Components used in Graphical User Interfaces

Objectives:

- implementation of Graphical User Interfaces (GUI)
- organizing components in GUI (Layout Managers)

Java components

Components are functional blocks that serve to build graphical user interfaces. There are several categories of components:

· Visual components that serve directly to build GUI (buttons, selection lists, etc.)
· Container-type components serving to spatially organize those in the first category (are derived directly from the Container class)
· menu components
. Layout Managers - control the size and position of components in a container

All components are derived from the classes `Component` (in AWT) and `JComponent` (in Swing). These classes contain, alongside the visual specifications of the basic components, graphic methods and event handling mechanisms.
All Swing components (except for top level containers) are lightweight components (on the other hand, the AWT components are the heavyweight ones).
A heavyweight component is rendered by a native operating system component.
A lightweight component is rendered by Java.

There are some basic methods that are common to any class that manages components (except menus), regardless of where they are placed into the above categorization. These methods are inherited from the class `java.awt.Component` and are available both in AWT and Swing.

https://docs.oracle.com/en/java/javase/17/docs/api/java.desktop/java/awt/Component.html

- `Dimension getsize()`
  => returns a *Dimension* object that contains the width and height of the component
- `void setBackground(Color c)`
  => set the background color of the component
- `void setForeground(Color c)`
  => sets the foreground color of the component (typically the color of the text that is displayed on the component)
- `void setFont(Font f)`
  => sets the font with which the characters that form the text displayed on the component will be displayed
- `void setEnabled(boolean b)`
  => if the received parameter is *false*, the component is greyed out and does not respond to the user's actions (is disabled)
- `void setSize(Dimension d)` sau `void setSize(int width, int height)`
  => sets the size (width and height) of the current component
- `void setBounds(int x, int y, int width, int height)`
  => sets the size (width and height) and position on the screen (upper left colt) of the current composition
- `void setVisible(boolean b)`
  => specifies whether the current component is visible or not
- *etc.*

Menu classes are derived from `java.awt.MenuComponent`.
https://docs.oracle.com/en/java/javase/17/docs/api/java.desktop/java/awt/MenuComponent.html

Description of basic AWT components

`Button` https://docs.oracle.com/en/java/javase/17/docs/api/java.desktop/java/awt/Button.html

The instances of this class represent buttons that can be "pressed" by the user. The buttons display textual information. When a button is pressed, it generates an `ActionEvent`.

`Canvas` https://docs.oracle.com/en/java/javase/17/docs/api/java.desktop/java/awt/Canvas.html

Components of this type do not have the default appearance and behavior. Objects of this type or instantiated in the subclasses of Canvas serve to define drawing areas, or some work areas to group the included components, etc.
Canvas objects generates `MouseEvent, MouseMotionEvent, KeyEvent`.

`Checkbox` https://docs.oracle.com/en/java/javase/17/docs/api/java.desktop/java/awt/Checkbox.html

Such a component is a two-state button (checked or un-checked).

Using the methods

```
boolean getState()
```
şi
```
void setState(boolean state)
```

the state of the button can be read or modified.

If more than one button of this type is grouped by in a `CheckboxGroup` component, they get radio buttons functionalities (pressing a button deselects any other selected button).

Example:

```
CheckboxGroup cbg = new CheckboxGroup();
add(new Checkbox(”Articol 1”, false, cbg));
add(new Checkbox(”Articol 2”, false, cbg));
add(new Checkbox(”Articol 3”, true, cbg));
```

In such a grouping, the selected button can be determined using the method

```
Checkbox getSelectedCheckbox()
```

and the checked component can be set using

```
void setSelectedCheckbox(Checkbox newSelection)
```

Checkbox components generate *ItemEvent*.

`Choice` https://docs.oracle.com/en/java/javase/17/docs/api/java.desktop/java/awt/Choice.html

Choice components offer the user the possibility to select an articol from an expandable list of possible options. The list of options can be populated by repeatedly calling the method `add()`.

Example:

```
Choice ch = new Choice();
ch.add(”Articol 1”);
ch.add(”Articol 2”);
ch.add(”Articol 3”);
```

The mechanisms for selecting and reading the selected articles are similar to those presented at `CheckboxGroup`. *Choice* components generate *ItemEvent.*

`FileDialog` https://docs.oracle.com/en/java/javase/17/docs/api/java.desktop/java/awt/FileDialog.html

This class provides the mechanism for creating a dialog window dedicated to saving or selecting a file. The appearance of this window depends on the operating system on which the Java environment is running.

