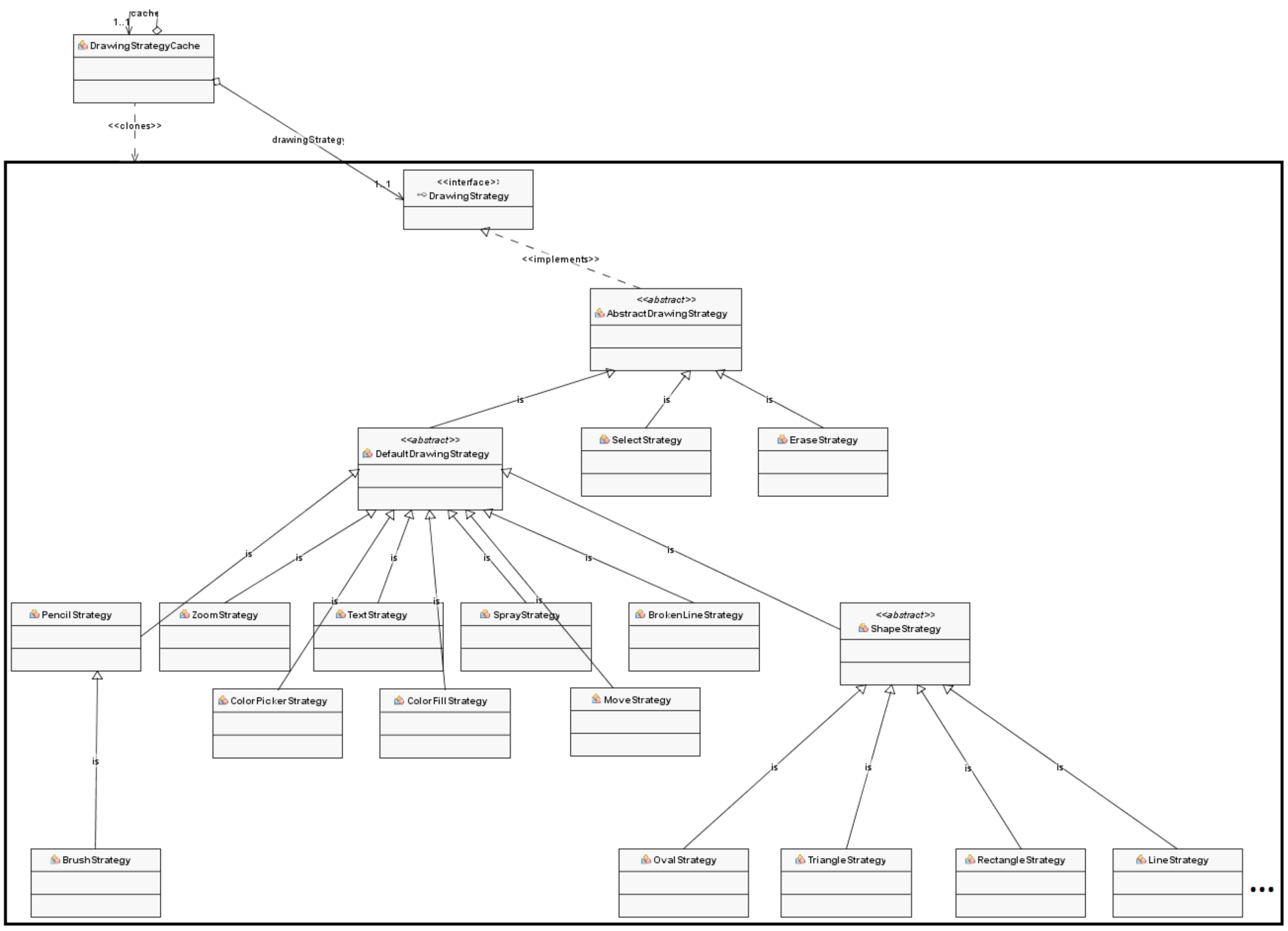
*Picture 6: Pattern – Template method. Abstract method drawShape is called in (finalnych) template methods mouseDragged and mouseReleased. Derived classess implement this method.*

7. Factory (Simple Factory) – when opening an image from a file, it may be interpreted differently depending on the extension (ARGB - png, gif, RGB - bmp, jpg; file containing the application state), therefore different classes will be needed to load the data into the appropriate format and vice versa - write to it. Using the "Factory" pattern, it will be possible to assign the appropriate load / save class based on the file extension.



