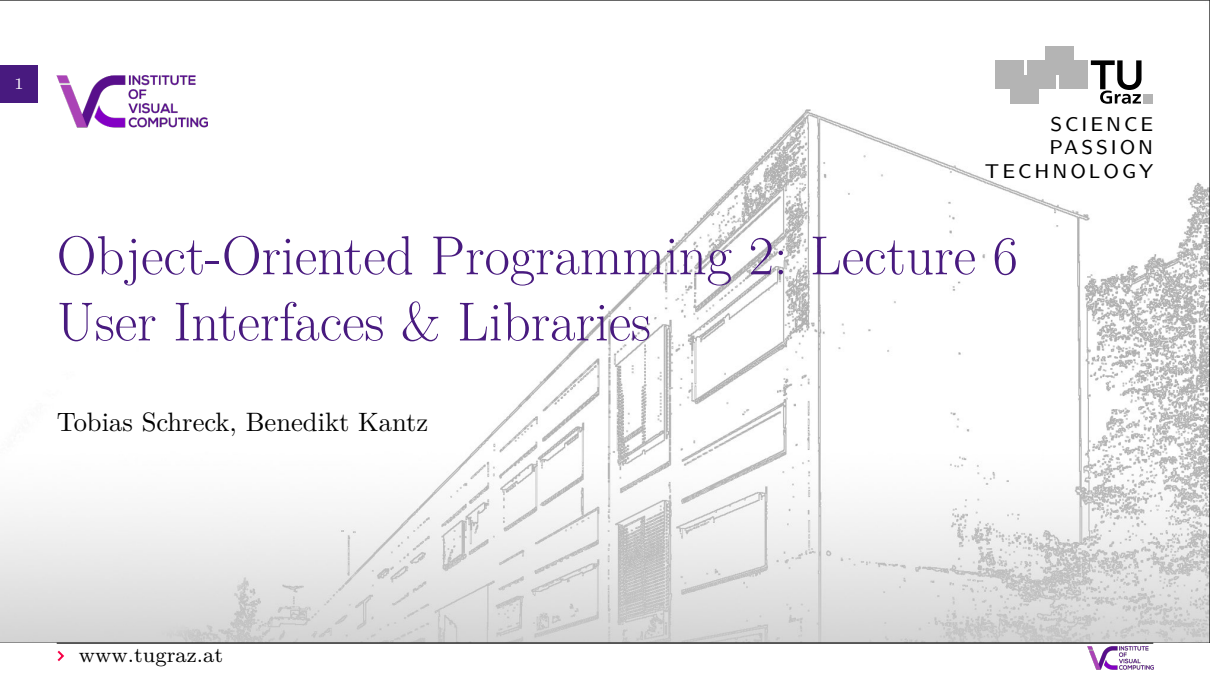
INSTITUTE OF VISUAL COMPUTING

TU Graz
SCIENCE
PASSION
TECHNOLOGY

Object-Oriented Programming 2: Lecture 6
User Interfaces & Libraries

Tobias Schreck, Benedikt Kantz



Which UI libraries have you used for your studies?

Why GUI Libraries?

- Encapsulate complex UI logic and components for reuse and maintainability.
- Promote separation of concerns (UI vs. business logic) → Model View Presenter (MVP)!
- Facilitate rapid development with pre-built, tested components.
- Improve consistency and scalability of user interfaces.

Why GUI Libraries?

- Encapsulate complex UI logic and components for reuse and maintainability.
- Promote separation of concerns (UI vs. business logic) → Model View Presenter (MVP)!
- Facilitate rapid development with pre-built, tested components.
- Improve consistency and scalability of user interfaces.

Why GUI Libraries?

- Encapsulate complex UI logic and components for reuse and maintainability.
- Promote separation of concerns (UI vs. business logic) → Model View Presenter (MVP)!
- Facilitate rapid development with pre-built, tested components.
- Improve consistency and scalability of user interfaces.

