Универзитет „Св. Кирил и Методиј“ - Скопје
ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

Object oriented programming

Exercises 7

Version 1.0, 11 April, 2017

Table of Contents

1. Inheritance

1.1. Tennis Player

Define a class for tennis player. Each player is described with first name, last name and if he plays in league.

From the class tennis player derive a class for ranked tennis player, that is player that plays on international level and is ranked with an integer for rank.

Solution `oop_av71_en.cpp`

```
#include <iostream>
#include <cstring>

using namespace std;

class TennisPlayer {
private:
    enum {STRMAX = 50};
    char name[STRMAX];
    char surname[STRMAX];
    bool playsInLeague;

public:
    TennisPlayer(const char* fn, const char* sn, bool pl);

    void print() const;

    bool PlaysInLeague();

    void setPlaysInLeague(bool pl);

    ~TennisPlayer();

    friend ostream& operator<<(ostream &o, const TennisPlayer &tp);
};

ostream& operator<<(ostream &o, const TennisPlayer &tp) {
    o << "-- Tennis Player --\n"
      << tp.name << " "
      << tp.surname << " - " << tp.playsInLeague << endl;
    return o;
}

TennisPlayer::TennisPlayer(const char* fn = "", const char* sn = "", bool pl = false) {
    cout << "Constructor:" << endl;
    strncpy(name, fn, STRMAX - 1);
    name[STRMAX - 1] = '\0';
    strncpy(surname, sn, STRMAX - 1);
    surname[STRMAX - 1] = '\0';
    playsInLeague = pl;
}

void TennisPlayer::print() const {
    cout << surname << ", " << name;
}

void TennisPlayer::setPlaysInLeague(bool pl) {
    this->playsInLeague = pl;
}

bool TennisPlayer::PlaysInLeague() {
    return this->playsInLeague;
```

```
}

TennisPlayer::~TennisPlayer() {
    cout << "Destructor TennisPlayer for: " << this->name << " " << this->surname << endl;
}

class RankedTennisPlayer : public TennisPlayer {
private:
    unsigned int rank;
public:
    RankedTennisPlayer(const char* n, const char* sn, bool pl = false, int r = 0)
        : TennisPlayer(n, sn, pl)
    {
        cout << "Constructor RankedTennisPlayer" << endl;
        this->rank = r;
    }

    RankedTennisPlayer(const TennisPlayer& t, unsigned int r = 0) : TennisPlayer(t) {
        this->rank = r;
    }

    ~RankedTennisPlayer() {
        cout << "Destructor RankedTennisPlayer\n" << endl;
    }

    friend ostream& operator<<(ostream& out, const RankedTennisPlayer &tp) {
        cout << "-- RANKED TENNIS PLAYER --\n";
        out << TennisPlayer (tp);
        out << "Rank: " << tp.rank << endl;
        return out;
    }
};

int main() {
    TennisPlayer rf("Roger", "Federer");
    TennisPlayer ng("Novak", "Djokovikj");
    cout << rf;
    cout << ng;
    //TennisPlayer t;
    RankedTennisPlayer rn("Rafael", "Nadal", true, 2750);
    cout << rn;
    TennisPlayer tp = rn;
    cout << tp;
    //RankedTennisPlayer copy(tp);
    RankedTennisPlayer copy(ng, 3320);
    cout << copy;
    return 0;
}
```

1.2. Accounts

Define a class for `DebitAccount` for working with debit bank account. For each bank account keep the name of the owner (char array max 100 chars), number of account (long number) and current balance (double). Enable methods for account review, deposit and withdrawing money from the account.

Then define a class `CreditAccount` that will enable the user to take loan from the bank. It should enable computing the interest if the user owns money to the bank.

Solution `oop_av72_en.cpp`

```
#include <iostream>
#include <cstring>
```

```
using namespace std;

class DebitAccount {
protected:
    char name[100];
    long number;
    double balance;
public:
    DebitAccount(const char *name = "----", const long number = 0,
                 const double balance = 0.0) {
        strncpy(this->name, name, 99);
        this->name[99] = 0;
        this->number = number;
        this->balance = balance;
    }
    void showInfo() {
        cout << name << '\n'
             << "\t Bank No: " << number << '\n'
             << "\t Balance: " << getBalance() << '\n';
    }

    void deposit(double amount) {
        if (amount >= 0 ) {
            balance += amount;
        }
        else {
            cout << "You can not add negative amount to your balance!" << endl;
        }
    }

    void withdraw(double amount) {
        if (amount < 0) {
            cout << "You can note withdraw negative amount from your account!" << endl;
            return;
        }

        if (amount <= balance) {
            balance -= amount;

        } else {
            cout << "You can not withdraw more money than you have on your account.\n"
                 << "Please upgrade your debut account to credit account!" << endl;
        }
    }

    double getBalance() {
        return this->balance;
    }

    ~DebitAccount() {}
};

class CreditAccount : public DebitAccount {
private:
    double limit;
    double interest; // % percent
    double minus;
public:
    CreditAccount(const char *name = "----", const long number = 0,
                  const double balance = 0, const double limit = 1000,
                  const double interest = 0.05, const double minus = 0):
        DebitAccount(name, number, balance) {

        this->limit = limit;
        this->interest = interest;
        this->minus = minus;
    }

    void withdraw(double amount) {
        int balance = getBalance();

        if (amount <= balance) {
            DebitAccount::withdraw(amount);
        }

        else if (amount <= balance + limit - minus) {
```

```
            double advance = amount - balance;
            this->minus += advance * (1.0 + interest);

            cout << "Minus: " << advance << "\n"
                 << "Minus with interest: " << advance*interest << endl;

            deposit(advance);
            DebitAccount::withdraw(amount);

        } else {
            cout << "The bank is not giving you that much money..." << endl;
            this->showInfo();
        }
    }

    void showInfo() {
        DebitAccount::showInfo();
        cout << "\t Limit: " << this->limit << "\n"
             << "\t In minus: " << this->minus << "\n"
             << "\t Interest: " << this->interest << "%\n";
    }


    double getInterest() {
        return this->interest;;
    }
};

int main() {
    DebitAccount d("Pero Perovski", 6, 100000);
    CreditAccount ca("Mitko Mitkovski", 10, 5000, 1000);
    d.showInfo();
    d.deposit(50000);
    d.withdraw(600000);
    d.showInfo();
    ca.showInfo();
    ca.deposit(500);
    ca.showInfo();
    ca.withdraw(6200);
    ca.showInfo();
    return 0;
}
```

2. Source code of the examples and problems

https://github.com/finki-mk/SP/

Source code ZIP