Hemtenta

Filip Wilhelm Sjöstrand

2023-08-25

Uppgift 1 – Husdjur

A)

Klassdefinition (implementation).

Klass definitionen berättar för oss vad en klass kan utföra. Den innehåller de attribut och metoder som tillhör en klass. Inom klassens ramar kan vi definiera ifall dessa funktioner och variabler skall vara publika eller privata.

```
class HusDjur {
private:
    string sort;
public:
    string getSort;
    void setSort;
};
```

Överlagrad konstruktor.

Den överlagrad konstruktorn är en ”kopia” på standardkonstruktorn där man kan välja att ange värden på specifika attributd för ett objekt vid initieirng.

```
HusDjur::HusDjur() {
    namn = "namnlös";
}

HusDjur::HusDjur(string n) {
    namn = n;
}
```

Statisk array av objekt.

Vid initiering av en array kan du välja att definiera antalet element som den skall innehålla, det vill säga att de är statiska, dess storlekt förändras sig då inte efteråt.

```
HusDjur arr[4];
```

Dynamisk array av objekt.

Till skillnad från en statisk array så behöver vi inte fördefiniera antalet element som skall ingå. Vi kan lägga till eller ta bort element allt efter programmets körning.

```
int n = 4;
HusDjur* arr = new HusDjur[n];
delete[] djurArray;
```

Initieringslista.

Kan användas för att initiera attributen hos ett objekt. Ett effektivare sätt att skriva t.e.x konstruktorer på (jämför med ovanstående överlagrade kosntruktor), när man har många attribut som skall initieras.

```
HusDjur::HusDjur(string n) : namn(n) {}
```

B)

Antag att getters och setters finns för alla attribut.

```
void HusDjur::unge(const HusDjur &m, const HusDjur &p) {
    string sort = m.getSort();
    string namn = m.getNamn() + p.getNamn();
    double medelVarde = (m.getLangd() + p.getLangd()) / 2.0;
    double langd = medelVarde / 2.0;
    bool hona;

    if (m.getLangd() < p.getLangd()) {
        hona = false;
    } else if (m.getLangd() > p.getLangd()) {
        hona = true;
    } else {
        hona = false; // Antar att ni ej menar att det skall vara slumpmässigt.
    }

    setSort(sort);
    setNamn(namn);
    setLangd(langd);
    setHona(hona);
}
```

Uppgift 2 – Telefonlista

Antar att när man adderar en ny person så har hen ett unikt namn.

`sok()` börjar med att öppna filen. Sedan läser den filen rad för rad. För varje rad jämförs innehållet med en angiven sträng. Om det inte finns en match så kommer `rad.find(str)` att returnerar string::npos (alltså att det saknas positioner för en match) och alltså kommer vi inte att gå in i if-loopen. Vid match gör vi det och skriver ut den raden. Sedan stänger vi filen.

`laggIn()` börjar likt `sok()` och läser in en fil. Sedan hämtar den rad för rad innehållet. Varje rad jämförs med det givna nya namnet. `rad.compare(namn)` returnerar 0 om namnen är samma, <0 om det givna

namnet är i bokstavsordning efter, och >0 om det är före. Tillagd håller koll på om namnet är adderat eller inte. Om personen på den inlästa raden kommer före den nya så skrivs hen in i den nya utfilen. Om inte så kommer den nya personen först skrivas in och sedan den inlästa raden.

```
void sok(const string& fil, const string& str) {
    ifstream infil(fil);
    string rad;

    while (getline(infil, rad)) {
        if (rad.find(str) != string::npos) {
            cout << rad << endl;
        }
    }

    infil.close();
}
```

```
void laggIn(const string& infil, const string& utfil, const string& namn, const string& number) {
    ifstream is(infil);
    ofstream os(utfil);
    string rad;
    bool tillagd = false;

    while (getline(is, rad)) {
        if (!tillagd && rad.compare(namn) > 0) {
            os << namn + "\t" + number << endl;
            tillagd = true;
        }
        os << rad << endl;
    }

    // Om nya namnet skall adderas på sista raden
    if (!tillagd) {
        os << namn + "\t" + number << endl;
    }

    is.close();
    os.close();
}
```

Uppgift 3 - Garnaffären

`sok()` använder sig av samma teknik som in föregående fråga, alltså om `beskrivning.find(nyckel)` inte returnerar `string::npos` så finns en match och vi returnerar `true`.

