# Zadatak 2

*Implementirajte konzolnu aplikaciju sa sljedećim funkcionalnostima*

|  |
| --- |
| 1. Dodaj korisnika 2. Dodaj racun 3. Dodaj uplatu 4. Dodaj isplatu 5. Stanje na racunu 6. Pretraga po min. iznosu transakcije 7. Pretraga po datumu transakcije 8. Deaktiviranje korisnika 9. Kraj |

*Implementirajte sljedeće strukture (u odvojene header fajlove)*

|  |
| --- |
| 1. struct Datum    1. int \_dan;    2. int \_mjesec;    3. int \_godina;    4. static Datum Napravi(int d, int m, int g) ;    5. void Ispis(); 2. enum VrstaTransakcije    1. UPLATA    2. ISPLATA 3. struct Transakcija    1. Datum \_datumTransakcije;    2. char \_TO[20];    3. float \_iznos;    4. VrstaTransakcije \_vrstaTransakcije;    5. bool \_aktivna;    6. static Transakcija NapraviUplatu(float iznos, Datum datum) ;    7. static Transakcija NapraviIsplatu(float iznos, char\* to, Datum datum) ;    8. void Ispis(); 4. struct Kolekcija   *(typedef Transakcija Tip; preuzeti rješenje zadataka br1)*   1. enum VrstaRacuna    1. TEKUCI    2. DEVIZNI    3. STEDNI 2. struct Racun    1. VrstaRacuna \_vrstaRacuna;    2. char \_brojRacuna[20];    3. Kolekcija transakcije;    4. bool \_aktivan;    5. static Racun\* Napravi(VrstaRacuna vrsta\_racuna, char\* broj\_racuna) ;    6. void Ispis();    7. void Uplata(float iznos, Datum datum) ;    8. void Isplata(float iznos, char\* to, Datum datum) ;    9. float Saldo() //stanje na računu: suma uplata minus suma isplata    10. Kolekcija GetTransakcijeByIznos(float iznos) ;    11. Kolekcija GetTransakcijeByDatum(Datum& datum) ; 3. struct Klijent    1. int maxR = 10;    2. char \_JMBG[14];    3. char \* \_imePrezime;    4. Racun \*\* \_racuni = new Racun\*[maxR];    5. int \_trenutnoRacuna;    6. bool \_aktivan;    7. static Klijent\* Napravi(char \* jmbg, char \* ime\_prezime) ;    8. void Ispis();    9. void Dealociraj();    10. void DodajRacun(VrstaRacuna vrstaRacuna, char\* brojRacuna) ;    11. void DodajUplatu(int indexRacuna, float iznos, Datum datum) ;    12. void DodajIsplatu(int indexRacuna, float iznos, char\* to, Datum datum) ;    13. Kolekcija GetTransakcijeByIznos(float iznos) ;// vraca Kolekciju transakcija koje su imale iznos veci od trazenog    14. Kolekcija GetTransakcijeByDatum(Datum datum) ;// vraca transakcija koje su obavljene za trazeni datum    15. void Deaktiviraj(); 4. struct Banka    1. int maxK = 100;    2. int trenutnoKlijenata = 0;    3. Klijent \*\* klijenti = new Klijent\*[maxK];    4. void DodajKlijenta(char\* jmbg, char\* ime\_prezime) ;    5. void Dealociraj(); |

|  |
| --- |
| Rješenje: Datum.h |
| #pragma once  #include <iostream>  using namespace std;  struct Datum  {  int \_dan, \_mjesec, \_godina;    static Datum Napravi(int d, int m, int g)  {  Datum datum;  datum.\_dan = d;  datum.\_mjesec = m;  datum.\_godina = g;  return datum;  }  bool jednako(Datum& drugi)  {  return  (\_dan == drugi.\_dan) &&  (\_mjesec == drugi.\_mjesec) &&  (\_godina == drugi.\_godina);  }  void Ispis()  {  cout << "Datum: " << \_dan << "." << \_mjesec << "." << \_godina << endl;  }  }; |

