CSCI 136:
Data Structures
and
Advanced Programming

Lecture 18

Iterators

Instructor: Dan Barowy

Williams

Outline

1. Iterators,
2. Iterators,
3. Iterators!

Quiz

Iterators

What do the following have in common?

```
double[] a
// … initialize a …
double sum = 0.0;
for (int i = 0; i < a.length; i++) {
  sum += a[i];
}
```

```
List<Double> ls = new SinglyLinkedList<>();
// … initialize ls …
double sum = 0.0;
for (int i = 0; i < ls.size(); i++) {
  sum += ls.get(i);
}
```
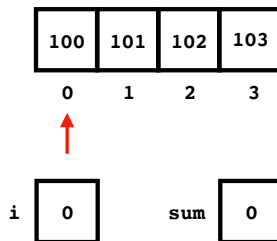
```
Stack<Double> s = new StackVector<>();
// … initialize s …
double sum = 0.0;
while (!s.isEmpty()) {
  sum += s.pop();
}
```

Iteration

Iteration is the **repetition of a process** in order to generate a (possibly unbounded) **sequence of outcomes**. Each repetition of the process is a single iteration, and the outcome of each iteration is then the starting point of the next iteration.
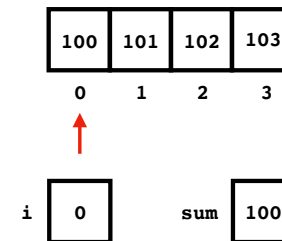
Each program iterates

```
double[] a
// … initialize a …
double sum = 0.0;
for (int i = 0; i < a.length; i++) {
  sum += a[i];
}
```

100	101	102	103
0	1	2	3

i | 0   sum | 0

Each program iterates

```
double[] a
// … initialize a …
double sum = 0.0;
for (int i = 0; i < a.length; i++) {
  sum += a[i];
}
```
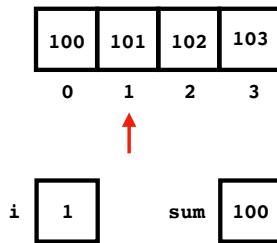
100	101	102	103
0	1	2	3

i | 0   sum | 100

Each program iterates

```
double[] a
// … initialize a …
double sum = 0.0;
→ for (int i = 0; i < a.length; i++) {
    sum += a[i];
}
```

```
100  101  102  103
 0    1    2    3

i  1        sum  100
```
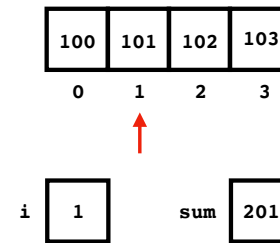
Each program iterates

```
double[] a
// … initialize a …
double sum = 0.0;
for (int i = 0; i < a.length; i++) {
→   sum += a[i];
}
```
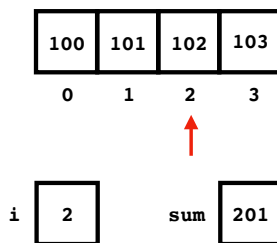
```
100  101  102  103
 0    1    2    3

i  1        sum  201
```

Each program iterates

```
double[] a
// … initialize a …
double sum = 0.0;
→ for (int i = 0; i < a.length; i++) {
    sum += a[i];
}
```
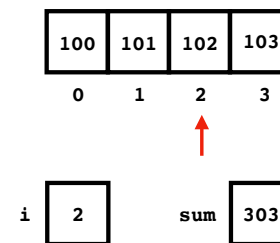
```
100  101  102  103
 0    1    2    3

i  2        sum  201
```

Each program iterates

```
double[] a
// … initialize a …
double sum = 0.0;
for (int i = 0; i < a.length; i++) {
→   sum += a[i];
}
```
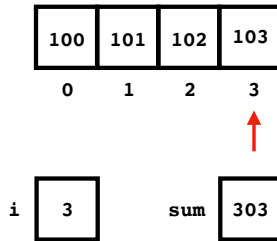
```
100  101  102  103
 0    1    2    3

i  2        sum  303
```

