```java
package org.example;

import com.fasterxml.jackson.databind.DeserializationFeature;
import com.fasterxml.jackson.databind.ObjectMapper;
import org.apache.http.Header;
import org.apache.http.HttpEntity;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.util.EntityUtils;
import org.apache.log4j.*;

import java.io.FileInputStream;
import java.io.File;
import java.io.IOException;
import java.io.InputStream;
import java.nio.charset.StandardCharsets;
import java.text.SimpleDateFormat;
import java.util.*;
import java.util.concurrent.*;

/**
 * APIRestComponentClient para trabajar con POJOs (por ejemplo DetalleRequest).
 */
public class APIRestComponentClient {

    private static final Logger logger;
    private static final Properties config;

    static {
        // 1. Configurar logger (Log4j con DailyRollingFileAppender)
        logger = Logger.getLogger(APIRestComponentClient.class);
        setupLogger();

        // 2. Cargar config.properties
        config = new Properties();
        try (InputStream in = new FileInputStream("src/main/resources/config.properties"
        )) {
            config.load(in);
        } catch (IOException e) {
            logger.error("No se pudo cargar config.properties: " + e.getMessage(), e);
        }
    }

    private static void setupLogger() {
        try {
            String logDir = "./logs/";
            String logFileName = "api-rest-" + new SimpleDateFormat("yyyy-MM-dd").format(
            new Date()) + ".log";

            File dir = new File(logDir);
            if (!dir.exists()) dir.mkdirs();

            DailyRollingFileAppender appender = new DailyRollingFileAppender(
                new PatternLayout("%d{ISO8601} [%t] %-5p %c - %m%n"),
                logDir + logFileName,
                "'.'yyyy-MM-dd"
            );
            logger.setLevel(Level.DEBUG);
            logger.addAppender(appender);

        } catch (IOException e) {
            System.err.println("Error al configurar logger: " + e.getMessage());
        }
    }

    /**
```

```java
     * Método original inalterado: recibe un POJO o cualquier objeto serializable por Jackson.
     */
    public GBDRespIngresarACHRecibidosRest invokeRestServices(String url, Object request, boolean useSSL) throws IOException {
        CloseableHttpClient httpClient = null;
        try {
            if (useSSL) {
                httpClient = HttpClients.createDefault(); // placeholder para cliente SSL
                logger.debug("Usando conexión SSL/TLS");
            } else {
                httpClient = HttpClients.createDefault();
                logger.debug("Usando conexión HTTP normal");
            }

            HttpPost httpPost = new HttpPost(url);
            httpPost.setHeader("Content-Type", "application/json");
            httpPost.setHeader("Accept", "application/json");
            httpPost.setHeader("application", "MiAplicacion");
            httpPost.setHeader("username", "usuario123");
            httpPost.setHeader("token", "token123");
            httpPost.setHeader("Accept-Encoding", "gzip, deflate, br");
            httpPost.setHeader("Connection", "keep-alive");

            logger.debug("URL: " + url);
            for (Header header : httpPost.getAllHeaders()) {
                logger.debug(header.getName() + ": " + header.getValue());
            }

            ObjectMapper objectMapper = new ObjectMapper();
            objectMapper.configure(DeserializationFeature.FAIL_ON_UNKNOWN_PROPERTIES, false);
            objectMapper.configure(DeserializationFeature.READ_DATE_TIMESTAMPS_AS_NANOSECONDS, false);
            objectMapper.setDateFormat(new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss"));

            // Jackson serializa automáticamente el POJO o la lista de POJOs a JSON:
            String jsonRequest = objectMapper.writeValueAsString(request);
            logger.debug("JSON Request: " + jsonRequest);

            StringEntity entity = new StringEntity(jsonRequest, StandardCharsets.UTF_8);
            httpPost.setEntity(entity);

            CloseableHttpResponse httpResponse = httpClient.execute(httpPost);
            HttpEntity responseEntity = httpResponse.getEntity();
            String jsonResponse = EntityUtils.toString(responseEntity, StandardCharsets.UTF_8);

            logger.debug("JSON Response: " + jsonResponse);

            int statusCode = httpResponse.getStatusLine().getStatusCode();
            if (statusCode < 200 || statusCode > 300) {
                logger.error("HTTP Error: " + statusCode + " - " + httpResponse.getStatusLine().getReasonPhrase());
            }

            // Mapear respuesta JSON al POJO de respuesta
            return objectMapper.readValue(jsonResponse, GBDRespIngresarACHRecibidosRest.class);

        } finally {
            if (httpClient != null) {
                try {
                    httpClient.close();
                } catch (IOException e) {
                    logger.warn("Error cerrando HttpClient: " + e.getMessage(), e);
                }
            }
        }
    }

    /**
     * Nuevo método multihilo que acepta List<DetalleRequest> (POJO con sus 25 campos +
     idEncabezado).
