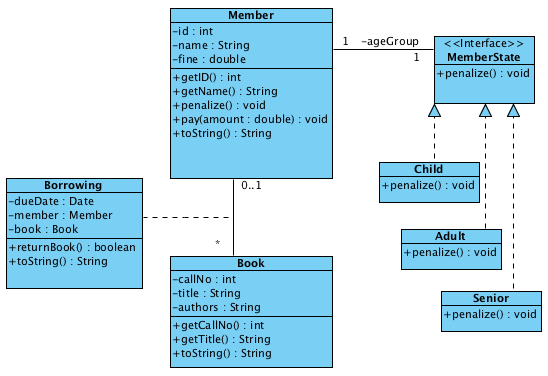
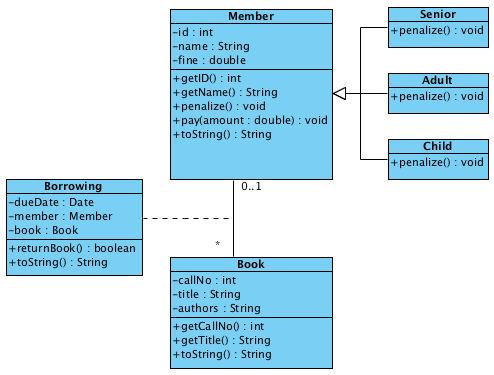
**CS3342 – Software Design**

**Tutorial No.2 – OO Basics and Examples**

**Section A – Library System**

In Library System Ver. 3.0, we need to support **child**, **adult** and **senior** library members, with different late book return penalty fine rate (i.e. $3/day for children, $10/day for adult, $5/day for seniors). We also know that a child will becoming an adult in one day and an adult will become a senior in one day. Explain why Design B is better than Design A?

**Design A: (Using Inheritance) Design B: (Interface/Implementation)**



**Task 1:** Using Visual Paradigm to create two separate projects (name **Library\_System\_Design\_A** and **Library\_System\_Design\_B),** and recreate the class diagrams for Design A and Design B.

**Task 2:** Using Visual Paradigm to generate Java Source Codes for **Library\_System\_Design\_A** and **Library\_System\_Design\_B** respectively.

**Task 3:** Open BlueJ and create two separate projects (Design\_A and Design\_B), and important (drag\_and\_drop) the generated code into the BlueJ system. Add “**import java.util.Date;”** to the first line of class **Borrowing**.

**Task 4:** Analyze the differences between the two designs. Attempt to answer why Design B is better than Design A?

**Task 5 (For Avanced Students, Optional):** Attempt to complete the implementation in BlueJ.

**Task 4 Answer:**

The difference between Design A and Design B is on whether we want to model each type of person in the real world by exactly one member object in the entire life of the system.

**Design A** assumes that at different stages of a *man* (modeled as a member), we need to represent the man by different types of objects. Design B assumes that at different stages, the same member refers to the same man, and to model different stages, Design B uses MemberState objects of different belonging classes.

**Design B** is better in the sense that the conceptual constraints of Member is closer to our real world and this conceptual constraint needs to be implemented as stated in the question “We also know that a child will become an adult in one day and an adult will become a senior in one day”.

i.e. it also models “the change of state” from one to another.

Note that if the conceptual constraint does not exist, i.e., there is no requirement to model the change of states of a member, Design A is better than Design B because Design B has over-engineered the solution.

**Section B – City Online Store**

Below shows a restructuring of the C++ program from using procedural approach to using object-oriented approach.

**Analyze the following code that models the program using the OO approach**.

*Procedural approach*

#include <iostream>

#include <string>

using namespace std;

void main() {

// 1. Get input.

int n;

cin >> n;

double \*amount = new double[n];

for (int i=0; i<n; i++)

cin >> amount[i];

// 2. Process input and display output

// 2.1 Setup Customer's initial state

double total = 0;

string state = "Typical";

// 2.2 Process each transaction

for (int i=0; i<n; i++) {

// 2.2.1. Setup discount

int discount;

if (state == "Typical")

discount = 100;

else if (state == "VIP")

discount = 80;

else if (state == "Golden")

discount = 50;

double final = amount[i] \* discount / 100;

// 2.2.2. Setup deposit to be paid

int depositRatio;

if (state == "Typical")

depositRatio = 100;

else if (state == "VIP")

depositRatio = 50;

else if (state == "Golden")

depositRatio = 0;

double deposit = final \* depositRatio / 100;

