UNIVERSITY OF WESTMINSTER

INFORMATICS INSTITUTE OF TECHNOLOGY

Software Development II

Lecture 09 – Queues and Stacks

Reading : Reading list: Big Java Ch13

Announcement

Coursework deadline is March 25th at 1pm.

- Submission instructions and links are available in Blackboard.
- Ensure to follow instructions to avoid penalties.
- Coursework VIVA Schedule will be published after the deadline.

From Last Week

- Classes and Objects
- File Handling
  - Create a file
  - Write into a file
  - Read from a file
- File related Exceptions
- Testing
  - Testing Strategies
  - Writing testing evidence

Outline

Queues

- FIFO
- Circular queue
- Linear Queue

Stacks

- LIFO

Other data structures

- Set
- List
- Queue
- Map
- Iterators

Data Structures (lists, queues, stacks,…)

- Data structures are standard ways to store and retrieve data – used throughout programming.
- Many programs need to keep lists of items in order to add, retrieve, sort, and delete item(s) from the list. For example, the list of programs your PC's processor is currently running.
- A list may hold items in a stack or queue (e.g. customer orders placed in a queue).
- Java includes library routines to store and retrieve data that you can use.

Stacks and Queues

- A stack is a collection of elements with “last-in, first-out” retrieval.
- Elements are added on the top and removed from the top.

- A queue is a collection of elements with “first- in, first-out” retrieval.
- Elements are added at the end and removed from the front.

Queues

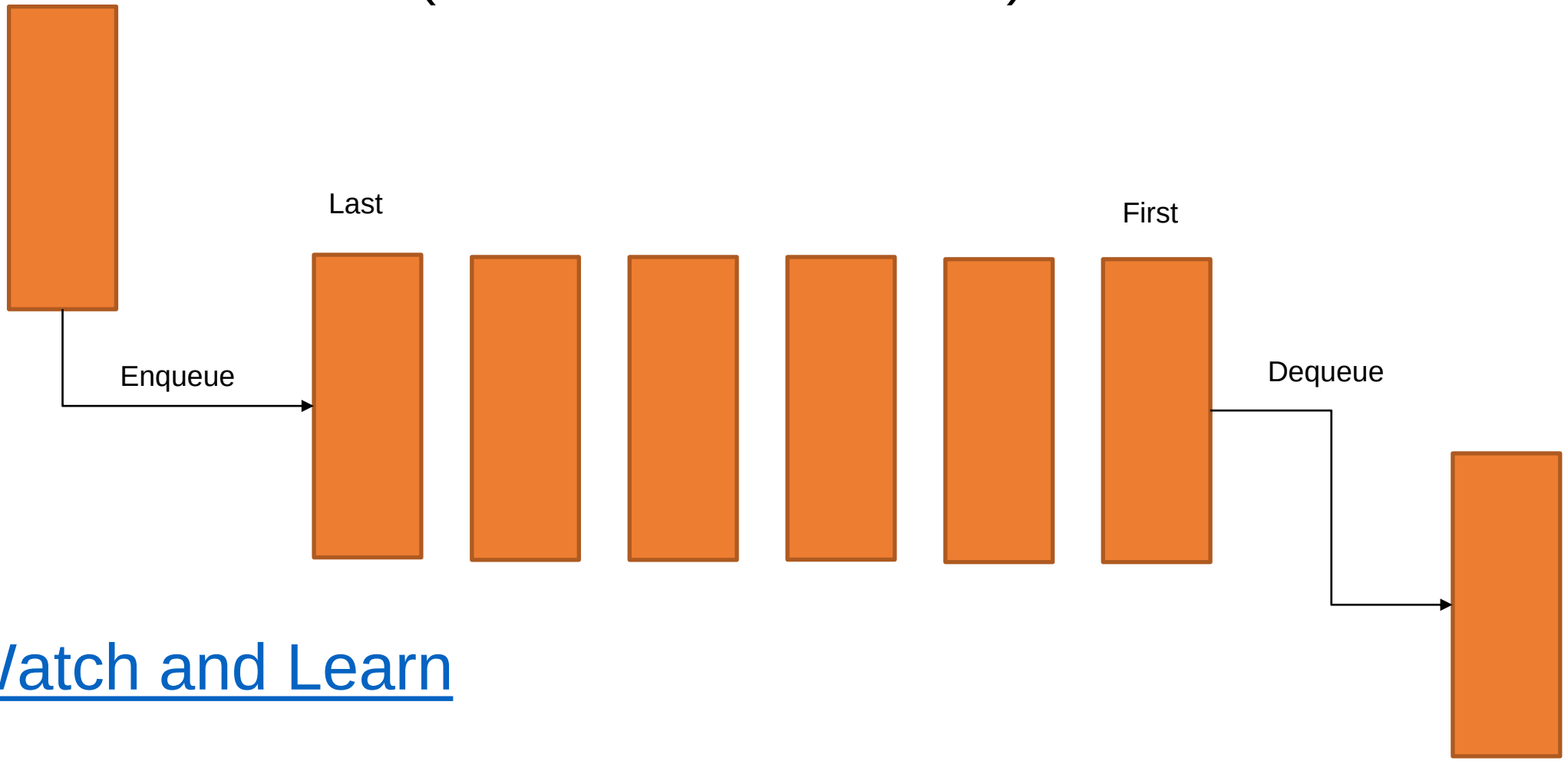
In computing, a queue is a data structure that stores and manages a collection of elements in a specific order.

