**build.gradle**

plugins **{** id 'java'  
 id 'info.solidsoft.pitest' version '1.7.0'  
**}**group = 'org.PaintProgram'  
version = '1.0-SNAPSHOT'  
  
repositories **{** mavenCentral()  
**}**dependencies **{** // Use JUnit test framework  
 testImplementation 'junit:junit:4.13.2'  
 testImplementation 'org.codehaus.groovy:groovy-all:3.0.8'  
 testImplementation 'org.spockframework:spock-core:2.0-groovy-3.0'  
 testImplementation 'pl.pragmatists:JUnitParams:1.1.1'  
**}**test **{** useJUnit()  
**}**pitest **{** targetClasses = ['org.PaintProgram.\*']  
 pitestVersion = '1.7.0'  
 threads = 4  
 outputFormats = ['XML', 'HTML']  
 timestampedReports = false  
**}**java **{** toolchain **{** languageVersion = JavaLanguageVersion.*of*(17)  
 **}  
}**

**PaintProgram.java**

// Package des Programms  
package org.PaintProgram;  
  
//Hauptklasse mit Main-Methode  
public class PaintProgram {  
 //Main-Methode  
 public static void main(String[] args) {  
 //Starten des Programms  
 new Frame("Zeichenprogramm");  
 }  
}

**Frame.java**

package org.PaintProgram;  
  
//importieren der benötigten Klassen  
import javax.swing.\*;  
import javax.swing.filechooser.FileNameExtensionFilter;  
import java.awt.\*;  
import java.awt.event.\*;  
import java.io.File;  
import java.io.IOException;  
import java.nio.file.Files;  
import java.nio.file.Path;  
import static java.awt.event.KeyEvent.\*;  
import static java.lang.Float.*parseFloat*;  
import static java.lang.Integer.*parseInt*;  
  
//Klasse, die das Anwendungsfenster darstellt  
public class Frame extends JFrame {  
 //Klassenvariablen  
 private final PaintPanel paintPanel;  
 private final JFileChooser fileChooser;  
 private JToolBar toolBar;  
 private JPanel colorPanel;  
 private JTextField strokeField;  
 private ButtonGroup colorGroup, toolGroup;  
 private JToggleButton brushButton, lineButton, rectangleButton, ellipseButton, eraserButton, blackButton, redButton, blueButton, yellowButton, whiteButton, cyanButton, greenButton, magentaButton, orangeButton, pinkButton, lightgrayButton, grayButton, darkgrayButton;  
 private boolean leftMouseButtonIsPressed;  
 private File outputFile;  
 //Konstanten für Werkzeuge, um Tippfehler z.B. bei den Action-commands zu vermeiden  
 private final String brushString = "brush", lineString = "line", rectangleString = "rectangle", ellipseString = "ellipse", eraserString = "eraser";  
  
 //Konstruktor  
 public Frame (String frameTitel) {  
 //Erstellen des Fensters  
 super(frameTitel);  
 //Größe des Fensters - Ermitteln der Auflösung des Betriebssystems  
 GraphicsDevice graphicsDevice = GraphicsEnvironment.*getLocalGraphicsEnvironment*().getDefaultScreenDevice();  
 //Setzen der Standard-Fenstergröße auf die Hälfte der Bildschirmgröße  
 this.setSize(graphicsDevice.getDisplayMode().getWidth()/2, graphicsDevice.getDisplayMode().getHeight()/2);  
 //Maximiert starten  
 this.setExtendedState(*MAXIMIZED\_BOTH*);  
 //Layout des Fensters  
 this.setLayout(new BorderLayout());  
 //Beenden des Programms beim Schließen  
 this.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
  
 //Erzeugen der Zeichenfläche mit Standardgröße  
 paintPanel = new PaintPanel(1600, 900);  
 //PreferredSize muss gesetzt werden, damit das Panel auch bei kleinerem Fenster seine Größe behält  
 paintPanel.setPreferredSize(new Dimension(1600, 900));  
 //Hinzufügen von Scrollbars  
 JScrollPane scrollPane = new JScrollPane(paintPanel);  
 //Hinzufügen des Panels zum Fenster  
 this.add(scrollPane, BorderLayout.*CENTER*);  
 //erstellen der MouseListener  
 paintPanel.addMouseListener(new MouseListener());  
 paintPanel.addMouseMotionListener(new MouseMotionListener());  
  
 //Methoden zum Erstellen der Menübar und der Symbolleiste inkl. interaktiver Elemente  
 createMenuBar();  
 createSymbolBar();  
  
 //Sichtbarkeit des Fensters  
 this.setVisible(true);  
  
 //Erstellen des FileChoosers  
 fileChooser = new JFileChooser();  
 //Standardfilter auf JPG-Dateien  
 fileChooser.setFileFilter(new FileNameExtensionFilter("JPG","jpg"));  
 //Standardverzeichnis beim Speichern/Laden im Projektverzeichnis im Ordner "savedPictures"  
 fileChooser.setCurrentDirectory(new File ("savedPictures"));  
 }  
  
 //Erstellen der MenuBar mit den einzelnen Untermenüs und Items inkl. Symbolen und ShortCuts mit lokalen Variablen  
 //Für die Items werden gesetzt: Text, Shortcut, Icon, ActionCommand; ActionListener werden erstellt  
 //mit F10 kann die MenuBar auch mit der Tastatur gesteuert werden  
 private void createMenuBar() {  
 JMenuBar menuBar = new JMenuBar();  
 JMenu fileMenu = new JMenu("Datei");  
 menuBar.add(fileMenu);  
 JMenu toolMenu = new JMenu("Werkzeuge");  
 menuBar.add(toolMenu);  
  
 JMenuItem newSameSizeItem = new JMenuItem("Neu");  
 newSameSizeItem.setIcon(new ImageIcon("icons/menu/Add16.gif"));  
 newSameSizeItem.setAccelerator(KeyStroke.*getKeyStroke*('N', InputEvent.*CTRL\_DOWN\_MASK*));  
 fileMenu.add(newSameSizeItem);  
 newSameSizeItem.setActionCommand("newSameSize");  
 newSameSizeItem.addActionListener(new ButtonAndTextFieldListener());  
  
