National University of San Agustin
System Engineer School

# TO - Lab 03 - Inheritance

Christian E. Portugal-Zambrano

September 17, 2018

## 1 Introduction

Almost always, new software expands on previous developments; the best way to create it is by imitation, refinement and combination. Traditional design methods largely ignored this aspect of system development. In object technology it is an essential concern.
The techniques studied so far are not enough. Classes do provide a good modular decomposition technique and possess many of the qualities expected of reusable components: they are homogeneous, coherent modules; you may clearly separate their interface from their implementation according to the principle of information hiding; genericity gives them some flexibility; and you may specify their semantics precisely thanks to assertions. But more is needed to achieve the full goals of reusability and extendibility.
For *reusability*, any comprehensive approach must face the problem of repetition and variation, analyzed in an earlier chapter. To avoid rewriting the same code over and over again, wasting time, introducing inconsistencies and risking errors, we need techniques to capture the striking commonalities that exist within groups of similar structures — all text editors, all tables, all file handlers — while accounting for the many differences that characterize individual cases.
For *extendibility*, the type system described so far has the advantage of guaranteeing type consistency at compile time, but prohibits combination of elements of diverse forms even in legitimate cases. For example, we cannot yet define an array containing data structured objects of different but compatible types such as List and Stacks.
Progress in either reusability or extendibility demands that we take advantage of the strong conceptual relations that hold between classes: a class may be an extension,

specialization or combination of others. We need support from the method and the language to record and use these relations. Inheritance provides this support.

## 2 Pre-requisites

For this practice you must have the implementation of a Simple Linked List, Double List, Circular List, Stack, Tree, and Hash and make some review of abstract class, interfaces and virtual functions.

## 3 Warm up

We have to understand that inheritance search for extendibility and some benefits of this features is the polymorphism which is defined as the ability to take several forms. In object-oriented development what may take several forms is a variable entity or data structure element, which will have the ability, at run time, to become attached to objects of different types, all controlled by the static declaration. Inheritance hierarchies will give us considerable flexibility for the manipulation of objects, while retaining the safety of static typing. At this section we will try to implement the hierarchical model presented on Figure 3.1 taken from [1], where a hierarchical model of shapes is presented, note here that at each level some data, behavior or functionality is added or eliminated.

## 4 To do

The first task to achieve is to design some other problem similar to the presented on Figurefig:shapes, consider a inheritance of at least 4 levels, this way you will practice your abstraction programming.
Other task to achieve is design the Object oriented model to integrate List, DList, CList, Stack, Queue, Tree and Hash, you will note that some classes share some behavior or structure besides others not, make abstracts classes and interfaces to keep reusability and extension the objective.

## 5 Deep inside

This section is just for anyone who wants to make a deep inside into the theory and like challenges[1], you will have to prepare a presentation of just 5 minutes to explain and run the code, then you will have to defend yourself another five minutes of questions. I want that all students benefit from your presentation, remember that we are here to learn. If you want to do this please email to cportugalz@unsa.edu.pe

[1] This must not be included into the report

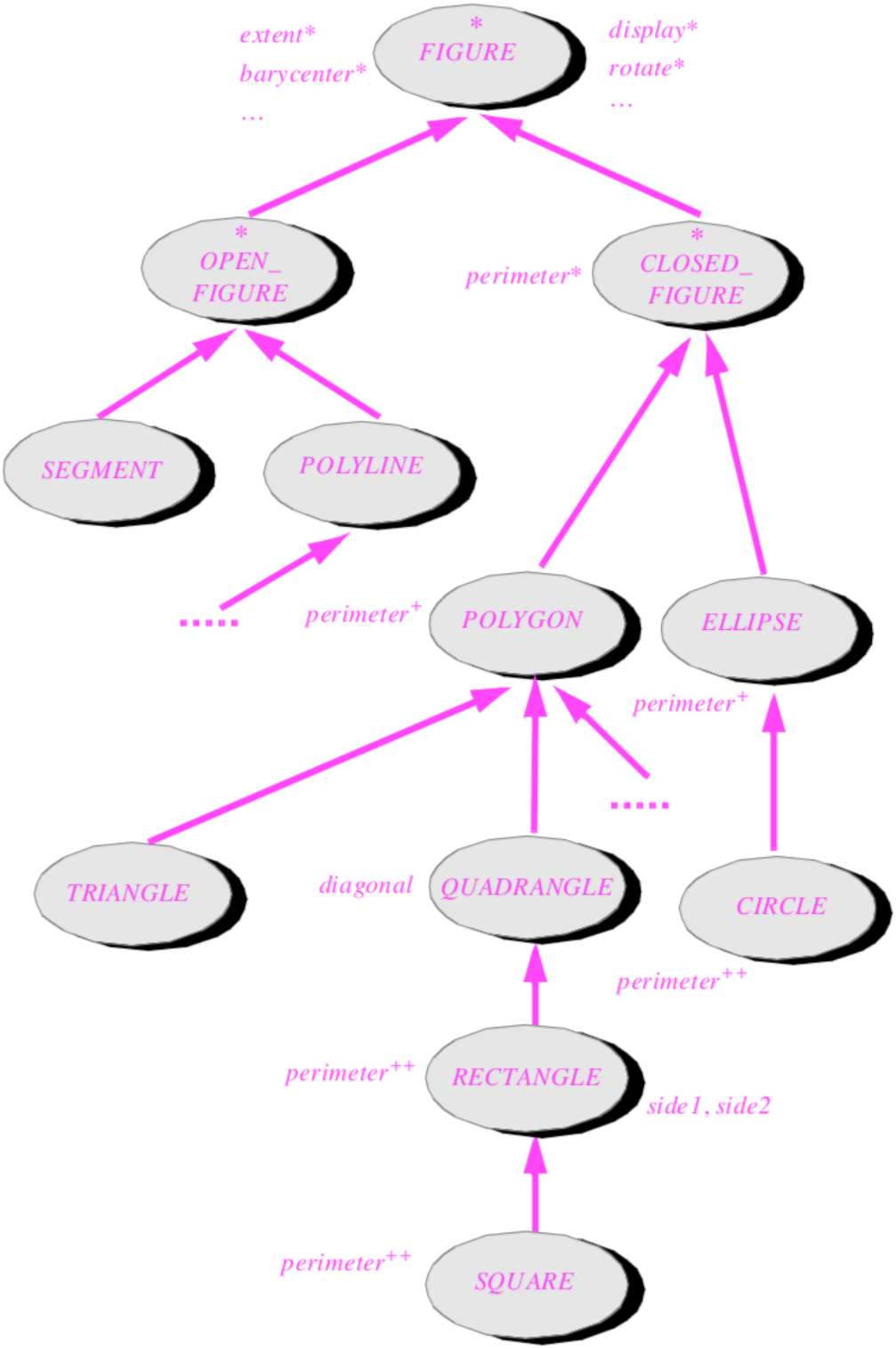


Figure 3.1: Shapes hierarchy to represent abstraction and reusability (Figure taken from [1]).

## 6 Deadline

This report will be qualified in the next class through the presentation of all the code required by this practice. Remember that plagiarism will be punished. All question and doubts must be done to the email.

## References

[1] B. Meyer, *Object-oriented software construction*, vol. 2. Prentice hall New York, 1988.