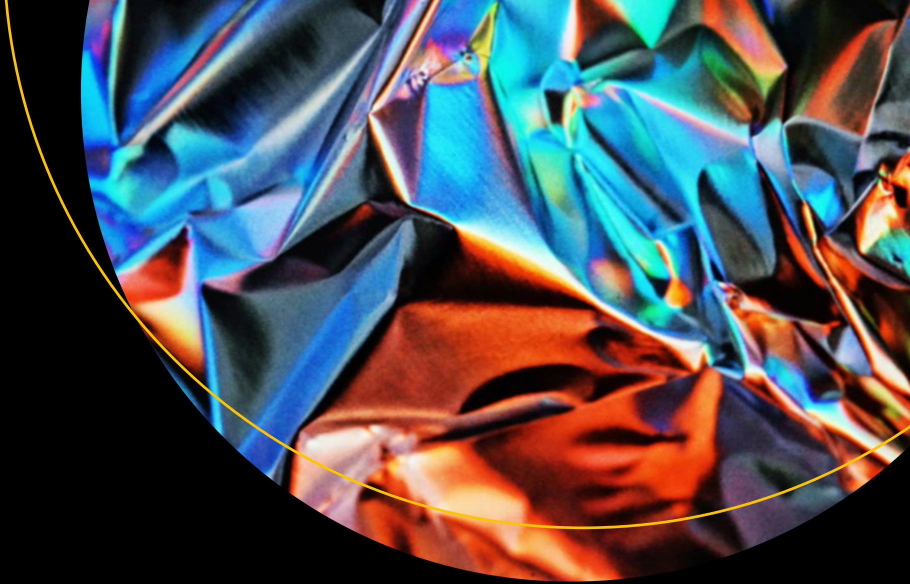

Hands-On Object-Oriented Programming

Mastering OOP Features for Real-World Software Systems Development

Anil Kumar Rangisetti

Apress®

Hands-On Object-Oriented Programming

Mastering OOP Features for Real-World Software Systems Development

Anil Kumar Rangisetti

Apress®

Hands-On Object-Oriented Programming: Mastering OOP Features for Real-World Software Systems Development

Anil Kumar Rangisetti
Kurnool, Andhra Pradesh, India

ISBN-13 (pbk): 979-8-8688-0523-3 ISBN-13 (electronic): 979-8-8688-0524-0
https://doi.org/10.1007/979-8-8688-0524-0

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Melissa Duffy
Development Editor: Laura Berendson
Coordinating Editor: Gryffin Winkler
Copyeditor: Kim Burton

Cover designed by eStudioCalamar

Cover image by Vinicius “amnx” Amano on Unsplash (unsplash.com)

Distributed to the book trade worldwide by Apress Media, LLC, 1 New York Plaza, New York, NY 10004, U.S.A. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail booktranslations@springernature.com; for reprint, paperback, or audio rights, please e-mail bookpermissions@springernature.com.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub (https://github.com/Apress). For more detailed information, please visit https://www.apress.com/gp/services/source-code.

If disposing of this product, please recycle the paper

To my teachers, Dr. Bheemarjuna Reddy and Shri Badrinadh Garu, for identifying my strengths, giving me wonderful opportunities to work with them, and guiding me to achieve my goals.

To my lovely wife, Sravani, for being a wonderful partner and supporting me in all situations. Without her love and support, I could not accomplish it.

Table of Contents

About the Author

Dr. Anil Kumar Rangisetti received his PhD in computer science and engineering from the Indian Institute of Technology (IIT) Hyderabad. He has nearly 10 years of teaching and research experience in computer science and engineering. During his career, he worked at prestigious Indian institutions such as IIIT Dharwad, SRM-AP, and GMR, and at software development and research labs such as Aricent. Currently, he is an assistant professor in the CSE Department at IIITDM, Kurnool. He trains students in OOP languages and how to use advanced simulators (NS-3), Docker, and networking tools for developing applications, and he has guided many undergraduate and postgraduate students in their projects.

Broadly, Dr. Rangisetti's research interests include Wi-Fi technologies, next-generation mobile networks, software-defined networking (SDN), network functions virtualization (NFV), and cloud computing. He also writes and reviews books on computer science technologies and programming languages, and he is the author of *Advanced Network Simulations Simplified* (Packt, 2023).

