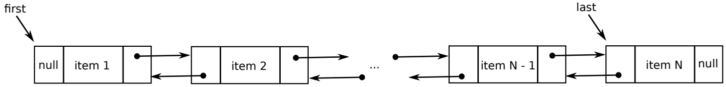
Problems

Problem 1 (*Deque*) Create a generic iterable data type `LinkedDeque<Item>` that uses a linked list to implement the following deque API:

method	description
`LinkedDeque()`	construct an empty deque
`boolean isEmpty()`	is the deque empty?
`int size()`	the number of items on the deque
`void addFirst(Item item)`	add $item$ to the front of the deque
`void addLast(Item item)`	add $item$ to the end of the deque
`Item removeFirst()`	remove and return the item from the front of the deque
`Item removeLast()`	remove and return the item from the end of the deque
`Iterator<Item> iterator()`	an iterator over items in the deque in order from front to end
`String toString()`	a string representation of the deque

Hints

- Use a doubly-linked list `Node` to implement the Deque API — each node in such a list stores a generic `item`, and pointers `next` and `prev` to the next and previous nodes



Problems

- Instance variables
    - Size of the deque, `int N`
    - Pointer to the head of the deque, `Node first`
    - Pointer to the tail of the deque, `Node last`
- `LinkedDeque()`
    - Initialize instance variables to appropriate values
- `boolean isEmpty()`
    - Return whether the deque is empty or not
- `int size()`
    - Return the size of the deque
- `void addFirst(Item item)`
    - Add the given item at the head end of the deque
    - Increment `N` by one
- `void addLast(Item item)`
    - Add the given item at the tail end of the deque
    - Increment `N` by one

Problems

- `Item removeFirst()`
  - Remove and return the item at the head end of the deque
  - Decrement `N` by one
- `Item removeLast()`
  - Remove and return the item at the tail end of the deque
  - Decrement `N` by one
- `Iterator<Item> iterator()`
  - Return an object of type `DequeIterator`
- Instance variable for `DequeIterator`
  - Pointer to current node in the iterator, `Node current`
- `DequeIterator()`
  - Initialize instance variable appropriately
- `boolean DequeIterator.hasNext()`
  - Return whether the iterator has more items to iterate or not
- `Item DequeIterator.next()`
  - Return the item in `current` and advance `current` to the next node

Problems

Problem 2 (*Random Queue*) Create a generic iterable data type `ResizingArrayRandomQueue<Item>` that uses a resizing array to implement the following random queue API:

method	description
`ResizingArrayRandomQueue()`	construct an empty queue
`boolean isEmpty()`	is the queue empty?
`int size()`	the number of items on the queue
`void enqueue(Item item)`	add $item$ to the queue
`Item dequeue()`	remove and return a random item from the queue
`Item sample()`	return a random item from the queue, but do not remove it
`Iterator<Item> iterator()`	an independent iterator over items in the queue in random order
`String toString()`	a string representation of the queue

Hints

- Use a resizing array to implement the Random Queue API
- Instance variables
  - Array to store the items of queue, `Item[] q`
  - Size of the queue, `int N`

Problems

- `ResizingArrayRandomQueue()`
  - Initialize instance variables appropriately — create `q` with an initial capacity of 2
- `boolean isEmpty()`
  - Return whether the queue is empty or not
- `int size()`
  - Return the size of the queue
- `void enqueue(Item item)`
  - If `q` is at full capacity, resize it to twice its current capacity
  - Insert the given item in `q` at index `N`
  - Increment `N` by one
- `Item dequeue()`
  - Save `q[r]` in `item`, where `r` is a random integer from the interval `[0, N)`
  - Set `q[r]` to `q[N - 1]` and `q[N - 1]` to `null`
  - If `q` is at quarter capacity, resize it to half its current capacity
  - Decrement `N` by one
  - Return `item`
- `Item sample()`
  - Return `q[r]`, where `r` is a random integer from the interval `[0, N)`

Problems

- `Iterator<Item> iterator()`
  - Return an object of type `RandomQueueIterator`
- Instance variables for `RandomQueueIterator`
  - Array to store the items of `q`, `Item[] items`
  - Index of the current item in `items`, `int current`
- `RandomQueueIterator()`
  - Create `items` with the same capacity as `q`
  - Copy the items of `q` into `items`
  - Shuffle `items`
  - Initialize `current` appropriately
- `boolean RandomQueueIterator.hasNext()`
  - Return whether the iterator has more items to iterate or not
- `Item RandomQueueIterator.next()`
  - Return the item in `items` at index `current` and advance `current` by one

Problems

Problem 3 (*Subset*) Write a client program `Subset.java` that takes a command-line integer k, reads in a sequence of strings from standard input using `StdIn.readString()`, and prints out exactly k of them, uniformly at random. Each item from the sequence can be printed out at most once. You may assume that $0 \leq k \leq N$, where N is the number of strings on standard input.

Hints

- Create an object `q` of type `ResizingArrayRandomQueue`
- Read strings from standard input and insert them into `q`
- Dequeue and print `k` (command-line argument) items from `q`