|  |  |
| --- | --- |
| **Subject** | Data [Structure and Design Patterns - ITE-5231-RNA](https://learn.humber.ca/webapps/blackboard/execute/courseMain?course_id=_177738_1) |
| **Professor** | Sarvananthan Jeganathan |
| **Final Project** | **Final Project (Group)** |
| **Submitted** | 10 August 2021 |
| **Student Names (Group 6)** | Ved Chandurkar & Oreoluwa Lawal |
| **Humber Student IDs** | N01436129 (Ved) & N01452264 (Oreoluwa) |
| **Semester II** | Summer 2021 |

**Final Project Details recorded after completion**

|  |  |
| --- | --- |
| **Final Project:** Ved Chandurkar & Oreoluwa Lawal | **Final Project - Joe’s Café** |
| **The Application:**   1. This “**Joe’s Café**” application would be a C# .Net application that would cater to ordering the different beverages and dishes available in a typical coffee shop. 2. Graphical User Interface for this application written as a C# .Net Forms, which will have several forms to serve the needs of the **Joe’s Café**. 3. In the code of this C# .Net application we showcase following 5 design patterns.    1. Singleton – BeverageDisplayRecordFactory which gives out the instances of the BeveragePanel prototype objects. This is required to be a single instance that can server the whole application.    2. Factory Method – BeverageDisplayRecordFactory that gives out the group of form controls that has to be staged for the Coffee Ordering form    3. Prototype – The instances of BeveragePanelProtytype class are a group of dynamically created form controls (like labels, checkboxes, drop downs, buttons, images) that have to be made again an again and are made by cloning the prototype instance.    4. Observer – The Grand Total label, is an observer object that observes the subject which is the form controls that could change the cost of a coffee order.    5. Iterator – Finally the Pattern that everyone uses to loop over the aggregate values. 4. The class that sells out the different beverage products like Coffee & Tea, does give us an ample scope in creating the above-mentioned design patterns. 5. We also use the following data structures classes:    1. List (Typed Array List class - Growing array equivalent, that supports Generics)    2. Dictionary (Associate Array of Key Value pairs)    3. Sorted Dictionary (Associate array that is sorted on the Key Object’s chosen field)    4. Tuple (A structure that holds multiple values together and has access method for retrieving the fields)    5. Queue (First in First out data structure that is generally required in any software that serves the customers) 6. This **Joe’s Café** applicationis not too complex and not trivial either. This helps set up a perfect stage to demonstrate the use of five design patterns and five data structures in the real world. |

|  |  |
| --- | --- |
| **Final Project:** Ved Chandurkar & Oreoluwa Lawal | **Final Project – Application forms** |
| All the screens of the application:    **Details of the Design Patterns Implemented:**  **Following Design Patterns are used in the application**   * 1. Singleton – BeverageDisplayRecordFactory which gives out the instances of the BeveragePanel prototype objects. This is required to be a single instance that can server the whole application.   2. Factory Method – BeverageDisplayRecordFactory that gives out the group of form controls that has to be staged for the Coffee Ordering form   3. Prototype – The instances of BeveragePanelPrototype class are a group of dynamically created form controls (like labels, checkboxes, drop downs, buttons, images) that have to be made again an again and are made by cloning the prototype instance.   4. Observer – The Grand Total label, is an observer object that observes the subject which is the form controls that could change the cost of a coffee order.   5. Iterator – Finally the Pattern that everyone uses to loop over the aggregate values.   **Singleton Design Pattern implemented in the Application:**     1. BeverageDisplayRecordFactory class is a single class. 2. It is having a private static ‘instance’ variable. 3. The clients have to call the GetInstance() method to retrieve the single instance of this class. 4. The instance is created the first time the client calls the GetInstance() method – Delayed initialization. |

|  |  |
| --- | --- |
| **Final Project:** Ved Chandurkar & Oreoluwa Lawal **Continued . . .** | **Final Project – Factory Method and Prototype Design Patterns in the Application** |
| **Factory Method & Prototype Design Patterns**     1. BeverageDisplayRecordFactory is a single class 2. BeverageDisplayRecordFactory implements a factory method ‘GetBeverageDisplayRecord’. 3. GetBeverageDisplayRecord method returns the BeverageDisplayRecord cloned from the prototype. 4. ‘FrmCustomerScreen’ form instance is a client for this factory method. 5. Factory method uses BeveragePanelPrototype to clone the new instance that will be returned. |

|  |  |
| --- | --- |
| **Final Project:** Ved Chandurkar & Oreoluwa Lawal **Continued . . .** | **Final Project – Iterator Design Patterns in the Application** |
| **Iterator Design Pattern**     1. The AddonIterator implements IIterator interface 2. AddonCollection implements IAggregate interface 3. ProductDB and FrmCustomerScreen classes are clients of these two. |

**Document continues below . . .**

|  |  |
| --- | --- |
| **Final Project:** Ved Chandurkar & Oreoluwa Lawal **Continued . . .** | **Final Project – Observer Design Pattern implemented in the Application** |
| **Observer Design Pattern**     1. The control elements in BeveragePanelPrototype affect the total cost (Grand Total)). Hence BeveragePanelPrototype becomes the Observed (Subject). 2. On the other side GrandTotalObserver class is the Observer. 3. When the elements that affect the total cost change, the Subject calls the Notify() and the Grand Total on the screen gets updated. |

**Document continues below . . .**

|  |  |
| --- | --- |
| **Final Project:** Ved Chandurkar & Oreoluwa Lawal **Continued . . .** | **Final Project – List<T> & Dictionary<K, V> Data Structures used in the Application** |
| **List<Type> Data Structure**    **Dictionary<K, V> Data Structure** |

|  |  |
| --- | --- |
| **Final Project:** Ved Chandurkar & Oreoluwa Lawal **Continued . . .** | **Final Project – SortedDictionary<K, V> and Tuple<T1, T2> Data Structures used in the Application** |
| **SortedDictionary<K, V> And Tuple<T1, T2> Data Structures**    **Queue<T> and Tuple <T1, T2> Data Structures** |