**Course-End Project: MLB Digital Platform Enhancement**

Problem Statement:

The Major League Baseball (MLB) is a famous league with one of the highest

viewership. They are planning to update their digital platform for faster load times

and superior user experience. As part of the development team, you have to support

the backend development. You are required to create modules to manage player

statistics, match schedules, ticket bookings, and other activities.

Objectives:

• To design and implement a backend system for MLB's digital platform

• To create and manage player statistics, match schedules, ticket bookings, and

team management

• To implement a multi-threaded report generation system for performance

efficiency

Steps to Perform:

1. Player Management System:

• Design the Player class with specified attributes and methods

• Implement methods to update and retrieve player statistics

**Remarks:**

* **The player class defines a specific player.**
* **The player class should contain a method \_\_init\_\_ that**
  + **receives a player\_id, name and team in arguments (AND ALSO, AS ALWAYS, IN ALL CLASSES, self) and declares attributes player\_id, name and team equal to player\_id, name and team received in arguments**
  + **The method \_\_init\_\_ should further declare an attribute equal to an empty dictionary of statistics**
  + **Remark: the command my\_dictionary = {} creates an empty dictionary called “my\_dictionary)**
* **The player class should further include a method that receives a new team in argument and replaces the current team (self.team) with the new team received in argument**
* **The player class should further include a method to add a statistic: the method receives a name of a statistic and a value for the statistic in arguments and adds, to the dictionary of statistics, an element with a key equal to the name of the statistic received and the value corresponding to the key is equal to the value of the statistic received**
* **The player class should further include a method that receives no arguments (i.e: only receives “self”) and returns the dictionary of statistics**

**Example  
def update\_team (self,new\_team):  
 self.team = new\_team**

**Example for the statistic**

**def add\_stat (self,stat\_name,stat\_value):  
 self.statistics\_dictionary[stat\_name] = stat\_value**

**Example of returning statistics**

**def return\_stat (self):  
 return self.statistics\_dictionary**

2. Match Schedule Management:

• Design the Schedule class with specified attributes and methods

• Implement methods to update and retrieve match details

**Remarks:**

* **The schedule class defines a specific match.**
* **The schedule class should contain a method \_\_init\_\_ that receives a match\_id, a dictionary of teams (defined later, see 4.), a date and a location in arguments (AND ALSO, AS ALWAYS, IN ALL CLASSES, self) and declares attributes match\_id, teams, date and location equal to the match\_id, teams, date and location received in arguments**
* **The schedule class should further include a method that receives a new date in argument and replaces the current date with the new date**
* **The schedule class should further include a method that receives a new location in argument and replaces the current location with the new location**
* **The schedule class should further include a method that receives no arguments (i.e: only receives “self”) and returns the current match\_id, teams, date and location in the form of a dictionary**

3. Ticket Booking System:

• Design the Ticket class with specified attributes and methods

• Implement methods to book, cancel, and retrieve ticket details

**Remarks:**

* **The ticket class defines a specific ticket.**
* **The ticket class should contain a method \_\_init\_\_ that**
  + **receives a ticket\_id, match\_id, seat\_number and price in arguments (AND ALSO, AS ALWAYS, IN ALL CLASSES, self) and declares attributes ticket\_id, match\_id, seat\_number and price equal to the ticket\_id, match\_id, seat\_number and price in arguments**
  + **The method \_\_init\_\_ should further declare a buyer\_id equal to None**
* **The ticket class should further include a method that books a ticket; the method receives a buyer\_id in argument and replaces the current buyer\_id with the buyer\_id received**
* **The ticket class should further include a method that books a ticket; the method receives no arguments (i.e: only receives “self”) and replaces the current buyer\_id with None.**
* **The ticket class should further include a method that receives no arguments (i.e: only receives “self”) and returns the ticket\_id, match\_id, seat\_number, price and buyer\_id in the form of a dictionary**

4. Team Management System:

• Design the Team class with specified attributes and methods

• Implement methods to manage team rosters

**Remarks:**

* **The team class defines a specific team.**
* **The team class should contain a method \_\_init\_\_ that**
  + **receives a team\_id and a name in arguments (AND ALSO, AS ALWAYS, IN ALL CLASSES, self) and declares attributes team\_id and name equal to the team\_id and name in arguments**
  + **The method \_\_init\_\_ should further declare an empty list of players**
* **The team class should further include a method that adds a player to the team; the method receives a player\_id in argument and appends the player\_id to the list of players, on condition that the player\_id does not already belong to the team.**
* **The team class should further include a method that removes a player from the team; the method receives a player\_id in argument and removes the player\_id from the list of players, on condition that the player\_id does already belongs to the team.**
* **~~The team class should further include a method that books a ticket; the method receives no arguments (i.e: only receives “self”) and replaces the current buyer\_id with None~~.**
* **The team class should further include a method that returns the list of players**

