

Learning Outcomes

Upon completing this assignment, students should be able to

- Modify an existing program to use a different class as its main data structure.
- Implement a *doubly-linked* list.
- Implement a *queue* as a doubly-linked list.
- Write a simple, *generic* container class.

Introduction

This program is about implementing a *queue* data structure. You will start with a working main program that makes use of the `java.util.Queue` interface (and the `java.lang.ArrayDeque` implementation of it). You can use this as a *baseline* against which you can test **your** implementation of the queue.

You will replace `java.util.Queue` with your own `util.Queue<E>` (package `util`, class `Queue`, generic on content elements, `E`).

Method

Getting Started

Clone the <https://cs-devel.potsdam.edu/F23-205/c20230927> repository so you have `Simulation.java` to get started. You can, of course, start with the whole Gradle project BUT you will need to make sure to set `origin` (the name of the remote where `git` pushes by default) to a repository to which you have *write* permissions.

How ever you want to do it, set up a Gradle project with an initialized `git` repository and `cpu.Simulation` as the main class. Get that to compile *and* add, commit, and push your changes to Gitea to confirm your setup.

Input/Output

The code is better than that seen in class: you can type any *prefix* of a command to use that command. So, instead of having to type out `enqueue`, `enq` or `en`, or even `e` will work.

Valid commands:

`quit` (or end-of-file) - end the command loop.

`enqueue <line>` - add `<line>` to the tail of the queue.

`dequeue` - remove and display the line at the head of the queue.

`peek` - display the line at the head of the queue.

After the loop ends, the contents of the queue are printed, in order, to the screen and the program terminates.

Design Considerations

Your task is to replace `java.util.Queue<E>` with your own implemented using a *doubly-linked list*.

Generic Queue

Add a package, `util`. In `util`, create a generic `Queue` class. The required public functions for the class are:

```
class Queue<E> { // the E will make it generic
    public Queue();
    public void add(E elementToEnqueue); // enqueue
    public E poll(); // dequeue
```

```
public E peek();  
public boolean isEmpty();  
}
```

(No need for a public version of `isFull`; if you find you need one, make it non-public.)

The names are, as mentioned in class, not the ones Dr. Ladd would choose but they are what `java.util.Queue` (and therefore the provided main program) uses.

Doubly-linked List

You will implement your queue as a linked structure. Your node class (the internal class in `Queue<E>`) will have a data field of type `E`. Since you will never call `new` with `E` (or an array of `E`), you do not need any *reflection-magic* as was needed for the circular buffer implementation done in class. You will take an `E` as a parameter to the node constructor and just set your data field.

You **must** implement a *doubly-linked list*. Each node has *two* references to other nodes, one called `prev` (points at the previous node in the list) and one called `next` (points at the next node in the list).

The first node has a `null` `prev` (nothing before it) and the last node has a `null` `next` (nothing after it).

The linked list class has two node references: `head` that refers to the **oldest** element that was enqueued and is still in the queue; and `tail` that refers to the **newest** element added to the queue.

See Horstmann's **Worked Example 16.1** for doubly-linked list structures.

Testing

This one will be tested by hand. List what cases you tested and how. The important thing is to test all the cases that Dr. Ladd can think of so that you already handle them correctly.

Part of that means knowing what “correctly” looks like.

Documentation

README

Must document how you tested. How do you know that it is right?

Must include instructions on how to **compile** and how to **run** the program as submitted.

Deliverables

Submission medium: git to Gitea at `cs-devel.potsdam.edu`.