There are 2 symbolic constants: `FileDialog.LOAD` and `FileDialog.SAVE` that encodes the operating mode of the dialog window.

After the user has specified a file or directory, its name can be read with the methods

```
String getFile()
String getDirectory()
```

Example:

```
FileDialog fd = new FileDialog(f, ”Alege un fişier”, FileDialog.LOAD);
fd.setVisible(true);
System.out.println(”A fost ales fişierul: ”+fd.getFile());
```

`Label` https://docs.oracle.com/en/java/javase/17/docs/api/java.desktop/java/awt/Label.html

This type of component is the simplest in terms of implementation or offered functionalities. A `Label` object contains textual information that can be displayed on the screen.

The methods for setting and retrieving label text are:

`void setText(String newText)` and
`String getText()`

`List` https://docs.oracle.com/en/java/javase/17/docs/api/java.desktop/java/awt/List.html

A list is a collection of texts, vertically arranged. If the list contains more items than it can display, it will automatically be equipped with a vertical *scrollbar* component.

Main *List* methods:

`void add(String text)` => adds a new row to the list
`void add(String text, int index)` => adds a new row to the list, at the specified position
`String getItem(int index)` => returns the text from the row received as parameter
`int getItemCount()` => returns the number of items in the list
`int getRows()` => returns the number of visible rows in the list
`int getSelectedIndex()` => returns the index of the selected item
`int[] getSelectedIndexes()` => returns an array of selected items indexes (if the list allows multiple selection)
`String getSelectedItem()` => returns the text of the selected item
`String[] getSelectedItems()` => returns an array of selected items texts (if the list allows multiple selection)

The `List` objects generate `ItemEvent`.

`ScrollPane` https://docs.oracle.com/en/java/javase/17/docs/api/java.desktop/java/awt/ScrollPane.html

This component type can contain any other component that can go beyond `ScrollPane` limits. Depending on the chosen policy, the `ScrollPane` component poate provides vertical and horizontal scrollbars to help the user in the complete visualization of the content.

Example:

```
ScrollPane sp = new ScrollPane();
Label eticheta = new Label(”AAABBBCCCDDDEEEFFFGGGHHHIIIJJJ”);
eticheta.setFont(new Font(”SansSerif”, Font.BOLD, 80));
sp.add(eticheta );
```

`Scrollbar` https://docs.oracle.com/en/java/javase/17/docs/api/java.desktop/java/awt/Scrollbar.html

This type of component serves to adjust the position of the contents of some components, if it is larger in size than the available space. The default position of a *scrollbar* is vertical.

`TextField` şi `TextArea`
https://docs.oracle.com/en/java/javase/17/docs/api/java.desktop/java/awt/TextField.html
https://docs.oracle.com/en/java/javase/17/docs/api/java.desktop/java/awt/TextArea.html

A `TextField` component is used to display texts (editable or not editable by the user) on a single line. `TextArea` fields can have multiple lines of text.

In order to read/set a text within the component, the following methods are used

`String getText(), String getSelectedText()` and
`void setText(String newText).`

The method *void* `setEditable(boolean editable)` specifies whether the text of that component can be changed by the user or not.
Escape characters, such as *n* or \t, can be entered within `Textarea` components.

Example:

```
TextField tf1 = new TextField(5);
tf1.setFont(new Font(”SansSerif”, Font.PLAIN, 8));
tf1.setText(”abcdefg”);

int val = 7;

TextArea ta1 = new TextArea(3, 20);
ta1.setText(”aaa\nbbb\nccc”+val);
ta1.append(tf1.getText());
```

`Container` components

The main container-type components are the following:

- **`Applet`** (deprecated)
- **`Frame`**
- **`Panel`**
- **`Dialog`**

Menu components

Java provides support for creating two types of menus: *pull-down* and *pop-up* (which appears in a position decided by code).

Pull-down menus can only exist within a container (like `Frame`). To create a window and attach a menu to it, the following steps must be completed:

- A menu bar is created an attached to the window;
- The menu is created and populated;
- The menu is associated with the menu bar.

To create a menu bar, a `MenuBar` instance is created. To attach it to a window, the method `setMenuBar()` (specific to `Frame` objects) is used.
Muenus generate `ActionEvent`.

Swing components

Java Swing technology introduces a series of new visual components. They can coexist with AWT components. The basic Swing components are similar to the AWT ones being preceded by the upper letter J, as: JFrame, JPanel, JButton, JLabel, etc., being available in the package *javax.swing.*;*

Initially, the management of swing components was done through an intermediate container obtained by:

Container content = getContentPane();

which provided complete management facilities, including background setting.

