```java
package net.wiredclub.translation;

import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.node.ArrayNode;
import com.flipkart.zjsonpatch.JsonDiff;
import com.flipkart.zjsonpatch.JsonPatch;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.io.IOException;
import java.util.Iterator;

import static net.wiredclub.translation.DeepLHelper.DeepLUsage;
import static net.wiredclub.translation.TranslationStatusCode.STATUS_BAD_AS_HELL;
import static net.wiredclub.translation.TranslationStatusCode.STATUS_FILE_NOT_FOUND;
import static net.wiredclub.translation.TranslationStatusCode.STATUS_JSON_INVALID;
import static net.wiredclub.translation.TranslationStatusCode.STATUS_OK;
import static net.wiredclub.translation.TranslationStatusCode.STATUS_TRANSLATION_FILE_INVALID;

/**
 * Translate all modified keys from source language into target language.
 * The previous version with translations is taken from previous commit.
 */
public class TranslationTool {

    private static final Logger LOG = LoggerFactory.getLogger(TranslationTool.class);

    private final JsonHelper jsonHelper;
    private final DeepLHelper deepLHelper;
    private final FileHelper fileHelper;
    private final CommandLineHelper commandLineHelper;

    private TranslationConfig cfg;

    /**
     * Without using CDI we instantiate all required classes here. For mocking
     * of some classes the second constructor is used.
     */
    TranslationTool() {
        this.jsonHelper = new JsonHelper();
        this.deepLHelper = new DeepLHelper(jsonHelper);
        this.fileHelper = new FileHelper();
        this.commandLineHelper = new CommandLineHelper(deepLHelper, fileHelper);
    }

    /**
     * Constructor for tests.
     *
     * @param jsonHelper either real or mock class
     * @param deepLHelper either real or mock class
     * @param fileHelper either real or mock class
     * @param commandLineHelper either real or mock class
     */
    TranslationTool(JsonHelper jsonHelper, DeepLHelper deepLHelper, FileHelper fileHelper,
                    CommandLineHelper commandLineHelper) {
        this.jsonHelper = jsonHelper;
        this.deepLHelper = deepLHelper;
        this.fileHelper = fileHelper;
        this.commandLineHelper = commandLineHelper;
    }

    public static void main(String[] args) {
        System.exit(new TranslationTool().run(args).exitCode());
    }

    TranslationStatusCode run(String[] args) {
        try {
            cfg = commandLineHelper.getTranslationConfig(args);
            processTranslation();
        } catch (TranslationException e) {
            String message = e.getMessage();
            if (message ≠ null && !message.isBlank()) {
                LOG.warn(message);
            }
```

```java
                return e.statusCode();
            } catch (Throwable e) {
                LOG.error(e.getMessage(), e);
                return STATUS_BAD_AS_HELL;
            }
            return STATUS_OK;
        }

        /**
         * <ol>
         *     <li>Read source file and find differences to the previous version</li>
         *     <li>For every target language</li>
         *     <ol>
         *         <li>Read target file and find differences to source file</li>
         *         <li>Translate text and create patch operation changes between target to source</li>
         *         <li>Translate remaining source text changes and create patch operation for source</li>
         *         <li>Write output file</li>
         *     </ol>
         * </ol>
         *
         * Look at the activity diagram in documentation folder for a graphical overview.
         *
         * @throws TranslationException throws exception if translation is not possible
         * @throws IOException throws exception if an error during file IO occurs
         */
        private void processTranslation() throws TranslationException, IOException {
            // run information for devs
            if (LOG.isDebugEnabled()) {
                LOG.info("Translation tool started.");
                LOG.info("Source language: {}", cfg.sourceLanguage());
                LOG.info("Target language(s): {}", cfg.targetLanguages());
                LOG.info("Translations directory: {}", cfg.translationsDirectory());
                DeepLUsage usage = deepLHelper.usage();
                LOG.info("DeepL translations possible: {}/{}", usage.characterCount(), usage.characterLimit());
            }

            // read actual main.json
            JsonNode sourceJson = getTranslationFile(cfg.sourceFileName());

            // find all changes from previous version of main.json to actual main.json
            JsonNode sourceDiffPatch = findChangesInSource(sourceJson);

            for (String targetLanguage : cfg.targetLanguages()) {
                boolean translationsFileChanged = false;

                JsonNode targetJson = getTranslationFile(cfg.targetFileName(targetLanguage));

                // This call is a bit weird, because we use target json as first parameter (source) and source as
                // second (target). This is because the names are used in a different context. We want to know which keys
                // need to be added to or removed from target json in comparison to source json. The target json will be
                // transformed into the same structure as source json.