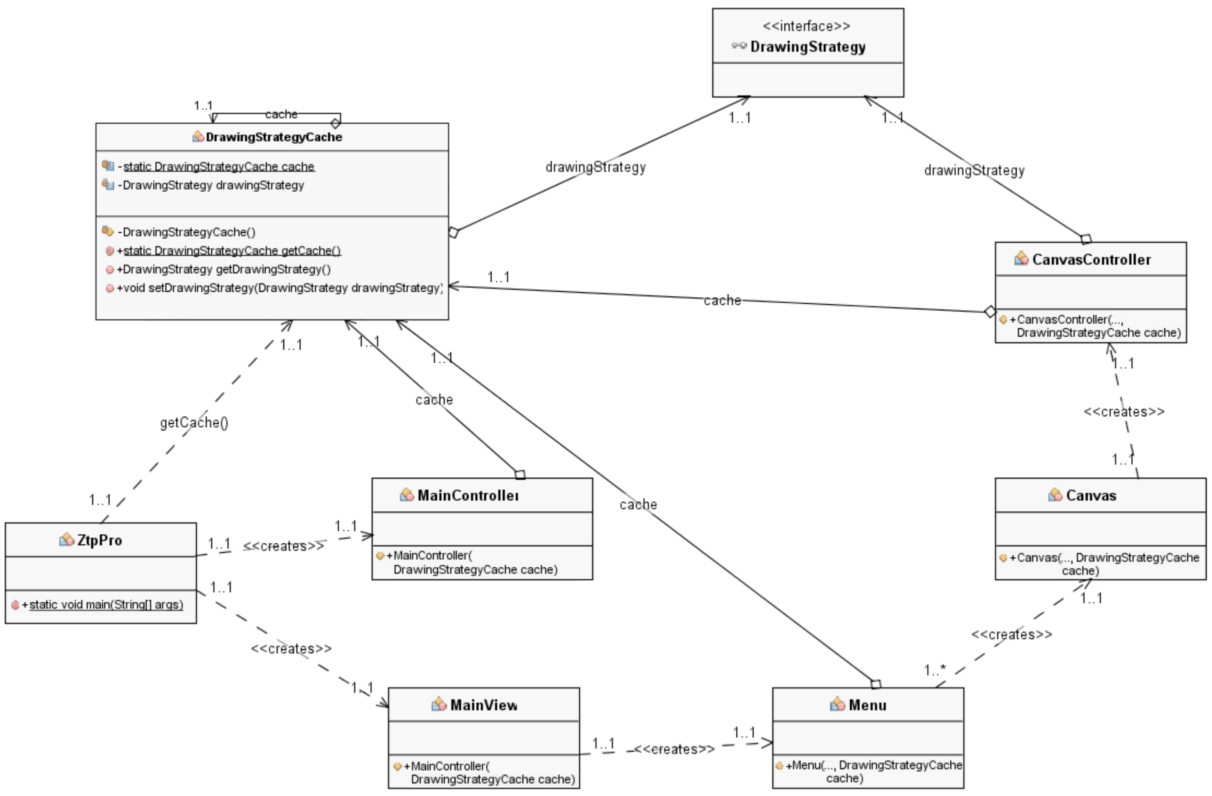
*Picture 7: Pattern – Factory (Simple Factory). On behalf of classes implementing the Controller interface, the class that opens or saves the given file type (image) is returned based on the file or its extension.*

8. Prototype – when creating a new controller, it should have the same strategy so that the same drawing options do not differ between the sheets and layers. The strategy is created on the basis of the prototype, but only its controller is changed.



