# Chapter 7

# Data Structures: Stacks & Queues

Ahmad Al-Rjoub

**CSC 113**

**King Saud University**
**College of Computer and Information Sciences**
**Department of Computer Science**

# Objectives

In this chapter you will learn:

- To create and manipulate dynamic data structures, such as stacks and queues.
- Various important applications of linked data structures.
- How to create reusable data structures with classes, inheritance and composition.

# Outline

1. Introduction
2. Stack
3. Queue

# 1. Introduction

- A *data structure* is organiTes information so that it efficient to access and process.

- In this chapter we study several dynamic data structures -- *stacks* and *queues*.

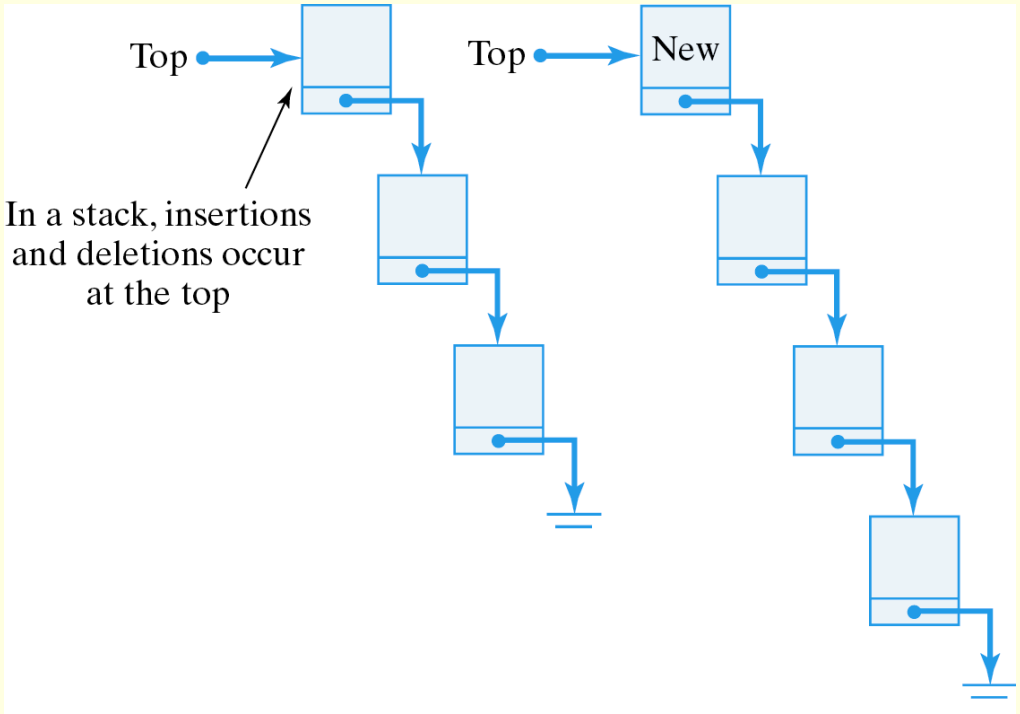
# 2. The Stack ADT

**Stacks**

- **Last-in, first-out (LIFO) data structure**
  - **Method `push` adds a new node to the top of the stack**
  - **Method `pop` removes a node from the top of the stack and returns the data from the popped node**
- **Program execution stack**
  - **Holds the return addresses of calling methods**
  - **Also contains the local variables for called methods**
- **Used by the compiler to evaluate arithmetic expressions**

# 2. The Stack ADT

- A *stack* is a list that limits insertions and removals to the front (*top*) of the list.



