CSIT121 Assignment 3 – Pokémon Pokedex System

Duc Toan Luong - 8933613

I. Introduction

This assignment implements a mini Pokémon Pokedex application using object-oriented programming in Python.

Goal: Extend A2 solution Pokédex to (a) load/save with JSON, (b) validate new “advanced info” with regex, (c) produce 3 charts with Matplotlib, (d) add 1 design pattern, (e) include unit tests + coverage, (f) deliver a report with UML + screenshots.

The main goal is to apply the principles of Object-Oriented Analysis (OOA), Object-Oriented Design (OOD), and Object-Oriented Programming (OOP) - focusing on object (de)serialization, regular expression, and data visualization using Matplotlib.

II. Object-Oriented Analysis (OOA)

The Pokémon Pokédex program was designed following object-oriented principles such as abstraction, inheritance, encapsulation, and composition. The system separates functionality into logical classes to ensure modularity, reusability, and clarity. The BasePokemon class serves as the abstract foundation that defines all shared Pokémon attributes and behaviours, while subclasses such as FireType and GrassType specialize it by adding type-specific information and methods. Concrete Pokémon classes like Charmander and Bulbasaur further extend these type classes to implement unique display behaviour.

The Stats class is composed within each Pokémon, representing a clear “has-a” relationship that encapsulates numerical attributes in a structured and maintainable way. The Pokedex class operates as a singleton manager for all Pokémon records, responsible for storage, file

input/output, searching, updating, and reporting operations. This ensures centralized data control and prevents inconsistencies between multiple instances.

To maintain robustness, the Validator class enforces strict data validation rules using regular expressions, ensuring consistent and error-free input formatting. The Visualizer class complements this by transforming numerical data into graphical forms, enhancing readability through bar, line, and pie charts. Exception handling is managed by the custom PokemonNotFoundError, which allows the program to handle invalid lookups gracefully without interruption.

Overall, the system demonstrates a cohesive and extensible design that balances clarity, maintainability, and functionality. Each class has a well-defined role, and their interactions follow clear hierarchies and dependencies, resulting in an organized, modular, and user-friendly application.

1. Assumptions:

- **Input Data Format**

- The program assumes all Pokémon data loaded from files is **well-structured and valid** according to the expected format.
- .txt files follow the exact key-value structure shown in the sample (e.g., Name: Charmander, Height: 0.6 m).
- .json files follow the format generated by the program's own save_json() method.

- **Pokémon Types**

- Only two Pokémon types are supported: **Fire** and **Grass**.
- Any Pokémon whose type is not Fire or Grass will raise an exception or be ignored.

- **Known and Generic Pokémon**

- The system contains **four predefined Pokémon**: Charmander, Vulpix, Bulbasaur, and Oddish.
- Any newly added Pokémon is treated as a **GenericFirePokemon** or **GenericGrassPokemon**, depending on its type.

- **National Number Format**

- The national number must follow the **strict format**:
  No. XXXX (where XXXX is exactly four digits, including leading zeros).
- No variations (e.g., no.25, No.25, or 25) are accepted.

- **Unit Validation**

- **Height** must be entered as a decimal number followed by m (e.g., 1.70 m).

- **Weight** must be entered as a decimal number followed by kg (e.g., 6.90 kg).
- Both must include a **space** before the unit.

    - **User Input Validation**
- When users add or update Pokémon data, all input is validated by the Validator class.
- Invalid input raises a ValueError and prompts the user to re-enter data.

    - **File Save Behaviour**
- Saving always overwrites the **original file** (the same .txt or .json file that was loaded).
- If no file was loaded initially, the user must specify a save path manually.

    - **Singleton Enforcement**
- The Pokedex class is a **singleton**. Only one instance can exist during runtime.
- Any attempt to instantiate another Pokedex object will raise an exception.

    - **Stat Values**
- All Pokémon stats (hp, attack, defense, sp_atk, sp_def, speed) must be **non-negative integers**.
- Invalid stat fields or negative values raise an exception.

    - **Visualization**

- Visualizations are generated using **Matplotlib**.
- Charts assume that valid numeric data is available.
- The user may choose to display the chart on screen or save it as an image file (e.g., .png).