Each program iterates

```
        double[] a
        // … initialize a …
        double sum = 0.0;
→       for (int i = 0; i < a.length; i++) {
          sum += a[i];
        }
```
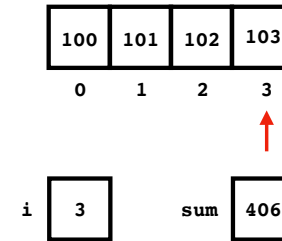
100	101	102	103
0	1	2	3

i [3]   sum [303]

Each program iterates

```
        double[] a
        // … initialize a …
        double sum = 0.0;
        for (int i = 0; i < a.length; i++) {
→         sum += a[i];
        }
```
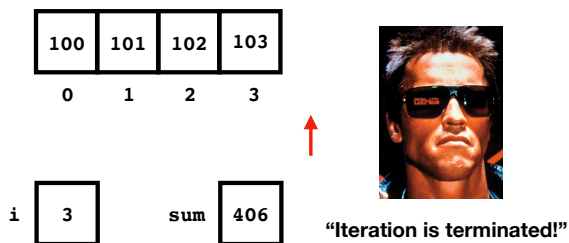
100	101	102	103
0	1	2	3

i [3]   sum [406]

Each program iterates

```
        double[] a
        // … initialize a …
        double sum = 0.0;
→       for (int i = 0; i < a.length; i++) {
          sum += a[i];
        }
```
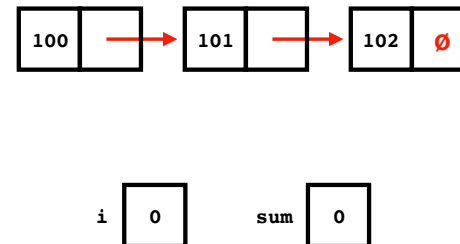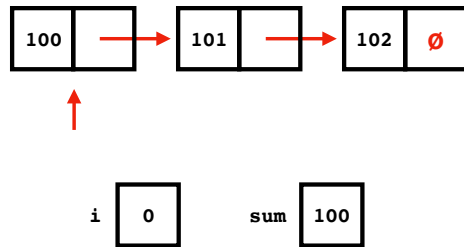
100	101	102	103
0	1	2	3

i [3]   sum [406]

"Iteration is terminated!"

Each program iterates

```
        List<Double> ls = new SinglyLinkedList<>();
        // … initialize ls …
        double sum = 0.0;
→       for (int i = 0; i < ls.size(); i++) {
          sum += ls.get(i);
        }
```

| 100 | → | 101 | → | 102 | Ø |

i [0]   sum [0]

Each program iterates

```
List<Double> ls = new SinglyLinkedList<>();
// … initialize ls …
double sum = 0.0;
for (int i = 0; i < ls.size(); i++) {
  sum += ls.get(i);
}
```
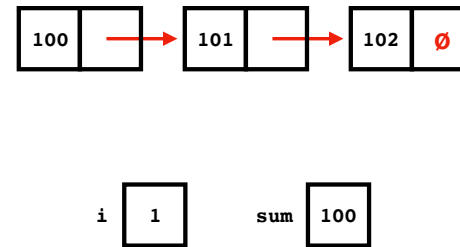
| 100 | | 101 | | 102 | Ø |

i 0   sum 100

Each program iterates

```
List<Double> ls = new SinglyLinkedList<>();
// … initialize ls …
double sum = 0.0;
for (int i = 0; i < ls.size(); i++) {
  sum += ls.get(i);
}
```
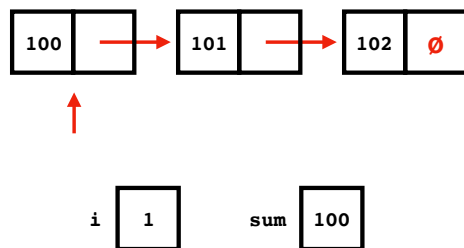
| 100 | | 101 | | 102 | Ø |

