* איך לעצור תהליכון
* להיכנס שוב לשאילתת נוסע, או אחרי יצירת קו ונסיעה
* תהליכון ב BL?
* לחיצה אחת במקום כפולה
* רקע כפתורי התחלה
* כל טיפוס יכלול תיעוד ע"י ///
* כל הגדרת פעולה (ז"א לא כולל מימושים של פעולות ממשק) תכלול תיעוד ע"י ///
* תסדירים ב-PL
* שימוש במשאבים (Resources)

ב-BLIMP:

* בגרסה הסופית אין להשתמש בלולאת foreach במקום שניתן להשתמש ב-LINQ
* חובה לכלול לפחות 4 שאילתות LINQtoObject
* חובה לכלול לפחות 4 ביטויי למבדה
* בשאילתות LINQ חובה להשתמש לפחות פעם אחת ב-let, ב-select new, בקיבוץ (grouping), במיון
* DAL: יתכן שחלק מהישויות לא זקוקות לחלק מפעולות CRUD לפי לוגיקת המערכת שהסטודנטים בונים - אין צורך לכתוב פונקציות שלא צריך אותן
* במימושים של הממשק IDal אסור לבצע כל חישוב או בדיקה לוגית למעט בדיקת שלמות הנתונים (למשל בדיקת הימצאות אובייקט שנדרש למוחקו או לעדכנו, או אי הימצאות אובייקט שנדרש להוסיפו)
* מספרים רצים ומידע נוסף על קונפיגורציה ינוהלו בעזרת קובץ XML של קונפיגורציה המתעדכן אוטומטית בכל עדכון (יצירה של מספר רץ חדש)
* מידע על **תחנת אוטובוס** כולל תחנת סיום
* אם ברצונכם לכפות עצירת הסימולטור גם באמצע שניית השינה של פועל הרקע - עליכם להפסיק את השינה ע"י הפעלת פעול ()Interrupt של תהליכון פועל הרקע. על מנת לעשות זאת - עליכם בתחילת פועל הרקע (לפני הלולאה) יהיה לשמור בצד את ההפניה לאובייקט התהליכון.

בסוף:

* בונוסים אפשריים: (אוטובוס) + סימון למחיקה (לא כולל תחנות עוקבות) + שיבוט + הצפנה
* לתעד בונוסים
* לתעד תוכנית
* לעבור על ההוראות של התרגיל
* למחוק קבצים מיותרים

//#region DrivingBuses

//public void addDrivingBus(DrivingBus drivingBus)

//{

// try

// {

// getBus(drivingBus.LicensePlate); // check if the bus exists

// drivingBus.ThisSerial = DataSource.serial++;

// DataSource.DrivingBuses.Add(drivingBus.Clone());

// }

// catch (BusException ex)

// {

// throw new BusException(ex.Message);

// }

//}

//public void removeDrivingBus(DrivingBus drivingBus)

//{

// DrivingBus d = DataSource.DrivingBuses.Find(item => item.ThisSerial == drivingBus.ThisSerial && item.LicensePlate == drivingBus.LicensePlate && item.Line == drivingBus.Line && item.Start == drivingBus.Start);

// if (d == null)

// throw new BusException("The driving bus does not exist.");

// DataSource.DrivingBuses.Remove(d); // remove the old driving bus

//}

//public void updateDrivingBus(DrivingBus drivingBus)

//{

// removeDrivingBus(drivingBus); // remove the old driving bus

// DataSource.DrivingBuses.Add(drivingBus.Clone()); // add the updated driving bus

//}

//public DrivingBus getDrivingBus(int thisSerial, string licensePlate, int line, DateTime start)

//{

// DrivingBus drivingBus = DataSource.DrivingBuses.Find(item => item.ThisSerial == thisSerial && item.LicensePlate == licensePlate && item.Line == line && item.Start == start);

// if (drivingBus == null)

// return null;

// return drivingBus.Clone();

//}

//public IEnumerable<DrivingBus> GetDrivingBuses()

//{

// return from item in DataSource.DrivingBuses

// select item.Clone();

