technische universität dortmund

Department of Computer Science
Chair 11 / Algorithm Engineering

Dr. Carsten Gutwenger

Winter 2011/12

Object-oriented Programming
Exam Sheet No. 4

Date: January 17— Due: January 24

Exam 4.1

Write a class `Account` for managing a bank account. Each account should have an account number (represented by an `int`). It shall be possible to make deposits and withdrawals, and to read out the current balance. The balance shall be stored in a protected data member of type `float`.

Remark: The access specification `protected` is similar as `private`, but derived classes are allowed to access protected data members.

Furthermore, for each transaction (deposit, withdrawal) a record shall be written to a file called `transactions.txt` (in particular, all accounts use the same file).

`Account` shall have (at least) the following public member functions:

- `virtual void withdraw(float amount)`
  withdraws `amount` from account.
- `virtual void deposit(float amount)`
  pays `amount` into account.
- `float balance() const`
  returns the current balance.

Implement these member functions and make sure that the `Account` constructor initializes the balance to zero. Moreover, it shall not be possible to withdraw an amount which is greater than the current balance.

Write a main program that tests the functionality of your `Account` class.

Exam 4.2

Derive two classes `DepositAccount` and `CreditAccount` from `Account`.

- A `DepositAccount` has to pay a fee for every withdrawal. This fee shall be stored in a private data member and shall be initialized by the constructor. Provide public member functions for reading and changing the withdrawal fee.
- A `CreditAccount` allows the account to have a negative balance (hence it is possible to withdraw an amount greater than the current balance), but only up to a certain credit line (e.g. -1000 Euro). The credit line shall be stored in a private data member and shall be initialized by the constructor. Provide public member functions for reading and changing the credit line. Make sure the credit line cannot be changed in such a way that the current balance would be less than the credit line!

Write a main program that tests the functionality of the two classes.

Exam 4.3

Modify and extend the class `SList` from Lecture 11 (available on the web page) in the following way.

- Change the value types for elements in the list from `int` to `double`.
- Add a member function `double sum() const`, which returns the sum of all the elements in the list (for an empty list, return 0).
- Add a member function `void conc(SList &L)` for concatenating two lists. This function shall append all elements from `L` to the list; the list `L` shall be empty afterwards. The function shall not allocate new elements, but only move the elements from `L` to the list.

Write a main program that tests the new functionality of `SList`.