Heaven's Light is Our Guide

Rajshahi University of Engineering & Technology

Course No.

ECE 3118

Course Title

Software Engineering & Information System Design Sessional

Lab Report-02

Submitted To	Submitted By
Oishi Jyoti Assistant Professor, Department of ECE, RUET	Tanha Farzana Roll:2010022 3rd year, Odd Semester

2.1 Experiment No.: 02

2.2 Name of the Experiment: Best Coding Practices.

2.3 Objectives:

1. To know about different coding style.
2. To reduce the effort needed to understand source code of any problem.
3. To know what conventions & indentions needed for best coding practices.

2.4 Theory:

Best coding practices are a set of guidelines and principles that developers follow to write high-quality, reliable, and maintainable code. They provide a standardized approach to coding, ensuring that code is clean, readable, and efficient. The Best coding practices include: using naming conventions for variables, placing whitespaces, indentations and tabs within code, adding comments throughout to aid its interpretation. Here are some reasons why following best coding practices is important:

- Code Readability: Writing code that is easy to read and understand is essential for collaboration among team members and future maintenance.
- Code Maintainability: Best coding practices promote code maintainability by organizing code in a structured and modular way. This allows for efficient bug fixing, feature addition, and code refactoring, even as the codebase grows in size and complexity.
- Code Reusability: Following best coding practices encourages the creation of reusable code modules and components. This saves development time, reduces redundancy, and promotes consistency throughout the codebase.
- Code Efficiency: Best coding practices emphasize writing efficient code that executes tasks quickly and optimally utilizes system resources.
- Code Consistency: Consistency is a fundamental aspect of best coding practices.

Programmers have used different case types to name different entities in their code. Some them are:

1. Camel Case
2. Snake Case
3. Pascal Case

2.5 Source Code:

```
1. #include <iostream>
2. using namespace std;
3. class Bank
4. {
5. private:
6.     int accountNo;
7.     char name[30];
8.     long balance;
```

```
9.
10.     public:
11.         void open_account()
12.         {
13.             cout << "Enter Account Number: ";
14.             cin >> accountNo;
15.             cout << "Enter Name: ";
16.             cin >> name;
17.             cout << "Enter  Balance: ";
18.             cin >> balance;
19.         }
20.         void show_account()
21.         {
22.             cout << "Account Number: " << accountNo << endl;
23.             cout << "Name: " << name << endl;
24.             cout << "Balance: " << balance << endl;
25.         }
26.         void deposit()
27.         {
28.             long amt;
29.             cout << "Enter Amount U want to deposit? ";
30.             cin >> amt;
31.             balance = balance + amt;
32.         }
33.         void withdrawal()
34.         {
35.             long amt;
36.             cout << "Enter Amount U want to withdraw? ";
37.             cin >> amt;
38.             if (amt <= balance)
39.                 balance = balance - amt;
40.             else
41.                 cout << "Less Balance..." << endl;
42.         }
43.         int search(int);
44.     };
45.
46.     int Bank::search(int a)
47.     {
48.         if (accountNo == a)
49.         {
50.             show_account();
51.             return (1);
52.         }
53.         return (0);
54.     }
55.
```

```
56.     int main()
57.     {
58.         int n;
59.         cout << "Enter the number of accounts : ";
60.         cin >> n;
61.         Bank C[n];
62.
63.         int found = 0, a, ch, i;
64.         for (i = 0; i < n; i++)
65.         {
66.             C[i].open_account();
67.         }
68.
69.         do
70.         {
71.             cout << "nn1:Display Alln2:By Account
    Non3:depositn4:Withdrawn5:Exit" << endl;
72.
73.             cout << "Please input your choice: ";
74.             cin >> ch;
75.
76.             switch (ch)
77.             {
78.             case 1: // displaying account info
79.                 for (i = 0; i <= 2; i++)
80.                 {
81.                     C[i].show_account();
82.                 }
83.                 break;
84.             case 2: // searching the record
85.                 cout << "Account Number? ";
86.                 cin >> a;
87.                 for (i = 0; i <= 2; i++)
88.                 {
89.                     found = C[i].search(a);
90.                     if (found)
91.                         break;
92.                 }
93.                 if (!found)
94.                     cout << "Record Not Found" << endl;
95.                 break;
96.             case 3: // deposit operation
97.                 cout << "Account Number To deposit Amount? ";
98.                 cin >> a;
99.                 for (i = 0; i <= 2; i++)
100.                {
101.                    found = C[i].search(a);
```

```
                    if (found)
                    {
                        C[i].deposit();
                        break;
                    }
                }
                if (!found)
                    cout << "Record Not Found" << endl;
                break;
            case 4: // withdraw operation
                cout << "Account Number To Withdraw Amount? ";
                cin >> a;
                for (i = 0; i <= 2; i++)
                {
                    found = C[i].search(a);
                    if (found)
                    {
                        C[i].withdrawal();
                        break;
                    }
                }
                if (!found)
                    cout << "Record Not Found" << endl;
                break;
            case 5: // exit
                cout << "Have a nice day" << endl;
                break;
            default:
                cout << "Wrong Option" << endl;
            }
        } while (ch != 5);
        return 0;
    }
```

2.6 Description:

Class Names: Here we use PascalCase in defining class name, where the first letter of each word is capitalized.

Example: `class Bank`

Public Function Names: Here we use snake_case in defining function names, starting with a lowercase letter and there is a underscore (‘_’) between two words.

Example:
```
public:
    void open_account()
```

Variables Names: Here we use camelCase in defining Variables name, starting with a lowercase and there is uppercase in starting of new words.

Example: `int accountNo`

Commenting in Code: Commenting in code is the practice of adding notes or explanations to make the code more understandable for humans.

Example: `// displaying account info`

Blank space and Indentation in Code: Blank space refers to the empty areas between characters, lines, or blocks of code. Indentation is the deliberate use of spaces or tabs to visually structure and organize code for readability.

2.7 Discussion & Conclusion:

Best coding practices are crucial for producing high-quality, readable, and maintainable code. They enhance code efficiency, security, and collaboration. By adhering to these practices, developers can ensure their code is reliable, easier to understand, and less prone to errors, resulting in better software development outcomes.