# SPL-1 Project Report

# PseudoCoder


Submitted by

**Shazzad Hossain**

**BSSE Roll No. : 1203**

**BSSE Session: 2019-2020**


Submitted to

**Dr. Md. Mahbubul Alam Joarder**

**Professor, IIT, DU**



**Institute of Information Technology**

**University of Dhaka**


30-05-2022

**Table of Contents**

## 1. Introduction

A Class Diagram in the Unified Modeling Language (UML) is a type of static structure diagram that depicts the structure of a system by displaying the system's classes, their attributes, operations (or methods), and the relationships among objects. The class diagram is the foundation of object-oriented modeling. It is used for both general conceptual modeling of the application's structure and detailed modeling, which involves translating the models into programming code.

This project translates the models into programming code i.e., create a basic structure of the project. It is useful for projects that are very large, as by using this tool the entire structure can be instantly created, and the programmer can then just focus on the logical implementation part. Moreover, since this program can generate source code in two different languages, situation-based structure generation can also come in handy.

## 2. Background of the Project

To implement this project, some prior study was necessary:

### 2.1. Object Oriented Concepts

Object-oriented programming (OOP) is a computer programming model that organizes software design around data, or objects, rather than functions and logic. An object can be defined as a data field that has unique attributes and behavior. The OOP concepts are as follows:

- Class
- Object
- Method and Method Passing
- 4 pillars: Abstraction, Encapsulation, Inheritance and Polymorphism.

### 2.2. UXF Language

UML eXchange Format (UXF) is an XML-based model interchange format for Unified Modeling Language (UML), which is a standard software modeling language.

### 2.3. Implementation of OOP concept in C++ and Java

Even though the same concepts are being implemented in 2 languages, there are some major differences in syntax of these languages, and there are some restrictions and drawbacks too. For example, C++ supports virtual constructors and destructors, multiple inheritances, pointers, whereas Java restricts multiple inheritances, classes under the same package do not need preprocessing, there are no pointers and so on.

## 3. Description of the Project

There are two parts in this project:

- Parsing UML class diagram
- Generating source code

### 3.1. Parsing UML class diagram

In this part, the main goal is to extract the data of a class. Firstly, a .uxf file is taken as input, and it is parsed (tokenized and stored in structures) to group the necessary data for a class, such as