About the Technical Reviewer

Saravan Nanduri is a seasoned senior full-stack web developer with nearly two decades of experience in the information technology sector, specializing in developing object-oriented applications. Having worked with prestigious companies such as Tech Mahindra and Accenture, Saravan brings expertise to every project he undertakes.

After graduating with computer science and engineering degrees in 2005, Saravan embarked on a remarkable journey leading him to the United States in 2015, where he worked for government agencies before assuming the role of senior software engineer at SS&C Innovest in 2019.

With a solid foundation in computer science, Saravan is adept at architecting and implementing both client and web-based enterprise applications. His proficiency spans a wide spectrum of technologies, including C++ and Microsoft .NET frameworks, such as C# and MVC.

Beyond his professional endeavors, Saravan values relationships and camaraderie.

Acknowledgments

First, I would like to thank Apress for accepting my book idea and giving me this wonderful opportunity. I would especially like to thank Melissa Duffy for keenly going through the book proposal and suggesting to me how to make the book proposal interesting and perfect. Melissa's support and encouragement throughout the book-writing process is highly helpful. Melissa's simple suggestions improved the quality of the book's content.

I want to thank Nirmal Selvaraj for his continuous support in the entire review and the book contents finalization process. His timely help and support helped me to finish on time.

I thank the book's technical reviewer, Sarvan Nanduri, for his valuable time and suggestions in all hands-on activities and technical concepts. His keen observations helped me correct all kinds of errors and incorporate necessary topics to improve the book's quality tremendously.

I would like to thank every member of the Apress for supporting me in writing this book. I would love to work with the Apress team again.

I give my heartfelt thanks to all my students for their interest in attending my lectures and working with me. All my students' curiosity, comments, and suggestions helped me to write this book.

Finally, I thank all my family members, friends, and colleagues for their love and support.

Introduction

Object-oriented programming (OOP) is an essential skill for implementing extendible, reusable, and easy-to-use software systems. To develop any application software or system software, learning OOP concepts and programming is necessary. OOP basic principles help in easily handling a wide variety of real-world software systems (games, application software, novel systems) implementations. This book blends OOP concepts and programming activities for active learning. All hands-on activities and real-time scenarios are described with step-by-step procedures in terms of designing, programming, and evaluations.

You will learn OOP features through real-world examples and practice through C++ programming hands-on activities. You will also learn advanced design and development skills, such as design patterns and event-driven programming for handling novel systems design and development. Finally, you are briefly introduced to OOP features practice through other important OOP languages: Python and Solidity.

This book is organized into three parts. In Part 1 (Chapters 1–4), you learn and practice OOP concepts using C++ for solving real-world software development problems.

Part 2 (Chapters 5–7) explains how to model real-world problems into reusable, extendible, and easy-to-use software development blocks using OOP concepts such as inheritance, object associations, and polymorphism.

In Part 3 (Chapters 8–10), you learn how to use design patterns and event-driven programming for handling complex software system object creation, behavior, and interactions. Finally, you are introduced to OOP using Python and Solidity.

By the end of this book, you will have learned how to design and implement a variety of real-world software systems from scratch using OOP principles, design patterns, and event-driven programming skills.

CHAPTER 1

The Importance of Object-Oriented Programming

Object-oriented programming (OOP) is essential for handling challenges in developing flexible, extendible, reusable, and easy-to-use software systems. OOP approaches simplify the complexity of modeling real-world application concepts into software building blocks.

OOP offers powerful programming constructs and principles to deal with the complexity of software development. OOP constructs such as classes help you to systematically map real-world entities, and it helps in hiding the implementation details of the entities, controlling their data access, and simplifying the software system interactions, activities, and tasks. Moreover, OOP principles such as inheritance and polymorphism help you to develop reusable and easy-to-use software systems.

Learning OOP helps you deal with the complexity of any software, such as e-commerce applications, system software (e.g., device drivers, compilers, operating systems, databases), next-generation applications such as IoT, industrial IoT (IIoT), smart applications, and many more. To appreciate the importance of learning OOP, this chapter discusses the following topics.