 JMenuItem newOtherSizeItem = new JMenuItem("Neu (Blattgroeße anpassen)");  
 newOtherSizeItem.setIcon(new ImageIcon("icons/menu/Add16.gif"));  
 newOtherSizeItem.setAccelerator(KeyStroke.*getKeyStroke*('N', InputEvent.*CTRL\_DOWN\_MASK* | InputEvent.*SHIFT\_DOWN\_MASK*));  
 fileMenu.add(newOtherSizeItem);  
 newOtherSizeItem.setActionCommand("newOtherSize");  
 newOtherSizeItem.addActionListener(new ButtonAndTextFieldListener());  
  
 JMenuItem loadItem = new JMenuItem("Laden");  
 loadItem.setAccelerator(KeyStroke.*getKeyStroke*('L', InputEvent.*CTRL\_DOWN\_MASK*));  
 fileMenu.add(loadItem);  
 loadItem.setActionCommand("load");  
 loadItem.addActionListener(new ButtonAndTextFieldListener());  
  
 JMenuItem saveItem = new JMenuItem("Speichern");  
 saveItem.setIcon(new ImageIcon("icons/menu/save16.gif"));  
 saveItem.setAccelerator(KeyStroke.*getKeyStroke*('S', InputEvent.*CTRL\_DOWN\_MASK*));  
 fileMenu.add(saveItem);  
 saveItem.setActionCommand("save");  
 saveItem.addActionListener(new ButtonAndTextFieldListener());  
  
 JMenuItem saveAsItem = new JMenuItem("Speichern unter");  
 saveAsItem.setIcon(new ImageIcon("icons/menu/save16.gif"));  
 saveAsItem.setAccelerator(KeyStroke.*getKeyStroke*('S', InputEvent.*CTRL\_DOWN\_MASK* | InputEvent.*SHIFT\_DOWN\_MASK*));  
 fileMenu.add(saveAsItem);  
 saveAsItem.setActionCommand("saveAs");  
 saveAsItem.addActionListener(new ButtonAndTextFieldListener());  
  
 JMenuItem closeItem = new JMenuItem("Beenden");  
 fileMenu.add(closeItem);  
 closeItem.setActionCommand("close");  
 closeItem.addActionListener(new ButtonAndTextFieldListener());  
  
 JMenuItem brushItem = new JMenuItem("Pinsel");  
 brushItem.setAccelerator(KeyStroke.*getKeyStroke*('P', InputEvent.*ALT\_DOWN\_MASK*));  
 toolMenu.add(brushItem);  
 brushItem.setActionCommand(brushString);  
 brushItem.addActionListener(new ButtonAndTextFieldListener());  
  
 JMenuItem lineItem = new JMenuItem("Linie");  
 lineItem.setAccelerator(KeyStroke.*getKeyStroke*('L', InputEvent.*ALT\_DOWN\_MASK*));  
 toolMenu.add(lineItem);  
 lineItem.setActionCommand(lineString);  
 lineItem.addActionListener(new ButtonAndTextFieldListener());  
  
 JMenuItem rectangleItem = new JMenuItem("Viereck");  
 rectangleItem.setAccelerator(KeyStroke.*getKeyStroke*('V', InputEvent.*ALT\_DOWN\_MASK*));  
 toolMenu.add(rectangleItem);  
 rectangleItem.setActionCommand(rectangleString);  
 rectangleItem.addActionListener(new ButtonAndTextFieldListener());  
  
 JMenuItem ellipseItem = new JMenuItem("Ellipse");  
 ellipseItem.setAccelerator(KeyStroke.*getKeyStroke*('E', InputEvent.*ALT\_DOWN\_MASK*));  
 toolMenu.add(ellipseItem);  
 ellipseItem.setActionCommand(ellipseString);  
 ellipseItem.addActionListener(new ButtonAndTextFieldListener());  
  
 JMenuItem eraserItem = new JMenuItem("Radierer");  
 //Alt+X wurde gewählt, weil Alt+R von Windows abgefangen wird und nicht funktioniert  
 eraserItem.setAccelerator(KeyStroke.*getKeyStroke*('X', InputEvent.*ALT\_DOWN\_MASK*));  
 toolMenu.add(eraserItem);  
 eraserItem.setActionCommand(eraserString);  
 eraserItem.addActionListener(new ButtonAndTextFieldListener());  
  
 setJMenuBar(menuBar);  
 }  
  
 // Erstellen der SymbolBar mit den einzelnen Buttons für Werkzeuge inkl. Symbolen und Hotkeys  
 private void createSymbolBar() {  
 toolBar = new JToolBar();  
 //Erzeugen der Buttons für die Tools. Hotkeys orientieren sich an den deutschen Begriffen. Abstandhalter vor und nach den Tools  
 //ButtonGroup erstellt, um dem Anwender das ausgewählte Tool anzuzeigen  
 toolBar.addSeparator(new Dimension(20,100));  
 toolGroup = new ButtonGroup();  
 brushButton = createButton("icons/tools/brush.png", *VK\_P*, brushString, "Pinsel (Alt+P)", false, true);  
 lineButton = createButton("icons/tools/linie.gif", *VK\_L*, lineString, "Linie (Alt+L)", false, false);  
 rectangleButton = createButton("icons/tools/rechteck.gif", *VK\_V*, rectangleString, "Viereck (Alt+V)", false, false);  
 ellipseButton = createButton("icons/tools/ellipse.gif", *VK\_E*, ellipseString, "Ellipse (Alt+E)", false, false);  
 eraserButton = createButton("icons/tools/eraser.png", *VK\_X*, eraserString, "Radierer (Alt+X)", false, false);  
 toolBar.addSeparator(new Dimension(20,100));  
  
