Object Oriented Programming & Data Structures

Object Oriented Programming

Object Oriented Programming (OOP):

- ✘ A programming paradigm.
- ✘ A design philosophy based on the concept of objects (ADTs).
- ✘ Programs consist of objects that interact with one another.
- ✘ Objects are instances of classes (the class is their type).

An Analogy Example

Think of a car as group of objects that interact (by message passing):

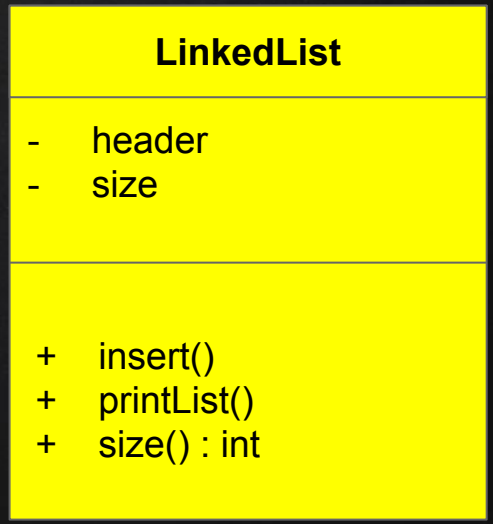
- ✘ Wheels that can roll.
- ✘ An engine or motor that rolls the wheels.
- ✘ A steering wheel that turns and can also ask the wheels to turn.
- ✘ Brakes that stop the wheels from rolling.
- ✘ An emission system that removes the flue gas produced by the engine.

Objects & Classes

Class:

- ✘ A description of the common properties of an object.
- ✘ A concept.
- ✘ A part of a program.

Class



Objects & Classes

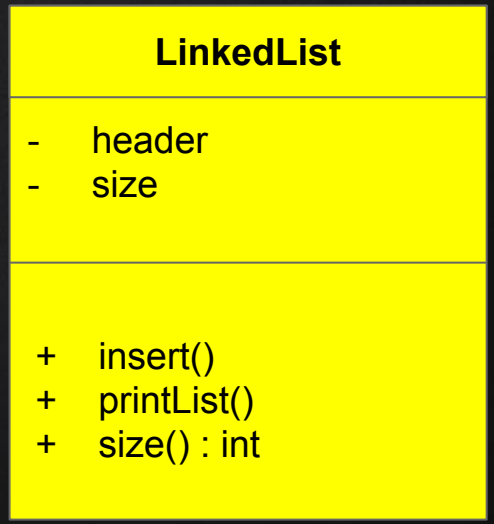
Object:

An encapsulation of data.

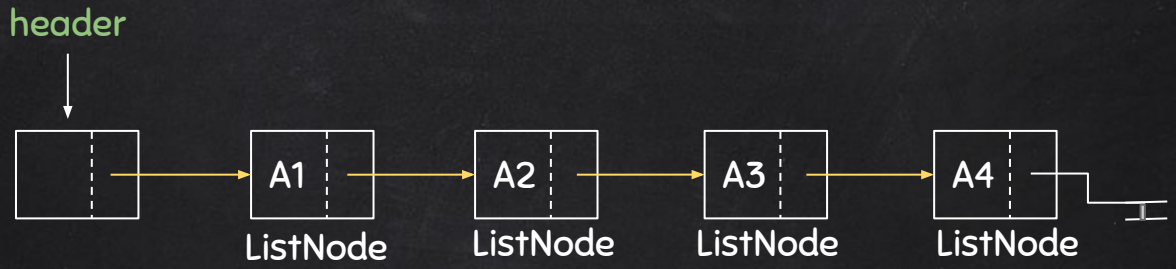
An object has:

- ✘ Identity (a unique reference).
- ✘ A state (variables) .
  - The depth of a tree.
  - The size of a list.
- ✘ Behavior (methods).
  - insert(Element e).
  - pop().

Class



Objects & Classes



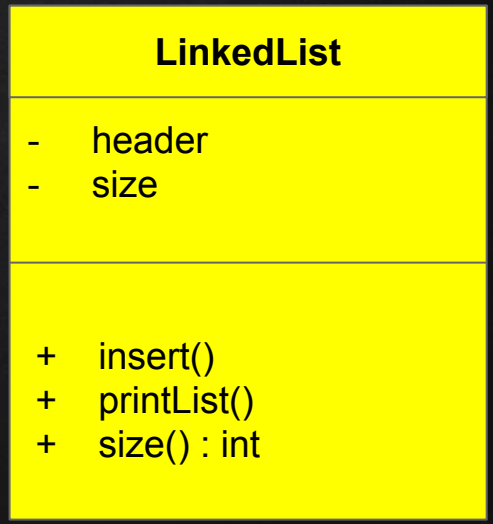

```
class LinkedList:

    def __init__(self):
        self.head = ListNode()

    def insert(self, data):
        new_node = ListNode(data, self.head.get_next())
        self.head.set_next(new_node)
        size += 1

    def printList(self):
        current = self.head.get_next()
        while current is not None:
            print current.get_data()
            current = current.get_next()
```

