Graphical User Interface using widgets

Object Oriented Programming
2024 First Semester
Shin-chi Tadaki (Saga University)

GUI (Graphical User Interface) in Java

- GUI libraries in general programming environments
  - X11 with c/c++, etc.
  - Generally OS dependent
- Java provides GUI libraries distributed with JDK.
  - OS independent: running on any OS with JVM
  - Working under OS dependent window managers

GUI programming as OOP

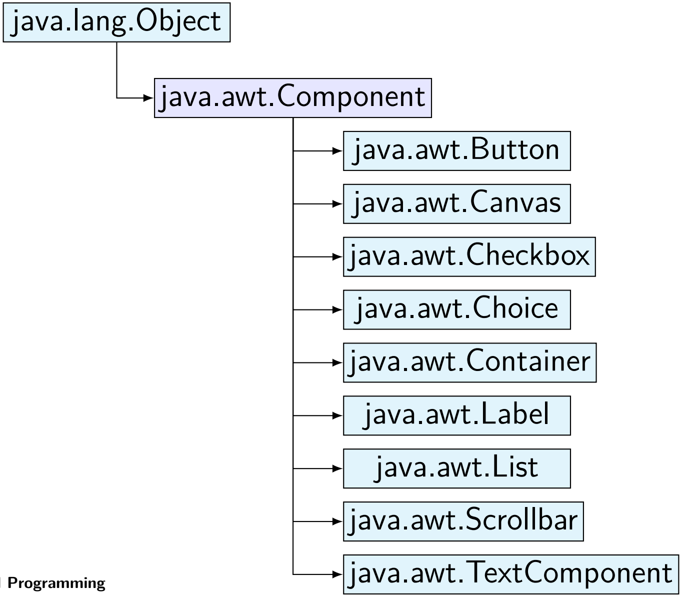
- GUI applications are constructed with various widgets (*window gadgets*)
- Each widget has own properties and methods
  - Properties: color, size, etc.
  - Methods: action, property change, visible, etc.
- Widgets communicates other widgets through methods in an application.
- Fundamental widgets are used for applications by extensions.
  - GUI main windows by extending `JFrame`
  - Widget containers by extending `JPanel`

java.awt: Abstract Windows Toolkit

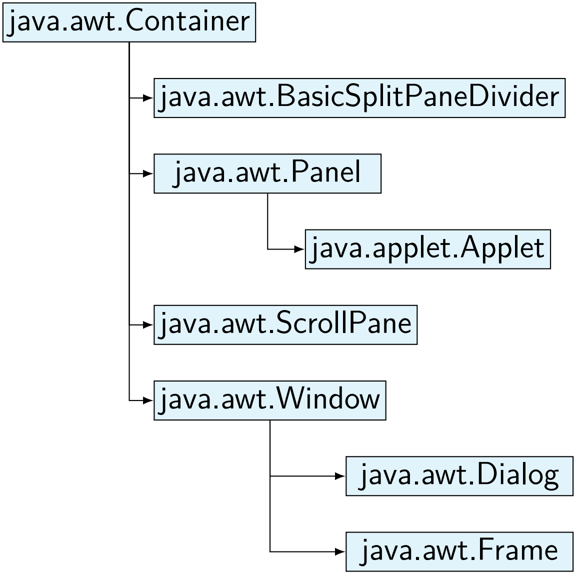
`java.awt` provides fundamental functionalities for GUI applications.

- Fundamental graphical properties
  - `Color`, `BasicStroke`, `Font`, etc.
- Fundamental widgets
  - panels, buttons, etc.
- Fundamental events
  - mouse, keyboard, property changes, etc.

Hieralchy of `java.awt`



Hieralchy of `java.awt`: cont.