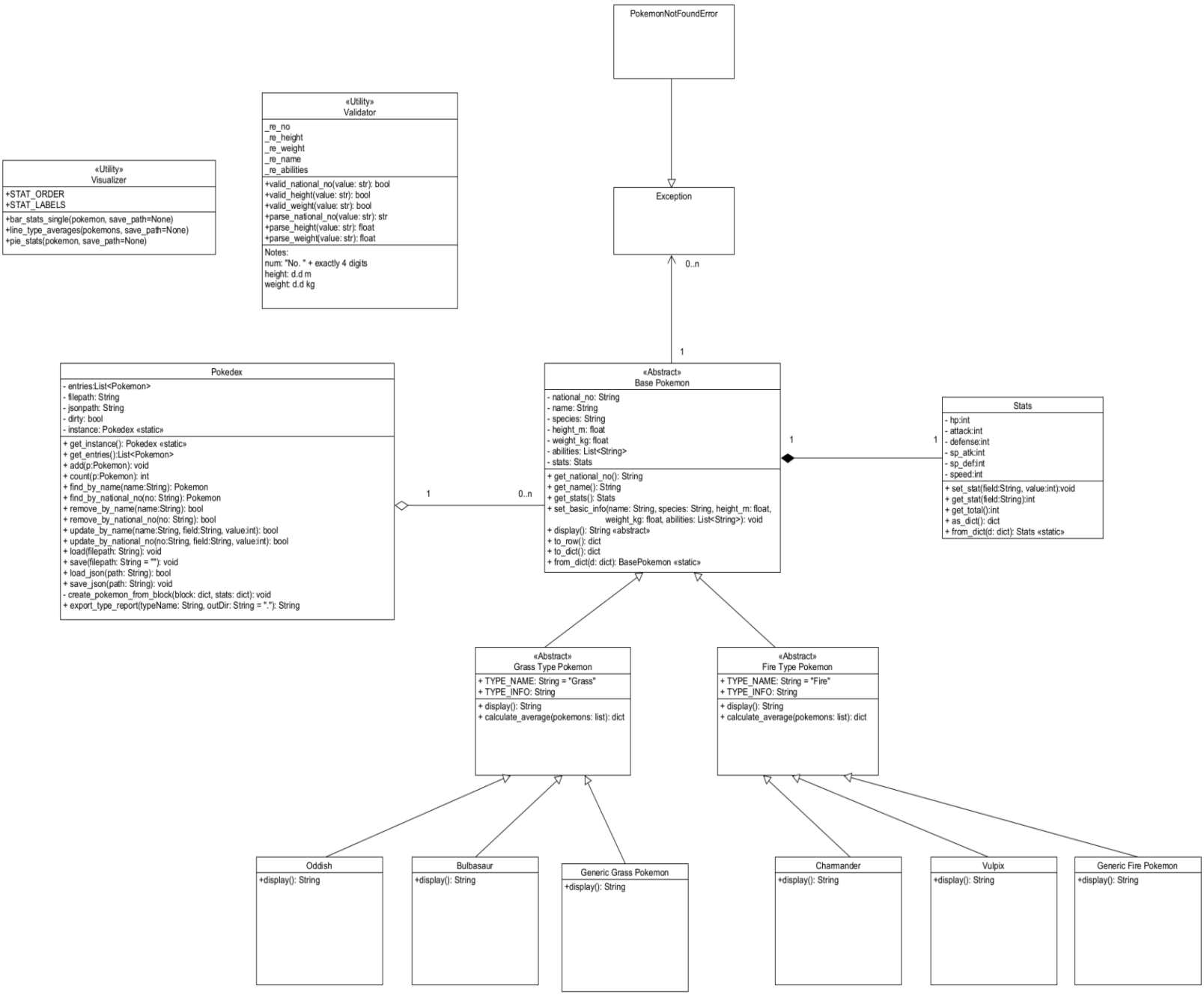
    - **Report Export**
- Type reports are generated as plain text files (e.g., fire.txt, grass.txt).
- Each report includes type information, Pokémon listings, and average stats.

    - **Data Integrity**
- The system assumes the user will not manually alter the text or JSON files while the program is running.
- No concurrent file access or data corruption handling is implemented.

    - **Environment**

- The program runs in a **standard Python 3 environment** with the following libraries available:
    - matplotlib, numpy, json, os, re, and abc.

III. Object-Oriented Design (OOD)



- BasePokemon (abstract): defines all common Pokémon attributes such as name, national number, species, height, weight, abilities, and stats. It provides shared methods including display(), to_row(), to_dict(), and from_dict() that are used by all subclasses.

- Stats: encapsulates the six main numerical attributes (HP, Attack, Defense, Special Attack, Special Defense, and Speed) with getter and setter methods, total calculation, and dictionary conversion for serialization.