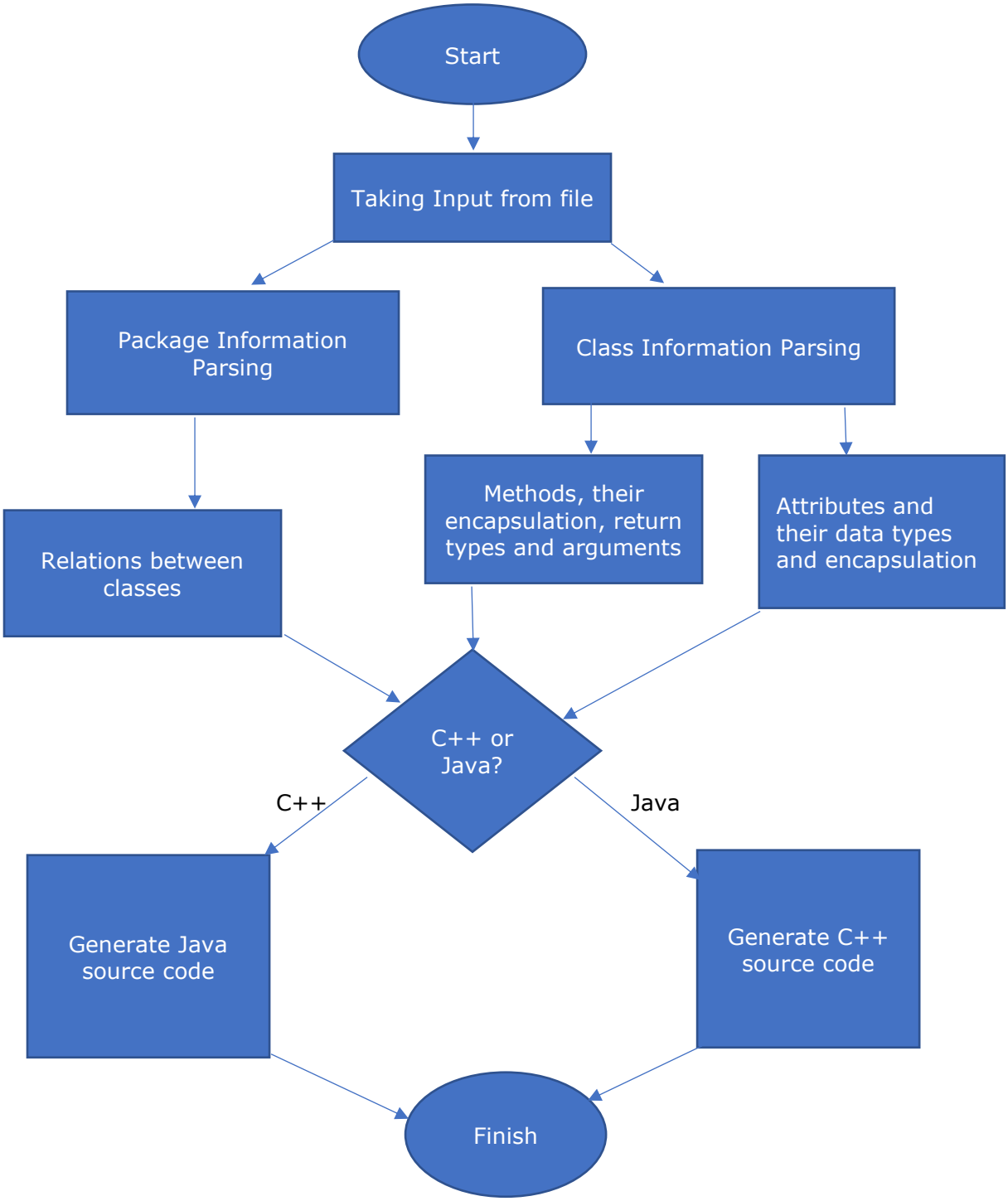
- class name

- attributes and their data types
- methods and their return type and arguments
- encapsulation of attributes and methods
- abstractions i.e., abstract class/interface
- Instance-level relationships (dependency, association, aggregation, composition)
- Class-level relationships (inheritance, implementation)

String matching algorithms are used in some parts of the parsing.

### 3.2. Generating Source Code

The user chooses which language the source codes are to be generated. Information obtained from the diagram is then implemented in that language by following the OOP concepts.



## 4. Implementation and Testing

I have implemented KMP algorithm for searching a particular pattern in the input and all basic OOP concepts that can be found in an UML class diagram. After implementation, I have taken several class diagrams for testing.

Some implementation code snippets are given:

```
Package parsePackage(const tinyxml2::XMLElement *xmlElement)
{
    Package package;

    xmlElement->FirstChildElement("coordinates") >> package.coords;
    package.name = xmlElement->FirstChildElement("panel_attributes")->GetText();
    package.name = package.name.substr(0, package.name.find('\n'));

    return package;
}
```

Fig: Package Parser

```
struct ClassInfo
{
    string name;
    Rectangle coords;
    vector<Attribute> attributes;
    vector<Method> methods;
    vector<string> dependantClasses;

    bool isInterface;
    vector<Inheritance> inheritances;
```

Fig: Class Information Structure

```
ClassInfo parseClassInfo(const tinyxml2::XMLElement *xmlElement)
{
    ClassInfo classInfo;

    xmlElement->FirstChildElement("coordinates") >> classInfo.coords;

    std::string content = xmlElement->FirstChildElement("panel_attributes")->GetText();

    auto lines = split(content, "\n");

    size_t n = 0;

    classInfo.isInterface = false;

    for (; n < lines.size() && lines[n] != "--"; n++)
        if (lines[n].empty())
            continue;
        else if (lines[n].find("interface") < lines[n].length())
            classInfo.isInterface = true;
        else if (lines[n].find("layer") == 0 || lines[n].find("bg") == 0 || lines[n].find('{') == 0)
            continue;
        else if (classInfo.name.empty())
            classInfo.name = lines[n];
        else
            throw std::runtime_error("Error parsing class title!");

    n++;

    for (; n < lines.size() && lines[n] != "--"; n++)
        if (lines[n].empty())
            continue;
        else
            classInfo.addAttribute(parseAttribute(lines[n]));

    n++;

    for (; n < lines.size() && lines[n] != "--"; n++)
        if (lines[n].empty())
            continue;
        else
            classInfo.methods.push_back(parseMethod(lines[n]));

    return classInfo;
}
```

Fig: Class Parser

```c
int find(char *pat, char *txt) //uses KMP Pat
{
    int M = strlen(pat);
    int N = strlen(txt);
    int lps[M];
    computePiTable(pat, M, lps);
    int i = 0, j = 0;
    while (i < N)
    {
        if (pat[j] == txt[i])
            j++, i++;

        if (j == M)
            return i - j;

        else if (i < N && pat[j] != txt[i])
        {
            if (j != 0)
                j = lps[j - 1];
            else
                i = i + 1;
        }
    }
}
```

```c
void computePiTable(char *pat, int M, int *lps)
{
    int len = 0;
    lps[0] = 0;
    int i = 1;
    while (i < M)
    {
        if (pat[i] == pat[len])
        {
            len++;
            lps[i] = len;
            i++;
        }
        else
        {
            if (len != 0)
                len = lps[len - 1];
            else
                lps[i++] = 0;

        }
    }
}
```

