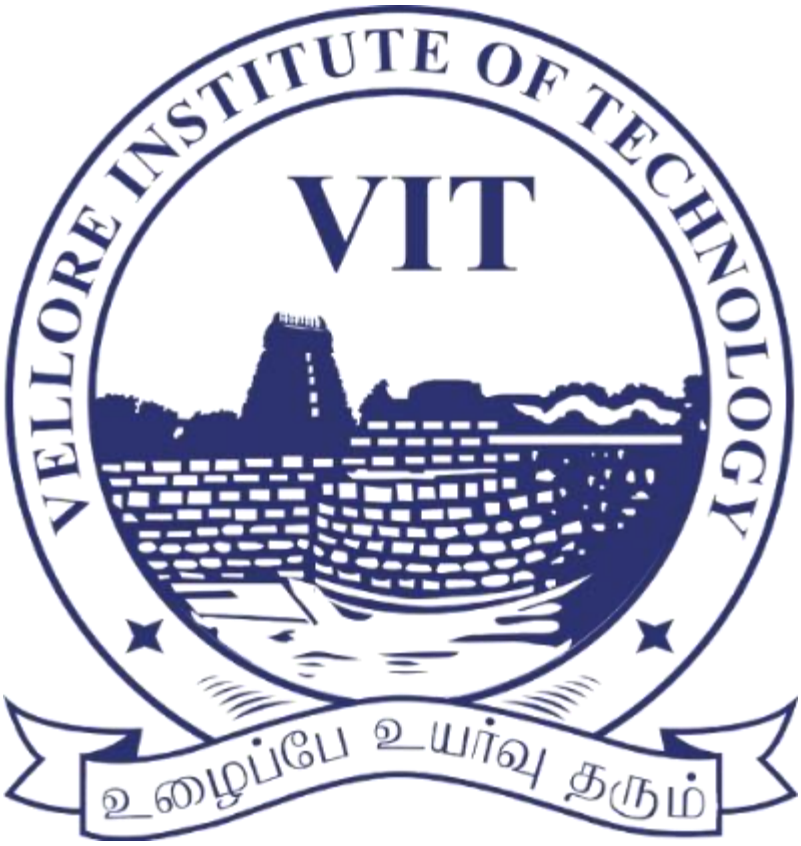

Documentation: Live Code Generator Application

Name	Harshit Mishra
Reg no.	23BAI10473
To	Anand Motwani Sir

Overview

The **Live Code Generator** is a Swing-based Java application designed to create class templates interactively. Users can input class details, attributes, methods, and additional configurations to generate and preview Java code dynamically. The generated code can also be saved to a file.

Features

1. **Dynamic Code Preview**: See real-time updates of the generated Java code as you input details.
2. **Customizable Options**:
    a. Class name
    b. Access modifier
    c. Superclass and interfaces
    d. Import statements
    e. Attributes and methods
    f. Constructor generation
    g. Getters and setters
3. **Save Functionality**: Save the generated code to a `.java` file.
4. **Clear Fields**: Reset all input fields to start fresh.

Key Concepts

Swing Framework:

Used for creating the graphical user interface (GUI), including **`JFrame`, `JPanel`, `JTextField`, `JTextArea`, `JButton`**, and other components.

Event Handling:

- **`ActionListener`** is used for button actions (e.g., Save and Clear).
- **`DocumentListener`** is used for detecting changes in text fields and areas for live preview updates.

Object-Oriented Programming (OOP):

- **Encapsulation**: Private fields (e.g., `classNameField`) are used to encapsulate data.
- **Inheritance**: The program supports adding a superclass and interfaces dynamically.
- **Polymorphism**: Overriding methods of `ActionListener` and `DocumentListener` interfaces.

I/O (Input/Output):

`FileWriter` and **`JFileChooser`** are used for file handling to save generated code.

Layout Management:

Various layout managers like **`BorderLayout`**, **`GridLayout`**, and **`FlowLayout`** are used to arrange components.

Collections:

`StringBuilder` is used for efficiently building the generated code.