Why GUI Libraries?

- Encapsulate complex UI logic and components for reuse and maintainability.
- Promote separation of concerns (UI vs. business logic) → Model View Presenter (MVP)!
- Facilitate rapid development with pre-built, tested components.
- Improve consistency and scalability of user interfaces.

Why GUI Libraries?

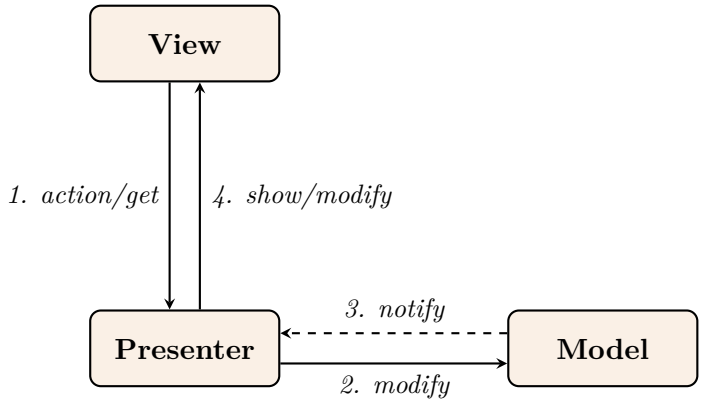
- Encapsulate complex UI logic and components for reuse and maintainability.
- Promote separation of concerns (UI vs. business logic) → Model View Presenter (MVP)!
- Facilitate rapid development with pre-built, tested components.
- Improve consistency and scalability of user interfaces.

Model View Presenter (MVP) I
Concept

Model View Presenter (MVP) by Potel [1]

- **Model**: State
- **View**: Display
- **Presenter**: Manager

MVP II
Concept



MVP

Advantages

- Clear separation of concerns (UI, logic, data)
- Easier to test and maintain
- Promotes code reuse and modularity
- Facilitates parallel development (UI vs. logic)

MVP

Advantages

- Clear separation of concerns (UI, logic, data)
- Easier to test and maintain
- Promotes code reuse and modularity
- Facilitates parallel development (UI vs. logic)

Disadvantages

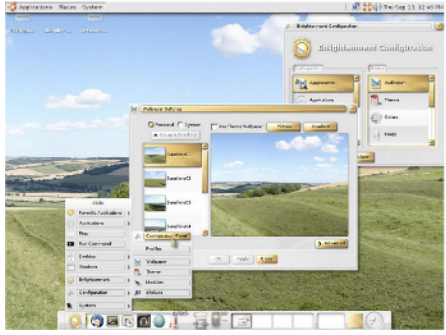
- More boilerplate code compared to simpler patterns
- Can be overkill for small applications
- Communication between components may become complex

GUI Libraries
UI Types

- Historic: Wiring, punched cards
- Command Line Interface (CLI)
- Graphical User Interface (GUI):
  - WIMP (Windows, Icons, Menus, Pointer)
  - Others
- Natural user interface (NUI)
  - Touch, gestures, eye tracking, speech, ...
- See also subject “Human-computer Interaction” and “User Interface Design”

GUI Libraries
WIMP Elements

- Windows
- Widgets, e.g., Labels, Buttons, CheckBoxes, RadioButtons, Sliders, Textbox, etc.
- Layouts, controlling the arrangement of elements, e.g., row, column, matrix
- Scrollbars, edges, close button



GUI Libraries

Examples: Python PyQt

- Modern, native-looking GUIs
- C++ bindings to Qt platform[a]
- Cross-platform (Windows, macOS, Linux)
- OOP-Style Access

```
from PyQt5.QtWidgets import QMainWindow,
↪  QApplication, QPushButton
class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Hello OOP2")
        button = QPushButton("Press me!")
        button.pressed.connect(lambda:
        ↪  button.setText("You clicked me!"))
        self.setCentralWidget(button)
        self.show()
app = QApplication([])
w = MainWindow()
app.exec()
```