*Picture 8: Pattern - Prototype. Currently selected Strategy is cloned during the retrival from the cache.*

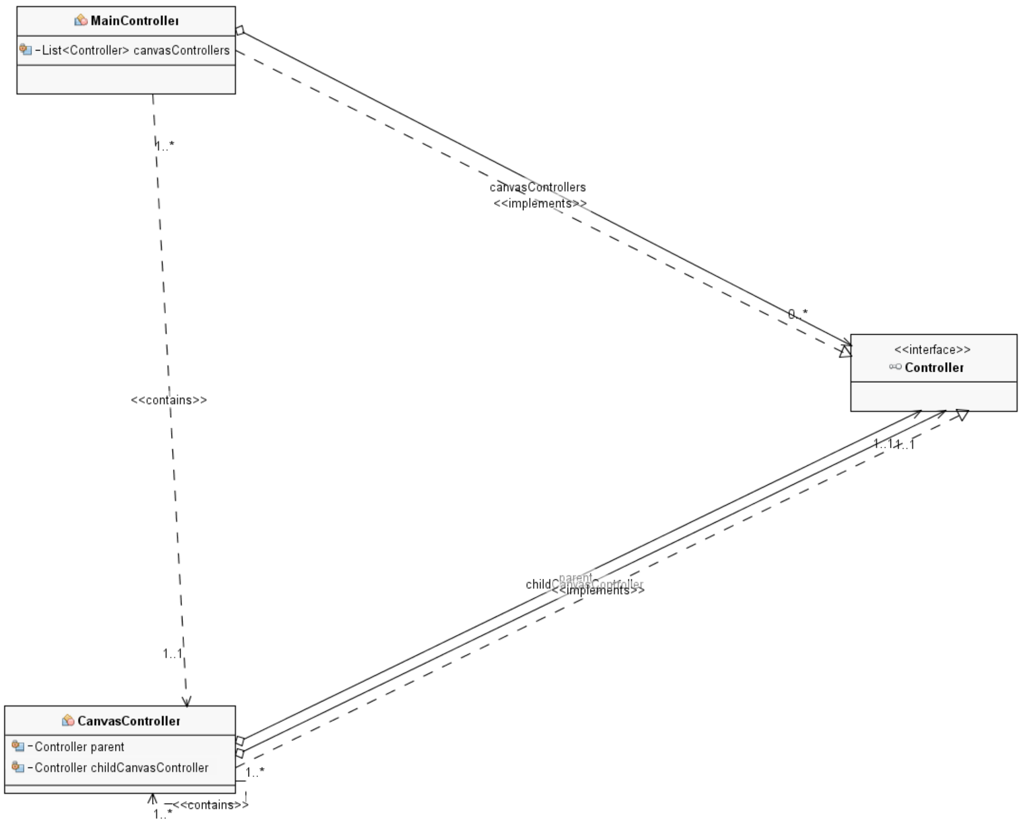
9. Singleton – deals with storing the strategy prototype (one type of strategy for the whole application), the so-called "cache".



<<interface>>
DrawingStrategy
1..1
cache
DrawingStrategyCache
-static DrawingStrategyCache cache
-DrawingStrategy drawingStrategy
-DrawingStrategyCache()
+static DrawingStrategyCache getCache()
+DrawingStrategy getDrawingStrategy()
+void setDrawingStrategy(DrawingStrategy drawingStrategy)
drawingStrategy
1..1
1..1
drawingStrategy
CanvasController
+CanvasController(..., DrawingStrategyCache cache)
1..1
cache
1..1
1..1
1..1
getCache()
cache
1..1
<<creates>>
1..1
1..1
MainController
+MainController( DrawingStrategyCache cache)
cache
Canvas
+Canvas(..., DrawingStrategyCache cache)
ZtpPro
+static void main(String[] args)
1..1
<<creates>>
1..1
1..1
<<creates>>
1..1
<<creates>>
1..*
1..1
MainView
+MainController( DrawingStrategyCache cache)
1..1
<<creates>>
1..1
Menu
+Menu(..., DrawingStrategyCache cache)