- Type Classes (FireType, GrassType): inherit from BasePokemon and include type-specific constants (TYPE_NAME, TYPE_INFO) and shared functionality such as calculate_average() for type-based statistical analysis.

- Concrete Classes (Bulbasaur, Oddish, Vulpix, Charmander, GenericFirePokemon, GenericGrassPokemon): extend their respective type classes and override the display() method to print detailed Pokémon information.

- Pokedex (singleton): manages a single instance of the Pokémon collection, supporting adding, removing, searching, updating, and exporting reports. It also handles both text and JSON file input/output and maintains a “dirty” flag to track unsaved changes.

- Validator (utility): provides static methods using regular expressions to validate and parse user input for fields such as national number, height, weight, names, and abilities, ensuring all data follows strict formatting rules.

- Visualizer (utility): generates bar, line, and pie charts using matplotlib and numpy to visually represent Pokémon statistics and type comparisons.

- PokemonNotFoundError: a self-defined exception class inherited from Exception, used to handle invalid Pokémon searches or updates gracefully.

IV. Implementation (OOP)

User operations (must work via menu):

- Load data (prefer JSON; else import from Pokemon.txt).
- List/search Pokémon (by name, national number, type).
- Create/remove Pokémon.
- Update stats (existing A2).
- Modify advanced info (new): national number, height, weight with strict formats.
- Save/export: to both text and JSON on exit/save.
- Visualize results (3 chart styles).

Building on Assignment 2, the new version introduces functions extensions and one design enhancement:

2. Object Serialization / Deserialization (JSON):

The Pokédex must now be able to save and load Pokémon data using JSON format. If the JSON file exists, the program loads Pokémon objects from it. If not, it imports data from Pokemon.txt and creates Pokémon objects, which will then be exported to JSON upon exit.Serialization is handled through each Pokémon object's to_dict() method and the Pokédex's save_json() method.

3. Regular Expression Validation:

The program allows users to modify advanced Pokémon details — national number, height, and weight - with strict case-sensitive input validation:

- National number: must match the format No. 0000
    - ('No.' prefix, space, and exactly four digits)
- Height: must match the format x.y m
    - (one digit before and after the decimal, followed by a space and lowercase 'm')
- Weight: must match the format x.y kg
    - (one digit before and after the decimal, followed by a space and lowercase 'kg')

Invalid inputs trigger error messages and require re-entry until the pattern is satisfied.

4. Visualization of Statistical Analysis:

Users can now request visual summaries of Pokémon data through Matplotlib. At least three different visualization styles must be supported:

- Bar Chart: individual Pokémon's six battle stats.
- Pie Chart: the same stats represented as proportional segments.
- Line Chart: average stats comparison between Fire- and Grass-type Pokémon.

Each chart is automatically saved to a /figs directory for documentation.

5. Design Pattern Application:

The **Singleton** pattern is applied to the Pokedex class to ensure only one instance of the Pokédex exists throughout the program.

This avoids data inconsistency between multiple instances and ensures all modifications and exports refer to a single shared dataset.

It reflects real-world design, where there is only one official Pokédex.

```python
# ----------------------
# Pokedex
# ----------------------

class Pokedex:
    """Singleton class that manages all Pokémon entries."""

    __instance = None     # class-level variable

    def __init__(self):
        """Private constructor — only one instance allowed."""
        if Pokedex.__instance is not None:
            raise Exception("Pokedex is a Singleton class! Use get_instance().")
        self.entries = []
        self.text_path = ""
        self.json_path = ""
        self.dirty = False
        Pokedex.__instance = self

    @classmethod
    def get_instance(cls):
        """Get or create the single Pokedex instance."""
        if cls.__instance is None:
            cls.__instance = Pokedex()
        return cls.__instance
```

