# Answer 4

I see the following code issues in the code:

1. ItineraryManager should employ Inversion of Control pattern. Instances of IDatastore and IDistanceCalculator should be provided as constructor parameters
2. CalculateAirlinePrices method should be async and return type should be Task< IEnumerable<Quote>>
3. In CalculateAirlinePrices,.”GetItinaryAsync(itineraryId).Result” needs to be replaced with await call
4. In CalculateAirlinePrices, Parallel.ForEach requires concurrency collection instead of List
5. In CalculateTotalTravelDistanceAsync .GetDistanceAsync(itinerary.Waypoints[i],  
    itinerary.Waypoints[i + 1]).Result;

.Result to be replaced with await call

1. In CalculateTotalTravelDistanceAsync .GetDistanceAsync(itinerary.Waypoints[i],  
    itinerary.Waypoints[i + 1]).Result;

Waypoints[i + 1] causes index out of range exception

1. FindAgent violates Single Responsibility Prinicple and needs to be split into two methods

# 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"]);

}

/// <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;

}

/// <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;

}

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);

}

}