**Olympic Games**

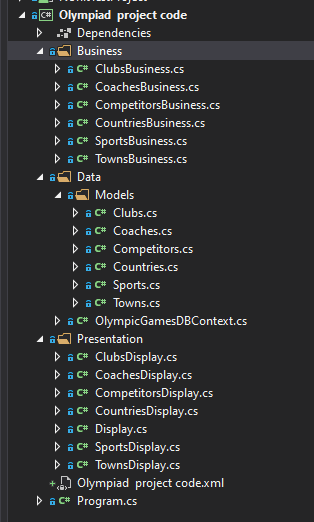
**Rio 2016**

*Изготвили: Надя Колева и Йоана Михайлова*

*От: ПМГ „Академик Боян Петканчин“, гр. Хасково*

*Проект по 7ми модул от НП „Обучение за ИТ кариера“*

**Olympiad Project code**



* Data Folder
  + Models Folder
    - **Clubs.cs**

public partial class Clubs

{

public Clubs()

{

Competitors = new HashSet<Competitors>();

}

public int Id { get; set; }

public string Name { get; set; }

public virtual ICollection<Competitors> Competitors { get; set; }

}

* + - **Coaches.cs**

public partial class Coaches

{

public Coaches()

{

Competitors = new HashSet<Competitors>();

}

public int Id { get; set; }

public string Name { get; set; }

public int SportId { get; set; }

public virtual Sports Sport { get; set; }

public virtual ICollection<Competitors> Competitors { get; set; }

}

* + - **Sports.cs**

public partial class Sports

{

public Sports()

{

Coaches = new HashSet<Coaches>();

Competitors = new HashSet<Competitors>();

}

public int Id { get; set; }

public string Name { get; set; }

public virtual ICollection<Coaches> Coaches { get; set; }

public virtual ICollection<Competitors> Competitors { get; set; }

}

* + - **Competitors.cs**

public partial class Competitors

{

public int Id { get; set; }

public string FullName { get; set; }

public string BirthDate { get; set; }

public int Age { get; set; }

public string Gender { get; set; }

public string Weight { get; set; }

public int TownId { get; set; }

public int? ClubId { get; set; }

public int? CoachId { get; set; }

public int SportId { get; set; }

public virtual Clubs Club { get; set; }

public virtual Coaches Coach { get; set; }

public virtual Sports Sport { get; set; }

public virtual Towns Town { get; set; }

}

* + - **Countries.cs**

public partial class Countries

{

public Countries()

{

Towns = new HashSet<Towns>();

}

public int Id { get; set; }

public string Name { get; set; }

public virtual ICollection<Towns> Towns { get; set; }

}

* + - **Towns.cs**

public partial class Towns

{

public Towns()

{

Competitors = new HashSet<Competitors>();

}

public int Id { get; set; }

public string Name { get; set; }

public int CountryId { get; set; }

public virtual Countries Country { get; set; }

public virtual ICollection<Competitors> Competitors { get; set; }

}

* Business Folder
  + **ClubsBusiness.cs**

/// <summary>

/// Тhe <c>ClubsBusiness</c> class is in Business layer.

/// It works as a bridge between the ClubsDisplay class and the database.

/// </summary>

/// <remarks>

/// The results are received in row data in Data Table format.

/// ClubsBusiness converts it into Value Objects.

/// </remarks>

public class ClubsBusiness

{

private OlympicGamesDBContext olympicGamesDBContext;

/// <summary>

/// Constructor for ClubsBusiness class.

/// </summary>

/// <param name="context"></param>

public ClubsBusiness(OlympicGamesDBContext context)

{

this.olympicGamesDBContext = context;

}

/// <summary>

/// Gets all clubs in Clubs table.

/// </summary>

/// <returns>A list of all club's names and ids.</returns>

public List<Clubs> GetAllClubs()

{

return olympicGamesDBContext.Clubs.ToList();

}

/// <summary>

/// Finds the club with the id the user has entered.

/// </summary>

/// <param name="id"></param>

/// <returns>The id and the name of the club.</returns>

public Clubs GetClubById(int? id)

{

return olympicGamesDBContext.Clubs.Find(id);

}

/// <summary>

/// Finds the club with the name the user has entered.

/// </summary>

/// <param name="name"></param>

/// <returns>The id and the name of the club.</returns>

public Clubs GetClubByName(string name)

{

var club = olympicGamesDBContext.Clubs

.Where(c => c.Name == name)

.FirstOrDefault();

return club;

}

/// <summary>

/// Adds a club in the database.

/// </summary>

/// <param name="club">The club that is being added.</param>

public void AddClub(Clubs club)

{

olympicGamesDBContext.Clubs.Add(club);

olympicGamesDBContext.SaveChanges();

}

/// <summary>

/// Updates a club in the database.

/// </summary>

/// <param name="club">The club that is being updated.</param>

public void UpdateClub(Clubs club)

{

var item = olympicGamesDBContext.Clubs.Find(club.Id);

if (item != null)

{

olympicGamesDBContext.Entry(item).CurrentValues.SetValues(club);

olympicGamesDBContext.SaveChanges();

}

}

/// <summary>

/// Deletes a club from the database by given id.

/// </summary>

/// <param name="id">The id of the club wanted to be deleted.</param>

public void DeleteClubById(int id)

{

var item = olympicGamesDBContext.Clubs.Find(id);

if (item != null)

{

olympicGamesDBContext.Remove(item);

olympicGamesDBContext.SaveChanges();

}

}

}

* + **CoachesBusiness.cs**

/// <summary>

/// Тhe <c>CoachesBusiness</c> class is in Business layer.

/// It works as a bridge between the CoachesDisplay class and the database.

/// </summary>

/// <remarks>

/// The results are received in row data in Data Table format.

/// CoachesBusiness converts it into Value Objects.

/// </remarks>

public class CoachesBusiness

{

private OlympicGamesDBContext olympicGamesDBContext;

/// <summary>

/// Constructor for CoachesBusiness class.

/// </summary>

/// <param name="context"></param>

public CoachesBusiness(OlympicGamesDBContext context)

{

this.olympicGamesDBContext = context;

}

/// <summary>

/// Gets all coaches in Coaches table.

/// </summary>

/// <returns>A list of all coach's names, ids and sport.</returns>

public List<Coaches> GetAllCoaches()

{

return olympicGamesDBContext.Coaches.ToList();

}

/// <summary>

/// Finds the coach with the id the user has entered.

/// </summary>

/// <param name="id"></param>

/// <returns>The id, the name of the coach and the sport they are coaching.</returns>

public Coaches GetCoachById(int? id)

{

return olympicGamesDBContext.Coaches.Find(id);

}

/// <summary>

/// Finds the coach with the name the user has entered.

/// </summary>

/// <param name="name"></param>

/// <returns>The id, the name of the coach and the sport they are coaching.</returns>

public Coaches GetCoachByName(string name)

{

var coach = olympicGamesDBContext.Coaches

.Where(c => c.Name == name)

.FirstOrDefault();

return coach;

}

/// <summary>

/// Adds a coach in the database.

/// </summary>

/// <param name="coach">The coach that is being added.</param>

public void AddCoach(Coaches coach)

{

olympicGamesDBContext.Coaches.Add(coach);

olympicGamesDBContext.SaveChanges();

}

/// <summary>

/// Updates a coach in the database.