                JsonNode targetDiffPatch = JsonDiff.asJson(targetJson, sourceJson);
                // LOG.debug("target to source diff patch: {}", targetKeyDiffPatch.toPrettyString());

                if (!targetDiffPatch.isEmpty()) {
                    // create patch with add or remove fields (field values will be translated).
                    JsonNode translationPatch = translateTargetDiffPatch(targetDiffPatch, targetLanguage);
                    if (!translationPatch.isEmpty()) {
                        translationsFileChanged = true;
                        LOG.info("Created patch (KEYS DIFF) with {} operation(s)/translation(s) for '{}'.",
                                translationPatch.size(), cfg.targetFileName(targetLanguage));
                        // LOG.debug("{}", translationPatch.toPrettyString());

                        // add or remove keys in target json
                        JsonPatch.applyInPlace(translationPatch, targetJson);
                    }
                }

                if (!sourceDiffPatch.isEmpty()) {
                    // create patch with replace operations
                    JsonNode translationPatch = translateSourceDiffPatch(sourceDiffPatch, targetLanguage);
                    if (!translationPatch.isEmpty()) {
                        translationsFileChanged = true;
                        LOG.info("Created patch (VALUE DIFF) with {} translation(s) for '{}'.",
                                translationPatch.size(), cfg.targetFileName(targetLanguage));
                        // LOG.debug("{}", translationPatch.toPrettyString());

                        // replace keys in target json
                        JsonPatch.applyInPlace(translationPatch, targetJson);
                    }
                }

                // write result into target directory and overwrite existing translation file.
                if (translationsFileChanged) {
                    writeTargetTranslationFile(targetJson, targetLanguage);
                }
            }
            LOG.info("Translation process finished but files were not committed and pushed. "
                    + "Please verify translation files and commit and push them.");
        }

        private JsonNode getTranslationFile(String filename) throws TranslationException {
            try {
                String source = fileHelper.readFile(cfg.repositoryDirectory() + "/" + filename);
                return jsonHelper.convertStringToJson(source);
                // LOG.debug("source json: {}", sourceJson.toPrettyString());
            } catch (TranslationJsonProcessingException e) {
                throw new TranslationException(
                        "Error: Invalid Json. Please verify that the file '" + filename + "' is valid json. "
                                + "Cause: " + e.getMessage(),
                        STATUS_JSON_INVALID);
            } catch (TranslationFileNotFoundException e) {
                throw new TranslationException("Error: '" + filename + "' not found. "
                        + "Please verify that the file exists.", STATUS_FILE_NOT_FOUND);
            }
        }

        private JsonNode findChangesInSource(JsonNode sourceJson) throws IOException, TranslationException {
            String sourceFileName = cfg.sourceFileName();
            try {
                return createDiffPatch(cfg.repositoryDirectory(), sourceFileName, sourceJson);
                // LOG.debug("diff json patch: {}", diffPatch.toPrettyString());
            } catch (TranslationFileNotFoundException e) {
                throw new TranslationException("Error: '" + sourceFileName + "' not found. "
                        + "Please verify that the previous version of file exists in git.", STATUS_FILE_NOT_FOUND);
            } catch (TranslationJsonProcessingException e) {
                throw new TranslationException(
                        "Error: Invalid Json. Please verify that the file '" + sourceFileName + "' is valid json.",
                        STATUS_JSON_INVALID);
            }
        }

        JsonNode createDiffPatch(String repositoryDirectory, String previousTranslationsFileName,
                                 JsonNode actualTranslationsJson)
                throws TranslationFileNotFoundException, TranslationJsonProcessingException, IOException {
            String previousTranslations =
                    fileHelper.readPreviousFileFromHistory(repositoryDirectory, previousTranslationsFileName, 1);

            JsonNode previousTranslationsJson = jsonHelper.convertStringToJson(previousTranslations);
            return JsonDiff.asJson(previousTranslationsJson, actualTranslationsJson);
        }

        /**
         * Translates target diff patch. Only add and remove operations are handled here.
         * Replace operation will be handled by translate source diff patch.
         * All other operations are not needed.
