лаба 8

Компоненты пользовательского интерфейса. Иерархия компонентов.

В Java графический интерфейс пользователя (GUI) строится на основе компонентов. Основные библиотеки для создания GUI:

- **AWT (Abstract Window Toolkit)** – основа для Swing.
- **Swing** – более современная, гибкая и кросс-платформенная.
- **JavaFX** – новейшая технология (не входит в стандартную JDK с версии 11).

AWT

Базовый набор компонентов (`Button`, `Label`, `TextField` и др.). Библиотечные методы создают и используют графические компоненты операционной среды. С одной стороны, это хорошо, так как программа на Java похожа на остальные программы в рамках данной ОС.

- Ограниченный набор компонентов.
- Могут возникать проблемы с кроссплатформенной совместимостью (различия в размерах, шрифтах и поведении).
- Сложно создавать кастомные компоненты.

Swing

Библиотека **Java Swing** построена поверх **Java Abstract Widget Toolkit**

Для отрисовки используется 2D, что принесло с собой сразу несколько преимуществ. Набор стандартных компонентов значительно превосходит AWT по разнообразию и функциональности.

Стало легко создавать новые компоненты, наследуясь от существующих.

На базе свинга есть много расширений типа SwingX;

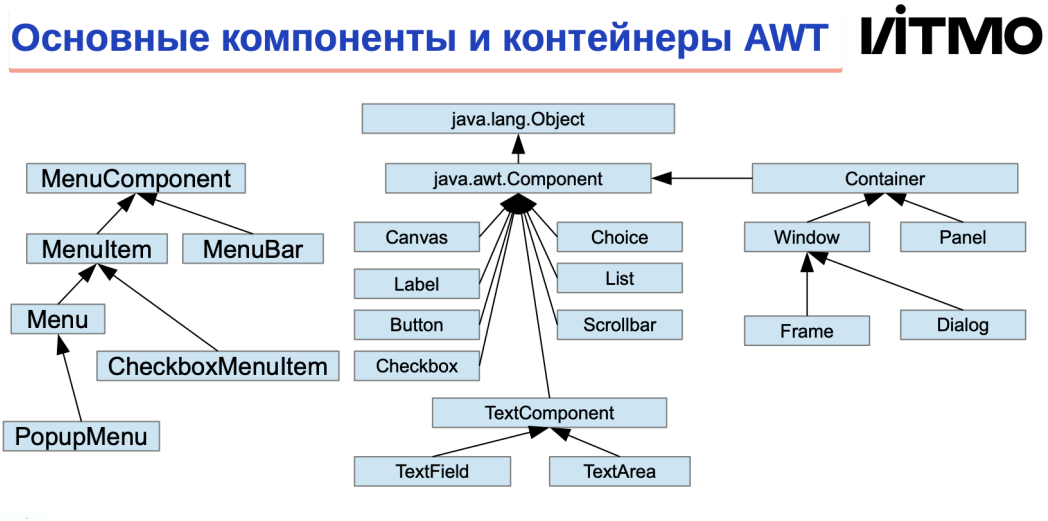
Поддержка различных стилей (Look and feel).

Java FX

Для отрисовки используется графический конвейер, что значительно ускоряет работу приложения. Набор встроенных компонентов обширен, есть даже отдельные компоненты для отрисовки графиков. Реализована поддержка

мультимедийного контента, множества эффектов отображения, анимации и даже мультитач. Внешний вид всех компонентов можно легко изменить с помощью CSS-стилей. В JavaFX входит набор утилит, которые позволяют сделать родной инсталлятор для самых популярных платформ: exe или msi для Windows, deb или rpm для Linux, dmg для Mac.

Иерархия компонентов



Базовые классы Component, Container, JComponent.

Компонент — отображаемый и взаимодействующий с пользователем элемент GUI

- java.awt.Component - абстрактный класс (элемент GUI)

Управляет:

- Цветом (передний план/фон)
- Размерами (высота/ширина)
- Положением (x/y координаты)
- Генерирует основные события

Контейнер — компонент, который содержит другие компоненты

- class Container extends Component
- Иерархия компонентов - дерево
- Компонент может находиться только в одном контейнере

Методы:

- add(Component) - добавляет компонент в контейнер
- setLayout(LayoutManager) - устанавливает менеджер компоновки, который определяет, **как компоненты располагаются внутри контейнера**
- validate() - применяет текущий layout, перерассчитывает размещение компонентов (для обновления интерфейса)

> Для создания графического интерфейса Java Swing нам нужен хотя бы один объект-контейнер.