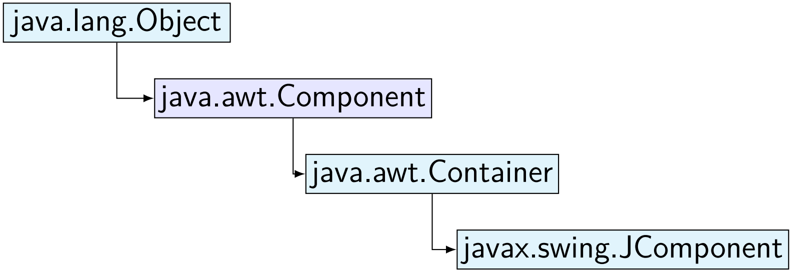


javax.swing

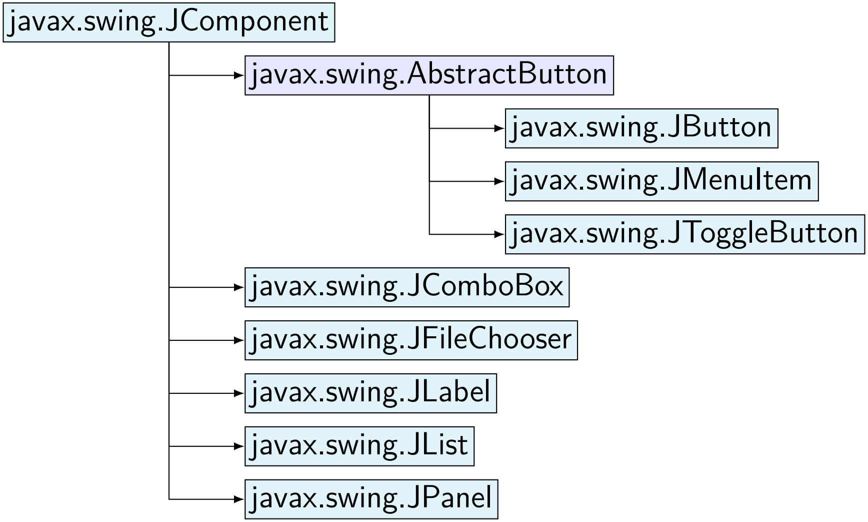
javax.swing is a library for GUI applications in Java. It is an extension of java.awt.

- Enriching widgets
- Completely OS independence
    - Control under OS window manager
    - Separate Look-and-Feel
- Lightweight
- Running as threads

Hieralchy of swing widgets



Hieralchy of swing widgets: cont.

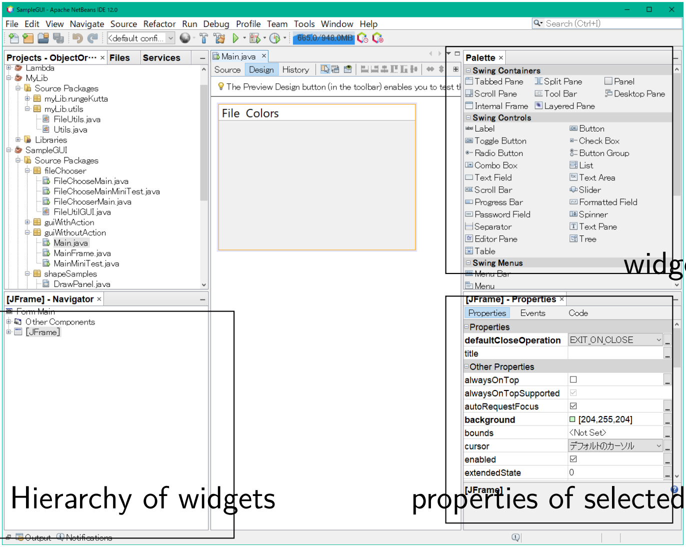


...

swing components

- `javax.swing.JFrame`
    - Main window of applications
    - Put `JPanel` and `JMenuBar` instances onto this component
- `javax.swing.JPanel`
    - Put widgets on this components
    - Used for drawing
- `javax.swing.JMenuBar`
    - Menu bar at the top of applications
    - Put `javax.swing.JMenu` instances on this component

Layout Design in NetBeans



Properties

defaultCloseOperation	EXIT_ON_CLOSE
title	

Other Properties

alwaysOnTop	☐
alwaysOnTopSupported	✓
autoRequestFocus	✓
background	[204,255,204]
bounds	<Not Set>
cursor	デフォルトのカーソル
enabled	✓
extendedState	0

widgets

Hierarchy of widgets

properties of selected widget

Constructing GUI in NetBeans

- Create a project as *Java with Ant* project
- Create `JFrame` form for *Main* class of the application
  - `New`→`JFrame form`
  - At widget hierarchy: `Set Layout`→`BorderLayout`
  - The *Main* class is defined as a new class by extending `JFrame`

Configuring widgets

- Configuring widgets using mouse
  - In `Navigation`: Drag a component from the palette
  - Change some properties if necessary
- Creating `JMenuBar`
  - Two `JMenu` instances `File` and `Edit` are added initially.
  - Add `JMenu` and `JMenuItem`

Exercise

- Create a new `JFrame` instance.
- Add a `JMenuBar` instance
- Add a `JMenuItem` instance to the `JMenuBar`
- Set some attributes to `JMenuBar` and `JMenuItem` instances

Notice at creating new JFrame instances

- Properties and layout are stored in `*.form` file.
    - Parts of source files are not allowed to edit, because some configurations are stored in `*.form` files.

