CSE 2123:
Collections: Queues and Stacks

Jeremy Morris
A queue is a specific type of collection

Imagine a line for a bank teller or a checkout lane at a store
- Contains many people
- But only the one at the head of the queue (line) will be leaving the queue to do anything

We call this kind of processing a FIFO queue (First In, First Out)
Collections – FIFO Queue

- FIFO queue processing relies on a few standard methods:
  - Add an object to the end of the queue:
    ```java
    void add(E obj)
    ```
  - Retrieve (and remove) the head of the queue:
    ```java
    E remove()
    ```
Collections – FIFO Queue

- Examine head of queue without removing it:
  ```java
  E peek()
  ```

- Test to see if the queue is empty:
  ```java
  boolean isEmpty()
  ```
In Java, the most common implementation of a queue is the `LinkedList` class

- Similar to `ArrayList` class
- Implements both a `List` interface and a `Queue` interface
  - Could use it like an `ArrayList`…
  - …but you shouldn’t
Collections – FIFO Queue

- Circumstances determine whether to use an ArrayList or a LinkedList
  - ArrayList more efficient for array-like operations
    - “Constant time” to access positions at random in the list
    - “Linear time” to add elements to front or iterate and remove elements
  - LinkedList more efficient for queue-like operations
    - “Constant time” to add elements to either end or iterate and remove elements
    - “Linear time” to access positions at random in the list
Collections – FIFO Queue

- ArrayList – block of contiguous elements
- LinkedList – connected through chain of references (links)
Collections – FIFO Queue

- Head of the list is removed for processing
Collections – FIFO Queue

- Head of the list is removed for processing
  - Head shifts to next node in list
Head of the list is removed for processing
- Head shifts to next node in list

New nodes appended to tail of the list
Collections – FIFO Queue

- Queue properties:
  - First node in queue is known as the *head*
  - New nodes are appended to the end of the queue
  - Use `remove()` to take nodes off the queue
  - Use `add()` to put nodes into the queue
  - Use `isEmpy()` to test to see if the queue is empty
  - Use `peek()` to see the head node without removing it
Collections – FIFO Queues

- We can declare a queue by creating a new instance of a LinkedList
  - Remember – LinkedList implements the Queue interface, so we can treat it like a Queue

```
Queue<Integer> intQueue = new LinkedList<Integer>();
```
Example – FIFO Queue

```java
public static void main(String[] args) {
    Queue<Integer> myList = new LinkedList<Integer>();
    myList.add(10);
    myList.add(5);
    myList.add(22);
    System.out.println("LIST: "+myList);

    while (!myList.isEmpty()) {
        int head = myList.remove();
        System.out.print("HEAD: "+head+" ");
        System.out.println("LIST: "+myList);
    }
}
```
Practice

Write a short program that:

- Creates three Student objects
- Places them into a queue
- Removes them in order and prints the first name, last name, and student ID of each in FIFO order
A stack is another type of collection

Imagine a stack of plates in a cafeteria

- We can only ever access the plate at the top of the stack
- To get a new plate, we take it off the top of the stack
- To add plates to the stack, we place them on the top of the stack

We call this behavior Last In, First Out (LIFO)
LIFO (stack) processing relies on a few standard methods:

- Add an object to the front of the queue:
  ```java
  void push(E obj)
  ```

- Retrieve (and remove) the head of the queue:
  ```java
  E pop()
  ```
Collections - Stack

- Examine head of queue without removing it:
  
  ```java
  E peek()
  ```

- Test to see if the Stack is empty:
  
  ```java
  boolean isEmpty()
  ```
Collections - Stack

As with, FIFO queue processing, a linked list is often used when we want to build a stack

- Even easier than with FIFO queue processing, since stacks only deal with the *head* node
Collections - Stack

- Head of the list is removed for processing
  - Head shifts to next node in list
- New nodes appended to front of the list
Collections - Stack

- Head of the list is removed for processing
  - Head shifts to next node in list
- New nodes appended to front of the list
Collections - Stack

- Head of the list is removed for processing
  - Head shifts to next node in list
- New nodes appended to front of the list
Head of the list is removed for processing
  - Head shifts to next node in list
New nodes appended to front of the list
  - New first node is now the head node
Collections – Stack

Stack properties:

- First node in stack is known as the *head*
- New nodes are pushed onto the front of the stack
- Use `pop()` to remove the head off the stack
- Use `push()` to add nodes onto to the top of the stack
- Use `isEmpty()` to test to see if the stack is empty
- Use `peek()` to see the head node without popping it
We use the Stack class to instantiate a new Stack object

```java
Stack<Integer> intStack = new Stack<Integer>();
```
public static void main(String[] args) {
    Stack<Integer> myList = new Stack<Integer>();
    myList.push(10);
    myList.push(5);
    myList.push(22);
    System.out.println("LIST: "+myList);

    while (!myList.isEmpty()) {
        int head = myList.pop();
        System.out.print("HEAD: "+head+" ");
        System.out.println("LIST: "+myList);
    }
}
Write a short program that:
- Creates three Student objects
- Places them into a stack
- Removes them in order and prints the first name, last name, and student ID of each in LIFO order