i 1   sum 100

Each program iterates

```
List<Double> ls = new SinglyLinkedList<>();
// … initialize ls …
double sum = 0.0;
for (int i = 0; i < ls.size(); i++) {
  sum += ls.get(i);
}
```
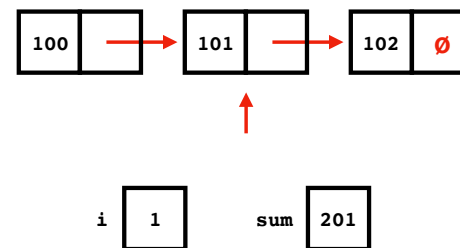
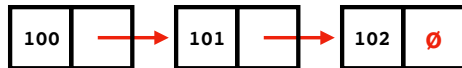| 100 | | 101 | | 102 | Ø |

i 1   sum 100

Each program iterates

```
List<Double> ls = new SinglyLinkedList<>();
// … initialize ls …
double sum = 0.0;
for (int i = 0; i < ls.size(); i++) {
  sum += ls.get(i);
}
```

| 100 | | 101 | | 102 | Ø |

i 1   sum 201

Each program iterates

```
List<Double> ls = new SinglyLinkedList<>();
// … initialize ls …
double sum = 0.0;
for (int i = 0; i < ls.size(); i++) {
  sum += ls.get(i);
}
```

```
100 → 101 → 102 Ø
```
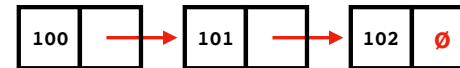
i 2   sum 201

Each program iterates

```
List<Double> ls = new SinglyLinkedList<>();
// … initialize ls …
double sum = 0.0;
for (int i = 0; i < ls.size(); i++) {
  sum += ls.get(i);
}
```

```
100 → 101 → 102 Ø
```

i 2   sum 201

Each program iterates

```
List<Double> ls = new SinglyLinkedList<>();
// … initialize ls …
double sum = 0.0;
for (int i = 0; i < ls.size(); i++) {
  sum += ls.get(i);
}
```

```
100 → 101 → 102 Ø
```
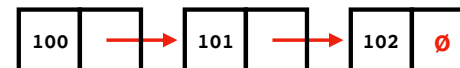
i 2   sum 201

Each program iterates

```
List<Double> ls = new SinglyLinkedList<>();
// … initialize ls …
double sum = 0.0;
for (int i = 0; i < ls.size(); i++) {
  sum += ls.get(i);
}
```

```
100 → 101 → 102 Ø
```
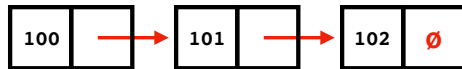
i 2   sum 303

Each program iterates

```
List<Double> ls = new SinglyLinkedList<>();
// … initialize ls …
double sum = 0.0;
for (int i = 0; i < ls.size(); i++) {
  sum += ls.get(i);
}
```
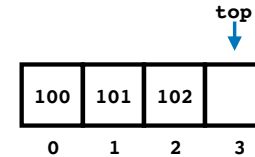
```
100 → 101 → 102 ∅
```

i 2   sum 303

"Iteration is terminated!"

Each program iterates

```
Stack<Double> s = new StackVector<>();
// … initialize s …
double sum = 0.0;
while (!s.isEmpty()) {
  sum += s.pop();
}
```

top

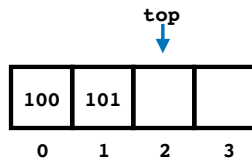| 100	101	102	
0	1	2	3

sum 0

Each program iterates

```
Stack<Double> s = new StackVector<>();
// … initialize s …
double sum = 0.0;
while (!s.isEmpty()) {
  sum += s.pop();
}
```

