```python
#!/usr/bin/python3.5
# -*-coding:Utf-8 -*

from json import loads
from re import findall
from operator import itemgetter

from .JSONEncoder import JSONEncoder
from .PathManager import PathManager
from src.factories.IAFactory import IAFactory
from src.EvolutiveGenerator.ProcessusState import ProcessusState
from src.EvolutiveGenerator.event_names import *


class Reader:
    """Read files

    This is a static class.

    Public API:
        processusExists(processus_id)
        getProcessusState(processus_id)
        getIa(processus_id, ia_id)
        getBestIa(processus_id, generation_id=None)
        getData(processus_id)
    """


    @staticmethod
    def getPath(*args, **kwargs):
        return PathManager.getPath(*args, **kwargs, read_only=True)


    @staticmethod
    def readJSON(path):
        return loads(path.read_text())


    @staticmethod
    def readGrading(path):
        return [tuple(loads(json_array)) for json_array in findall('\[.+\]', path.read_text())]


    @classmethod
    def getProcessusParams(cls, processus_id):
        path = cls.getPath(processus_id)
        if not path.parent.exists():
            raise ValueError("Processus {} doesn't exists.".format(processus_id))
        if not path.exists():
            raise ValueError("Processus {} doesn't have processus.json file.".format(processus_id))

        return cls.readJSON(path)


    @classmethod
    def getLastGeneration(cls, processus_id, generations):
        '''Get id of the processus' last generation, else -1 '''
        # Get first inexistant generation
        generation_id = 0
        while cls.getPath(processus_id, generations, generation_id).parent.exists():
            generation_id += 1

        return generation_id - 1


    @classmethod
    def getLastGradedGeneration(cls, processus_id, generations):
        # Get first inexistant final_grading file's generation
        generation_id = 1
        while cls.getPath(processus_id, generations, generation_id, 'final_grading').exists():
            generation_id += 1

        return generation_id - 2


    @classmethod
    def getGenerationOf(cls, processus_id, generations, ia_id):
        generation_id = 1
        while True:
            path = cls.getPath(processus_id, generations, generation_id, 'final_grading')
            if not path.exists():
                raise ValueError("IA {} doesn't exist !".format(ia_id))
            final_grading = cls.readJSON(path)
            for score, _ia_id in final_grading:
                if _ia_id == ia_id:
                    return generation_id - 1
            generation_id += 1

        raise RuntimeError


    @classmethod
    def getPopulation(cls, processus_id, generation_id, generations):
        population = set()
        for ia_file in (
            cls.getPath(processus_id, generations, generation_id).parent
            / ('population' if generation_id > 0 else 'initial_pop')
        ).iterdir():
            population.add(IAFactory.hydrate(cls.readJSON(ia_file)))
        return population


    @classmethod
    def getIa(cls, processus_id, ia_id):
        generations = cls.getProcessusParams(processus_id)['generations']
        generation_id = cls.getGenerationOf(processus_id, generations, ia_id)
        ia_file = cls.getPath(processus_id, generations, generation_id, ia_id)
        return IAFactory.hydrate(cls.readJSON(ia_file)), generation_id


    @classmethod
    def getBestIa(cls, processus_id, generation_id = None):
        generations = cls.getProcessusParams(processus_id)['generations']
        if generation_id is None:
            generation_id = generations if type(generations) is int else cls.getLastGradedGeneration(processus_id, generations)
        grading = cls.readJSON(cls.getPath(processus_id, generations, generation_id + 1, 'final_grading'))
        grading.sort(key=lambda c: c[0]['score'], reverse=True)
        ia_id = grading[0][1]
        ia_file = cls.getPath(processus_id, generations, generation_id, ia_id)
        return IAFactory.hydrate(cls.readJSON(ia_file)), generation_id


    @classmethod
    def processusExists(cls, processus_id):
        path = cls.getPath(processus_id)
        if not path.parent.exists():
            return False
        return True


    @classmethod
    def getData(cls, processus_id):
        generation_id = 1
        generations = cls.getProcessusParams(processus_id)['generations']
        data = []

        while True:
            path = cls.getPath(processus_id, generations, generation_id, 'final_grading')
            if not path.exists():
                break
            final_grading = cls.readJSON(path)
            data.append((generation_id - 1, final_grading))
            generation_id += 1

        return data


    @classmethod
    def getProcessusState(cls, processus_id):
        '''
        for state.event_name in (
            PROCESSUS.START,
            CREATION.START,
            CREATION.DONE,
            GENERATION.START,
            GRADING.START,
            GRADING.PROGRESS,
            GRADING.DONE,
            SELECTION.START,
            SELECTION.DONE,
            BREEDING.START,
            BREEDING.PROGRESS,
            BREEDING.DONE,
            GENERATION.DONE,
            PROCESSUS.DONE
        )
        '''

        state = ProcessusState()
        state.processus_id = processus_id
        state.__dict__.update(cls.getProcessusParams(processus_id))

        getPath = lambda generation_id, file_name = None: cls.getPath(
            processus_id, state.generations, generation_id, file_name
        )

        generation_id = cls.getLastGeneration(processus_id, state.generations)
        # If none generation folder exist
        if generation_id == -1:
            state.event_name = PROCESSUS.START
            return state
        state.generation_id = generation_id

        # Get event_name
        state.event_name = cls.readJSON(getPath(state.generation_id))['event_name']

        if state.event_name in (CREATION.DONE, BREEDING.DONE, GENERATION.DONE, PROCESSUS.DONE):
            state.population = cls.getPopulation(state.processus_id, state.generation_id, state.generations)
        else:
            state.population = cls.getPopulation(state.processus_id, state.generation_id - 1, state.generations)

        if state.event_name in (GRADING.PROGRESS):
            state.grading = cls.readGrading(getPath(state.generation_id, 'grading'))
        elif state.event_name in (GRADING.DONE, SELECTION.START):
            state.grading = cls.readJSON(getPath(state.generation_id, 'final_grading'))
        if state.grading is not None:
            indexed_pop = dict([(ia.id, ia) for ia in state.population])
            state.grading = [(score, indexed_pop[ia_id]) for (score, ia_id) in state.grading]

        if state.event_name in (SELECTION.DONE, BREEDING.START, BREEDING.PROGRESS):
            state .selection = cls.readJSON(getPath(state.generation_id , 'selection'))

        return state
```