A. K. Rangisetti, *Hands-On Object-Oriented Programming*,
https://doi.org/10.1007/979-8-8688-0524-0_1

- Algorithms vs. software
- Software development challenges
- Introduction to OOP concepts
- How OOP approaches simplify the software complexity
- Systematically modeling real-world entities into software

Algorithms vs. Software

Before exploring software, you should know how to start writing a program for solving well-defined problems, such as mathematical, computational, searching, and sorting problems. Solving these problems through a program involves considering all necessary inputs and defining a logical sequence of computational steps to get the desired results. Formally, it is known as writing an algorithm.

This section briefly introduces the following topics.

- Algorithm characteristics
- Writing an algorithm
- Software characteristics
- Software development challenges

Algorithm Characteristics

An algorithm defines a logical sequence of instructions or commands to solve a problem. For instance, algorithms are highly suitable for implementing programs to solve specific problems such as searching, sorting, data structures accessing problems, computational problems, and many mathematical problems. Algorithms can be easily converted into computer programs using basic programming constructs such as data structures, conditional statements, loops, and functions.

- Simple modeling approaches such as flow charts are helpful to write algorithms.
- Algorithms' logical sequence of steps can be converted into programs using procedural program languages such as C.
- An algorithm's success mainly depends on its performance. Algorithm performance is usually defined in terms of space and time complexity.
- Developing efficient algorithms is all about reducing space and time complexity. For example, many sorting algorithms have evolved to reduce time complexity from bubble sort (O(n^2)) to quick sort (O(logn)). Here, the time complexity is represented in Big O notation to represent the upper bounds of algorithms.
- Algorithms can be developed into programs with smaller teams or individuals.
- Procedural-oriented programming languages (e.g., C) are sufficient to convert algorithms into programs.

Next, let's look at how to write an algorithm and convert it into a program using procedural programming language constructs.

Write an Algorithm

Let's solve a problem related to searching for an element from any given list of elements.

- Input: List of elements (list [0 to n]) and a searching element (key)
- Output: Element found (True), Element not found (False)

1. Index=0
2. Traverse through the list of elements until the list ends.

   Check the following conditions:

   In case key presents in the list:

   return True

   otherwise

   Go to 2:
3. If list ends:

   return False

Now it can be easily converted into any procedural-oriented program constructs such as `if-else, for loop`, and functions ().

For example, let's write a C function to solve the search problem.

```
int search(int list [], int n, int key)
{
  int i=0;
  for (i=0;i<n;i++)
  {
    if (list[i] == key)
      return 1;
  }

  return 0;
}
```

You have seen how easy it is to convert a well-defined algorithm into a program using procedural language programming constructs. Next, let's quickly explore software and its characteristics.

Software

Software is evolved to solve a variety of real-world complex problems, which range from system software (editors, compilers, operating systems, databases, protocol stacks, etc.) to application software (e-commerce, online reservations, entertainment software, gaming applications, etc.) and current trending smart applications such as drone applications, IoT, and smart cities.

Unlike well-defined problem-solving using algorithm approaches, software development must follow suitable systematic software engineering procedures and models (e.g., waterfall model, iterative, spiral, and DevOps) to ensure the following features.

- Verifying and validating all requirements of stakeholders
- Reliable in terms of fault tolerance and zero downtime
- Scalable software components to meet the dynamic demands of users
- Flexible software components in terms of making necessary changes or introducing new features
- Extendible software components for producing new versions of the software to meet market needs or introducing innovative features

Besides these features, software success depends on the following.

- How quickly it can be developed and tested
- An easy-to-use interface

- How quickly modifications can be made
- Multiple teams able to work on components in parallel
- Reusable and easily extendible software components

Software Development Challenges

By following suitable software engineering principles and models, it is possible to get all requirements from users involved in using the software. However, translating user requirements into software design blocks is not straightforward. For example, in e-commerce applications, a few basic requirements are that software users should interact with the system easily to browse items, select items into their basket, and place an order.

These requirements cannot be easily translated into software by following algorithm design principles and procedural-oriented programming constructs. Unlike algorithms, software development involves a lot of ambiguity to be dealt with. It is very challenging to completely map all real-world entities, their transactions, and all requirements into software.

You face the following challenges when you want to develop software using algorithm and procedural programming approaches.