Defaultkonstruktorn för `GarnAffar`, initiarar `antal` till 0 och sätter `garner` arrayen till nollpekare.

`laggTill(Garn nytt)` börjar med att initiera en temporär array, `tempArr` som har ett mer element än `garner`. Sedan kopierar vi alla element från `garner` till `tempArr` och tömmer `garner` från minnet. Sedan adderar vi det nya tillägget av garner, `nytt`, till `tempArr` och kopierar allting till `garner` igen och utökar `antal` med 1.

`hitta(string nyckel[], int n_nycklar)` tar in en array av nycklar att jämföras med och hur många de är. Sedan iterar vi igenom alla garn so finns i `garner`. Varje garn jämförs med nycklarna i `nyckel[]` och om

inte alla nycklar returnerar en match så gör vi ingenting och avslutar loopen. Vid fullständig match hämtar vi mängden av detta garn och adderar till resultat.

```
#include <iostream>
#include <fstream>
#include <string>
#include <iomanip>
using namespace std;

class Garn
{
private:
    string beskrivning;
    double pris;
    int antal;
public:
    Garn();
    Garn( string beskr, double pris, int antal );
    string geBeskrivning();
    double gePris();
    int geAntal();
    bool sok( const string& nyckel );
};

class GarnAffar
{
private:
    Garn *garner;
    int antal;
public:
    GarnAffar();
    ~GarnAffar();
    void laggTill(Garn nytt);
    int hitta(string nyckel[], int n_nycklar);
};

int main() {

    //Några Garn-objekt skapas

    Garn alpacka = Garn("grått garn av sorten alpacka, materialet är alpacka-ull",35.00,30);
    Garn g1("akrylgarn, gult", 20.50, 50 );
    Garn g2("gult ullgarn, tvåtrådigt", 35.00, 10 );


    GarnAffar stickan;  //En garnaffär som heter stickan
    stickan.laggTill( alpacka );
    stickan.laggTill(g1);
    stickan.laggTill ( g2 );


    // Här anropas metoden hitta med nyckeln "ull", nyckel-arrayen
    // är ett (1) element lång.
    string n1[] = {"ull"};
```

```
    cout << "Det finns " << stickan.hitta( n1,1) << " nystan med ull. " << endl;


    // Här anropas metoden hitta med nyckeln "akryl","gult", nyckel-arrayen är två
    // (2) element lång.
    string n[] = {"akryl","gult"};
    cout << "Det finns " << stickan.hitta(n,2) << " nystan akryl och gult. " << endl;


    return 0;
}

Garn::Garn():beskrivning(""),pris(0.0),antal(0){}

Garn::Garn( string beskr, double pris, int antal )
:beskrivning(beskr),pris(pris),antal(antal){}

string Garn::geBeskrivning(){
    return beskrivning;
}

double Garn::gePris(){
    return pris;
}

int Garn::geAntal(){
    return antal;
}

bool Garn::sok( const string& nyckel ) {
    if (beskrivning.find(nyckel) != string::npos) {
        return true;
    }
    return false;
}


GarnAffar::GarnAffar(): garner(0), antal(0) {}

GarnAffar::~GarnAffar() {
    delete[] garner;
}

void GarnAffar::laggTill(Garn nytt) {
    Garn *tempArr = new Garn[antal + 1];

    for (int i = 0; i < antal; i++) {
        tempArr[i] = garner[i];
    }

    tempArr[antal] = nytt;

    delete[] garner;
```

```
    garner = tempArr;
    antal++;
}

int GarnAffar::hitta(string nyckel[], int n_nycklar) {
    int resultat = 0;

    for (int i = 0; i < antal; i++) {
        bool match = true;
        Garn ettGarn = garner[i];

        for (int j = 0; j < n_nycklar; j++) {
            string jamforsMed = nyckel[j];

            if (!ettGarn.sok(jamforsMed)) {
                match = false;
                break;
            }
        }

        if (match) {
            resultat += ettGarn.geAntal();
        }

    }

    return resultat;
}
```

Uppgift 4 - Anställning

klassen `Anstalning` har två atribut som pekar till ett Person och ett Arbetsgivar-objekt. Dess länkar samman anställda med deras företag. inom klassen finns även getters för att hämta namnet på en person och dess arbetsgivare.

klassen `AnstallningsLista` är en samling av anställningar. Den har tre huvudmetoder. `laggIn(const Anstallning& anstallning)` tar ett nytt anställningsobjekt som skall aderas till listan. Detta görs genom att skapa en temporär array som har en storlek större än `anstallningar`. sedan kopierar vi över hela `anstallningar`, adderar den nya `anstallning` och tömmer `anstallningar` på minne. Kopierar den temporära arrayen till `anstallningar` och ökar antal med 1.