 //Bereich für die Strichstärke, Erzeugen eines neuen Panels mit BorderLayout  
 JPanel strokePanel = new JPanel();  
 strokePanel.setLayout(new GridLayout(0,1, 0, 2));  
 //Festlegen der Größe des Panels  
 strokePanel.setMaximumSize(new Dimension(50,90));  
 //Erstellen der Überschrift  
 JLabel strokeLabel1 = new JLabel("Strich-");  
 strokeLabel1.setHorizontalAlignment(JLabel.*CENTER*);  
 JLabel strokeLabel2 = new JLabel("breite");  
 strokeLabel2.setHorizontalAlignment(JLabel.*CENTER*);  
 //Erstellen des Textfeldes zum Eingeben der Strichstärke  
 strokeField = new JTextField("5,0");  
 strokeField.setToolTipText("Strichbreite in Pixeln (Alt+D");  
 strokeField.setHorizontalAlignment(JTextField.*CENTER*);  
 //Hinzufügen des ActionCommand und des ActionListener  
 strokeField.setActionCommand("stroke");  
 strokeField.addActionListener(new ButtonAndTextFieldListener());  
 //Standardmäßig ist das Textfeld nicht fokussierbar, um den Cursor im Feld nicht anzuzeigen und die Bearbeitung aus Versehen zu verhindern  
 strokeField.setFocusable(false);  
 //Anonyme Klasse zum Fokussieren des Textfeldes beim Klick auf das Textfeld  
 strokeField.addMouseListener(new MouseAdapter() {  
 @Override  
 public void mousePressed(MouseEvent e) {  
 strokeField.setFocusable(true);  
 strokeField.grabFocus();  
 }  
 });  
 //Keybinding um das Textfeld mit der Tastatur aktivieren zu können  
 InputMap inputMap = strokeField.getInputMap(JComponent.*WHEN\_IN\_FOCUSED\_WINDOW*);  
 ActionMap actionMap = strokeField.getActionMap();  
 KeyStroke altD = KeyStroke.*getKeyStroke*(KeyEvent.*VK\_D*, InputEvent.*ALT\_DOWN\_MASK*);  
 inputMap.put(altD, "focusStrokeField");  
 actionMap.put("focusStrokeField", new AbstractAction() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 strokeField.setFocusable(true);  
 strokeField.grabFocus();  
 }  
 });  
 //Bestätigungsbutton, um Strichdicke mit der Maus bestätigen zu können  
 JButton confirmStroke = new JButton("OK");  
 confirmStroke.setActionCommand("stroke");  
 confirmStroke.addActionListener(new ButtonAndTextFieldListener());  
 confirmStroke.setFont(new Font("Arial", Font.*BOLD*, 11));  
 confirmStroke.setFocusable(false);  
 //Hinzufügen zur Toolbar und Abstandshalter  
 strokePanel.add(strokeLabel1);  
 strokePanel.add(strokeLabel2);  
 strokePanel.add(strokeField);  
 strokePanel.add(confirmStroke);  
 toolBar.add(strokePanel);  
 toolBar.addSeparator(new Dimension(20,100));  
  
 //Bereich für die Farbauswahl  
 //ButtonGroup erstellt, um dem Anwender die ausgewählte Farbe anzuzeigen, Erstellen der Buttons  
 colorGroup = new ButtonGroup();  
 colorPanel = new JPanel();  
 colorPanel.setLayout(new GridLayout(3,5));  
 colorPanel.setMaximumSize(new Dimension(180,150));  
 blackButton = createButton("icons/colors/black.jpg", *VK\_1*, "black", "schwarz (Alt+1)", true, true);  
 redButton = createButton("icons/colors/red.jpg", *VK\_2*, "red", "rot (Alt+2)", true, false);  
 blueButton = createButton("icons/colors/blue.jpg", *VK\_3*, "blue", "blau (Alt+3)", true, false);  
 yellowButton = createButton("icons/colors/yellow.jpg", *VK\_4*, "yellow", "gelb (Alt+4)", true, false);  
 whiteButton = createButton("icons/colors/white.jpg", *VK\_5*, "white", "weiß (Alt+5)", true, false);  
 cyanButton = createButton("icons/colors/cyan.jpg", *VK\_6*, "cyan", "cyan (Alt+6)", true, false);  
 greenButton = createButton("icons/colors/green.jpg", *VK\_7*, "green", "grün (Alt+7)", true, false);  
 magentaButton= createButton("icons/colors/magenta.jpg", *VK\_8*, "magenta", "magenta (Alt+8)", true, false);  
 orangeButton= createButton("icons/colors/orange.jpg", *VK\_9*, "orange", "orange (Alt+9)", true, false);  
 pinkButton = createButton("icons/colors/pink.jpg", *VK\_0*, "pink", "pink (Alt+0)", true, false);  
 lightgrayButton = createButton("icons/colors/lightgray.jpg", *VK\_NUMPAD1*, "lightgray", "hellgrau (Alt+NUMPAD1)", true, false);  
 grayButton = createButton("icons/colors/gray.jpg", *VK\_NUMPAD2*, "gray", "grau (Alt+NUMPAD2)", true, false);  
 darkgrayButton = createButton("icons/colors/darkgray.jpg", *VK\_NUMPAD3*, "darkgray", "dunkelgrau (Alt+NUMPAD3)", true, false);  
 toolBar.add(colorPanel);  
  
 this.add(toolBar, BorderLayout.*NORTH*);  
 }  
  
 //Eigene Methode um Buttons vollständig zu implementieren  
 //jeder Button kann bekommen: Icon, Hotkey, Tooltip, ActionCommand, ActionListener  
 //Button sind nicht fokussierbar, um das Selektieren (das Hervorheben) mit den Pfeiltasten zu verhindern.  
 //setSelected wird verwendet, um die Standardtools zu Beginn auszuwählen  
 private JToggleButton createButton(String imageIconFilename, int mnemonic, String actionCommand, String tooltip, boolean isColor, boolean select) {  
 JToggleButton button = new JToggleButton();  
 button.setIcon(new ImageIcon(imageIconFilename));  
 button.setMnemonic(mnemonic);  
 button.setToolTipText(tooltip);  
 //Wenn der Button eine Farbe ist, kommt er in das entsprechende Panel, ansonsten direkt zur Toolbar.  
 if (isColor) {  
 colorGroup.add(button);  
 colorPanel.add(button);  
 }  
 else {  
 toolGroup.add(button);  
 toolBar.add(button);  
 }  
 button.setFocusable(false);  
 button.setSelected(select);  
 button.setActionCommand(actionCommand);  
 button.addActionListener(new ButtonAndTextFieldListener());  
 return button;  
 }  
  