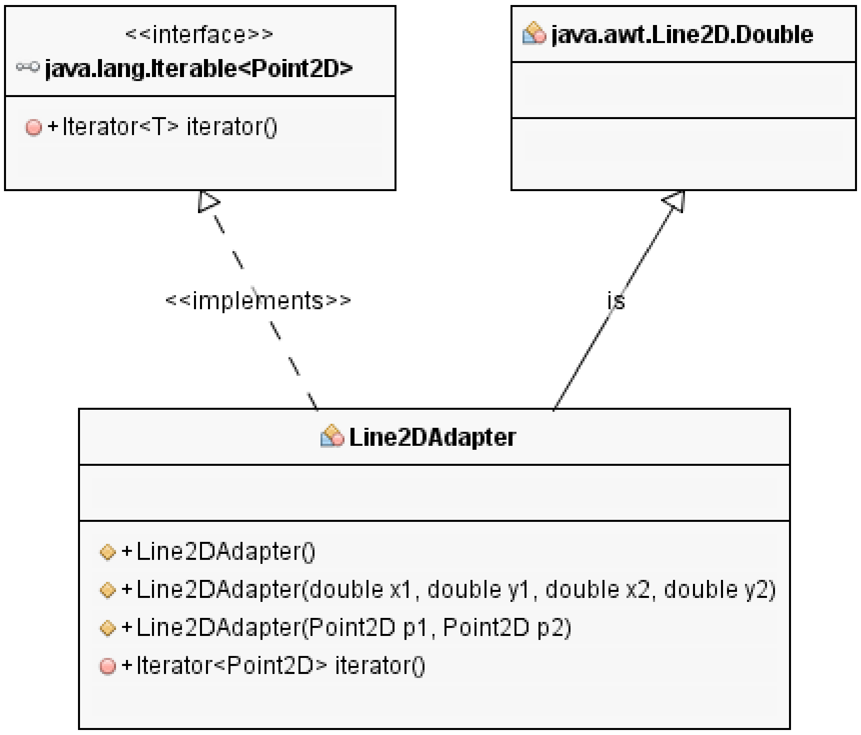
*Picture 9: Pattern - Singleton. Responsible for allowing only one instance of the object in the application. Global access is not used which is shown by a chain of DrawingStrategyCache creation and passing.*

10. Composite – visible especially in the Controller layer. The main controller contains layer controllers (one for each sheet), which in turn contain a reference to the controller from the upper layer.



