**Iterator Pattern**

**Overview**

Iterator pattern provides a way to access elements of an aggregate object sequentially without exposing its underlying representation. Freeman Eric., Freeman Elisa. (2004). It allows traversing of elements of collections without exposing underlying representation ( list, stack, tree etc). Usually with one type of data structure such as array or array-list, it would be easier to iterate in data structure but iterator pattern comes handy when you would want to hide logic of data objects structure such as list, stack and tree but want to safely iterate in all those data structures through a pattern.

Iterator interface declares operations required for traversing a collection while collection interface lists a list of collection methods which

* It provides uniform way to access different collections of object
* An iterator can cycle uniformly through all different collections

**Benefits**

* In a codebase where multiple types of data objects are stored and are to be iterated through without further modifying data structures themselves iterator pattern would be very useful.
* With iterator pattern the normal behaviour of iterating through collection of object is extracted and put into a single object called iterator so that collection remains untouched time and again.
* An iterator object provides primary method for fetching elements of collection.
* A polymorphic code can be referred to each collection as they will implement same interface
* When new object is added , new iterator could be added without having to change the collection or the client.

**Problem/Scenario**

A pizza shop sells two types of pizzas; one is vegetarian pizza and the other is non-vegetarian pizzas. Pizza house wants to separate two types of pizzas in separate menus; one as vegetarian and other as non-vegetarian pizzas but pizza house owner does want to separate both menus, and behind the scenes code must address his/her choice of this by implementing separate objects/list/array list/array etc. as required and iterate through separately. That is main intention when menu item is omitted or added developers would only touch respective codebase/data than looking at whole codebase and get confused.

* Every menu item includes name, a description of ingredients and price.

**Before Refactor**

Before pattern is implemented pizza menu would have vegetarian and non-vegetarian objects to store array, and array list of pizzas which show different implementation of data structures as this design pattern is to show variation in data structure. This structure would just normally iterate through those array, array list from main class by accessing methods which exist in vegetarian and non-vegetarian pizzas and pizzas from menu.

**After Refactor**

After refactor, we would create Iterator interface which is implemented by independent iterators defined by VegetarianPizzaIterator and NonVegetarianPizzaIterator and independent objects which store array list and array of vegetarian pizza and non-vegetarian pizza. Iterators are accessed by independent menu iterator and print list of pizza available in menu from vegetarian and non-vegetarian pizzas.

With refactor it adds independent iterator classes such as vegetarian-pizza iterator and non-vegetarian pizza iterator on top of respective veg/non-veg pizza objects which can iterate through with respective objects when needed. Benefit of such methodology would be, suppose if codebase got non-veg pizza and veg. pizza is added later or removed, then we would only worry about pizza that is added or removed for adding/removing iteration or do subsequent change in object without modifying core logic. It can solve quite complex problem in big codebase. It reduces time and effort of developers as well because they would not need to read existing code for long time and can separate concern of object/iteration properly.

* Implementation returns iterator interface. Clients don’t need to know how data is maintained inside vegetarian menu and non-vegetarian menu done by Dominos. It will just need to use iterator to step through the items in the menu.

Before/after link

<https://github.com/sppanday/S120-PRT583-Group-A/tree/feat/design-pattern/s260598-PandaySurendra/Sprint-3-Deliverables/Task017_Iterator_Pattern/PizzaHouseIteratorPattern/PizzaHouseIteratorPattern>

**UML Diagram**

Please find it inside relevant project UML in pattern folder

/IteratorPattern.png

Direct Access

<https://app.lucidchart.com/documents/edit/85d4043f-1c46-49fe-b7e8-77f0a9244e72/0_0>

**Reference**

Freeman Eric., Freeman Elisa. 2004. 1st ed. O’Reily Media Inc. 1995, Sebasttopo, CA