 //Klasse zum Ausführen von Befehlen nach Knopfdruck bzw. beim Bearbeiten von Textfeldern  
 class ButtonAndTextFieldListener implements ActionListener {  
 //Klassenvariable für das Standardverzeichnis für gespeicherte Bilder  
 Path savedPictures = Path.*of*("savedPictures");  
  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 //Funktion, um beim Zurückwechseln vom Radierer wieder die ursprüngliche Farbe zugeordnet zu bekommen  
 if (paintPanel.getTool().equals(eraserString) && (e.getActionCommand().equals(brushString) || e.getActionCommand().equals(lineString) ||  
 e.getActionCommand().equals(rectangleString) || e.getActionCommand().equals(ellipseString))) {  
 Color lastColor = paintPanel.getLastColor();  
 paintPanel.setColor(lastColor);  
 //Anzeige der Farbauswahl nach dem Radieren  
 if (lastColor == Color.*BLACK*) blackButton.setSelected(true);  
 if (lastColor == Color.*RED*) redButton.setSelected(true);  
 if (lastColor == Color.*BLUE*) blueButton.setSelected(true);  
 if (lastColor == Color.*YELLOW*) yellowButton.setSelected(true);  
 if (lastColor == Color.*WHITE*) whiteButton.setSelected(true);  
 if (lastColor == Color.*CYAN*) cyanButton.setSelected(true);  
 if (lastColor == Color.*GREEN*) greenButton.setSelected(true);  
 if (lastColor == Color.*MAGENTA*) magentaButton.setSelected(true);  
 if (lastColor == Color.*ORANGE*) orangeButton.setSelected(true);  
 if (lastColor == Color.*PINK*) pinkButton.setSelected(true);  
 if (lastColor == Color.*LIGHT\_GRAY*) lightgrayButton.setSelected(true);  
 if (lastColor == Color.*GRAY*) grayButton.setSelected(true);  
 if (lastColor == Color.*DARK\_GRAY*) darkgrayButton.setSelected(true);  
 }  
  
 //Funktionen, um das Werkzeug auszuwählen  
 if (e.getActionCommand().equals(brushString)) {  
 paintPanel.setTool(brushString);  
 brushButton.setSelected(true);  
 }  
 if (e.getActionCommand().equals(lineString)) {  
 paintPanel.setTool(lineString);  
 lineButton.setSelected(true);  
 }  
 if (e.getActionCommand().equals(rectangleString)) {  
 paintPanel.setTool(rectangleString);  
 rectangleButton.setSelected(true);  
 }  
 if (e.getActionCommand().equals(ellipseString)) {  
 paintPanel.setTool(ellipseString);  
 ellipseButton.setSelected(true);  
 }  
 //Radierer darf nur gewählt werden, wenn er noch nicht ausgewählt ist, sonst wird die vorher gewählte Farbe mit weiß überschrieben  
 if (e.getActionCommand().equals(eraserString) && !paintPanel.getTool().equals(eraserString)) {  
 eraserButton.setSelected(true);  
 //Speichern der aktuellen Farbe  
 paintPanel.setLastColor(paintPanel.getColor());  
 //Setzen der aktuellen Farbe auf Weiß, da der Radierer eigentlich ein weißer Pinsel ist  
 paintPanel.setColor(Color.*WHITE*);  
 paintPanel.setTool(eraserString);  
 //Aufheben der Farbauswahl beim Radieren  
 colorGroup.clearSelection();  
 }  
  
 //Funktion zum Setzen der Strichstärke  
 if (e.getActionCommand().equals("stroke")) {  
 String strokeStringWithPoint;  
 String strokeStringWithKomma;  
 try {  
 //Einlesen der Strichstärke  
 strokeStringWithKomma = strokeField.getText();  
 //Überprüfung, ob Strichstärke eine Zahl ist  
 if (!strokeStringWithKomma.matches("^-?(?:\\d+,?\\d\*|\\.\\d+)$")) throw new IOException("Ungültige Eingabe: Es muss eine Zahl eingegeben werden!");  
 //tauschen von eingegebenen Komma zu Punkt  
 strokeStringWithPoint = strokeStringWithKomma.replace(',', '.');  
 float stroke = *parseFloat*(strokeStringWithPoint);  
 //Überprüfung, ob Strichstärke größer als null ist  
 if (stroke <= 0) throw new IOException("Ungültige Eingabe: Der Wert muss größer als Null sein!");  
 //Ändern der Strichstärke  
 paintPanel.setStroke(stroke);  
 //Neuerzeugen des Textes, damit konstant wenigstens eine Nachkommastelle angezeigt wird z. B. Eingabe 8 und Anzeige 8,0  
 strokeStringWithPoint = "" + paintPanel.getStroke();  
 strokeStringWithKomma = strokeStringWithPoint.replace("." , ",");  
 strokeField.setText(strokeStringWithKomma);  
 } catch (IOException ex) {  
 //zurücksetzen des Textfeldes auf ursprüngliche Strichdicke  
 strokeStringWithPoint = "" + paintPanel.getStroke();  
 strokeStringWithKomma = strokeStringWithPoint.replace("." , ",");  
 strokeField.setText(strokeStringWithKomma);  
 }  
 //Fokussierbarkeit des Textfeldes entfernen, damit nicht aus Versehen weitergeschrieben wird  
 strokeField.setFocusable(false);  
 }  
  
