KIE1008: Programming 2
Week 6: Object-Oriented Programming – Polymorphism

1. Suppose that you have the following classes, `small` and `notSmall`:

```
class small
{
public:
    void print() const;
    int add() { return x + y; }
    small() {}
    small(int a, int b) { x = a; y = b; }

private:
    int x = 0;
    int y = 0;
};

void small::print() const
{
    cout << "small: -- "<< endl
         << "x: " << x << ", y = " << y << endl;
}

class notSmall: public small
{
public:
    void print() const;
    int add();
    notSmall() {}
    notSmall(int a, int b, int c)
        : small(a, b) { z = c; }

private:
    int z;
};

void notSmall::print() const
{
    small::print();
    cout << "noSmall--- z: " << z << endl;
}

int notSmall::add()
{
    return z + small::add();
}
```

What is the output of the following function `main`?

```
int main()
{
    small *ptrSmall;
    small objectSmall(2, 3);
    notSmall objectBig(3, 5, 9);
    ptrSmall = &objectSmall;
    cout << ptrSmall->add() << endl;
    ptrSmall->print();
    cout << "*-*-*-*-*-*-*-*-*-*-*-*" << endl << endl;
    ptrSmall = &objectBig;
    cout << ptrSmall->add() << endl;
```

```
    ptrSmall->print();
    return 0;
}
```

2. Consider the following definition of the class `student`.

```
class studentType: public personType
{
public:
    void print();
    void calculateGPA();
    void setID(long id);
    void setCourses(const string c[], int noOfC);
    void setGrades(const char cG[], int noOfC);

    void getID();
    void getCourses(string c[], int noOfC);
    void getGrades(char cG[], int noOfC);
    studentType(string fName = "", string lastName = "",
                long id = -1, string *c = nullptr,
                char *cG = nullptr, int noOfC = 0);

private:
    long studentId;
    string courses[6];
    char coursesGrade[6];
    int noOfCourses;
};
```

   Rewrite the definition of the class `studentType` so that the functions `print` and `calculateGPA` are pure `virtual` functions.

3. Use abstract classes and pure virtual functions to design classes to manipulate various types of accounts.
   a) Create an abstract class called `Account`
      i. Two `private` member data: `number` (`string`) and `balance` (`double`).
      ii. Create a constructor that takes in two parameters to set the value of `number` and `balance`.
      iii. Create the accessor and mutator functions.
      iv. Create a `virtual` function `displayInfo()`.
      v. Create a pure `virtual` function `calculateBalance(int)`. (This function takes an integer as argument to calculate the balance after a few months)
   b) Create a concrete class called `Person`
      i. Two `protected` member data: `name` (`string`) and `nric` (`string`).
      ii. Create a constructor that takes in two parameters to set the value of `name` and `nric`.
      iii. Create the accessor and mutator functions.
      iv. Create a `virtual` function called `displayInfo()`.
   c) Create a class called `Saving` that is derived from `Account` and `Person`
      i. One `private` member data for the monthly interest rate in percentage: `rate` (`double`).
      ii. Constructor will take 5 arguments; 2 to set `Account`, 2 to set `Person`, 1 to set `rate`.
      iii. Create the accessor and mutator functions for `rate`.
      iv. Override the `displayInfo()` to display an output as shown in the sample below.

v. Write a function `calculateBalance(int)` to add the interest to the balance every month.

d) Create a class called `Current` that is derived from `Account`
   i. One `private` member data for the monthly service charge: `charge` (`double`).
   ii. Constructor will take 5 arguments; 2 to set `Account`, 2 to set `Person`, 1 to set `charge`.
   iii. Create the accessor and mutator functions for `charge`.
   iv. Override the `displayInfo()` to display an output as shown in the sample below.
   v. Write a function `calculateBalance(int)` to deduct the service charge from the balance every month.

e) Write a `main()` function to test the classes. Show the balance of a saving account `s1` and a current account `c1` for the first 12 months.

Sample output:

```
Name: Kelvin Lee (NRIC: 960113-12-4567)
Account Number: 751-2569-10
Initial Balance: 1000
Account Type: Saving.
Monthly Interest Rate: 0.15%
Balance after 1 month(s): 1001.5
Balance after 2 month(s): 1003
Balance after 3 month(s): 1004.51
Balance after 4 month(s): 1006.01
Balance after 5 month(s): 1007.52
Balance after 6 month(s): 1009.03
Balance after 7 month(s): 1010.55
Balance after 8 month(s): 1012.06
Balance after 9 month(s): 1013.58
Balance after 10 month(s): 1015.1
Balance after 11 month(s): 1016.62
Balance after 12 month(s): 1018.15


Name: Danny Lau (NRIC: 861010-13-8902)
Account Number: 752-9625-12
Initial Balance: 2000
Account Type: Current.
Monthly Charge: 10
Balance after 1 month(s): 1990
Balance after 2 month(s): 1980
Balance after 3 month(s): 1970
Balance after 4 month(s): 1960
Balance after 5 month(s): 1950
Balance after 6 month(s): 1940
Balance after 7 month(s): 1930
Balance after 8 month(s): 1920
Balance after 9 month(s): 1910
Balance after 10 month(s): 1900
Balance after 11 month(s): 1890
Balance after 12 month(s): 1880
```