CS 2110, FA23

Discussion 6: Iterators for linked lists

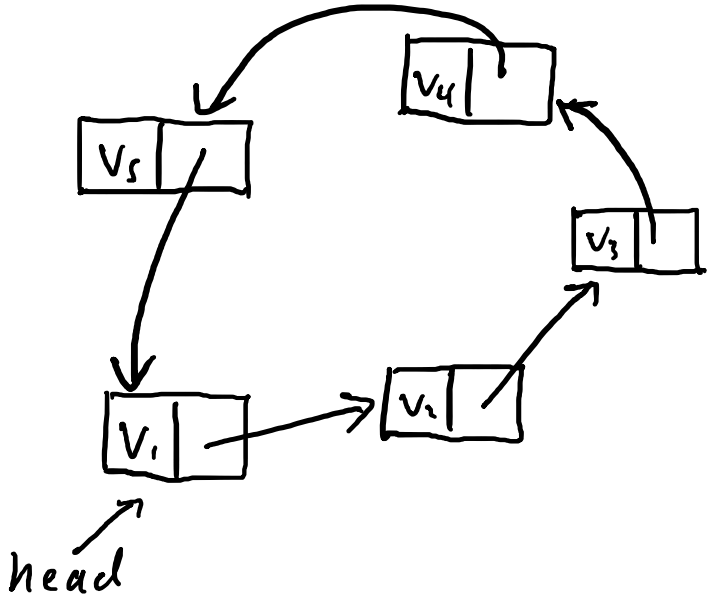
Warmup: circular linked lists

```
Node<T> {
    T value;
    Node<T> next;
}
```

`next` might never be null – could point back to *beginning* of list

(could also point to middle; assume it doesn't)

- **Write a loop** to call `process(v)` for every `String` value `v` in a circular linked list whose first node is `head`.



Generalized iteration

- Traditional for-loops iterate over indices, but that only makes sense for Lists (and even then, may be inefficient)
- Want a way to iterate over all elements, even if they don't have an associated index (or even a defined ordering)
- Iterator pattern: yield each element exactly once
  - Operations: get next element, ask whether there are any elements left

Java Iterator

- Generic interface expressing Iterator ADT
- Methods:
  - boolean hasNext();
  - T next();

Usage:

```
Iterator<String> it = …;
while (it.hasNext()) {
  String s = it.next();
  // Do something with s
}
```

Enhanced for-loops

```
List<String> names = …;
for (int i=0; i<names.size(); ++i)
{
    String name = names.get(i);
    …
}
```

```
List<String> names = …;
for (String name : names) {
    …
}
```

… are translated into while loops ("syntactic sugar")

```
List<String> names = …;
for (String name : names) {
    …
}
```

```
List<String> names = …;
Iterator<String> it =
  names.iterator();
while (it.hasNext()) {
    String name = it.next();
    …
}
```

Iteration interfaces

Iterable<T>

- "Something that can be iterated over"
- Can use in an enhanced for-loop
- Yields Iterators

- `Iterator<T> iterator();`

Iterator<T>

- Helper class for actually doing the iteration
- Mutable (one-time use) - need a new one for each loop
- Yields values

- `boolean hasNext();`
- `T next();`

Implementing Iterators

1. Move loop guard logic to `hasNext()`
2. Move advancement logic to `next()`
3. Remember important context in fields

Task: implement `CNodeIterator<T>` to yield every value in a circular list exactly once

1. Identify appropriate fields
2. Define class
3. Implement `hasNext()` and `next()`

Bonus: Nested classes

- Classes declared inside other classes (usually a "helper" of some kind)
- Static: Outer class acts as a namespace, can hide class from other potential clients
- Non-static ("inner classes"): Inner class objects are attached to an outer class *instance*
  - Can only be created from an instance of the outer class
  - Can access outer object's fields and methods
  - Common choice for Iterators
    - Enables more encapsulation (private fields)