Starting with Java SE7 a default *contentPane* can be used.

The default way to display components is *Java Metal*, which, however, can be adapted to the current platform or set to another LAF (Look And Feel) available using the method:

UIManager.setLookAndFeel(…);

Swing components offer facilities far superior to those of AWT, but some are not considered to be thread safe, producing unpredictable errors that are difficult to reproduce.

The official documentation enumerates all the Swing components (https://docs.oracle.com/en/java/javase/17/docs/api/java.desktop/javax/swing/package-summary.html) Some of the components are presented below:

- `JPasswordField` – password entry field;
- `JEditorPane`, `JTextPane` – allows the display of HTML and RTF content;
- `JSpinner` - allows displaying a list of values and navigating among them with the help of arrows;
- `JToggleButton` – button that has two different states depending on the interaction with the user;
- `JSlider` – allows selecting a value with a cursor;
- `JProgressBar` – helps to visualize the progress of a task;
- `JFormattedTextField` – allows formatting the text field to enter only a specific type of text (e.g. phone number);
- `JTable` – allows the display of data in tabular form;

- `JTree` – allows the display of hierarchical data;
- `JToolTip` – displays pop-up text;
- `JSeparator` – separator within the menu bar;
- `JOptionPane` – creates pop-ups with varied content;
- `JColorChooser` – allows the selection of a color;
- etc.

Layout Managers

For the organisation of container components or containers in other containers, Java provides the interface `java.awt.LayoutManager` and the classes that use it, the most important being: `BorderLayout, FlowLayout, CardLayout, GridLayout` and `GridBagLayout`. In Swing are also used other Layout Managers such as Spring, Box, Group.

To use a different layout than the default one, either use the method:
```
contentPane.setLayout(new SpecificLayout())
```

or specify the new layout as constructor parameter:
```
JPanel jp = new JPanel(new SpecificLayout());
```

The addition of components to containers that use specific Layouts depends on the particularities of each layout.

Examples
Ex. 1 – AWT application, FlowLayout

```
import java.awt.Button;
import java.awt.FlowLayout;
import java.awt.Frame;
import java.awt.Label;
import java.awt.TextField;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

public class AWTExample extends Frame{
      public AWTExample(){
            //apelare constructorul clasei de baza
            super("AWT Example");
            //definire dimensiune fereastra
            this.setSize(300,300);
            //definire amplasare componenete
            this.setLayout(new FlowLayout());
            //adaugare componente vizuale
            Label l = new Label("Exemplu de aplicatie AWT");
            this.add(l);
            TextField td = new TextField(30);
            this.add(td);
            Button b = new Button("ok");
            this.add(b);
            //metoda prin care fereastra este inchisa de la
            //butonul 'x'. Preambul pentru capitolul de evenimente
            this.addWindowListener (new WindowAdapter(){
                  public void windowClosing(WindowEvent e){
                        dispose();
                  }
```

```
            });

            //metoda prin care fereastra este afisata
            //fara aceasta metoda, aplicatia nu va afisa nimic
            this.setVisible(true);
      }

      public static void main (String[] args) {
            new AWTExample();
      }
}
```

Ex. 2 – Swing application, FlowLayout

```
import java.awt.FlowLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTextField;

public class SwingExample extends JFrame{
      public SwingExample(){
            //apelare constructorul clasei de baza
            super("Swing Example");
            //definire dimensiune fereastra
            this.setSize(300,300);
            //definire amplasare componenete
            this.setLayout(new FlowLayout());//modificare din implicit BorderLayout

            //adaugare componente vizuale
            JLabel l = new JLabel("Exemplu de aplicatie Swing");
            this.add(l);
            JTextField td = new JTextField(30);
            this.add(td);
            JButton b = new JButton("ok");
            this.add(b);
            //metoda prin care fereastra este inchisa de la butonul 'x'.
            this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

            //metoda de afisare a ferestrei fara aceasta, aplicatia nu va face nimic
            this.setVisible(true);
      }

      public static void main (String[] args) {
            new SwingExample();
      }
}
```