JComponent

- Встроенная двойная буферизация при отрисовке (плавная отрисовка)

Основные компоненты-наследники

Простые элементы:

- `JButton` - кнопки
- `JLabel` - текстовые метки
- `JTextField` - поля ввода

Сложные виджеты:

- `JTable` - таблицы данных
- `JTree` - древовидные структуры
- `JComboBox` - выпадающие списки

Контейнеры:

- `JPanel` - универсальная панель
- `JScrollPane` - панель с прокруткой

Главный метод:

```
protected void paintComponent(Graphics g)
```

- Вызывается автоматически при необходимости перерисовки
- Всегда должен начинаться с:

```
super.paintComponent(g);// Очистка фона
```

```
repaint();// Запрос на перерисовку
```

Используется **Graphics**, это графический контекст компонента

Менеджеры компоновки.

Управляют расположением и размером компонентов внутри контейнера

1. `FlowLayout`
   - Компоненты располагаются в строку, переносятся при нехватке места. (слева на право по умолчанию)
   - По умолчанию для `JPanel`.
2. `BorderLayout`
   - Компоненты располагаются в 5 областях: `NORTH`, `SOUTH`, `EAST`, `WEST`, `CENTER`.
   - По умолчанию для `JFrame`.
3. `GridLayout`
   - Компоненты размещаются в таблице с заданным числом строк и столбцов.
   - Контейнер делится на одинаковые ячейки по строкам и столбцам. Все компоненты будут одного размера
4. `GridBagLayout`

- Контейнер делится на ячейки по строкам и столбцам

Задаются ограничения

- объединение ячеек
- заполнение ячеек
- привязка к краю ячеек
- распределение пространства

5. `BoxLayout`

- Компоненты располагаются в один ряд вертикально или горизонтально

6. `CardLayout`
   - Аналог колоды карт (виден только верхний компонент)
   - Позволяет выбрать одну из панелей
   - Базовый аналог вкладок
   - Переключение между картами нужно реализовывать отдельно

Модель обработки событий. Класс-слушатель и класс-событие.

В Java используется **делегированная модель событий**:

- **Источник события** (компонент, например, `JButton`).
- **Слушатель (Listener)** – объект, который обрабатывает событие.
- **Событие (Event)** – объект, содержащий информацию о событии (например, `ActionEvent`).

```
class A implements ActionListener {
    Button b = new Button("OK");
    Label l = new Label("Button pressed");
    l.setVisible(false);
    b.addActionListener(this);  - подписка на событие
......
public void actionPerformed(ActionEvent e) {
    l.setVisible("true");   - реакция на событие
}
```

- Анонимным классом

```
b.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    l.setVisible("true");
  }
});
```

- Лямбда-выражением

```
b.addActionListener((e) -> l.setVisible("true"));
```

Слушатели (Listeners) - интерфейсы с методами-обработчиками:

- **ActionListener** - слушатель действия (actionPerformed)

- **MouseListener** - для нажатия/отжатия кнопок и входа/ухода курсора мыши с области компонента (**mouseClicked()**, **mouseEntered()**)
- **MouseMotionListener** - для движения курсора мыши или перетаскивании мышью (**mouseMoved()**, **mouseDragged()**)
- **MouseWheelListener()** - для прокрутки колеса мыши.
- **KeyListener** - клавиатура (**keyPressed() когда нажимается клавиша**, **keyTyped() когда вводится символ**)
- **WindowListener** - события окна (**windowClosing()**, **windowOpened()**)

События (Events) - объекты с информацией о действии:

- **ActionEvent** - событие, определяемое компонентном
- **MouseEvent** - событие мыши
- **KeyEvent** - событие ввода с клавиатуры
- **WindowEvent** - события окна, как активация и свертывание.

Адаптеры (Adapters) - упрощают обработку (не нужно реализовывать все методы)

```
addMouseListener(new MouseAdapter() {
    @Override
    public void mouseClicked(MouseEvent e) {
        System.out.println("Клик в " + e.getPoint());
    }
});
```

Технология JavaFX. Особенности архитектуры, отличия от AWT / Swing.