/// </summary>

/// <param name="coach">The coach that is being updated.</param>

public void UpdateCoach(Coaches coach)

{

var item = olympicGamesDBContext.Coaches.Find(coach.Id);

if (item != null)

{

olympicGamesDBContext.Entry(item).CurrentValues.SetValues(coach);

olympicGamesDBContext.SaveChanges();

}

}

/// <summary>

/// Deletes a coach from the database by given id.

/// </summary>

/// <param name="id">The id of the coach wanted to be deleted.</param>

public void DeleteCoachById(int id)

{

var item = olympicGamesDBContext.Coaches.Find(id);

if (item != null)

{

olympicGamesDBContext.Remove(item);

olympicGamesDBContext.SaveChanges();

}

}

}

* + **CompetitorsBusiness.cs**

/// <summary>

/// Тhe <c>CompetitorsBusiness</c> class is in Business layer.

/// It works as a bridge between the CompetitorsDisplay class and the database.

/// </summary>

/// <remarks>

/// The results are received in row data in Data Table format.

/// CompetitorsBusiness converts it into Value Objects.

/// </remarks>

public class CompetitorsBusiness

{

private OlympicGamesDBContext olympicGamesDBContext;

/// <summary>

/// Constructor for CompetitorsBusiness class.

/// </summary>

/// <param name="context"></param>

public CompetitorsBusiness(OlympicGamesDBContext context)

{

this.olympicGamesDBContext = context;

}

/// <summary>

/// Gets all competitors in Competitors table.

/// </summary>

/// <returns>A list of all competitors and everyting about them.</returns>

public List<Competitors> GetAllCompetitors()

{

return olympicGamesDBContext.Competitors.ToList();

}

/// <summary>

/// Finds the competitor with the id the user has entered.

/// </summary>

/// <param name="id"></param>

/// <returns>The name of the competitor and everyting about them.</returns>

public Competitors GetCompetitorById(int id)

{

return olympicGamesDBContext.Competitors.Find(id);

}

/// <summary>

/// Finds the competitor with the name the user has entered.

/// </summary>

/// <param name="name"></param>

/// <returns>The name of the competitor and everyting about them..</returns>

public Competitors GetCompetitorByName(string name)

{

var competitor = olympicGamesDBContext.Competitors

.Where(c => c.FullName == name)

.FirstOrDefault();

return competitor;

}

/// <summary>

/// Adds a competitor in the database.

/// </summary>

/// <param name="competitors">The competitor that is being added.</param>

public void AddCompetitors(Competitors competitors)

{

olympicGamesDBContext.Competitors.Add(competitors);

olympicGamesDBContext.SaveChanges();

}

/// <summary>

/// Updates a competitor in the database.

/// </summary>

/// <param name="competitor">The competitor that is being updated.</param>

public void UpdateCompetitor(Competitors competitor)

{

var item = olympicGamesDBContext.Competitors.Find(competitor.Id);

if (item != null)

{

olympicGamesDBContext.Entry(item).CurrentValues.SetValues(competitor);

olympicGamesDBContext.SaveChanges();

}

}

/// <summary>

/// Deletes a competitor from the database by given id.

/// </summary>

/// <param name="id">The id of the competitor wanted to be deleted.</param>

public void DeleteCompetitorById(int id)

{

var item = olympicGamesDBContext.Competitors.Find(id);

if (item != null)

{

olympicGamesDBContext.Remove(item);

olympicGamesDBContext.SaveChanges();

}

}

}

* + **CountriesBusiness.cs**

/// <summary>

/// Тhe <c>CountriesBusiness</c> class is in Business layer.

/// It works as a bridge between the CountrieDisplay class and the database.

/// </summary>

/// <remarks>

/// The results are received in row data in Data Table format.

/// CountriesBusiness converts it into Value Objects.

/// </remarks>

public class CountriesBusiness

{

private OlympicGamesDBContext olympicGamesDBContext;

/// <summary>

/// Constructor for CountriesBusiness class.

/// </summary>

/// <param name="context"></param>

public CountriesBusiness(OlympicGamesDBContext context)

{

this.olympicGamesDBContext = context;

}

/// <summary>

/// Gets all countries in Countries table.

/// </summary>

/// <returns>A list of all countries's names and their ids.</returns>

public List<Countries> GetAllCountries()

{

return olympicGamesDBContext.Countries.ToList();

}

/// <summary>

/// Finds the country with the id the user has entered.

/// </summary>

/// <param name="id"></param>

/// <returns>The id and the name of the country.</returns>

public Countries GetCountryById(int id)

{

return olympicGamesDBContext.Countries.Find(id);

}

/// <summary>

/// Finds the country with the name the user has entered.

/// </summary>

/// <param name="name"></param>

/// <returns>The id and the name of the country.</returns>

public Countries GetCountryByName(string name)

{

var country = olympicGamesDBContext.Countries

.Where(c => c.Name == name)

.FirstOrDefault();

return country;

}

}

* + **SportsBusiness.cs**

/// <summary>

/// Тhe <c>SportsBusiness</c> class is in Business layer.

/// It works as a bridge between the SportsDisplay class and the database.

/// </summary>

/// <remarks>

/// The results are received in row data in Data Table format.

/// SportsBusiness converts it into Value Objects.

/// </remarks>

public class SportsBusiness

{

private OlympicGamesDBContext olympicGamesDBContext;

/// <summary>

/// Constructor for SportsBusiness class.

/// </summary>

/// <param name="context"></param>

public SportsBusiness(OlympicGamesDBContext context)

{

this.olympicGamesDBContext = context;

}

/// <summary>

/// Gets all sports in Sports table.

/// </summary>

/// <returns>A list of all sport's names and their ids.</returns>

public List<Sports> GetAllSports()

{

return olympicGamesDBContext.Sports.ToList();

}

/// <summary>

/// Finds the sport with the id the user has entered.

/// </summary>

/// <param name="id"></param>

/// <returns>The id and the name of the sport.</returns>

public Sports GetSportById(int id)

{

return olympicGamesDBContext.Sports.Find(id);

}

/// <summary>

/// Finds the sport with the name the user has entered.

/// </summary>

/// <param name="name"></param>

/// <returns>The id and the name of the sport.</returns>

public Sports GetSportByName(string name)

{

var sport = olympicGamesDBContext.Sports

.Where(c => c.Name == name)

.FirstOrDefault();

return sport;

}

}

* + **TownsBusiness.cs**

/// <summary>

/// Тhe <c>TownsBusiness</c> class is in Business layer.

/// It works as a bridge between the TownsDisplay class and the database.

/// </summary>

/// <remarks>

/// The results are received in row data in Data Table format.

/// TownsBusiness converts it into Value Objects.

/// </remarks>

public class TownsBusiness

{

private OlympicGamesDBContext olympicGamesDBContext;

/// <summary>

/// Constructor for TownsBusiness class.

/// </summary>

/// <param name="context"></param>

public TownsBusiness(OlympicGamesDBContext context)

{

this.olympicGamesDBContext = context;

}

/// <summary>

/// Gets all towns in Towns table.

/// </summary>

/// <returns>A list of all town's names, ids and countries.</returns>

public List<Towns> GetAllTowns()

{

return olympicGamesDBContext.Towns.ToList();

}

/// <summary>

/// Finds the town with the id the user has entered.

