Object Oriented Programming – Assignment 3
Spring 2022

Note: There are two questions for this assignment.

Q1: BANK MANAGEMENT SYSTEM

All Bank Accounts

1. 1. All bank accounts have a balance that is stored as an amount in pennies.
2. 2. All money amounts are expressed in pennies.
3. 3. All debit amounts will be subtracted from the balance, and all credit amounts will be added to the balance.
4. 4. All bank accounts have a non-negative interest rate (0.02 would be a 2% interest rate).
5. 5. When applying interest, interest amount is calculated by multiplying the balance by the interest rat
6. 6. When applying interest, interest amount is **added** to the balance
7. 7. Interest will not be applied to any account with a balance of zero.
8. 8. Debit methods will return false if the transaction cannot be made because of insufficient balance or insufficient credit limit Otherwise they will return true.

Savings accounts

1. 1. A savings account cannot have a negative balance
2. The debit method will return false if an attempt to overdraw the account is made.
3. 2. The getAccoutInfoAsString method will return a string formatted like this:

   Account type : Savings

   Account #   : 101101

   Balance   : 12345

Interest rate : 0.01

<u>Checking accounts</u>

1. 1. Any CheckingAccount debit that ends with a negative balance will incur an overdraftFee (i.e. the overdraftFee amount will be subtracted from the balance)
2. 2. The getAccoutInfoAsString method will return a string formatted like this:

   Account type : Checking

   Account #   : 202202

   Balance   : 200000

   Interest rate : 0.02

   Overdraft fee : 2000

<u>Credit card accounts</u>

1. 1. The balance of a Creditcard account cannot overrun its credit limit
2. The debit method will return false if an attempt to overdraw the account is made.
3. 2. Interest will not be applied to a Creditcard account with a positive balance
4. 3. The getAccoutInfoAsString method will return a string formatted like this:

   Account type : Creditcard

   Account #   : 303303

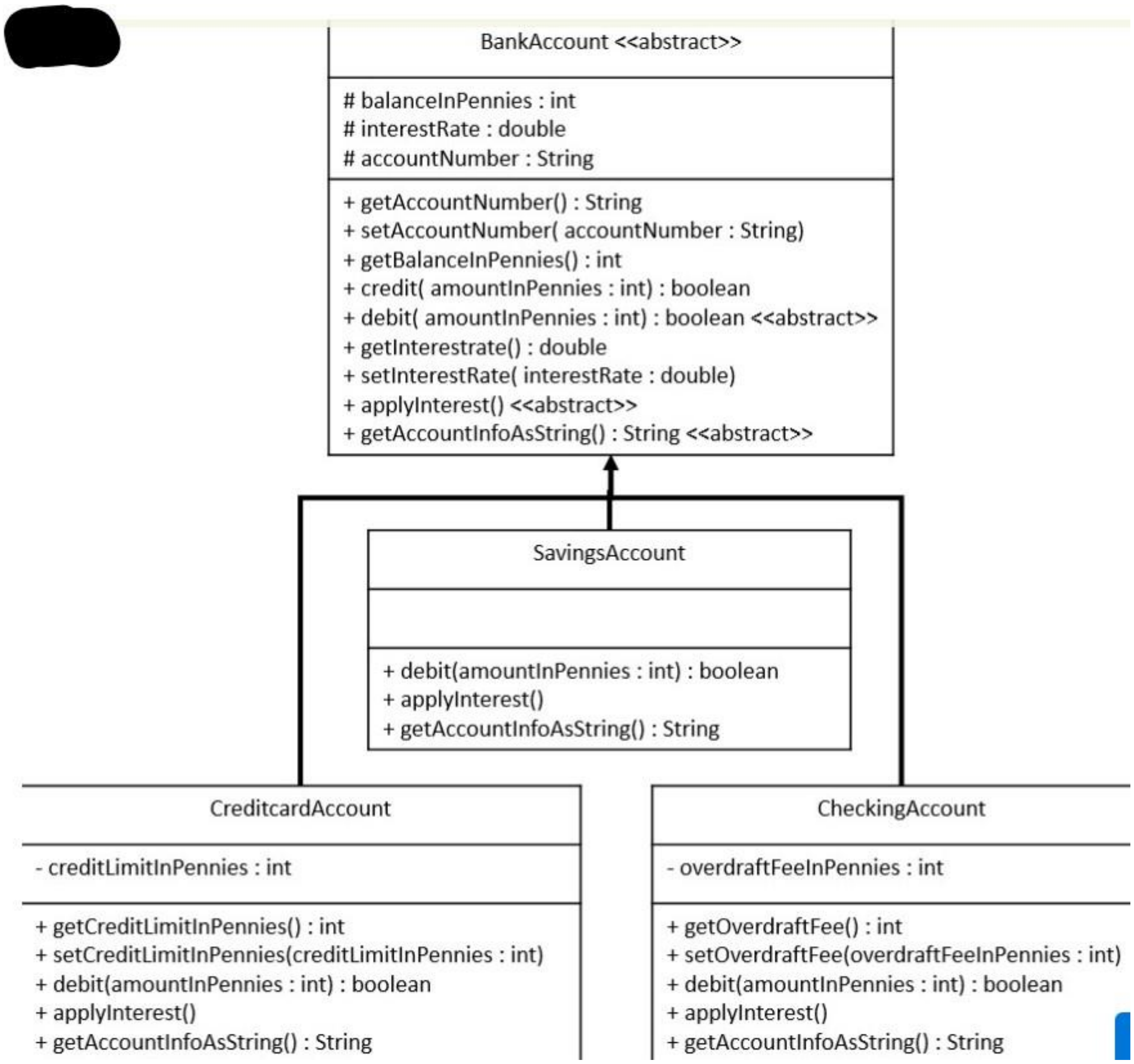
   Balance   : -27550

   Interest rate : 0.03

Credit limit : 1000000

UML DAIGRAM: Below are the prototypes of function and class members you have to implement (Polymorphism and Interest).

[redacted]


BankAccount <<abstract>>
# balanceInPennies : int
# interestRate : double
# accountNumber : String
+ getAccountNumber() : String
+ setAccountNumber(accountNumber : String)
+ getBalanceInPennies() : int
+ credit(amountInPennies : int) : boolean
+ debit(amountInPennies : int) : boolean <<abstract>>
+ getInterestrate() : double
+ setInterestRate(interestRate : double)
+ applyInterest() <<abstract>>
+ getAccountInfoAsString() : String <<abstract>>
SavingsAccount
+ debit(amountInPennies : int) : boolean
+ applyInterest()
+ getAccountInfoAsString() : String
CreditcardAccount
- creditLimitInPennies : int
+ getCreditLimitInPennies() : int
+ setCreditLimitInPennies(creditLimitInPennies : int)
+ debit(amountInPennies : int) : boolean
+ applyInterest()
+ getAccountInfoAsString() : String
CheckingAccount
- overdraftFeeInPennies : int
+ getOverdraftFee() : int
+ setOverdraftFee(overdraftFeeInPennies : int)
+ debit(amountInPennies : int) : boolean
+ applyInterest()
+ getAccountInfoAsString() : String

Q2: Polymorphism Exercise

Exercise 1:

Consider the following hierarchy as it exists in a university:

- There are two types of persons in the university i.e. Student and Faculty
- Every Person has some basic information that is common to all persons i.e. the first_name and last_name stored as private attributes and age which is a protected attribute.
- A student can in turn be either an Undergraduate or a Graduate student, every student has a cgpa.
- An undergraduate student has a fyp_name as his private attribute.
- A graduate student has a thesis topic as his private attribute.
- A faculty member has private attributes about the number of courses he is currently teaching, i.e. his course_count and a three digit telephone extension number.

Draw Class Diagram to represent the hierarchy of classes that you see in the description above. Include all the attributes in your diagram.

Exercise 2:

Implement the entire hierarchy of the Class diagram you created in Exercise 1 i.e. define all the classes along with their attributes and their inheritance. Every class should be defined in a separate header file named according to the class name.

Exercise 3:

Add appropriate constructors and destructors to all the classes created in Exercise 2. For example the constructor for the Person class should take three inputs (for **`first_name`**, **`last_name`** and **`age`**). The student constructor should take four inputs, three for its parent class (i.e. Person) and one float value to be assigned to the **`cgpa`** attribute.

