Iterator Pattern

Design Patterns

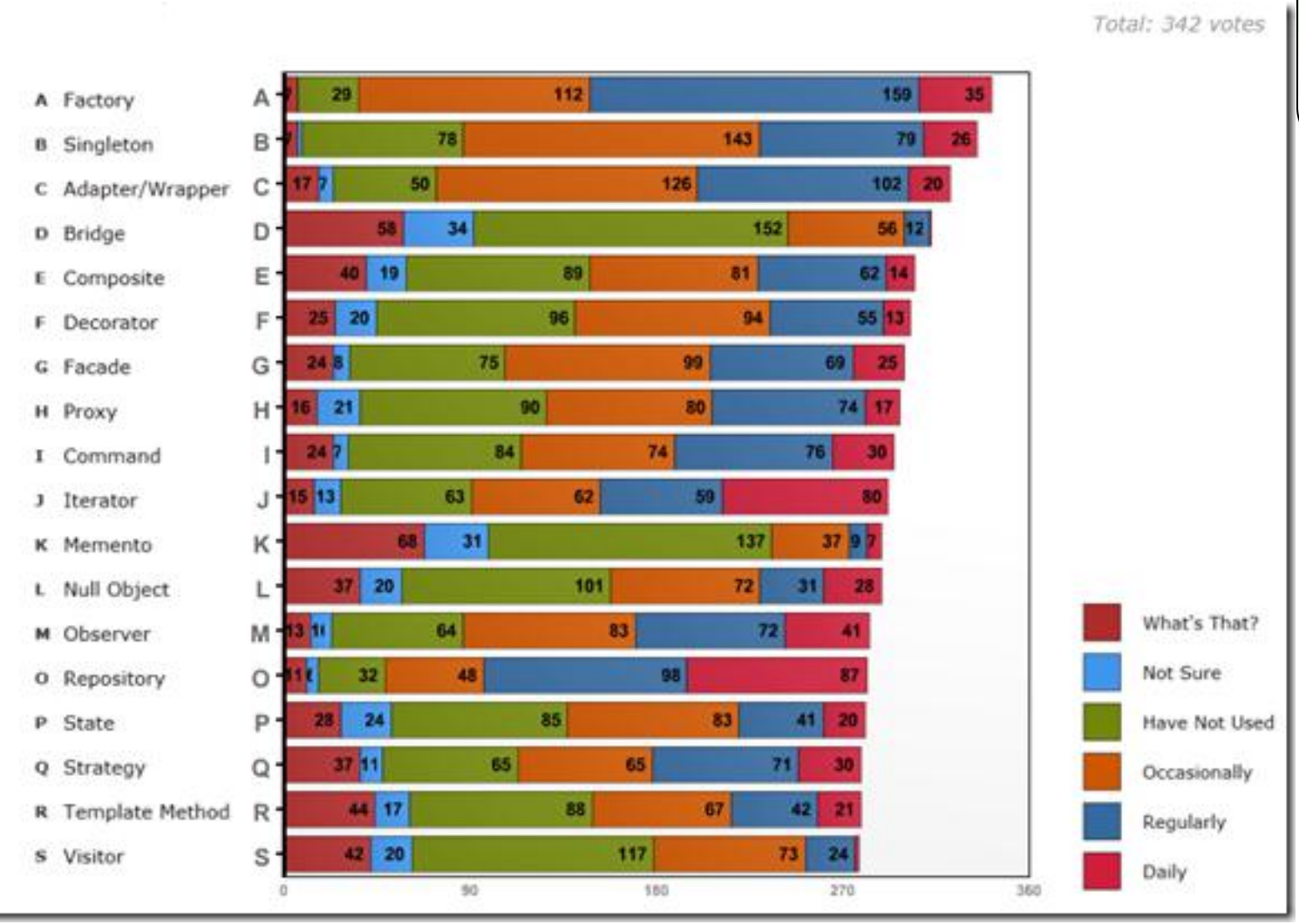
Motivating Example

Iterator Pattern

- You work with a variety of collection types
- You want to be able to traverse various kinds of collections without knowing about their internal structures
- You don't want to bloat every collection's interface with traversal operations
- The *iterator pattern* describes a way to access collections' members sequentially without violating encapsulation or SRP.