- Поддержка стилей CSS
- Поддержка 2D- и 3D-графики
- Легковесные компоненты
- Интеграция с библиотекой Swing

Для отрисовки используется графический конвейер, что значительно ускоряет работу приложения. Набор встроенных компонентов обширен, есть даже отдельные компоненты для отрисовки графиков. Реализована поддержка мультимедийного контента, множества эффектов отображения, анимации и даже мультитач. Внешний вид всех компонентов можно легко изменить с помощью CSS-стилей. И самое прекрасное — в JavaFX входит набор утилит, которые позволяют сделать родной инсталлятор для самых популярных платформ: exe или msi для Windows, deb или rpm для Linux, dmg для Mac.

javafx.application.Application — класс-предок всех приложений JavaFX

- void init() — инициализация приложения (стартовый поток)
- abstract void start(Stage s) — основной поток приложения
- void stop() — освобождение ресурсов
- public static void launch(String args) — запуск приложения

javafx.stage.Stage — основная платформа

- Контейнер верхнего уровня (аналог JFrame)
- Предоставляется системой при запуске приложения
- Обеспечивает связь с графической подсистемой ОС
- setTitle(String)
- setScene(Scene)
- show()

javafc.scene.Scene — контейнер для элементов сцены

- Должен быть хотя бы один объект класса Scene
- Элементы сцены — узлы (Node)
- Узлы образуют граф (scene graph)

Граф включает не только контейнеры и компоненты, но также графические примитивы (текст и графические примитивы)

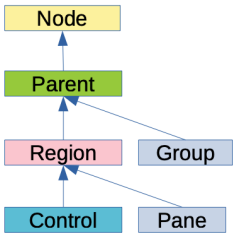
- Узел с дочерними узлами — Parent (extends Node)
- Корневой узел (root node) — узел без родительского узла

- Scene sc = new Scene(root node,300,150);

Основные типы узлов JavaFX

ИТМО

- Node - узел
- Parent - содержит другие узлы
- Region - имеет стиль
- Group - общие эффекты
- Control - управляемый пользователем
- Pane - панель с компоновкой



Интернационализация. Локализация. Хранение локализованных ресурсов.

Интернационализация — проектирование программы так, чтобы ее можно было локализовать без конструктивных изменений

Текстовые сообщения не хардкодятся, а динамически подгружаются

Локализация – адаптация приложения под конкретный язык/регион

Локаль — совокупность характеристик, определяющих географический, политический или культурный регион

- Класс java.util.Locale

Примеры: it, en_UK, ru_RU.CP1251

Элементы локали:

- язык — 2 строчные буквы (иногда 3) (ru)
- страна (регион) — 2 заглавные буквы (RU) или 3 цифры
- вариант (например, кодировка для русской локали)
- письменность — 4 буквы, первая заглавная (Cyrl)
- расширение

Наборы ресурсов

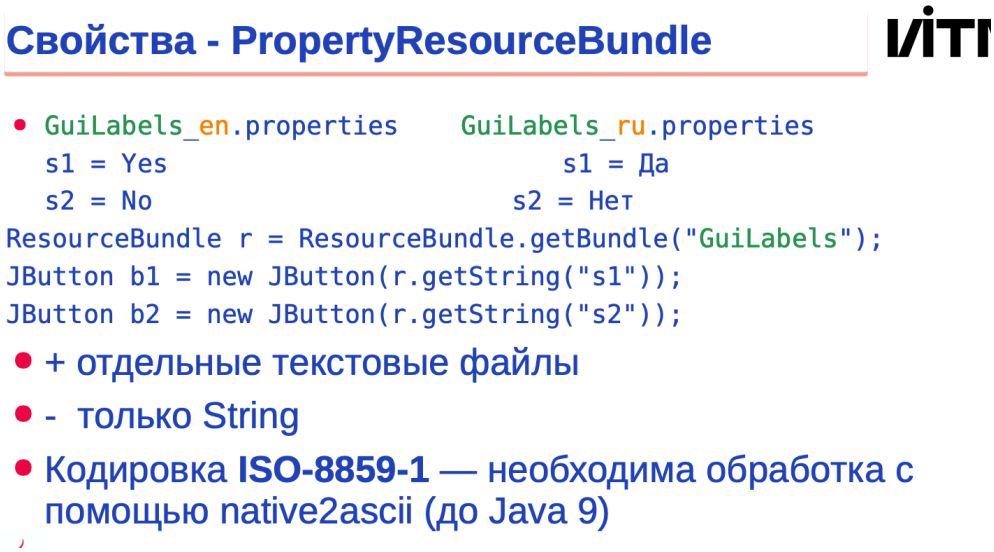
Класс java.util.ResourceBundle

- Формат "ключ-значение"

ключ - условная строка, значение - перевод

2 варианта:

- Файл свойств *.properies (PropertyResourceBundle)
- Класс со списком (ListResourceBundle)



Получение нужного ресурса

- getString(key)
- getStringArray(key)
- getObject(key)
- Просматривается список кандидатов. Возвращается первая найденная по ключу строка
- MissingResourceException

Форматирование локализованных числовых данных, текста, даты и времени.
Классы NumberFormat, DateFormat, MessageFormat, ChoiceFormat.