/// </summary>

/// <param name="id"></param>

/// <returns>The id, the name of the town and the country it is in.</returns>

public Towns GetTownById(int id)

{

return olympicGamesDBContext.Towns.FirstOrDefault(p => p.Id == id);

}

/// <summary>

/// Finds the town with the name the user has entered.

/// </summary>

/// <param name="name"></param>

/// <returns>The id, the name of the town and the country it is in.</returns>

public Towns GetTownByName(string name)

{

var town = olympicGamesDBContext.Towns

.Where(c => c.Name == name)

.FirstOrDefault();

return town;

}

/// <summary>

/// Adds a town in the database.

/// </summary>

/// <param name="town">The town that is being added.</param>

public void AddTown(Towns town)

{

olympicGamesDBContext.Towns.Add(town);

olympicGamesDBContext.SaveChanges();

}

/// <summary>

/// Updates a town in the database.

/// </summary>

/// <param name="town">The town that is being updated.</param>

public void UpdateTown(Towns town)

{

var item = olympicGamesDBContext.Towns.Find(town.Id);

if (item != null)

{

olympicGamesDBContext.Entry(item).CurrentValues.SetValues(town);

olympicGamesDBContext.SaveChanges();

}

}

/// <summary>

/// Deletes a town from the database by given id.

/// </summary>

/// <param name="id">The id of the town wanted to be deleted.</param>

public void DeleteTownById(int id)

{

var item = olympicGamesDBContext.Towns.Find(id);

if (item != null)

{

olympicGamesDBContext.Towns.Remove(item);

olympicGamesDBContext.SaveChanges();

}

}

}

* Presentation Folder
  + **ClubsDisplay.cs**

/// <summary>

/// Тhe <c>ClubsDisplay</c> class in Presentation.

/// This is the layer which is directly connected to ClubsBusiness.

/// </summary>

/// <remarks>

/// This class receive information from the user.

/// Then it is passed to ClubsBusiness.

/// </remarks>

class ClubsDisplay

{

private ClubsBusiness clubsBusiness;

/// <summary>

/// Constructor for ClubsDisplay class.

/// </summary>

/// <param name="context"></param>

public ClubsDisplay(OlympicGamesDBContext context)

{

clubsBusiness = new ClubsBusiness(context);

}

/// <summary>

/// "Calls" method "GetAllTowns" from ClubsBusiness.

/// Then it shows all clubs in table Clubs.

/// </summary>

public void GetAllClubs()

{

Console.WriteLine("Clubs:");

List<Clubs> clubs = clubsBusiness.GetAllClubs();

if(clubs.Count == 0)

{

Console.WriteLine("There are no clubs in the table!");

}

else

{

Console.WriteLine("Id" + new string(' ', 4)//6

+ "ClubName" + new string(' ', 36));

Console.WriteLine(new string('-', 50));

foreach (var club in clubs)

{

string output = $"{club.Id}" + new string(' ', 6 - club.Id.ToString().Length)

+ $"{club.Name}" + new string(' ', 44 - club.Name.Length);

Console.WriteLine(output);

}

Console.WriteLine(new string('-', 50));

}

}

/// <summary>

/// After the user has inputed id, the program "Calls" method "GetClubById" from ClubsBusiness.

/// Shows the Club who has this id.

/// </summary>

public void GetClubById()

{

Console.Write("Enter Club ID to fetch: ");

int id = int.Parse(Console.ReadLine());

Clubs club = clubsBusiness.GetClubById(id);

if(club != null)

{

PrintClub(club);

}

else

{

Console.WriteLine($"There is no club with ID = {id} in the table!");

}

}

/// <summary>

/// After the user has inputed idname the program "Calls" method "GetClubById" from ClubsBusiness.

/// Shows the Club who has this name.

/// </summary>

public void GetClubByName()

{

Console.Write("Enter Club Name to fetch: ");

string name = Console.ReadLine();

Clubs club = clubsBusiness.GetClubByName(name);

if (club != null)

{

PrintClub(club);

}

else

{

Console.WriteLine($"There is no club with name = {name} in the table!");

}

}

private void PrintClub(Clubs club)

{

Console.WriteLine(new string('-', 40));

Console.WriteLine($"ID: {club.Id}");

Console.WriteLine($"Name: {club.Name}");

Console.WriteLine(new string('-', 40));

}

/// <summary>

/// Makes the user to input data about the club.

/// Passes the information to ClubsBusiness, using the method "AddClub"

/// </summary>

public void AddClub()

{

Clubs club = new Clubs();

Console.Write("Enter Club Name: ");

club.Name = Console.ReadLine();

clubsBusiness.AddClub(club);

Console.WriteLine($"New club successfully added!");

}

/// <summary>

/// Finds the club wished to be updated.

/// Makes the user to enter the new information.

/// Then passes it to ClubsBusiness, using the method "UpdateClub".

/// </summary>

public void UpdateClub()

{

Console.Write("Enter Club ID to update: ");

int id = int.Parse(Console.ReadLine());

Clubs club = clubsBusiness.GetClubById(id);

if(club == null)

{

Console.WriteLine($"There is no club with ID = {id} in the table!");

}

else

{

Console.WriteLine("Enter Club Name: ");

club.Name = Console.ReadLine();

clubsBusiness.UpdateClub(club);

Console.WriteLine("Club successfully updated!");

}

}

/// <summary>

/// Finds the club wished to be deleted.

/// Passes the information to ClubsBusiness, using the method "DeleteClubById".

/// </summary>

public void DeleteClubById()

{

Console.Write("Enter Club Id to delete: ");

int id = int.Parse(Console.ReadLine());

if(clubsBusiness.GetClubById(id) == null)

{

Console.WriteLine($"There is no club with ID = {id} in the table!");

}

else

{

clubsBusiness.DeleteClubById(id);

Console.WriteLine("Done!");

}

}

}

* + **CoachesDisplay.cs**

/// <summary>

/// Тhe <c>CoachesDisplay</c> class in Presentation.

/// This is the layer which is directly connected to CoachesBusiness.

/// </summary>

/// <remarks>

/// This class receive information from the user.

/// Then it is passed to CoachesBusiness.

/// </remarks>

class CoachesDisplay

{

private CoachesBusiness coachesBusiness;

private SportsBusiness sportsBusiness ;

/// <summary>

/// Constructor for CoachesDisplay class.

/// </summary>

/// <param name="context"></param>

public CoachesDisplay(OlympicGamesDBContext context)

{

coachesBusiness = new CoachesBusiness(context);

sportsBusiness = new SportsBusiness(context);

}

/// <summary>

/// "Calls" method "GetAllCoaches" from CoachesBusiness.

/// Then it shows all coaches in table Coaches.

/// </summary>

public void GetAllCoaches()

{

Console.WriteLine("Coaches: ");

List<Coaches> coaches = coachesBusiness.GetAllCoaches();

if(coaches.Count == 0)

{

Console.WriteLine("There are no coaches in the table!");

}

else

{

Console.WriteLine("Id" + new string(' ', 4) + "Name" + new string(' ', 25) + "Sport Name");

Console.WriteLine(new string('-', 56));

foreach (var coach in coaches)

{

var sport = sportsBusiness.GetSportById(coach.SportId);

string output = $"{coach.Id}" + new string(' ', 6 - coach.Id.ToString().Length)

+ $"{coach.Name}" + new string(' ', 29 - coach.Name.Length)

+ $"{sport.Name}" + new string(' ', 21 - sport.Name.Length);

Console.WriteLine(output);

}

Console.WriteLine(new string('-', 56));

}

}

/// <summary>

/// After the user has inputed id, the program "Calls" method "GetCoachById" from CoachesBusiness.