Examples:

- People waiting to be served at a restaurant
- Supermarket tills
- Website to buy tickets for a concert (and any shopping system)

First In First Out (FIFO) queue

- The first element that is added to the queue is the first one to be removed (first comes first serves).



Watch and Learn

Programming a Queue

- Recall that a queue is a data structure containing an ordered collection of values of the same data type which can be accessed at both ends.
- Data items are added at the rear (aka end) of the queue and removed from the front of the queue.
- Data items are retrieved in the same order as they are added to the queue, that is, data is processed on a first come, first served basis.
- A queue is known as a **First-In-First-Out (FIFO)** data structure since the first item added to the queue will be the first item removed.

Linear Queue – Operations

- Adding Items to the queue (Enqueue)
- Removing Items from the queue (Dequeue)
- Checking whether the queue is empty
- Clear the queue

Linear Queue –Operations(enqueue)

```
public class FIFO {
    private String[] queue;
    int num_elements = 0;

    public FIFO(int capacity) {
        queue = new String[capacity];
    }

    public void enqueue(String item) {
        if (num_elements == queue.length) { // full
            System.out.println("Queue is full. Element not added.");
            return;
        }
        queue[num_elements] = item;
        num_elements++;
    }
```

Linear Queue –Operations contd.(dequeue)

