

## Könyv.hpp

```
1 #pragma once
2 #include "konyvtar.h"
3 #include "konyv.h"
4 #include "kolcson.h"
5 #include "mufaj.h"
6 #include "szemely.h"
7 #include <string>
8 #include <iostream>
9 class Kolcson;
10 class Konyvtar;
11 class Mufaj;
12 class Szemely;
13
14 using namespace std;
15
16 class Konyv{
17 private:
18     int azon, oldal;
19     string cim, szerzo;
20     bool kinn;
21     Mufaj*mufaj;
22     Kolcson*fej;
23
24 public:
25     Konyv()=default;
26     Konyv(string cim, string szerzo,int oldal, Mufaj* mufaj):azon(0), szerzo(szerzo),oldal(oldal),mufaj(mufaj),cim(cim), fej(0),kinn(false){}
27     int Dij(int ma);
28     ~Konyv();
29
30     void set_azon(int azon){this->azon=azon;}
31     int get_azon(){ return azon;}
32     bool get_kinn(){ return kinn;}
33     void set_kinn(bool k) {kinn=k;}
34     Kolcson* get_fej(){ return fej;}
35     void set_fej(Kolcson*k){fej=k;}
36     string get_cim(){return cim;}
37     string get_szerzo(){return szerzo;}
38     int get_oldal(){return oldal;}
39     Mufaj* get_mufaj(){return mufaj;}
40 }
```

## Könyv.cpp

```
#include "konyv.h"
int Konyv::Dij(int ma)
{
    int keses= ma -(fej->get_datum() + 30);
    if(keses>0)
    {
        return keses*mufaj->Dij();
    }
    return 0;
}

Konyv::~~Konyv(){
    delete mufaj;
}
```

## Műfaj.hpp

```
1 #pragma once
2 #include "konyvtar.h"
3 #include "konyv.h"
4 #include "kolcson.h"
5 #include "mufaj.h"
6 #include "szemely.h"
7 #include <string>
8 #include <iostream>
9
10 using namespace std;
11
12 class Kolcson;
13 class Konyvtar;
14 class Konyv;
15 class Szemely;
16
17 class Mufaj{
18
19 public:
20     virtual int Dij()=0;
21     virtual ~Mufaj()=default;
22 };
23
```

```

24 class Termesztudomanyos : public Mufaj{
25     private:
26         static Termesztudomanyos* _instance;
27         Termesztudomanyos(){}
28         Kolcson* fej;
29     public:
30         static Termesztudomanyos* instance(){
31             if(_instance==nullptr)
32             {
33                 _instance= new Termesztudomanyos();
34             }
35             return _instance;
36         }
37         static void destroy(){
38             if(_instance!=nullptr)
39             {
40                 delete _instance;
41             }
42             _instance=nullptr;
43         }
44         int Dij() override{
45             return 100;
46         }
47     };
48
49 class Szepirodalmi : public Mufaj{
50     private:
51         static Szepirodalmi* _instance;
52         Szepirodalmi(){}
53     public:
54         static Szepirodalmi* instance(){
55             if(_instance==nullptr)
56             {
57                 _instance= new Szepirodalmi();
58             }
59             return _instance;
60         }
61         static void destroy(){
62             if(_instance!=nullptr)
63             {
64                 delete _instance;
65             }
66             _instance=nullptr;
67         }
68         int Dij() override{
69             return 50;
70         }
71     };
72
73
74 class Ifjusagi : public Mufaj{
75     private:
76         static Ifjusagi* _instance;
77         Ifjusagi(){}
78         Kolcson* fej;
79     public:
80         static Ifjusagi* instance(){
81             if(_instance==nullptr)
82             {
83                 _instance= new Ifjusagi();
84             }
85             return _instance;
86         }
87         static void destroy(){
88             if(_instance!=nullptr)
89             {
90                 delete _instance;
91             }
92             _instance=nullptr;
93         }
94         int Dij() override{
95             return 20;
96         }
97     };

```

## Műfaj.cpp

```
1 #include "mufaj.h"
2
3 Termesztudomanyos* Termesztudomanyos::_instance=nullptr;
4 Ifjusagi* Ifjusagi::_instance=nullptr;
5 Szepirodalmi* Szepirodalmi::_instance=nullptr;
```

## Kölcsön.hpp

```
1 #pragma once
2 #include "konyvtar.h"
3 #include "konyv.h"
4 #include "kolcson.h"
5 #include "mufaj.h"
6 #include "szemely.h"
7 #include <string>
8 #include <iostream>
9 #include <vector>
10
11 using namespace std;
12 class Mufaj;
13 class Konyvtar;
14 class Konyv;
15 class Szemely;
16
17 class Kolcson{
18     private:
19         int datum;
20         Konyvtar* konyvtar;
21         Szemely* tag;
22         vector<Konyv*> tetelek;
23
24     public:
25         Kolcson(Konyvtar* k, Szemely* sz, int d): datum(d),konyvtar(k),tag(sz){}
26         ~Kolcson() {}
27
28         //*****
29
30         vector<Konyv*> &get_tetelek(){ return tetelek; }
31         int get_datum(){return datum;}
32
33 };
34
```

## Személy.hpp

```
1 #pragma once
2 #include "konyvtar.h"
3 #include "konyv.h"
4 #include "kolcson.h"
5 #include "mufaj.h"
6 #include "szemely.h"
7 #include <string>
8 #include <iostream>
9 #include <vector>
10
11 using namespace std;
12 class Kolcson;
13 class Konyvtar;
14 class Mufaj;
15 class Konyv;
16
17
18
19 class Szemely{
20     private:
21         string nev;
22         vector<Kolcson*> kolcs;
23         Konyvtar* konyvtar;
24
25     public:
26         Szemely(string n):nev(n){}
27         void Rogzit(Kolcson*k);
28
29
30         void set_konyvtar(Konyvtar* k){ konyvtar = k;}
31         string get_nev(){ return nev;}
32         vector<Kolcson*> get_kolcs(){return kolcs;}
33         Konyvtar* get_konyvtar(){return konyvtar;}
34
35 };
36
```