*Picture 10: Pattern - Composite.*

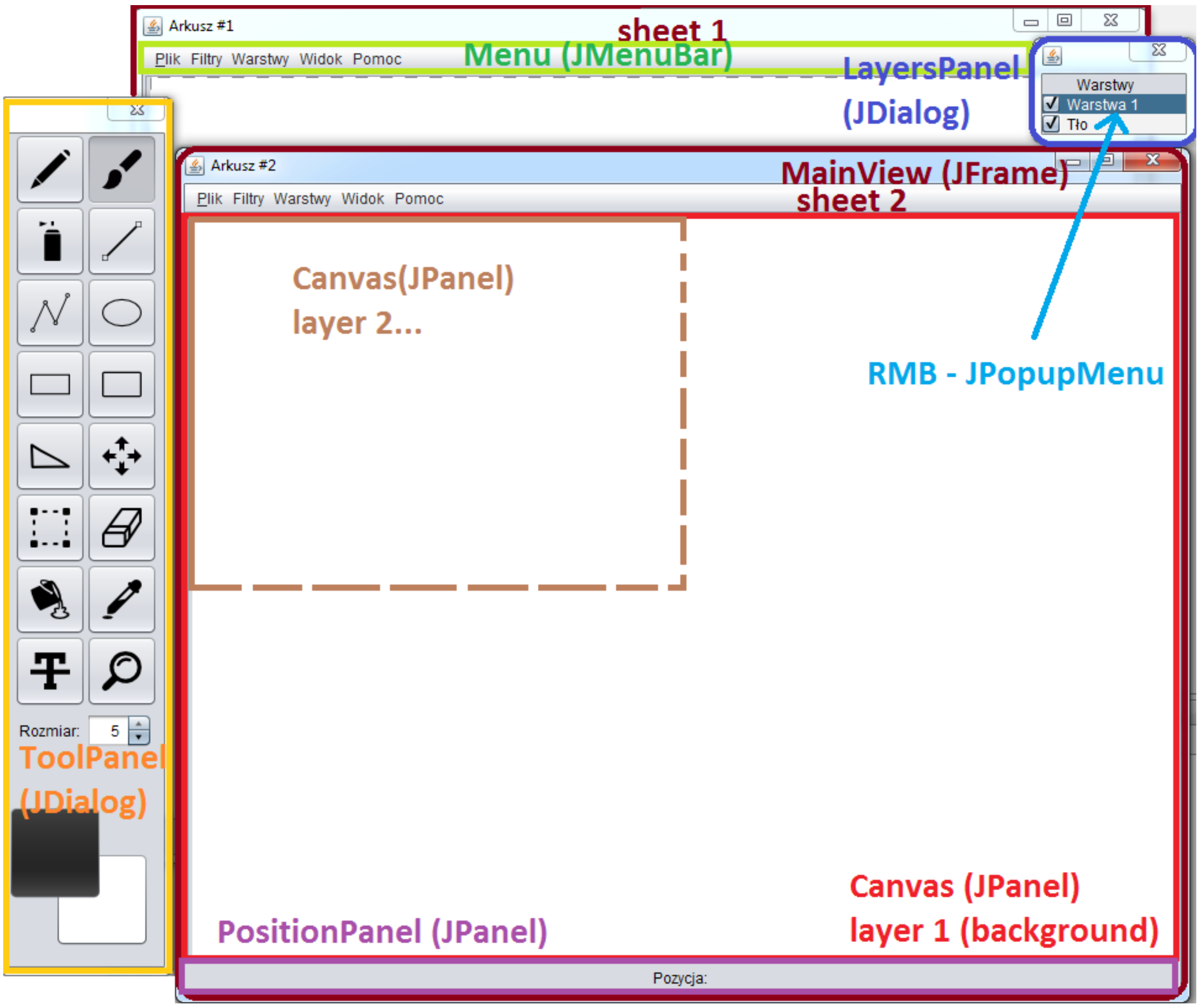
11. Adapter + Iterator – when drawing with a brush, the information about the mouse's route may not be continuous. In that case, the circles drawn by the brush will not be connected to each other and we will not get the continous effect. It is therefore necessary to designate a line between two points and draw a circle for each point on this line. Thanks to the use of the adapter, it is possible to adapt the existing class (Line2D) to the new interface containing the return method created for the iterator line by points.



<<interface>>
java.lang.Iterable<Point2D>
+Iterator<T> iterator()
java.awt.Line2D.Double
<<implements>>
is
Line2DAdapter
+Line2DAdapter()
+Line2DAdapter(double x1, double y1, double x2, double y2)
+Line2DAdapter(Point2D p1, Point2D p2)
+Iterator<Point2D> iterator()

*Picture 11: Pattern - Adapter and Iterator. The usage allows for easy access the consecutive intrapolated points which follow the low reporting mouse drag position frequency.*

## 4. GUI Prototype (View layer)



*Picture 12: GUI Prototype*