V. Testing

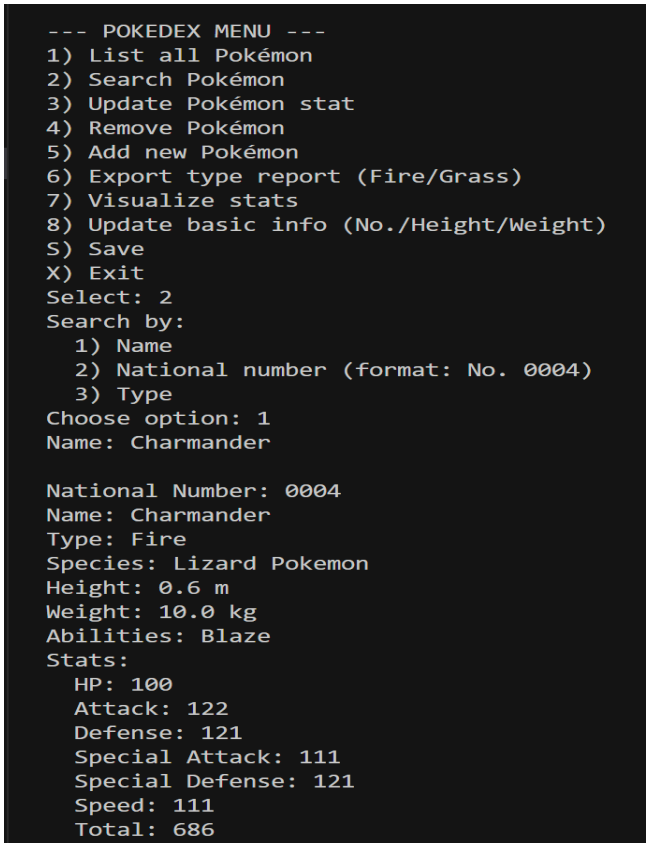
Unit tests were created using Python's unittest module for object serialization - deserialization methods.

```python
class TestSerialization(unittest.TestCase):
    def setUp(self):
        self.dex = Pokedex.get_instance()
        self.dex.entries = []
        self.dex.json_path = ""
        self.dex.text_path = ""
        self.dex.dirty = False

    # Sample pokemon for testing
        self.bulbasaur = Bulbasaur(
            national_no="0001",
            name="Bulbasaur",
            species="Seed Pokémon",
            height_m=0.7,
            weight_kg=6.9,
            abilities=["Overgrow"],
            stats=Stats(45, 49, 49, 65, 65, 45)
        )

        self.charmander = Charmander(
            national_no="0004",
            name="Charmander",
            species="Lizard Pokémon",
            height_m=0.6,
            weight_kg=8.5,
            abilities=["Blaze"],
            stats=Stats(39, 52, 43, 60, 50, 65)
        )
```

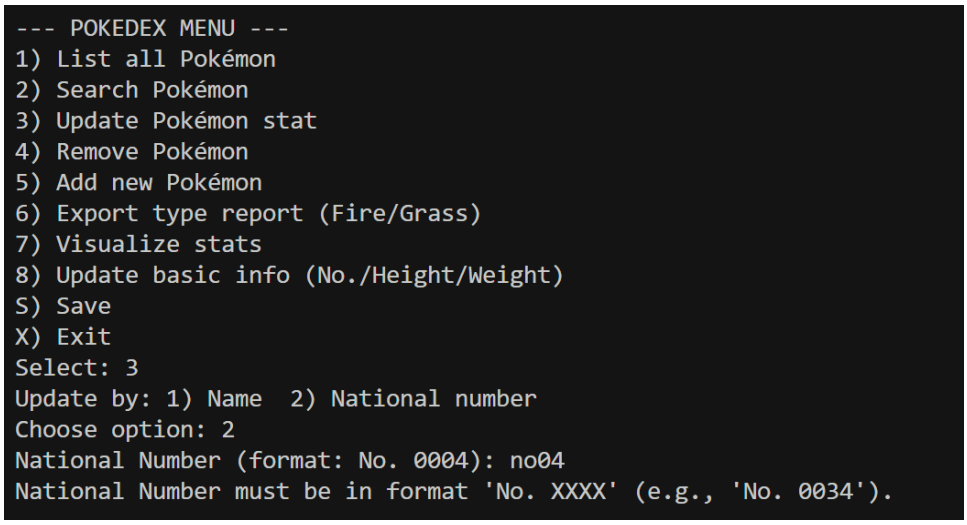
```python
    def test_to_from_dict(self):
        """Test BasePokemon.to_dict() and from_dict() reconstruct the same data."""
        data = self.bulbasaur.to_dict()
        restored = BasePokemon.from_dict(data)
        self.assertEqual(data["name"], restored.to_dict()["name"])
        self.assertEqual(data["type"], restored.to_dict()["type"])
        self.assertEqual(data["stats"], restored.to_dict()["stats"])
        self.assertAlmostEqual(data["height_m"], restored.get_height(), places=2)
        self.assertAlmostEqual(data["weight_kg"], restored.get_weight(), places=2)

    def test_json_save_load(self):
        """Test saving to and loading from JSON file."""
        self.dex.add(self.bulbasaur)
        self.dex.add(self.charmander)

        with tempfile.NamedTemporaryFile(delete=False, suffix=".json") as tmpfile:
            json_path = tmpfile.name

        try:
            self.dex.save_json(json_path)
            # Clear current entries
            self.dex.entries = []
            self.dex.load_json(json_path)

            self.assertEqual(self.dex.count(), 2)
            names = [p.get_name() for p in self.dex.get_entries()]
            self.assertIn("Bulbasaur", names)
            self.assertIn("Charmander", names)
        finally:
            os.remove(json_path)

if __name__ == "__main__":
    unittest.main()
```

