English Name: ____________________

Assignment: `class RandomizeIndexOrder`

This question involves the creation and use of a random index sequence generator for sampling without replacement. A RandomIndexOrder object generates a random permutation of indices from 0 to n-1 and allows sequential access to these indices. The class supports the following behaviors:

- Creating a new generator with a specified number of indices
- Getting the next index in the random sequence
- Reporting the number of indices already returned
- Reporting the number of indices remaining

The following table contains a sample code execution sequence and the corresponding results.

	Statements	**Value Returned**	**Comment**
	`RandomIndexOrder r = new RandomIndexOrder(5);`	`(none)`	Creates a new random sequence generator for 5 indices (0 through 4)
	`r.next();`	`3`	Returns the first random index in the sequence. In this case, 3 is returned.
	`r.getReturnedCount();`	`1`	Returns the count of indices already returned. One index has been returned so far.
	`r.getRemainingCount();`	`4`	Returns the count of indices still available. Four indices remain to be returned.
	`r.next();`	`1`	Returns the next random index in the sequence. In this case, 1 is returned.
	`r.getReturnedCount();`	`2`	Two indices have been returned.
	`r.next();`	`4`	Returns the next random index.
	`r.next();`	`0`	Returns the next random index.
	`r.next();`	`2`	Returns the next random index. This is the fifth and final index.
	`r.getReturnedCount();`	`5`	All five indices have been returned.
	`r.getRemainingCount();`	`0`	No indices remain to be returned.
	`r.next();`	`-1`	Returns -1 when no more indices are available.

Write the complete `RandomizeIndexOrder` class. Your implementation must meet all specifications and conform to the example.

Assignment: `class RandomizeIndexOrder`


```java
public class RandomIndexOrder {
    private int[] shuffledIndexes;
    private int currentIndex;
    public RandomIndexOrder(int size) {
        shuffledIndexes = new int[size];
        currentIndex = 0;
        populateIndexes();
    }
    public int next() {
        if(currentIndex < shuffledIndexes.length) {
            int index = shuffledIndexes[currentIndex];
            currentIndex++;
            return index;
        } else {
            return -1;
        }
    }
    public int getReturnedCount() {
        return currentIndex;
    }
    public int getRemainingCount() {
        return shuffledIndexes.length – currentIndex;
    }
    public boolean hasNext() {
        return currentIndex < shuffledIndexes.length;
    }
    private void populateIndexes() {
        // Track filled positions
        boolean[] filled = new boolean[shuffledIndexes.length];
        for(int value = 0; value < shuffledIndexes.length; value++) {
            int index;
            // find a random index that has not been written to
            do {
                index = (int)(Math.random() * shuffledIndexes.length);
            } while(filled[index]);
            filled[index] = true;
            shuffledIndexes[index] = value;
        }
    }
}
```