Objects & Classes
Encapsulation

Encapsulation:

- ✘ Classes are defined as a list of a abstract attributes.
- ✘ Storing data and functions in a single unit (class) is encapsulation.
- ✘ Data cannot be accessible to the outside world and only functions stored in the class can access it.

Class



Interfaces

Interface:

- ✘ Provides the description of the behaviour of a class.
- ✘ Considered the set of public methods and attributes.
- ✘ Operations must be independent from each other.
- ✘ Good practice: Separate interface and its implementation (e.g., Java).

Interfaces

Example:

Informatics Forum buys a cctv model named SuperSpy9000.

It monitors the building.

When there is an intruder it sets an alarm on and does stuff to kick him out.

A year later IF buys the new model named SuperSpy10000.

This model also fires warning laser shots in order to scare the intruder.

If the devices implement the same interface we do not need to change our code in order to use the new system in the building!

Abstract Data Types & Data structures

Abstract Data Types (ADT):

- ✘ A model that describes a particular set of behaviours from the perspective of a user of the data.
- ✘ An object.
- ✘ Ex: stack, queue, dictionary, set, etc.

Data Structures:

- ✘ Concrete representations of the data, or a strategy to implement ADT.
- ✘ Ex: array, linked list, hash table, trees, graphs, etc.

Data Structures

- ✘ A container of stuff (data)
- ✘ Operations that we can do:
  - `add(x)`
  - `remove(x)`
  - `empty(x)`
  - `find(x)`

List

Index	1	2	3	4

- ✘ A collection of items. Each item has a position.
- ✘ Can access any item using its index.
- ✘ Example: arrays
  - Allow random access
  - Insertion/deletion is expensive!

Stack

- ✘ Access is restricted to the most recently inserted item.
- ✘ Operations take a constant amount of time.

Queue

- ✘ Access is restricted to the least recently inserted item.
- ✘ Operations take a constant amount of time.

Priority Queue

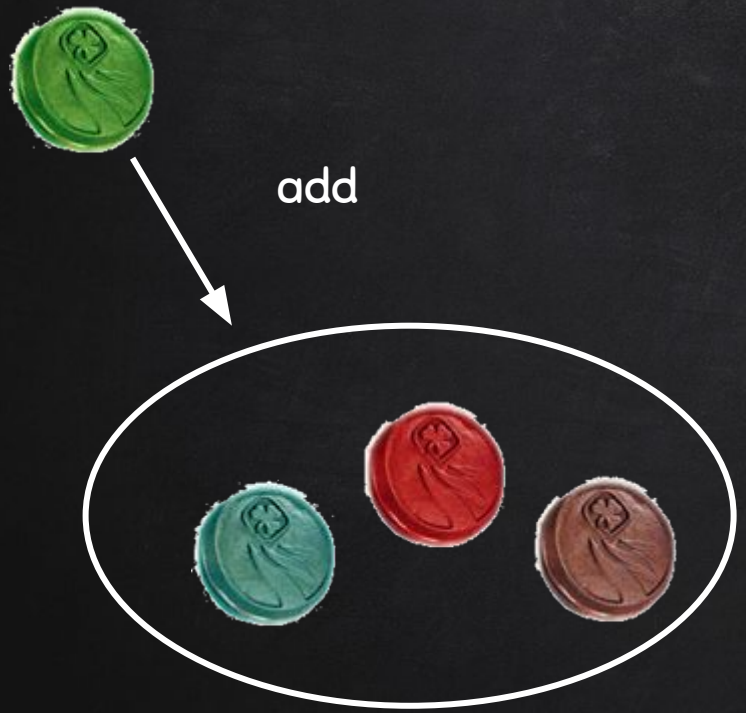
- ✘ A queue in which each item can have a priority value.
- ✘ Access is restricted to item with the highest priority.

Maps

Keys	Z01	AB3	UY6	HAA

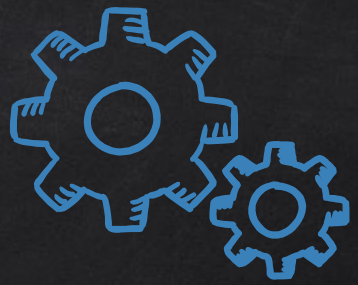
- ✘ A collection of (key, value) items.
- ✘ Keys must be unique, but values don't.
- ✘ Access to an item by using its key.