|  |
| --- |
| Rješenje: Transakcija.h |
| #pragma once  #include <iostream>  using namespace std;  #include "Datum.h"  enum VrstaTransakcije { UPLATA, ISPLATA };  char \*vrstaTransakcije[] = { "UPLATA", "ISPLATA" };  struct Transakcija  {  Datum \_datumTransakcije;  char \_TO[20];  float \_iznos;  VrstaTransakcije \_vrstaTransakcije;  /\* ako je isplata vrijednost \_TO postaviti na brojRacuna sa koga se transakcija vrsi\*/  bool \_aktivna;  /\*u slucaju da se transakcija izbrise ona postaje neaktivna, ali je bitno da o njoj ostane zapis\*/  void Ispis()  {  if (\_aktivna)  {  cout << "Vrsta transakcije: " << \_vrstaTransakcije << endl;  cout << "Iznos: " << \_iznos << endl;  cout << "To: " << \_TO << endl;  }  else  {  cout << "Transakcija je obrisana" << endl;  }  }  static Transakcija NapraviUplatu(float iznos, const Datum& datum)  {  Transakcija t;  t.\_vrstaTransakcije = UPLATA;  strcpy\_s(t.\_TO, "-");  t.\_iznos = iznos;  t.\_aktivna = true;  t.\_datumTransakcije = datum;  return t;  }  static Transakcija NapraviIsplatu(float iznos, char\* to, Datum& datum)  {  Transakcija t;  t.\_vrstaTransakcije = ISPLATA;  strcpy\_s(t.\_TO, to);  t.\_iznos = iznos;  t.\_aktivna = true;  t.\_datumTransakcije = datum;  return t;  }  }; |

|  |
| --- |
| Rješenje: Kolekcija.h |
| #pragma once  #include <iostream>  #include <functional>  using namespace std;  typedef int Tip;  const int max\_size = 30;  struct Kolekcija  {  Tip podaci[max\_size];  int brojac = 0;  };  void kDodaj(Kolekcija& k, Tip newV)  {  //dodaj newV u podaci na poziciju brojac  //uvecati brojac  k.podaci[k.brojac++] = newV;  }  Kolekcija kFilter(Kolekcija& k, function<bool(Tip&)> f)  {  Kolekcija newK;  for (size\_t i = 0; i < k.brojac; i++)  {  Tip &x = k.podaci[i];  if (f(x)) {  kDodaj(newK, x);  }  }  //iz kolekcije k u kolekciju newK kopirati elemente koji zadovoljavaju uslov f  return newK;  }  void kForEach(Kolekcija& k, function<void(Tip&)> f)  {  //za svaki element iz kolekcije k izvrsi zadatak f  for (int i = 0; i < k.brojac; i++)  {  Tip &x = k.podaci[i];  f(x);  }  } |

|  |
| --- |
| Rješenje: Račun.h |
| #pragma once  #include <iostream>  using namespace std;  #include "Kolekcija.h"  enum VrstaRacuna { TEKUCI, DEVIZNI, STEDNI };  char \*vrstaRacuna[] = { "TEKUCI", "DEVIZNI", "STEDNI" };  struct Racun  {  VrstaRacuna \_vrstaRacuna;  char \_brojRacuna[20];  bool \_aktivan;  Kolekcija transakcije;    void Ispis()  {  cout << "Vrsta računa: " << \_vrstaRacuna << endl;  cout << "Broj računa: " << \_brojRacuna << endl;  cout << "Broj transakcija: " << transakcije.brojac << endl;  }  void Dealociraj()  {  }  void Deaktiviraj()  {  \_aktivan = false;  }  float Saldo()  {  float s = 0;  transakcije.ForEach([&s](Transakcija& t)  {  if (t.\_vrstaTransakcije == UPLATA)  s = t.\_iznos;  if (t.\_vrstaTransakcije == ISPLATA)  s = -t.\_iznos;  });  return s;  }  void Uplata(float iznos, Datum datum)  {  Transakcija new\_v = Transakcija::NapraviUplatu(iznos, datum);  transakcije.Dodaj(new\_v);  }  void Isplata(float iznos, char \*to, Datum datum)  {  Transakcija new\_v = Transakcija::NapraviIsplatu(iznos, to, datum);  transakcije.Dodaj(new\_v);  }  Kolekcija GetTransakcijeByIznos(float iznos)  {  Kolekcija k = transakcije.Filter([&iznos](Tip&t) {return t.\_iznos >= iznos; });  return k;  }  Kolekcija GetTransakcijeByDatum(Datum& datum)  {  Kolekcija k = transakcije.Filter([&datum](Tip&t) {  return t.\_datumTransakcije.jednako(datum); });    return k;  }  static Racun\* Napravi(VrstaRacuna vrsta\_racuna, char\* broj\_racuna)  {  Racun\* r = new Racun;    r->\_vrstaRacuna = vrsta\_racuna;  strncpy\_s(r->\_brojRacuna, broj\_racuna,19);  return r;  }  }; |