 //Funktion zum Setzen der Farbe, nur wenn ein anderes Tool als der Radierer ausgewählt ist, ansonsten Aufheben der Auswahl  
 if (!paintPanel.getTool().equals("eraser")){  
 if (e.getActionCommand().equals("black")) paintPanel.setColor(Color.*BLACK*);  
 if (e.getActionCommand().equals("red")) paintPanel.setColor(Color.*RED*);  
 if (e.getActionCommand().equals("blue")) paintPanel.setColor(Color.*BLUE*);  
 if (e.getActionCommand().equals("yellow")) paintPanel.setColor(Color.*YELLOW*);  
 if (e.getActionCommand().equals("white")) paintPanel.setColor(Color.*WHITE*);  
 if (e.getActionCommand().equals("cyan")) paintPanel.setColor(Color.*CYAN*);  
 if (e.getActionCommand().equals("green")) paintPanel.setColor(Color.*GREEN*);  
 if (e.getActionCommand().equals("magenta")) paintPanel.setColor(Color.*MAGENTA*);  
 if (e.getActionCommand().equals("orange")) paintPanel.setColor(Color.*ORANGE*);  
 if (e.getActionCommand().equals("pink")) paintPanel.setColor(Color.*PINK*);  
 if (e.getActionCommand().equals("lightgray")) paintPanel.setColor(Color.*LIGHT\_GRAY*);  
 if (e.getActionCommand().equals("gray")) paintPanel.setColor(Color.*GRAY*);  
 if (e.getActionCommand().equals("darkgray")) paintPanel.setColor(Color.*DARK\_GRAY*);  
 } else colorGroup.clearSelection();  
  
 //Funktion zum Erstellen eines neuen, weißen Zeichenblatts in der gleichen Größe des aktuellen Blatts mit Bestätigungsdialog  
 if (e.getActionCommand().equals("newSameSize")) {  
 int confirmation = JOptionPane.*showConfirmDialog*(paintPanel, "Wollen Sie wirklich ein neues Blatt erstellen? Ungespeicherter Fortschritt geht verloren.", "Neues Blatt", JOptionPane.*YES\_NO\_OPTION*);  
 if (confirmation == JOptionPane.*YES\_OPTION*) {  
 paintPanel.newPanel();  
 //Zurücksetzen der Buttons und des Textfeldes  
 strokeField.setText("5,0");  
 brushButton.setSelected(true);  
 blackButton.setSelected(true);  
 //zurücksetzen des Speicherpfades  
 outputFile = null;  
 }  
 }  
 //Funktion zum Erstellen eines neuen, weißen Zeichenblatts in neu zu bestimmender Größe mit Bestätigungsdialog  
 if (e.getActionCommand().equals("newOtherSize")) {  
 int confirmation = JOptionPane.*showConfirmDialog*(paintPanel, "Wollen Sie wirklich ein neues Blatt erstellen? Ungespeicherter Fortschritt geht verloren.", "Neues Blatt", JOptionPane.*YES\_NO\_OPTION*);  
 if (confirmation == JOptionPane.*YES\_OPTION*) {  
 try {  
 //Abfrage der gewünschten Größe der Zeichenfläche  
 String widthString = JOptionPane.*showInputDialog*("Breite der Zeichenfläche in Pixeln");  
 String heightString = JOptionPane.*showInputDialog*("Höhe der Zeichenfläche in Pixeln");  
 //Überprüfung, ob Eingabe eine Ganzzahl ist  
 if (!widthString.matches("^-?\\d+$") || !heightString.matches("^-?\\d+$")) throw new IOException("Ungültige Eingabe: Es müssen ganze Zahlen eingegeben werden!");  
 int width = *parseInt*(widthString);  
 int height = *parseInt*(heightString);  
 //Überprüfung, ob Eingaben positiv sind  
 if (width <= 0 || height <= 0) throw new IOException("Ungültige Eingabe: Die Werte müssen größer als Null sein!");  
 //Erstellen der neuen Zeichenfläche  
 paintPanel.newPanel(width, height);  
 //Zurücksetzen der Buttons und des Textfeldes  
 strokeField.setText("5,0");  
 brushButton.setSelected(true);  
 blackButton.setSelected(true);  
 //zurücksetzen des Speicherpfades  
 outputFile = null;  
 } catch (IOException ex) {  
 //Fehlermeldung, bei falscher Eingabe  
 JOptionPane.*showMessageDialog*(paintPanel, ex.getMessage());  
 }  
 }  
 }  
 //Funktion zum Beenden des Programms mit Bestätigungsdialog  
 if (e.getActionCommand().equals("close")) {  
 int confirmation = JOptionPane.*showConfirmDialog*(paintPanel, "Wollen Sie die Anwendung wirklich beenden? Ungespeicherter Fortschritt geht verloren.", "Beenden", JOptionPane.*YES\_NO\_OPTION*);  
 if (confirmation == JOptionPane.*YES\_OPTION*) System.*exit*(0);  
 }  
 //Funktion zum Speichern des Bildes. Nur wenn noch keine Datei vorhanden ist, wird der Speichern-Dialog aufgerufen  
 if (e.getActionCommand().equals("save")) {  
 //Dialog wird nur angezeigt, wenn noch keine Datei vorhanden ist  
 if (outputFile == null) {  
 //Abfrage, ob wirklich gespeichert werden soll  
 if (fileChooser.showSaveDialog(paintPanel) == JFileChooser.*APPROVE\_OPTION*) {  
 //Bei Bedarf erstellen des Standardverzeichnisses für gespeicherte Bilder  
 try {  
 if (!Files.*exists*(savedPictures)) Files.*createDirectory*(savedPictures);  
 } catch (IOException ex) {  
 JOptionPane.*showMessageDialog*(paintPanel, "Standardverzeichnis für gespeicherte Bilder konnte nicht erstellt werden!");  
 }  
 //es wird an einen Dateinamen nur die Endung .jpg angehängt, wenn diese noch nicht vorhanden ist  
 if (String.*valueOf*(fileChooser.getSelectedFile()).endsWith(".jpg")) outputFile = new File (String.*valueOf*(fileChooser.getSelectedFile()));  
 else outputFile = new File (fileChooser.getSelectedFile() + ".jpg");  
 }  
 }  
 if (outputFile != null) paintPanel.save(outputFile);  
 }  
 //Funktion zum Speichern des Bildes. Immer mit Speichern-Dialog  
 if (e.getActionCommand().equals("saveAs")) {  
 if (fileChooser.showSaveDialog(paintPanel) == JFileChooser.*APPROVE\_OPTION*) {  
 //Bei Bedarf erstellen des Standardverzeichnisses für gespeicherte Bilder  
 try {  
 if (!Files.*exists*(savedPictures)) Files.*createDirectory*(savedPictures);  
 } catch (IOException ex) {  
 JOptionPane.*showMessageDialog*(paintPanel, "Standardverzeichnis für gespeicherte Bilder konnte nicht erstellt werden!");  
 }  
 //es wird an einen Dateinamen nur die Endung .jpg angehängt, wenn diese noch nicht vorhanden ist  
 if (String.*valueOf*(fileChooser.getSelectedFile()).endsWith(".jpg")) outputFile = new File (String.*valueOf*(fileChooser.getSelectedFile()));  
 else outputFile = new File (fileChooser.getSelectedFile() + ".jpg");  
 paintPanel.save(outputFile);  
 }  
 }  
 //Funktion zum Laden eines Bildes mit Bestätigungsdialog  
 if (e.getActionCommand().equals("load")) {  
 int confirmation = JOptionPane.*showConfirmDialog*(paintPanel, "Wollen Sie wirklich ein neues Bild laden? Ungespeicherter Fortschritt geht verloren.", "Laden", JOptionPane.*YES\_NO\_OPTION*);  
 if (confirmation == JOptionPane.*YES\_OPTION*) {  
 fileChooser.showOpenDialog(paintPanel);  
 File inputFile = new File (String.*valueOf*(fileChooser.getSelectedFile()));  
 paintPanel.load(inputFile);  
 //anpassen des Speicherpfades  
 outputFile = inputFile;  
 }  
 }  
 }  
 }  
  
