UFAZ / Strasbourg University

Object Oriented Programming

Year 1 – Common curriculum

Tutorial / Lab session #2: Classes and instances

Exercise 1 – First class: Point

1. Write a class Point that represents a 2D point in the Euclidean space. Your class has two private attributes x and y (float).
2. Write a constructor so you can instantiate a Point object that initializes x and y.
3. Since you defined the attributes x and y as private, you're going to need setters and getters. Write these methods in your class.
4. Write a method `float distance()` that returns the distance between the point and the frame origin.
5. Write a method `float distance(Point point)` that returns the distance between the current point (`this`) and the point given as argument.
   a. Did you overload or override the method `distance()`?
   b. Explain the difference between overloading and overriding a method.
6. Write a method `void translate(float dx, float dy)` that translate the point by quantity (dx, dy)
7. Write a method `Point barycenter(Point[] points)` that returns the barycenter of a set of point. Should this be a `static` method? Why?
8. Create a class `Segment` that has two attributes p1 and p2 (Point) representing the endpoints of a segment.
   a. Write getters and setters for the attributes

b. Write a constructor that allows you to instantiate a Segment object given two Points

9. Write a method `float length()` that returns the length of a segment. Do you think that a part of this method can be delegated to the class Point? Why?

Exercise 2 – Binary trees

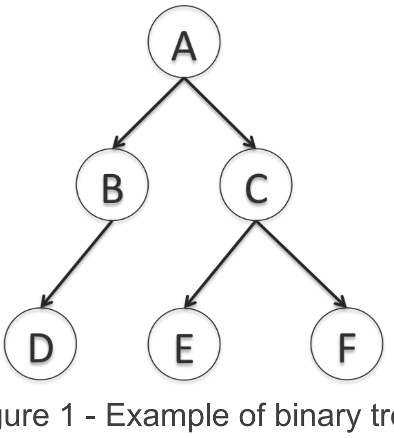

Figure 1 - Example of binary tree

Binary trees are a common data structure. A binary tree is made of nodes. Except for the root of the tree, all nodes have a parent. In other words, each node of the structure has a list of children (that may me empty: in this case, the node is a leaf). In the case of a binary tree, the number of children can be at most two.

The aim of this exercise is not to master (binary) trees because some operations that we can perform on such structures are quite complex, but rather think about how we can implement the basic structure of a binary tree using Java classes.

Questions:

1. Create a class Node that represents a node in a binary tree. The data contained in each node will be a character string representing the label of the node.
    1. What attributes will you create in the class?
    2. How do you intend to implement the notion of children in a binary tree?
    3. How would you make this model more generic (that is, what if you want to be able to implement a tree with any number of children?)
2. Override the method `string toString()` so you can print a node in a human-readable way (print the label of the node).
3. Write a method `void addChild(Node n)` to add a child node to a node. Remember that we are working on binary trees, that is, each node of the tree has 2 children at most!
4. Write a method `void printDescendant(Node n)` that prints a node and all its descendant.

Exercice 3 – Working with stacks

In this exercise, we are going to implement a stack to stores integers. A stack is a container having a fixed size. Elements are accessed in a LIFO manner: Last In First Out, as depicted in Fig. 2. A stack is characterized by:

- A fixed size array containing elements;
- An index, `top`, referencing the top element of the stack;
- The capacity of the stack (i.e. the maximal number of elements the stack can contain).

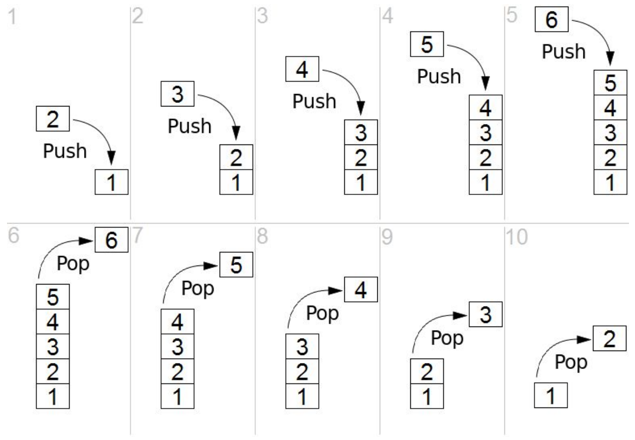

Figure 2 - Stack example

Questions:

1. Create a class `MyStack`. Declare the member variables and add setters and getters for these members.
2. Write a method `void push(int element)` that pushes an element on the top of the stack.
3. Write a method `int peek()` that returns the element at the top of the stack but without removing it.
4. Write a method `int pop()` that removes the element at the top of the stack and that returns it.
5. Write a method `boolean isEmpty()` that returns true if the stack is empty; else it returns false.