[a] https://www.qt.io/

GUI Libraries

Examples: Python tkinter

- Built-in standard Python library[a]
- Simple API for basic GUIs
- Cross-platform (Windows, macOS, Linux)

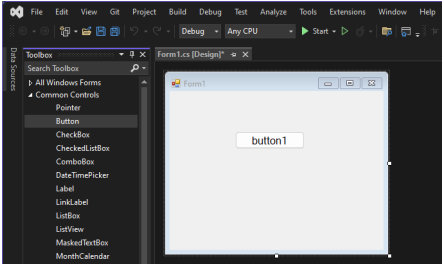
[a] https://tkdocs.com/

```
import tkinter as tk
window = tk.Tk()
window.title("Hello OOP2")

button = tk.Button(text="Press me :)")
def handle_button_press(event):
    button.config(text="You clicked
    ↪ me!")
button.bind("<ButtonPress>",
↪ handle_button_press)
button.pack()

# Start the event loop.
window.mainloop()
```

GUI Libraries

Examples: .NET WinForms

- Classic .NET desktop UI framework
- Rapid application development with drag-and-drop designer [a]
- Windows-only (!)



[a] Tutorial

GUI Libraries

Examples: .NET WPF (Windows Presentation Foundation)

- Modern .NET desktop UI framework
- Rapid application development with drag-and-drop designer [a]
- Declarative UI with XAML
- Windows-only (!)

[a] Tutorial

```
<Grid>
    <TextBlock
    ↪ HorizontalAlignment="Left"
    ↪ Margin="252,47,0,0"
    ↪ TextWrapping="Wrap"
      Text="Hello OOP2."
      ↪ VerticalAlignment="Top" />
    <RadioButton x:Name="HelloBtn"
    ↪ Content="Hello"
    ↪ HorizontalAlignment="Left"
      Margin="297,161,0,0"
      ↪ VerticalAlignment="Top" />
</Grid>
```

GUI Libraries

Examples: Java AWT

- Abstract Window Toolkit AWT
- Early Java GUI framework (since Java 1.0),
- Provides basic UI components (Button, Label, TextField, etc.)
- Heavyweight components
- Platform-dependent look and feel

```
import java.awt.*;
public class AwtExample {
    public static void main(String[]
    ↪ args) {
        Frame frame = new
        ↪ Frame("OOP2");
        Button button = new
        ↪ Button("Click Me :)");
        button.addActionListener(e ->
        ↪ button.setLabel("You
        ↪ clicked me!"));
        frame.add(button);
        frame.setSize(300, 200);
        frame.setVisible(true);
    }
}
```

GUI Libraries

Examples: Java Swing

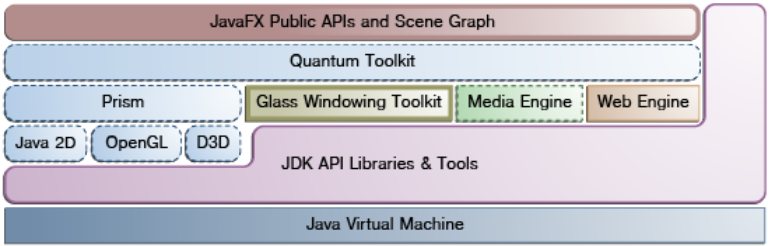
- Successor to AWT (since Java 1.2)
- Lightweight components (drawn in Java)
- Pluggable look and feel
- Rich set of widgets (JButton, JLabel, JTable, etc.)
- Supports Model View Controller (MVC)

```
import javax.swing.*;
public class SwingExample {
    public static void main(String[]
    ↪  args) {
        JFrame frame = new
        ↪  JFrame("OOP2");
        JButton button = new
        ↪  JButton("Click Me :)");
        button.addActionListener(e ->
        ↪  button.setText("You clicked
        ↪  me!"));
        frame.add(button);
        frame.setSize(300, 200);
        frame.setVisible(true);
    }
}
```