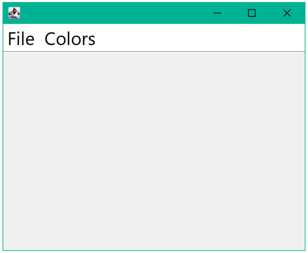
Today's example program

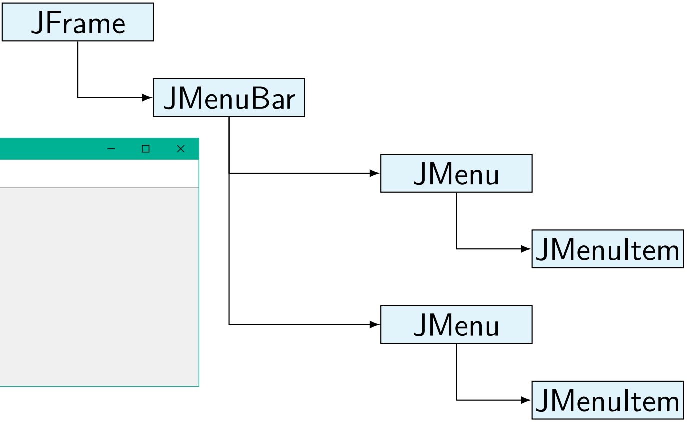
`https://github.com/oop-mc-saga/GUI1`

- `guiWithoutAction`
- `guiWithAction`
- `fileChooser`
- `simpleTimer`

GUI without actions

- This application has a menu bar with two menus.
- Those does not have any actions.




$$\begin{tikzcd}
\text{JFrame} \arrow[d] & & \\
\text{JMenuBar} \arrow[r] \arrow[rd] & \text{JMenu} \arrow[r] & \text{JMenuItem} \\
& \text{JMenu} \arrow[r] & \text{JMenuItem}
\end{tikzcd}$$

Two menus in this application

- The first menu `fileMenu`
  - has menu item `exit`,
  - which is added through the *design* interface of NetBeans.
- The second menu `selectColors`
  - has menu items for selecting a color defined in the enum type `ColorItem`.
  - Those menu items are added in the constructor

Main part

```java
public class Main extends javax.swing.JFrame {

    public enum ColorItem {//Define colors as enum instance
        ORANGE(Color.ORANGE), YELLOW(Color.YELLOW), GREEN(Color.GREEN);
        private final Color color;

        ColorItem(Color color) {//Each color has a java.awt.color
        ↪   instance
            this.color = color;
        }

        public Color getColor() {  return color;  }
    }

    public Main() {
        initComponents();
        Font font = new Font("MS UI Gothic", 0, 24);
        for (ColorItem m : ColorItem.values()) {
            JMenuItem item = new JMenuItem(m.toString());
            item.setFont(font);
            selectColors.add(item);
        }
    }
    ...
}
```

`initComponents()`

- This method is generated automatically with `form` file through NetBeans
- What `initComponents()` does is
    - Inserting widgets and laying out them
    - Setting properties of widgets
- Operations in `initComponents()` are defined through the *design* interface of NetBeans.

`enum` type

- `enum` allows us to define a set of named constants.
- Items in `enum` can have properties and methods.
- `enum` types are useful for `switch-case` clauses.

Example 5.1: enum

```java
public class EnumExample {
    public static enum ColorName{
        RED, GREEN, BLUE;
    }

    public static void main(String[] args) {
        ColorName colorName = ColorName.BLUE;
        String colorCode = null;
        switch(colorName){
            case RED -> colorCode = "#FF0000";
            case GREEN -> colorCode=  "#00FF00";
            case BLUE -> colorCode = "#0000FF";
            default -> {
            }
        }
        System.out.println(colorCode);
    }

}
```

Exercise

Add a new menu for selecting color (see quiz).