

VIETNAM NATIONAL UNIVERSITY - HO CHI MINH CITY  
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY  
**FACULTY OF COMPUTER SCIENCE AND ENGINEERING**



---

**ADVANCED PROGRAMMING - CO2039**

---

**SUBMIT ASSIGNMENT FOR SESSION 1**

---

**In-class Exercises**

---

Advisor(s): Tran Truong Tuan Phat

Student(s): Nguyễn Đăng Gia Đạo

HO CHI MINH CITY, JANUARY 2026



## Contents

<b>1 Exercise 1 - Multiple constructor</b>	<b>3</b>
<b>2 Exercise 2 - Constructor Chaining</b>	<b>4</b>
<b>3 Exercise 3 (Bonus) - Library System</b>	<b>4</b>



## 1 Exercise 1 - Multiple constructor

### Topic

Create a `Car` class with the following private attributes:

- `brand` (string)
- `model` (string)
- `year` (int)

Add the following:

- A default constructor that sets default values.
- A parameterized constructor to initialize all attributes.
- Setters and getters for all attributes.

Write a main function to:

- Create an object using the default constructor and display its details.
- Create another object using the parameterized constructor and display its details.

### Solution

The complete source code implementation for this exercise, including versions in Java, C++, and Python, is hosted on my GitHub repository:

**GitHub Repository:** [Click here to view source code](#)

(Or visit: [https://github.com/daonguyenhp/C02039\\_In\\_class\\_Ex\\_1](https://github.com/daonguyenhp/C02039_In_class_Ex_1))

**Approach:** The solution demonstrates **Encapsulation** by defining a `Car` class with the following key components:

- **Data Hiding:** Attributes (`brand`, `model`, `year`) are declared `private` to prevent unauthorized external access.
- **Constructor Overloading:** Both *Default* and *Parameterized* constructors are implemented to allow flexible object initialization.
- **Access Control:** Public `Getter` and `Setter` methods are provided to safely read and modify the private data.



## 2 Exercise 2 - Constructor Chaining

### Topic

Create a `Laptop` class with private attributes: `brand`, `model`, and `price`. Implement the following to demonstrate **Constructor Chaining**:

- A default constructor.
- A parameterized constructor to initialize all attributes.
- Use constructor chaining to call one constructor from another.

Write a main function to test the class.

### Solution

The complete source code implementation for this exercise is hosted on my GitHub repository:

**GitHub Repository:** [Click here to view source code](#)  
(Or visit: [https://github.com/daonguyenhp/C02039\\_In\\_class\\_Ex\\_1](https://github.com/daonguyenhp/C02039_In_class_Ex_1))

**Approach:** The solution focuses on code reusability and the **DRY (Don't Repeat Yourself)** principle through Constructor Chaining:

- **Constructor Chaining:** The implementation minimizes code duplication by having simpler constructors call more complex ones (using `this()` in Java or delegation in C++).
- **Overloading:** Multiple constructors are defined with different signatures, providing flexibility in object instantiation.
- **Encapsulation:** Standard getters, setters, and private attributes are maintained to ensure data security.

## 3 Exercise 3 (Bonus) - Library System

### Topic

Create a Library system using encapsulation and constructors.



- **Book Class:** Attributes include `title`, `author`, `ISBN`, `price`, and `stock`. Include a method to reduce stock when borrowed.
- **Library Class:** Create a Library class with: An array of Book objects.

Implement methods to:

1. Add new books to the library.
2. Search for books by title or author.
3. Borrow books (reduce stock).

Write a main function to simulate library operations.

## Solution

The complete source code implementation for this exercise is hosted on my GitHub repository:

**GitHub Repository:** [Click here to view source code](#)  
(Or visit: [https://github.com/daonguyenhp/C02039\\_In\\_class\\_Ex\\_1](https://github.com/daonguyenhp/C02039_In_class_Ex_1))

**Approach:** The solution simulates a real-world system by combining **Object Composition** and logical processing:

- **Object Composition:** The Library class "has" a collection of Book objects, demonstrating the relationship between container and contained objects.
- **Dynamic Storage:** Dynamic arrays (`ArrayList` in Java, `vector` in C++) are used to manage the book list efficiently.
- **Business Logic:** The system implements validation logic (e.g., checking `stock > 0` before borrowing) and string matching algorithms for the search function.