TIME TABLE SCHEDULING

**Members**:

|  |  |
| --- | --- |
| Name | Roll |
| Anjori Sarabhai | 22csu197 |
| Rudra Rishi | 22csu |
| Shubh Pundir | 22csu165 |

**Definition of Done:**

In this project, you'll design a system to efficiently schedule exams for multiple courses and students, considering various constraints and preferences such as:

1. One subject can have only 1 class in a day.

2. Lab classes can have 2 slots.

3. No Faculty should be able to take more than one class at a particular time.

4. Any clash intra or inter section will not be allowed.

5. All courses should fill up their lecture and laboratory hours in the week

**UML Diagram of base classes:**

A screenshot of a computer

Description automatically generated

## Connectivity of classes is seen by passing common references as parameters

**Data Structure used:**

**Hash table:** Hash tables, also known as hash maps, are a fundamental data structure in computer science. Here are some brief notes on hash tables:

1. **Definition**:

- A hash table is a data structure that implements an associative array abstract data type, a structure that can map keys to values.

- It uses a hash function to compute an index into an array of buckets or slots, from which the desired value can be found.

2. **Components**:

- **Key**: The input data that is mapped to a unique index in the hash table.

- **Value**: The data associated with a key.

- **Hash Function**: A function that takes a key as input and produces a hash code, which is used to determine the index where the value will be stored.

3. **Hash Function**:

- Should be deterministic: For the same key, the hash function should always produce the same hash code.

- Should distribute keys uniformly: To minimize collisions (two different keys hashing to the same index).

- May result in collisions: Different keys may produce the same hash code. Collisions need to be resolved using collision resolution techniques.

4. **Collision Resolution**:

- **Open Addressing**: In case of a collision, search for the next available slot in the hash table.

- **Closed Addressing (Chaining)**: Each slot in the hash table contains a linked list of elements that hash to the same index.

5. **Load Factor**:

- The load factor is a measure of how full the hash table is. It is calculated as the ratio of the number of elements to the number of slots.

- Load factor = Number of elements / Number of slots

6. **Common Operations**:

- **Insertion**: Place a key-value pair in the hash table.

- **Deletion**: Remove a key-value pair from the hash table.

- **Lookup (Search)**: Find the value associated with a given key.

7. **Time Complexity**:

- On average, the time complexity for insertion, deletion, and lookup operations is O(1) (constant time) if the hash function is well-designed and the load factor is kept low.

8. **Applications**:

- Hash tables are widely used in various applications, including symbol tables in compilers, databases, caches, and as the underlying data structure for Python dictionaries.

9. **Examples**:

- In Python, dictionaries are implemented using a form of hash tables.

- Java has the `HashMap` class, and C++ has `std::unordered\_map`, both of which are implementations of hash tables.

**Base classes**

`**Course**` **Class**:

- Represents a course with attributes like course code, course name, number of lectures, and number of labs.

- Instances of this class include `course1`, `course2`, `course3`, and `course4` for basic courses, and `major1`, `major2`, `major3`, `major4`, and `recess` for major courses and a recess period.

### `**Faculty**` **Class**:

- Represents a faculty member with attributes like faculty ID, faculty name, and courses taught.

- Instances of this class include `anuradha`, `yogita`, `pooja`, etc., each associated with the courses they teach.

### `**Section**` **Class**:

- Represents a section with attributes like section name, a dictionary of courses, and a timetable.

- Instances of this class include `section\_A`, `section\_B`, `section\_C`, `section\_D`, and `section\_E`. Each section has its own list of courses and a timetable.

### **Operations and Relationships**:

- Courses are associated with faculties, and faculties are associated with sections.

- Sections have a timetable that can be used to schedule courses.

- The script creates instances of these classes and populates them with data to model a simplified academic scenario.

**### `Course` Class:**

The `Course` class is designed to represent an academic course and its attributes. Here's a breakdown of its key components:

- **Attributes**:

- `course\_code`: A unique identifier for the course.

- `course\_name`: The name or title of the course.

- `course\_lectures`: The number of lecture hours per week for the course.

- `course\_labs`: The number of lab hours per week for the course.

- **Instances**:

- `course1`, `course2`, `course3`, `course4`: Instances representing basic courses.

- `major1`, `major2`, `major3`, `major4`: Instances representing major courses.

- `recess`: An instance representing a recess period.

**### `Faculty` Class:**

The `Faculty` class represents a faculty member and the courses they teach. Key aspects of this class include:

- **Attributes**:

- `faculty\_id`: A unique identifier for the faculty member.

- `faculty\_name`: The name of the faculty member.

- `courses\_taught`: A list of courses that the faculty member teaches.

- **Instances**:

- Instances such as `anuradha`, `yogita`, `pooja`, etc., each associated with the courses they teach.

- **Timetable**:

- The class contains a `timetable` attribute, which is a dictionary representing the faculty member's schedule for each day of the week.

**### `Section` Class:**

The `Section` class models an academic section, providing structure for managing courses and creating a timetable. Key features include:

- **Attributes**:

- `section\_name`: A unique identifier for the section.

- `courses`: A dictionary where courses are mapped to the faculty member teaching them.

- `timetable`: A dictionary representing the section's timetable for each day of the week.

- `reference`: A 2D list used for reference in building the timetable.

- **Methods**:

- `add\_course(course, faculty)`: Adds a course to the section, associating it with the respective faculty member.

- `section\_info()`: Prints information about the section, including course details.

- **Instances**:

- Instances like `section\_A`, `section\_B`, `section\_C`, `section\_D`, and `section\_E` represent different sections.

**ALGORITHM IMPLEMENTED:**

1. Default course labs fill up of sections:
2. Major course labs fill up of sections:
3. Default course lecture fill up of sections:
4. Major course lecture fill up of sections:
5. **Default course lab fill up**:
6. Randomly allot day and slot from 1 to 5 for a 2 hour lab
7. If slot taken: recall the function randomly until labs/week aren’t over
8. Mark the slot of section’s time table with course code and faculty’s time table with section name
9. **Major course lab fill up**:
   1. Randomly allot day and slot from 1 to 5 for a 2 hour fill up
   2. If slot taken: recall the function randomly until labs/week aren’t over
   3. Mark the slot of section’s time table with course code and faculty’s time table with section name
10. **Default course lecture**:
    1. Randomly allot the number of days/week of the course from 5 working days of the week
    2. In such way, no course will be taken again in the same day
    3. Traverse through empty slots of the same
    4. Mark the slot of section’s time table with course code and faculty’s time table with section name
11. **Major course lecture**:
    1. Randomly allot the number of days/week of the course from 5 working days of the week
    2. In such way, no course will be taken again in the same day
    3. Traverse through empty slots of the same
    4. Mark the slot of section’s time table with course code and faculty’s time table with section name