using System;

using System.Collections;

using System.Collections.Generic;

using System.Collections.ObjectModel;

namespace SportsClub

{

// Enum for male/female

public enum Gender { Male, Female }

// abstract cannot be instantiated (protected constructor)

public abstract class SportsPlayer

{

// auto-implemented properties

public String Name { get; set; }

public int Age { get; set; }

public Gender Gender { get; set; }

// constructor

protected SportsPlayer(String name, int age, Gender gender)

{

this.Name = name;

this.Age = age;

this.Gender = gender;

}

// inherited member from Object

public override string ToString()

{

return "Name: " + Name + " Age: " + Age + " Gender: " + Gender;

}

}

// various positions for a football player

public enum SoccerPosition { Goalkeeper, Defender, Midfielder, Striker }

public class SoccerPlayer : SportsPlayer

{

public SoccerPosition Position { get; set; }

// default constructor

public SoccerPlayer() : this("", 0, Gender.Male, SoccerPosition.Defender)

{

// chainage

}

// constructor

public SoccerPlayer(String name, int age, Gender gender, SoccerPosition position) : base(name, age, gender)

{

this.Position = position;

}

// inherited member from Object

public override string ToString()

{

return base.ToString() + " Position: " + Position;

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// custom equality comparison for .Equals, used by Contains() on List<>

public override bool Equals(Object obj)

{

SoccerPlayer player = obj as SoccerPlayer;

if (player == null)

{

return false;

}

// must match all 4 attributes to be equal

if ((player.Name == this.Name) && (player.Age == this.Age) && (player.Gender == this.Gender) && (player.Position == this.Position))

{

return true;

}

else

{

return false;

}

}

// needed if overriding Equals

public override int GetHashCode()

{

// return a hash (key) value for this object

return Name.GetHashCode() + Age.GetHashCode() + Gender.GetHashCode() + Position.GetHashCode();

}

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// a team of players

// enumerable type: can find specifics in an collection

public class SoccerTeam : IEnumerable

{

// minimum and no age limit for team

// SoccerTeam.MinAge

public const int MinAge = 5;

public const int NoAgeLimit = Int32.MaxValue;

// each team has a name, gender and an age limit

public String TeamName { get; set; }

public Gender TeamGender { get; set; }

private int ageLimit;

// player collection

private List<SoccerPlayer> players;

// constructor

public SoccerTeam(String teamName, Gender teamGender, int ageLimit)

{

this.TeamName = teamName;

this.TeamGender = teamGender;

this.AgeLimit = ageLimit;

this.players = new List<SoccerPlayer>();

}

// read only property to get players collection

public Collection<SoccerPlayer> Players

{

get

{

return new Collection<SoccerPlayer>(players);

}

}

// read/write property for ageLimit for a team

public int AgeLimit

{

get

{

return ageLimit;

}

set

{

if (value >= MinAge)

{

this.ageLimit = value;

}

else

{

throw new ArgumentException("Exception: Age Limit for a Team must be >= " + MinAge);

}

}

}

// indexer property, return a player with specified name

public SoccerPlayer this[String playerName]

{

get

{

SoccerPlayer player = null;

Boolean found = false;

for (int i = 0; i < players.Count; i++)

{

if (String.Compare(players[i].Name, playerName, StringComparison.OrdinalIgnoreCase) == 0) // ignore case in comparison

{

found = true;

player = players[i];

}

}

if (found)

{

return player;

}

else

{

throw new ArgumentException("Exception: Player " + playerName + " is not in the soccer team " + TeamName);

}

}

}

// iterate over players collection

public IEnumerator GetEnumerator()

{

foreach (SoccerPlayer player in players)

{

yield return player;

}

}

// add a player to the team, if not already in the team

public void AddPlayer(SoccerPlayer player)

{

// empty team

if (players == null)

{

players.Add(player);

}

else

{

// using the contains method to check if player is in squad

if (players.Contains(player))

{

throw new ArgumentException("Exception: player " + player.Name + " is already in the team");

}

else

{

// add the plater

if (player.Gender == TeamGender)

{

// player is less than or equal to the age limit

if (player.Age <= AgeLimit)

{

players.Add(player);

}

else

{

throw new ArgumentException("Exception: player " + player.Name + " is too old for team " + TeamName);

}

}

else

{

throw new ArgumentException("Exception: player " + player.Name + " is " + player.Gender + " while team is " + TeamGender);

}

}

}

}

}

}