/// Shows the Coach who has this id.

/// </summary>

public void GetCoachById()

{

Console.Write("Enter Coach ID to fetch: ");

int id = int.Parse(Console.ReadLine());

var coach = coachesBusiness.GetCoachById(id);

var sport = sportsBusiness.GetSportById(coach.SportId);

if (coach != null)

{

PrintCoach(coach, sport);

}

else

{

Console.WriteLine($"There is no coach with ID = {id} in the table!");

}

}

/// <summary>

/// After the user has inputed name, the program "Calls" method "GetCoachById" from CoachesBusiness.

/// Shows the Coach who has this name.

/// </summary>

public void GetCoachByName()

{

Console.Write("Enter Coach Name to fetch: ");

string name = Console.ReadLine();

var coach = coachesBusiness.GetCoachByName(name);

var sport = sportsBusiness.GetSportById(coach.SportId);

if (coach != null)

{

PrintCoach(coach, sport);

}

else

{

Console.WriteLine($"There is no coach with name = {name} in the table!");

}

}

private void PrintCoach(Coaches coach, Sports sport)

{

Console.WriteLine(new string('-', 40));

Console.WriteLine($"ID: {coach.Id}");

Console.WriteLine($"Name: {coach.Name}");

Console.WriteLine($"Sport Name: {sport.Name}");

Console.WriteLine(new string('-', 40));

}

/// <summary>

/// Makes the user to input data about the coach.

/// Passes the information to CoachesBusinessiness, using the method "AddCoach"

/// </summary>

public void AddCoach()

{

var coach = new Coaches();

Console.Write("Enter Coach Name: ");

coach.Name = Console.ReadLine();

Console.Write("Enter Sport Name: ");

string sportName = Console.ReadLine();

coach.SportId = sportsBusiness.GetSportByName(sportName).Id;

coachesBusiness.AddCoach(coach);

Console.WriteLine($"New coach successfully added!");

}

/// <summary>

/// Finds the coach wished to be updated.

/// Makes the user to enter the new information.

/// Then passes it to CoachesBusinessiness, using the method "UpdateCoach".

/// </summary>

public void UpdateCoach()

{

Console.Write("Enter ID to update: ");

int id = int.Parse(Console.ReadLine());

Coaches coach = coachesBusiness.GetCoachById(id);

if(coach == null)

{

Console.WriteLine($"There is no club with ID = {id} in the table!");

}

else

{

Console.Write("Enter Coach Name: ");

coach.Name = Console.ReadLine();

Console.Write("Enter Sport Name: ");

string sportName = Console.ReadLine();

coach.SportId = sportsBusiness.GetSportByName(sportName).Id;

coachesBusiness.UpdateCoach(coach);

Console.WriteLine("Coach successfully updated!");

}

}

/// <summary>

/// Finds the town wished to be deleted.

/// Passes the information to CoachesBusinessiness, using the method "DeleteCoachById".

/// </summary>

public void DeleteCoachById()

{

Console.Write("Enter ID to delete: ");

int id = int.Parse(Console.ReadLine());

if (coachesBusiness.GetCoachById(id) == null)

{

Console.WriteLine($"There is no club with ID = {id} in the table!");

}

else

{

coachesBusiness.DeleteCoachById(id);

Console.WriteLine("Done!");

}

}

}

* **CompetitorsDisplay.cs**

using Olympiad\_\_project\_code.Business;

using Olympiad\_\_project\_code.Business\_layer;

using Olympiad\_\_project\_code.Models;

using System;

using System.Collections.Generic;

using System.Runtime.InteropServices.WindowsRuntime;

using System.Text;

using System.Xml;

namespace Olympiad\_\_project\_code.Presentation

{

    /// <summary>

    /// Тhe <c>CompetitorsDisplay</c> class in Presentation.

    /// This is the layer which is directly connected to CompetitorsBusiness.

    /// </summary>

    /// <remarks>

    /// This class receive information from the user.

    /// Then it is passed to CompetitorsBusiness.

    /// </remarks>

    class CompetitorsDisplay

    {

        private CoachesBusiness coachesBusiness;

        private SportsBusiness sportsBusiness;

        private ClubsBusiness clubsBusiness;

        private TownsBusiness townsBusiness;

        private TownsDisplay townsDisplay;

        private ClubsDisplay clubsDisplay;

        private CompetitorsBusiness competitorsBusiness;

        private CoachesDisplay coachesDisplay;

        //Ъм тука имам въпорс - i az

        /// <summary>

        /// Constructor for CompetitorsDisplay class.

        /// </summary>

        /// <param name="context"></param>

        public CompetitorsDisplay(OlympicGamesDBContext context)

        {

            competitorsBusiness = new CompetitorsBusiness(context);

            coachesBusiness = new CoachesBusiness(context);

            clubsBusiness = new ClubsBusiness(context);

            townsBusiness = new TownsBusiness(context);

            sportsBusiness = new SportsBusiness(context);

            townsDisplay = new TownsDisplay(context);

            clubsDisplay = new ClubsDisplay(context);

            coachesDisplay = new CoachesDisplay(context);

        }

        /// <summary>

        /// "Calls" method "GetAllCompetitors" from CompetitorsBusiness.

        /// Then it shows all competitors in table Competitors.

        /// </summary>

        public void GetAllCompetitors()

        {

            Console.WriteLine("Competitors: ");

            List<Competitors> competitors = competitorsBusiness.GetAllCompetitors();

            if (competitors.Count == 0)

            {

                Console.WriteLine("There are no competitors in the table!");

            }

            else

            {

                Console.WriteLine(new string(' ', 2) + "Id" + new string(' ', 2)//6

                    + new string(' ', 13) + "FullName" + new string(' ', 13)//34

                    + new string(' ', 5) + "BirthDate" + new string(' ', 5)//19

                    + new string(' ', 2) + "Age" + new string(' ',2)//8

                    + new string(' ', 2) + "Gender" + new string(' ', 2)//10

                    + new string(' ', 2) + "Weight" + new string(' ', 2)//10

                    + new string(' ', 10) + "TownName" + new string(' ', 10)//28

                    + new string(' ', 18) + "ClubName" + new string(' ', 18)//44

                    + new string(' ', 10) + "CoachName" + new string(' ', 10)//29

                    + new string(' ', 6) + "SportName" + new string(' ', 6));//21

                Console.WriteLine(new string('-', 210));

                foreach (var competitor in competitors)

                {

                    var town = townsBusiness.GetTownById(competitor.TownId);

                    string clubName = GetClubAndCoachNames(competitor, "club");

                    string coachName = GetClubAndCoachNames(competitor, "coach");

                    var sport = sportsBusiness.GetSportById(competitor.SportId);

                    string output = $"{competitor.Id}" + new string(' ', 6 - competitor.Id.ToString().Length)

                        + $"{competitor.FullName}" + new string(' ', 34 - competitor.FullName.Length)

                        + $"{competitor.BirthDate}" + new string(' ', 19 - competitor.BirthDate.Length)

                        + new string(' ', 2) + $"{competitor.Age}" + new string(' ', 2)

                        + new string(' ', 3) + $"{competitor.Gender}" + new string(' ', 8 - competitor.Gender.Length)

                        + new string(' ', 3) + $"{competitor.Weight}" + new string(' ', 9 - competitor.Weight.Length)

                        + $"{town.Name}" + new string(' ', 28 - town.Name.Length)

                        + $"{clubName}" + new string(' ', 44 - clubName.Length)

                        + $"{coachName}" + new string(' ', 29 - coachName.Length)

                        + $"{sport.Name}" + new string(' ', 21 - sport.Name.Length);

                    Console.WriteLine(output);

                }

                Console.WriteLine(new string('-', 210));

            }

        }

        /// <summary>

        /// After the user has inputed id, the program "Calls" method "GetCompetitorById" from CompetitorsBusiness.