Set

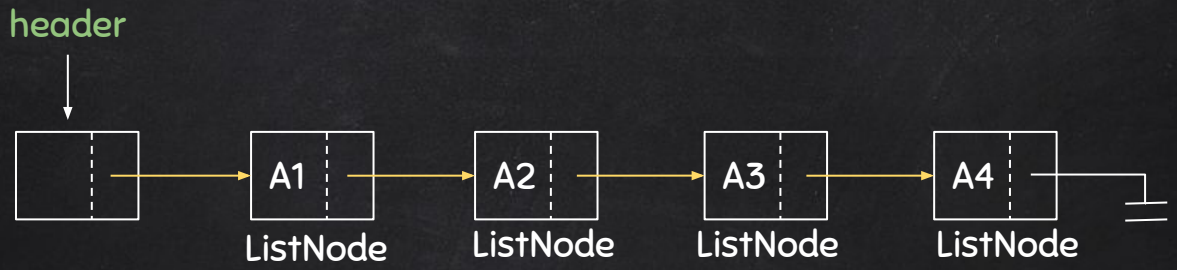


- ✘ A data structure that contains no duplicates.

Examples



Linked List




```
class ListNode:

    def __init__(self, d=None, n=None):
        self.data = d
        self.next_node = n

    def get_data(self):
        return self.data

    def get_next(self):
        return self.next_node

    def set_next(self, new_next):
        self.next_node = new_next
```

```
class LinkedList:

    def __init__(self):
        self.head = ListNode()

    def insert(self, data):
        new_node = ListNode(data, self.head.get_next())
        self.head.set_next(new_node)

    def printList(self):
        current = self.head.get_next()
        while current is not None:
            print current.get_data()
            current = current.get_next()
```

Superclass

```
class List:
    def __init__(self):
    def isEmpty(self):
    def makeEmpty(self):
    def insert(self,x,current):
    def remove(self,x)
```

```
class LinkedList(List):
    def __init__(self):
    def insert(self,x,current):
    def remove(self,x)
    ...
```

Subclass

```
class DoublyLinkedList(List):
    def __init__(self):
    def insert(self,x,current):
    def remove(self,x)
    ...
```

Subclass

Stack

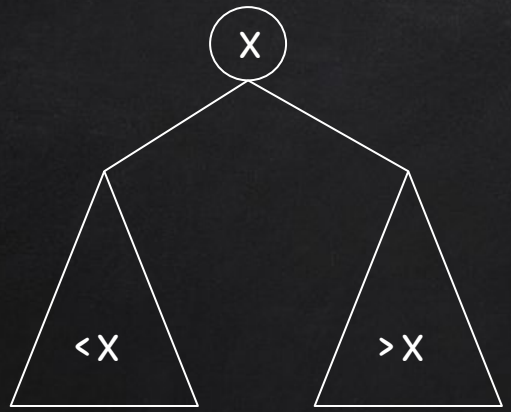
Implementation: Linked List

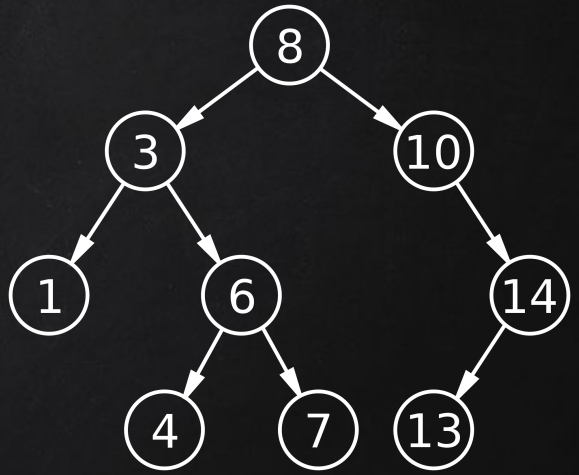

```
class MyLinkedListStack:

    def __init__(self):
        self.topOfStack = ListNode()

    def push(self, data):
        new_node = ListNode(data, self.topOfStack)
        self.topOfStack = new_node

    def top(self):
        data = self.topOfStack.get_data()
        self.topOfStack = self.topOfStack.get_next
        return data
```

Trees

Binary Search Tree (BST)