//}

//public IEnumerable<DrivingBus> GetDrivingBuses(Predicate<DrivingBus> condition)

//{

// return from item in DataSource.DrivingBuses

// where condition(item)

// select item.Clone();

//}

//#endregion

מ-BLIMP:

#region DrivingBuses

/// <summary>

/// Func that converts driving bus of BO to driving bus of DO

/// </summary>

/// <param name="drivingBus">driving bus of BO</param>

/// <returns>driving bus of DO</returns>

DrivingBus convertToDrivingBusDO(BO.DrivingBus drivingBus)

{

return new DrivingBus()

{

ThisSerial = drivingBus.ThisSerial,

Line = drivingBus.Line,

LicensePlate = drivingBus.LicensePlate,

ActualStart = drivingBus.ActualStart,

Start = drivingBus.Start,

PreviousStationID = drivingBus.PreviousStationID,

PreviousStationTime = drivingBus.PreviousStationTime,

NextStationTime = drivingBus.NextStationTime

};

}

/// <summary>

/// Func that converts driving bus of DO to driving bus of BO

/// </summary>

/// <param name="drivingBus">driving bus of DO</param>

/// <returns>driving bus of BO</returns>

BO.DrivingBus convertToDrivingBusBO(DrivingBus drivingBus)

{

return new BO.DrivingBus()

{

LicensePlate = drivingBus.LicensePlate,

Line = drivingBus.Line,

Start = drivingBus.Start,

ThisSerial = drivingBus.ThisSerial,

ActualStart = drivingBus.ActualStart,

PreviousStationID = drivingBus.PreviousStationID,

PreviousStationTime = drivingBus.PreviousStationTime,

NextStationTime = drivingBus.NextStationTime

};

}

public void addDrivingBus(BO.DrivingBus drivingBus)

{

try

{

dal.addDrivingBus(convertToDrivingBusDO(drivingBus));

}

catch (BusException ex)

{

throw new BO.BusException(ex.Message, ex);

}

}

public void removeDrivingBus(BO.DrivingBus drivingBus)

{

try

{

dal.removeDrivingBus(convertToDrivingBusDO(drivingBus));

}

catch (BusException ex)

{

throw new BO.BusException(ex.Message, ex);

}

}

public void updateDrivingBus(BO.DrivingBus drivingBus)

{

try

{

dal.updateDrivingBus(convertToDrivingBusDO(drivingBus));

}

catch (BusException ex)

{

throw new BO.BusException(ex.Message, ex);

}

}

public BO.DrivingBus getDrivingBus(int thisSerial, string licensePlate, int line, DateTime start)

{

try

{

return convertToDrivingBusBO(dal.getDrivingBus(thisSerial, licensePlate, line, start));

}

catch (BusException ex)

{

throw new BO.BusException(ex.Message, ex);

}

}

public IEnumerable<BO.DrivingBus> GetDrivingBuses()

{

try

{

return from drivingBus in dal.GetDrivingBuses()

select convertToDrivingBusBO(drivingBus);

}

catch (BusException ex)

{

throw new BO.BusException(ex.Message, ex);

}

}

public IEnumerable<BO.DrivingBus> GetDrivingBuses(Predicate<BO.DrivingBus> condition)

{

try

{

return from item in GetDrivingBuses()

where condition(item)

select item;

}

catch (BO.BusException ex)

{

throw new BO.BusException(ex.Message);

}

}

#endregion

public class Bus : INotifyPropertyChanged

{

static private Random rand = new Random(DateTime.Now.Millisecond);

public event PropertyChangedEventHandler PropertyChanged;

#region

private string licensePlate; public string LicensePlate

{

get { return licensePlate; }

private set { licensePlate = value; }

}

private DateTime startOfWork; public DateTime StartOfWork

{

get { return startOfWork; }

private set { startOfWork = value; }

}

private DateTime lastService; public DateTime LastService

{

get { return lastService; }

set

{

lastService = value;

if (PropertyChanged != null)

PropertyChanged(this, new PropertyChangedEventArgs("LastService"));

}

}

private int totalKms; public int TotalKms

{

get { return totalKms; }

private set

{

totalKms = value;

if (PropertyChanged != null)

