Encapsulation: Structures & Objects

Lecture Notes

Overview

- Built-in types (int, double, etc.) are inadequate, we have need of *user defined types*
- Objects or structures to model real world objects/entities
    - Records in a database
    - A person consists of multiple pieces of info, first name, last name, SSN, birth date, etc.
    - Such entities could contain sub-entities: date: month, day, year
- Motivating example: managing pieces of data related to the same Album becomes tedious, lots of extra book keeping when handling collections of such data
- *Encapsulation* is a mechanism by which a programming language may allow 1) the bundling of related pieces of data along with the methods or functions that act on that data; and 2) protect or restrict access to an object's components and 2) group

Structures in C

1. C provides functionality to group several data types into one *structure* or `struct`
2. Syntax
    a. Example:
    ```
    typedef struct {
      int a;
      double b;
    } StructureName;
    ```
    b. May contain any number of simple data types or even other structures
3. Weak encapsulation:
    a. Cannot (easily) encapsulate functions (only data)
    b. Cannot protect data (all members are "public" and available); one convention: "private" members start with an underscore
4. Usage
    a. Declaration
    b. Component selection operator: period
    c. Pointers to structures
        i. Syntax
        ii. malloc: `sizeof(StructureName)`
    d. Indirect Component selection operator: (arrow) –>
    e. Function parameters
    f. Return Types
5. Conventions

a. Builder/factory functions to facilitate creation of structures
b. Header file usage: define structures and functions involving structures in the same header file
c. Avoid passing/returning by value

Objects in Java

1. Introduction
    - Java is an Object-Oriented Programming (OOP) language: a paradigm that is concerned with the interaction of objects
    - Contrast with C: Structural or Procedural paradigm (program's state changes in a linear fashion)
    - An object is an entity that has data and methods that act on that data
    - Java achieves objects by defining classes: blueprints for defining what an object is, how it can be created and how it may be used
2. Defining Classes
    a. Syntax
    b. Member variables (instance variables)
    c. Member methods
3. Creating objects: constructors
    a. Defining constructors
        i. Syntax: method has the same name as the class, no return type
        ii. Default constructor (if no constructors are defined, a default one is provided; if a non-default constructor is provided, default one is unavailable; you can explicitly define it though)
    b. Using constructors: new keyword (invokes a constructor)
4. Visibility keywords
    a. Member fields (data) and methods can be hidden (protected) by changing their *visibility* (to code outside the object)
    b. Contrast with `static` keyword: when a member is static, it belongs to the class itself; requires no instance to be invoked
    `c. public`
        i. world accessible
        ii. Best practice: no data member should ever be public; use getters/setters to control side effects and data validation
        iii. Best practice: only methods that need to be part of the object's interface should be public (helper methods private)
    `d. protected`
        i. Class and subclasses may see (involves inheritance, 156)
    e. (default)
        i. No keyword, aka "Package Protected"
        ii. Class, subclass, and any other class in the same package can view

f. `private`
    i. Only the class may access it

5. Abstraction through an interface (publicly available methods)
    a. Dot operator
        i. Allows you to access instance variables
        ii. Allows you to invoke an instance's method
        iii. No pointers in Java, so no arrow operator
6. Misc
    a. Open Recursion
        i. The outside world has access to an object through a reference variable; we need to allow an object to refer to itself (invoke "my" methods, access "my" variables)
        ii. Keyword: `this`
    b. Mutators & Accessors (getters, setters)
    c. Composition: objects may "own" other objects
    d. Good encapsulation: place the functionality that acts on an object *IN* an object
    e. Object Design
        - Bottom up rather than top-down
        - Objects are your building blocks
        - Object Oriented Programming: interaction of objects rather than a "structural" program (program state is changed by procedures)

Exercises

1. Design a struct/object to model a complex number and several functions to compute arithmetic operations on it (addition, subtraction, multiplication, division).
2. Design a struct/object to model an album. Write several builder functions, a toString function, etc.
3. Design a student struct/object along with some other structs/objects for tests and/or homework assignments. Add functionality to compute a student's grade.