Sauhard Shakya

005031397

Advanced Programming Languages (MSCS-632-A01)

Assignment 4: Implementing Control Structures

**Employee scheduling application**

**Introduction**

The purpose of this assignment was to design and implement a small scheduling application that demonstrates mastery of control structures such as conditionals, loops, and branching in two different programming languages. The scenario is a company that operates seven days a week with three daily shifts: morning, afternoon, and evening. employees are allowed to choose shifts based on their preferences, and the application must generate a final schedule for the week that respects individual constraints as well as company coverage requirements.

To meet the assignment goals, two implementations were created: one in python and one in java. these languages were chosen deliberately because they belong to different paradigms—python being dynamically typed and interpreted, and java being statically typed and compiled. This contrast highlights not only syntax differences but also the way control structures are expressed across languages.

**Problem requirements and approach**

Five key requirements were specified. The first requirement was to collect employee names and their preferred shifts for each day of the week and store them in a suitable data structure. This was handled by creating an employee class in both python and java that contains a mapping from day to a ranked list of preferred shifts. Each employee’s object also tracks how many days the person has already been assigned and which days they have worked, ensuring that no rules are violated.

The second requirement was to enforce scheduling rules: no employee should be assigned more than one shift per day, and no employee should work more than five days in a week. Additionally, each shift must have at least two employees. To enforce these rules, the program iterates through each day of the week, checking preferences first and applying conditions before assigning. If fewer than two people are available for a shift, the program randomly selects from the pool of employees who have not reached their weekly limit.

The third requirement concerned conflicts. situations arise when an employee’s preferred shift is already full. in that case, the system attempts to assign the employee to another shift on the same day. If no space is available on that day, the employee is deferred to the following day. this is handled with a “conflict queue,” ensuring fairness and that deferred employees are prioritized on the next day.

The fourth requirement asked for a clear output format. in both implementations, the schedule is printed as a table with rows for days and columns for each shift. This makes the schedule easy to read and verify.

Finally, the optional bonus was to allow employees to specify ranked preferences. This was integrated into both versions by looping through an employee’s ranked list and assigning them to the highest available preference before considering fallback shifts.