PropertyChanged(this, new PropertyChangedEventArgs("TotalKms"));

}

}

private int kmsSinceFuel; public int KmsSinceFuel

{

get { return kmsSinceFuel; }

set

{

kmsSinceFuel = value;

if (PropertyChanged != null)

PropertyChanged(this, new PropertyChangedEventArgs("KmsSinceFuel"));

}

}

private int kmsSinceService; public int KmsSinceService

{

get { return kmsSinceService; }

set

{

kmsSinceService = value;

if (PropertyChanged != null)

PropertyChanged(this, new PropertyChangedEventArgs("KmsSinceService"));

}

}

private State status; public State Status

{

get { return status; }

set

{

status = value;

if (PropertyChanged != null)

PropertyChanged(this, new PropertyChangedEventArgs("Status"));

}

}

private bool canBeFueled; public bool CanBeFueled

{

get { return canBeFueled; }

private set

{

canBeFueled = value;

if (PropertyChanged != null)

PropertyChanged(this, new PropertyChangedEventArgs("CanBeFueled"));

}

}

private bool canBeServiced; public bool CanBeServiced

{

get { return canBeServiced; }

private set

{

canBeServiced = value;

if (PropertyChanged != null)

PropertyChanged(this, new PropertyChangedEventArgs("CanBeServiced"));

}

}

private string time; public string Time

{

get { return time; }

set

{

time = value;

if (PropertyChanged != null)

PropertyChanged(this, new PropertyChangedEventArgs("Time"));

}

}

#endregion

/// <summary>

/// constructor

/// change dates and length if they are wrong

/// </summary>

/// <param name="start">date of start</param>

/// <param name="service">date of last service</param>

/// <param name="licensePlate">license plate number</param>

/// <param name="totalKms">total kms since start</param>

/// <param name="kmsSinceFuel">total kms since the last fuel</param>

/// <param name="kmsSinceService">total kms since the last service</param>

public Bus(DateTime start, DateTime service, string licensePlate, int totalKms, int kmsSinceFuel = 0, int kmsSinceService = 0)

{

if (service > DateTime.Now)

service = DateTime.Now.Date;

if (start > service)

start = service;

StartOfWork = start;

LastService = service;

if (totalKms < 0)

throw new BusException("Negative distance of drive is invalid.");

if (totalKms < kmsSinceFuel)

kmsSinceFuel = 0;

if (totalKms < kmsSinceService)

kmsSinceService = 0;

TotalKms = totalKms;

KmsSinceFuel = kmsSinceFuel;

KmsSinceService = kmsSinceService;

LicensePlate = setLicensePlate(licensePlate);

Status = setState();

setCanBeFueled();

setCanBeServiced();

Time = "";

}

/// <summary>

/// set the license plate number

/// throw exceptions if the input is wrong

/// </summary>

/// <param name="value">string of license plate number</param>

/// <returns>license plate number</returns>

private string setLicensePlate(string value)

{

if (value.Length < 7 || value.Length > 8) // wrong length

throw new BusException("Wrong length of license plate number.");

int num = int.Parse(value);

string tmp = Convert.ToString(num);

if (StartOfWork.Year < 2018 && tmp.Length == 7) // 7 digits

{

tmp = tmp.Insert(2, "-");

tmp = tmp.Insert(6, "-");

}

else if (StartOfWork.Year > 2018 && tmp.Length == 8) // 8 digits

{

tmp = tmp.Insert(3, "-");

tmp = tmp.Insert(6, "-");

}

else // length does not fit the year

throw new BusException("Length of license plate number does not fit the year.");

return tmp;

}

/// <summary>

/// set the bus status: canDrive, cannotDrive, driving, gettingFueled, gettingServiced

/// </summary>

/// <returns>bus status</returns>

public State setState()

{

TimeSpan timeSinceLastTreat = DateTime.Now - LastService;

if (timeSinceLastTreat.TotalDays >= 365 || KmsSinceService >= 20000 || KmsSinceFuel >= 1200)

return State.cannotDrive;

return State.canDrive;