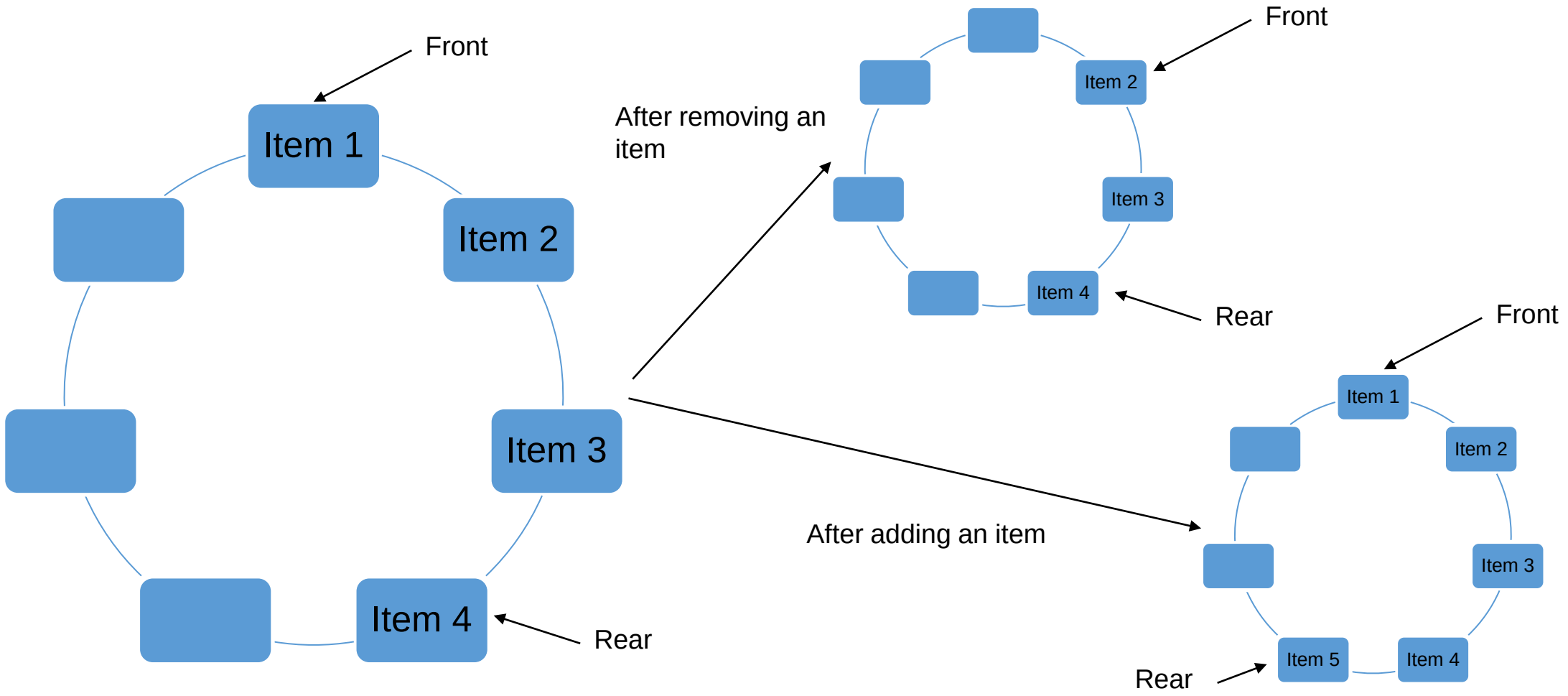
```
    public String dequeue() {
        if (num_elements == 0) { // empty
            System.out.println("Queue is empty.");
            return "";
        }
        String item = queue[0];
        num_elements--;
        for (int i = 0; i <= num_elements-1; i++){
            queue[i] = queue[i+1];
        }
        return item;
    }
    public void print() {
        if (num_elements == 0) {
            System.out.println("Queue is empty.");
            return;
        }
        for (int i = 0; i < num_elements; i++) {
            System.out.print("| " + i + ": " + queue[i] + " |");
        }
        System.out.println();
    }
}
```

FIFO Queue : Usage

```
public static void main(String[] args){
        FIFO queue = new FIFO(3);
        queue.enqueue("A");
        queue.enqueue("B");
        queue.enqueue("C");
        // Testing if queue is full
        queue.enqueue("D"); // This should print "Queue is full. Element not added."
        // Dequeueing 2 elements
        System.out.println(queue.dequeue()); // This should print "A"
        System.out.println(queue.dequeue()); // This should print "B"
        // Enqueueing 2 more elements
        queue.enqueue("D");
        queue.enqueue("E");
        // Dequeueing all elements
        System.out.println(queue.dequeue()); // This should print "C"
        System.out.println(queue.dequeue()); // This should print "D"
        System.out.println(queue.dequeue()); // This should print "E"
        // Testing if queue is empty
        System.out.println(queue.dequeue()); // This should print "Queue is empty."
}
```

Circular queue

- A circular queue is a data structure where last element is connected to the first element to form a circular structure, where the front and rear of the queue are no longer fixed positions.



