```java
package com.uber.example.boxintegration;

import com.box.sdk.BoxAPIException;
import com.box.sdk.BoxCollaboration;
import com.box.sdk.BoxConfig;
import com.box.sdk.BoxDeveloperEditionAPIConnection;
import com.box.sdk.BoxFile;
import com.box.sdk.BoxFileUploadSession;
import com.box.sdk.BoxFileUploadSessionPart;
import com.box.sdk.BoxFileUploadSessionPartList;
import com.box.sdk.BoxFolder;
import com.box.sdk.BoxItem;
import com.box.sdk.BoxSearch;
import com.box.sdk.BoxSearchParameters;
import com.box.sdk.BoxSharedLink;
import com.box.sdk.BoxUser;
import com.box.sdk.IAccessTokenCache;
import com.box.sdk.InMemoryLRUAccessTokenCache;
import com.box.sdk.PartialCollection;
import com.uber.example.boxintegration.config.Configuration;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.Reader;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.security.DigestInputStream;
import java.security.MessageDigest;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.concurrent.atomic.AtomicReference;
import java.util.logging.Level;
import java.util.logging.Logger;

import org.yaml.snakeyaml.Yaml;

import javax.xml.bind.DatatypeConverter;


public final class UberBoxUtil {
  private static final Logger LOGGER = Logger.getLogger(UberBoxUtil.class.getName());
  private static final String YAML_PATH = "resources/box.yaml";
  private static final String XXUBER_TOP = "XXUBER_TOP";

  public static final String BOX_ORACLE_NONPROD = "test-oracle";
  public static final String BOX_ORACLE_PROD = "oracle_ebs";

  private static final int MAX_CACHE_ENTRIES = 100;
  public static final int CHUNKED_UPLOAD_MINIMUM = 20000;

  private static Configuration config = null;
  private static final Map<String, BoxDeveloperEditionAPIConnection> clientMap = new HashMap<>();


  private UberBoxUtil() {
    LOGGER.log(Level.SEVERE, "UberBoxUtil private constructor should never be instantiated. Throwing RuntimeException."
    throw new RuntimeException("UberBoxUtil private constructor should never be instantiated");
  }


  /**
   * private method to load yaml file, which holds the paths to JWT files.
   */
  private static void loadYaml(String yamlPath) {
    Yaml yaml = new Yaml();
    if (yamlPath == null || yamlPath.isEmpty()) {
//      yamlPath = String.format("%s/%s", System.getenv(XXUBER_TOP), YAML_PATH);
      yamlPath = "/Users/vkalyani/Downloads/box.yaml";
    }

    try( InputStream in = Files.newInputStream(Paths.get(yamlPath))) {
      config = yaml.loadAs(in, Configuration.class );
      LOGGER.log(Level.INFO, config.toString() );
    } catch (IOException e) {
      LOGGER.log(Level.SEVERE, "Exception during loadYaml method" + e.getMessage());
      e.printStackTrace();
    }
  }


  /**
   * private method to get the Box Developer API Connection object
   * @param key   -- box custom app name
   * @param defKey  -- the app's jwt file location
   * @return BoxDeveloperAPIConnection object
   */
  private static BoxDeveloperEditionAPIConnection getBoxDeveloperEditionAPIConnection(String key, String defKey) {
    BoxConfig boxConfig = null;
    BoxDeveloperEditionAPIConnection client;
    try(Reader reader = new FileReader(defKey)) {
      boxConfig = BoxConfig.readFrom(reader);
    } catch (IOException e) {
      e.printStackTrace();
    }
    IAccessTokenCache accessTokenCache = new InMemoryLRUAccessTokenCache(MAX_CACHE_ENTRIES);
    client = BoxDeveloperEditionAPIConnection.getAppEnterpriseConnection(boxConfig, accessTokenCache);
    return client;
  }


  /**
   * utility method to get a Box API Connection given the custom app name
   * @param appName - Box custom app
   * @return BoxDeveloperAPIConnection object
   */
  public static BoxDeveloperEditionAPIConnection getClient(String appName) {
    return getClient(appName, null);
  }



  /**
   * utility method to get a Box API Connection given the custom app name
   * @param appName - Box custom app
   * @param yamlPath -
   * @return BoxDeveloperAPIConnection object
   */
  public static BoxDeveloperEditionAPIConnection getClient(String appName, String yamlPath) {
    Map<String, String> serviceAccounts;

    AtomicReference<BoxDeveloperEditionAPIConnection> client = new AtomicReference<>(clientMap.get(appName));
    if (client.get() == null) {
      if (config == null) {
        loadYaml(yamlPath);
      }
      serviceAccounts = config.getServiceAccounts();
//      String svcAcctPath;
//      if (XXUBER_TOP.equalsIgnoreCase(config.getBasePath())) {
//        svcAcctPath = String.format("%s/%s", System.getenv(XXUBER_TOP), serviceAccounts.get(appName));
//      } else {
//        svcAcctPath = String.format("%s/%s", config.getBasePath(), serviceAccounts.get(appName));
//      }
//      client.set(UberBoxUtil.getBoxDeveloperEditionAPIConnection(appName, svcAcctPath));
      client.set(UberBoxUtil.getBoxDeveloperEditionAPIConnection(appName, serviceAccounts.get(appName)));
      clientMap.put(appName, client.get());
    }
    return client.get();
  }



  /**
   * utility method to return a default Box API Connection for the NON-PROD deemed custom app
   * @return BoxDeveloperAPIConnection object
   */