        /// Shows the Competitor who has this id.

        /// </summary>

        public void GetCompetitorById()

        {

            Console.Write("Enter Competitor ID to fetch: ");

            int id = int.Parse(Console.ReadLine());

            var competitor = competitorsBusiness.GetCompetitorById(id);

            var town = townsBusiness.GetTownById(competitor.TownId);

            string clubName = GetClubAndCoachNames(competitor, "club");

            string coachName = GetClubAndCoachNames(competitor, "coach");

            var sport = sportsBusiness.GetSportById(competitor.SportId);

            if (competitor != null)

            {

                PrintCompetitor(competitor, town, clubName, coachName, sport);

            }

            else

            {

                Console.WriteLine($"There is no competitor with ID = {id} in the table!");

            }

        }

        /// <summary>

        /// After the user has inputed name, the program "Calls" method "GetCompetitorByName" from CompetitorsBusiness.

        /// Shows the Competitor who has this name.

        /// </summary>

        public void GetCompetitorByName()

        {

            Console.Write("Enter Competitor Name to fetch: ");

            string name = Console.ReadLine();

            var competitor = competitorsBusiness.GetCompetitorByName(name);

            var town = townsBusiness.GetTownById(competitor.TownId);

            string clubName = GetClubAndCoachNames(competitor, "club");

            string coachName = GetClubAndCoachNames(competitor, "coach");

            var sport = sportsBusiness.GetSportById(competitor.SportId);

            if (competitor != null)

            {

                PrintCompetitor(competitor, town, clubName, coachName, sport);

            }

            else

            {

                Console.WriteLine($"There is no competitor with Name = {name} in the table!");

            }

        }

        /// <summary>

        /// "Calls" method "CreateCompetitor".

        /// Passes the information to CompetitorsBusiness, using the method "AddCompetitors"

        /// </summary>

        public void AddCompetitor()

        {

            Competitors competitors = new Competitors();

            competitorsBusiness.AddCompetitors(CreateCompetitor(competitors));

            Console.WriteLine($"New competitor successfully added!");

        }

        /// <summary>

        /// Finds the competitor wished to be updated.

        /// "Calls" method "CreateCompetitor".

        /// Then passes it to CompetitorsBusiness, using the method "UpdateCompetitor".

        /// </summary>

        public void UpdateCompetitor()

        {

            Console.Write("Enter ID to update: ");

            int id = int.Parse(Console.ReadLine());

            Competitors competitor = competitorsBusiness.GetCompetitorById(id);

            if (competitor == null)

            {

                Console.WriteLine($"There is no competitor with ID = {id} in the table!");

            }

            else

            {

                //competitor = ;

                competitorsBusiness.UpdateCompetitor(CreateCompetitor(competitor));

                Console.WriteLine("Competitor successfully updated!");

            }

        }

        /// <summary>

        /// Finds the competitor wished to be deleted.

        /// Passes the information to CompetitorsBusiness, using the method "DeleteCompetitorById".

        /// </summary>

        public void DeleteCompetitorById()

        {

            Console.Write("Enter ID to delete: ");

            int id = int.Parse(Console.ReadLine());

            if (competitorsBusiness.GetCompetitorById(id) == null)

            {

                Console.WriteLine($"There is no competitor with ID = {id} in the table!");

            }

            else

            {

                competitorsBusiness.DeleteCompetitorById(id);

                Console.WriteLine("Done!");

            }

        }

        private void PrintCompetitor(Competitors competitor, Towns town, string clubName, string coachName, Sports sport)

        {

            Console.WriteLine(new string('-', 40));

            Console.WriteLine($"ID: {competitor.Id}");

            Console.WriteLine($"Full Name: {competitor.FullName}");

            Console.WriteLine($"Birth Date: {competitor.BirthDate}");

            Console.WriteLine($"Age: {competitor.Age}");

            Console.WriteLine($"Gender: {competitor.Gender}");

            Console.WriteLine($"Town Name: {town.Name}");

            Console.WriteLine($"Club Name: {clubName}");

            Console.WriteLine($"Coach Name: {coachName}");

            Console.WriteLine($"Sport Name: {sport.Name}");

            Console.WriteLine(new string('-', 40));

        }

        private string GetClubAndCoachNames(Competitors competitor, string name)

        {

            if (name == "club")

            {

                string clubName = "";

                if (competitor.ClubId != null)

                {

                    clubName = clubsBusiness.GetClubById(competitor.ClubId).Name;

                }

                return clubName;

            }

            else

            {

                string coachName = "";

                if (competitor.CoachId != null)

                {

                    coachName = coachesBusiness.GetCoachById(competitor.CoachId).Name;

                }

                return coachName;

            }

        }

        private Competitors CreateCompetitor(Competitors competitor)

        {

            Console.Write("Enter Competitor Full Name: ");

            competitor.FullName = Console.ReadLine();

            Console.Write("Enter Competitor Birth Date: ");

            competitor.BirthDate = Console.ReadLine();

            Console.Write("Enter Competitor Age: ");

            competitor.Age = int.Parse(Console.ReadLine());

            Console.Write("Enter Competitor Gender: ");

            competitor.Gender = Console.ReadLine();

            Console.Write("Enter Competitor Weight: ");

            competitor.Weight = Console.ReadLine();

            Console.Write("Enter Competitor Town Name: ");

            string townName = Console.ReadLine();

            if (townsBusiness.GetTownByName(townName) == null)

            {

                Console.WriteLine($"There is no town with the name {townName}");

                Console.WriteLine("Do you want to create new town?");

                Console.WriteLine("1. Yes" + "\n" + "2. No");

                if (int.Parse(Console.ReadLine()) == 1)

                {

                    townsDisplay.AddTown();

                }

                else

                {

                    Console.Write("Enter existing town: ");

                    townName = Console.ReadLine();

                }

            }

            competitor.TownId = townsBusiness.GetTownByName(townName).Id;

            Console.WriteLine("Do your competitor have a club?" + "\n" + "1.Yes" + "\n" + "2.No");

            if (int.Parse(Console.ReadLine()) == 1)

            {

                Console.Write("Enter Competitor Club Name: ");

                string clubName = Console.ReadLine();

                if (clubsBusiness.GetClubByName(clubName) == null)

                {

                    Console.WriteLine($"There is no club with the name {clubName}");

                    Console.WriteLine("Do you want to create new club?");

                    Console.WriteLine("1. Yes" + "\n" + "2. No");

                    if (int.Parse(Console.ReadLine()) == 1)

                    {

                        clubsDisplay.AddClub();

                    }

                    else

                    {

                        Console.WriteLine("Enter existing club: ");