VI. Screenshot

1. Collecting user request



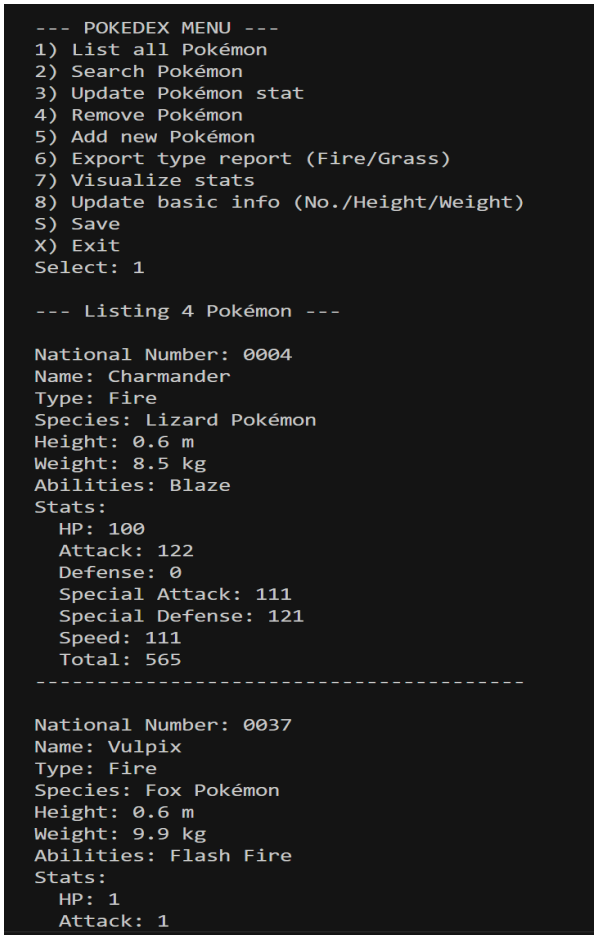
2. Handling invalid request



3. Collecting new Pokemon basic info (national_no, height, weight)

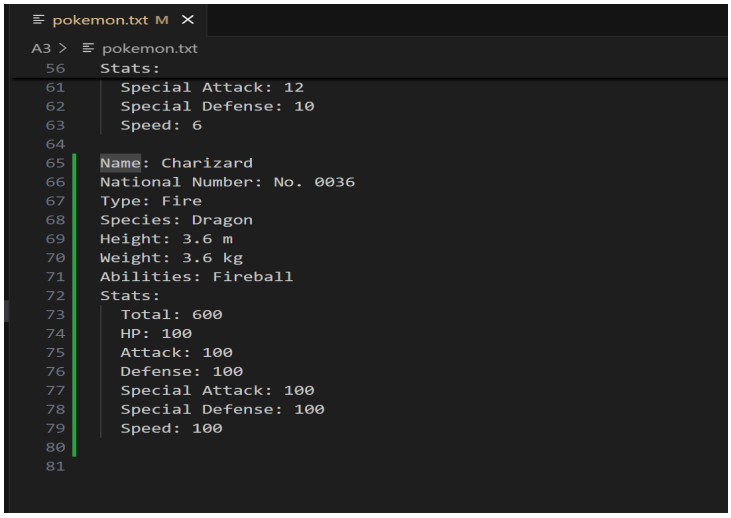
```
--- POKEDEX MENU ---
1) List all Pokémon
2) Search Pokémon
3) Update Pokémon stat
4) Remove Pokémon
5) Add new Pokémon
6) Export type report (Fire/Grass)
7) Visualize stats
8) Update basic info (No./Height/Weight)
S) Save
X) Exit
Select: 8
Update by: 1) Name  2) National number
Choose option: 1
Name: Charmander

Current info for Charmander:
  National Number: 0004
  Height: 0.6 m
  Weight: 8.5 kg
Field (national_no / height_m / weight_kg): weight_kg
New value: 10 kg
Pokédex data saved to JSON file 'pokemon.json'
Saved back to original file.
Updated and saved.
```