 //Klasse zum Ausführen von Befehlen nach Mausklick  
 class MouseListener extends MouseAdapter {  
 @Override  
 public void mousePressed(MouseEvent e) {  
 //Filter nach Aktionen mit der linken Maustaste  
 if (e.getButton() == MouseEvent.*BUTTON1*) {  
 leftMouseButtonIsPressed = true;  
 //Speichern der Mausposition beim Mausdruck in der Variable lastMousePosition  
 paintPanel.setLastMousePosition(e.getPoint());  
 //Zurücksetzen der Endposition der Figur vor dem Ziehen, um ungewolltes Verhalten der Vorschau zu vermeiden  
 paintPanel.setEndPointOfShape(e.getPoint());  
 //Beim Pinsel(Radierer) kann man auch nur mit einem Mausklick zeichnen  
 if (paintPanel.getTool().equals(brushString)) paintPanel.brush(e.getPoint());  
 if (paintPanel.getTool().equals(eraserString)) paintPanel.erase(e.getPoint());  
 }  
 }  
  
 //Klasse zum Ausführen von Befehlen beim Loslassen der Maus  
 @Override  
 public void mouseReleased(MouseEvent e) {  
 //Filter nach Aktionen mit der linken Maustaste  
 if (e.getButton() == MouseEvent.*BUTTON1*) {  
 //Linien, Rechtecke und Ellipsen werden beim Loslassen der Maus zusammen mit der Mausposition beim Klicken der Maus gezeichnet  
 if (paintPanel.getTool().equals(lineString)) paintPanel.line(e.getPoint());  
 if (paintPanel.getTool().equals(rectangleString)) paintPanel.rectangle(e.getPoint());  
 if (paintPanel.getTool().equals(ellipseString)) paintPanel.ellipse(e.getPoint());  
 leftMouseButtonIsPressed = false;  
 }  
 }  
 }  
  
 //Klasse zum Ausführen von Befehlen beim Ziehen der Maus  
 class MouseMotionListener extends MouseMotionAdapter {  
 @Override  
 public void mouseDragged(MouseEvent e) {  
 //Filter nach Aktionen mit der linken Maustaste  
 if (leftMouseButtonIsPressed) {  
 //Pinsel(Radierer) arbeiten beim Ziehen der Maus  
 if (paintPanel.getTool().equals(brushString)) paintPanel.brush(e.getPoint());  
 if (paintPanel.getTool().equals(eraserString)) paintPanel.erase(e.getPoint());  
 //Positionsübergabe der Endposition von Linie, Rechteck und Ellipse für die Vorschau und Aktualisierung der Vorschau beim Ziehen  
 if (paintPanel.getTool().equals(lineString) || paintPanel.getTool().equals(rectangleString) || paintPanel.getTool().equals(ellipseString) ) {  
 paintPanel.setEndPointOfShape(e.getPoint());  
 paintPanel.repaint();  
 }  
 }  
 }  
 }  
}

**PaintPanel.java**

package org.PaintProgram;  
  
//importieren der benötigten Klassen  
import javax.imageio.ImageIO;  
import javax.swing.\*;  
import java.awt.\*;  
import java.awt.image.BufferedImage;  
import java.io.File;  
import java.io.IOException;  
  
//Klasse, die die Zeichenfläche darstellt  
public class PaintPanel extends JPanel {  
 //Klassenvariablen  
 private BufferedImage picture;  
 private Graphics2D g2Image;  
 private String tool;  
 private Point lastMousePosition;  
 private Color lastColor;  
 private Point endPointOfShape;  
  
 //Getter  
 public String getTool() {  
 return tool;  
 }  
 public Color getColor() {  
 return g2Image.getColor();  
 }  
 public Color getLastColor() {  
 return lastColor;  
 }  
 public float getStroke() {  
 BasicStroke stroke = (BasicStroke) g2Image.getStroke();  
 return stroke.getLineWidth();  
 }  
  
 //Setter  
 public void setTool(String tool) {  
 this.tool = tool;  
 }  
 public void setColor(Color color) {  
 g2Image.setColor(color);  
 }  
 public void setLastColor(Color color) {  
 this.lastColor = color;  
 }  
 public void setStroke(float stroke) {  
 g2Image.setStroke(new BasicStroke(stroke));  
 }  
 public void setLastMousePosition(Point point) {  
 this.lastMousePosition = point;  
 }  
 public void setEndPointOfShape(Point point) {this.endPointOfShape = point;}  
  
 //Konstruktor  
 public PaintPanel(int width, int height) {  
 //Erstellen des BufferedImage mit weißem Hintergrund  
 picture = new BufferedImage(width, height, BufferedImage.*TYPE\_INT\_RGB*);  
 g2Image = picture.createGraphics();  
 g2Image.setColor(Color.*WHITE*);  
 g2Image.fillRect(0, 0, width, height);  
  
 //Initialisierung vom ausgewählten Werkzeug, der Farbe und der Strichdicke  
 tool = "brush";  
 g2Image.setColor(Color.*BLACK*);  
 g2Image.setStroke(new BasicStroke(5.0F));  
  