top

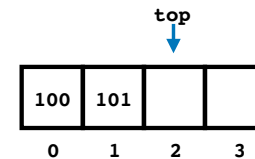| 100	101		
0	1	2	3

sum 102

Each program iterates

```
Stack<Double> s = new StackVector<>();
// … initialize s …
double sum = 0.0;
while (!s.isEmpty()) {
  sum += s.pop();
}
```

top

100	101		
0	1	2	3

sum 102

Each program iterates

```
Stack<Double> s = new StackVector<>();
// … initialize s …
double sum = 0.0;
while (!s.isEmpty()) {
  sum += s.pop();
}
```

top

100			
0	1	2	3

sum | 203 |

Each program iterates

```
Stack<Double> s = new StackVector<>();
// … initialize s …
double sum = 0.0;
while (!s.isEmpty()) {
  sum += s.pop();
}
```
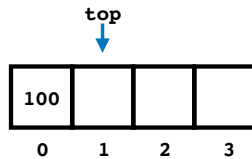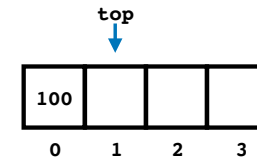
top

100			
0	1	2	3

sum | 203 |

Each program iterates

```
Stack<Double> s = new StackVector<>();
// … initialize s …
double sum = 0.0;
while (!s.isEmpty()) {
  sum += s.pop();
}
```

top

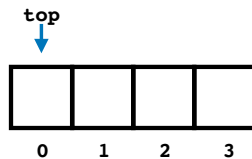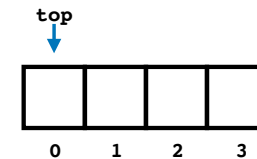|			
0	1	2	3

sum | 303 |

Each program iterates

```
Stack<Double> s = new StackVector<>();
// … initialize s …
double sum = 0.0;
while (!s.isEmpty()) {
  sum += s.pop();
}
```

top

0	1	2	3

sum | 303 |

"Iteration is terminated!"

Essentially the same algorithm!

```
double[] a
// … initialize a …
double sum = 0.0;
for (int i = 0; i < a.length; i++) {
  sum += a[i];
}
```

```
List<Double> ls = new SinglyLinkedList<>();
// … initialize ls …
double sum = 0.0;
for (int i = 0; i < ls.size(); i++) {
  sum += ls.get(i);
}
```

```
Stack<Double> s = new StackVector<>();
// … initialize s …
double sum = 0.0;
while (!s.isEmpty()) {
  sum += s.pop();
}
```

But the code looks different.

Problems

- **Different data structures** yield **different code for same algorithm**.

- **Data hiding** potentially causes **efficiency problems**.

- **Inspecting** data structure "from the outside" can **change the state** of a data structure (e.g., `pop()`'ing a `Stack`).

What if I told you that you could solve

all of these problems with **abstraction**?

Iteration abstraction to the rescue.

```
double[] a
// … initialize a …
double sum = 0.0;
for (double d : a) {
  sum += d;
}
```

```
List<Double> ls = new SinglyLinkedList<>();
// … initialize ls …
double sum = 0.0;
for (double d : ls) {
  sum += d;
}
```

```
Stack<Double> s = new StackVector<>();
// … initialize s …
double sum = 0.0;
for (double d : s) {
  sum += d;
}
```

Brought to you by **Iterators**.

Iterators are a really good idea.

- Invented by Barbara Liskov in 1974.

- Incidentally, **abstract data types** were also invented by Barbara Liskov in 1974.

- Both debuted in the influential PL called **CLU**.

- Barbara won the **Turing Award in 2008** for this work and more.

How does "for each" work?

All of these data structures implement `Iterable<T>`

structure5
Interface Stack<E>

All Superinterfaces:
java.lang.Iterable<E>, Linear<E>, Structure<E>

All Known Implementing Classes:
AbstractStack, StackArray, StackList, StackVector

structure5
Interface List<E>

All Superinterfaces:
java.lang.Iterable<E>, Structure<E>

All Known Implementing Classes:
AbstractList, CircularList, DoublyLinkedList, SinglyLinkedList, Vector

(array is a special case)

What is an `Iterable<T>`?

```
public interface Iterable<T>
{
    Iterator<T> iterator();
}
```

It's a class that returns an `Iterator<T>`.