                        clubName = Console.ReadLine();

                    }

                }

                competitor.ClubId = clubsBusiness.GetClubByName(clubName).Id;

            }

            else

            {

                competitor.ClubId = null;

            }

            Console.WriteLine("Do your competitor have a coach?" + "\n" + "1.Yes" + "\n" + "2.No");

            if (int.Parse(Console.ReadLine()) == 1)

            {

                Console.Write("Enter Competitor Coach Name: ");

                string coachName = Console.ReadLine();

                if (coachesBusiness.GetCoachByName(coachName) == null)

                {

                    Console.WriteLine($"There is no coach with the name {coachName}");

                    Console.WriteLine("Do you want to create new coach?");

                    Console.WriteLine("1. Yes" + "\n" + "2. No");

                    if (int.Parse(Console.ReadLine()) == 1)

                    {

                        coachesDisplay.AddCoach();

                    }

                    else

                    {

                        Console.WriteLine("Enter existing coach: ");

                        coachName = Console.ReadLine();

                    }

                }

                competitor.CoachId = coachesBusiness.GetCoachByName(coachName).Id;

            }

            else

            {

                competitor.CoachId = null;

            }

            Console.Write("Enter Competitor Sport Name: ");

            string sportName = Console.ReadLine();

            competitor.SportId = sportsBusiness.GetSportByName(sportName).Id;

            return competitor;

        }

    }

}

* + **CountriesDisplay.cs**

/// <summary>

/// Тhe <c>CountriesDisplay</c> class in Presentation.

/// This is the layer which is directly connected to CountriesBusiness.

/// </summary>

/// <remarks>

/// This class receive information from the user.

/// Then it is passed to CountriesBusiness.

/// </remarks>

class CountriesDisplay

{

private CountriesBusiness countriesBusiness;

private TownsBusiness towns;

/// <summary>

/// Constructor for CountriesDisplay class.

/// </summary>

/// <param name="context"></param>

public CountriesDisplay(OlympicGamesDBContext context)

{

countriesBusiness = new CountriesBusiness(context);

towns = new TownsBusiness(context);

}

/// <summary>

/// "Calls" method "GetAllCountries" from CountriesBusiness.

/// Then it shows all countries in table Countries.

/// </summary>

public void GetAllCountries()

{

Console.WriteLine("Countries: ");

List<Countries> countries = countriesBusiness.GetAllCountries();

if (countries.Count == 0)

{

Console.WriteLine("There are no countries in the table. ");

}

else

{

Console.WriteLine("Id" + new string(' ', 4) + "CountyName");

Console.WriteLine(new string('-', 35));

foreach (var country in countries)

{

string output = $"{country.Id}" + new string(' ', 6 - country.Id.ToString().Length)

+ $"{country.Name}" + new string(' ', 34 - country.Name.Length);

Console.WriteLine(output);

}

Console.WriteLine(new string('-', 35));

}

}

/// <summary>

/// After the user has inputed id, the program "Calls" method "GetCountryById" from CountriesBusiness.

/// Shows the Country who has this id.

/// </summary>

public void GetCountryById()

{

Console.WriteLine("Enter Country Id to fetch: ");

int id = int.Parse(Console.ReadLine());

Countries country = countriesBusiness.GetCountryById(id);

if (country != null)

{

PrintCountry(country);

}

else

{

Console.WriteLine($"There is no country with id = {id} in the table!");

}

}

/// <summary>

/// After the user has inputed name, the program "Calls" method "GetCountryByName" from CountriesBusiness.

/// Shows the Country who has this name.

/// </summary>

public void GetCountryByName()

{

Console.WriteLine("Enter Country Name to fetch: ");

string name = Console.ReadLine();

Countries country = countriesBusiness.GetCountryByName(name);

if (country != null)

{

PrintCountry(country);

}

else

{

Console.WriteLine($"There is no counrty with name = {name} in the table!");

}

}

private void PrintCountry(Countries country)

{

Console.WriteLine(new string('-', 40));

Console.WriteLine($"ID: {country.Id}");

Console.WriteLine($"Name: {country.Name}");

Console.WriteLine(new string('-', 40));

}

}

* + **SportsDisplay.sc**

/// <summary>

/// Тhe <c>SportsDisplay</c> class in Presentation.

/// This is the layer which is directly connected to SportsBusiness.

/// </summary>

/// <remarks>

/// This class receive information from the user.

/// Then it is passed to SportsBusiness.

/// </remarks>

class SportsDisplay

{

private SportsBusiness sportsBusiness;

/// <summary>

/// Constructor for SportsDisplay class.

/// </summary>

/// <param name="context"></param>

public SportsDisplay(OlympicGamesDBContext context)

{

sportsBusiness = new SportsBusiness(context);

}

/// <summary>

/// "Calls" method "GetAllSports" from SportsBusiness.

/// Then it shows all sports in table Sports.

/// </summary>

public void GetAllSports()

{

Console.WriteLine("Sports: ");

List<Sports> sports = sportsBusiness.GetAllSports();

if (sports.Count == 0)

{

Console.WriteLine("There are no sports in the table!");

}

else

{

Console.WriteLine("Id" + new string(' ', 6) + "Name");

Console.WriteLine(new string('-', 27));

foreach (var sport in sports)

{

string output = $"{sport.Id}" + new string(' ', 6 - sport.Id.ToString().Length)

+ $"{sport.Name}" + new string(' ', 21 - sport.Name.Length);

Console.WriteLine(output);

}

Console.WriteLine(new string('-', 27));

}

}

/// <summary>

/// After the user has inputed id, the program "Calls" method "GetSportById" from SportsBusiness.

/// Shows the Sport who has this id.

/// </summary>

public void GetSportById()

{

Console.Write("Enter Sport Id to fetch: ");

int id = int.Parse(Console.ReadLine());

Sports sport = sportsBusiness.GetSportById(id);

if (sport != null)

{

PrintSport(sport);

}

else

{

Console.WriteLine($"There is no sport with id = {id} in the table!");

}

}

/// <summary>

/// After the user has inputed name, the program "Calls" method "GetSportByName" from SportsBusiness.

/// Shows the Sport who has this name.

/// </summary>

public void GetSportByName()

{

Console.Write("Enter Sport Name to fetch: ");

string name = Console.ReadLine();

Sports sport = sportsBusiness.GetSportByName(name);

if (sport != null)

{

PrintSport(sport);

}

else

{

Console.WriteLine($"There is no sport with name = {name} in the table!");

}

}

private void PrintSport(Sports sport)

{

Console.WriteLine(new string('-', 40));

Console.WriteLine($"ID: {sport.Id}");

Console.WriteLine($"Name: {sport.Name}");

Console.WriteLine(new string('-', 40));

}

}

* + **TownsDisplay.cs**

/// <summary>

/// Тhe <c>TownsDisplay</c> class in Presentation.

/// This is the layer which is directly connected to TownsBusiness.

/// </summary>

/// <remarks>

/// This class receive information from the user.

/// Then it is passed to TownsBusiness.

/// </remarks>

public class TownsDisplay

{

private TownsBusiness townsBusiness;

private CountriesBusiness countriesBusiness;

/// <summary>

/// Constructor for TownsDisplay class.

