```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.HeaderElement;
import org.apache.http.HeaderElementIterator;
import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.client.config.RequestConfig;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.conn.ConnectionKeepAliveStrategy;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.impl.conn.PoolingHttpClientConnectionManager;
import org.apache.http.message.BasicHeaderElementIterator;
import org.apache.http.protocol.HTTP;
import org.apache.http.protocol.HttpContext;
import org.apache.http.util.EntityUtils;
import org.apache.log4j.*;

import java.io.File;
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.text.SimpleDateFormat;
import java.util.Date;

public class APIRestComponentClient<T> {

    private static final Logger logger;

    // ✅ CAMBIO PRINCIPAL: HttpClient como variable de instancia reutilizable
    private static volatile CloseableHttpClient httpClient;
    private static volatile CloseableHttpClient sslHttpClient;
    private static final Object LOCK = new Object();
    private final ObjectMapper objectMapper;

    static {
        logger = Logger.getLogger(APIRestComponentClient.class);
        setupLogger();
    }

    // ✅ NUEVO: Constructor inicializa ObjectMapper
    public APIRestComponentClient() {
        this.objectMapper = createObjectMapper();
    }

    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());
        }
    }

    // ✅ NUEVO: Método para obtener HttpClient optimizado (singleton)
    private static CloseableHttpClient getHttpClient(boolean useSSL) {
        if (useSSL) {
            if (sslHttpClient == null) {
                synchronized (LOCK) {
                    if (sslHttpClient == null) {
                        sslHttpClient = createOptimizedHttpClient();
                        logger.debug("HttpClient SSL creado y optimizado");
                    }
                }
            }
            return sslHttpClient;
        } else {
            if (httpClient == null) {
                synchronized (LOCK) {
                    if (httpClient == null) {
                        httpClient = createOptimizedHttpClient();
                        logger.debug("HttpClient regular creado y optimizado");
                    }
                }
            }
            return httpClient;
        }
    }

    // ✅ NUEVO: Crear HttpClient optimizado con pooling y keep-alive
    private static CloseableHttpClient createOptimizedHttpClient() {
        // Configurar connection manager para reutilizar conexiones
        PoolingHttpClientConnectionManager connectionManager = new PoolingHttpClientConnectionManager();
        connectionManager.setMaxTotal(20);
        connectionManager.setDefaultMaxPerRoute(10);

        // Configurar keep-alive strategy
        ConnectionKeepAliveStrategy keepAliveStrategy = new ConnectionKeepAliveStrategy() {
            @Override
            public long getKeepAliveDuration(HttpResponse response, HttpContext context) {
                HeaderElementIterator it = new BasicHeaderElementIterator(
                    response.headerIterator(HTTP.CONN_KEEP_ALIVE));
                while (it.hasNext()) {
                    HeaderElement he = it.nextElement();
                    String param = he.getName();
                    String value = he.getValue();
                    if (value != null && param.equalsIgnoreCase("timeout")) {
                        return Long.parseLong(value) * 1000;
                    }
                }
                return 30 * 1000; // 30 segundos por defecto
            }
        };

        // Configurar timeouts
        RequestConfig requestConfig = RequestConfig.custom()
            .setConnectTimeout(5000)
            .setSocketTimeout(10000)
            .setConnectionRequestTimeout(3000)
            .build();

        return HttpClients.custom()
            .setConnectionManager(connectionManager)
            .setKeepAliveStrategy(keepAliveStrategy)
            .setDefaultRequestConfig(requestConfig)
            .build();
    }
```

```java

    // ✅ NUEVO: Crear ObjectMapper reutilizable
    private ObjectMapper createObjectMapper() {
        ObjectMapper mapper = new ObjectMapper();
        mapper.configure(DeserializationFeature.FAIL_ON_UNKNOWN_PROPERTIES, false);
        mapper.configure(DeserializationFeature.READ_DATE_TIMESTAMPS_AS_NANOSECONDS,
        false);
        mapper.setDateFormat(new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss"));
        return mapper;
    }

    // ✅ MODIFICADO: Tu método principal con optimizaciones
    public T invokeRestServices(String url, Object request, boolean useSSL, Class<T>
    responseClass) throws IOException {

        // ✅ CAMBIO: Obtener cliente reutilizable en lugar de crear uno nuevo
        CloseableHttpClient client = getHttpClient(useSSL);
        CloseableHttpResponse httpResponse = null;

        try {
            if (useSSL) {
                logger.debug("Usando conexión SSL/TLS optimizada");
            } else {
                logger.debug("Usando conexión HTTP optimizada");
            }

            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);
            logger.debug("HEADER del request:");
            Header[] requestHeaders = httpPost.getAllHeaders();
            for (Header header : requestHeaders) {
                logger.debug(header.getName() + ": " + header.getValue());
            }

            // ✅ CAMBIO: Usar ObjectMapper reutilizable
            String jsonRequest = objectMapper.writeValueAsString(request);
            logger.debug("JSON Request: " + jsonRequest);

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

            httpResponse = client.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) {
                String errorMsg = "HTTP Error: " + statusCode + " - " + httpResponse.
                getStatusLine().getReasonPhrase();
                logger.error(errorMsg);
                throw new IOException(errorMsg + ". Response: " + jsonResponse);
            }

            T result = objectMapper.readValue(jsonResponse, responseClass);

            // ✅ NUEVO: Consumir completamente la respuesta para liberar conexión
            EntityUtils.consume(responseEntity);

```

```java
                return result;

            } finally {
                // ✅ CAMBIO CRÍTICO: NO cerrar httpClient, solo cerrar httpResponse
                if (httpResponse != null) {
                    try {
                        httpResponse.close();
                    } catch (IOException e) {
                        logger.warn("Error cerrando HttpResponse: " + e.getMessage());
                    }
                }
                // ❌ REMOVIDO: httpClient.close() - ya no se cierra aquí
            }
        }

        // ✅ NUEVO: Método para cerrar recursos al final de la aplicación
        public static void shutdown() {
            if (httpClient != null) {
                try {
                    httpClient.close();
                    logger.debug("HttpClient regular cerrado");
                } catch (IOException e) {
                    logger.warn("Error cerrando HttpClient regular: " + e.getMessage());
                }
            }

            if (sslHttpClient != null) {
                try {
                    sslHttpClient.close();
                    logger.debug("HttpClient SSL cerrado");
                } catch (IOException e) {
                    logger.warn("Error cerrando HttpClient SSL: " + e.getMessage());
                }
            }
        }

        // ✅ NUEVO: Shutdown hook automático
        static {
            Runtime.getRuntime().addShutdownHook(new Thread(() -> {
                logger.debug("Ejecutando shutdown automático de APIRestComponentClient");
                shutdown();
            }));
        }
    }
```