```
from datetime import datetime
import numpy as np
import os
import pandas as pd

from src.settings import REGISTRY, TEST_SIZE
from utils.fms import get_registry_inputdir_path, get_registry_outputdir_path, get_registry_codes
from utils.helpers import ld_to_dl, sanitize_str_cols, str_to_int


class Registry:
    def __init__(self, extract=True, transform=True, load=True, verbose=False, is_test=False, test_size=TEST_SIZE):
        """
        Parent container for extracting, transforming, and loading aircraft registry data from a remote source

        :param (bool) extract: if 'True' run child method to extract aircraft registry data from remote source
        :param (bool) transform: if 'True' run child method to transform aircraft registry data extracted from source
        :param (bool) load: if 'True' run method to write remote source's transformed aircraft registry data
        :param (bool) verbose: if 'True' print progress messages
        :param (bool) is_test: if 'True' sample aircraft registry data source for development purposes
        :param (int) test_size: number of observations to sample from registry data source when 'test' argument is 'True'
        """

        # assign argument values
        self.extract = extract
        self.transform = transform
        self.load = load
        self.verbose = verbose
        self.is_test = is_test
        self.test_size = test_size

        # initialize constant values for remote source's aircraft registry data
        self.retrieved = None
        self.source_url = None
        self.is_official = None
        self.source_name = None
        self.updated = None
        self.registry_code = None

        # initialize storage for path to retrieved-date file for remote source's registry input data
        self.retrieved_path = None
        # initialize storage for remote source's registry data
        self.registry = None
        # initialize storage for remote source's registry input data extraction method
        self.extractor = None
        # initialize storage for updated-date scraping method for remote source's registry input data
        self.updated_scraper = None
        # initialize storage for remote source's registry input data reading method
        self.reader = None
        # initialize storage for remote source's registry data wrangling method
        self.wrangler = None
        # initialize storage for remote source's transformed registry data
        self.wrangled_records = []

    def etl_registry(self):
        """
        Runs steps to extract aircraft registry data from a remote source, transform the extracted data into a
        standardized format, and load the transformed data into a tabular store
        """

        if self.check_inheritance():
            # assign path to retrieved-date file for remote source's registry input data
            self.retrieved_path = self.get_inputfile_path(self.registry_code.lower() + "_retrieved_date.txt")

            self.__extract_registry()
            self.__scrape_registry_updated()
            self.__read_registry()
            self.__transform_registry()
            self.__validate_registry()
            self.__load_registry()

    def check_inheritance(self):
        """
        Verifies whether class instance is used appropriately as an inherited parent class (not an independent instance)

        :return: 'True' if instance is used appropriately
        """

        if self.registry_code is None or self.registry_code not in get_registry_codes():
            print("Warning: the Registry class is meant to be inherited by an instance of a class in the 'registries' "
                  "module and has no utility as an independent object")

            return False
        else:
            return True

    def get_inputdir_path(self):
        """
        Builds path to remote source's registry input data or test directory

        :return: path to target directory
        """

        path = os.path.join(get_registry_inputdir_path(self.is_test), self.registry_code.lower())

        return path

    def get_inputfile_path(self, filename):
        """
        Builds path to specific remote source's registry input data or test file

        :param (str) filename: filename with extension of target file
        :return: path to target file
        """

        path = os.path.join(self.get_inputdir_path(), filename)

        return path

    def set_input_registry(self, registries):
        """
        Assigns remote source's aircraft registry input data and builds test set if indicated

        :param (list) registries: list of pandas.DataFrame that comprise remote source's raw registry
        :return: effect - modifies self.registry
        """

        if self.is_test:
            for i in range(len(registries)):
                # subset remote source's registry data for development purposes
                registries[i] = registries[i].loc[0:self.test_size - 1, ]

        self.registry = pd.concat(registries, ignore_index=True)

    def __extract_registry(self):