4. Displaying imported Pokemon



5. Update Pokemon.txt after creating new/modifying existing

```
--- POKEDEX MENU ---
1) List all Pokémon
2) Search Pokémon
3) Update Pokémon stat
4) Remove Pokémon
5) Add new Pokémon
6) Export type report (Fire/Grass)
7) Visualize stats
8) Update basic info (No./Height/Weight)
S) Save
X) Exit
Select: 5
Add new Pokémon
Type (Fire/Grass): Fire
National Number (format: No. 0025): No. 0036
Name: Charizard
Species: Dragon
Height (e.g., 0.6 m): 3.6 m
Weight (e.g., 8.5 kg): 3.6 kg
Abilities (separate with ';'): Fireball
HP: 100
Attack: 100
Defense: 100
Special Attack: 100
Special Defense: 100
Speed: 100
Pokédex saved to 'pokemon.txt'
Saved back to original file.
Fire Pokémon added and saved.
```



6. Export a JSON file

```
--- POKEDEX MENU ---
1) List all Pokémon
2) Search Pokémon
3) Update Pokémon stat
4) Remove Pokémon
5) Add new Pokémon
6) Export type report (Fire/Grass)
7) Visualize stats
8) Update basic info (No./Height/Weight)
S) Save
X) Exit
Select: 5
Add new Pokémon
Type (Fire/Grass): Grass
National Number (format: No. 0025): No. 0020
Name: Ivysaur
Species: Frog
Height (e.g., 0.6 m): 3.6 m
Weight (e.g., 8.5 kg): 3.6 kg
Abilities (separate with ';'): Watergun
HP: 100
Attack: 100
Defense: 100
Special Attack: 100
Special Defense: 100
Speed: 100
Pokédex data saved to JSON file 'pokemon.json'
Saved back to original file.
Grass Pokémon added and saved.
```

