Related: [[Ex-poly-iterator]], [[Ex-iterator-IntSet]]

> [!bug] **The problem** For all of the elements of the set, do something.

(For example, summing up all the values of an IntSet.)

A possible solution would be to **extend the ADT:** Implementing features like `sum()`... But what if we need to: - Compute the sum of elements < k - Compute the sum of x^2 - compute the sum of $x^2 \ \forall \ x < k$ We would need to continuously extend the operations of the ADT -> **not great design**

Iterators!

This is what the client would see:

```
IntSet s4;

// list items of s4
ResettableIntSetIterator it = s4.iterator();
    for (int i=0; it.hasNext(); i++){
        System.out.format("\n s4[%d] = %d", i, it.next());
    }
```

Defining Iterators

```
// MISSION is to provide an iterator over the elements of an IntSet.
public class IntSetIterator implements Iterator<Integer>{
    /**
    "elements" contains a copy of the elements of the IntSet when this
       iterator is created. The data that have yet to be visited are
       elements[current, ...].
    **/
    private int current;
    final private Vector<Integer> elements;
}
```

Once the iterator is created, if the original set changes the iterator continues to work with the original copy.

`Iterator<Integer>` is an [[Interface]] that specifies:

```
public boolean hasNext()    // Are there more elements to iterate over?
public Integer next()       // Moves the iterator to the next value and
   returns its *value*
public void remove()        // Removes the element last returned by next
   ()
```

(The contract for these methods is already defined by the interface)

Creating Iterators

```
class IntSet...

    /** Create an iterator Object.
    @return an iterator over this set. The iterator is not sensible to
       mutations of this set.
    **/
    public ResettableIntSetIterator iterator(){
        assert (this.elements!=null);
        return (new ResettableIntSetIterator(this));
    }
```

Using the iterator

```
// Print all items
ResettableIntSetIterator it = s4.iterator();
for (int i=0; it.hasNext(); i++){
    System.out.format("\n s4[%d] = %d", i, it.next());
}

// compute the sum
int tot = 0;
it.reset();
for (int i=0; it.hasNext(); i++){
    tot = tot + it.next();
}
System.out.format("\n tot= %d", tot);

// Compute the sum of x^2 for each x: x<k
int tot2 = 0;
int k = 10;
it.reset();
for (int i=0; it.hasNext(); i++){
    Integer x = it.next();
    if(x<k){
        tot2 = tot2+x*x;
    }
}
System.out.format("\n tot2= %d", tot2);
```

Implementing methods for the new class

```
/**
Initialize the iterator with current index = 0
Store a copy of its elements in "elements"
@param s the source of data to initialize this iterator
@throws NullPointerException if s is null
**/

ResettableIntSetIterator(IntSet s){ // constructor that initializes the
    iterator.
    if (s == null){
```

```
10          throw new NullPointerException("s cannot be null");
11      }
12      this.elements = (Vector<Integer>) s.elements.clone();
13      this.current = 0;
14  }
15
16  @Override
17  public boolean hasNext(){
18      return (this.current < this.elements.size());
19  }
20
21  @Override
22  public Integer next(){
23      if (this.currente < this.elements.size()){
24          Integer res = this.elements.get(this.current);
25          return(res);
26      } else {
27          throw new NoSuchElementException("Went beyond the available
                values");
28      }
29  }
30
31  @Override
32  public void remove(){
33      throw new UnsupportedOperationException("Remove is not yet
            supported");
34  }
35
36  // MODIFY: Change the current index and set it to 0.
37  public void reset(){
38      this.current = 0;
39  }
```

The `ResettableIntSetIterator` class is a custom **iterator** designed for IntSet objects. It *implements* the Iterator interface.

Iterators as inner class

- Encapsulation
- Access to private members
- Improved maintainability
- Thread safety
- Reusability
- **No dependency**
- **Cleaner code**

Done by defining an inner (private) class that implements the Iterator interface:

```
private class ResettableIntSetIterator implements Iterator<Integer>{
    ...
}
```

And this is how the client uses it:

```
Iterator<Integer> it2 = s4.iterator();
for (int i=0; it2.hasNext();; i++){
    System.out.format("\n s4[%d] = %d", i, it2.next());
}
// compute the sum
it2 = s4.iterator(); // need to create a new iterator(?)
tot = 0;
for (int i = 0; it2.hasNext(); i++){
    tot = tot + it2.next();
}
System.out.format("\n  tot = %d". tot);
```

Issues with iterators

What if we modify the elements while looping with an iterator?

> [!attention]- code
>
> ```
> List<Integer> s4 = new ArrayList<Integer>();
> s4 = Arrays.asList(1,2,3,4);
> Iterator<Integer> it2 = s4.iterator();
> for (int i = 0; it2.hasNext(); i++){
>   s4.insert(1);
>   s4.remove(2);
>   System.out.format("\n s4[%d] = %d", i, it2.next());
> }
> ```

Possible solutions: - Creating a copy of the data - Throwing a `ConcurrentModificationException` when the count of modifications > 1. - Doing a `CopyOnWrite` operation when the first modification occurs.