Iterator
Iterator Intent

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

(Behavioral)
What do the Java Collection Framework iterators provide for you?

- Function to call to get iterator
- Iterator implements a standard interface
- Common approach to access elements in the collection
Iterator Structure

**Collection**
- `createIterator()`

**Client**

**Iterator**
- `first()`
- `next()`
- `isDone()`
- `currentItem()`

**List**
- `ListIterator`

**Stack**
- `StackIterator`
Iterators are classified along two dimensions.

- **Location of iterator**
  - Iterator is *embedded* in the collection object
  - Iterator is a *separate* class from the collection class

- **Location of iteration control**
  - Control is *external* to iterator, i.e. iterator client has control
  - Control is *internal* to iterator, i.e. iterator has control
One iterator characteristic is the location of the iterator class with respect to the collection class.

- The iterator definition can be in a class **separate** from the collection.
  - Good separation of concerns: traversal vs. maintenance of collection structure
  - Multiple traversal types (forward, backward, matching) without unneeded ones
  - Easier to do multiple traversals at the same time

- The iterator definition can be **embedded** within the collection class.
  - Preserves encapsulation of collection
Who controls iteration when you use the Java Collections Framework iterators?

- Consider the Java Collection Framework:

```java
Iterator iter = theCollection.iterator();

while( iter.hasNext() ) {
    process( iter.next() );
}
```

Internal or external?

Embedded or separate?
Internal iterators require the client to provide a processing function.

- Iterator has a function to perform iteration

  ```java
  public interface InternalIterator {
    public boolean iterate(Processor p) ;
  }
  ```

- Client requests that the iterator iterate through the collection and process each object

  ```java
  public interface Processor {
    public boolean process(Object o) ;
  }
  ```
The client has no iteration control with an internal iterator except to possibly terminate early.

```java
public class ACollectionIterator implements InternalIterator {
    private SomeCollection theCollection;

    public ACollectionIterator(SomeCollection c) {
        theCollection = c;
    }

    public boolean iterate(Processor p) {
        boolean result = true;

        Start at the beginning of theCollection

        while ( still elements && result) {
            if ( ! p.process( next element ) ) {
                result = false;
            }

            Move to the next element
        }
        return result;
    }
}
```
There are advantages and disadvantages with external and internal iterators.

- External iterators are more flexible
  - Example: Compare two lists for equality

- Internal iterators are easier to use
  - Iteration logic handled for client
Recursive collections, such as Composites, present special problems for iteration.

- Does the Java Collections Framework directly support internal or external iteration?
- Which form of iterator is easier to implement when traversing a recursive collection like a tree?
- Can you do the traversal with the other approach, and if so, what issues arise?
It is easier to iterate through a recursive collection with an internal iterator.

- **External iterator**
  - *iterator stores path for retreat (Hansel & Gretel)*
  - *composite provides back links for iterator to use (parent / sibling / children)*

- **Internal iterator**
  - *Traverse recursively – get backtrack for free*
  - *Get iterator for children of current element*
  - *Use null iterators at leaves.*
Placement of knowledge of how to do traversal involves trade-offs of O-O design principles.

- **Iterator determines next element**
  - *Here iterator needs to know structure of collection*
  - *Violates encapsulation of collection if a separate iterator*

- **Collection determines next element**
  - *Iterator is just a marker/cursor of where traversal left off*
  - *Traversal logic is in the collection even if a separate iterator*
  - *Complicates collection itself*
  - *Does not separate iteration from collection maintenance*