# 23. GUI 프로그래밍

## AWT와 Swing 개요

### 1️⃣ AWT (Abstract Window Toolkit)

#### 💡 개요

- **Java 초창기(1.0)**부터 제공된 GUI 라이브러리
- OS 고유의 **네이티브 위젯**(버튼, 텍스트 필드 등)을 사용함
- **플랫폼 종속적** 특성 (운영체제마다 다르게 보일 수 있음)
- 성능이 상대적으로 빠르지만 유연성과 확장성은 부족

#### 🧩 주요 컴포넌트

- `Frame`, `Button`, `TextField`, `Label`, `Panel`, `Checkbox`, `Choice`, `List`

#### 🔧 기본 AWT 예제

```
import java.awt.*;

public class AwtExample {
    public static void main(String[] args) {
        Frame frame = new Frame("AWT Example");
        Button btn = new Button("Click Me");

        frame.add(btn); // 기본 BorderLayout - Center에 배치됨
        frame.setSize(300, 200);
        frame.setVisible(true);
    }
}
```

---

### 2️⃣ Swing (Java Foundation Classes 일부)

#### 💡 개요

- **JDK 1.2**부터 도입된 AWT의 상위 확장 GUI 프레임워크
- AWT 위에 구현되어 있지만, 대부분의 컴포넌트는 **순수 Java 기반**
- **플랫폼 독립적**이고 **경량 컴포넌트** 제공
- **룩앤필(Look and Feel)**을 설정 가능 (예: 윈도우 스타일, 모티프, Nimbus 등)

#### 🧩 주요 컴포넌트

- `JFrame`, `JPanel`, `JButton`, `JTextField`, `JLabel`, `JCheckBox`, `JComboBox`, `JList`, `JTable`, `JDialog` 등
- 모든 컴포넌트 이름이 `J`로 시작함

### 🧪 기본 Swing 예제

```
import javax.swing.*;

public class SwingExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Swing Example");
        JButton btn = new JButton("Click Me");

        frame.add(btn);
        frame.setSize(300, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // 닫기 버튼 동작
        frame.setVisible(true);
    }
}
```

---

## 3️⃣ AWT vs Swing 비교표

| 항목 | AWT | Swing |
| --- | --- | --- |
| 도입 시기 | Java 1.0 | Java 1.2 |
| 컴포넌트 종류 | OS 네이티브 컴포넌트 사용 | 순수 Java로 작성된 경량 컴포넌트 사용 |
| 플랫폼 독립성 | 낮음 (OS 따라 다름) | 높음 (모든 OS에서 동일한 UI 제공) |
| 커스터마이징 | 어려움 | 자유로움 (룩앤필 지원) |
| 스레드 모델 | 싱글스레드 UI | 이벤트 디스패치 스레드(EDT) 사용 |
| 예: 버튼 이름 | `Button` | `JButton` |

---

## 4️⃣ Swing 활용 팁

- **EDT(Event Dispatch Thread)**를 통해 UI를 조작해야 함:

```
SwingUtilities.invokeLater(() -> {
    // UI 생성 또는 갱신 작업
});
```

- **레이아웃 관리자** 사용이 중요:
  - `BorderLayout`, `FlowLayout`, `GridLayout`, `BoxLayout`, `GridBagLayout` 등

---

## 5️⃣ 언제 Swing을 쓰고, 언제 안 쓸까?

| 사용 추천 | 사용 비추천 |
| --- | --- |
| 간단한 데스크탑 도구 | 현대적 UI가 필요한 경우 |
| 사내 전용 툴 | 웹, 모바일 앱 개발 시 |
| 빠른 프로토타이핑 | 크로스 플랫폼 미지원 경우 |

---

## ✅ 요약

- **AWT는 고전적이고 OS 의존적인 GUI 프레임워크**
- **Swing은 더 강력하고 유연한 순수 Java 기반 GUI 도구**
- 현재는 **JavaFX 또는 웹 기반 UI**가 Swing보다 더 선호되지만, 여전히 수많은 레거시 앱이 Swing 기반으로 존재

# 주요 컴포넌트 (`JFrame`, `JButton`, `JTextField`)

## 1️⃣ `JFrame` – 윈도우 창(프레임)

### ✅ 설명

- 애플리케이션의 **기본 창** 역할을 하는 컨테이너
- 기본적으로 `BorderLayout`을 가짐
- `setVisible(true)` 호출 전까지는 화면에 표시되지 않음
- 닫기 버튼 이벤트 설정은 `setDefaultCloseOperation()`으로 처리

### ✨ 예제

```
JFrame frame = new JFrame("My Application");
frame.setSize(400, 300);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // 창 닫을 때 종료
frame.setVisible(true);
```

---

## 2️⃣ `JButton` – 버튼

### ✅ 설명

- 클릭 가능한 버튼
- 버튼에 문자열이나 아이콘을 설정할 수 있음
- 클릭 이벤트는 `ActionListener`로 처리

### ✨ 예제

```
JButton button = new JButton("Click Me");

button.addActionListener(e -> {
    System.out.println("Button clicked!");
});
```