// 2.2.3 Check for promotion

total += amount[i];

if (total > 10000)

state = "Golden";

else if (total > 5000)

state = "VIP";

// 2.2.4 Display output

cout << "Selling price: " << amount[i] << ", "

<< "Discounted price: " << final << ", "

<< "Deposit: $" << deposit << ", "

<< "Type: " << state << "\n";

}

}

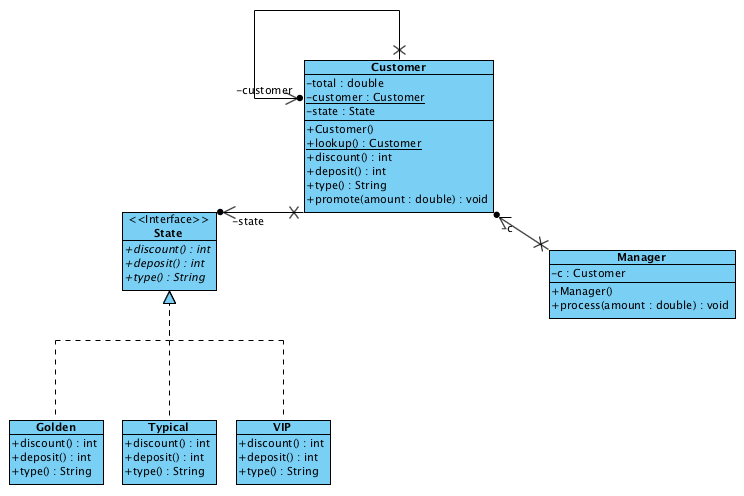
Here is a sample input and output of the program:

|  |  |
| --- | --- |
| Input:  5  3000  3000  3000  3000  3000 | Output:  Selling price: 3000, Discounted price: 3000, Deposit: $3000, Type: Typical  Selling price: 3000, Discounted price: 3000, Deposit: $3000, Type: VIP  Selling price: 3000, Discounted price: 2400, Deposit: $1200, Type: VIP  Selling price: 3000, Discounted price: 2400, Deposit: $1200, Type: Golden  Selling price: 3000, Discounted price: 1500, Deposit: $0, Type: Golden |

*Object-oriented approach in C++*

|  |  |
| --- | --- |
| #include <iostream>  #include <string>  using namespace std;  class State {  public:  virtual int discount() = 0;  virtual int deposit() = 0;  virtual string type() = 0;  };  class Typical: public State {  public:  int discount() { return 100; }  int deposit() { return 100; }  string type() { return "Typical"; }  };  class VIP: public State {  public:  int discount() { return 80; }  int deposit() { return 50; }  string type() { return "VIP"; }  };  class Golden: public State {  public:  int discount() { return 50; }  int deposit() { return 0; }  string type() { return "Golden"; }  };  class Customer {  public:  Customer(): total(0) { state = new Typical(); }  static Customer\* lookup() { return customer; }  int discount() { return state->discount(); }  int deposit() { return state->deposit(); }  string type() { return state->type(); }  void promote(double amount) {  total += amount;  if (total > 10000)  state = new Golden();  else if (total > 5000)  state = new VIP();  }  private:  static Customer\* customer;  double total;  State\* state;  }; | Customer\* Customer::customer = new Customer();  class Manager {  public:  void process(double amount, ostream& outs) {  Customer \*c = Customer::lookup();  double final = amount \* c->discount() / 100;  double deposit = final \* c->deposit() / 100;  c->promote(amount);    outs << "Selling price: " << amount << ", "  << "Discounted price: " << final << ", "  << "Deposit: $" << deposit << ", "  << "Type: " << c->type() << "\n";  }  };  class IO {  public:  void getInput() {  Manager manager;  int n;  cin >> n;  double \*amount = new double[n];  for (int i=0; i<n; i++)  cin >> amount[i];  for (int i=0; i<n; i++)  manager.process(amount[i], cout);  }  };  void main() {  IO io;  io.getInput();  } |

**Task 1 - Design:** VP: Below is a Simple Class diagram (BlueJ) Please attempt to use Visual Paradigm to generate a more complete Class Diagram.



**Task 2 - Implementation:** BlueJ: Create a BlueJ project (see below), and convert the C++ code into Java, solution provided on Canvas for your reference.