Fig: KMP Pattern Matching Algorithm

```java
class Bank {
    private int bankID;
    private String name;
    private String address;
    private Teller tellers;
    private Customer customers;

    public int getBankID() {
        return bankID;
    }

    public void setBankID(int newBankID) {
        this.bankID = newBankID;
    }

    public String getName() {
        return name;
    }

    public void setName(String newName) {
        this.name = newName;
    }
```

Fig: A generated class in Java as output

## 5. User Interface

It is a simple console-based application that generates source code files as output.

Some screenshots of UI are provided below:

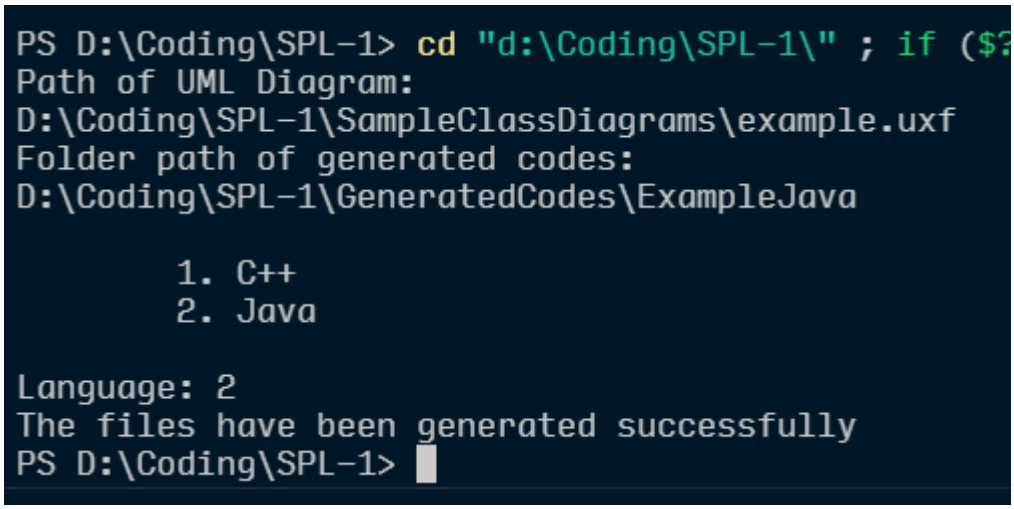
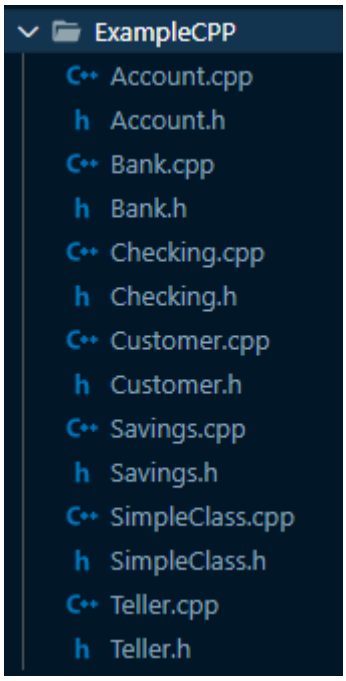