JavaFX
Introduction

- Successor to Swing for Java desktop applications
- Supports FXML for declarative UI design
- Includes built-in controls, charts, and CSS styling
- Enables hardware-accelerated graphics and multimedia

JavaFX
Architecture



JavaFX Architecture, source

JavaFX I

Terminology

- Scene Graph
  - Tree structure representing all UI elements
- Stage
  - A window
- First stage
  - The initial window given to an application as a parameter
  - Additional stages can be created on the fly

JavaFX II
Terminology

- Scene
  - A container to fill a stage; contains the graphical application content, e.g., widgets, shapes, subcontainers etc.
- Layout
  - How to layout the UI elements in a scene

JavaFX III
Terminology

- Event
  - Caused by user input (e.g., mouse click)
  - Handled by event listeners
  - Event provides details for the processing (e.g., mouse coordinates)
- JavaFX Application Thread

JavaFX
Basic Example

- Extends `Application`
- Override `start(Stage)`
- Create UI elements and set up the scene
- Call `launch()` in `main`

```
import javafx.application.Application;
import javafx.scene.*;
public class JavaFXExample
extends Application {
    public static void main(String[] args) {
        launch(args);
    }
    @Override
    public void start(Stage stage) {
        Label label = new Label("Hey!");
        Scene scene = new Scene(new
        ↪  StackPane(label), 300, 200);
        stage.setScene(scene);
        stage.setTitle("Hello OOP2!");
        stage.show();
    }
}
```

JavaFX
Properties

- JavaFX uses **properties** to enable observability and binding
- Properties wrap values (e.g., `StringProperty`, `IntegerProperty`)
- Support listeners for change notification
- Used for UI controls, models, and more

```
StringProperty name = new
↪  SimpleStringProperty("Alice");
name.addListener((obs, oldVal, newVal)
↪  -> {
    System.out.println("Name changed to
    ↪  " + newVal);
});
name.set("Bob");
```

JavaFX
Bindings Example

- **Bindings** connect properties so that changes propagate automatically
- Example: Bind a `Label` to a `StringProperty`

```
StringProperty text = new
↪  StringProperty();
Label label = new Label();
label.textProperty().bind(text);
```

JavaFX

Unidirectional vs. Bidirectional Binding

- **Unidirectional binding:** One property observes another; changes propagate in one direction only.
- **Bidirectional binding:** Both properties observe each other; changes propagate in both directions.
- Useful for synchronizing UI controls and model data.

```
// Unidirectional
label.textProperty().bind(textProperty);
// Bidirectional
textField.textProperty()

    ↪  .bindBidirectional(model.textPrope
```

JavaFX

Observable Collections

- JavaFX provides `ObservableList`, `ObservableMap`, and `ObservableSet`
- Collections notify listeners about changes (add, remove, update)
- Used for dynamic UI elements (e.g., `ListView`, `TableView`)

```
ObservableList<String> items =
↪  FXCollections.observableArrayList();
items.addListener(
    (ListChangeListener<String>) change -> {
    while (change.next()) {
        if (change.wasAdded()) {
            System.out.println("Added: " +
            ↪  change.getAddedSubList());
        }
    }
});
items.add("Hello");
```

JavaFX

Threading and UI Access

```
Label label = new Label("Hello OOP2!");
//...
Thread t = new Thread(()->{
    Thread.sleep(100);
    label.setText("Updated from
    ↪  background thread");
});
```

JavaFX

Threading and UI Access

```
Label label = new Label("Hello OOP2!");
//...
Thread t = new Thread(()->{
    Thread.sleep(100);
    label.setText("Updated from
    ↪  background thread");
});
```