Design Pattern Poll Results

Iterator Pattern



http://stevesmithblog.com/blog/common-design-patterns-resources/

Without an Iterator

Iterator Pattern

(Forget about .NET's built-in Iterator support)

- Traversing an Array

```
var stocks = new string[] { "MSFT", "GOOG", "AAPL" };
for (int i = 0; i < stocks.Length; i++)
{
    Console.WriteLine(stocks[i]);
}
```

- Traversing another collection type

```
var stocks = new SuperCollection() { "MSFT", "GOOG", "AAPL" };
for (int i = 0; i < stocks.Count; i++)
{
    Console.WriteLine(stocks.Get(i));
}
```

- How can we abstract iteration so we can do it polymorphically?

Intent

Iterator Pattern

"Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation."

Design Patterns

The *iterator* pattern defines interfaces for the *aggregate* and the *iterator*, each of which must be implemented for each collection that is to support the pattern.

Also Known As: *Cursor*

Applicability

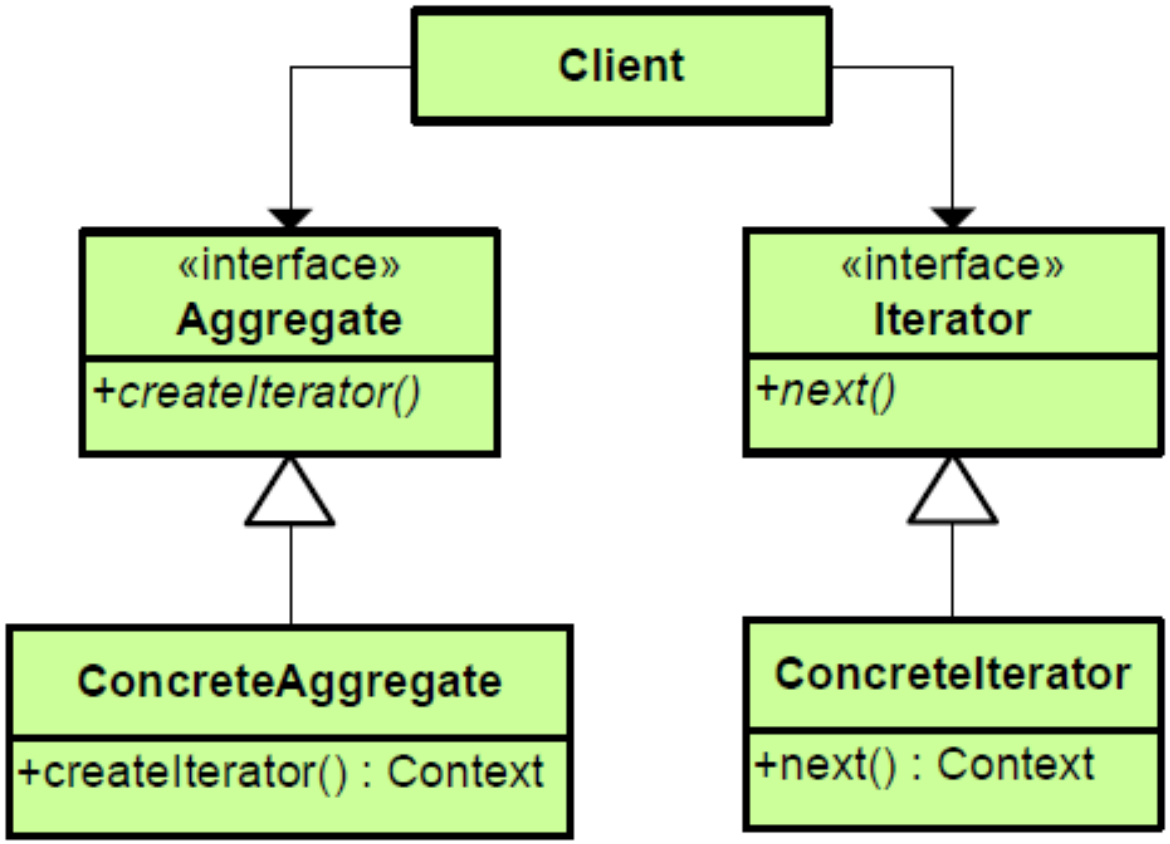
Iterator Pattern

Use the Iterator Pattern when:

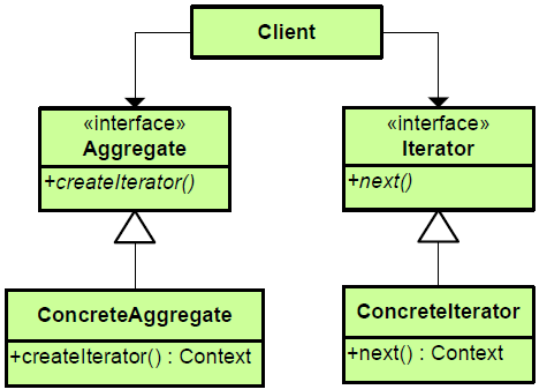
- **You need to traverse a collection**
- **You want to abstract the collection iteration logic**
  - Follow SRP and Don't Repeat Yourself (DRY)
  - Learn more in the Principles of Object Oriented Design course
- **You do not wish to break encapsulation and expose your collections' internal organization/design globally**

