**Notes:**

* createIterator() should return a std::unique\_ptr<Iterator<T>> so that client code doesn’t need to manage memory manually.
* To support iteration for a new container, simply define a new IterableCollection subclass and implement its corresponding ConcreteIterator.

**When to use/ not to use:**

* Use the Iterator pattern to encapsulate the details of working with complex data structure
* Use the iterator pattern for the ability to traverse different Data Structures or when these types of structure are unknown beforehand
* Avoid using the iterator pattern if the data structure is simple or when initializing the iterator costs more than a direct recursive or indexed traversal.

**Pros**

* **Single Responsibility:** Traversal logic lives in separate iterator classes, keeping collections and client code focused.
* **Open/Closed:** New iterators or collection types can be added without modifying existing code.

**Cons**

* **Overkill for Simple Collections:** Writing iterator classes adds unnecessary complexity for basic data structures.
* **Efficiency Overhead:** Using an iterator may be slightly slower than a direct loop for specialized collections.