- It is highly challenging to model all real-world entities, requirements, and constraints in a limited number of phases.
- It is highly difficult to deal with initial ambiguity (getting ready with initial designs and models) and define logical steps.
- Starting points are not evident in implementing the system components.

- It is difficult to connect software components for integrating the complete system.
- It is difficult to develop scalable, flexible, and extendible software components.
- It is unrealistic development and release deadlines.
- It is unpredictable software success.

Next, you are introduced to OOP concepts and how OOP features are helpful for software development.

Introduction to OOP Concepts

OOP offers excellent features to simplify software development by converting high-level requirements and design processes into software implementation.

- Class
- Data abstraction
- Encapsulation
- Data hiding
- Inheritance
- Polymorphism

Let's go over OOP basic programming constructs called *classes*.

Class

A class is the most important programming construct of the OOP. It helps you easily model any real-world entity (a customer, a drone, or any transactions) into a software block. OOP basic construct called class is defined with its related data (data members or fields) and member functions for accessing its data members. This book uses "data members" and "fields" synonymously. The class structure is shown in Figure 1-1.

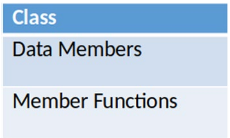

Figure 1-1. *Class structure in OOP*

For instance, customer entities related to an online shopping context can be easily modeled, as shown in Figure 1-2.

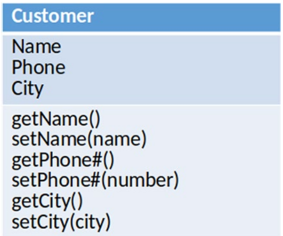

Figure 1-2. *Online shopping application example class: Customer*

Let's inspect the Customer class definition carefully. The data members section includes the customer's name, phone number, and address.

Under the member functions section, you define corresponding access functions for each data member, such as get and set functions. Usually, the "get" member functions are defined to retrieve data members' values, and set member functions are defined to update the values of data members. For example, the City field of the `Customer` class, `getCity()`, is useful for retrieving a customer entity's city, and `setCity(city)` is useful for setting or updating a customer entity's city.

Having the necessary set and get member functions defined in the class, you can later easily include complex online shopping application tasks. For instance, in online shopping applications, customer phone verification and update tasks can be easily done using `getPhone#()` and `setPhone#()` member functions. Similarly, other member functions are useful for accessing the respective data of the Customer class.

Next, let's look at another example in a gaming application context: modeling a duck character into software as a class (see Figure 1-3).

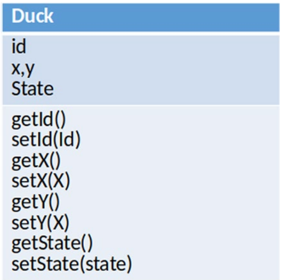

Figure 1-3. *Gaming application example class: Duck*

The `Duck` class includes a duck identifier (id), its location (`x,y`), and its state (dead or alive). For accessing these data members corresponding set and get member functions are defined inside the class.

Now, checking whether a duck is live or dead can be easily done by accessing the duck state using its `getState()` member function. Similarly, you can easily track duck position (`x,y`) using get and set location functions.

Another interesting example of class structure is IoT sensor modeling, as shown in Figure 1-4.

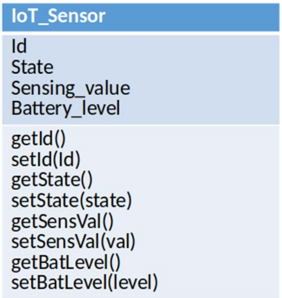

Figure 1-4. *Smart application example class: IoT_Sensor*

The `IoT_sensor` class includes data members related to the sensor identifier (`Id`), its `State` (sensing, sleeping, and dead), `Sensing_value`, and `Battery_level`. Under member functions, sections corresponding to set and get functions are defined for accessing the data members.

Suppose you want to keep a particular sensor in a sleep state in your IoT application. It can be easily done by accessing the sensor state using `setState(state)` member function. Similarly, you can access a sensor's battery status using `getBatLevel()` and `setBatLevel(level)` functions.

Besides simplifying modeling real-world entities, classes are powerful programming constructs whose definition captures the following important OOP principles.

