Lecture 15 - The OO Paradigm - Encapsulation and Abstraction

Collin Roberts

November 2, 2023

Outline

Encapsulation

1. We briefly studied modules having high cohesion and loose coupling from §7.2 and §7.3.
2. These are key ingredients in understanding the OO paradigm.
3. We introduce another key ingredient now.

Encapsulation

Definition 1

In OO programming, **encapsulation** *refers to one of two related but distinct notions, and sometimes to the combination thereof:*

1. *A language construct for restricting direct access to some parts of a module.*
2. *A language construct for* **bundling** *data with the methods (or other functions) operating on that data.*

We will adopt Definition # 2.

Remarks

1. Why we adopt Definition # 2: In many OO languages, hiding of components is not automatic or can be overridden; thus, **information hiding** is defined as a separate notion.
2. Encapsulation plus **information hiding** is used to hide the values of a structured data module, preventing unauthorized parties' direct access to them.
3. Publicly accessible methods are provided (so-called **getters** and **setters**) to access the values; other client modules call these methods to retrieve/modify the values within the module.

Remarks

4. Hiding the internals of the module protects its integrity by preventing users from setting the internal data of the module into an invalid / inconsistent state.
5. A benefit of encapsulation is that it can reduce system complexity, and thus increase **reliability**, by allowing the developer to limit the inter-dependencies between S/W components (i.e. this provides a technique for achieving **separation of concerns**).
6. The features of encapsulation are supported by using **classes** in OO programming languages.

Remarks

7. Encapsulation is not unique to OO programming. Implementations of **abstract data types** offer a similar form of encapsulation.
8. See the Example (text pp 199-201) of refining a S/W product from an initial design having low cohesion into a better design having encapsulation.
9. In the first solution to the Cohesion/Coupling example (last lecture), we could have achieved high cohesion and loose coupling by simply copying all the needed code into both modules. But this would indicate a failure to **abstract** effectively (Definition 2). We would have duplicated code in the two modules.
10. **Moral:** Doing OO effectively requires doing a good job on **all** of its ingredients.

Remarks

1. **Abstraction** is a way of simplifying things so that they become easier to understand. E.g. representing the motion of the objects in the solar system by abstracting planets to points.
2. Effective abstraction helps us to see how things which appear different at first glance are actually the same in all relevant ways.
3. In S/W development, abstraction lets us focus on **what** a module does and not on **how** the module does it.

Abstraction

Definition 2

Abstraction *is suppressing irrelevant details and accentuating relevant details.*

Definition 3

A **data abstraction** *is an abstraction done on data.*

Definition 4

A **procedural abstraction** *is an abstraction done on code.*

Remarks

1. **Abstraction** is a means of achieving **stepwise refinement**.
2. As a recommendation to the programmer, the **abstraction principle** reads

   > *Each significant piece of functionality in a program should be implemented* **in just one place** *in the source code. Where similar functions are carried out by distinct pieces of code, combine them into one, abstracting out the varying parts.*

   In short, "Don't repeat yourself."
3. Effective abstraction guides us to good choices of what to encapsulate when we design and develop our S/W.

Remarks

4. Abstraction and encapsulation are different, but go hand-in-hand in OO design and development.
5. Abstraction permits a designer to temporarily ignore the details of the levels **above** and **below** the level currently being worked on, both in terms of data and procedures. An example of a **data abstraction** (Definition 3) is:
    1. A database designer focuses on designing a table, temporarily ignoring the details of
        1. the whole database (the level above), and
        2. the other tables having foreign key relationships to the current table (the level below).

Idea

Design a S/W product to encapsulate the parts that are most likely to change in the future. Doing this effectively will minimize the impact of inevitable changes, on the other components. N.B. There is no algorithm for deciding how to do this. Human intuition and experience are required.

1. Data structures tend not to change very frequently (but **data abstraction** helps if they do).
2. Business rules tend to change more frequently (and **procedural abstraction** helps when they do).