# Exercise 4:

Examine the following code which is a partial implementation of an ItineraryManager with respect to good coding standards and design. Outline (bullet point list) any issues you can see in the code (if any), and what should be done to improve them.

/// <summary>

/// Provides capabilities for managing a customers itinerary.

/// </summary>

public class ItineraryManager

{

private readonly IDataStore \_dataStore;

private readonly IDistanceCalculator \_distanceCalculator;

public ItineraryManager()

{

\_dataStore = new SqlAgentStore(ConfigurationManager.ConnectionStrings["SqlDbConnection"].ConnectionString);

\_distanceCalculator = new GoogleMapsDistanceCalculator(ConfigurationManager.AppSettings["GoogleMapsApiKey"]);

}

Use Dependency Inject to replace it

public ItineraryManager(IDataStore datastore, IDistanceCalculator distanceCalculator)

{

\_dataStore = datastore;

\_distanceCalculator = distanceCalculator;

}

The new SqlAgentStore(ConfigurationManager.ConnectionStrings["SqlDbConnection"].ConnectionString) implementation should put into DataStore class, DataStore class inherit interface IDataStore

The new GoogleMapsDistanceCalculator(ConfigurationManager.AppSettings["GoogleMapsApiKey"]) implementation should put into DataStore class, DistanceCalculator class inherit interface IDistanceCalculator

/// <summary>

/// Calculates a quote for a customers itinerary from a provided list of airline providers.

/// </summary>

/// <param name="itineraryId">The identifier of the itinerary</param>

/// <param name="priceProviders">A collection of airline price providers.</param>

/// <returns>A collection of quotes from the different airlines.</returns>

public IEnumerable<Quote> CalculateAirlinePrices(int itineraryId, IEnumerable<IAirlinePriceProvider> priceProviders)

{

var itinerary = \_dataStore.GetItinaryAsync(itineraryId).Result;

if (itinerary == null)

throw new InvalidOperationException();

List<Quote> results = new List<Quote>();

Parallel.ForEach(priceProviders, provider =>

{

var quotes = provider.GetQuotes(itinerary.TicketClass, itinerary.Waypoints);

foreach (var quote in quotes)

results.Add(quote);

});

return results;

}

Update to below:

public Task<IEnumerable<Quote>> CalculateAirlinePrices(int itineraryId, IEnumerable<IAirlinePriceProvider> priceProviders)

{

var itinerary = await \_dataStore.GetItinaryAsync(itineraryId).Result;

if (itinerary == null)

throw new InvalidOperationException();

var results = new List<Quote>();

ParallelLoopResult result =

Parallel.ForEach(priceProviders, provider =>

{

var quotes = provider.GetQuotes(itinerary.TicketClass, itinerary.Waypoints);

foreach (var quote in quotes)

results.Add(quote);

});

return results;

}

/// <summary>

/// Calculates the total distance traveled across all waypoints in a customers itinerary.

/// </summary>

/// <param name="itineraryId">The identifier of the itinerary</param>

/// <returns>The total distance traveled.</returns>

public async Task<double> CalculateTotalTravelDistanceAsync(int itineraryId)

{

var itinerary = await \_dataStore.GetItinaryAsync(itineraryId);

if (itinerary == null)

throw new InvalidOperationException();

double result = 0;

for(int i=0; i<itinerary.Waypoints.Count-1; i++)

{

result = result + \_distanceCalculator.GetDistanceAsync(itinerary.Waypoints[i],  
 itinerary.Waypoints[i + 1]).Result;

result += await \_distanceCalculator.GetDistanceAsync(itinerary.Waypoints[i],  
 itinerary.Waypoints[i + 1]).Result;

}

return result;

}

/// <summary>

/// Loads a Travel agents details from Storage

/// </summary>

/// <param name="id">The id of the travel agent.</param>

/// <param name="updatedPhoneNumber">If set updates the agents phone number.</param>

/// <returns>The travel agent if located, otherwise null.</returns>

public TravelAgent FindAgent(int id, string updatedPhoneNumber)

{

var agentDao = \_dataStore.GetAgent(id);

if (agentDao == null)

return null;

if (!string.IsNullOrWhiteSpace(updatedPhoneNumber))

{

agentDao.PhoneNumber = updatedPhoneNumber;

\_dataStore.UpdateAgent(id, agentDao);

}

return Mapper.Map<TravelAgent>(agentDao);

}

}

Based on good coding standards and design, list the issues I found (please see yellow background color):

1. Constructor method should use dependency injection
2. CalculateAirlinePrices method

\_dataStore.GetItinaryAsync(itineraryId).Result is async method, should use await \_dataStore.GetItinaryAsync(itineraryId).Result;

Return type update to Task<IEnumerable<Quote>>

Use var results = new List<Quote>(); to repleace List<Quote> results = new List<Quote>();

Parallel.ForEach() will return ParallelLoopResult.

1. CalculateTotalTravelDistanceAsync method

\_distanceCalculator.GetDistanceAsync(itinerary.Waypoints[i],  
 itinerary.Waypoints[i + 1]).Result is async method, should use result += await \_distanceCalculator.GetDistanceAsync(itinerary.Waypoints[i],  
 itinerary.Waypoints[i + 1]).Result;

1. FindAgent(int id, string updatedPhoneNumber)

I didn’t find any issue in this method