Code Output:

A screen shot of a computer

Description automatically generated

UML Diagram:

A screenshot of a cell phone

Description automatically generated

**using** **System**;

**namespace** **FinalProject**

{

**public** **class** **Location**

{

**private** int \_posX;

**private** int \_posY;

**public** int PosX

{

**get** => \_posX;

**set** => \_posX = **value**;

}

**public** int PosY

{

**get** => \_posY;

**set** => \_posY = **value**;

}

**public** Location() { }

**public** Location(int posX, int posY)

{

**this**.PosX = posX;

**this**.PosY = posY;

}

**public** (int, int) Position()

{

**return** (PosX, PosY);

}

**public** **override** string ToString()

{

**return** ($"Location: ({PosX},{PosY})");

}

}

}

**using** **System**;

**using** **System.Text.RegularExpressions**;

**namespace** **FinalProject**

{

**public** **class** **Temperature**

{

**private** double \_degree = 100;

**private** char \_scale = 'f';

**public** double Degree

{

**get** => \_degree;

**set** => \_degree = **value**;

}

**public** char Scale

{

**get** => \_scale;

**set** => \_scale = char.ToLower(**value**);

}

**public** Temperature() { }

**public** Temperature(double degree, char scale)

{

**this**.Degree = degree;

**this**.Scale = scale;

}

**public** **void** TemperatureConversion(char scale)

{

scale = char.ToLower(scale);

double degree;

**switch** (scale)

{

**case** 'c':

**if** (Scale == 'f')

{

degree = (Degree - 32) \* (5 / 9);

Console.WriteLine($"{this.ToString()} --> {degree} °{scale}");

}

**else**

{

Console.WriteLine(**this**.ToString());

}

**break**;

**case** 'f':

**if** (Scale == 'c')

{

degree = (Degree \* (9 / 5)) + 32;

Console.WriteLine($"{this.ToString()} --> {degree} °{scale}");

}

**else**

{

Console.WriteLine(**this**.ToString());

}

**break**;

**default**:

Console.WriteLine($"[-] {scale} > not valid scale.");

**break**;

}

}

**public** **override** string ToString()

{

**return** ($"Temperature: {Degree}°{Scale}");

}

}

}

**using** **System**;

**namespace** **FinalProject**

{

**public** **class** **Pool**

{

**public** Temperature Temperature { **get**; **set**; }

**public** Location Location { **get**; **set**; }

**public** **static** int Count { **get**; **set**; }

**public** char Identifier { **get**; **set**; }

**public** Pool()

{

Console.Write($"Pool count before creation: {Count} ;\n");

**this**.Temperature = **new** Temperature();

**this**.Location = **new** Location();

Count++;

Console.Write($" count after creation: {Count}\n");

}

**public** Pool(Location location, char id)

{

Console.Write($"Pool count before creation: {Count} ;\n");

**this**.Temperature = **new** Temperature();

**this**.Location = location;

**this**.Identifier = id;

Count++;

Console.Write($" count after creation: {Count}\n");

}

**public** Pool(Temperature temperature, Location location, char id)

{

Console.Write($"Pool count before creation: {Count} ;\n");

**this**.Temperature = temperature;

**this**.Location = location;

**this**.Identifier = id;

Count++;

Console.Write($" count after creation: {Count}\n");

}

**public** **static** double GetDistance(Location l1, Location l2)

{

var (x1, y1) = l1.Position();

var (x2, y2) = l2.Position();

double distance = (Math.Sqrt(Math.Pow((x2 - x1), 2) + Math.Pow((y2 - y1), 2)));

**return** Math.Round(distance, 2);

}

**public** **override** string ToString()

{

**return** ($"Pool {Identifier}: {Temperature}\t{Location}");

}

}

}

**using** **System**;

**using** **System.Collections.Generic**;

**namespace** **FinalProject**

{

**class** **Driver**

{

**static** **void** Main(string[] args)

{

Header();

*// Setup list containing all 7 pools from handout*

Message("Instantiating pools.", ConsoleColor.Green);

List<Pool> myPools = **new** List<Pool>();

myPools.Add(**new** Pool(**new** Location(4, 8), 'A'));

myPools.Add(**new** Pool(**new** Location(1, 3), 'B'));

myPools.Add(**new** Pool(**new** Location(4, 2), 'C'));

myPools.Add(**new** Pool(**new** Location(13, 1), 'D'));

myPools.Add(**new** Pool(**new** Location(12, 9), 'E'));

myPools.Add(**new** Pool(**new** Location(10, 5), 'F'));

myPools.Add(**new** Pool(**new** Location(6, 6), 'G'));

Pool closestPool = myPools[0];

Message("Done instantiating pools.\n", ConsoleColor.Green);

*// Write out all pools in list*

Message("Listing all pools on map.", ConsoleColor.Green);

**foreach** (Pool pool **in** myPools)

{

Console.WriteLine(pool);

}

*/\**

*Choose coordinate for start location.*

*Set start location to current location (which changes)*

*Create two lists, unvisited pools; visitied pools.*

*Initialize all poools as unvisited.*

*Find shortest path from start/current location to unvisited pool*

*Remove that pool from unvisited, add to visited, and set as current location*

*Repeat until there are no more pools in unvisited pools.*

*\*/*

Location startLocation = **new** Location(0, 0);

Location currentLocation = startLocation;

List<Pool> unvisitedPools = **new** List<Pool>(myPools);

List<Pool> visitedPools = **new** List<Pool>();

double minDistance;

Console.ForegroundColor = ConsoleColor.Yellow;

Console.WriteLine("\n[+] Determining pool visit path route...");

**while** (unvisitedPools.Count > 0)

{

minDistance = 100;

**foreach** (Pool pool **in** unvisitedPools)

{

double dist = Pool.GetDistance(currentLocation, pool.Location);

**if** (dist <= minDistance)

{

minDistance = dist;

closestPool = pool;

}

}

currentLocation = closestPool.Location;

unvisitedPools.Remove(closestPool);

visitedPools.Add(closestPool);

}

Console.WriteLine("[+] Determined optimal path route.");

Console.WriteLine("[+] Maintenance on route and changing temp...");

Console.WriteLine("[+] Maintenance will take route:");

Console.ResetColor();

Console.Write($"Start" + startLocation.Position());

Random rand = **new** Random();

**foreach** (Pool pool **in** visitedPools)

{

pool.Temperature.Degree = rand.Next(98, 104) + Math.Round(rand.NextDouble(), 1);

Console.Write(" → " + pool.Identifier);

}

*// Output pool in order of visits with altered temperature*

Console.WriteLine(); Console.WriteLine();

Message("Updated pool map after maintenance visit.", ConsoleColor.Green);

**foreach** (Pool pool **in** visitedPools)

{

Console.WriteLine(pool);

}

}

**public** **static** **void** Message(string msg, ConsoleColor color)

{

Console.ForegroundColor = color;

Console.WriteLine($"[+] {msg}");

Console.ResetColor();

}

**public** **static** **void** Header()

{

Console.WriteLine("-------------------------------------");

Console.WriteLine("| FINAL PROJECT |");

Console.WriteLine("-------------------------------------");

}

}

}