}

/// <summary>

/// check if the bus can be fueled

/// </summary>

public void setCanBeFueled()

{

if (KmsSinceFuel >= 800 && (Status == State.canDrive || Status == State.cannotDrive))

CanBeFueled = true;

else

CanBeFueled = false;

}

/// <summary>

/// check if the bus can be serviced

/// </summary>

public void setCanBeServiced()

{

if ((KmsSinceService >= 19500 || (DateTime.Now - LastService).TotalDays >= 350) && (Status == State.canDrive || Status == State.cannotDrive))

CanBeServiced = true;

else

CanBeServiced = false;

}

/// <summary>

/// fuel the bus

/// each refueling takes 2 real hours = 12 unreal seconds

/// </summary>

public void fuel()

{

Status = State.gettingFueled;

setCanBeServiced();

setCanBeFueled();

List<object> parameters = new List<object>();

parameters.Add(12);

parameters.Add(this);

parameters.Add((float)-1); // mark refueling

//new MainWindow().worker.RunWorkerAsync(parameters);

}

/// <summary>

/// service the bus

/// each service takes 24 real hours = 144 unreal seconds

/// </summary>

public void service()

{

Status = State.gettingServiced;

setCanBeServiced();

setCanBeFueled();

List<object> parameters = new List<object>();

parameters.Add(144);

parameters.Add(this);

parameters.Add((float)-2); // mark service

//new MainWindow().worker.RunWorkerAsync(parameters);

}

/// <summary>

/// print the licence plate number of the bus

/// </summary>

/// <returns>licence plate number</returns>

public override string ToString() { return LicensePlate; }

}

public class DrivingBus : INotifyPropertyChanged

{

public event PropertyChangedEventHandler PropertyChanged;

private static int serial = 1;

#region

private int thisSerial; public int ThisSerial

{

get { return thisSerial; }

private set { thisSerial = value; }

}

private string licensePlate; public string LicensePlate

{

get { return licensePlate; }

private set { licensePlate = value; }

}

private int line; public int Line

{

get { return line; }

private set { line = value; }

}

private DateTime start; public DateTime Start

{

get { return start; }

private set { start = value; }

}

private DateTime actualStart; public DateTime ActualStart

{

get { return actualStart; }

private set

{

actualStart = value;

if (PropertyChanged != null)

PropertyChanged(this, new PropertyChangedEventArgs("ActualStart"));

}

}

private int previousStationID; public int PreviousStationID

{

get { return previousStationID; }

private set

{

previousStationID = value;

if (PropertyChanged != null)

PropertyChanged(this, new PropertyChangedEventArgs("PreviousStationID"));

}

}

private DateTime previousStationTime; public DateTime PreviousStationTime

{

get { return previousStationTime; }

private set

{

previousStationTime = value;

if (PropertyChanged != null)

PropertyChanged(this, new PropertyChangedEventArgs("PreviousStationTime"));

}

}

private DateTime nextStationTime; public DateTime NextStationTime

{

get { return nextStationTime; }

private set

{

nextStationTime = value;

if (PropertyChanged != null)

PropertyChanged(this, new PropertyChangedEventArgs("NextStationTime"));

}

}

#endregion

public DrivingBus(string licensePlate, int line, DateTime start)

{

ThisSerial = serial++;

LicensePlate = licensePlate;

Line = line;

Start = start;

ActualStart = DateTime.Now;

PreviousStationID = 0;

PreviousStationTime = DateTime.Now;

NextStationTime = DateTime.Now;

}

public DrivingBus(int thisSerial, string licensePlate, int line, DateTime start, DateTime actualStart, int previousStationID, DateTime previousStationTime, DateTime nextStationTime)

{

ThisSerial = thisSerial;

LicensePlate = licensePlate;

Line = line;

Start = start;

ActualStart = actualStart;

PreviousStationID = previousStationID;

PreviousStationTime = previousStationTime;

NextStationTime = nextStationTime;