 //Initialisierung weiterer Variablen  
 lastMousePosition = new Point(0,0);  
 endPointOfShape = new Point(0,0);  
 }  
  
 //Methode paintComponent muss zum Zeichnen auf der Zeichenoberfläche überschrieben werden  
 @Override  
 public void paintComponent(Graphics g) {  
 //Initialisieren der Zeichenfläche  
 super.paintComponent(g);  
 Graphics2D g2 = (Graphics2D) g;  
 //Zeichnen des Bildes auf die Zeichenfläche  
 g2.drawImage(picture, 0, 0, null);  
  
 //Farbe und Strichdicke der Vorschau beim Ziehen an Auswahl anpassen  
 g2.setColor(getColor());  
 g2.setStroke(g2Image.getStroke());  
 //Zeichnen der Vorschau beim Ziehen für Linien  
 if (tool.equals("line")) g2.drawLine(lastMousePosition.x,lastMousePosition.y, endPointOfShape.x, endPointOfShape.y);  
 //Zeichnen der Vorschau beim Ziehen für Rechtecke  
 if (tool.equals("rectangle")) {  
 int [] rectangle = checkOrientation(lastMousePosition, endPointOfShape);  
 g2.drawRect(rectangle[0], rectangle[1], rectangle[2], rectangle[3]);  
 }  
 //Zeichnen der Vorschau beim Ziehen für Ellipsen  
 if (tool.equals("ellipse")) {  
 int [] ellipse = checkOrientation(lastMousePosition, endPointOfShape);  
 g2.drawOval(ellipse[0], ellipse[1], ellipse[2], ellipse[3]);  
 }  
 }  
  
 //Methode zum Arbeiten mit dem Pinsel  
 public void brush (Point actualMousePosition) {  
 //Es wird eine kleine Linie von der jeweils letzten gespeicherten Mausposition bis zur aktuellen Position gezeichnet  
 line(actualMousePosition);  
 //Anschließend wird die letzte Mausposition aktualisiert  
 lastMousePosition.x = actualMousePosition.x;  
 lastMousePosition.y = actualMousePosition.y;  
 //So entsteht ein Pinselstrich entlang des Mauspfades  
 }  
  
 //Methode zum Zeichnen von Linien  
 public void line (Point actualMousePosition) {  
 //gezeichnet wird nur, wenn der Startpunkt ungleich dem Endpunkt ist oder der Pinsel ausgewählt ist  
 if (tool.equals("brush") || (lastMousePosition.x != actualMousePosition.x & lastMousePosition.y != actualMousePosition.y)) {  
 //Zeichnen der Linie  
 g2Image.drawLine(lastMousePosition.x, lastMousePosition.y, actualMousePosition.x, actualMousePosition.y);  
 repaint();  
 }  
 }  
  
 //Methode zum Zeichnen von Rechtecken  
 public void rectangle (Point actualMousePosition) {  
 //gezeichnet wird nur, wenn der Startpunkt ungleich dem Endpunkt ist  
 if (lastMousePosition.x != actualMousePosition.x & lastMousePosition.y != actualMousePosition.y) {  
 //Berechnung der Abmaße des Rechtecks  
 int [] rectangle = checkOrientation(lastMousePosition, actualMousePosition);  
 //Zeichnen des Rechtecks  
 g2Image.drawRect(rectangle[0], rectangle[1], rectangle[2], rectangle[3]);  
 repaint();  
 }  
 }  
  
 //Methode zum Zeichnen von Ellipsen  
 public void ellipse (Point actualMousePosition) {  
 //gezeichnet wird nur, wenn der Startpunkt ungleich dem Endpunkt ist  
 if (lastMousePosition.x != actualMousePosition.x & lastMousePosition.y != actualMousePosition.y) {  
 //Berechnung der Abmaße der Ellipse  
 int [] ellipse = checkOrientation(lastMousePosition, actualMousePosition);  
 //Zeichnen der Ellipse  
 g2Image.drawOval(ellipse[0], ellipse[1], ellipse[2], ellipse[3]);  
 repaint();  
 }  
 }  
  
 //Methode zum Radieren  
 public void erase (Point actualMousePosition) {  
 //Farbe Weiß wurde bereits im Frame gesetzt. Der Radierer arbeitet exakt wie der Pinsel und ruft nur diese Funktion auf.  
 brush(actualMousePosition);  
 }  
  
 //Methode zum Überprüfen, in welche Richtung Figuren gezogen werden (Behandlung von negativen Werten)  
 public int [] checkOrientation(Point startPoint, Point endPoint) {  
 //Lokale Variablen, um Sie bei Bedarf vertauschen zu können  
 int x1 = startPoint.x, y1 = startPoint.y, x2 = endPoint.x, y2 = endPoint.y;  
 //Berechnung der Abmaße der Figur  
 int width = x2 - x1;  
 int height = y2 - y1;  
 //Überprüfung, in welche Richtung die Maus gezogen wird. Falls ein negativer Wert entsteht, wird der Anfangspunkt  
 //auf den ursprünglichen Endpunkt gesetzt  
 if (width < 0) {  
 x1 = x2;  
 width = -width;  
 }  
 if (height < 0) {  
 y1 = y2;  
 height = -height;  
 }  
 return new int [] {x1, y1, width, height};  
 }  
  
 //Methode zum Speichern des Bildes  
 public void save (File outputFile) {  
 //Speichern des Bildes am übergebenen Ort. Falls ein Fehler auftritt, kommt ein Popup mit entsprechender Meldung.  
 try {  
 ImageIO.*write*(picture, "jpg", outputFile);  
 } catch (IOException exc){  
 JOptionPane.*showMessageDialog*(this, "Fehler beim Speichern des Bildes!");  
 }  
 }  
  
