PopcornQueue

Summary:
The program is an implementation of a queue data structure that stores items in an array called PopcornQueue. It has methods for inserting, removing, sampling, and iterating items in a random order. I believe that the program successfully achieves all the deliverables. This includes implementing the queue functionality, shuffling the items for a random iteration, handles resizing of the array and provides an iterator for for loops. I also included error handling for cases such as inserting null items, empty queues, and unsupported remove operation on the iterator. I believe that the Array based implementation is appropriate and meeted the requirements of ensuring that the memory usage is optimized by making sure that resizing of the array is done when it becomes sufficiently empty. I also made sure to take care of loitering by nullifying the reference to removed items in the array to prevent them from being held in memory.

Process:
For organizational strategy I started by understanding the requirements for each of the methods provided. I then made an IntelliJ class and pasted said methods into the PopcornQueue class. After that I made comments on the methods for how I plan to complete them and made sure to mark down the methods I didn't understand. I then thought about if I wanted to do an Array or a LinkedList. I found the Array for me personally easy to understand because I was most used to using arrays. For the methods I didn't understand I used the Algorithms fourth edition I also reviewed information given in the Possible Progress step in the PDF for Popcorn queue. If I was totally lost on what to do I always find the princeton website to be helpful in giving examples and giving a basic and good understanding. I also made sure to follow the PDF to ensure I was meeting requirements and trying my best to meet said requirements. For testing my code I used the code provided in the PDF to test it and would see if it matched closely with the example output. I would then use web-cat to make small changes and suggestions to properly optimize my code. I would also use IntelliJ built- in show context actions to help guide me when lost on a method which surprisingly came in handy on some parts of the assignment.

Design:
Describe any approaches to solving this problem that you considered, including the one you settled on. Discuss why your chosen design was the preferred one over the others you considered. Describe how your design met the
requirements of the assignment.
I knew right away that I wanted to implement an array based queue because I felt more knowledgeable with that than a Linked List. I also did some research when finding out what to use and found that Arrays provide constant-time access to indexes making it easier to retrieve a random item from a queue by just generating a random index. I feel that this is more efficient than a linked list where it would require traversing the list. I also found that arrays can be more memory efficient than linked lists because they do not require additional memory for things like pointers.

public class PopcornQueue<Item> implements Iterable<Item>

```
public class PopcornQueue<Item> implements Iterable<Item> {
    private Item[] items = (Item[]) new Object[1];
    private int size;
    private final Random random;
```

Implements iterable so it allows me to iterate over using a for-each loop. And uses type parameter Item that represents the type of item stored in the queue. Items is made to be a private instance variable that holds the items in queue and it is an array of Item[] and is initialized with an array size of 1 so it contains a single object instance.

{
public PopcornQueue() // construct an empty popcorn queue

```
public PopcornQueue() {
    size = 0;
    random = new Random();
}
```

Constructs an empty popcorn queue and initializes size to be 0 and creates a new random instance for the random variable.

public boolean empty() // is the popcorn queue empty?

```
// is the popcorn queue empty?
public boolean empty() {
    return size == 0;
}
```

A public method that checks if the queue is empty by comparing size with 0.

public int size() // return the number of items on the popcorn queue
This returns the current number of items in the queue Using return size;.

public void insert(Item item) // add the item

```
public void insert(Item item) {
    if (item == null) throw new IllegalArgumentException("Cant insert null item");
    if (size == items.length) resize( capacity: items.length * 2);
    items[size++] = item;
}
// remove and return a random item
```

This method inserts an item into the queue it first checks to see if the item is null and if so throws an Illegalargumentexception so a null item cant be inserted.then it checks size of the queue and if it's equal to the length of items array and if it does it calls the resize method to double the capacity of the array as shown above.then it adds the item to the items array at the index size representing the end of the queue.

public Item remove() // remove and return a random item

```
public Item remove() {
    if (empty()) throw new NoSuchElementException("PopcornQueue empty");
    int randomIndex = random.nextInt(size);
    Item removed = items[randomIndex];
    items[randomIndex] = items[--size];
    items[size] = null;
    if (size > 0 && size == items.length / 4) resize( capacity: items.length / 2);
    return removed;
}
```

This method is used to remove a random item from the popcorn queue first it uses the if empty throw new NoSuchElementException which checks if it is empty by calling empty method. Then it generates a random index within a range of the size of the queue and used to select a random item that will be removed. Then it retrieves the item randomly and assigns it to the removed variable. It then replaces the item with a randomly generated index with the item at the last index of items removing and replacing the item of the queue. It then sets the last item at the index of items as null which clears the reference and removes the item. It then checks if size if the queue is less than the current capacity of items and if so it resizes the items to prevent the array from wasting memory.

public Item sample() // return a random item (but do not remove it)

```
public Item sample() {
    if (empty()) throw new NoSuchElementException("PopcornQueue empty");
    int randomIndex = random.nextInt(size);
    return items[randomIndex];
}
```

This sample method returns a random item from the queue without removing it. It first will check if the queue is empty and if so throws NoSuchElementException stating such it will then generate a random index and then return the item at the index from items.

public Iterator<Item> iterator() // return an independent iterator over items in random order

```
private class PopcornQueueIterator implements Iterator<Item> {
    private int currentIndex;
    private final Item[] shuffledItems = (Item[]) new Object[size];

    public PopcornQueueIterator() {
        currentIndex = 0;
        System.arraycopy(items, srcPos: 0, shuffledItems, destPos: 0, size);
        shuffle(shuffledItems);
    }

    public boolean hasNext() {
        return currentIndex < size;
    }

    public Item next() {
        if (!hasNext()) throw new NoSuchElementException("No items");
        return shuffledItems[currentIndex++];
    }

    public void remove() {
        throw new UnsupportedOperationException("Remove op not supported");
    }

    private void shuffle(Item[] array) {
        for (int i = 0; i < array.length; i++) {
            int randomIndex = random.nextInt( bound: i + 1);
            Item temp = array[randomIndex];
            array[randomIndex] = array[i];
            array[i] = temp;
        }
    }
}
```

It first has a construct for the iterator and it initializes the current index to 0 and creates a copy of the items array called shuffled items it then will use the System.array copy method to copy items from items. Then it will call the shuffle method to shuffle the items. The hasNext checks if there are any more items to iterate and it will return true if the current index is less than the size of the queue. It then uses Item next to return the next item in a shuffled order first checking if there are any more items to iterate. It then shuffles the items with the shuffle method using the Fisher-Yates shuffle algorithm.

Lessons learned:
When starting this assignment I was pretty intimidated because I felt I was still lacking in my knowledge of queues with linked-lists and arrays. After completing the assignment I feel my understanding of using arrays with queues is better than it was before. Using resources like the algorithms books and the princeton website were helpful when understanding how to do this assignment and gave me more of a knowledgeable understanding while also helping me apply it to my code. In future assignments using queue I will try to branch out more using things I'm not comfortable with like Linked Lists so that I may broaden my knowledge like I have done when implementing arrays in this assignment. I should also remember for future projects that teachers are there to help and when stuck on what to do go to them for guidance and they can help in the right direction. Overall I feel that this assignment was challenging and made me think and problem solve and also gave me a better understanding of queues for going into the final and future applications.