}

}

public class DrivingLine

{

#region

private int numberLine; public int NumberLine

{

get { return numberLine; }

private set { numberLine = value; }

}

private DateTime start; public DateTime Start

{

get { return start; }

private set { start = value; }

}

private int frequency; public int Frequency

{

get { return frequency; }

private set { frequency = value; }

}

private DateTime end; public DateTime End

{

get { return end; }

private set { end = value; }

}

#endregion

public DrivingLine(int numberLine, DateTime start, int frequency, DateTime end)

{

NumberLine = numberLine;

Start = start;

Frequency = frequency;

End = end;

}

}

public class Line : IComparable<Line>, INotifyPropertyChanged

{

public event PropertyChangedEventHandler PropertyChanged;

//static private Random rand = new Random(DateTime.Now.Millisecond);

private static int serial = 1;

#region

public ObservableCollection<LineStation> Path = new ObservableCollection<LineStation>();

private int thisSerial; public int ThisSerial

{

get { return thisSerial; }

private set { thisSerial = value; }

}

private int numberLine; public int NumberLine

{

get { return numberLine; }

private set { numberLine = value; }

}

private Regions region; public Regions Region

{

get { return region; }

set

{

region = value;

if (PropertyChanged != null)

PropertyChanged(this, new PropertyChangedEventArgs("Region"));

}

}

public int FirstStation

{

get { return Path.First().ID; }

}

public int LastStation

{

get { return Path.Last().ID; }

}

#endregion

/// <summary>

/// constructor - gets the first station

/// </summary>

/// <param name="firstStation">first station in path</param>

public Line(int numberLine, Regions region, LineStation firstStation)

{

ThisSerial = serial++;

NumberLine = numberLine;

Region = region;

Path.Add(firstStation);

}

/// <summary>

/// constructor - gets list of stations

/// </summary>

/// <param name="newPath">path</param>

public Line(int numberLine, Regions region, ObservableCollection<LineStation> newPath) : this(numberLine, region, newPath[0]) // call the first constructor

{

for (int i = 1; i < newPath.Count(); i++)

Path.Add(newPath[i]);

}

/// <summary>

/// copy constructor

/// </summary>

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

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

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

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

public Line(int thisSerial, int numberLine, Regions region, ObservableCollection<LineStation> path)

{

ThisSerial = thisSerial;

NumberLine = numberLine;

Region = region;

Path = path;

}

/// <summary>

/// ovarride about "ToString".

/// </summary>

/// <returns>string of: number of line, region of the activity of the line and the phat of the line in back and forth.</returns>

public override string ToString()

{

string descriptionOfBus = $"Line: {NumberLine}. Region: {Region}. Stations: ";

foreach (LineStation station in Path)

descriptionOfBus += station.ID + " -> ";

descriptionOfBus = descriptionOfBus.Remove(descriptionOfBus.Length - 4, 4); // remove the last " -> "

return descriptionOfBus;

}

/// <summary>

/// chack if the station exists in the path of the bus.

/// </summary>

/// <param name="stationID">number of station to check</param>

/// <returns>true if the station exists</returns>

public bool stopsAtStation(int stationID)

{

foreach (LineStation station in Path)

if (stationID == station.ID)

return true;

return false;

}

/// <summary>

/// Calculate travel distance between two stations

/// </summary>

/// <param name="FirstID">start station to calculate</param>

/// <param name="SecondID">end</param>

/// <returns>the travel distance between two stations</returns>

public double distanceBetweenTwoStations(int FirstID, int SecondID)

{

int fir = -1;

int sec = -1;

int i = 0;

double meters = 0;

foreach (LineStation station in Path)

{

if (FirstID == station.ID)

fir = i;

if (fir > i)

meters += station.LengthFromPreviousStations;

if (SecondID == station.ID)

{

sec = i;

break;

}

i++;

}

if (fir == -1 || sec == -1)

throw new LineException("one of the station is not exist or the stations not in the true order.");

return meters;

}

/// <summary>

/// Calculate travel time between two stations

/// </summary>

/// <param name="FirstID">start station to calculate</param>

/// <param name="SecondID">end</param>

/// <returns>the travel time between two stations</returns>

public int MinutesBetweenTwoStations(int FirstID, int SecondID)

