CS F213 Object Oriented Programming: Class and Lab Notes

Ramprasad S Joshi

August 2023

Abstract

We focus on practice sessions and examples or experiments that consolidate the discussions dyring lectures.

Chapter 1

Introduction and Preliminaries

Object oriented programming is not about classes and methods, or about some condensed jargon like encapsulation and code reusability. Computational models of real-life interaction follow conceptualization of the entities involved in that interactions, the various modes of that interaction, and the relationships, constraints and interdependence amongst the entities that shape up the interaction patterns. This conceptualization is best done in a *functional* object-oriented design.

Our textbook says about itself [Blaha and Rumbaugh, 2004, pp.xv–xvi, Preface]:

> Our emphasis differs from that of some in the object-oriented programming community but is in accord with the information modeling and design methodology communities. We emphasize object-oriented constructs as models of real things, rather than as techniques for programming. We elevate interobject relationships to the same semantic level as classes, rather than hiding them as pointers inside objects. We place somewhat less emphasis on inheritance and methods. We downplay fine details of inheritance mechanisms. We come down strongly in favor of typing, classes, modeling, and advance planning. We also show how to apply object-oriented concepts to state machines.
>
> ...
>
> There are many books on the market that cover object-oriented technology. This book differs from most in that it teaches how to think about object-oriented modeling, rather than just presenting the mechanics of a programming language or modeling notation.
>
> Many of the available object-oriented books are about programming issues, often from the point of view of a single language. Some of these books do discuss design issues, but they are still mainly about programming. Few books focus on object-oriented analysis or design. We show that object-oriented concepts can and should be applied throughout the entire software life cycle.
>
> In addition there are a number of books that present the concepts of the UML. This book is different than most in that it not only explains the concepts, but it also explains their fundamental purpose and shows how to use them to build software. We do not explain every concept and nuance, but we do strive to explain the core of UML–enough to help you learn how to use the UML to build better software.
>
> ...
>
> We have carried forward the first edition's focus on "the three models" because we believe such an emphasis is helpful for teaching and learning OO modeling. However, we dropped the functional model, because it was not as useful as we had expected. In its place, we added the interaction model to incorporate use cases and sequence diagrams and to give a more holistic understanding of behavior among several objects.

This tells us several things:

1. The textbook focuses on OO modeling of real things (including real systems).
2. Languages and syntax are considered secondary, instrumental.
3. Software system level design tools and techniques are emphasized.
4. Modeling interaction among system components explicitly is emphasized.

Note that the functional model that the authors are referring to from the first edition is not about the functional programming paradigm but about modeling real systems by functions of and relationships amongst entities for object-oriented design.

Our Approach We will indeed follow the textbook in the focus and emphasis described above. However, the book itself was written for advanced undergraduates and graduates and professionals. Therefore, we will begin somewhat leisurely, with top-down modular design, and pose real, practical questions for modeling real systems before introducing the tools, techniques and programming language constructs required to model and implement them computationally. These tools, techniques and constructs will appear to be necessary and enabling after we struggle without planning with the usual programming toolchains and habits that we often resort to.

1.1 Some Slang for Ourselves

We will memorise some colloquial terminology that will have temporary meanings in our labs and classes this semester. From time to time, we will align these terms with the more "standard" terms in use in textbooks and tech communities. This section will grow as the semester proceeds.

1.1.1 Separations

We seek the following separations (among many others, of course) with specific purposes:

What	**How**
Function	Implementation
Behaviour	State and Methods
Privacy, Integrity, Security	Interfaces and Protocols
Events and Synchronization	Message Passing and Real-time exceptions and interrupts

1.2 The First Separation Structure: Modularity

We will write a multi-module program for the first lab (Tuesday 22 August 2023).

The Task

1. The top level: Build an application that offers a generic storage type `Dictionary` equipped with a retrieval mechanism for arbitrary amounts of numbers.
2. Desirable design goals: Offer the end user a choice of data invariants. The invariants may be
   - The stored data are always sorted.
   - The data may be sorted on one of the pre-fixed functions on the data. e.g., Sorting them on squares, on absolute values, or on a given polynomial value. Thus, [-2,-1,0,1,2] is sorted on the identity function $f(x) = x$, but on squares $f(x) = x^2$ or absolute values $f(x) = |x|$ they are sorted as [0,-1,1,-2,2]. Note that this is a stable sort. On the polynomial $f(x) = ax^2 + bx + c$, if $b^2 - 4ac = 0$ and $b > 0$ then again the sort will be [0,-1,1,-2,2].
   - In general, then provide a sorting function that takes a key function and the

1.2.1 Skeletal Code

There will be two modules (at least): that means two .c (source) files and one .h (header).

Module 1: The `main` module.

```
// main.c
#include "dict.h"
#include<stdlib.h>
// ... anything needed ...
// ...
int main(int argc, char *argv[]) {
   Dict d1;
   d1=ConsDict(argc-1);
   // -- Make a dictionary of as many elements
   //    as passed on the command line
   int i;
   for (i=1; i<=argc; i++) {
       InsertDict(d1,atoi(argv[i]));
   }
   DisplayDict(d1);
   SortDict(d1);
   DisplayDict(d1);
}
```

Module 2: The `dict` module

```
//dict.c
#include<stdio.h>
#include<malloc.h>
#include<stdlib.h>
#include<error.h>
typedef int *Dict;
Dict ConsDict(int size) {
   int *p=(int*)malloc((size+2)*sizeof(int));
   p[0] = 0; p[1] = size;
   return p;
}
void SortDict(Dict d) {
   BubbleSort(d+2,d[0]);
   // Sort d[0]=actual count elements in it
}
void InsertDict(Dict d, int i) {
   if(d[0]==d[1]) error(-1,0,"Dictionally full!");
   d[d[0]+2]=i;
   d[0]+=1;
}
void DisplayDict(Dict d) {
   int i;
   printf("Displaying dictionary with %d elements, size %d: ",d[0],d[1]);
   for(i=2; i<d[0]+2; i++) {
       printf("Element%02d=%d ",i-1,d[i]);
   }
   printf("\n");
}
```

The header file is not much :

```
// dict.h
typedef void *Dict;
extern Dict ConsDict(int);
extern void SortDict(Dict);
extern DisplayDict(Dict);
extern InsertDict(Dict,int);
```

Copy these code snippets into the suggested files, provide the usual bubblesort in dict.c, and compile by

```
$ gcc main.c dict.c
```

and run (the program, not from the lab) as:

```
$ ./a.out 5 4 2 1 3
```

Then you should be able to see

```
Displaying dictionary with 5 elements, size 5: Element01=5 Element02=4 Element03=2 Element04=1
Element05=3
Displaying dictionary with 5 elements, size 5: Element01=1 Element02=2 Element03=3 Element04=4
Element05=5
```

Provision for the compare functions will be made and explained tomorrow in the lab.

Bibliography

Michael Blaha and James Rumbaugh. *Object-Oriented Modeling and Design with UML*. Prentice Hall/Pearson, 2nd edition, 2004.