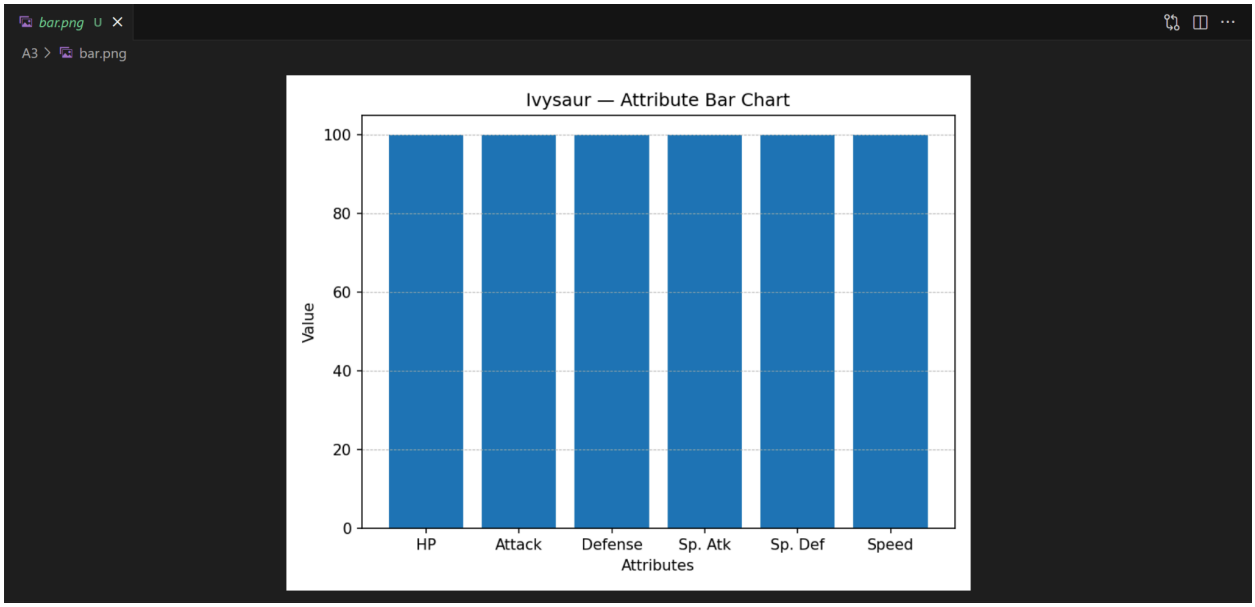
{} pokemon.json M

A3 > {} pokemon.json > {} 0 > # weight_kg

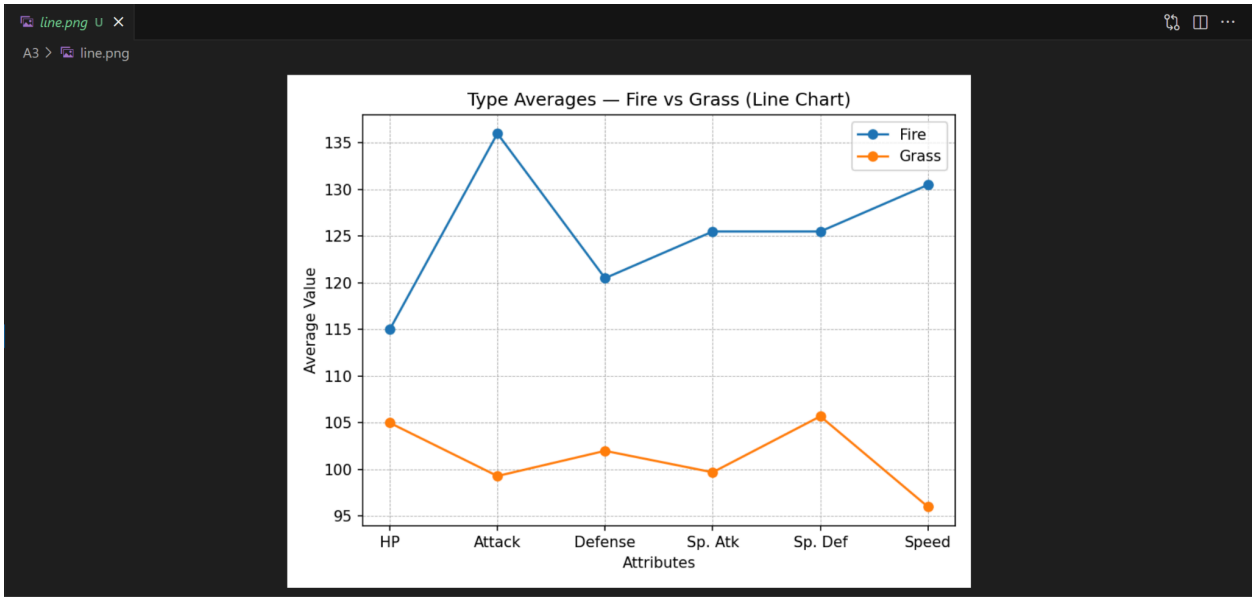
```
66          {
78              "stats": {
                    "sp_def": 95,
84                  "speed": 70,
85                  "total": 515
86              }
87          },
88          {
89              "class": "GenericGrassPokemon",
90              "type": "Grass",
91              "national_no": "0020",
92              "name": "Ivysaur",
93              "species": "Frog",
94              "height_m": 3.6,
95              "weight_kg": 3.6,
96              "abilities": [
97                  "Watergun"
98              ],
99              "stats": {
100                 "hp": 100,
101                 "attack": 100,
102                 "defense": 100,
103                 "sp_atk": 100,
104                 "sp_def": 100,
105                 "speed": 100,
106                 "total": 600
107             }
108         }
109     ]
```

