**Coding Problem**

Your solution will be evaluated based on: object oriented design and principles, code structure, readability, maintainability, and testability

Build an application that takes in a time of day and the types of dishes we want to eat, returning the foods to make.

Rules:

1. You must enter time of day as “morning” or “night”
2. You must enter a comma delimited list of dish types with at least one selection
3. The output must print food in the following order: entrée, side, drink, desert
4. There is no desert for morning meals
5. Input is not case sensitive
6. If invalid selection is encountered, display valid selections up to the error, then print error
7. In the morning, you can order multiple cups of coffee
8. At night, you can have multiple orders of potatoes
9. Except for the above rules, you can only order 1 of each dish type

**Dishes for Each time of day**

|  |  |  |
| --- | --- | --- |
| Dish Type | morning | night |
| 1 (entrée) | eggs | steak |
| 2 (side) | Toast | potato |
| 3 (drink) | coffee | wine |
| 4 (desert) | *Not Applicable* | cake |

**Sample Input and Output:**

Input: morning, 1, 2, 3

Output: eggs, toast, coffee

Input: morning, 2, 1, 3

Output: eggs, toast, coffee

Input: morning, 1, 2, 3, 4

Output: eggs, toast, coffee, error

Input: morning 1, 2, 3, 3, 3

Output: eggs, toast, coffee(x3)

Input: night, 1, 2, 3, 4

Output: steak, potato, wine, cake

Input: night, 1, 2, 2, 4

Output steak, potato(x2), cake

Input: night, 1, 2, 3, 5

Output: steak, potato, wine, error

Input: night, 1, 1, 2, 3, 5

Output: steak, error

**Database Problem**

|  |
| --- |
| **Status** |
| EntityId  StatusDate  Status |

Given the following database structure:

|  |  |  |  |
| --- | --- | --- | --- |
| **Entity** |  |  | |
| EntityId |  | |  | |
| EntityName |  |  | |
| … |  |  | |

|  |
| --- |
| **Balances** |
| EntityId  BalanceDate  Balance |

1. Write a query that returns a list of all balances for entity named “ABCD” and, for each date shown, if available, the status on that date [defined as the “most recent up to but not exceeding the balance date”].

Example: Given data below:

|  |  |  |
| --- | --- | --- |
| **EntityID** | **BalanceDate** | **Balance** |
| 1 | 5/1/2010 | 100.00 |
| 1 | 4/1/2010 | 50.00 |
| 1 | 3/1/2010 | 75.00 |
| 1 | 2/1/2010 | 85.00 |
| 2 | 5/1/2010 | 110.00 |

|  |  |
| --- | --- |
| **EntityID** | **EntityName** |
| 1 | ABCD |
| 2 | XYZ |

|  |  |  |
| --- | --- | --- |
| **EntityID** | **StatusDate** | **Status** |
| 1 | 5/29/2010 | A |
| 1 | 4/16/2010 | B |
| 1 | 4/2/2010 | C |
| 1 | 2/26/2010 | D |
| 2 | 5/1/2010 | B |

the following output would be expected:

|  |  |  |  |
| --- | --- | --- | --- |
| **EntityName** | **Date** | **Balance** | **Status** |
| XYZ | 5/1/2010 | 110.00 | B |
| ABCD | 5/1/2010 | 100.00 | B |
| ABCD | 4/1/2010 | 50.00 | D |
| ABCD | 3/1/2010 | 75.00 | D |
| ABCD | 2/1/2010 | 85.00 | <null> |

1. Write a query that returns average balance and most recent status for every entity

Example: given the data example in step 1, the following output would be expected:

|  |  |  |  |
| --- | --- | --- | --- |
| **EntityName** | **Status** | **As Of** | **Avg Balance** |
| XYZ | B | 5/1/2010 | 110.00 |
| ABCD | A | 5/29/2010 | 77.50 |
|  |  |  |  |