|  |
| --- |
| Rješenje: Klijent.h |
| #pragma once  #include <iostream>  using namespace std;  #include "Racun.h"  struct Klijent  {  char \_JMBG[14];  char \* \_imePrezime;  int maxR = 10;  Racun \*\* \_racuni = new Racun\*[maxR];  int \_trenutnoRacuna = 0; //brojac  bool \_aktivan;  static Klijent\* Napravi(char \* jmbg, char \* ime\_prezime)  {  Klijent\* k = new Klijent;  int x = strlen(ime\_prezime) + 1;  k->\_imePrezime = new char[x];  strcpy\_s(k->\_imePrezime, x, ime\_prezime);  strcpy\_s(k->\_JMBG, jmbg);  return k;  }  void Ispis()  {  cout << "KLIJENT jmbg: " << \_JMBG <<  ", Ime i prezime: " << \_imePrezime <<  ", Broj racuna: " << this->\_trenutnoRacuna << endl;  }  void Dealociraj()  {  delete \_imePrezime;  delete[] \_racuni;  }  void DodajRacun(VrstaRacuna vrstaRacuna, char\* brojRacuna)  {  \_racuni[\_trenutnoRacuna++] = Racun::Napravi(vrstaRacuna, brojRacuna);  }  void DodajUplatu(int indexRacuna, float iznos, Datum datum)  {  \_racuni[indexRacuna]->Uplata(iznos, datum);  }  void DodajIsplatu(int indexRacuna, float iznos, char\* to, Datum datum)  {  \_racuni[indexRacuna]->Isplata(iznos, to, datum);  }  Kolekcija GetTransakcijeByIznos(float iznos)  {  Kolekcija zbirnaKolekcija;  for (int i = 0; i < \_trenutnoRacuna; i++)  {  zbirnaKolekcija.Dodaj(\_racuni[i]->GetTransakcijeByIznos(iznos));  }  return zbirnaKolekcija;  }  Kolekcija GetTransakcijeByDatum(Datum datum)  {  Kolekcija zbirnaKolekcija;  for (int i = 0; i < \_trenutnoRacuna; i++)  {  zbirnaKolekcija.Dodaj(\_racuni[i]->GetTransakcijeByDatum(datum));  }  return zbirnaKolekcija;  }  void Deaktiviraj()  {  \_aktivan = false;  }  /\*  Unos / Ispis / Dealociraj / DodajRacun / Deaktiviraj /  DodajUplatu  DodajIsplatu  GetTransakcijeByIznos - vraca Kolekciju transakcija koje su imale iznos veci od trazenog  GetTransakcijeByDatum - vraca Kolekciju transakcija koje su obavljene za trazeni datum  \*/  }; |

|  |
| --- |
| Rješenje: Banka.h |
| #pragma once  #include "Klijent.h"  struct Banka  {  int maxK = 100;  int trenutnoKlijenata= 0;  Klijent \*\* klijenti = new Klijent\*[maxK];  void DodajKlijenta(char\* jmbg, char\* ime\_prezime)  {  klijenti[trenutnoKlijenata++] = Klijent::Napravi(jmbg, ime\_prezime);  }  void Dealociraj()  {  for (int i = 0; i < trenutnoKlijenata; i++)  {  klijenti[i]->Dealociraj();  }  }  }; |