Circular queue: code

```
public class FIFOCircular {

    private String[] queue;
    int front = 0;
    int rear = 0;
    int num_elements = 0;

    public FIFOCircular(int capacity) {
        queue = new String[capacity];
    }

    public void enqueue(String item) {
        if (num_elements == queue.length) { // full
            System.out.println("Queue is full. Element not added.");
            return;
        }
        queue[rear] = item;
        rear = (rear + 1) % queue.length;
        num_elements++;
    }
```

Circular Queue: code (cont.)

```
    public String dequeue() {
        if (num_elements == 0) { // empty
            System.out.println("Queue is empty.");
            return "";
        }
        String item = queue[front];
        front = (front + 1) % queue.length;
        num_elements--;
        return item;
    }

    public void print() {
        if (num_elements == 0) {
            System.out.println("Queue is empty.");
            return;
        }
        for (int i = front; i != rear; i = (i + 1) % queue.length) {
            System.out.print("| " + i + ": " + queue[i] + " |");
        }
        System.out.println();
    }
}
```

Circular Queue: usage

```
public static void main(String[] args){
        FIFOCircular queue = new FIFOCircular(3);
        queue.enqueue("A");
        queue.enqueue("B");
        queue.enqueue("C");
         // Testing if queue is full
        queue.enqueue("D"); // This should print "Queue is full. Element not added."
        // Dequeueing 2 elements
        System.out.println(queue.dequeue()); // This should print "A"
        System.out.println(queue.dequeue()); // This should print "B"
        // Enqueueing 2 more elements
        queue.enqueue("D");
        queue.enqueue("E");
        // Dequeueing all elements
        System.out.println(queue.dequeue()); // This should print "C"
        System.out.println(queue.dequeue()); // This should print "D"
        System.out.println(queue.dequeue()); // This should print "E"
        // Testing if queue is empty
        System.out.println(queue.dequeue()); // This should print "Queue is empty."
}
```

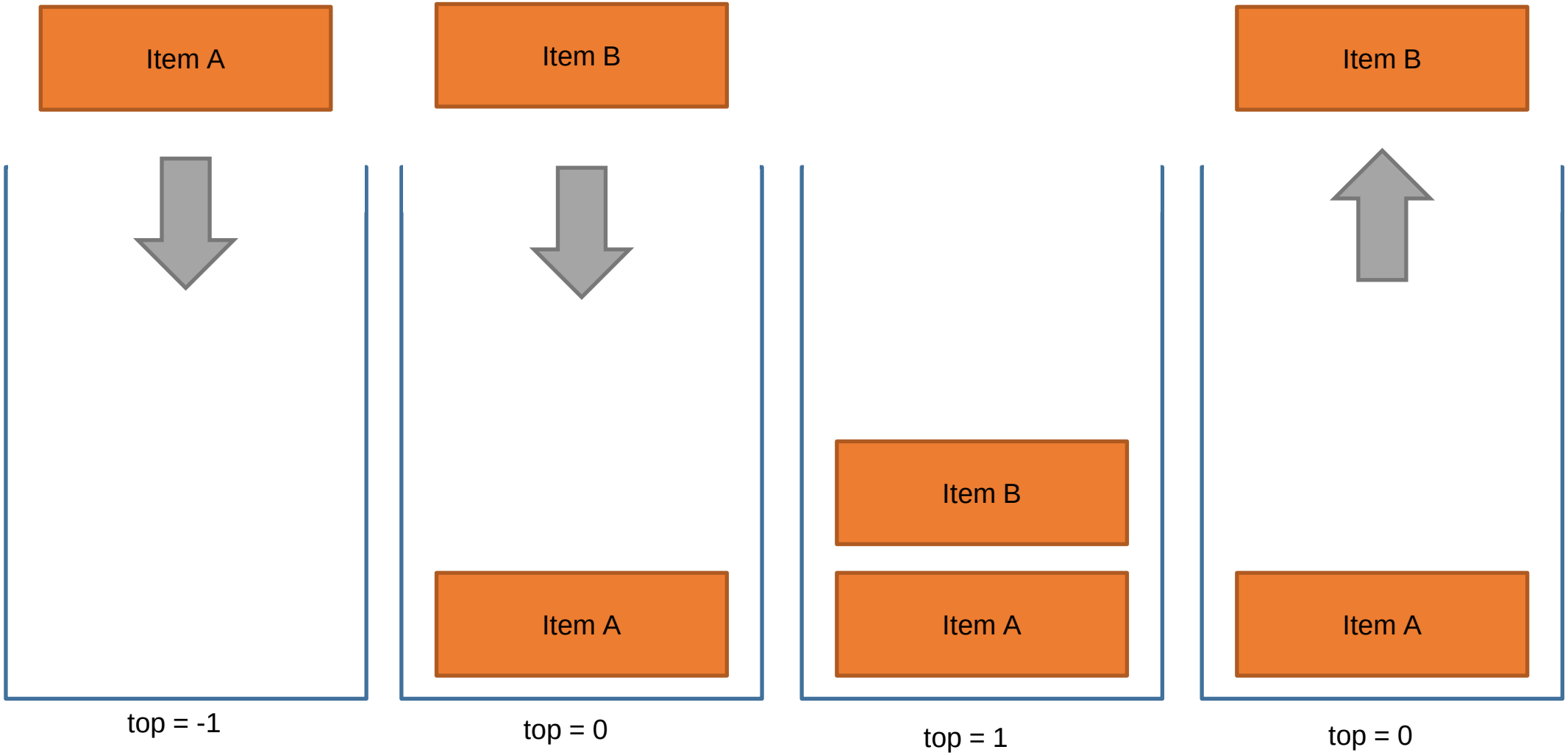
Stacks: Last In First Out (LIFO)