What's an `Iterator<T>`???

```
public interface Iterator<E>
{
    boolean hasNext();
    E next();
    …
}
```

It's an object that lets you **iterate through a data structure**.

Importantly, `Iterator`s are **stateful**.

Why does statefulness matter? It can **save work**.

Slide 1

Let's look at `SinglyLinkedList<T>`

(see `structure5`)

Slide 2

How does `for` use an `Iterator<T>`?

The following code

```
List<Integer> ls = new SinglyLinkedList<>();
// …
for (int i : ls) {
  // … work …
}
```

is the moral equvalent to

```
List<Integer> ls = new SinglyLinkedList<>();
// …
for (Iterator<Integer> i = ls.iterator(); i.hasNext();) {
  int n = i.next();
  // … work …
}
```

1. Get `Iterator<T>`
2. Get next element.
3. If there is a next element, go to 2.

Slide 3

Example.

```
List<Double> ls = new SinglyLinkedList<>();
// … initialize ls …
double sum = 0.0;
for (double d : ls) {
  sum += d;
}
```
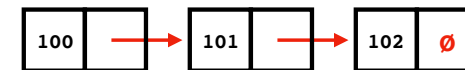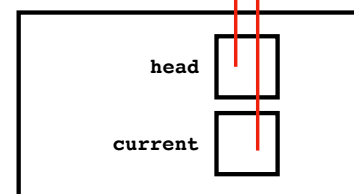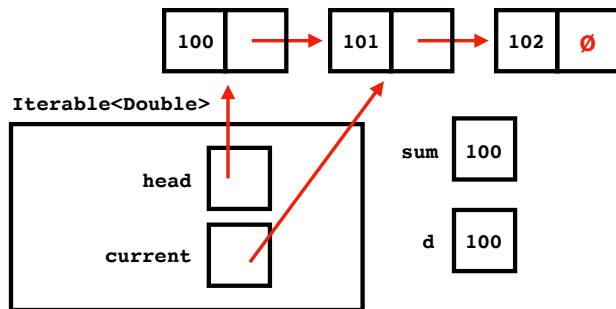
```
100 → 101 → 102 Ø
```

sum 0

d 0

Slide 4

Example.

```
List<Double> ls = new SinglyLinkedList<>();
// … initialize ls …
double sum = 0.0;
for (double d : ls) {
  sum += d;
}
```

```
100 → 101 → 102 Ø
```

Iterable<Double>

head

current
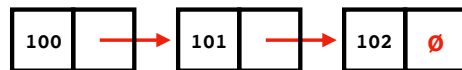
sum 0

d 0

Example.

```
List<Double> ls = new SinglyLinkedList<>();
// … initialize ls …
double sum = 0.0;
for (double d : ls) {
  sum += d;
}
```

Iterable<Double>
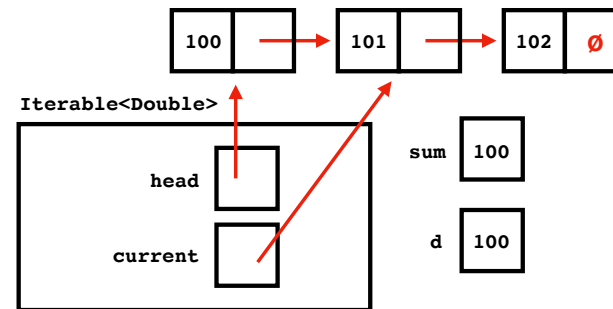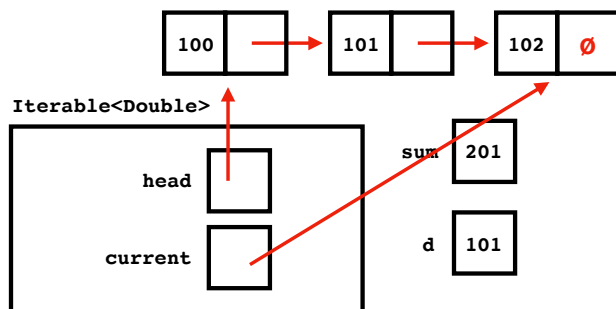
head