  public static BoxDeveloperEditionAPIConnection getDefaultNonProdClient() {
    return getClient(BOX_ORACLE_NONPROD);
  }



  /**
   * utility method to return a default Box API Connection for the PROD deemed custom app
   * @return BoxDeveloperAPIConnection object
   */
  public static BoxDeveloperEditionAPIConnection getDefaultProdClient() {
    return getClient(BOX_ORACLE_PROD);
  }



  private static String createSubFolders(BoxDeveloperEditionAPIConnection client, String folderName) {
    String _folderId = null;
    String[] destFolersList = folderName.split("/");
    String tFolderId = getFolderId(client, destFolersList[0]);
    for (int i = 1; i < destFolersList.length; i++) {
      _folderId = getFolderId(client, destFolersList[i]);
      if (_folderId == null || _folderId.isEmpty()) {
        _folderId = createFolder(client,tFolderId, destFolersList[i]);
        LOGGER.log(Level.INFO,"folder with name: " + folderName + " is created, its ID: "+_folderId);
      }
      tFolderId = _folderId;
    }
    return _folderId;
  }


  private static BoxFolder getBoxFolder(BoxDeveloperEditionAPIConnection client, String folderName, String folderId) {
    BoxFolder folder = null;
    if (folderId != null && !(folderId.isEmpty())) {
      folder = new BoxFolder(client, folderId);
    } else {
      String _folderId = null;
      if (folderName.contains("/")) {
        _folderId = createSubFolders(client, folderName);
      } else {
        _folderId = getFolderId(client, folderName);
        if (_folderId == null || _folderId.isEmpty()) {
          _folderId = createFolder(client,"0", folderName);
          LOGGER.log(Level.INFO,"folder with name: " + folderName + " is created, its ID: "+_folderId);
        }
      }
      folder = new BoxFolder(client, _folderId);
    }
    return folder;
  }

  /**
   * method to upload a file to a box folder
   * @param filePath - full path of the file being uploaded
   * @param appName - Box custom app
   * @param folderId - box folderId to where the file is being uploaded
   * @return boolean value
   */
  public static Boolean putFile(String appName, String folderId, String filePath) throws IOException {
    return putFile(appName, "", folderId, filePath, "");
  }
  /**
   * method to upload file given appname, destination folder or its id, input file path and its to-file name
   * @param appName - box servcie acct. name
   * @param folderName - box destination folder name
   * @param folderId - box destination folder Id
   * @param filePath - full path of file being uploaded
   * @param toFileName - uploaded name of input file
   * @return
   * @throws IOException
   */
  public static Boolean putFile(String appName, String folderName, String folderId, String filePath, String toFileName)
      throws IOException {
    BoxDeveloperEditionAPIConnection client = getClient(appName);
    return putFile(client,filePath, toFileName, folderName, folderId);
  }


  /**
   * method to upload file given box api client, destination folder or its id, input file path and its to-file name
   * @param client
   * @param filePath
   * @param toFileName
   * @param folderName
   * @param folderId
   * @return
   * @throws FileNotFoundException
   */
  public static Boolean putFile(BoxDeveloperEditionAPIConnection client, String filePath, String toFileName, String fol
      , String folderId) throws FileNotFoundException {
    BoxFolder folder = getBoxFolder(client,folderName, folderId);
//    BoxFolder.Info folderInfo = folder.getInfo("size", "id", "owned_by", "name");
//    LOGGER.log(Level.INFO,"name = "+folderInfo.getName() + " ,size = "+folderInfo.getSize()+", owned_by = "+ folderIn
    File file = new File(filePath);
    FileInputStream stream = new FileInputStream(file);
    BoxFile.Info newFileInfo = folder.uploadFile(stream, (toFileName == null ? file.getName() : toFileName));
    return Boolean.TRUE;
  }


  /**
   * method to upload file and get its shared link
   * @param filePath
   * @param folderId
   * @param appName
   * @return
   */
  public  static String putFileAndGetSharedLink(String filePath, String folderId, String appName) {
    BoxFolder folder = new BoxFolder(UberBoxUtil.getClient(appName), folderId);
    File file = new File(filePath);
    String url = null;
    try (FileInputStream stream = new FileInputStream(file)) {
      BoxFile uploadedFile = folder.uploadFile(stream, file.getName()).getResource();

      BoxSharedLink.Permissions permissions = new BoxSharedLink.Permissions();
      permissions.setCanDownload(true);
      permissions.setCanPreview(true);
      BoxSharedLink sharedLink = uploadedFile.createSharedLink(BoxSharedLink.Access.OPEN, null, permissions);
      url = sharedLink.getURL();
    } catch (FileNotFoundException e) {
      LOGGER.log(Level.SEVERE,e.getMessage());
      e.printStackTrace();
    } catch (IOException e) {
      LOGGER.log(Level.SEVERE,e.getMessage());
      e.printStackTrace();
    }
    return url;
  }

  /**
   * method to download a file given the file's box fileId, its destination directory and box app name
   * @param fileId  box fileID
   * @param localPath local directory path
```