**Name:** Iterator

**Category:** Behavioral

**Description:** The Iterator design pattern is a simple pattern which allows a given data structure to be traversed without the underlying implementation being exposed. To accomplish this the pattern is mainly seen using an Iterator interface which allows for multiple different types of iterators to be created and used. The interface uses two abstract methods, one for retrieving the next data value and another to determine if there is another data value to retrieve. If the given data structure you want to iterate over was a binary tree made up of Profiles, then you would create a ProfileIterator that derives from the Iterator interface. Inside the ProfileIterator you would provide the implementation for how to traverse the Profile binary tree. This way you would only see the getNext() method and not how the iterator is getting the next value.

**When to use:** The main reason you would want to use the iterator method is when you want a single established way to traverse through data structures and would like to hide what the actual data structure is.

**Advantages:** The obvious advantage of using the iterator design pattern is disguising the underlying data structure while still being able to properly traverse through it.

**Disadvantages:** While the benefits of using the iterator design pattern is clear there are clear drawbacks such as the overhead of having to potentially create several different types of iterators for the different data structures being iterated through. There is also the problem of having to keep track of the state of the iterator and invoking its methods separately. All of this could lead to more management of the iterator, processing time, and memory concerns.