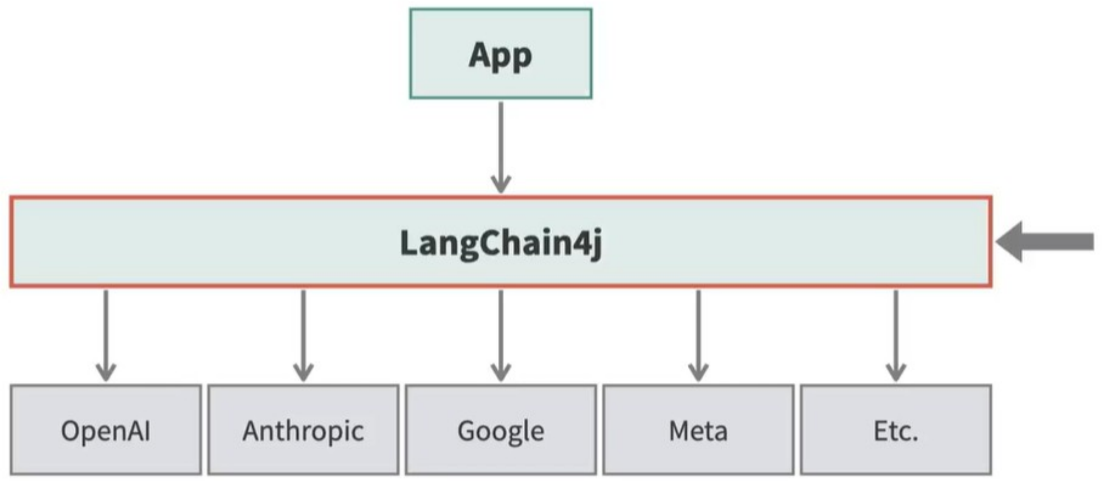
LangChain4J intro:
It’s straight-forward to write an application to communicate with one LLM, while it is quite challenging to write a Java application that uses GenAI features and that works across all the popular LLMs like OpenAI, Gemini etc, and all available tooling. An abstraction layer that provides a consistent interface to the major LLMs.



Langchain4J is a Java wrapper and abstraction that works with 20 different popular LLMs. It simplifies integrating LLMs into a Java application.

Langchain4J has Core components like Models, Prompts, Tools, Memory and Parsers, and RAG components like Document Loaders, Embeddings, Chunkers. RAG components are very helpful in building RAG systems. Higher level services of Langchain4J makes it easier to write Production code.

Basic Langchain4J abstractions:
1. ChatModel: represents an LLM
2. ChatMessage: message to/from the ChatLanguageModel
3. ChatResponse: response from the LLM
4. ChatMemory: conversation context (needed for chatbots)
ChatMemory is necessary when we are trying to create a chatbot.

There is a separate chatmodel for every LLMs, Langchain4J supports.
Class OpenAIChatModel => User message (based on our requirements or what the user is asking), System messages (instruction for LLMs on behavior). Language model sends back a Langchain4J response, inside that response, there is an embedded object called AIMessage. AImessage object contains the actual response from the language model. Sometimes, AIMessage is called as an “Assistant message” in some vendor documentation. AIMessage object has method called as text(), which returns the pure text of the response.