NumberFormat (форматирование чисел)

Форматирует числа в соответствии с локалью (разделители, валюта и др.).

Методы:

- `getInstance()` - для обычных чисел
- `getCurrencyInstance()` - для валют
- `getPercentInstance()` - для процентов

DateFormat (форматирование даты и времени)

Преобразует дату в строку с учетом локали.

Стили:

- `SHORT` (27.05.25)
- `MEDIUM` (27 мая 2025 г.)
- `LONG` (27 мая 2025 года)
- `FULL` (вторник, 27 мая 2025 г.)

MessageFormat (подстановка значений в строки)

Динамическая вставка данных в шаблон.

Плейсхолдеры: `{0}`, `{1}`, ...

```
String pattern = "Hello, {0}! Today is {1,date,long}.";
MessageFormat mf = new MessageFormat(pattern, Locale.US);
Object[] args = {"John", new Date()};
System.out.println(mf.format(args));  // "Hello, John! Today is May 27, 2025."
```

ChoiceFormat (выбор варианта по числу)

Сопоставляет числовые диапазоны с текстом.

Синтаксис:

`limits` - границы диапазонов, `formats` - соответствующие строки.

```
double[] limits = {0, 1, 2};
String[] formats = {"нет яблок", "одно яблоко", "{0} яблока"};
ChoiceFormat cf = new ChoiceFormat(limits, formats);
```

```
System.out.println(cf.format(0));  // "нет яблок"
System.out.println(cf.format(1.5));  // "1.5 яблока" (используется последний подход
```

Форматирование числовых данных

ИТМО

- Класс java.text.NumberFormat — абстрактный
  - NumberFormat nf = NumberFormat.getNumberInstance()
  - NumberFormat cf = NumberFormat.getCurrencyInstance()
  - NumberFormat pf = NumberFormat.getPercentInstance()
  - nf.format(new Float(999.8));
- Класс java.text.DecimalFormat
  - df = (DecimalFormat) nf;
  - df.applyPattern("##,#0.00");
  - df.format(new Float(888.7));
- Класс DecimalFormatSymbols
  - ds.setDecimalSeparator('=');
  - df.setDecimalFormatSymbols(ds);
  - df.format(new Float(777.6));

Форматирование даты и времени

ИТМО

- Класс java.text.DateFormat — абстрактный
  - DateFormat df = DateFormat.getDateInstance(DateFormat.FULL)
  - DateFormat tf = DateFormat.getTimeInstance(DateFormat.LONG)
  - DateFormat dtf = DateFormat.getDateTimeInstance(DateFormat.SHORT)
  - df.format(new Date());
- Класс java.text.SimpleDateFormat
  - sdf = (SimpleDateFormat) df;
  - sdf.applyPattern("yyyy-MM-dd");
  - sdf.format(new Date());
- Класс DateFormatSymbols
  - ds.setShortWeekdays("пнд","втр","срд","чтв","птн","сбт","вск");
  - sdf.setDateFormatSymbols(ds);
  - sdf.format(new Date());

Сообщения - MessageFormat

ИТМО

- 8 апреля 2024 в 18:17 произошло полное солнечное затмение.
- Total solar eclipse happened at 18:17PM on April 8, 2024.

Eclipse_en.properties

```
msg = {0} solar eclipse happened at
{1,time,short} on {1,date,short}.

full = Total
part = Partial
ring = Annular
```

Eclipse_ru.properties

```
msg = {1,date,short} в {1,time,short}
произошло {0} солнечное затмение.

full = полное
part = частное
ring = кольцеобразное
```

```
ResourceBundle r = ResourceBundle.getBundle("Eclipse");
MessageFormat mf = new MessageFormat(r.getString("msg");
Date date = Date.from(Instant.parse("2024-04-08T18:17"));
Object[] args = {r.getString("full", date.getTime()};
mf.format(args);
```

Формат с выбором

ИТМО

- Класс ChoiceFormat extends NumberFormat
  - 0 friends like it
  - 1 friend likes it
  - 1000 friends like it

Like_en.properties

```
msg = {0} it
one = {0,number} friend likes
many = {0,number} friends like
```

```
ResourceBundle r = ResourceBundle.getBundle("Like");
MessageFormat mf = new MessageFormat(r.getString("msg");
double[] lims = { 0, 1, 2 };
String one = r.getString("one");
String many = r.getString("many");
String[] msgs = { many, one, many };
ChoiceFormat cf = new ChoiceFormat(lims, msgs);
mf.setFormatByArgumentIndex(0, cf);
Object[] args = { new Integer(15) };
mf.format(args);
```

```
15 friends like it
```

167