/// </summary>

/// <param name="context"></param>

public TownsDisplay(OlympicGamesDBContext context)

{

townsBusiness = new TownsBusiness(context);

countriesBusiness = new CountriesBusiness(context);

}

/// <summary>

/// "Calls" method "GetAllTowns" from TownsBusiness.

/// Then it shows all towns in table Towns.

/// </summary>

public void GetAllTowns()

{

Console.WriteLine("Towns: ");

List<Towns> towns = townsBusiness.GetAllTowns();

if (towns.Count == 0)

{

Console.WriteLine("There are no towns in the table!");

}

else

{

Console.WriteLine("Id" + new string(' ', 4)

+ "Name" + new string(' ', 28) + "Country");

Console.WriteLine(new string('-', 68));

foreach (var town in towns)

{

var country = countriesBusiness.GetCountryById(town.CountryId);

string output = $"{town.Id}" + new string(' ', 6 - town.Id.ToString().Length)

+ $"{town.Name}" + new string(' ', 28 - town.Name.Length)

+ $"{country.Name}" + new string(' ', 34 - country.Name.Length);

Console.WriteLine(output);

}

Console.WriteLine(new string('-', 68));

}

}

/// <summary>

/// After the user has inputed id, the program "Calls" method "GetTownById" from TownsBusiness.

/// Shows the Town who has this id.

/// </summary>

public void GetTownById()

{

Console.WriteLine("Enter Town Id to fetch: ");

int id = int.Parse(Console.ReadLine());

Towns town = townsBusiness.GetTownById(id);

if (town != null)

{

PrintTown(town);

}

else

{

Console.WriteLine($"There is no town with id = {id} in the table!");

}

}

/// <summary>

/// After the user has inputed name, the program "Calls" method "GetTownByName" from TownsBusiness.

/// Shows the Town who has this name.

/// </summary>

public void GetTownByName()

{

Console.WriteLine("Enter Town Name to fetch: ");

string name = Console.ReadLine();

Towns town = townsBusiness.GetTownByName(name);

if (town != null)

{

PrintTown(town);

}

else

{

Console.WriteLine($"There is no town with Name = {name} in the table!");

}

}

private void PrintTown(Towns town)

{

Console.WriteLine(new string('-', 40));

Console.WriteLine($"ID: {town.Id}");

Console.WriteLine($"Name: {town.Name}");

Console.WriteLine(new string('-', 40));

}

/// <summary>

/// Makes the user to input data about the town.

/// Passes the information to TownsBusiness, using the method "AddTown"

/// </summary>

public void AddTown()

{

Towns town = new Towns();

Console.Write("Enter name: ");

town.Name = Console.ReadLine();

Console.Write("Enter Country Name: ");

string countryName = Console.ReadLine();

town.CountryId = countriesBusiness.GetCountryByName(countryName).Id;

townsBusiness.AddTown(town);

Console.WriteLine($"New town successfully added!");

}

/// <summary>

/// Finds the town wished to be updated.

/// Makes the user to enter the new information.

/// Then passes it to TownsBusiness, using the method "UpdateTown".

/// </summary>

public void UpdateTown()

{

Console.Write("Enter Id: ");

int id = int.Parse(Console.ReadLine());

Towns town = townsBusiness.GetTownById(id);

if (town != null)

{

Console.Write("Enter new name: ");

town.Name = Console.ReadLine();

Console.Write("Enter country name: ");

town.CountryId = countriesBusiness.GetCountryByName(Console.ReadLine()).Id;

townsBusiness.UpdateTown(town);

}

else

{

Console.WriteLine("Town not found!");

}

Console.WriteLine("Town successfully updated!");

}

/// <summary>

/// Finds the town wished to be deleted.

/// Passes the information to TownsBusiness, using the method "DeleteTownById".

/// </summary>

public void DeleteTownById()

{

Console.Write("Enter id: ");

int id = int.Parse(Console.ReadLine());

if (townsBusiness.GetTownById(id) == null)

{

Console.WriteLine($"There is no town with ID = {id} in the table!");

}

else

{

townsBusiness.DeleteTownById(id);

Console.WriteLine("Done!");

}

}

}

* + **Display.cs**

using Microsoft.EntityFrameworkCore.Query.SqlExpressions;

using Olympiad\_\_project\_code.Models;

using System;

using System.Collections.Generic;

using System.Text;

namespace Olympiad\_\_project\_code.Presentation

{

    /// <summary>

    /// Тhe main <c>Display</c> class in Presentation.

    /// This is the layer which is directly connected to the user.

    /// </summary>

    /// <remarks>

    /// This class receive information from the user.

    /// Then it is passed to one of the other displays.

    /// </remarks>

    class Display

    {

        private static OlympicGamesDBContext context = new OlympicGamesDBContext();

        private static CountriesDisplay countriesDisplay = new CountriesDisplay(context);

        private static TownsDisplay townsDisplay = new TownsDisplay(context);

        private static SportsDisplay sportsDisplay = new SportsDisplay(context);

        private static CoachesDisplay coachesDisplay = new CoachesDisplay(context);

        private static CompetitorsDisplay competitorsDisplay = new CompetitorsDisplay(context);

        private static ClubsDisplay clubsDisplay = new ClubsDisplay(context);

        public static void wait(double x)

        {

            DateTime t = DateTime.Now;

            DateTime tf = DateTime.Now.AddSeconds(x);

            while (t < tf)

            {

                t = DateTime.Now;

            }

        }

        private static int numberTable = -1;

        /// <summary>

        /// Receive information from the user.

        /// According to the input the program will continue with specific method.

        /// </summary>

        public static void Input()

        {

            do

            {

                ShowMenu();

                try

                {

                    numberTable = int.Parse(Console.ReadLine());

                }

                catch

                {

                    Console.WriteLine();

                    Console.WriteLine("Invalid command!");

                    Console.WriteLine("Try again...");

                    wait(4);//IN SECONDS

                    Console.Clear();

                    continue;

                }

                switch (numberTable)

                {

                    case 1:

                        ListOnlyAvailableOpperations();

                        break;

                    case 2:

                        ListOnlyAvailableOpperations();

                        break;

                    case 3:

                        ListAllOpperations();

                        break;

                    case 4:

                        ListAllOpperations();

                        break;

                    case 5:

                        ListAllOpperations();

                        break;

                    case 6:

                        ListAllOpperations();

                        break;

                    default:

                        break;

                }

            } while (numberTable != 7);

        }

        /// <summary>

        /// Shows on the screen all the sports included in the Olypmic Games.

        /// </summary>

        private static void ShowMenu()

        {

            Console.WriteLine(new string('-', 40));

            Console.WriteLine(new string(' ', 10) + "Olympic Games Rio 2016");

            Console.WriteLine(new string('-', 40));

            Console.WriteLine("1. Sports");

            Console.WriteLine("2. Countries");

            Console.WriteLine("3. Towns");

            Console.WriteLine("4. Competitors");

            Console.WriteLine("5. Coaches");

            Console.WriteLine("6. Clubs");

            Console.WriteLine("0. Exit entry");

            Console.Write("Enter the number of the choosen table: ");

        }

        /// <summary>

        /// Shows to the screen all the operations the user can do with the chosen table.

        /// </summary>

        /// <remarks>

        /// According to the user's answer the program "calls" specific method.

