Module 133.1.1.1 :: OOP Concepts ::The Basic Idea of OOP

The Basic Idea of OOP

Outline

The idea of OOP

- Introduction
- Introduction to OOP Properties
  - Encapsulation
  - Inheritance
  - Polymorphism

Classic procedural languages, such as C, focus on the *procedural* nature of program. That is, the question "what should the program do next" drove the *decomposition* of the software, or, how the software is *decomposed* into parts. This is likely the way that you broke down your first programs as well, even if they were written in an object oriented language such as Java. In C, or procedural java, you structure the program by:

1. Splitting it up into the tasks that must be performed and their subtasks
2. Write functions for each of the tasks.
3. Write code that calls these functions in the proper sequence.

This is a great way to get started! However, as the tasks become more complex and involve more data, it becomes difficult to manage the increasing complexity. In procedural languages we store data in various structures that are defined independently of the functions that make use of them. We then often pass these structures around as function parameters and/or return values.

Consider a simple application that manages course registrations at a university. We have courses, students, instructors, and not just one of each, many of each. We have to manage which students are in which courses that are taught by which instructors. We need functions to add students to a course, assign an instructor to a course, report which courses a student is taking, and more. The list of functions, data structures, and associated variables can become unmanageable!

What many procedural languages lack is a mechanism to manage this complexity. Most make use of the underlying operating systems's file system in some way to define some notion of packaging or modularity. For example, most C compilers support the idea of an include file and separate file compilation such that we can group related operations in a file. This mechanism, however, is limited in what value it provides to the programmer. Yes, it helps to organize our code, but we are still free to call any compiled function that we wish and/or operate on any data structures.

What object oriented languages do is to provide language defined, as opposed to operating system defined, mechanisms that help us group data and code. The language gives us tools to package data and the code that works on that data into objects. It frees our minds to think about how objects interact with each other. OOP has been successful, in part, because it models how we think about things in the real world.

OOP does more than just provide a language based mechanism to package code with data, it also provides features that hide details from the world outside of the object. These mechanisms free us to

focus on the larger picture of interacting objects. As a programmer, we are able to limit the surface area of the things that we have to think about at any one time. We can think of the components of the software more abstractly. This has important cognitive benefits and facilitates the scaling of systems to the large and complex applications that are prevalent today.

Generally OOP languages have the following three properties.

1. Encapsulation
   Encapsulation is the grouping of data and functions together as objects . In Java the **class** provides a blueprint for objects and defines their *interface*.

2. Inheritance
   Allows code to be reused between related types.

3. Polymorphism
   The mechanism by which a value may be one of several types. Behavior at runtime, i.e., which function, *method* in Java, to call is determined by type.

Some authors add abstraction as a property, however, abstraction is not unique to object oriented programming. Abstraction is an important part of programming in general. Functions, for example, are an abstraction of the internal code.

Reading Material

1. Start reading Chapter 2 of **Object-Oriented Design & Patterns**
   1. Sections 2.1 - 2.3

Practice Exercises

There are no practice exercises for this Module. We will return to much of the detail of this chapter in later modules.

Video Lecture

In this brief introduction to this section I talk about the idea of abstraction and how to think about classes and objects. We will return to inheritance and polymorphism in the next module. For now, be sure to watch the excellent alternate video below by **Steve Bagley** .

Alternate Video Lecture

Dr **Steve Bagley** introduces the ideas of OOP in a nice discussion of Pong.