{

int fir = -1;

int sec = -1;

int i = 0;

int minutes = 0;

foreach (LineStation station in Path)

{

if (FirstID == station.ID)

fir = i;

if (fir > i)

minutes += station.TimeFromPreviousStations;

if (SecondID == station.ID)

{

sec = i;

break;

}

i++;

}

if (fir == -1 || sec == -1)

throw new LineException("one of the station is not exist or the stations not in the true order.");

return minutes;

}

/// <summary>

/// calculate time of drive

/// </summary>

/// <returns>time of drive</returns>

private int durationDrive()

{

int minutes = 0;

foreach (LineStation station in Path)

minutes += station.TimeFromPreviousStations;

return minutes;

}

/// <summary>

/// compare time of travel of two lines

/// </summary>

/// <param name="secondBus">line compared to current line</param>

/// <returns>whether or not this bus drive longer time than the other bus</returns>

public int CompareTo(Line secondBus)

{

return durationDrive().CompareTo(secondBus.durationDrive());

}

}

public class LineStation : INotifyPropertyChanged

{

public event PropertyChangedEventHandler PropertyChanged;

#region

private int numberLine; public int NumberLine

{

get { return numberLine; }

private set { numberLine = value; }

}

private int id; public int ID

{

get { return id; }

private set { id = value; }

}

private int pathIndex; public int PathIndex

{

get { return pathIndex; }

set

{

pathIndex = value;

if (PropertyChanged != null)

PropertyChanged(this, new PropertyChangedEventArgs("PathIndex"));

}

}

private double lengthFromPreviousStations; public double LengthFromPreviousStations // meters

{

get { return lengthFromPreviousStations; }

set

{

lengthFromPreviousStations = value;

if (PropertyChanged != null)

PropertyChanged(this, new PropertyChangedEventArgs("LengthFromPreviousStations"));

}

}

private int timeFromPreviousStations; public int TimeFromPreviousStations // minutes

{

get { return timeFromPreviousStations; }

set

{

timeFromPreviousStations = value;

if (PropertyChanged != null)

PropertyChanged(this, new PropertyChangedEventArgs("TimeFromPreviousStations"));

}

}

#endregion

/// <summary>

/// constructor

/// </summary>

/// <param name="number">line number</param>

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

/// <param name="index">index in the stations path</param>

public LineStation(int number, int id, int index)

{

numberLine = number;

ID = id;

PathIndex = index;

lengthFromPreviousStations = 0;

timeFromPreviousStations = 0;

}

}

public class Station : INotifyPropertyChanged

{

public event PropertyChangedEventHandler PropertyChanged;

#region

private int id; public int ID

{

get { return id; }

private set { id = value; }

}

private string name; public string Name

{

get { return name; }

private set

{

name = value;

if (PropertyChanged != null)

PropertyChanged(this, new PropertyChangedEventArgs("Name"));

}

}

private double latitude; public double Latitude // קו רוחב

{

get { return latitude; }

private set

{

latitude = value;

if (PropertyChanged != null)

PropertyChanged(this, new PropertyChangedEventArgs("Latitude"));

}

}

private double longitude; public double Longitude // קו אורך

{

get { return longitude; }

private set

{

longitude = value;

if (PropertyChanged != null)

PropertyChanged(this, new PropertyChangedEventArgs("Longitude"));

}

}

#endregion

/// <summary>

/// constructor

/// </summary>

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

/// <param name="name">name of station</param>

/// <param name="latitude">latitude of station</param>

/// <param name="longitude">longitude of station</param>

public Station(int id, string name, double latitude, double longitude)

{

ID = id;

Name = name;

Latitude = latitude;

Longitude = longitude;

}

/// <summary>

/// describe the station's attributes

/// </summary>

/// <returns>string of code and location on the globe</returns>

public override string ToString()

{

return $"{ID} {Name} ({Latitude}°N, {Longitude}°E)";

}

}

int calculateDistance(BO.Station first, BO.Station second)

{

return (int)(new GeoCoordinate(first.Latitude, first.Longitude).GetDistanceTo(new GeoCoordinate(second.Latitude, second.Longitude)) \* 1.5);

}