//Turram Toussaint

// OOP Program

// Dr Griffin

import json

import os

from typing import Any, Dict, List, Optional, Union

# ---------- BASE CLASS ----------

class JsonDB:

def \_\_init\_\_(self, file\_path: str):

self.file\_path = file\_path

self.data = self.\_load\_data()

def \_load\_data(self) -> Union[List, Dict]:

if not os.path.exists(self.file\_path):

return []

with open(self.file\_path, 'r') as f:

return json.load(f)

def \_save\_data(self) -> None:

with open(self.file\_path, 'w') as f:

json.dump(self.data, f, indent=4)

def read(self, filters: Optional[Dict[str, Any]] = None) -> List[Dict]:

results = []

for record in self.data:

if not filters or all(record.get(k) == v for k, v in filters.items()):

results.append(record)

return results

def create(self, record: Dict[str, Any]) -> int:

record['id'] = self.\_generate\_id()

self.data.append(record)

self.\_save\_data()

return record['id']

def update(self, record\_id: int, updates: Dict[str, Any]) -> bool:

for record in self.data:

if record.get('id') == record\_id:

record.update(updates)

self.\_save\_data()

return True

return False

def delete(self, record\_id: int) -> bool:

for i, record in enumerate(self.data):

if record.get('id') == record\_id:

del self.data[i]

self.\_save\_data()

return True

return False

def \_generate\_id(self) -> int:

ids = [record.get('id', 0) for record in self.data if isinstance(record, dict)]

return max(ids, default=0) + 1

# ---------- SUBCLASS: PeopleDB ----------

class PeopleDB(JsonDB):

def create\_person(self, person: Dict[str, Any]) -> int:

required = ['first\_name', 'last\_name']

if not all(k in person for k in required):

raise ValueError("Missing required fields.")

return self.create(person)

def find\_by\_name(self, first\_name: Optional[str] = None, last\_name: Optional[str] = None) -> List[Dict]:

return [

person for person in self.data

if (first\_name is None or person.get('first\_name') == first\_name) and

(last\_name is None or person.get('last\_name') == last\_name)

]

# ---------- SUBCLASS: MeteoriteDB ----------

class MeteoriteDB(JsonDB):

def find\_by\_year\_range(self, start\_year: int, end\_year: int) -> List[Dict]:

return [

m for m in self.data

if start\_year <= int(m.get('year', 0)) <= end\_year

]

def find\_heaviest\_meteorites(self, count: int = 5) -> List[Dict]:

return sorted(self.data, key=lambda m: float(m.get('mass', 0)), reverse=True)[:count]

# ---------- SUBCLASS: EarthquakeDB ----------

class EarthquakeDB(JsonDB):

def filter\_by\_magnitude(self, min\_magnitude: float) -> List[Dict]:

return [e for e in self.data if float(e.get('magnitude', 0)) >= min\_magnitude]

def filter\_by\_place(self, place: str) -> List[Dict]:

return [e for e in self.data if place.lower() in e.get('place', '').lower()]

# ---------- MAIN USAGE ----------

def main():

# Sample usage for PeopleDB

people\_db = PeopleDB("people.json")

people\_db.create\_person({"first\_name": "Teresa", "last\_name": "Chen"})

found = people\_db.find\_by\_name("Teresa")

print("Found person:", found)

# Sample usage for MeteoriteDB

meteor\_db = MeteoriteDB("meteorites.json")

print("Heaviest meteorites:", meteor\_db.find\_heaviest\_meteorites())

# Sample usage for EarthquakeDB

eq\_db = EarthquakeDB("earthquakes.json")

print("Earthquakes above magnitude 6:", eq\_db.filter\_by\_magnitude(6.0))

if \_\_name\_\_ == "\_\_main\_\_":

main()