Method Chaining in C++

Method Chaining in C++

- Method chaining is a **programming technique** where **multiple function calls** on the **same object are linked together in a single statement.**
- This approach **enhances code readability** and **reduces the need for intermediate variables.**
- It is commonly used in scenarios such as configuring objects, performing sequential operations, and implementing fluent APIs.
- A technique where multiple methods are called in a single statement.
- Each method returns a reference to the calling object (*this), enabling subsequent method calls.

Benefits of Method Chaining

- **Improved Code Readability**
  - Eliminates redundant intermediate statements.
  - Keeps related operations in a single line.
- **Fluent Interface Design**
  - Makes APIs more intuitive, especially in object configuration and data manipulation.
- **Reduced Code Complexity**
  - Avoids unnecessary temporary variables and extra function calls.

Example Use Case: String Formatter

```
formatter.toUpperCase().append(" World").show();
```

Without Method Chaining:

```
formatter.toUpperCase();
formatter.append(" World");
formatter.show();
```

- **Method chaining leads to cleaner, more expressive code.**

```
class Student {
private:
   string name;
   string courses; // Stores course names as a single string
public:
   Student(string studentName) : name(studentName), courses("") {}

   Student& addCourse(string course) {
      if (!courses.empty()) {
         courses += ", "; // Add separator between courses
      }
      courses += course;
      return *this;
   }

   Student& showCourses() {
      cout << "Student: " << name << "\nCourses: " << (courses.empty() ? "None" : courses) << endl;
      return *this;
   }
};
```

```
int main() {

   Student s("Ali");

   s.addCourse("Math").addCourse("Physics").showCourses()
    .addCourse("Computer Science").showCourses();

   return 0;
}
```

Chaining Usage:

```
s.addCourse("Math").addCourse("Physics").showCourses().addCourse("ComputerScience").showCourses();
```

Returning *this from showCourses() doesn't mean the result must be stored. It just allows seamless chaining, making the code more concise

Output:

```
Student: Ali
Courses: Math, Physics
Student: Alice
Courses: Math, Physics, Computer Science
```

```
class Rectangle {
private:
   int width, height;
public:
   Rectangle() : width(0), height(0) {}

   Rectangle& setWidth(int w) {
      width = w;
      return *this;
   }

   Rectangle& setHeight(int h) {
      height = h;
      return *this;
   }

   Rectangle& showArea() {
      cout << "Area = " << (width * height) << endl;
      return *this;
   }
};
```

```
int main() {
   Rectangle r;

r.setWidth(5).setHeight(10).showArea().setWidth(7).setHeight(3).showArea();
   return 0;
}
```

Returning *this from showArea() doesn't mean the result must be stored. It just allows seamless chaining, making the code more concise

Output:

```
Area = 50
Area = 21
```

```
class BankAccount {
private:
   double balance;
public:
   BankAccount(double initial) : balance(initial) {}

   BankAccount& deposit(double amount) {
      balance += amount;
      return *this;
   }

   BankAccount& withdraw(double amount) {
      if (amount <= balance) {
         balance -= amount;
      } else {
         cout << "Insufficient funds!" << endl;
      }
      return *this;
   }

   BankAccount& showBalance() {
      cout << "Balance: $" << balance << endl;
      return *this;
   }
};
```

```
int main() {
   BankAccount acc(100);
   acc.deposit(50).showBalance().withdraw(30).showBalance().withdraw(150).showBalance();
   return 0;
}
```

Returning *this from showBalance() doesn't mean the result must be stored. It just allows seamless chaining, making the code more concise

Output:

```
Balance: $150
Balance: $120
Insufficient funds!
Balance: $120
```

```cpp
class StringFormatter {
private:
    string text;
public:
    StringFormatter(string str) : text(str) {}

    StringFormatter& toUpperCase() {
        for (char &c : text) c = toupper(c);
        return *this;
    }

    StringFormatter& toLowerCase() {
        for (char &c : text) c = tolower(c);
        return *this;
    }

    StringFormatter& append(string extra) {
        text += extra;
        return *this;
    }

    StringFormatter& show() {
        cout << "Formatted String: " << text << endl;
        return *this;
    }
};
```

```cpp
int main() {
    StringFormatter sf("Hello");
    sf.toUpperCase().show().append(" World!").show().toLowerCase().show();
    return 0;
}
```

Returning *this from show() doesn't mean the result must be stored. It just allows seamless chaining, making the code more concise

Output:

```
Formatted String: HELLO
Formatted String: HELLO World!
Formatted String: hello world!
```

```
class FileWriter {
private:
    ofstream file;
public:
    FileWriter(string filename) {
        file.open(filename);
    }

    FileWriter& write(string text) {
        if (file.is_open()) {
            file << text << endl;
        }
        return *this;
    }

    FileWriter& close() {
        if (file.is_open()) {
            file.close();
        }
        return *this;
    }
};
```

```
int main() {
    FileWriter fw("output.txt");
    fw.write("Hello, World!").write("This is C++ chaining
example.").close();
    return 0;
}
```

```
class Rectangle {
private:
   double length, width;
public:
   Rectangle() : length(0), width(0) {}

   Rectangle& setLength(double l) {
      length = l;
      return *this;
   }

   Rectangle& setWidth(double w) {
      width = w;
      return *this;
   }

   Rectangle& showArea() {
      cout << "Area: " << length * width << endl;
      return *this;
   }
};
```

```
int main() {
   Rectangle r;
   r.setLength(10).setWidth(5).showArea();
   return 0;
}
```

```
class Query {
private:
    string query;
public:
    Query() : query("SELECT * FROM students") {}

    Query& where(string condition) {
        query += " WHERE " + condition;
        return *this;
    }

    Query& orderBy(string column) {
        query += " ORDER BY " + column;
        return *this;
    }

    Query& execute() {
        cout << "Executing: " << query << endl;
        return *this;
    }
};
```

```
int main() {
    Query q;
    q.where("age > 18").orderBy("name").execute();
    return 0;
}
```