Fig: User Interface



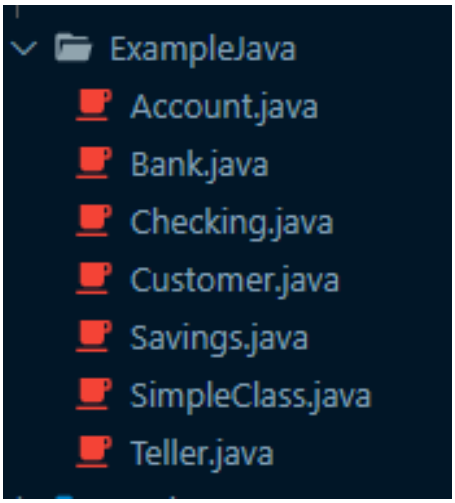




Fig: Folders where source codes are generated

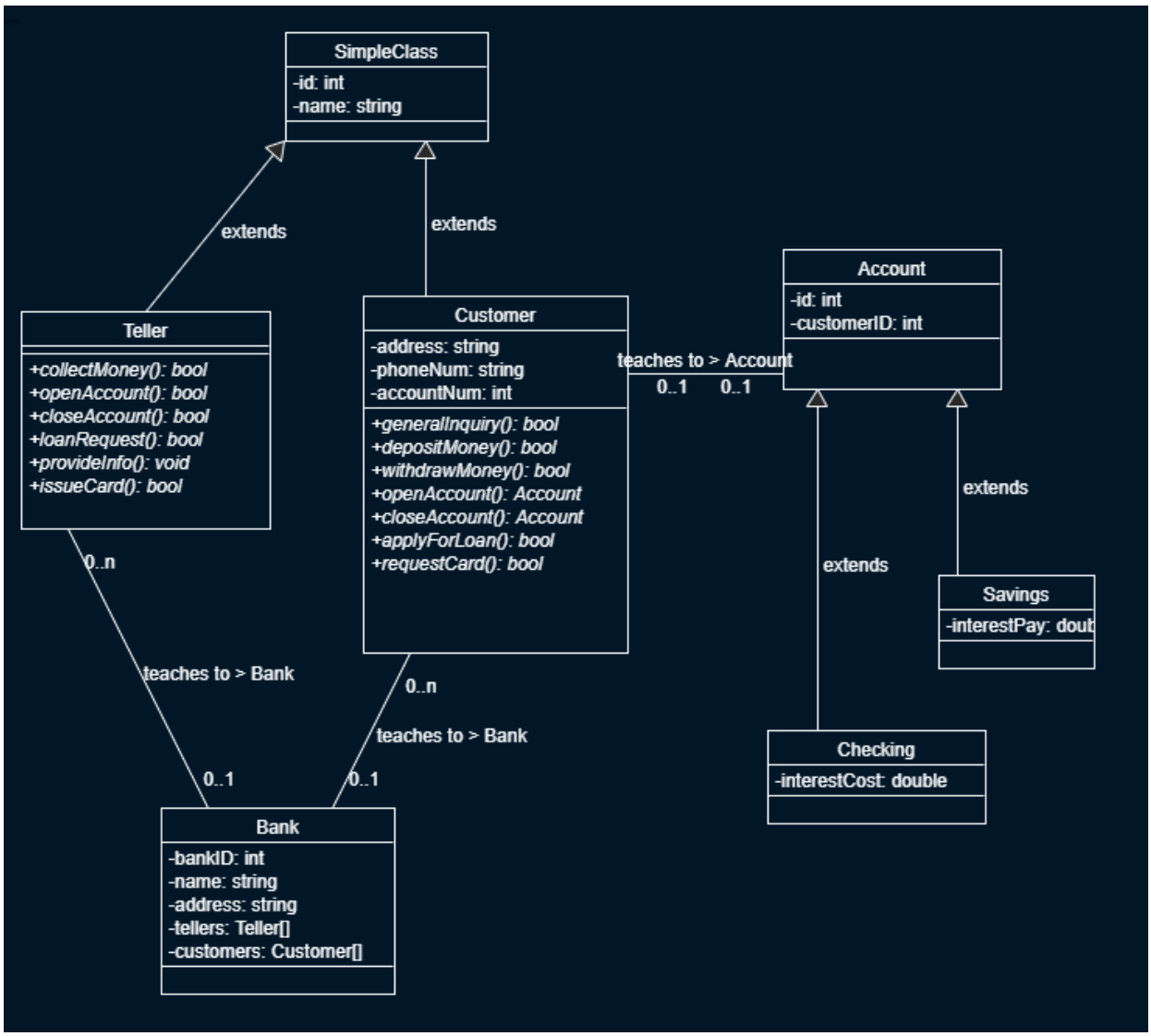



Fig: UML Diagram that was given as input

## 6. Challenges Faced

- First time using header files and multiple C++ files
- UXF file handling
- Implementation of OOP Concepts in 2 different languages
- Handling large packages
- Manually checking if the generated source codes were correct

## 7. Conclusion

While doing this project, I had to learn about OOP modeling and implementation in detail. This project can be extended to have a UML class diagram drawing tool to be inclusive, thus, users can draw and then generate the structure of a large project in the same application.

**GitHub link:**

**https://github.com/shazzad5709/SPL-1**

**Reference**

https://www.javatpoint.com/uml-class-diagram (last accessed on 26/05/22)

https://www.uml-diagrams.org/index-examples.html (last accessed on 28/05/22)

https://en.wikipedia.org/wiki/Class_diagram (last accessed on 29/05/22)

https://www.geeksforgeeks.org/object-oriented-programming-in-cpp/ (last accessed on 09/05/22)

https://github.com/leethomason/tinyxml2 (last accessed on 20/05/22)