Object-oriented programming

CSC100 / Introduction to programming in C/C++

Marcus Birkenkrahe

May 5, 2024

README

- This script introduces Object-Oriented Programming (OOP) in C++
- PDF version of this file available in GitHub.
- This section, including some sample code, is based on Stroustrup (2014) and Hansen (2013).

The Mythical Man-Month

- "As a project's complexity increases, the number of man-months to complete it goes up exponentially." (Brooks, 1975)
- *Software engineering* struggles with the realities of software development, which is based on programming *paradigms*.
- What's a *paradigm*, especially in science?

  > A paradigm is a pattern or a model, a scientific paradigm is the set of concepts and practices that define a scientific discipline uses and is based on. According to Kuhn (1962), a paradigm shift leads to a scientific revolution when anomalies can no longer be explained using the old paradigm. Examples from physics: behavior of light as particle and wave (1900), Structure of the solar system (1500) (cp. Wikipedia).

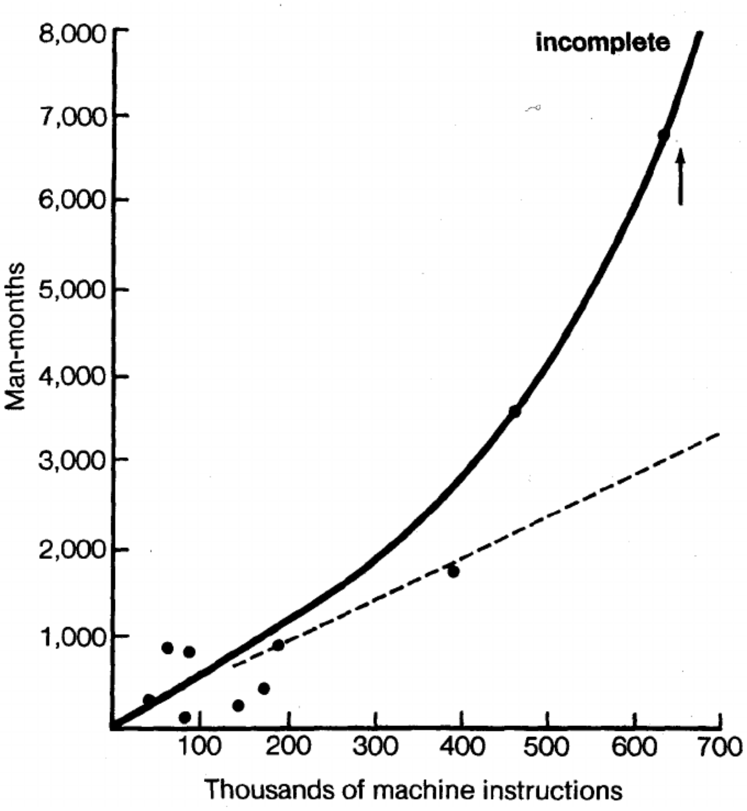


Figure 1: Source: The Mythical Man-Month, Brooks (1975)

Procedural programming

- **Procedural** programming is what you already know:
  - Programs are collection of *functions*
  - Data is *declared* separately
  - Data is passed as *arguments* to functions
  - Fairly easy to learn b/c of **modularization**
- Limitations of procedural programming:
  - Functions need to know the structure of the data
  - Can you think of an example?

    ```
    int add (int x, int y)
    {
      return x + y; // this only works for int data
    }
    printf("%d\n", add(2,2));     // works well
    printf("%g\n", add(2.0,2.0)); // returns machine 0
    ```

    ```
    4
    0
    ```

    - Large programs become difficult to understand/maintain/debug
    - Large programs are hard to maintain/extend/reuse
  - When an approach generates too many **anomalies**, a totally new approach, or a new **paradigm** often emerges - paradigms turn people's worldviews upside down.
  - Can you think of *new paradigms* in science, history, etc.?
    - Darwin's model of evolution based on genetic mutations
    - Idea of climate change as man-made phenomenon linked to CO2
    - Focus on germs as the origin of disease
    - Cosmological model of the universe (and heliocentric model)
    - Relativity (special: of mass/energy, general: mass/spacetime)

    * Quantum mechanical model of the world at smallest distances
      Note: none of these are true in the "biblical" sense but they are scientifically true, that is they describe some of the world as an approximation, through abstraction, and are in continuous development.

- Object orientation (Software Engineering concept)
- The greatest conceptual and practical difference between C and C++ is the explicit use of *object orientation* (OO).
- OO can extend to general design, analysis, testing, even management - whenever you focus not on the procedure but on the *objects* involved and their ability to exchange *messages*.
- *Classes* model real-world domain entities (modeling), e.g.
  * for a school application: `student`, `professor`, `course`, etc.
  * for a photo application: `slideshow`, `location`, `photo` etc.
- Higher level of *abstraction* during development (less detail)
  * When coding a `student` class, think about what a student, as an instance of the class, might do (*method*) or have (*attribute*)
  * You need to concern yourself with interactions and relationships between the different objects of your world
- What are examples for *methods* (= abilities) of a `student` class?

  E.g.
    * `student.enrol()`
    * `student.attend()`
    * `student.graduate()`
    * `student.dropClass()`
    * etc.
- What are examples for *attributes* (= properties) of a `student` class?