7. Creating figures

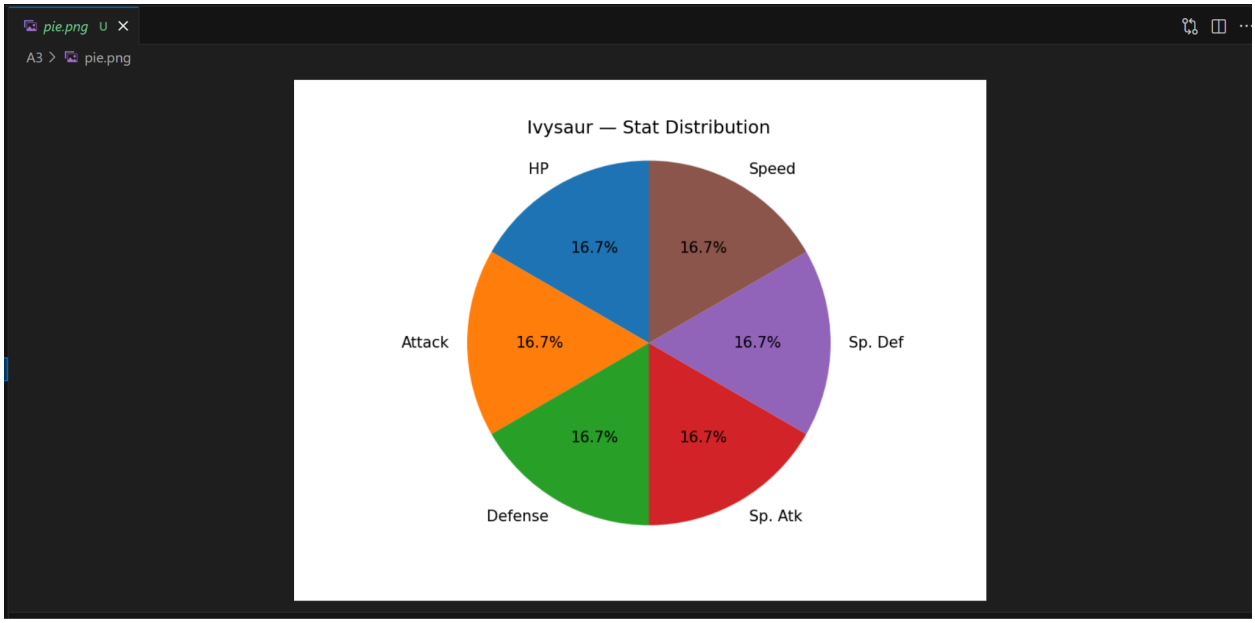
```
--- POKEDEX MENU ---
1) List all Pokémon
2) Search Pokémon
3) Update Pokémon stat
4) Remove Pokémon
5) Add new Pokémon
6) Export type report (Fire/Grass)
7) Visualize stats
8) Update basic info (No./Height/Weight)
S) Save
X) Exit
Select: 7
Choose visualization type:
  1) Bar chart (single Pokémon’s stats)
  2) Line chart (Fire vs Grass averages)
  3) Pie chart (single Pokémon’s stat distribution)
Select option: 1
Save chart to file? (y/n): y
Filename (e.g., chart.png): bar.png
Enter Pokémon name: Ivysaur
Saved bar.png
```



```
--- POKEDEX MENU ---
1) List all Pokémon
2) Search Pokémon
3) Update Pokémon stat
4) Remove Pokémon
5) Add new Pokémon
6) Export type report (Fire/Grass)
7) Visualize stats
8) Update basic info (No./Height/Weight)
S) Save
X) Exit
Select: 7
Choose visualization type:
  1) Bar chart (single Pokémon’s stats)
  2) Line chart (Fire vs Grass averages)
  3) Pie chart (single Pokémon’s stat distribution)
Select option: 2
Save chart to file? (y/n): y
Filename (e.g., chart.png): line.png
Saved line.png
```



```
--- POKEDEX MENU ---
1) List all Pokémon
2) Search Pokémon
3) Update Pokémon stat
4) Remove Pokémon
5) Add new Pokémon
6) Export type report (Fire/Grass)
7) Visualize stats
8) Update basic info (No./Height/Weight)
S) Save
X) Exit
Select: 7
Choose visualization type:
  1) Bar chart (single Pokémon’s stats)
  2) Line chart (Fire vs Grass averages)
  3) Pie chart (single Pokémon’s stat distribution)
Select option: 3
Save chart to file? (y/n): y
Filename (e.g., chart.png): pie.png
Enter Pokémon name: Ivysaur
Saved pie.png
```



8. Unit test coverage

```
                                                                                coverage run -m test_A3
Pokédex data saved to JSON file 'C:\Users\test\AppData\Local\Temp\tmp8qe1xg0a.json'
Loaded 2 Pokémon from JSON 'C:\Users\test\AppData\Local\Temp\tmp8qe1xg0a.json'
..
----------------------------------------------------------------------
Ran 2 tests in 0.006s

OK
PS C:\OneDrive - University of Wollongong\CSIT121 (S225) Object Oriented Design and Programming\Assignments\A3>
```

```
PS C:\OneDrive - University of Wollongong\CSIT121 (S225) Object Oriented Design and Programming\Assignments\A3> coverage report
Name         Stmts   Miss  Cover
------------------------------
A3.py          711    544    23%
test_A3.py      38      0   100%
------------------------------
TOTAL          749    544    27%
PS C:\OneDrive - University of Wollongong\CSIT121 (S225) Object Oriented Design and Programming\Assignments\A3>
```