`JButton`은 다양한 형태로 생성 가능:

```
new JButton(); // 빈 버튼
new JButton("텍스트");
new JButton(new ImageIcon("image.png"));
```

---

## 3️⃣ `JTextField` – 한 줄 텍스트 입력 필드

### ✅ 설명

- 사용자로부터 문자열 입력을 받는 **단일 행 입력창**
- `getText()`, `setText()`로 값 읽고 쓰기 가능
- 기본 길이는 열 수(int columns)로 지정

### ✨ 예제

```
JTextField textField = new JTextField(20); // 20열 크기

String input = textField.getText(); // 입력값 읽기
textField.setText("Hello!");        // 텍스트 설정
```

---

## 🧪 종합 예제: 버튼 클릭 시 텍스트 필드 값 읽기

```
import javax.swing.*;

public class SwingMain {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Swing Components");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(400, 150);

        JTextField textField = new JTextField(20);
        JButton button = new JButton("Print Text");

        button.addActionListener(e -> {
            System.out.println("입력한 텍스트: " + textField.getText());
        });

        JPanel panel = new JPanel();
```

```
        panel.add(textField);
        panel.add(button);

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

---

## 📝 요약

| 컴포넌트 | 설명 | 주요 메서드 |
|---|---|---|
| `JFrame` | 윈도우 프레임 (창) | `setSize()`, `setVisible()`, `setDefaultCloseOperation()` |
| `JButton` | 클릭 가능한 버튼 | `addActionListener()`, `setText()` |
| `JTextField` | 단일 행 입력창 | `getText()`, `setText()` |

# 이벤트 처리 모델

## 1 이벤트 처리란?

> 사용자의 **입력**(버튼 클릭, 키보드 입력 등)에 반응하여 특정 **행동**을 수행하도록 만드는 메커니즘

Java의 이벤트 모델은 **델리게이션 이벤트 모델(Delegation Event Model)** 이라고 불린다.
이 모델은 **이벤트 발생 → 이벤트 객체 생성 → 이벤트 리스너에 전달**로 구성됨.

---

## 2 주요 구성 요소

| 구성 요소 | 설명 |
|---|---|
| **이벤트 소스** | 이벤트를 발생시키는 UI 컴포넌트 (예: `JButton`, `JTextField`) |
| **이벤트 객체** | 발생한 이벤트에 대한 정보가 담긴 객체 (예: `ActionEvent`, `MouseEvent`) |
| **이벤트 리스너** | 이벤트를 감지하고 처리하는 객체 (예: `ActionListener`, `MouseListener`) |
| **이벤트 핸들러** | 실제 이벤트가 발생했을 때 실행할 **로직(코드)** |

---

## 3 이벤트 처리 기본 구조

### 🔸 1. 리스너 인터페이스 구현

```
class MyListener implements ActionListener {
    @Override
    public void actionPerformed(ActionEvent e) {
        System.out.println("버튼 클릭됨!");
    }
```

```
}
```

### 🔸 2. 리스너 객체를 이벤트 소스에 등록

```
JButton button = new JButton("Click Me");
button.addActionListener(new MyListener());
```

---

## 4 자주 쓰는 이벤트 리스너 인터페이스

| 인터페이스 | 이벤트 대상 | 처리 메서드 |
|---|---|---|
| `ActionListener` | 버튼 클릭, 메뉴 선택 등 | `actionPerformed(ActionEvent)` |
| `MouseListener` | 마우스 클릭 등 | `mouseClicked`, `mouseEntered` 등 |
| `MouseMotionListener` | 마우스 이동 | `mouseMoved`, `mouseDragged` |
| `KeyListener` | 키보드 입력 | `keyPressed`, `keyReleased` |
| `FocusListener` | 포커스 변경 | `focusGained`, `focusLost` |
| `ItemListener` | 체크박스, 라디오 등 | `itemStateChanged` |

---

## 5 람다식으로 간단하게 처리 (Java 8+)

```
JButton button = new JButton("Click");
button.addActionListener(e -> {
    System.out.println("버튼 클릭됨!");
});
```

---

## 6 예제: 버튼 클릭 이벤트 처리

```
import javax.swing.*;
import java.awt.event.*;

public class EventExample {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Event Demo");
        JButton button = new JButton("Click Me");

        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                JOptionPane.showMessageDialog(frame, "버튼이 눌렸습니다!");
            }
        });

        frame.add(button);
```

```
        frame.setSize(300, 100);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

---

## 7️⃣ 이벤트 처리 흐름 요약도

```
[사용자 입력]
     ↓
[이벤트 소스 (버튼, 입력 필드 등)]
     ↓
[이벤트 객체 생성 (ActionEvent 등)]
     ↓
[리스너 인터페이스 구현체로 전달]
     ↓
[콜백 메서드 실행 → 사용자 정의 코드 수행]
```

---

## ✅ 정리

- Java의 이벤트 처리는 **델리게이션 모델 기반**
- 대부분의 이벤트는 특정 **리스너 인터페이스**를 통해 처리
- `addXXXListener()`를 통해 리스너를 등록
- Java 8 이후 **람다식**으로 더 간결하게 처리 가능

# 레이아웃 관리자

## 1️⃣ 레이아웃 관리자의 개념

Java에서는 컴포넌트를 **윈도우 창 내에서 일정한 규칙에 따라 배치**하는 기능이 필요하다.
이를 자동으로 처리해주는 클래스가 바로 **레이아웃 관리자**이다.