|  |
| --- |
| Rješenje: Source.cpp |
| #include <iostream>  #include "Banka.h"  using namespace std;  char \* crt = "\n---------------------------\n";  int PrikaziMenu()  {  int izbor;  cout << crt << "\t::MENU::" << crt;  cout << "1. Dodaj korisnika" << endl;  cout << "2. Dodaj racun" << endl;  cout << "3. Dodaj Uplatu" << endl;  cout << "4. Dodaj Isplatu" << endl;  cout << "5. Stanje na računu" << endl;  cout << "6. Pretraga po min. iznosu transakcije" << endl;  cout << "7. Pretraga po datumu transakcije" << endl;  cout << "8. Deaktiviranje korisnika" << crt; // racuna ili korisnika  cout << "9. Kraj" << crt;  do  {  cout << "Vas izbor -> ";  cin >> izbor;  } while (izbor < 1 || izbor>9);  system("cls");  return izbor;  }  Klijent\* IzborKlijenta(Banka &B)  {  for (size\_t i = 0; i < B.trenutnoKlijenata; i++)  cout << i + 1 << ". " << B.klijenti[i]->\_imePrezime << endl;  int odabir;  cout << "Unesite id korisnika: ";  while (true)  {  cin >> odabir;  if (odabir > 0 && odabir <= B.trenutnoKlijenata)  break;  cout << "Los unos!!!" << endl;  }  odabir--;  Klijent\* odabrani = B.klijenti[odabir];  if (odabrani != nullptr)  {  cout << "Odabran klijent ---> ";  odabrani->Ispis();  }  return odabrani;  }  Racun\* IzborRacuna(Klijent &K)  {  for (int i = 0; i < K.\_trenutnoRacuna; i++)  {  cout << i + 1 << ": ";  K.\_racuni[i]->Ispis();  }  int x;  cout << "Unesite RB: ";  cin >> x;  return K.\_racuni[x-1];  }  void main()  {  Banka B1;  int izborMenu;  do  {  izborMenu = PrikaziMenu();  if (izborMenu == 1)  {  //Dodaj korisnika  cin.ignore();  cout << "Unesite jmbg"<<endl;  char jmbg[14];  cin.getline(jmbg, 14);  cout << "Unesite ime i prezime" << endl;  char ime\_prezime[30];  cin.getline(ime\_prezime, 30);  B1.DodajKlijenta(jmbg, ime\_prezime);  }  if (izborMenu == 2)  {  //Dodaj racun  Klijent\* k = IzborKlijenta(B1);  cin.ignore();  cout << "Unesite broj racuna" << endl;  char racun[13];  cin.getline(racun, 13);  k->DodajRacun(DEVIZNI, racun);  }  if (izborMenu == 3)  {  //Uplata  Klijent\* k = IzborKlijenta(B1);  Racun\* r = IzborRacuna(\*k);  cout << "Unesite iznos uplate " << endl;  float iznos;  cin >> iznos;  cout << "Unesite datum uplate" << endl;  Datum date;  cin >> date.\_dan >> date.\_mjesec >> date.\_godina;  r->Uplata(iznos, date);  }  if (izborMenu == 4)  {  //Isplata  Klijent\* k = IzborKlijenta(B1);  Racun\* r = IzborRacuna(\*k);  cout << "Unesite iznos isplate " << endl;  float iznos;  cin >> iznos;  cout << "Unesite datum isplate" << endl;  Datum date;  cin >> date.\_dan >> date.\_mjesec >> date.\_godina;  r->Isplata(iznos, "??", date);  }  if (izborMenu == 5)  {  //Stanje  Klijent\* k = IzborKlijenta(B1);  Racun\* r = IzborRacuna(\*k);  cout << "Stanje na racunu je " << r->Saldo();  }  if (izborMenu == 6)  {  //Pretraga po min. iznosu  Klijent\* k = IzborKlijenta(B1);  cout << "Pretraga: Unesite min. iznos transakcije" << endl;  float f;  cin >> f;  Kolekcija transakcije = k->GetTransakcijeByIznos(f);  transakcije.ForEach([](Transakcija& t) { t.Ispis(); });  }  if (izborMenu == 7)  {  //Pretraga po datumu  Klijent\* k = IzborKlijenta(B1);  cout << "Pretraga: Unesite datum transakcije" << endl;  cout << "Unesite datum isplate" << endl;  Datum date;  cin >> date.\_dan >> date.\_mjesec >> date.\_godina;  Kolekcija transakcije = k->GetTransakcijeByDatum(date);  transakcije.ForEach([](Transakcija& t) { t.Ispis(); });  }  if(izborMenu == 8)  {  cout << "Deaktiviranje klijenta: Izaberite klijenta" << endl;  Klijent\* k = IzborKlijenta(B1);  k->Deaktiviraj();  }  } while (izborMenu != 9);  B1.Dealociraj();  } |