Structure

Iterator Pattern



How It Gets Used



- **Each collection type requires two implementations**
  - The Aggregate interface is implemented on the collection itself, providing a way to get an Iterator
    - This is an example of the *Abstract Factory* pattern
  - The Iterator interface must be implemented for the collection type in question
- **The Client retrieves an Iterator from the Aggregate**
- **The Client traverses the collection using the Iterator**

Iterators in C# and .NET

Iterator Pattern

- **Aggregate interface: IEnumerable**
- **Iterator interface: IEnumerator**

- **C# language feature: foreach**
  - Operates on any type implementing IEnumerable (including basic arrays)
  - Compare the following:

```
var stocks = new string[] { "MSFT", "GOOG", "AAPL" };
foreach (var stock in stocks)
{
    Console.WriteLine(stock);
}
```

```
var stocks = new string[] { "MSFT", "GOOG", "AAPL" };
var enumerator = stocks.GetEnumerator();
while(enumerator.MoveNext())
{
    Console.WriteLine(enumerator.Current);
}
```

- **LINQ**
- **C# *yield* keyword**

Collaboration

Iterator Pattern

- A concrete Aggregate implementation returns a specific, concrete Iterator
- A concrete Iterator implementation works with a particular collection
- Client accesses the iterator (enumerator) from the collection type (enumerable type)
- When using foreach or LINQ, enumerators never need to be referenced directly by client code

Consequences

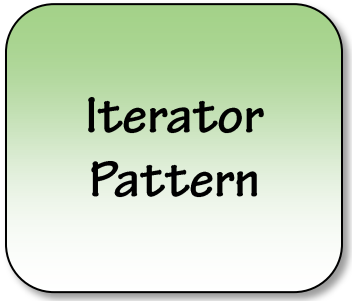
Iterator Pattern

- **Each Iterator implementation may traverse the aggregate in a different fashion.**
  - To change the algorithm used, simply create a new iterator instance.
- **Iterators reduce the surface area of the Aggregate interface, simplifying it.**
- **By separating iteration from the aggregate itself, more than one traversal operation can occur at the same time. Iterators keep track of their own traversal state.**

Demo

Adding .NET's Iterator Support
to a Custom Collection Type

Related Patterns



- **Factory**
  - Iterators are typically created via factory methods on the aggregate
  - Polymorphic iterators use factory methods to instantiate the appropriate iterator subclass

- **Composite**
  - Iterators can be used to recursively traverse composite structures, such as trees.

You can learn more about these patterns in the Pattern Library at PluralSight On Demand.

References

Iterator Pattern

- **Books**
  - Design Patterns, http://amzn.to/95q9ux
  - Design Patterns in C#, http://amzn.to/bqJgdU
  - Head First Design Patterns, http://amzn.to/aA4RS6

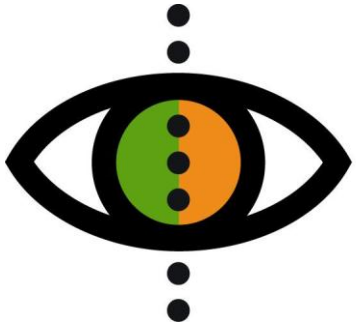
- **Online**
  - http://en.wikipedia.org/wiki/Iterator_pattern
  - http://www.codeproject.com/Articles/186188/Iterator-Design-Pattern.aspx
  - Discover the Design Patterns You're Already Using in the .NET Framework (MSDN Magazine)
    - http://bit.ly/cejAkX
  - http://stevesmithblog.com/blog/common-design-patterns-resources/

Summary

Iterator Pattern

- The Iterator pattern separates the logic of iterating an aggregate (a collection) from the aggregate itself
- Allows common aggregate operations to be done generically and via enhanced language features (foreach, yield, LINQ, etc.)
- Simplifies aggregate's interface by eliminating iteration methods
- Allows multiple iteration strategies to be implemented independent of the collection
- Allows multiple iterations to take place simultaneously

For more in-depth online developer training visit



pluralsight

see what you can learn

on-demand content from authors you trust

Blog: SteveSmithBlog.com

Twitter: @ardalis