In computing, a stack is a data structure that allows adding or removing items only from the top.

Examples:

- Undo/redo operations
- Browser history (back)

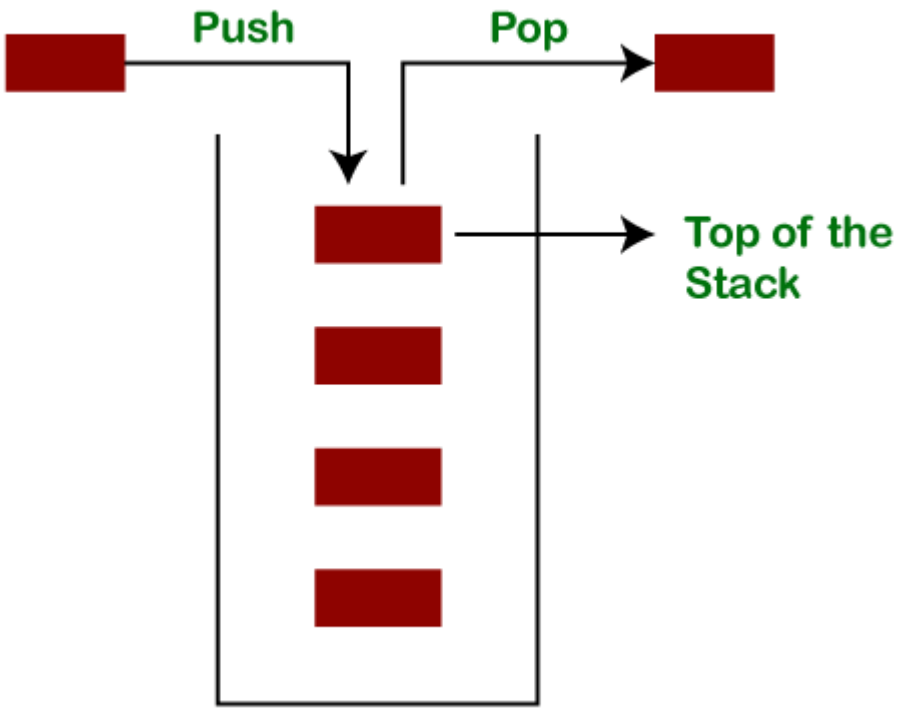
LIFO


Item A
Item B
Item B
Item A
Item B
Item A
Item A
top = -1
top = 0
top = 1
top = 0

Programming a Stack

- Recall that a stack is a data structure containing a collection of items of the same data type which can only be accessed at one end, the top of the stack.
- Items can be added ("pushed") onto the top of the stack, and items can be removed ("popped") from the top of the stack.
- A stack is known as a **Last-In-First-Out (LIFO)** data structure since the last item added to the stack will be the first item removed.