모든 **컨테이너(Container)** (예: `JFrame`, `JPanel`)는 자신만의 **LayoutManager**를 가지고 있고, 이를 통해 자식 컴포넌트들의 위치와 크기를 정해준다.

---

## 2️⃣ 주요 레이아웃 관리자 종류

| 레이아웃 관리자 | 설명 |
|---|---|
| `FlowLayout` | 왼→오 순서로 흐르듯 배치. 기본 정렬은 가운데. |
| `BorderLayout` | 상, 하, 좌, 우, 중앙 5개 영역으로 구분 |
| `GridLayout` | 격자 형태로 균등 분할 |
| `BoxLayout` | 수직 또는 수평 박스 정렬 |

| 레이아웃 관리자 | 설명 |
| --- | --- |
| `GridBagLayout` | 가장 유연하고 복잡한 배치 |
| `CardLayout` | 여러 화면 중 하나만 보여줌 (탭처럼) |
| `null` | 직접 위치 지정 (절대 좌표 배치) |

---

## 3️⃣ 각 레이아웃별 예시

### 📌 FlowLayout

```
JPanel panel = new JPanel(new FlowLayout());
panel.add(new JButton("버튼 1"));
panel.add(new JButton("버튼 2"));
```

- 컴포넌트들을 좌→우로, 줄 바꿈 하며 정렬
- 마치 웹페이지 줄 흐름처럼 동작

---

### 📌 BorderLayout

```
JFrame frame = new JFrame();
frame.setLayout(new BorderLayout());

frame.add(new JButton("북쪽"), BorderLayout.NORTH);
frame.add(new JButton("남쪽"), BorderLayout.SOUTH);
frame.add(new JButton("서쪽"), BorderLayout.WEST);
frame.add(new JButton("동쪽"), BorderLayout.EAST);
frame.add(new JButton("중앙"), BorderLayout.CENTER);
```

- 화면을 5개 영역(NORTH, SOUTH, EAST, WEST, CENTER)으로 나눠서 배치

---

### 📌 GridLayout

```
JPanel panel = new JPanel(new GridLayout(2, 3));
panel.add(new JButton("1"));
panel.add(new JButton("2"));
panel.add(new JButton("3"));
panel.add(new JButton("4"));
panel.add(new JButton("5"));
```

- 지정된 행(row)과 열(column)로 균등하게 컴포넌트를 배치

---

## 📌 BoxLayout

```
JPanel panel = new JPanel();
panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));
panel.add(new JButton("위"));
panel.add(new JButton("아래"));
```

- X_AXIS: 수평 정렬, Y_AXIS: 수직 정렬

---

## 📌 GridBagLayout (고급)

```
JPanel panel = new JPanel(new GridBagLayout());
GridBagConstraints gbc = new GridBagConstraints();

gbc.gridx = 0;
gbc.gridy = 0;
panel.add(new JButton("버튼 1"), gbc);

gbc.gridx = 1;
gbc.gridy = 0;
panel.add(new JButton("버튼 2"), gbc);
```

- **위치, 크기, 여백, 정렬** 등을 자유롭게 설정할 수 있는 가장 유연한 레이아웃

---

## 📌 CardLayout

```
CardLayout card = new CardLayout();
JPanel panel = new JPanel(card);

panel.add(new JLabel("첫 번째 카드"), "card1");
panel.add(new JLabel("두 번째 카드"), "card2");

card.show(panel, "card2"); // 두 번째 카드 표시
```

- 마치 카드처럼 컴포넌트를 교체해가며 보여줌

---

## 📌 null Layout (직접 좌표 설정)

```
JPanel panel = new JPanel();
panel.setLayout(null);

JButton button = new JButton("클릭");
button.setBounds(50, 100, 120, 30); // x, y, width, height
panel.add(button);
```

- 절대 위치 및 크기 지정 ( 💡 비추천, 반응형 불가)

---

## 4️⃣ GUI Builder가 사용하는 방식

IDE (IntelliJ, Eclipse 등)의 GUI Designer는 보통 내부적으로 `GroupLayout` 혹은 `GridBagLayout`을 사용한다.
하지만 실무에서는 주로 **BoxLayout**, **BorderLayout**, **GridLayout** 등이 많이 쓰인다.

---

## ✅ 정리

- 레이아웃 관리자는 **자동 배치**를 위한 필수 도구
- 상황에 따라 적절한 Layout을 조합하면 뛰어난 UI 구성 가능
- `null` Layout은 최대한 피하고, 표준 Layout 사용 권장