Ex. 3 - AWT application, menu components

```
import java.awt.CheckboxMenuItem;
import java.awt.Frame;
import java.awt.Menu;
import java.awt.MenuBar;
import java.awt.MenuItem;
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;

class Test{
      public static void main(String... args) {
            Frame f = new Frame("Fereastra principala");
```

```
            f.setSize(200, 200);

            //bara de meniu
            MenuBar mb = new MenuBar();
            //atasarea barei de meniu
            f.setMenuBar(mb);

            //un prim meniu
            Menu menu1 = new Menu("Primul meniu");
            menu1.add(new MenuItem("Articol 1"));
            menu1.add(new MenuItem("Articol 2"));
            menu1.addSeparator();
            menu1.add(new CheckboxMenuItem("Articol checkbox 1"));

            //un submeniu
            Menu submenu1 = new Menu("Submeniu 1");
            submenu1.add(new MenuItem("AAA"));
            submenu1.add(new MenuItem("BBB"));

            menu1.add(submenu1);

            //adaugarea meniului la bara de meniu
            mb.add(menu1);

            f.addWindowListener(new WindowListener() {
                  public void windowOpened(WindowEvent e) {
                                    }
                  public void windowClosing(WindowEvent e) {
                        System.exit(1);
                                    }
                  public void windowClosed(WindowEvent e) {
                                    }
                  public void windowIconified(WindowEvent e) {
                                    }
                  public void windowDeiconified(WindowEvent e) {
                                    }
                  public void windowActivated(WindowEvent e) {
                                    }
                  public void windowDeactivated(WindowEvent e) {
                                    }
                  });
            //fereastra este facuta vizibila
            f.setVisible(true);
      }
}
```

Ex.4 : Menu, Swing components

```
import javax.swing.*;
import java.awt.*;      //BorderLayout
public class SwingGuiM{
      public static void main(String args[ ]){
// Crearea ferestrei principale Frame
JFrame frame = new JFrame("Chat Frame");
frame.setSize(400,400);
// Crearea unei bare de meniu MenuBar si adaugare componente
JMenuBar mb = new JMenuBar( );
JMenu m1 = new JMenu("FILE");
JMenu m2 = new JMenu("Help");
mb.add(m1);
mb.add(m2);
JMenuItem m11 = new JMenuItem("Open");
```

```
JMenuItem m22 =new JMenuItem("Save as");
m1.add(m11);
m1.add(m22);

JPanel panel = new JPanel( );
JLabel label = new JLabel("Enter Text");
JTextField tf = new JTextField(10);// accepts upto 10 characters
JButton send = new JButton("Send");
JButton reset = new JButton("Reset");
panel.add(label); //Componente adaugate folosind implicit FlowLayout la JPanel
panel.add(tf);
panel.add(send);
panel.add(reset);
//Text Area plasat in centru
JTextArea ta = new JTextArea( );
//adaugare componente in frame folosind implicit BorderLayout
frame.add(BorderLayout.SOUTH, panel);
frame.add(BorderLayout.NORTH, mb);
frame.add(BorderLayout.CENTER, ta);
frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
frame.setVisible(true);       }
      }
```

Ex.5: Jradio, ButtonGroup

```
import java.awt.Color;
import java.awt.GridLayout;
import javax.swing.BorderFactory;
import javax.swing.ButtonGroup;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JRadioButton;

public class SixChoicePanel extends JPanel {
      private static final long serialVersionUID = 1L;
public SixChoicePanel(String title, String[ ] buttonLabels){
      super(new GridLayout(3, 2));
      setBackground(Color.cyan);
      setBorder(BorderFactory.createTitledBorder(title));
      ButtonGroup group = new ButtonGroup();
      JRadioButton option;

      int halfLength = buttonLabels.length/2;
      for(int i=0; i<halfLength; i++) {
         option = new JRadioButton(buttonLabels[i]);
         group.add(option);
         add(option);
         option = new JRadioButton(buttonLabels[i+halfLength]);
         group.add(option);
         add(option);
         }

      }
      public static void main(String args[ ]) {
          JFrame frame = new JFrame("SixChoicePanel");
          String title = "SixChoice";
          String [ ] bL= {"a", "b", "c", "d", "e", "f"};
          frame.add(new SixChoicePanel(title,bL));
          frame.pack();
          //frame.setSize(300, 200);
          frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
          frame.setVisible(true);
        }
       }
```

