Stadium Creator

My biggest program is the „Stadium Creator" a **console application** designed to help users **manage stadiums, events, performers, and ticket sales**. The program implements numerous classes, so managing them, and making sure that created objects are added to relevant lists, was one of the biggest challenge. Additionally, C# sensitivity to data types makes it complicated to make sure that user inputs are always valid.

Code available on my GitHub:

https://github.com/DominikaBomba/stadiumcreator/blob/main/stadium_creator/Program.cs

Program specifics:

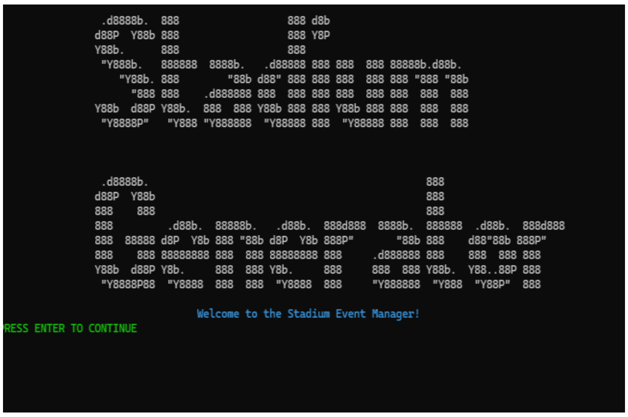
It is programmed in C#(on a .NET platform), using object-oriented programming. It has about 1300 lines of code

It uses:

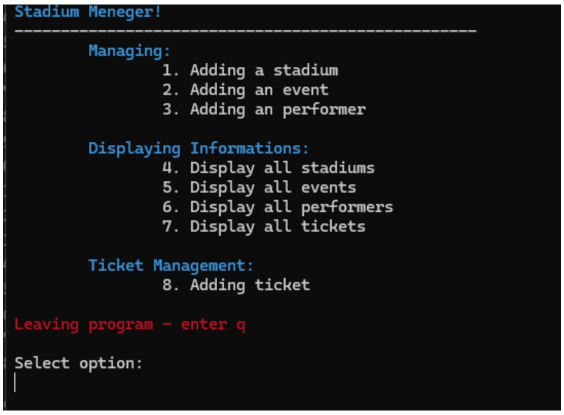
→ Objects, Classes, Constructors
→ Inheritance, Composition, Polymorphism
→ Interfaces

Program functionality:

→ It has a **menu-driven console** interface that guides users through various options for managing stadiums and events.



```
 .d8888b.  888                 888 d8b
d88P  Y88b 888                 888 Y8P
Y88b.      888                 888
 "Y888b.   888888  8888b.  .d88888 888 888  888 88888b.d88b.
    "Y88b. 888        "88b d88" 888 888 888  888 888 "888 "88b
      "888 888    .d888888 888  888 888 888  888 888  888  888
Y88b  d88P Y88b.  888  888 Y88b 888 888 Y88b 888 888  888  888
 "Y8888P"   "Y888 "Y888888  "Y88888 888  "Y88888 888  888  888



 .d8888b.                                              888
d88P  Y88b                                             888
888    888                                             888
888         .d88b.  88888b.   .d88b.  888d888  8888b.  888888  .d88b.  888d888
888  88888 d8P  Y8b 888 "88b d8P  Y8b 888P"       "88b 888    d88""88b 888P"
888    888 88888888 888  888 88888888 888     .d888888 888    888  888 888
Y88b  d88P Y8b.     888  888 Y8b.     888     888  888 Y88b.  Y88..88P 888
 "Y8888P88  "Y8888  888  888  "Y8888  888     "Y888888  "Y888  "Y88P"  888

                 Welcome to the Stadium Event Manager!
RESS ENTER TO CONTINUE
```

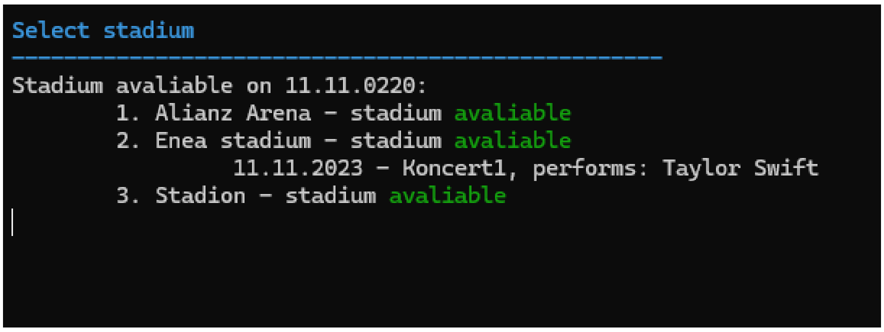


```
Stadium Meneger!
-----------------------------------------------------
        Managing:
                1. Adding a stadium
                2. Adding an event
                3. Adding an performer

        Displaying Informations:
                4. Display all stadiums
                5. Display all events
                6. Display all performers
                7. Display all tickets

        Ticket Management:
                8. Adding ticket

Leaving program - enter q

Select option:
|
```

→ **Adding and Displaying**:

- Users can **add new stadiums** and **view existing ones**.