        /// </remarks>

        private static void ListAllOpperations()

        {

            Console.Clear();

            Console.WriteLine($"Number of chosen table: {numberTable}");//проверка и да се изведе името на таблицата

            Console.WriteLine("1. List all entries");

            Console.WriteLine("2. Show entry by Id");

            Console.WriteLine("3. Show entry by  Name");

            Console.WriteLine("4. Add new entry");

            Console.WriteLine("5. Update entry");

            Console.WriteLine("6. Delete entry by ID");

            Console.WriteLine("0. Return");

            Console.WriteLine();

            Console.Write("Enter the number of the operation: ");

            int operation2 = -1; // = int.Parse(Console.ReadLine());

            try

            {

                operation2 = int.Parse(Console.ReadLine());

            }

            catch

            {

                Console.WriteLine();

                Console.WriteLine("Invalid command");

                Console.WriteLine("Try again...");

                wait(4);//IN SECONDS

                Console.Clear();

               // continue;

            }

            switch (operation2)

            {

                case 1:

                    GetAllEntries(numberTable);

                    break;

                case 2:

                    GetEntryById();

                    break;

                case 3:

                    GetEntryByName();

                    break;

                case 4:

                    AddNewEntry();

                    break;

                case 5:

                    UpdateEntry();

                    break;

                case 6:

                    DeleteEntry();

                    break;

                default:

                    break;

            }

        }

        /// <summary>

        /// Shows to the screen all the operations the user can do with the chosen table.

        /// </summary>

        /// <remarks>

        /// According to the user's answer the program "calls" specific method.

        /// The difference here is that there are less operations available.

        /// </remarks>

        private static void ListOnlyAvailableOpperations()

        {

            Console.Clear();

            Console.WriteLine($"Number of chosen table: {numberTable}");//проверка и да се изведе името на таблицата

            Console.WriteLine("1. List all entries");

            Console.WriteLine("2. Show entry by Id");

            Console.WriteLine("3. Show entry by Name");

            Console.WriteLine("0. Return");

            Console.WriteLine();

            Console.Write("Enter the number of the operation: ");

            int operation = -1;// int.Parse(Console.ReadLine());

            try

            {

                operation = int.Parse(Console.ReadLine());

            }

            catch

            {

                Console.WriteLine();

                Console.WriteLine("Invalid command");

                Console.WriteLine("Try again...");

                wait(4);//IN SECONDS

                Console.Clear();

                // continue;

            }

            switch (operation)

            {

                case 1:

                    GetAllEntries(Display.numberTable);

                    break;

                case 2:

                    GetEntryById();

                    break;

                case 3:

                    GetEntryByName();

                    break;

                default:

                    break;

            }

        }

        /// <summary>

        /// According to the table the user chose to work with in the begining the information is passed to one of the other displays.

        /// </summary>

        /// <param name="numberTable">The number of the table which the user chose.</param>

        private static void GetAllEntries(int numberTable)

        {

            Console.Clear();

            switch (numberTable)

            {

                case 1:

                    sportsDisplay.GetAllSports();

                    break;

                case 2:

                    countriesDisplay.GetAllCountries();

                    break;

                case 3:

                    townsDisplay.GetAllTowns();

                    break;

                case 4:

                    competitorsDisplay.GetAllCompetitors();

                    break;

                case 5:

                    coachesDisplay.GetAllCoaches();

                    break;

                case 6:

                    clubsDisplay.GetAllClubs();

                    break;

                default:

                    break;

            }

            Console.WriteLine();

            Console.WriteLine("Press any key to clear the screen..."); Console.ReadKey(); Console.Clear();

        }

        /// <summary>

        /// According to the table the user chose to work with in the begining the information is passed to one of the other displays.

        /// </summary>

        /// <remarks>The method gets all data in the table.</remarks>

        private static void AddNewEntry()

        {

            Console.Clear();

            if (numberTable == 3)

            {

                townsDisplay.AddTown();

            }

            else if (numberTable == 4)

            {

                competitorsDisplay.AddCompetitor();

            }

            else if (numberTable == 5)

            {

                coachesDisplay.AddCoach();

            }

            else if (numberTable == 6)

            {

                clubsDisplay.AddClub();

            }

            Console.WriteLine();

            Console.WriteLine("Press any key to clear the screen..."); Console.ReadKey(); Console.Clear();

        }

        /// <summary>

        /// According to the table the user chose to work with in the begining the information is passed to one of the other displays.

        /// </summary>

        /// <remarks>The method gets data by name.</remarks>

        private static void GetEntryByName()

        {

            Console.Clear();

            switch (numberTable)

            {

                case 1:

                    sportsDisplay.GetSportByName();

                    break;

                case 2:

                    countriesDisplay.GetCountryByName();

                    break;

                case 3:

                    townsDisplay.GetTownByName();

                    break;

                case 4:

                    competitorsDisplay.GetCompetitorByName();

                    break;

                case 5:

                    coachesDisplay.GetCoachByName();

                    break;

                case 6:

                    clubsDisplay.GetClubByName();

                    break;

                default:

                    break;

            }

            Console.WriteLine();

            Console.WriteLine("Press any key to clear the screen..."); Console.ReadKey(); Console.Clear();

        }

        /// <summary>

        /// According to the table the user chose to work with in the begining the information is passed to one of the other displays.

        /// </summary>

        /// <remarks>The method gets data by id.</remarks>

        private static void GetEntryById()

        {

            Console.Clear();

            switch (numberTable)

            {

                case 1:

                    sportsDisplay.GetSportById();

                    break;

                case 2:

                    countriesDisplay.GetCountryById();

                    break;

                case 3:

                    townsDisplay.GetTownById();

                    break;

                case 4:

                    competitorsDisplay.GetCompetitorById();

                    break;

                case 5:

                    coachesDisplay.GetCoachById();

                    break;

                case 6:

                    clubsDisplay.GetClubById();

                    break;

                default:

                    break;

            }

            Console.WriteLine();

            Console.WriteLine("Press any key to clear the screen..."); Console.ReadKey(); Console.Clear();

        }

        /// <summary>

        /// According to the table the user chose to work with in the begining the information is passed to one of the other displays.

        /// </summary>

        /// <remarks>The method updates data.<remarks>

        private static void UpdateEntry()

        {

            Console.Clear();

            if (numberTable == 3)

            {

                townsDisplay.UpdateTown();

            }

            else if (numberTable == 4)

            {

                competitorsDisplay.UpdateCompetitor();

            }

            else if (numberTable == 5)

            {

                coachesDisplay.UpdateCoach();

            }

            else if (numberTable == 6)

            {

                clubsDisplay.UpdateClub();

            }

            Console.WriteLine();

            Console.WriteLine("Press any key to clear the screen..."); Console.ReadKey(); Console.Clear();

        }

        /// <summary>

        /// According to the table the user chose to work with in the begining the information is passed to one of the other displays.

        /// </summary>

        /// <remarks>The method deletes data.</remarks>

        private static void DeleteEntry()

        {

            Console.Clear();

            if (numberTable == 3)

            {

                townsDisplay.DeleteTownById();

            }

            else if (numberTable == 4)

            {

                competitorsDisplay.DeleteCompetitorById();

            }

            else if (numberTable == 5)

            {

                coachesDisplay.DeleteCoachById();

            }

            else if (numberTable == 6)

            {

                clubsDisplay.DeleteClubById();

            }

            Console.WriteLine();

            Console.WriteLine("Press any key to clear the screen..."); Console.ReadKey(); Console.Clear();

        }

    }

}