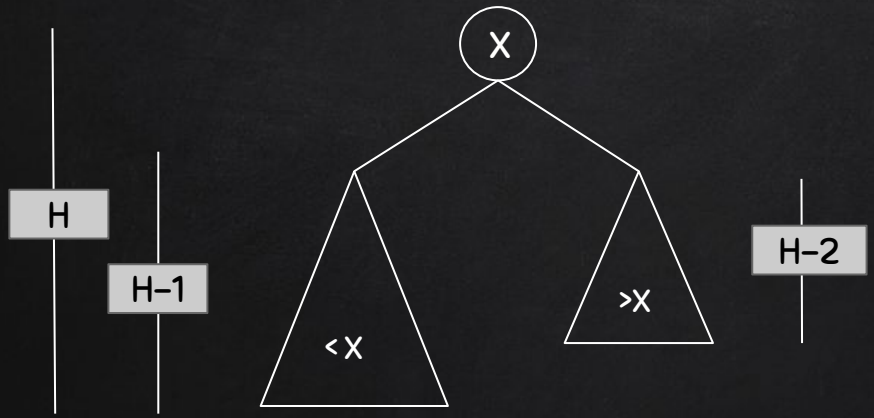
- ✘ Elements have keys (no duplicates allowed)
- ✘ Property:

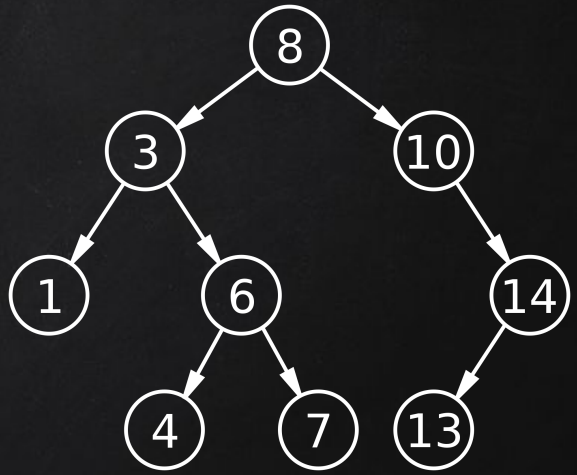




AVL Tree

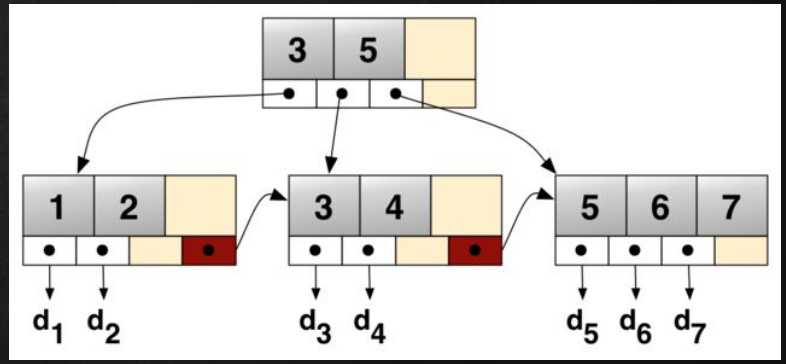
- ✘ Unbalanced BST are bad. How to maintain balance?
- ✘ Property:





B+ Trees

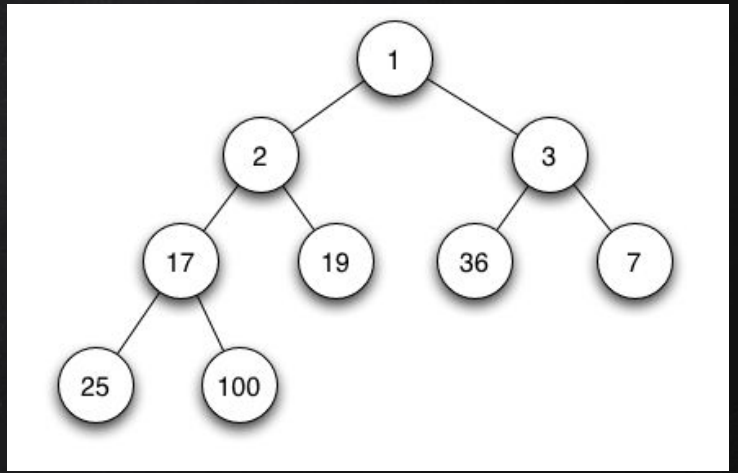
- ✘ B stands for Balanced.
- ✘ All the leaves are the same distance from the root.
- ✘ B Tree of degree m:
  - All non-leaf nodes (except root) have between m/2 (ceil) and m non-empty children.



Binary Heap

- ✘ One technique to implement priority queue.
- ✘ The tree is balanced, so all operations are guaranteed O(log n) worst case.
- ✘ Property:

Parent <= Child
(minheap)



References

- Data Structures and Algorithm course slides, *Faculty of Computer Science, University of Indonesia*, 2014.