# Exercise 1:

Demonstrate your knowledge of asynchronous programming by creating a method that downloads three resources and aggregates the content length of all 3 responses. The caller should be able to cancel the operation at any time.

Notes / Assumptions:

* The method should be written as efficiently as possible.
* You can assume each resource you download is a string via a HTTP GET request.
* You can assume each resource exists. Eg. No error handling for HTTP responses.
* You can assume that each response returns all standard header you would expect from a normal HTTP rest request.

# Exercise 2:

A coupon management system stores a definition of a coupon and also every time a user redeems a coupon. A coupon consists of an id, a title, a start date, end date, maximum number of coupons per user and also a maximum number of coupons across all users. When a user redeems a coupon the system keeps track of the users identifier, the datetime when the redemption occurred and a unique code that is generated by the application.

The coupon management system is required to provide the following capabilities:

* Provide an active list of coupons.
* Determine if a consumer can redeem a coupons.
* Store redemptions as they occur.
* Provide reporting on the redemptions for a specific offer.

The coupon management system is expected to have the following capacity:

* Coupons up to 50000 unique coupons. Generally 100-500 active at any point in time.
* Redemptions – Upwards of 1 billion rows. Expect at least 1 million redemptions per day.

Requirements:

* Create a data model to represent the data in the above scenario. You are free to choose any storage mechanism you feel appropriate. The model should take into account the volume and expected operations into account.
* Define additional optimizations that you would apply to the basic data model. (For example if you were to choose SQL you might want indexes on certain columns) Note: Implementation/Code is not required.

# Exercise 3:

Write unit tests for the following class:

public class CouponManager

{

private readonly ILogger \_logger;

private readonly ICouponProvider \_couponProvider;

public CouponManager(ILogger logger, ICouponProvider couponProvider)

{

\_logger = logger ?? throw new ArgumentNullException(nameof(logger));

\_couponProvider = couponProvider ?? throw new ArgumentNullException(nameof(couponProvider));

}

public async Task<bool> CanRedeemCoupon(Guid couponId, Guid userId, IEnumerable<Func<Coupon, Guid, bool>> evaluators)

{

if (evaluators == null)

throw new ArgumentNullException(nameof(evaluators));

var coupon = await \_couponProvider.Retrieve(couponId);

if (coupon == null)

throw new KeyNotFoundException();

if (!evaluators.Any())

return true;

bool result = false;

foreach (var evaluator in evaluators)

result |= evaluator(coupon, userId);

return result;

}

}

# 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>

// 1. No Interface: Its good to have an interface implementation for DI & Unit test mockings

public class ItineraryManager

{

private readonly IDataStore \_dataStore;

private readonly IDistanceCalculator \_distanceCalculator;

public ItineraryManager()

{

//2. It’s always good to use DI(Dependency Injection) passed on the constructor for object creation.

//3. Also: Magic strings can be avoided by using string const (Ex: "SqlDbConnection", "GoogleMapsApiKey" etc)

\_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>

//4. This Method should be an **Async** Method

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

{

//5. Better to use **await** keyword here

[‘.Result’ is a bad practice since the accessing the property's get accessor blocks the calling thread until the asynchronous operation is complete]

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

if (itinerary == null)

throw new InvalidOperationException();

//6. It’s dangerous to use the List inside any concurrent processes because it is not thread safe.

// So we can use ‘BlockingCollection’ / ‘ConcurrentBag’ instead.

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;

//7. Appears like a logical issue here since we will be missing the last record.

// the below condition should be like “i<itinerary.Waypoints.Count”

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

{

//8. Same as #5. Better to use **await** keyword here instead of ‘.Result’

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);

}

//9. Automapper: Something wrong with mapper here.

// ‘Map’ is not a static method so, the .Map should be called with object instead

// Ex: \_mapper.Map (or) mapper.Map etc

return Mapper.Map<TravelAgent>(agentDao);

}

}