 //Methode zum Laden eines Bildes  
 public void load (File inputFile) {  
 //Laden eines Bildes vom übergebenen Ort. Falls ein Fehler auftritt, kommt ein Popup mit entsprechender Meldung.  
 try {  
 //Laden des Bildes  
 picture = ImageIO.*read*(inputFile);  
 //Verknüpfen des Graphics2D zum Bild  
 g2Image = picture.createGraphics();  
 //Darstellen des geladenen Bildes  
 repaint();  
 //Auswählen der Standardtools  
 g2Image.setColor(Color.*BLACK*);  
 tool = "brush";  
 g2Image.setStroke(new BasicStroke(5.0F));  
 } catch (IOException exc){  
 JOptionPane.*showMessageDialog*(this, "Fehler beim Laden des Bildes!");  
 }  
 }  
  
 //Überladene Funktion zum Erzeugen einer neuen Zeichenoberfläche mit gleicher Größe  
 public void newPanel() {  
 //Zeichenfläche wird weiß übermalt  
 g2Image.setColor(Color.*WHITE*);  
 g2Image.fillRect(0, 0, picture.getWidth(), picture.getHeight());  
 repaint();  
 //Auswählen der Standardtools  
 g2Image.setColor(Color.*BLACK*);  
 tool = "brush";  
 g2Image.setStroke(new BasicStroke(5.0F));  
 }  
  
 //Überladene Funktion zum Erzeugen einer neuen Zeichenoberfläche mit anderer Größe  
 public void newPanel(int width, int height) {  
 //Setzen der neuen Größe der Zeichenfläche  
 setPreferredSize(new Dimension(width, height));  
 //Erstellen eines neuen Bildes mit Grafik und anpassen dessen Größe  
 picture = new BufferedImage(width, height, BufferedImage.*TYPE\_INT\_RGB*);  
 g2Image = picture.createGraphics();  
 setSize(new Dimension(width, height));  
 //Aufruf der überlagerten Funktion  
 newPanel();  
 }  
}

**TestPaintPanel.java**

package org.PaintProgram;  
  
import junitparams.JUnitParamsRunner;  
import junitparams.Parameters;  
import org.junit.After;  
import org.junit.Before;  
import org.junit.Test;  
import org.junit.runner.RunWith;  
  
import java.awt.\*;  
import java.util.Arrays;  
  
import static org.junit.Assert.\*;  
  
@RunWith(JUnitParamsRunner.class)  
public class TestPaintPanel {  
 PaintPanel paintPanel;  
  
 //Initialisieren der Klassen vor dem Testen  
 @Before  
 public void before() {  
 paintPanel = new PaintPanel(1200, 600);  
 }  
  
 //Löschen der Zuweisung nach dem Testen  
 @After  
 public void after() {  
 paintPanel = null;  
 }  
  
 //Testen der Getter- und Setter-Methode für Tools in einem Test  
 @Test  
 @Parameters({  
 "",  
 "brush",  
 "line",  
 "rectangle",  
 })  
 public void testSetAndGetTool (String expectedTool) {  
 paintPanel.setTool(expectedTool);  
 String actualTool = paintPanel.getTool();  
 *assertEquals*(expectedTool, actualTool);  
 }  
  
 //Testen der Getter- und Setter-Methode für Farben in einem Test  
 @Test  
 @Parameters({  
 "",  
 "yellow",  
 "red",  
 "blue",  
 })  
 public void testSetAndGetColor (String colorString) {  
 Color expectedColor = Color.*BLACK*;  
 if (colorString.equals("yellow")) expectedColor = Color.*YELLOW*;  
 if (colorString.equals("red")) expectedColor = Color.*RED*;  
 if (colorString.equals("blue")) expectedColor = Color.*BLUE*;  
 paintPanel.setColor(expectedColor);  
 Color actualColor = paintPanel.getColor();  
 *assertEquals*(expectedColor, actualColor);  
 }  
  
 //Testen der Getter- und Setter-Methode für die letzte gespeicherte Farbe in einem Test  
 @Test  
 @Parameters({  
 "",  
 "yellow",  
 "red",  
 "blue"  
 })  
 public void testSetAndGetLastColor (String colorString) {  
 Color expectedColor = Color.*BLACK*;  
 if (colorString.equals("yellow")) expectedColor = Color.*YELLOW*;  
 if (colorString.equals("red")) expectedColor = Color.*RED*;  
 if (colorString.equals("blue")) expectedColor = Color.*BLUE*;  
 paintPanel.setLastColor(expectedColor);  
 Color actualColor = paintPanel.getLastColor();  
 *assertEquals*(expectedColor, actualColor);  
 }  
  
 //Testen der Getter- und Setter-Methode für die Strichdicke in einem Test  
 @Test  
 @Parameters({  
 //"",  
 //"-5",  
 //"0",  
 "0.1",  
 "1",  
 "2",  
 "1000",  
 "5.5"  
 })  
 public void testSetAndGetStroke (float expectedStroke) {  
 paintPanel.setStroke(expectedStroke);  
 float actualStroke = paintPanel.getStroke();  
 *assertEquals*(expectedStroke, actualStroke, 0.0);  
 }  
  
 //Testen der Methode checkOrientation, Kommentare zu den einzelnen Tests sind in der Dokumentation  
 @Test  
 @Parameters({  
 "100 , 100 , 500 , 500 , 100-100-400-400",  
 "500 , 500 , 100 , 100 , 100-100-400-400",  
 "100 , 500 , 500 , 100 , 100-100-400-400",  
 "500 , 100 , 100 , 500 , 100-100-400-400",  
 "000 , 000 , 100 , 100 , 000-000-100-100",  
 "100 , 100 , 000 , 000 , 000-000-100-100",  
 //"-100 , -100 , -50 , -50 , ",  
 "100 , 100 , 100 , 500 , 100-100-000-400",  
 "100 , 100 , 500 , 100 , 100-100-400-000",  
 "500 , 500 , 500 , 500 , 500-500-000-000",  
 })  
 public void testCheckOrientation (int startPointX, int startPointY, int endPointX, int endPointY, String expectedString) {  
 Point startPoint = new Point(startPointX, startPointY);  
 Point endPoint = new Point(endPointX, endPointY);  
 int [] expectedArray = Arrays.*stream*(expectedString.split("-")).map(Integer::*valueOf*).mapToInt(Integer::intValue).toArray();  
 int [] actualArray = paintPanel.checkOrientation(startPoint, endPoint);  
 *assertArrayEquals*(expectedArray, actualArray);  
 }  
}