Programming a Stack



Push operation

[Watch and Learn]

Stack Operations (Push)

```
public class StackExample {
    private String []stack;
    private int first;

    public StackExample(int capacity){
        stack=new String[capacity];
        first=-1;
    }

    public void push(String item) {
        if (first == stack.length - 1) { //full
            System.out.println("Stack is full. Element not added.");
            return;
        }
        first++;
        stack[first] = item;
        System.out.println("Element added.");
    }
```

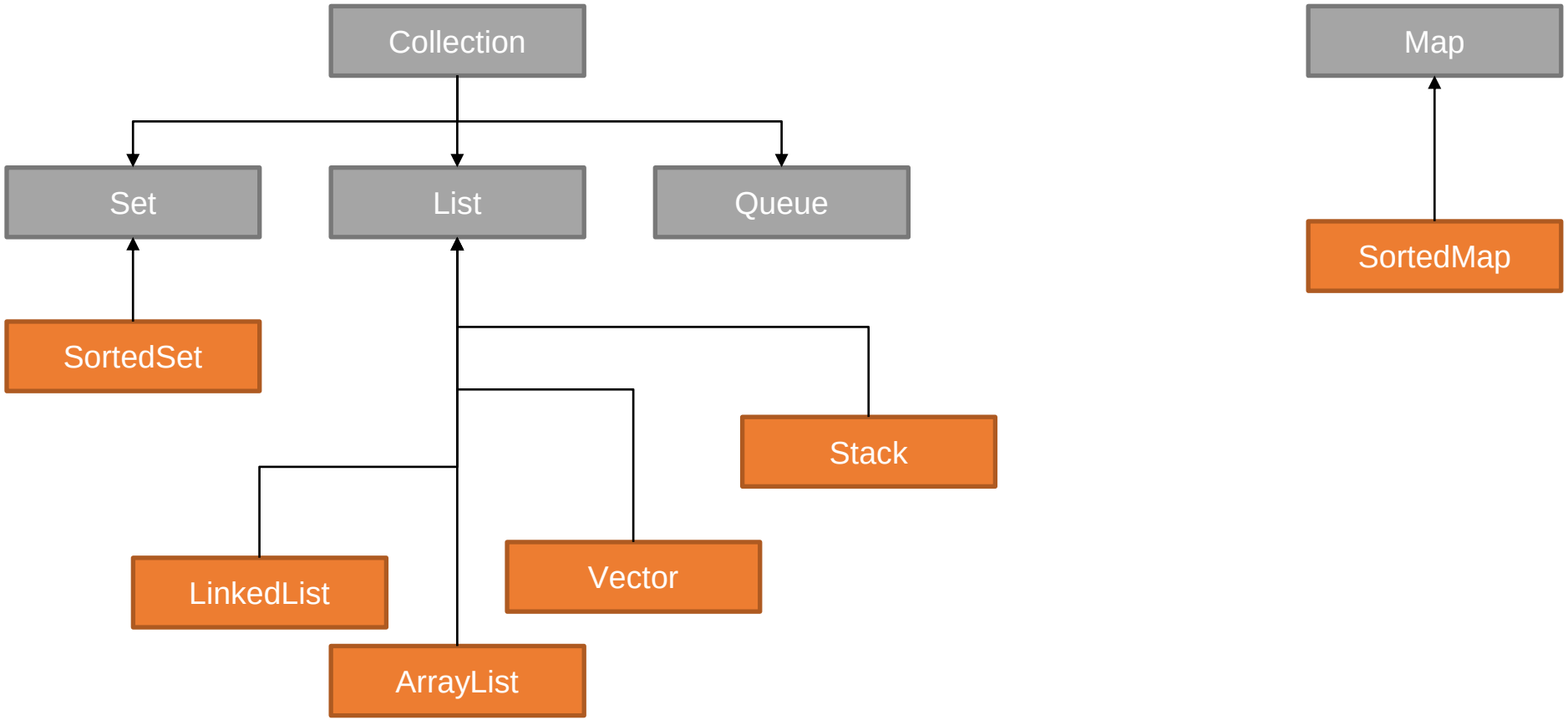
Stack Operations contd. (Pop)

```
// Remove element from stack
    public String pop() {
        if (first == -1) { //empty
            System.out.println("Stack is empty.");
            return "";
        }
        String item = stack[first];
        first--;
        System.out.println("Element removed.");
        return item;
    }

    public void print() {
        if (first == -1) {
            System.out.println("Stack is empty.");
            return;
        }
        for (int i = first; i >= 0; i--) {
            System.out.print("| " + i + ": " + stack[i] + " |");
        }
        System.out.println();
    }
}
```