  E.g.
    * `student.name`
    * `student.level`
    * `student.grade`
    * `student.gender`
    * `student.enrolled`

    - etc.
  - To compute things, e.g. find out if a student is registered this term, I can send a message to an *instance* of the `student` class, e.g. the student `Frank`, and ask him if he's registered this term:

    ```
    Student Frank;   // Frank is a student
    cout << Frank.enrolled();  // is Frank enrolled?
    ```

  - This is very different from procedural programming where I would have to pass the student to that function:

    ```
    int enrolled(student) {...} // function definition - returns enrolment status
    int status = enrolled("Frank"); // check Frank's status
    ```

- The function depends on the business logic, as does the method of the Student class, but it is defined on *one* place - one change is enough.
- Objects contain data + their operations (= *encapsulation*)
- All of this is a little like developing your own video game (C++ based engines dominate video game and graphics development)[1]
- OOP is used successfully in very large program applications

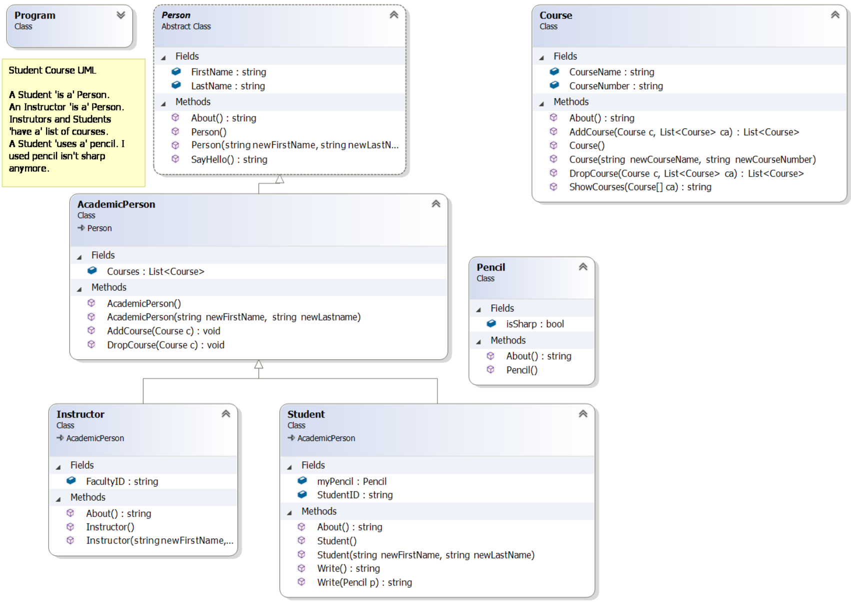
OOP concepts (overview)

- Information-hiding via *encapsulation* (e.g. `student.enrolled()` hides specific implementation from users)
- *Inheritance* = creation of new classes (e.g. `InternationalStudent` as a class derived from `Student`.)
- *Polymorphism* = add new logic to a derived class without touching the original class (e.g. for `IntStudent.applyVisa()`).

Here is an example of how this looks like in UML (a modeling language, like BPMN):

(Link: `http://imamp.colum.edu/mediawiki/images/e/eb/ClassDiagramStudentCourses.png`)

[1] This is also why I got started in C++ rather than in C: for my PhD, I had to develop a large library of graphical objects (which in turn represented particle physics entities), and C++, which had only been developed a few years earlier, was just the right tool for that. Neither Java (1995) nor Python (2000) existed at the time!


Student Course UML

A Student 'is a' Person.
An Instructor 'is a' Person.
Instrutors and Students
'have a' list of courses.
A Student 'uses a' pencil. I
used pencil isn't sharp
anymore.

Figure 2: Class diagram (UML, source: Columbia U).

Limitations

- OO Programming does not make bad code better
- Not everything decomposes into a class
- Steeper learning curve (especially for C++)
- Upfront investment because of design requirements
- Programs are larger, slower, more complex

Further study

- The "Rook's Guide to C++" (Hansen, 2013) which is freely (and legally - Creative Commons license) available as a PDF online covers the basics of C++ in 130 pages.
- Much more thorough is the book by Stroustrup (2014). It's expensive (though copies are floating around, and I got one copy for the library). It contains 1200 pages of C++ goodness.

- For a quick, high ROI overview of C++ in 40 min only, check out Mike Dane's "C++ Programming | In One Video" (2017). Annoying: ads. Talk about OOP begins about 30 min into the course. You may infer that about 1/3 of C++ is not C, which is about right.

- FreeCodeCamp offers a free C++ course on YouTube (2022), which leads to advanced topics - and takes 31 hours to watch. Uses VS Code editor with GCC and explains how to set it up.

- Udemy offers this 46-hour video-based course (2022) which is very nicely presented, contains exercises, but costs a little money (I got it for $10).

- See also "How to teach yourself programming in 10 years", or "Why is everyone in such a rush?" by Peter Norvig (director of research at Google and author of the standard textbook on AI, 2021).

- History and context: listen to the 2 hour podcast/interview with creator of C++ - Bjarne Stroustrup: C++ | Lex Fridman Podcast #48 (2020), which contains a wide range of C++ and programming related issues. (Lex Fridman is an AI/ML professor at MIT.)

References

- Brooks (1975). The Mythical Man-Month, Addison-Wesley. URL: fermatslibrary.com (extract)

- Hansen (2013). The Rook's Guide to C++. URL: rooksguide.org.

- Kernighan/Ritchie (1978). The C Programming Language (1st). Prentice Hall.

- Orgmode.org (n.d.). 16 Working with Source Code [website]. URL: orgmode.org

- Stroustrup (2014). Programming – Principles and Practice Using C++. Addison Wesley. URL: stroustrup.com.