Absolutely! Let's go through your Anubius Java chatbot project **line-by-line** and understand **everything**, especially the OpenRouter API integration. I'll break it into logical sections for clarity and explain what each part does.

🔧 Imports

```
import javax.swing.*;

import java.awt.*;

import java.io.*;

import java.net.*;

import java.util.ArrayList;

import com.google.gson.*;

import com.sun.speech.freetts.*;
```

- javax.swing.* – For building the GUI (windows, buttons, fields, etc.).
- java.awt.* – For advanced GUI layout, fonts, colors.
- java.io.* – For input/output (reading/writing from the API).
- java.net.* – For HTTP connections to the OpenRouter API.
- java.util.ArrayList – For storing chat messages.
- com.google.gson.* – Google's JSON library to build and parse JSON data.
- com.sun.speech.freetts.* – FreeTTS (Text-to-Speech) voice library.

🎤 Class and Voice Initialization

```
public class Anubius extends JFrame {

   ...

   private final Voice voice;
```

- Anubius extends JFrame, meaning it's a Swing-based window application.
- Voice voice is the object used to speak text aloud using FreeTTS.

🧱 Constructor (UI Setup)

```
public Anubius() {

   setTitle("AnubisGPT");

   setSize(1000, 700);

   setDefaultCloseOperation(EXIT_ON_CLOSE);

   ...

}
```

- Sets the title, size, and exit behavior for the app window.

🗣️ FreeTTS Setup

```
System.setProperty("freetts.voices", "...");

VoiceManager vm = VoiceManager.getInstance();

voice = vm.getVoice("kevin16");

if (voice != null) voice.allocate();
```

- Initializes the FreeTTS engine and selects the "Kevin" voice for speaking.
- `.allocate()` prepares the voice for use.

🔝 Top Bar (Logo & App Name)

```
JPanel topBar = new JPanel(...);

...

JLabel logoLabel = new JLabel(new ImageIcon("logo.png"));
```

- Creates a horizontal top bar with the app logo and name.

💻 Chat Area (Main display)

```
chatArea = new JTextArea();

chatArea.setEditable(false);

...

JScrollPane chatScroll = new JScrollPane(chatArea);
```

- Displays the ongoing chat conversation.
- `JScrollPane` adds scrolling support.

💬 Input Field & Buttons

```
inputField = new JTextField();

voiceToggleButton = new JButton("Enable Voice");

sendButton = new JButton("Send");
```

- Text field for typing user input.
- `sendButton` sends the message.
- `voiceToggleButton` enables/disables text-to-speech.

📋 Adding Components to Frame

```
add(topBar, BorderLayout.NORTH);
add(mainPanel, BorderLayout.CENTER);
```

- Adds the top and main panels to the window in appropriate positions.

🧠 Initial Message

```
showGreetingMessage();
```

- Shows a welcome message and speaks it aloud using `speak()`.

🗣️ Toggle Voice

```
private void toggleVoice() { ... }
```

- When the voice button is clicked, toggles speech on/off and updates the button appearance.

📢 Speaking Text

```
private void speak(String text) {

  if (voiceEnabled && voice != null) {

    new Thread(() -> voice.speak(text)).start();

  }

}
```

- Speaks the provided text using FreeTTS on a separate thread.

📝 Sending a Message

```
private void sendMessage() {

  ...

  new Thread(() -> {

    ...

    String aiResponse = getAIResponse(userMessage);

    ...

  }).start();

}
```

- Gets the user input, adds it to the chat area, then fetches the AI response in a background thread.
- It avoids freezing the UI while waiting for the network response.

🌐 Calling the OpenRouter API

private String getAIResponse(String prompt) throws IOException {

URL url = new URL("https://openrouter.ai/api/v1/chat/completions");

HttpURLConnection conn = (HttpURLConnection) url.openConnection();

- Opens a POST connection to the OpenRouter endpoint used for chatting.

Headers and Body

conn.setRequestMethod("POST");

conn.setRequestProperty("Content-Type", "application/json");

conn.setRequestProperty("Authorization", "Bearer " + OPENROUTER_API_KEY);

- Sets HTTP headers including:
  - JSON content type.
  - Your **API Key** for authentication (Bearer token).
- 🧾 **JSON Body Construction**
- JsonObject userMsg = new JsonObject();
- userMsg.addProperty("role", "user");
- userMsg.addProperty("content", prompt);
-
- JsonArray messages = new JsonArray();
- messages.add(userMsg);
-
- JsonObject body = new JsonObject();
- body.addProperty("model", "deepseek/deepseek-r1-distill-llama-8b");
- body.add("messages", messages);
  - Constructs the JSON request body for the chat:
- {
-   "model": "deepseek/deepseek-r1-distill-llama-8b",
-   "messages": [
-     {"role": "user", "content": "Your prompt here"}
-   ]
- }
- 🚀 **Sending Request**
- OutputStream os = conn.getOutputStream();
- os.write(body.toString().getBytes());
- os.flush();
  - Sends the constructed JSON body to the API server.
- 📥 **Reading the Response**
- BufferedReader in = new BufferedReader(new InputStreamReader(conn.getInputStream()));
- StringBuilder response = new StringBuilder();
- String line;
- while ((line = in.readLine()) != null) {
-     response.append(line);
- }
- in.close();
  - Reads the full response from the OpenRouter server.

- 🧪 **Parsing the Response**
- JsonObject json = JsonParser.parseString(response.toString()).getAsJsonObject();
- JsonArray choices = json.getAsJsonArray("choices");
  - Converts the response string into a JSON object.
  - Extracts the "choices" array which contains the AI's reply.
- ✂ **Cleaning & Returning the Text**
- String aiResponse = choices.get(0).getAsJsonObject()
-   .getAsJsonObject("message")
-   .get("content").getAsString();
- return aiResponse.replaceAll("[*#_']", "");
  - Extracts the response content and removes unwanted characters like * or #.
- 🧪 **main() Method**
- public static void main(String[] args) {
-   SwingUtilities.invokeLater(() -> {
-     Anubius app = new Anubius();
-     app.setVisible(true);
-   });
- }
  - Launches the GUI safely using SwingUtilities (recommended for UI apps).
- 🧠 **API Understanding Summary**
  - You are using **OpenRouter API**, which is like an API gateway for many AI models (like GPT, LLaMA, Claude).
  - The /chat/completions endpoint works similarly to OpenAI's API — you send a **list of messages**, and it replies as the assistant.
  - You must provide:
    - Authorization (Bearer key)
    - model name (like deepseek/deepseek-r1-distill-llama-8b)
    - A message list (role: user, content: prompt)
      - It returns a JSON with a "choices" array → each has a message.content → the AI's reply.
    - If your teacher asks about:
      - **JsonObject, JsonArray** – These are used to build and read JSON in Java.
      - **Voice** – Comes from FreeTTS, used to convert text to spoken audio.
      - **Threads** – Used so the app doesn't freeze while waiting for replies.
    - Would you like a **printable document** of this explanation to carry with you? I can generate that too.