Iterator Pattern

Provide a way to access the elements of a collection sequentially without exposing its underlying representation
Iterator Solution

Java

LinkedList<String> strings = new LinkedList<String>;

code to add strings

for (String element : strings) {
    if (element.size % 2 == 0)
        System.out.println(element);
}

Iterator<String> list = strings.iterator();
while (list.hasNext()){
    String element = list.next();
    if (element.size % 2 == 0)
        System.out.println(element);
}

This is 1/2 the way to a good solution.
Ruby has a richer set of iterators than Java. Smalltalk, which inspired Ruby's iterators, has a richer set of iterators that Ruby. Perhaps the language that replaces Ruby will match the power that Smalltalk had 20 years ago.
Pattern Parts

Intent
Motivation
Applicability
Structure
Participants
Collaborations
Consequences
Implementation
Sample Code
Iterator Structure

```
return new Concreteliterator(this)
```
Iterator Structure & Names

Aggregate, ConcreteAggregate, Client, ConcretelIterator

Roles that classes will perform
Classes will not have those names

```
return new ConcretelIterator(this)
```
Issue - What is the big deal?

```
var numbers = new LinkedList();

code to add numbers

Iterator list = numbers.iterator();
while ( list.hasNext() ) {
    Integer a = (Integer) list.next();
    int b = a.intValue();
    if ((b % 2) == 0)
        System.out.println( x );
}
```

```
var numbers = new LinkedList();

code to add numbers

for (int k =0; k < numbers.size(); k++) {
    Integer a = (Integer) numbers.get(k);
    int b = a.intValue();
    if ((b % 2) == 0)
        System.out.println( x );
}
```

Java's Enumerations and iterators were awkward to use. C# pushed Sun to add better syntax.
Java's Enumerations and iterators were awkward to use. C# pushed Sun to add better syntax.
In Java 8

var numbers = new LinkedList();

code to add numbers

numbers.forEach(each ->
    if ((each % 2) == 0)
        System.out.println(each);});

var numbers = new LinkedList();

code to add numbers

numbers.stream()
    .filter(each -> 0 == (each % 2))
    .forEach(each ->
        System.out.println(each));
Issues - Concrete vs. Polymorphic Iterators

Concrete
Reader iterator = new StringReader( "cat" );
int c;
while (-1 != (c = iterator.read() ))
    System.out.println( (char) c );

Polymorphic
Vector listOfStudents = new ArrayList();
    // code to add students not shown
Iterator list = listOfStudents.iterator();
while ( list.hasNext() )
    System.out.println( list.next() );

Memory leak issue in C++, Why?
**Issue - Who Controls the Iteration?**

External (Active)

```java
var numbers = new LinkedList();

code to add numbers

Vector evens = new Vector();
Iterator list = numbers.iterator();
    while ( list.hasNext() ) {
        Integer a = (Integer) list.next();
        int b = a.intValue();
        if ((b % 2) == 0)
            evens.add(a);
    }
```

Internal (Passive)

```ruby
numbers = LinkedList.new

code to add numbers

evens = numbers.find_all { |element| element.even? }
```
Issue - Who Defines the Traversal Algorithm

Object being iterated

Iterator
**Issue - Robustness**

What happens when items are added/removed from the iteratee while an iterator exists?

Vector listOfStudents = new Vector();

// code to add students not shown

Iterator list = listOfStudents.iterator();
listOfStudents.add( new Student( "Roger") );

list.hasNext(); //What happens here?
Strategy Pattern
Favor Composition over Inheritance
Orderable List

Sorted
Reverse Sorted
Random
OrderableList

SortedList

ReverseList

RandomList
One size does not fit all
Issue 1 - Orthogonal Features

Order
- Sorted
- Reverse Sorted
- Random

Threads
- Synchronized
- Unsynchronized

Mutability
- Mutable
- Non-mutable
Change behavior at runtime

OrderableList x = new OrderableList();
x.makeSorted();
x.add(foo);
x.add(bar);
x.makeRandom();
Configure objects behavior at runtime
Strategy Pattern

class OrderableList {
    private Object[] elements;
    private Algorithm orderer;

    public OrderableList(Algorithm x) {
        orderer = x;
    }

    public void add(Object element) {
        elements = ordered.add(elements, element);
    }
}

Diagram:

- Algorithm
  - Sorted
  - Random
  - ReverseSorted
Structure

```
Context
  contextInterface()

Strategy
  algorithmInterface()

ConcreteStrategyA
  algorithmInterface()

ConcreteStrategyB
  algorithmInterface()
```
The algorithm is the operation

Context contains the data

How does this work?
Prime Directive
Data + Operations
How does Strategy Get the Data?

Pass needed data as parameters in strategy method

Give strategy object reference to context
Strategy extracts needed data from context
import java.awt.*;

class  FlowExample  extends Frame  {

    public FlowExample( int  width, int height ) {
        setTitle( "Flow Example" );
        setSize( width, height );
        setLayout( new FlowLayout( FlowLayout.LEFT) );

        for ( int label = 1; label < 10; label++ )
            add( new Button( String.valueOf( label ) ) );
        show();
    }

    public static void main( String  args[] ) {
        new  FlowExample( 175, 100 );
        new  FlowExample( 175, 100 );
    }
}
Example - Smalltalk Sort blocks

| list |
list := #( 1 6 2 3 9 5 ) asSortedCollection.
Transcript
  print: list;
  cr.
list sortBlock: [:x :y | x > y].
Transcript
  print: list;
  cr;
  flush.
Java Sorting

How to sort a Collection in Java?

ArrayList List method - sort(Comparator<? super E> c)

Create a subclass of Comparator

Pass in comparator object to sort method

List<Students> students = new ArrayList<>();
add students
students.sort(new SortByGPA());
Lambda & Strategy Pattern

If strategy only contains one method
Can replace Strategy classes with lambda

In Java may need to define lambda type
Java Sorting Using Lambda

List<Students> students = new ArrayList<>();

add students

students.sort( (a, b) -> (a.gpa() <= b.gpa()) ? -1 : 1);

java comparing is a pain. Need to convert boolean into -1, 0, 1. The above code ignores the 0 case, which it should handle.
Does Lambda eliminate need for Strategy?

Some strategies require multiple methods
   Java Layout managers
   May make more sense to combine in a class than have multiple lambdas

What is the essence of the pattern?

   Wrapping functions in a class

   Injecting functions/methods into context
Costs

Clients must be aware of different Strategies

Communication overhead between Strategy and Context

Increase number of objects
Benefits

Alternative to subclassing of Context

Eliminates conditional statements

Replace in Context code like:

```java
switch ( flag ) {
    case A: doA(); break;
    case B: doB(); break;
    case C: doC(); break;
}
```

With code like:

```java
strategy.do();
```

Gives a choice of implementations
Refactoring: Replace Conditional Logic with Strategy

Conditional logic in a method controls which of several variants of a calculation are executed

so

Create a Strategy for each variant and make the method delegate the calculation to a Strategy instance
Replace Conditional Logic with Strategy

class Foo {
    public void bar() {
        switch ( flag ) {
            case A: doA(); break;
            case B: doB(); break;
            case C: doC(); break;
        }
    }
}

class Foo {
    private strategy;
    public void bar() {
        strategy.do(data);
    }
}