LIFO stack: usage

```
public class Main {
    public static void main(String[] args) {

        StackExample stack = new StackExample(3);
        stack.push("A");
        stack.push("B");
        stack.push("C");
        // Testing if queue is full
        stack.push("D"); // This should print "stack is full. Element not added."
        // Pop 2 elements
        System.out.println(stack.pop()); // This should print "C"
        System.out.println(stack.pop()); // This should print "B"
        // Enqueueing 2 more elements
        stack.push("D");
        stack.push("E");
        // Dequeueing all elements
        System.out.println(stack.pop()); // This should print "E"
        System.out.println(stack.pop()); // This should print "D"
        System.out.println(stack.pop()); // This should print "A"
        // Testing if queue is empty
        System.out.println(stack.pop()); // This should print “Stack is empty."
    }
}
```

Other Data Structures :Java Collections Framework



Java Collections Framework – Other Data Structures

- **Set**: A collection that contains no duplicates

Methods such as: add, clear, contains, isEmpty, iterator, remove, size, …
https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/Set.html

- **List**: A collection that may contain duplicates

Methods such as: add, clear, contains, get, indexOf, isEmtpy, remove, size, …
https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/List.html

- **Queue**:

Methods such as: add, element, peek, remove, etc.
https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/Queue.html

- **Map**:

Methods such as: clear, containsKey, containsValue, get, isEmpty, keySet, put,
https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/Map.html

List Iterators (Revision from Arrays)

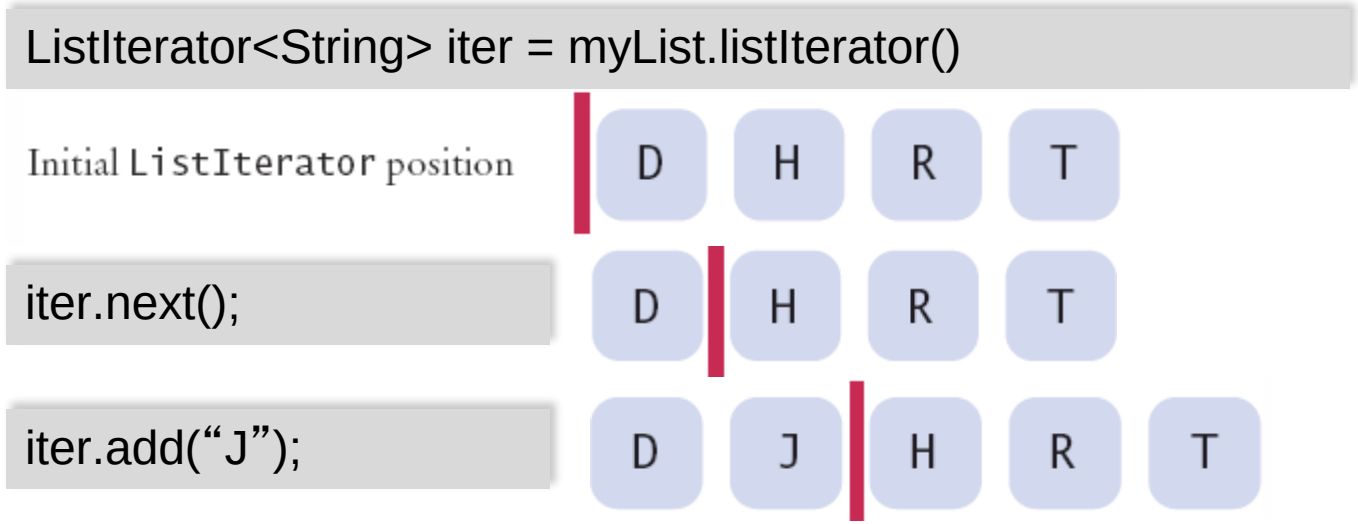
- When traversing a LinkedList, use a ListIterator
  - Keeps track of where you are in the list.

```
LinkedList<String> employeeNames = . . .;
ListIterator<String> iter = employeeNames.listIterator()
```