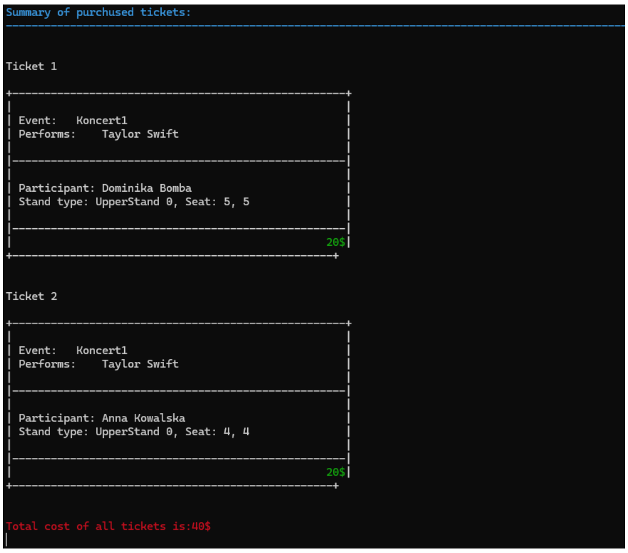
- Users can **add events**, check availability of stadiums, add details like event title, date, and performers.
- Performer Management: Users can **add performers** linked to events.



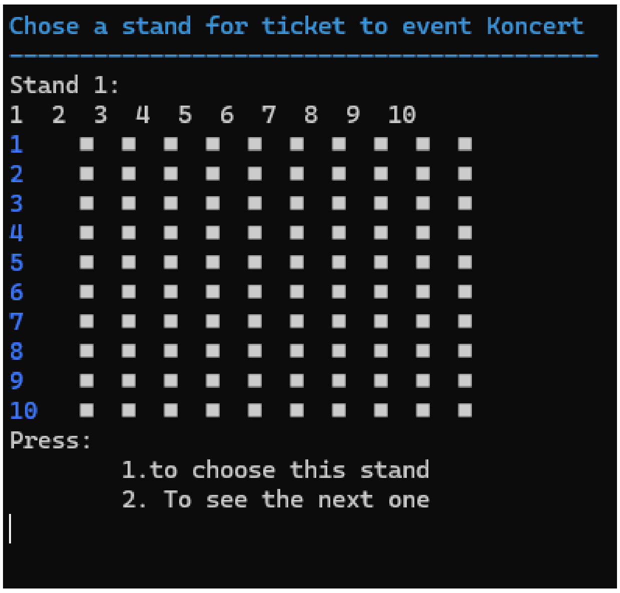
```
Select stadium
--------------------------------------------------
Stadium avaliable on 11.11.0220:
        1. Alianz Arena - stadium avaliable
        2. Enea stadium - stadium avaliable
                 11.11.2023 - Koncert1, performs: Taylor Swift
        3. Stadion - stadium avaliable
```

→ **Ticket Management**:

- Users can select events and **add tickets for those events**.
- The application handles ticket details, including stand type, **seat information**(checking their availability)**,** and participant names.
- **A summary of purchased tickets** is displayed, including event details **and total cost.**



```
Summary of purchused tickets:
------------------------------------------------------------------------------------------------


Ticket 1

+-----------------------------------------------------+
|                                                     |
| Event:   Koncert1                                   |
| Performs:    Taylor Swift                           |
|                                                     |
|-----------------------------------------------------|
|                                                     |
| Participant: Dominika Bomba                         |
| Stand type: UpperStand 0, Seat: 5, 5                |
|                                                     |
|-----------------------------------------------------|
|                                                  20$|
+---------------------------------------------------+


Ticket 2

+-----------------------------------------------------+
|                                                     |
| Event:   Koncert1                                   |
| Performs:    Taylor Swift                           |
|                                                     |
|-----------------------------------------------------|
|                                                     |
| Participant: Anna Kowalska                          |
| Stand type: UpperStand 0, Seat: 4, 4                |
|                                                     |
|-----------------------------------------------------|
|                                                  20$|
+---------------------------------------------------+


Total cost of all tickets is:40$
```



```
Chose a stand for ticket to event Koncert
------------------------------------------
Stand 1:
1  2  3  4  5  6  7  8  9  10
1    ■  ■  ■  ■  ■  ■  ■  ■  ■  ■
2    ■  ■  ■  ■  ■  ■  ■  ■  ■  ■
3    ■  ■  ■  ■  ■  ■  ■  ■  ■  ■
4    ■  ■  ■  ■  ■  ■  ■  ■  ■  ■
5    ■  ■  ■  ■  ■  ■  ■  ■  ■  ■
6    ■  ■  ■  ■  ■  ■  ■  ■  ■  ■
7    ■  ■  ■  ■  ■  ■  ■  ■  ■  ■
8    ■  ■  ■  ■  ■  ■  ■  ■  ■  ■
9    ■  ■  ■  ■  ■  ■  ■  ■  ■  ■
10   ■  ■  ■  ■  ■  ■  ■  ■  ■  ■
Press:
        1.to choose this stand
        2. To see the next one
```

Program Testing and bugs

→ I've created and run some **unit tests using nUnit** framework, to varify the basic functionality of Classes and Constructors.

→ To find bugs I used Console.WriteLine statements to **print variable values** and program states at different points, to see what's happening internally.

→ I've also added various **checks to ensure that user inputs are valid**, prompting users to enter values within acceptable ranges (by using **try-catch** and **TryParse** methods) and providing feedback for invalid inputs. (e.g. The program ensures the year of birth cannot be later than the current year.)