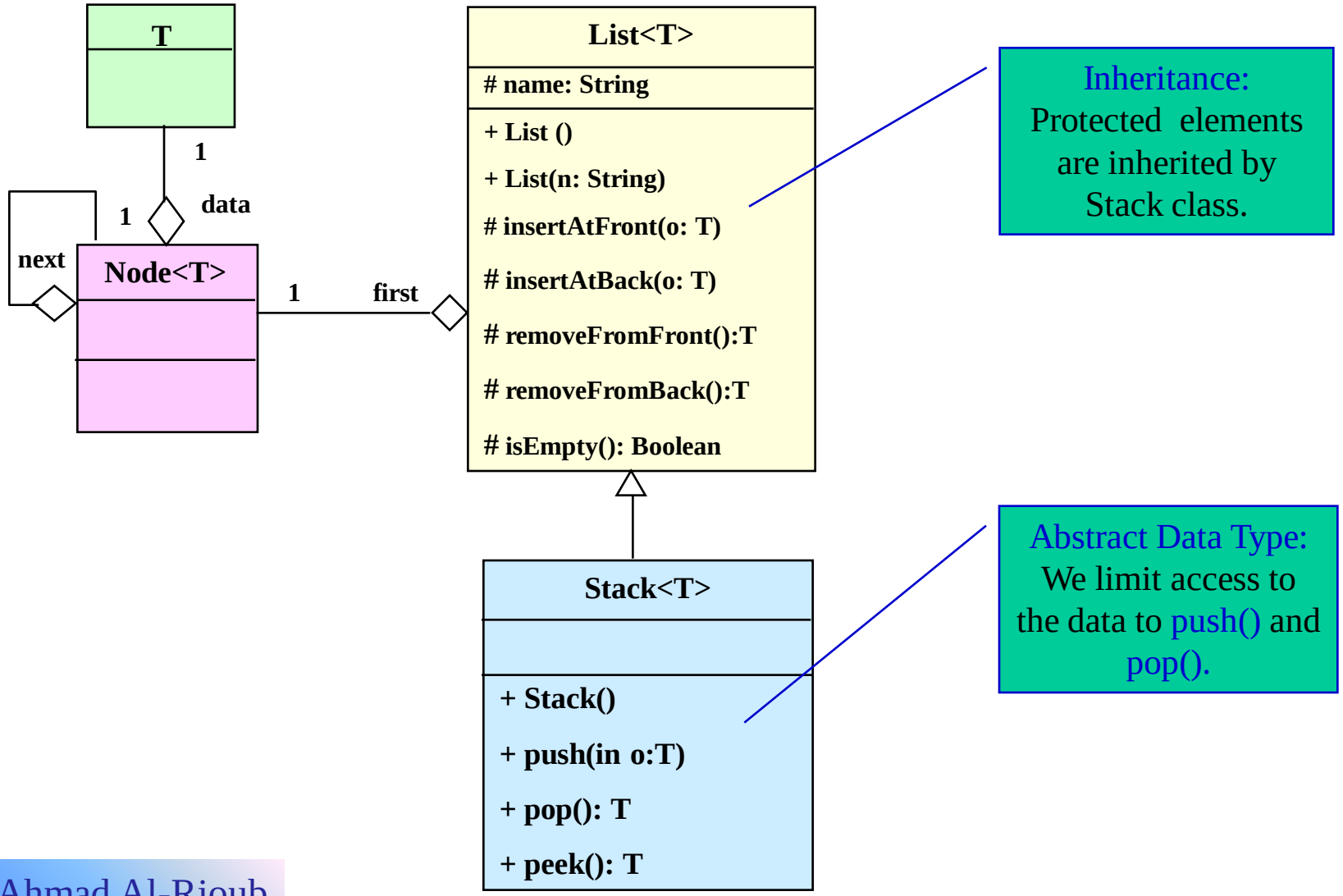
# Stack class that inherits from List

**Stack methods `push`, `pop`, `isEmpty` and `print` are performed by inherited methods `insertAtFront`, `removeFromFront`, `isEmpty` and `print`**

- **`push` calls `insertAtFront`:** insert an object onto the top of the stack.
- **`pop` calls `removeFromFront`**: remove the top object from the stack.
- **`isEmpty` and `print` can be called as inherited:** returns true if the stack is empty.
- **`.Peek`**: retrieve the top object without removing it.

# Stack class that inherits from List

- A Stack is very easy to design as an extension of our generic List structure.



# The Generic Stack Class: Implementation

```
public class Stack<T>  extends List<T>{

 public Stack() {
   super("ST");
 }

public void push( T obj )  {
   insertAtFront(obj);
 }

public T pop() {

      if (isEmpty())

         throw new Exception("Stack is empty");

     return removeFirst();
 }

public T peek()  {

    if (isEmpty())

       throw new Exception("Stack is empty");

    return getFromFront();  }

} // Stack
```

# Testing the Stack Class

```
Testing the Stack Class//==Stack of Student elements =
Stack<Student>  ST= new Stack<Student>();
Student s1 =new Student("Saad");
s1.setScore(10,20,15);
s1.computeCourseGrade();
Student s2=new Student("Ali");
s2.setScore(10,50,40);
s2.computeCourseGrade();
Student s3=new Student("Nabil");
s3.setScore(30,10,15);
s3.computeCourseGrade();
Student s4=new Student("Sami");
s4.setScore(32,14,44);
s4.computeCourseGrade();
ST.push(s1);
ST.push(s2);
ST.push(s3);
ST.push(s4);
```

```
// display all inofrmations inside Stack  and calculate the
average of the totalmarks  for all passed students
  int sum=0, nb=0;
  try {
    while (true) {
      Student st =ST.pop();
      System.out.println(st.getName() + "  has " +
           st.getCourseGrade()+ "  and total marks = "+
           st.getTotal());
       if (st.getCourseGrade().equals("Pass"))
       {
          sum+=st.getTotal();
          nb++;
        }
      }
    } catch(Exception e) {}
   System.out.println("The average of total marks for passed
     students is : "+ 1.0*sum/nb);
    }
}// End of main
```