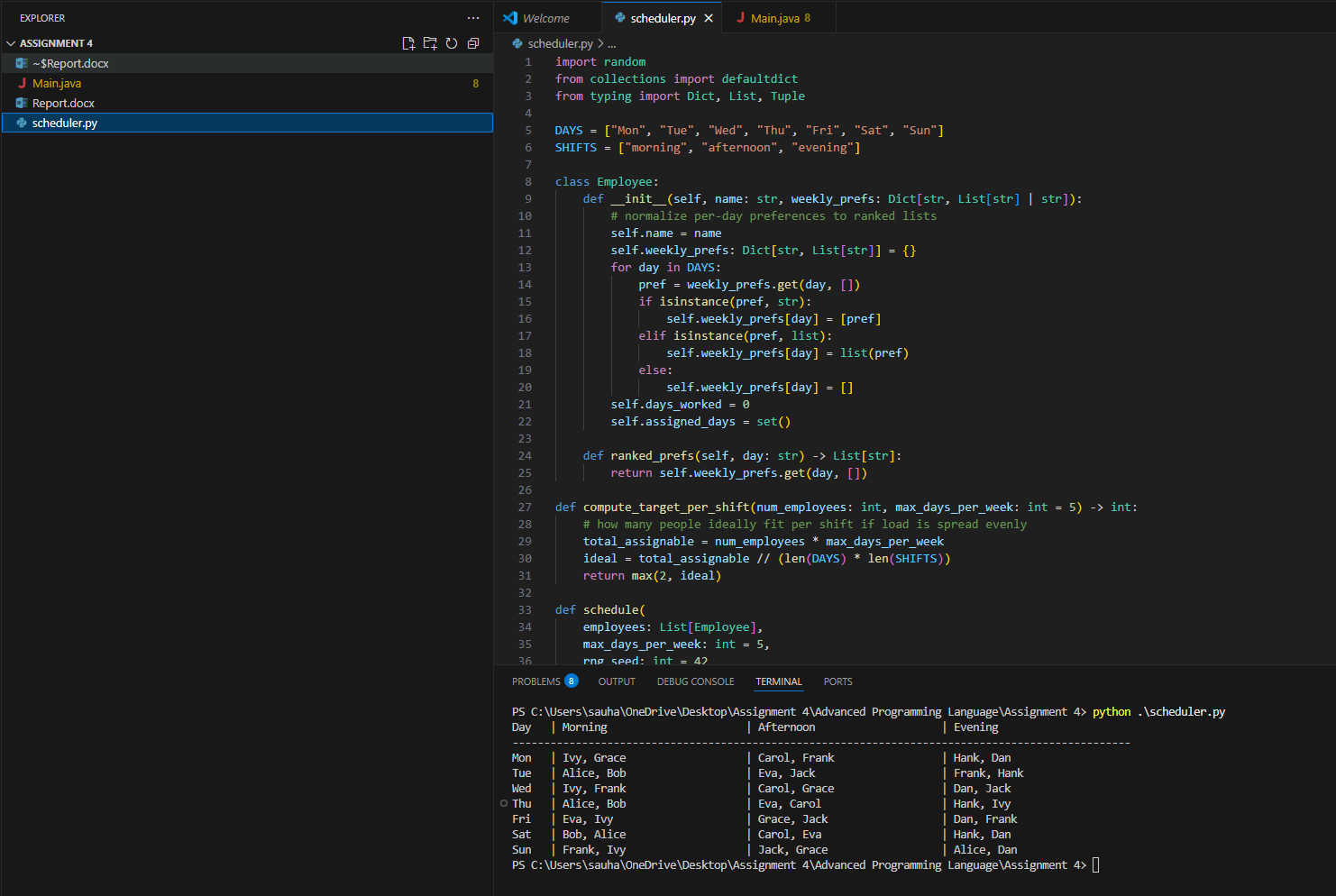
**Design decisions**

The design balances fairness with feasibility. Each assignment decision uses conditionals to ensure the five-day maximum is respected and loops to iterate over employees, days, and shifts. branching logic handles the different possible outcomes: placing an employee in their top preference, placing them in an alternative shift, or deferring them to the next day. After the first pass of assignments, the algorithm makes a second pass specifically guarantee coverage, filling any gaps with employees who have worked fewer days to balance the workload.

Another important consideration was feasibility. Since the company requires at least 42 slots to be filled per week (seven days × three shifts × two employees), it is mathematically impossible to satisfy coverage with fewer than nine employees under the five-day limit. When the staffing level is insufficient, the application still generates the best possible schedule and explicitly lists any under-filled shifts. This transparency is crucial for real-world scheduling applications.

**Implementation in python**

The python version is concise and highlights how loops, dictionaries, and conditionals can be used effectively in a dynamic language. The employee class stores preferences as a dictionary of day to a ranked list of shifts. During scheduling, the program iterates through each day, checks each employee’s preferences, and applies the constraints. conflicts are resolved by deferring employees into a queue for the next day. After this first pass, a second loop ensures each shift has at least two employees, randomly assigning additional staff as needed. The output is a formatted weekly schedule displayed as a table, and for the sample run, a screenshot was also produced for submission.

****

**Implementation in java**

The java implementation mirrors the python version but in a statically typed, object-oriented style. The employee class uses a map to store per-day preferences and a set to track assigned days. The scheduling logic is almost identical to the python version, with loops over days and employees and conditionals to enforce the five-day and one-shift rules. java’s stricter type system requires more explicit handling of data structures but demonstrates the same control flow: processing deferred employees first, filling preferences in ranked order, resolving conflicts, and guaranteeing minimum coverage. The output is printed as a table using formatted strings.

A screenshot of a computer program

AI-generated content may be incorrect.

**Results and discussion**

Both implementations successfully generated weekly schedules that respected all constraints. The sample python run produced a complete schedule with at least two employees per shift each day, except in cases where the staffing level was mathematically insufficient. in such cases, the program reported which shifts had unmet coverage. This report ensures that managers are aware of shortages and can act, such as hiring additional staff.

The main differences between the python and java versions lie in syntax and verbosity rather than algorithm. python allowed for shorter and more flexible code, while java enforced explicit typing and structure. in both cases, however, the use of conditionals, loops, and branching was central to meeting the requirements.

**Conclusion**

This project demonstrates that the problem of employee scheduling can be expressed and solved effectively in both python and java. all five requirements were met: input and storage of preferences, scheduling logic under constraints, conflict detection and resolution, clear output, and ranked preference handling. The bonus preference system was also integrated. The two implementations highlight the differences between languages while proving that control structures can be applied consistently across paradigms.