Ex. 6 Swing, tabbed pane

```
import java.awt.*;
import javax.swing.*;
import javax.swing.UIManager.LookAndFeelInfo;

public class TabbedView {
       final static String BUTTONPANEL = "Tab with JButtons";
       final static String TEXTPANEL = "Tab with JTextField";
       final static int extraWindowWidth = 100;

       public void addComponentToPane(Container pane) {
            JTabbedPane tabbedPane = new JTabbedPane();
            //Create the "cards".
            JPanel card1 = new JPanel();

            card1.add(new JButton("Button 1"));
            card1.add(new JButton("Button 2"));
            card1.add(new JButton("Button 3"));

            JPanel card2 = new JPanel();
            card2.add(new JTextField("TextField", 20));

            tabbedPane.addTab(BUTTONPANEL, card1);
            tabbedPane.addTab(TEXTPANEL, card2);

            pane.add(tabbedPane, BorderLayout.CENTER);
       }

       private static void createAndShowGUI() {
            JFrame frame = new JFrame("TabbedView");
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

            TabbedView demo = new TabbedView();
            demo.addComponentToPane(frame.getContentPane());

            frame.pack();
            frame.setVisible(true);
       }

       public static void main(String[] args) {
            LookAndFeelInfo[ ] lafs =
                        UIManager.getInstalledLookAndFeels();
            for (LookAndFeelInfo laf: lafs)
                  System.out.println(laf.getClassName());
            try {
               // UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");
               // UIManager.setLookAndFeel("javax.swing.plaf.nimbus.NimbusLookAndFeel");
        //UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
              UIManager.setLookAndFeel("com.sun.java.swing.plaf.motif.MotifLookAndFeel");
                  }
            catch (UnsupportedLookAndFeelException ex) {
                        ex.printStackTrace();
            }
            catch (IllegalAccessException ex) {
                        ex.printStackTrace();
            }
```

```
            catch (InstantiationException ex) {
                  ex.printStackTrace();
            }
            catch (ClassNotFoundException ex) {
                  ex.printStackTrace();
            }

                   createAndShowGUI();
                  }//main
}
```

Ex.7 TextField, TextArea, button events

```
import java.awt.Button;
import java.awt.Font;
import java.awt.Frame;
import java.awt.TextArea;
import java.awt.TextField;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;


class CEditF extends Frame{

      private static final long serialVersionUID = 1L;
      TextField tf1;
      TextArea ta1;
      Button b;
      public CEditF(String s) {
            super(s);
            setLayout(null);
            this.addWindowListener(new WindowAdapter() {
               public void windowClosing(WindowEvent windowEvent){
                   System.exit(0);   }
             });
      init();
      }

      public void init() {
            tf1 = new TextField();
            tf1.setFont(new Font("SansSerif", Font.PLAIN, 10));
            tf1.setBounds(50, 50, 100, 30);
            b=new Button("Copy");
            b.setBounds(200, 50, 50, 30);
            b.addActionListener(new ActionListener() {
                  public void actionPerformed(ActionEvent ae) {
                        ta1.append(tf1.getText()+'\n');
                  }
            });
            add(b);
            ta1 = new TextArea();
            ta1.setBounds(50, 100, 100, 100);
            add(tf1);
            add(ta1);
```

```
            }
}

public class EditWithEv{
       public static void main(String args[])
         {
               CEditF f=new CEditF("Campuri de editare");
               f.setSize(200, 250);
               f.setVisible(true);
         }
}
```

Individual work

Complexity *

1. Implement a Java application that uses **GridLayout** for arranging the components specific to a computing application. (TextField for I/O operations, buttons for numbers, mathematical operations, screen cleaning, etc.).

2. Implement a Java application that uses **GridBagLayout** for arranging the components specific to a computing application. (TextField for I/O operations, buttons for numbers, mathematical operations, screen cleaning, etc.).

3. Implement a **TabbedView** Java application that displays in each graphical panel:
- a drawing area
- the necessary components for selecting a font type, size and color
- the necessary components for choosing a geometrical shape (from a predefined set) and a size for the side / radius (as the case)

Complexity **

4. Create an application which takes from two text fields your name and the group you are part of and displays this info in a label colored in RGB (122,123,129).

5. Implement a graphical application that displays 3 TextField components for setting the amounts of R, G and B from a color. After pressing a button, a Label component will display a text representing the specified quantities, with the resulting color, if the values are correct (numerical, between 0-255). Oherwise, an error message will be displayed with red. Try to use exceptions for validating the input.

6. Write a Java application which includes a sign-up form for an online course. The form includes information regarding the name, surname, year of study, faculty, financing (tax/budget) and the course. The year of study, faculty and course are drop-down lists, and the financing is a check-box field. În a TextArea field print the filled-in information after the Sign-up button is pressed.

Complexity ***

7. Implement a Java application that concatenates in a TextArea component the content of the text files selected by the user, as the user chooses the files. Each file's content is preceded by the file's name.