         *
         * @param diffPatch changes of source file
         * @param targetLanguage the target language
         *
         * @return a translation patch
         *
         * @throws TranslationException thrown if the translation patch is an invalid json
         * @throws IOException thrown if an error occurs during file access
         */
        private JsonNode translateTargetDiffPatch(JsonNode diffPatch, String targetLanguage)
                throws TranslationException, IOException {
            ArrayNode translationPatch = jsonHelper.createNewTranslationPatch();

            if (diffPatch.isArray()) {
                for (int i = 0; i < diffPatch.size(); i++) {
                    JsonNode command = diffPatch.get(i);
                    // LOG.debug("command: {}", command);

                    String op = command.get("op").asText();
                    String path = command.get("path").asText();
                    switch (op) {
                        case "add":
                            JsonNode value = command.get("value");
                            translationPatch.add(jsonHelper.createPatchOperationAdd(path, value));
                            traverse(translationPatch, targetLanguage, path, value);
                            break;
                        case "remove":
                            translationPatch.add(jsonHelper.createPatchOperationRemove(path));
                            break;
                        default:
                            // replace would do an unnecessary translation.
                            // translations for operation move and copy is also not needed.
                    }
                }
            }

            return translationPatch;
        }

        /**
         * Translates source diff patch. Only the replace operation is handled here.
         * All other operations are already handled or not needed.
         *
         * @param diffPatch changes of source file
         * @param targetLanguage the target language
         *
         * @return a translation patch
         *
         * @throws TranslationException thrown if the translation patch is an invalid json
         * @throws IOException thrown if an error occurs during file access
         */
        private JsonNode translateSourceDiffPatch(JsonNode diffPatch, String targetLanguage)
                throws TranslationException, IOException {
            ArrayNode translationPatch = jsonHelper.createNewTranslationPatch();

            if (diffPatch.isArray()) {
                for (int i = 0; i < diffPatch.size(); i++) {
                    JsonNode command = diffPatch.get(i);
                    // LOG.debug("command: {}", command);

                    String op = command.get("op").asText();
                    // for operations add, remove, move, and copy translation is not needed
                    if ("replace".equals(op)) {
                        traverse(translationPatch, targetLanguage, command.get("path").asText(), command.get("value"));
                    }
                }
            }

            return translationPatch;
        }

        /**
         * Recursive approach to iterate through json tree.
         *
         * @param patch operations how to change the target json will be added to the patch
         * @param targetLanguage the target language
         * @param path the path is a unique identifier. it is build from all successor field names and the actual field name.
         * @param jsonNode the json node to be evaluated
         *
         * @throws TranslationException thrown if an array is defined in json, or translation has an invalid json
         * @throws IOException thrown if an error occurs during file access
         */
        private void traverse(ArrayNode patch, String targetLanguage, String path, JsonNode jsonNode)
                throws TranslationException, IOException {
            if (jsonNode.isObject()) {
                Iterator<String> fieldNames = jsonNode.fieldNames();
```

```java
                while (fieldNames.hasNext()) {
                    String fieldName = fieldNames.next();
                    JsonNode fieldValue = jsonNode.get(fieldName);
                    traverse(patch, targetLanguage, path + "/" + fieldName, fieldValue);
                }
            } else if (jsonNode.isArray()) {
                throw new TranslationException("Error: Arrays are not allowed in translation file 'main.json'.",
                        STATUS_TRANSLATION_FILE_INVALID);
            } else {
                String textToTranslate = jsonNode.asText();
                String translation = deepLHelper.translate(textToTranslate, cfg.sourceLanguage(), targetLanguage);
                JsonNode command = jsonHelper.createPatchOperationReplace(path, translation);
                patch.add(command);
            }
        }

    /**
     * @param appliedTranslationPatch a json that holds all values which should be written to an output file
     * @param targetLanguage the desired target language
     *
     * @throws TranslationException thrown if the translation patch is an invalid json
     * @throws IOException thrown if an error occurs during file access
     */
    private void writeTargetTranslationFile(JsonNode appliedTranslationPatch, String targetLanguage)
            throws TranslationException, IOException {
        try {
            String target = jsonHelper.convertJsonToString(appliedTranslationPatch);
            String targetFileName = cfg.repositoryDirectory() + "/" + cfg.targetFileName(targetLanguage);
            fileHelper.writeFile(targetFileName, target);
            LOG.info("File written to '{}'.", targetFileName);
        } catch (TranslationJsonProcessingException e) {
            throw new TranslationException(
                    "Error: Could not create a valid json file. Something has gone wrong. Please check.",
                    STATUS_JSON_INVALID);
        }
    }
}
```