This is accomplished in the following manner:

```
Person (char* fname, char* lname, int age)
{
       ...
       cout << ”Person() called”;
}

Student (char* fname,char* lname,int age,float cgpa): Person(fname,lname,age)
{
       ...
       cout << ”Student() called”;
}
```

This syntax can be generalized to any parent and child constructor accordingly. Following this syntax, define and implement constructors and destructors for all the classes. Also, Notice that you have to add a print statement in every constructor which announces that the constructor has been called.

Also add a print statement to every destructor which announces that the destructor has been called. For example, the destructor for Person should look like:

```
~Person()
{
       cout << ”~Person() called”;
}
```

Exercise 4:

In separate implementation files, add getters and setters for all attributes in all the classes that you have defined.

Exercise 5:

Create a C++ source file called **`Inheritance.cpp`**. This file contains the **`main()`** function. In this main function create an undergraduate student “Ted Thompson” with cgpa 3.91 who is 22 years of age and a faculty member “Richard Karp” who is 45 years of age and who is teaching 2 courses this semester and his extension number is 420. Build and execute the code, copy the output and paste inside /* comments */ in your **`Inheritance.cpp`** file.

It should be pasted like this:

```
/*
 Output for Exercise 5:

~Person() called
...
...
*/
```

Exercise 6:

You should notice that the **`age`** attribute in a Person is protected, while the **`first_name`** and **`last_name`** attributes are private. What could be the reason for this?

Add a member function **`void printInformation()`** in the Person class. This method should print the name and age of the person.

Sample output: “Ted Thompson is 22 years old”

Add a member function **`void printStudent()`** in the Student class. This method should print the name, cgpa and age of the student.

Sample output: “Ted Thompson is 22 years old, his cgpa is 3.91”

Try to use the following implementation for this function.

```
void Student::printStudent()
{

      cout << first_name << ” ” << last_name
           << ”is  ” << age <<” years old, his cgpa is ” << cgpa;
}
```

Now call the **`printStudent()`** function for the student created in **`main()`** in the last exercise. Build the code, you will get an error. Paste the error in the following box.

—

Why did you get this error?

Exercise 7:

Now change the implementation of the **`printStudent()`** function in order to remove the error, but you must still print the required output. Can you use a member function of the base class inside this function? Try that!

Also add a member function **`void printFaculty()`** in the **`Faculty`** class. This function should print the name, age, number of courses and extension number of the faculty member.

Sample output: “Faculty Member name: Richard Karp, Age: 45, Number of courses: 2, Ext. 420”

Use the following **`main()`** in the **`Inheritance.cpp`** file. Build and execute the program and paste the output inside comments in the file **`Inheritance.cpp`**.

```
void main()
{
	Student s("Ted","Thompson",22,3.91);
	Faculty f("Richard","Karp",45,2,420);
		//here the number of courses is 2
		//and the extension number is 420

	s.printStudent();
	f.printFaculty();
}
```

Exercise 8:

Now add two new member functions to the Graduate and Undergraduate students. These are **`void printGraduate()`** and **`void printUndergraduate()`** respectively.

Their outputs should look like as follows:

Sample output for `void printGraduate()`
“Ted Thompson is a graduate student, his cgpa is 3.91 and his thesis topic is Distributed Algorithms”

Sample output for `void printUndergraduate()`
“Ted Thompson is an undergraduate student, his cgpa is 3.91 and his final year project is titled The Even Locator”

Change the main function in **`Inheritance.cpp`** to the following:

```
void main()
{
	Undergraduate u ("Ted","Thompson",22,3.91,"The Event Locator");
	Graduate      g ("Arnold","Gates",25,3.01,"Distributed Algorithms");


	u.printUndergraduate();
	g.printGraduate();

	u.printStudent();
}
```

Build and execute the program.

Now change the inheritance type of the **`Undergraduate`** (which is inheriting from **`Student`**) to **`protected`** (previously it was **`public`**).

```
class Undergraduate : protected Student
{
	...
};
```

Build the code again. Do you get any errors? Paste the error message in the following box.

Why did you get this error?

Now change the inheritance type to **`private`** and build the code again. Do you get any errors? Paste the error message in the following box.

Why did you get this error?

-- END ---