- Use an iterator to:
  - Access elements inside a linked list or Array List
  - Visit other than the first and the last nodes

Using Iterators

- Think of an iterator as pointing **between** two elements



- Note that the generic type for the listIterator must match the generic type of the LinkedList

Iterator and ListIterator Methods

- Iterators allow you to move through a list easily
  - Similar to an index variable for an array

Table 3 Methods of the `Iterator` and `ListIterator` Interfaces

`String s = iter.next();`	Assume that `iter` points to the beginning of the list `[Sally]` before calling `next`. After the call, `s` is `"Sally"` and the iterator points to the end.
`iter.previous();` `iter.set("Juliet");`	The `set` method updates the last element returned by `next` or `previous`. The list is now `[Juliet]`.
`iter.hasNext()`	Returns `false` because the iterator is at the end of the collection.

`if (iter.hasPrevious())`
`{`
`  s = iter.previous();`
`}`
	`hasPrevious` returns `true` because the iterator is not at the beginning of the list. `previous` and `hasPrevious` are `ListIterator` methods.

`iter.add("Diana");`	Adds an element before the iterator position (`ListIterator` only). The list is now `[Diana, Juliet]`.
`iter.next();` `iter.remove();`	`remove` removes the last element returned by `next` or `previous`. The list is now `[Diana]`.

Iterators and Loops

- Iterators are often used in while and “for-each” loops
  - hasNext returns true if there is a next element
  - next returns a reference to the value of the next element

```
while (iterator.hasNext())
{
    String name = iterator.next();
    // Do something with name
}
```

```
for (   String name : employeeNames)

{
        // Do something with name
}
```

- Where is the iterator in the “for-next” loop?
  - It is used ‘behind the scenes’

Exercise 1

- You are given a list of tasks to perform.
- Each task has a priority level.
- Which structure would you use, a queue or a stack?

Exercise 2

- A food delivery service for a restaurant has asked you to implement their system to process orders.
- Each order contains the food or drink that the customer wants and the time the order was placed.
- Which structure would you use, a queue or a stack?

Exercise 3

- During COVID, many supermarket’s website had a limit on the number of people who could use online shopping service at one time.
- When the number of people using the service reached the maximum, customers were asked to wait.
- You are asked to implement the waiting system for these customers.
- Which structure would you use, a queue or a stack?

Exercise 4

- You are given a string (Java code) that contains parentheses ‘(‘ or ‘)’, square brackets ‘[‘ or ‘]’ and curly braces ‘{‘ or ‘}’.
- You need to check if the string has balanced brackets, parentheses and curly braces.
- Which structure would you use, a queue or a stack?

Exercise 5

- The emergency department of a hospital has asked you to implement their waiting list system.
- Patients are given a priority after triage.
- When a new patient comes, is given a priority and is added into the waiting list.
- Which structure would you use, a queue or a stack?

Feedback: Formative test week 9 (Blackboard)

- In Learning Resources > Week 9 you will find a formative test to get feedback on the content of this lecture.

HackerRank: Interview questions on this topic

Solve the following exercises in HackerRank on exception handling:

- Queues using Two Stacks
- Balanced Brackets
- Equal Stacks
- Truck Tour
- Game of Two Stacks

Thank You !!