`listArbetsgivare(ostream& os, Person* p)` och `listAnstallda(ostream& os, Arbetsgivare* a)` är väldigt lika. För den första metoden så itererar vi igenom alla anställningar, jämför alla namnen för dessa med det givna. Vid match, skriv ut personens arbetsgivare. För den andra metoden har vi samma approach men jämför med arbestfivare istället och vid match skriver ut alla dess anställda.

```
#include <iostream>
#include <fstream>
#include <string>
#include <iomanip>
using namespace std;
```

```
class Person {
public:
    Person( const string& name ) : name(name) {}
    string namnet() const { return name; }
private:
    string name;
};

class Arbetsgivare {
public:
    Arbetsgivare( const string& name) : name( name) {}
    string namnet() const { return name; }
private:
    string name;
};

class Anstallning {
private:
    Person* person;
    Arbetsgivare* arbetsgivare;
public:
    Anstallning();
    Anstallning(Person* p, Arbetsgivare* a);
    string getPerson() const;
    string getArbetsgivare() const;
};

class AnstallningsLista {
private:
    int antal;
    Anstallning** anstallningar;
public:
    AnstallningsLista();
    ~AnstallningsLista();
    void laggIn(const Anstallning& anstallning);
    void listArbetsgivare(ostream& os, Person* p);
    void listAnstallda(ostream& os, Arbetsgivare* a);
};


int main(){

    AnstallningsLista anst;
    Person p1("Kalle");
    Person p2("Pelle");
    Person p3("Olle");
    Person p4("Eva");
    Person p5("Stina");
    Arbetsgivare a1( "Uppsala universitet");
    Arbetsgivare a2( "Uppsala kommun");
    Arbetsgivare a3( "Ericsson");
    Arbetsgivare a4( "Microsoft");
```

```
    anst.laggIn( Anstallning(&p1,&a1) ); // p1 arbetar hos a1
    anst.laggIn( Anstallning(&p2,&a1) );
    anst.laggIn( Anstallning(&p3,&a2) );
    anst.laggIn( Anstallning(&p1,&a3) );
    anst.laggIn( Anstallning(&p3,&a3) );

    anst.listArbetsgivare( cout, &p1);  // lista alla Kalles arbetsgivare
    anst.listAnstallda( cout, &a3);     // lista alla Ericsson anställda

    return 0;
}


Anstallning::Anstallning() : person(0), arbetsgivare(0) {}

Anstallning::Anstallning(Person* p, Arbetsgivare* a) : person(p), arbetsgivare(a) {}

string Anstallning::getPerson() const {
    return person->namnet();
}

string Anstallning::getArbetsgivare() const {
    return arbetsgivare->namnet();
}


AnstallningsLista::AnstallningsLista(): antal(0), anstallningar(0) {}

AnstallningsLista::~AnstallningsLista() {
    for (int i = 0; i < antal; i++) {
        delete anstallningar[i];
    }
    delete[] anstallningar;
}

void AnstallningsLista::laggIn(const Anstallning& anstallning) {
    Anstallning** tempArr = new Anstallning*[antal + 1];
    for (int i = 0; i < antal; ++i) {
        tempArr[i] = anstallningar[i];
    }
    tempArr[antal] = new Anstallning(anstallning);

    delete[] anstallningar;

    anstallningar = tempArr;
    antal++;
}

void AnstallningsLista::listArbetsgivare(ostream &os, Person* p) {
    for (int i = 0; i < antal; i++) {
        if (anstallningar[i]->getPerson() == p->namnet()) {
            os << p->namnet() << " jobbar hos " << anstallningar[i]->getArbetsgivare() << endl;
        }
```

```
    }
}


void AnstallningsLista::listAnstallda(ostream& os, Arbetsgivare* a) {


    for (int i = 0; i < antal; i ++) {
        if (anstallningar[i]->getArbetsgivare() == a->namnet()) {
            os << a->namnet() << " har " << anstallningar[i]->getPerson() << " anställd" << endl;;
        }
    }

}
```