==========

EXPLICAÇÃO DOS PRINCIPAIS MÉTODOS E BIBLIOTECAS DO SCRIPT EM JAVA

==========

VISÃO GERAL DO PROJETO

O projeto tem como objetivo gerar automaticamente um cardápio semanal
de almoço e jantar utilizando uma API de Inteligência Artificial,
no caso o modelo Gemini.

O programa recebe como entrada uma lista de ingredientes informados
pelo usuário, constrói um prompt descritivo, envia à API e processa
a resposta em formato JSON, transformando-a em um cardápio estruturado.

O sistema pode ser dividido em três partes principais:
1. Montagem e envio do prompt (requisição para a IA)
2. Validação e tratamento da resposta
3. Retorno dos dados formatados em JSON para o front-end

A seguir, são descritos os principais métodos e as bibliotecas utilizadas.

BIBLIOTECAS UTILIZADAS

---

1. dotenv (em Java: dotenv-java)
   - Usada para carregar variáveis de ambiente a partir de um arquivo ".env".
   - É por meio dela que o script obtém a chave da API (GEMINI_API_KEY)
     sem precisar deixá-la exposta no código-fonte.
   - Exemplo de uso:

```
Dotenv dotenv = Dotenv.load();
String apiKey = dotenv.get("GEMINI_API_KEY");
```

2. google-genai
   - Biblioteca oficial para integração com o modelo Gemini da Google.
   - Permite criar o cliente de comunicação (Client) e enviar prompts
     para geração de texto, imagens ou respostas estruturadas.
   - Exemplo conceitual:

```
GenAiClient client = new GenAiClient(apiKey);
String resposta = client.generateContent(model, prompt);
```

3. jsonschema
   - Utilizada para validar se o conteúdo JSON recebido da IA segue
     a estrutura esperada (campos obrigatórios, tipos de dados, etc.).
   - Garante que cada receita e o cardápio completo estejam no formato correto.
   - É importante para evitar erros de parsing e dados incompletos.

4. flask / flask-cors (no caso de uso web)
   - São bibliotecas Python equivalentes a um framework web leve,
     mas aqui representariam, em Java, frameworks como Spring Boot ou Spark.
   - Servem para criar endpoints REST, aceitar requisições HTTP e
     permitir que o front-end acesse a API com segurança (CORS habilitado).

5. gson / org.json (para parsing JSON em Java)
   - Responsáveis por converter texto em objetos e vice-versa.
   - O método parseJsonSeguro() depende dessas funções para transformar
     o texto retornado pela IA em um objeto manipulável no código.

Essas bibliotecas em conjunto tornam o projeto modular, seguro e
facilitam tanto o desenvolvimento local quanto a integração com sistemas externos.

PRINCIPAIS MÉTODOS DO SCRIPT

---

1. Método montarPromptPorIngredientes()

Responsável por gerar a mensagem (prompt) enviada à Inteligência Artificial.

Parâmetros:

- List<String> ingredientes → lista dos ingredientes disponíveis;
- boolean permitirExtras → define se pode usar itens básicos adicionais.

O método monta uma string com instruções claras para o modelo Gemini:
solicita um cardápio de segunda a domingo, com duas refeições por dia
(almoço e jantar), descrevendo o formato de saída e as restrições.

Esse método é essencial porque define a qualidade da resposta da IA:
um prompt bem estruturado gera resultados mais coerentes e fáceis de validar.

---

2. Método gerarComRetry()

Executa a comunicação com a API Gemini e implementa uma estratégia
de repetição automática em caso de falhas (retry com backoff exponencial).

Ele tenta até três vezes enviar o mesmo prompt.
Caso ocorra erro de rede ou tempo limite, o método espera alguns segundos
e tenta novamente, dobrando o tempo de espera a cada tentativa.

Isso torna o sistema mais confiável e evita que falhas temporárias
comprometam o fluxo completo do programa.

---

3. Método parseJsonSeguro()

Após a API retornar o texto gerado, este método tenta converter
a resposta em um objeto JSON.

Etapas principais:

- Remove blocos de código extras como ```json.
- Tenta converter o texto usando um parser JSON.
- Caso falhe, exibe mensagem de erro para depuração.

Esse método impede que o sistema quebre caso o modelo Gemini
retorne uma resposta mal formatada.

---

4. Método validarReceitaObj()

Verifica se cada receita segue o formato obrigatório:

```
{
  "nome": "Frango assado",
  "ingredientes": ["frango", "batata", "sal"],
  "modo_preparo": "Asse o frango com as batatas."
}
```

Utiliza a biblioteca jsonschema para validar os campos e os tipos de dados.
Caso algum campo esteja ausente ou inválido, a receita é descartada.

---

5. Método validarCardapioSemana()

Analisa o cardápio completo, garantindo que:

- Todos os dias da semana estejam presentes;
- Cada dia tenha duas refeições (almoço e jantar);
- Cada refeição esteja formatada corretamente.

Internamente, o método chama validarReceitaObj() para verificar
as receitas individualmente.

---

6. Método limparJson()

Tem a função de sanitizar o texto retornado pela IA,
removendo caracteres desnecessários, quebras de linha e marcações
como ```json e ``` no início e fim das respostas.

Ele é executado antes do parsing, garantindo que o texto esteja limpo
e pronto para ser convertido em JSON.

---

7. Endpoint gerarCardapio()

No contexto web, esse método representa o endpoint principal da API.

Fluxo resumido:

- Recebe via POST um JSON com os ingredientes e a opção de permitir extras;
- Monta o prompt usando montarPromptPorIngredientes();
- Chama gerarComRetry() para enviar à IA;
- Converte e valida a resposta com parseJsonSeguro() e validarCardapioSemana();
- Retorna o cardápio final em formato JSON para o front-end.

Esse endpoint é o ponto de integração entre o usuário e o modelo de IA,
concentrando toda a lógica do sistema.

CONCLUSÃO

O script combina diversas boas práticas de desenvolvimento:
- Separação de responsabilidades entre métodos.
- Validação e tratamento de erros robustos.
- Uso de bibliotecas específicas para cada etapa do processo.
- Comunicação eficiente entre back-end e modelo de IA.

Cada método cumpre um papel importante dentro da arquitetura:
- montarPromptPorIngredientes() constrói a lógica de entrada;
- gerarComRetry() garante a confiabilidade;
- parseJsonSeguro() e validarCardapioSemana() asseguram a integridade;
- gerarCardapio() serve de interface entre o usuário e o modelo.

O resultado é uma aplicação completa, modular e funcional,
que transforma dados simples (ingredientes) em um produto útil
e automatizado (cardápio semanal completo).

==========