Error Handling:

`try-catch` blocks handle **`IOException`** when saving files.

Multithreading:

`SwingUtilities.invokeLater` ensures the GUI runs on the Event Dispatch Thread (EDT) for thread safety.

Dynamic Code Generation:

Combines user input and templates to generate Java code dynamically.

How to Start

Using Command Line

1. Open a terminal/command prompt go to project folder and navigate to the `src` folder:

```
cd src
```

2. Compile the code:

```
javac app.java
```

   This generates a `app.class` file.

3. Run the program:

```
java app
```

Components

Main Frame

- **Title**: `Live Code Generator`
- **Layout**: `BorderLayout`
- **Size**: `1000 x 700`
- **Split Pane**: Divides the input panel and the code preview panel.

Input Panel

1. **Class Name Field**: Input the name of the class.
2. **Access Modifier Dropdown**: Select from `public`, `private`, `protected`, or `default`.

3. **Superclass and Interfaces Fields**: Specify a superclass and implemented interfaces.
4. **Import Statements Area**: Add import statements (one per line).
5. **Attributes Input**: Add class attributes in `type name` format (e.g., `String name`).
6. **Methods Input**: Add method declarations in `returnType name` format (e.g., `void display`).
7. **Options**:
    a. `Generate Constructor`: Automatically generates a constructor using the provided attributes.
    b. `Generate Getters/Setters`: Adds getter and setter methods for attributes.

Preview Panel

- Displays the dynamically generated Java code in a non-editable text area.

Buttons

1. **Save Code**: Save the generated code to a `.java` file.
2. **Clear Fields**: Reset all inputs after confirmation.

Key Methods

`addLivePreviewListeners()`

- Attaches `DocumentListener` and `ActionListener` to relevant input fields to dynamically update the code preview.

`updatePreview()`

- Generates and updates the Java code in the preview area based on current user inputs.

`saveCode(ActionEvent e)`

- Prompts the user to save the generated code as a `.java` file using a `JFileChooser`.

`clearFields(ActionEvent e)`

- Resets all fields and options to their default states after confirmation.

Code Structure

Imports

```
import java.awt.*;
import java.awt.event.ActionEvent;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import javax.swing.*;
import javax.swing.event.DocumentEvent;
import javax.swing.event.DocumentListener;
```

Main Class

```
public class app {
    // Frame and Component Declarations
    final private JFrame frame;
    final private JTextField classNameField;
    final private JTextArea attributesArea, methodsArea, previewArea,
importsArea;
    final private JCheckBox generateConstructorCheckbox,
generateGettersSettersCheckbox;
    final private JTextField superClassField, interfacesField;
    final private JComboBox<String> accessModifierDropdown;

    public static void main(String[] args) {
        SwingUtilities.invokeLater(app::new);
    }

    public app() {
        // Initialize Components and Layout
    }
}
```

Methods Breakdown

updatePreview()

- Builds the Java class template:
  - Adds import statements.
  - Creates the class declaration.
  - Includes attributes.
  - Optionally generates:
    - Constructor
    - Getters and setters
  - Adds user-defined methods.

saveCode(ActionEvent e)

- Saves the previewed code to a user-specified file.
- Uses `FileWriter` for file handling.

clearFields(ActionEvent e)

- Clears all input fields, checkboxes, and resets dropdowns.

Usage Instructions

1. Run the application.
2. Fill out the fields for the class name, attributes, methods, and other options.
3. View the generated code live in the preview area.
4. Use the **Save Code** button to export the file.
5. Use the **Clear Fields** button to reset all fields.

Example Inputs and Outputs

Inputs

- **Class Name**: `Person`

- **Attributes**:

  ```
  String name
  int age
  ```

- **Methods**:

  ```
  void display
  ```

- **Options**:
  - Enable `Generate Constructor`
  - Enable `Generate Getters/Setters`

Output

```
public class Person {

    private String name;
    private int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public void display() {
        // TODO: Implement this method
```

```
    }
}
```

Error Handling

- Alerts the user if:
    - No code is available to save.
    - File saving fails due to an `IOException`.