        """
        Extracts aircraft registry input data from a remote source

        :return: effect - creates [DATA_DIR|TEST_DIR]/[REGISTRY_INPUT_DIR]/[registry_code]/[file or directory] asset(s);
            effect - modifies self.retrieved
        """

        if self.extract:
            self.__print_progress("extracting")

            # run remote source's registry input data extraction method
            self.extractor()

            # assign and write retrieved-date for remote source's registry input data
            self.retrieved = datetime.now().isoformat()

            with open(self.retrieved_path, "w") as file:
                file.write(self.retrieved)

    def __scrape_registry_updated(self):
        """
        Scrapes the updated-date for remote source's registry input data

        :return: effect - modifies self.updated
        """

        if self.updated_scraper is not None and self.updated is None:
            if len(os.listdir(self.get_inputdir_path())) > 0:
                # handle when remote source's registry input data has been extracted
                self.__print_progress("scraping updated-date for")

                self.updated_scraper()
            else:
                # handle when remote source's registry input data has not been extracted
                print("Warning:", self.registry_code, "registry data does not exist... ensure it has been extracted")

    def __read_registry(self):
        """
        Reads remote source's registry input data

        :return: effect - modifies self.registry
        """

        if self.registry is None:
            if len(os.listdir(self.get_inputdir_path())) > 0:
                # handle when remote source's registry input data has been extracted
                self.__print_progress("reading")

                self.reader()
            else:
                # handle when remote source's registry input data has not been extracted
                print("Warning:", self.registry_code, "registry data does not exist... ensure it has been extracted")

    def __transform_registry(self):
        """
        Transforms extracted aircraft registry data into a standardized format

        :return: effect - modifies self.registry
        """

        if self.transform and self.registry is not None:
            self.__print_progress("transforming")

            if self.retrieved is None and os.path.isfile(self.retrieved_path):
                # handle when remote source's registry input data has been extracted but not in current session
                with open(self.retrieved_path, "r") as file:
                    # assign retrieved-date for remote source's registry input data
                    self.retrieved = file.read()

            # run remote source's wrangling method for transforming registry data
            self.wrangler()
            # initialize remote source's final registry with constant values populated
            registry = self.__get_registry_template()

            # populate remote source's final registry with transformed values
            for key, value in ld_to_dl(self.wrangled_records).items():
                registry[key] = value

            self.registry = registry

    def __get_registry_template(self):
        """
        Initializes remote source's final aircraft registry with constant values populated

        :return: model of remote source's final registry with constant values populated
        """

        [illegible]

    def __validate_registry(self):
        """
        Standardizes, cleans, and formats remote source's registry data to facilitate SQL table integration

        :return: effect - modifies self.registry
        """

        if self.transform and self.registry is not None:
            self.registry = sanitize_str_cols(self.registry)
            # convert year to integer values
            self.registry["year"] = self.registry["year"].apply(lambda x: str_to_int(x))
            # cast types to match SQL table fields
            self.registry = self.registry.astype(REGISTRY["schema"])
            # standardize null values
            self.registry.replace({np.nan: None}, inplace=True)

    def __load_registry(self):
        """
        Writes remote source's transformed aircraft registry data to tabular store

        :return: effect - creates [DATA_DIR|TEST_DIR]/[REGISTRY_OUTPUT_DIR]/[self.registry_code]/[self.registry_code]
[REGISTRY.name] file
        """

        if self.load and self.registry is not None:
            self.__print_progress("writing")

            filename = self.registry_code.lower() + REGISTRY["name"]
            path = os.path.join(get_registry_outputdir_path(self.is_test), self.registry_code.lower(), filename)
            self.registry.to_csv(path, index=False, line_terminator="\n")

    def __print_progress(self, step):
        """
        Displays progress messages

        :param (str) step: gerund of step in ETL process
        """

        if self.verbose:
            print("Progress:", step, self.registry_code, "registry data")
```