current

sum 100

d 100

100 → 101 → 102 Ø

Example.

```
List<Double> ls = new SinglyLinkedList<>();
// … initialize ls …
double sum = 0.0;
for (double d : ls) {
  sum += d;
}
```

Iterable<Double>

head

current

sum 100

d 100

100 → 101 → 102 Ø

Example.

```
List<Double> ls = new SinglyLinkedList<>();
// … initialize ls …
double sum = 0.0;
for (double d : ls) {
  sum += d;
}
```

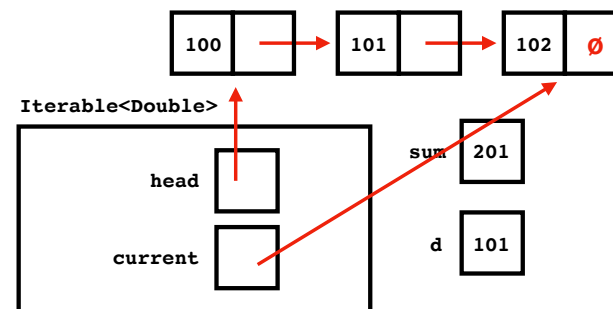Iterable<Double>

head

current

sum 201

d 101

100 → 101 → 102 Ø

Example.

```
List<Double> ls = new SinglyLinkedList<>();
// … initialize ls …
double sum = 0.0;
for (double d : ls) {
  sum += d;
}
```

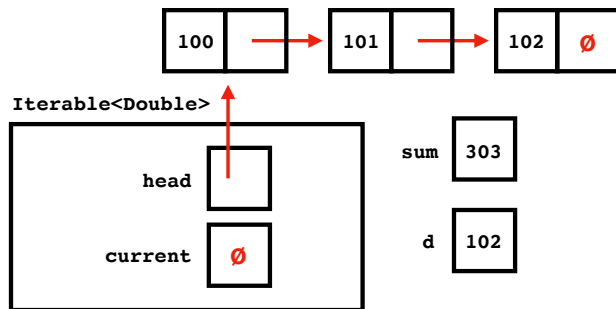Iterable<Double>

head

current

sum 201

d 101

100 → 101 → 102 Ø

Example.

```
List<Double> ls = new SinglyLinkedList<>();
// … initialize ls …
double sum = 0.0;
for (double d : ls) {
  sum += d;
}
```

```
100 → 101 → 102 Ø
```

Iterable<Double>

head

current Ø

sum 303

d 102

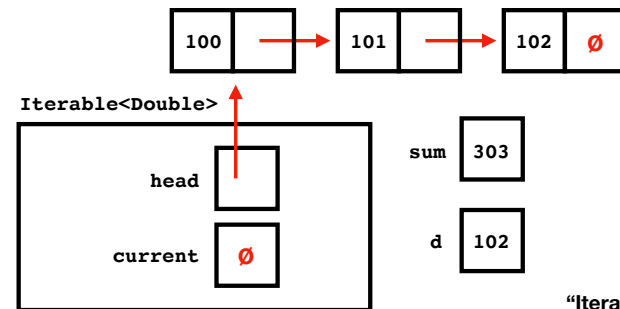## Example.

```
List<Double> ls = new SinglyLinkedList<>();
// … initialize ls …
double sum = 0.0;
for (double d : ls) {
  sum += d;
}
```

```
100 → 101 → 102 Ø
```

Iterable<Double>

head

current Ø

sum 303

d 102

"Iteration is terminated!"

Recap & Next Class

Today we learned:

Iterators

Next class:

Ordered structures,

resubmission procedure