5. Booking Management System:

• Design the Booking class with specified attributes and methods

• Implement methods to manage ticket bookings

**Remarks:**

* **The booking class defines a specific booking.**
* **The booking class should contain a method \_\_init\_\_ that receives no arguments (ie: only “self”) and declares an empty dictionary of bookings (as you will see in the next step, the values of the dictionary will be lists of ticket ids)**
* **The booking class should further include a method book\_ticket; the method receives a match\_id and a ticket\_id in arguments and performs the following:**
  + **Either: If the dictionary of bookings does not have a key equal to the match\_id, adds an element to the dictionary of bookings with a key equal to match\_id and the corresponding value equal to a list containing the ticket\_id only**
  + **Or: if the dictionary of bookings already has a key equal to the match\_id, appends the tcket\_id to the list that forms the value of the dictionary of bookings at the key equal to the match\_id.**
* **The booking class should further include a method that cancels a ticket; the method receives a match\_id and a ticket\_id in arguments and performs the following: if the the dictionary of bookings already has a key equal to the match\_id, and the value (ie: list) of the dictionary of bookings at the key equal to the match\_id contains the elements equal to ticket\_id, remove the ticket\_id from the list**
* **The booking class should further include a method that receives a match\_id in argument and returns all the ticket\_id found in the dictionary of bookings with the match\_id as a key**
* **The booking class should further include a method that receives a ticket\_id in argument and returns a list of all the match\_id‘s, from the dictionary of bookings keys, that include the ticket\_id in their list.**

6. Multi-Threaded Report Generation:

• Design the MLB Backend class with specified attributes and methods

• Implement a multi-threaded report generation system for player statistics

**Remarks:**

* **Create a class called “backend”**
* **The backend class should contain a method \_\_init\_\_ that receives no input (ie: only receives “self”) and declares 6 attributes: an empty dictionary of players, an empty dictionary of schedules, an empty dictionary of tickets, an empty dictionary of teams, an empty dictionary of reports and an instance of the “booking” class, called “bookings”:  
  Remark: the command my\_dictionary = {} creates an empty dictionary called “my\_dictionary).**
* **The backend class should further include a method to add a player: the method receives a player (ie: defined by the player class) in argument and adds, to the dictionary of players, an element with a key equal to the player\_id of the player received and the value corresponding to the key is equal to the player received.**
* **The backend class should further include a method to add a team: the method receives a team (ie: defined by the team class) in argument and adds, to the dictionary of teams, an element with a key equal to the team\_id of the team received and the value corresponding to the key is equal to the team received.**
* **The backend class should further include a method to add a schedule: the method receives a schedule (ie: defined by the schedule class) in argument and adds, to the dictionary of schedules, an element with a key equal to the match\_id of the schedule received and the value corresponding to the key is equal to the schedule received.**
* **The backend class should further include a method to add a ticket: the method receives a ticket (ie: defined by the ticket class) in argument and adds, to the dictionary of tickets, an element with a key equal to the ticket\_id of the ticket received and the value corresponding to the key is equal to the ticket received**
* **The backend class should further include a method book\_ticket: the method receives a ticket\_id and a buyer\_id in arguments and performs the following**
  + **If the dictionary of tickets includes the key equal to the ticket\_id received, get the ticket with the key ticket\_id from the dictionary of tickets**
  + **Call the function, from the ticket class, that books a ticket using the “buyer\_id” received in argument**
  + **Call the function book\_ticket, from the instance of the booking class declared as “bookings”, using ticket.match\_id and ticket\_id as arguments**
* **The backend class should further include a method generate\_player\_report: the method receives a player\_id in argument and performs the following:**
  + **Get the player from dictionary of player with the key equal to the player\_id received**
  + **Add an element to the dictionary of reports with a key equal to the player\_id and the value equal to the self.statistics\_dictionary of the player**

**7. Show examples of use of your classes. For EXAMPLE (example only, feel free to do your own):**

1. **Create an instance of the backend class**
2. **Create at least 2 players (i.e.: at least 2 instances of the player class)**
3. **Add the players to the “players” dictionary of the backend class**
4. **Create some statistics for each of the players using the method to add statistics from the player class**
5. **Create at least 2 teams using the team class**
6. **Add the teams to the backend class**
7. **Add players to each of the teams**
8. **Create a schedule of a match between 2 teams, using the schedule class**
9. **Add the schedule to the dictionary of schedules of the backend class**
10. **Create a ticket using the ticket class**
11. **Add the ticket to the dictionary of tickets from the backend class**
12. **Create a booking using the method “book\_ticket” from the backend class**
13. **Generate a player report for all the players in the teams of the match from H, using the method generate\_player\_report from the backend method.**