Traversable

A *traversal* of a collection ”visits” each element, once. Ex: summing, averaging or printing elements of a collection are all traversals.

You know how to traverse an array: for-loop. Ex: summing up entire array of ints:

```
int sum = 0;
for(int i=0; i<arr.length; i++)
    sum += arr[i];
```

or

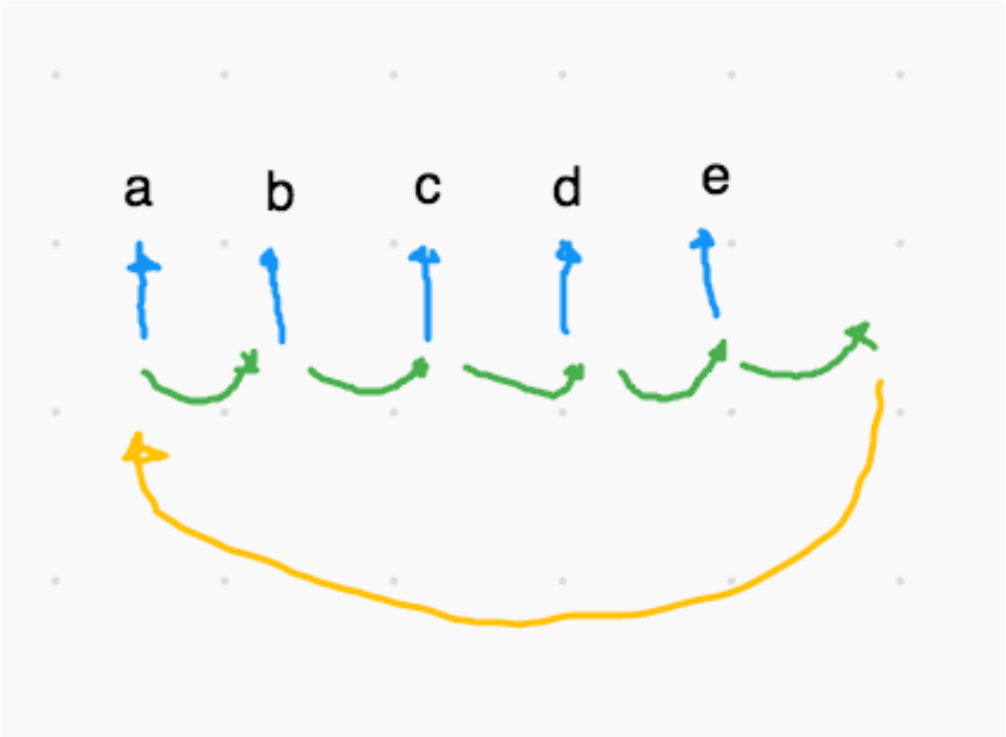
```
int sum = 0;
for(int x : arr)
   sum += x;
```

We can generalize the idea of a traversal with some operations. Here they are in pseudo-code:

```
while( "has next" ) {
    x = "next"
}
```

Since we may want to have more than one travesal, we will also add a `reset` operation, to go back to the beginning.

In this representation of the traversal the blue arrow is keeping track of the position in the traversal, the green arrows are what happens when we advance with `next()` and the yellow arrow is a `reset()` (at the end in this case).



Traversable API

hasNext()

Description	Determine if there are elements remaining in the traversal.
Signature	`boolean hasNext()`
Preconditions	Traversal has been started.
Returns	true if the traversal has element(s) remaining, false otherwise.

next()

Description	Get the next element in the traversal
Signature	`T next()`
Preconditions	Traversal has been started, there is still elements remaining in the traversal.
Returns	The next element in the traversal.

reset()

Description	Start a new traversal.
Signature	`void reset()`
Preconditions	None.
Returns	None.

```
public interface Traversable<T> {
    boolean hasNext();
    T next();
    void reset();
}
```

Example: sum a traversable collection of integers

```
public static int sum(Traversable<Integer> traversable) {
      int sum = 0;
      traversable.reset();
      while(traversable.hasNext()) {
            sum += traversable.next();
      }
      return sum;
}
```

Example: print all elements in a traversable collection.

```
public static <T> void print(Traversable<T> traversable) {
      traversable.reset();
      while(traversable.hasNext()) {
            T tmp = traversable.next();
            if(tmp == null)
               throw new RuntimeException();
            sout(tmp);
      }
}
```

Example: a simple traversable implementation

A range is a set of integer values between a lower and high value. Ex: [3, 7[–> {3, 4, 5, 6}

```
public class Range implements Traversable<Integer> {

    private int lower;
    private int upper;
    private int current;

    public Range(int upper) {
        this(0, upper);
    }

    public Range(int lower, int upper) {
        if(lower > upper)
            throw new RuntimeException("Impossible range.");
        this.lower = lower;
        this.upper = upper;
    }

    public boolean in(int x) {
        return lower ⩽ x && x < upper;
    }

    public void reset() {
        current = lower;
    }

    public int next() {
        return current++;
    }

    public boolean hasNext() {
        return current < upper;
    }

    @Override
    public String toString() {
        return "[" + lower + ".." + upper + "[";
    }
}

Range range = new Range(1, 10);
sout(sum(range));
print(range);
```

Exercise: traversable and queue

1. Write a Queue constructor that takes it's elements from a Traversable:

```
public Queue(int capacity, Traversable<T> traversable) { ... }
```

1. Make Queue traversable:

```
public class Queue<T> implements Traversable<T> {
    ...
}
```

Videos

- Traversable [1/2] `https://youtu.be/zLppbiZUwYc`
- Traversable [2/2] `https://youtu.be/5j35WPu-bqM`