Iteration abstraction ⟶ generic access -> collection's elements

unlimited size

specified as an interface

Generator: generate elements *incrementally*

- Pre-populated collections: generate indices incrementally (eg. linkedList)
- Auto-populated collections: generate elements & indices incrementally + condition (eg. PrimeList) – possible up to max value

| << interface >> java.util.Iterator | |
|---|---|
| Iterator | iterator method |
| | return an Iterator object<br>can alone |
| + hasNext(): Boolean<br>ask<br>check if more results return | iterator()<br>reverseIterator() |
| + next(): Object<br>get<br>check if more results return<br>if new returned result<br>➔ **modify** state of generator | |
| + remove() optional<br>use for modifiable collections | |

```
/**
* @effects
* if there are more elements to yield
*     return true
* else
*     return false
*
*/
public boolean hasNext();
```

```
/**
* @modifies this
* @effects
* if there are more elements to yield
*     returns the next result
*     modifies state of this -> record the yield
* else
*     throws NoSuchElementException
*
*/
public Object next() throws NoSuchElementException;
```

```
/**
* @effects
* remove from the underlying collection
* the last element returned by the call to next
*
* if remove is not supported
*    throw UnsupportedOperationException
* if next has not yet been called or remove has already been called
* after the last call to next
*    throw IllegalStateException
*
* if the underlying collection is modified while iteration in progress
* in anyway other than by calling this method
*    the behavior: unspecified
*
*/
public void remove() throws UnsupportedOperationException,
                                    IllegalStateException;
```

java.util.Iterable

- Collection
- List: ArrayList, LinkedList
- Set
- Queue

iterator method:

- common names: elements, iterator
- @effects (before): describe generator
- @requires: ‘this must not be modified while generator is in use’

generator

❖ private inner class (of collection class)
❖ java.util.Iterator

✓ attributes: keep track of iteration state
✓ abstract_properties: - elements sequence
- refer to elements attribute (of enclosing class)

➢ Operations: - hasNext
- next
- remove: (if @modifies)
- ~~repOK~~

implement

- iterator method: return a new generator object
- generator:

| | pre-populated | auto-populated |
|---|---|---|
| ✓ hasNext | check size | bound condition |
| ✓ next | return next element | generate + return next elements |

- ✓ access outer class attributes
- ✓ invoke methods

eg.

- iterator: return a new instance (of LikedListGen)
- LinkedListGen:
  hasNext: *check* ind(variable) against LinkedList.*size*
  next : return element at index ind -> incremental
  throw exception (message points to *iterator*) if fail

```
/**
* @effects <pre>
* if this is empty
*     throw EmptyException
* else
*     return a generator - produce all elements of this in sequence
* </pre>
* @requires <tt> this </tt> must not be modified while generator is in
*                                  use
*
*/
@DOpt(type = OptType.ObserverIterator)
public Iterator<E> iterator() throws EmptyException {
      if (size() == 0)
            throw new EmtyException("LinkedList.iterator");
      return new LinkedListGen();
}
```

```
/**
* @overview
* LinkedList.LinkedListGen represents a generator of elements of an LinkedList
* @effects
* ind      Integer
* @abstract_properties
* mutable(ind) = false  /\  min(ind) = 0  /\
* ind < LinkedList.size()  /\
* LinkedListGen.new = [x1, …] where each xi is in LinkedListGen.LinkedList
* and xis are arranged in same order as LinkedListGen.LinkedList's elements
*/
private class LinkedListGen<T> implements Iterator<E> {
@DomainConstraint(type = "Integer", mutable = false, min = 0)
private int ind;  // next index
// constructor method
public LinkedListGen() {
      ind = 0;
}
```

```
@Override
public boolean hasNext() {
      return (ind < size());
}
```

```
@Override
public E next() throws NoSuchElementException {
      if (hasNext()) {
            E next = get(ind);
            ind++;
            return next;
      }
      throw new NoSuchElementException("LinkedList.iterator");
}

@Override
public void remove() {
// do nothing
}
}
```

Iterator method: observe @requires , use generator object

while… loop -> iterator elements

eg. evenNumbersUpTo: pre-populated a set of even numbers up to some value & print

Loop condition controlled by

hasNext

```
// create an even number list
LinkedList<Integer> list = new LinkedList<>();
for (int i = 0; I < 100; i++) {
      if (i % 2 == 0)
            list.add(i);
}
Iterator<Integer> g = list.iterator();
// loop controlled by hasNext
while (g.hasNext())
      Integer x = g.next();
      // use x
```

NoSuchElementException thrown by *next*

```
// create an even number list
LinkedList<Integer> list = new LinkedList<>();
for (int i=0; i<100; i++)
      if (i%2 == 0) list.add(i);
Iterator<Integer> g = list.iterator();
try {
      while(true)
            Integer x = g.next();
            // use x
} catch (NoSuchElementException e) {
// no more elements
}
```