JavaFX

Threading and UI Access

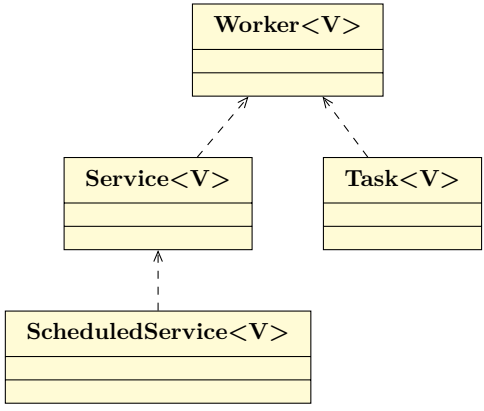
- **JavaFX Application Thread** is responsible for all UI updates and event handling.
- **Rule:** Only the JavaFX Application Thread may access or modify UI elements.
- Use `Platform.runLater()` to schedule code on the JavaFX Application Thread:

```
Label label = new Label("Hello OOP2!");
//...
Thread t = new Thread(()->{
        Thread.sleep(100);
        Platform.runLater(() -> {
            label.setText("Updated from
            ↪ background thread");
        });
});
```

JavaFX

`javafx.concurrent`

- `Worker`: Interface for background operations with progress and state tracking.
- `Task`: designed for background computation.
- `Service`: Reusable wrapper for `Task`, manages task lifecycle and restartability.
- `ScheduledService`: Extension of `Service` for periodic execution.



JavaFX
Example code

- See Gitlab example code!

8 Golden Rules (by Shneiderman et al. [2]) I

- A set of rules for Visualizations/User Interfaces!
- Again: use it as guidance, not strict rules...

8 Golden Rules (by Shneiderman et al. [2]) II

1. Strive for consistency

- Use familiar icons, terminology, and layouts.
- Maintain uniformity in colors, fonts, and actions.
- Predictable behavior reduces user confusion.

Example: All boxes in use the same button order for "OK" and "Cancel".

8 Golden Rules (by Shneiderman et al. [2]) III

2. Seek universal usability

- Provide accelerators (e.g., keyboard shortcuts), but also explanations for novices.
- Allow customization for power users.

Example: Pressing `Ctrl+C` to copy text instead of using the menu.

8 Golden Rules (by Shneiderman et al. [2]) IV

3. Offer informative feedback

- Give immediate, clear responses to user actions.
- Use appropriate feedback for different actions.

Example: Showing a progress bar during file upload.

8 Golden Rules (by Shneiderman et al. [2]) V

4. Design dialogs to yield closure

- Group actions into meaningful sequences.
- Provide clear beginnings, middles, and ends.

Example: Displaying a confirmation message after submitting a form.

8 Golden Rules (by Shneiderman et al. [2]) VI

5. Prevent errors.

- Prevent errors where possible.
- Provide helpful, specific error messages.
- Allow easy recovery from mistakes.

Example: Highlighting invalid fields in a form and suggesting corrections.

8 Golden Rules (by Shneiderman et al. [2]) VII

6. Permit easy reversal of actions

- Support undo and redo functionality.
- Make actions reversible to encourage exploration.

Example: Undoing the last drawing stroke in a graphics editor.

8 Golden Rules (by Shneiderman et al. [2]) VIII

7. Support internal locus of control

- Let users initiate and control actions.
- Avoid unexpected system behaviors.

Example: Users choose when to save or submit their work.

8 Golden Rules (by Shneiderman et al. [2]) IX

8. Reduce short-term memory load

- Minimize information users must remember.
- Use recognition over recall.

Example: Autofilling address fields based on previous entries.

Feedback

Bibliography I

References

[1] M. Potel, **Mvp: Model-view-presenter the taligent programming model for c++ and java,** *Taligent Inc*, vol. 20, 1996.

[2] B. Shneiderman, M. Cohen, S. Jacobs, C. Plaisant, N. Diakopoulos, and N. Elmqvist, **Designing the User Interface**. London, England, UK: Pearson International, 2017, ISBN: 978-1-29215392-6. [Online]. Available: https://elibrary.pearson.de/book/99.150005/9781292153926.