Basics of procedural and object-oriented programming

Procedure-oriented programming

- Design some data structure(s) to hold the data
- Write some function(s) to use/change the data
- The main focus is on the functions (procedures)
- Functional decomposition: Thinking about the problem in terms of the functions; the problem is split into a number of chunks (modules) each of which can be solved by a function
- Top-down design

We use the following terms interchangeably:

- function
- procedure
- subprogram
- routine

Software maintenance: Ripple effect of changes in functional decomposition

- Main task

{ Sub-task 1
_ Part 1 of Sub-task 1
_ Part 2 of Sub-task 1
{ Sub-task 2
_ Part 1 of Sub-task 2
_ Part 2 of Sub-task 2
_ Part 3 of Sub-task 2
{ Sub-task 3
_ Part 1 of Sub-task 3
_ Part 2 of Sub-task 3

A change in main ripples through the entire hierarchy of functions.

An example: Bank account
Procedure-oriented approach

Data structure:
*(Do not pay much attention to the syntax now.)

```
struct account
{
int account_id;
char* name;
double balance;
/* other details */
};
```

Functions:

```
deposit(...);
withdraw(...);
show_balance(...);
/* other functions */
```

Object-oriented approach

Data and functions put together:
*(Do not pay much attention to the syntax now)

```
class account
{
int account_id;
char* name;
double balance;
/* other details */
public:
deposit(...);
withdraw(...);
show_balance(...);
/* other functions */
};
```

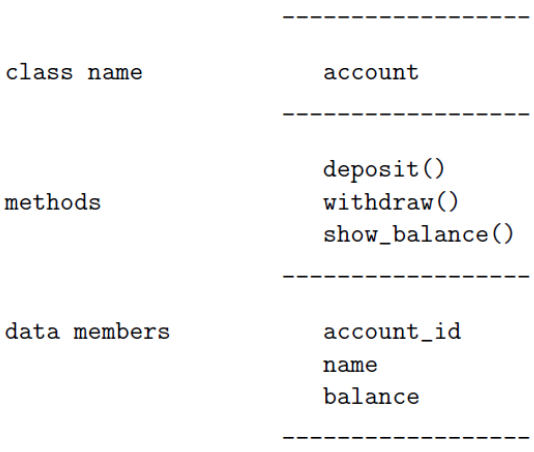
Consider two objects: myaccount and youraccount of the account class
account myaccount, youraccount; /* ignore syntax */

Note: account is a class, and we can create any number of account instances, each one of which is an object of the class account.

Operations:
myaccount.deposit(500);
youraccount.withdraw(100);

Classes and objects

The key concept is that of a class and object. A class captures the common properties of a group of objects. An object is an instance of a class.



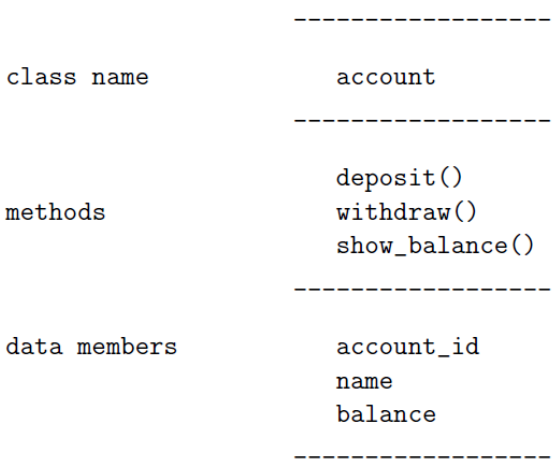
The following terms are used interchangeably:

- member function
- method
- message
- operation

And sometimes the following terms are used interchangeably:

- data member
- variable
- field

_In object-oriented programming, the focus is on the data that is being operated upon and not on the functions that implement these operations.
_ In O-O programming, the data and the operations are not treated as separate entities. The data comes with a set of operations.
_ Object-oriented programming is a method of programming in which programs consist of cooperating objects (objects communicating by message passing) and the classes are probably related via inheritance. (Inheritance will be discussed later.)
_ Functions that want to open a bank account and use it are the clients. The objects (e.g., myaccount) and classes (e.g., account) are the servers. This is the so-called client-server model in OOP.
_ The public operations (i.e, the interface) are important to a client. Clients need not know how the operations are implemented. The private data members can only be accessed from within the member functions. Clients cannot use the private data members directly.
_ We refer to the private data as encapsulated data. This is called data encapsulation. This results in information hiding. (Hiding from whom?)



The object model
_ In procedural programming we talk of calling procedures, e.g., call procedure f1, or simply, call f1. In OOP, we usually say "invoke f1 on some object", e.g., invoke deposit on the object youraccount.

_ In procedure-oriented programming, by looking at a data structure it is in general hard to see what it is supposed to do and how it is supposed to be used. This is

because all information required to use it correctly is buried deep in some collection of functions elsewhere in the program. In OOP, what a client can do with a class is clearly stated in the class itself; the client does not have to look elsewhere.

How is an object created?
_ In the simplest case, an object is instantiated as follows:
account my account;
account my_account, his_account;

What is contained in an object?
_ Each and every instantiated object gets its own copy of the data members. Data members (except static data members, to be discussed later) are not shared among objects of a class.
_ The code for the member functions is not copied. There is only one copy of the implementation code of the member functions in one process (running program or task).

An object is characterized by:

- Identity: This is a unique name.
- State: This is the sum of the properties and values of its data members.
- Behavior: This is how an object responds (reacts) to messages.

Thus the state is the cumulative result of its behavior. In other words, the state is a memory of its past activities and influences future actions.

Who alters the state of an object?
_ Member functions (note that some member functions may cause a change of state of the object, some may not)
_ Friend functions