     *
     * @param listaDTO    Lista de DetalleRequest ya parseados (cada uno lleva
     idEncabezado).
     * @param urlDetalle  URL del endpoint para envío de detalles.
     * @param useSSL      true si es HTTPS, false si es HTTP.
     */
    public void processDetallesDTO(List<DetalleRequest> listaDTO,
                                   String urlDetalle,
                                   boolean useSSL) throws InterruptedException {

        // 1. Leer configuración
        boolean enabled   = Boolean.parseBoolean(config.getProperty("multithread.enabled"
        , "false"));
        int maxThreads    = Integer.parseInt(config.getProperty("multithread.maxThreads",
         "4"));
        int chunkSize     = Integer.parseInt(config.getProperty("multithread.chunkSize",
        "500"));
        boolean sendBatch = Boolean.parseBoolean(config.getProperty(
        "multithread.sendBatch", "false"));

        logger.info("=== processDetallesDTO: enabled=" + enabled
                    + ", maxThreads=" + maxThreads
                    + ", chunkSize=" + chunkSize
                    + ", sendBatch=" + sendBatch
                    + " ===");

        // Si multithread está desactivado, procesar secuencial
        if (!enabled) {
            for (DetalleRequest dr : listaDTO) {
                try {
                    invokeRestServices(urlDetalle, dr, useSSL);
                } catch (IOException e) {
                    logger.error("Error enviando DetalleRequest ID=" + dr.getDetalleId()
                    + ": " + e.getMessage(), e);
                }
            }
            return;
        }

        // 2. Configurar pool de hilos
        ExecutorService executor = Executors.newFixedThreadPool(maxThreads);
        List<Future<Void>> futures = new ArrayList<>();

        // 3. Dividir listaDTO en sublistas de tamaño chunkSize
        List<List<DetalleRequest>> listaChunks = new ArrayList<>();
        for (int i = 0; i < listaDTO.size(); i += chunkSize) {
            listaChunks.add(listaDTO.subList(i, Math.min(i + chunkSize, listaDTO.size
            ())));
        }

        // 4. Para cada chunk, crear y enviar una tarea
        for (List<DetalleRequest> chunk : listaChunks) {
            Callable<Void> task = () -> {
                if (sendBatch) {
                    // Envío batch: Jackson serializará List<DetalleRequest> a un array
                    JSON
                    logger.debug("[" + Thread.currentThread().getName() + "] Enviando
                    batch de "
                                 + chunk.size() + " DetalleRequest.");
                    try {
                        invokeRestServices(urlDetalle, chunk, useSSL);
                    } catch (IOException e) {
                        logger.error("Error enviando batch en hilo "
                                     + Thread.currentThread().getName()
                                     + ": " + e.getMessage(), e);
                    }

                } else {
                    // Envío individual dentro del mismo hilo
                    for (DetalleRequest dr : chunk) {
                        logger.debug("[" + Thread.currentThread().getName()
                                     + "] Enviando DetalleRequest ID=" + dr.getDetalleId());
                        try {
                            invokeRestServices(urlDetalle, dr, useSSL);
                        } catch (IOException e) {
                            logger.error("Error enviando DetalleRequest ID="
                                         + dr.getDetalleId() + " en hilo "
                                         + Thread.currentThread().getName()
                                         + ": " + e.getMessage(), e);
                        }
                    }
                }
                return null;
            };
            futures.add(executor.submit(task));
        }

        // 5. Esperar a que todas las tareas terminen
        for (Future<Void> f : futures) {
            try {
                f.get();
            } catch (ExecutionException ex) {
                logger.error("Tarea devolvió excepción: " + ex.getMessage(), ex);
            }
        }

        // 6. Cerrar pool
        executor.shutdown();
        executor.awaitTermination(1, TimeUnit.HOURS);
        logger.info("processDetallesDTO finalizado.");
    }
}
```