Data Encapsulation

If you are an experienced C programmer, you can easily understand `structure` data type helps you to combine related data elements under a single structure variable. However, you cannot control its data and their related accessing functions together into a structure.

The following is an example.

```
struct customer
{
  char  name[30];
```

```
  int phone;
  char address[30];
};
```

Any function can use `struct customer` variables to change internal data of the customer variable as follows.

```
void function1 (struct customer c1)
{
/* It can access customer data */
}
void function2 (struct customer c1)
{
/* It can access customer data */
}
```

Passing a c1 variable to any C functions, then those functions can change the corresponding `struct customer variable's` data members. It means you are not able to combine data and their accessing functions. It can lead to no control over the sensitive data of real-world entities.

Interestingly, OOP classes allow you to combine related data and its member functions into a `Class` definition. It is known as data encapsulation. Then, you can model a specific real-world entity from the `class` by creating an object and interacting with the object through `class` member functions.

You can observe their data and respective accessing functions from the example classes—Customer, Duck, and IoT_Sensor. As discussed, tasks related to the corresponding entities can only be done through their class member functions. For example, the `IoT_Sensor` entity's `Sensing_value` access can be changed through its object and class member functions: `setSenseValue()` and `getSenseValue()`.

Data Abstraction

Having encapsulated data types support such as classes in OOP, accessing variables of the complex data types also gets simplified. In your program, you define objects (variables) for the respective `Class` (complex data type) and invoke necessary member functions from the objects to access their details. For example, to set an IoT sensor state to “sleep,” you can easily do it with the following lines of code.

```
IoT_Sensor i1;
i1.setState(2);   // Example, 0: Dead, 1: Sensing, 2: Sleeping.
```

Similarly, you can check whether the duck is alive with the following lines of code.

```
Duck d1;
int state = d1.getState(); //1: Alive 2: Dead
if (state == 2)
  cout<<d1.getId()<<"is dead";
```

To access the `IoT_sensor` or `Duck` details, focus on their objects and accessing functions, not their implementation details. You need not know its internal details to access a complex data type.

By checking these examples, you can understand that OOP classes greatly simplify accessing complex entities’ data using its related member functions defined inside the class.

Data Hiding

You have observed how to combine `class` data and its member functions to simplify accessing its objects. Besides these features, the OOP class offers a powerful way to control access to an object’s data.

It means controlling objects data members access from the outside of a class. It can be achieved by attaching access control modes (access specifiers) with data and member functions of a class. OOP languages generally offer three access specifiers: public, private, and protected access.

- **Public access**: Data and member functions defined under the public section can be accessed by any function through the respective class objects.
- **Private access**: Data and member functions defined under the private section are allowed to be accessed by only member functions of the class.
- **Protected access**: Data and member functions defined under the protected section are allowed to be accessed by only member functions of the class and its inherited classes.

You have just seen how to limit an object's data access using the OOP access specifiers. Later chapters discuss an object's data access control in detail. Now that you have explored the OOP basic construct class, let's discuss instances and variables of the `class` data type.

Objects

Objects are powerful ways to create software components and implement tasks, transactions, activities, operations, and functions. For example, it is easier to model the real-world entities such as customers and their transactions or related activities as objects to develop an online shopping application.

Object is an instance of a class, it contains data members and member functions. Hence, any interactions related to the object are done through the class member functions.

In OOP, for example, having a `class` defined for customers simplifies online shopping customer entities as `Customer` objects. Then, all the following tasks implementaion gets simplified: registering a customer, updating customer details, and checking customer details by creating and interacting with customer objects.

Moreover, an object's powerful combination with its data and accessing functions helps you easily realize several identical software components.

The following are examples.

- A `Drone` class that creates multiple drones is nothing but defining multiple drone objects.
- A `Robot` class that creates multiple robots is nothing but defining multiple robot objects.

Similarly, think of real-world applications entities modelling such as e-commerce, gaming, and system software.

Objects Details

To understand an object, you can view it as a variable of a particular data type. Similarly, an object is a variable of the `class` data type. In OOP, objects are instances of classes. During program execution, objects are created by allocating necessary memory space for their data member's access.

For example, to create a variable of `int`.

```
int a; // int is data type and a is variable
```

Similarly, in C++, you can create objects from the `Customer` class as follows.

```
Customer c1, c2;
```