Notes 1

Tuesday, August 25, 2020 1:03 PM

What is Data Abstraction?

- *What is the OO Concept?*
  - It really boils down to the idea that we can model things in software using terminology that we use to describe the same "objects" in the domain world.
- Pillars of OO Programming Paradigm:
  - Inheritance
  - Polymorphism
  - Data Abstraction
  - Encapsulation
- Encapsulation
  - Objects combine data and operations.
- Inheritance
  - Classes inherit properties from other classes.
- Polymorphism
  - Objects determine appropriate operations at execution time.
- Abstraction
  - ?

What is Abstraction???

- Separate purpose of a module from its implementation
- Specifications do not indicate how to implement
- Able to use without knowing implementation

Book Notes: Ch.1

A module is a self-contained unit of code and could be a single, stand-alone function, a class method, a class itself, a group of several functions or classes that work closely together, or other blocks of code

Note: ADTs versus data structures

- An abstract data type is a specification for a group of values and the operations on those values.
- A data structure is an implementation of an ADT within a programming language.

A class combines the attributes —or characteristics—of objects of a single type together with the objects' operations— or behaviors—into a single unit. The individual data items specified in a class are called data members. The operations specified in the class are referred to as methods or member functions . Attributes are typically data, and the behaviors, or methods, often operate on that data. In programming languages such as C++ and Java, classes specify the attributes and operations for the objects.

Encapsulation is a technique that hides inner details. Whereas functions encapsulate behavior, objects encapsulate data as well as behavior.

Note: A guiding principle of OOD is that each class should have a single, well-defined responsibility. The methods of a class should be highly cohesive and related directly to supporting the responsibility of the class. The responsibilities of a class are functionally equivalent to the tasks that the class needs to perform. If a class has too many responsibilities, it should be split into multiple classes, each with a single responsibility taken from the original class.

Note: An operation contract completely specifies a module's purpose, assumptions, input, and output.

Note: The program component that uses a module is the module's **client**. The **user** is a person who uses a program.

Note: A first draft of a module's specifications often overlooks or ignores situations that you really need to consider. You might intentionally make these omissions to simplify this first draft. Once you have written the major portions of the specifications, you can concentrate on the details that make the specifications complete.

Consider now a collection of data and a set of operations on the data. The operations might include ones that add new data to the collection, remove data from the collection, or search for some data. **Data abstraction** focuses on what the operations do with the collection of data, instead of on how you implement them. The other modules of the solution "know" what operations they can perform, but they do not know how the data is stored or how the operations are performed.

Book Notes: C++ Classes:

- Class templates: Allow you to def
  that are independent of the type
  Stored in the data structure
- Header files partially separate the d
  from its implementation.
- A constructor allocates memory for new insta
  initialize the object's data to specified value
  an instance of a class when the object's li
- To provide a public interface for an
  an abstract base class. This allows the
  advantage of polymorphism.
- Virtual Method: tells the compiler tha
  executes is determined at runtime. W
  - Declaring ADT methods as virtual all
    using our class to take advantage of Pol
    methods are invoked.
- A pure virtual method is a virtual
  implementation. An abstract class is one that
- Abstract classes cannot be directly insta

C++ Review

```
class Rectangle
{
        int width,height;
    public:
        Rectangle(int,int);
        int area()          ;
        {
            return width*height;
        }
};
```

```
class Rectangle
{
        int width,height;
    public:
        Rectangle(int x,int y);
        int area();
```

```
#include <iostream>
class Rectangle
{
        int width,height;
    public:
        Rectangle();
        Rectangle(int w,int h)
        {
            width = w;
            height = h;
        }
        void printArea()
        {
            std::cout << width*height;
        }
};
```

every rectangle has w,h → properties

→ default constructor

Object-Oriented Programming

- Ok – so when we first learned how to write a C++ program we put all of our code into a file with an extension of .cpp.
- But before when we created the file we did not create something with the name class or wrap it in { }
  - No less the crazy semi-colon there at the end of the file...
- With Java everything was self-contained in the same file – but we mentioned that in C++ we should separate the definition from the implementation.
- How can this be achieved?
  - Header (.h)/Source (.cpp)
- What do we separate and how do we format our code?
- How do we separate our definitions from our implementation?
- What would we put in our Header file in the previous slides code?
- What would we put in our Implementation file in the previous slides code?

C++ Header File (.h)

```
class Rectangle
{
    private:
        // Private Members
        int width,height;
    public:
        // Default (void) Constructor
        Rectangle();
        // Another constructor…
        Rectangle(int w,int h);
        // Print Area Function
        void printArea();
};
```

C++ Source File (.cpp)

```
#include "Rectangle.h"
// Default Constructor
Rectangle::Rectangle()
{
    // Empty Constructor
}
```

- What do we see here?
  - What can we say about what is going on?
- How is this different than the previous example we saw?

- Clone -> Bring a repository that is hosted somewhere like Github into a folder on your local machine
- add -> Track your files and changes in Git
- commit -> Save your files in Git
- push -> Upload Git commits to a remote repo, like Github
- pull -> Download changes from remote repo to your local machine, the opposite of push

Book Review: Ch.1

- Encapsulation: hide the inner details of functions, methods and objects.
  Functions and methods encapsulate behavior, and objects (instances of a class) encapsulate data as well as behavior.
- Inheritance: allows you to reuse already defined classes by extending their
  slight modifications

ine classes
e of data

esign of a class

nces of a class and can
s. A destructor destroys
fetime ends.

ADT, you can write
client to take full

the code this method
hy use one??
ows an application
ymorphism when the ADT's

method that has no
has at least 1 pure virtual methods.

ntiated

functions and methods ... encapsulate data as well as behavior.

- **Inheritance:** allows you to reuse already defined classes by extending their definitions or making slight modifications.
- **Polymorphism:** objects determine appropriate operations at execution time.
- **Data Abstraction:** technique for controlling the interaction between a program and its data structures. It builds walls around a program's data structures, just as other aspects of modularity build walls around a program's algorithms. These walls make programs easier to design, implement, read and modify.
- **Coupling:** a measure of the dependence among modules. Modules should be loosely coupled. A function or method should be as independent as possible.

- For problems that primarily involve data management, encapsulate data with operations on that data by designing classes. Practice abstraction: focus on *what* a module does instead of *how*.

ADT: collection of data and a set of operations on the data.

C++ Review:

- Class templates allow you to define classes that are independent of the type of data stored in the data structure. When a client is ready to instantiate an object of the class, the client can specify the type of data the object holds.
- Header (.h) files partially separate the design of a class from the implementation.
- To provide a public interface for an ADT (abstract data type), you can write an abstract base class, separating design from implementation and this allows us to take advantage polymorphism when using our class.

dependency lines
Skill lines
rules
Inference lines
macros
comments