## Személy.cpp

```
1  #include "szemely.h"
2
3  void Szemely::Rogzit(Kolcson*k) {
4      kolcs.push_back(k);
5  }
```

## Könyvtár.hpp

```
1  #pragma once
2  #include "konyvtar.h"
3  #include "konyv.h"
4  #include "kolcson.h"
5  #include "mufaj.h"
6  #include "szemely.h"
7  #include <string>
8  #include <iostream>
9  #include <vector>
10 class Mufaj;
11 class Kolcson;
12 class Konyv;
13 class Szemely;
14
15 using namespace std;
16
17 class Konyvtar{
18
19     private:
20         vector<Kolcson*> kolcs;
21         vector<Konyv*> konyvek;
22         vector<Szemely*> tagok;
23         int c;
24     public:
25         Konyvtar():c(1){}
26         void Bevetelez(Konyv* k);
27         void Belep(Szemely* &sz);
28         bool Tag(Szemely* sz);
29         bool Keres(int az, Konyv* &k);
30         void Kolcsonoz(Szemely* &sz, vector<int> lista, int ma);
31
32
33
34         vector<Konyv*> get_konyvek(){return konyvek;}
35         vector<Kolcson*> get_kolcs(){return kolcs;}
36         vector<Szemely*> get_tagok(){return tagok;}
37
38
39
40     };
```

## Könyvtár.cpp

```
1  #include "konyvtar.h"
2
3  void Konyvtar::Bevetelez(Konyv* k)
4  {
5      k->set_azon(c);
6      c++;
7      konyvek.push_back(k);
8  }
9
10
11 void Konyvtar::Belep(Szemely* &sz){
12     if(!Tag(sz))
13     {
14         tagok.push_back(sz);
15         sz->set_konyvtar(this);
16     }
17 }
18
19
20 bool Konyvtar::Tag(Szemely* sz)
21 {
22     for(Szemely* e : tagok)
23     {
24         if(e->get_nev()==sz->get_nev())
25         {
26             return true;
27         }
28     }
29     return false;
30 }
31
32
33 bool Konyvtar::Keres(int az,Konyv* &k)
34 {
35     bool van=false;
36     for(Konyv* e : konyvek)
37     {
38         if(e->get_azon()==az)
39         {
40             k=e;
41             van=true;
42             return true;
43         }
44     }
45     return van;
46 }
47
48 void Konyvtar::Kolcsonoz(Szemely* &sz, vector<int> lista, int ma)
49 {
50     if(!Tag(sz) && lista.size()>5) { throw exception();}
51     Kolcson* kg = new Kolcson(this,sz,ma);
52     Konyv* k;
53
54     for(int az : lista)
55     {
56         bool van=false;
57         van=this->Keres(az,k);
58         if(van && !k->get_kinn())
59         {
60             kg->get_tetelek().push_back(k);
61             k->set_kinn(true);
62             k->set_fej(kg);
63         }
64     }
65     kolcs.push_back(kg);
66     sz->Rogzit(kg);
67 }
68 }
```

## main.cpp

```
2  #include "konyvtar.h"
3  #include "konyv.h"
4  #include "kolcson.h"
5  #include "mufaj.h"
6  #include "szemely.h"
7
8  #include <iostream>
9
10
11  int testCounter = 1;
12  bool jo = true;
13
14  void check(bool l)
15  {
16      if(!l)
17      {
18          jo = false;
19          std::cerr<<testCounter<<". teszt sikertelen."<<std::endl;
20      }
21      testCounter++;
22  }
23
24  int main()
25  {
26
27      Konyvtar* konyvtar = new Konyvtar();
28
29      Konyv* konyv1 = new Konyv("Cim1", "Szerzo1", 100, Termesztudomanyos::instance());
30      Konyv* konyv2 = new Konyv("Cim2", "Szerzo2", 200, Szepirodalmi::instance());
31
32      Szemely* szemely1 = new Szemely("Szemely1");
33
34      konyvtar->Bevetelez(konyv1);
35      check(konyvtar->get_konyvek().size() == 1);
36      check(konyvtar->get_konyvek()[0] == konyv1);
37      konyvtar->Bevetelez(konyv2);
38
39      check(konyvtar->get_konyvek()[0]->get_azon() == 1);
40      check(konyvtar->get_konyvek()[0]->get_cim() == "Cim1");
41      check(konyvtar->get_konyvek()[0]->get_szerzo() == "Szerzo1");
42      check(konyvtar->get_konyvek()[0]->get_oldal() == 100);
43      check(konyvtar->get_konyvek()[0]->get_kinn() == false);
44      check(konyvtar->get_konyvek()[0]->get_mufaj() == Termesztudomanyos::instance());
45
46      konyvtar->Belep(szemely1);
47      check(szemely1->get_konyvtar() == konyvtar);
48      check(konyvtar->get_tagok().size() == 1);
49      check(konyvtar->get_tagok()[0]->get_nev() == "Szemely1");
50
51      konyvtar->Kolcsonoz(szemely1, {1, 2}, 0);
52      check(konyv1->get_kinn() == true);
53      check(szemely1->get_kolcs()[0]->get_tetelek()[0] == konyv1);
54      check(szemely1->get_kolcs()[0]->get_tetelek().size() == 2);
55      check(konyv1->get_fej() == konyvtar->get_kolcs()[0]);
56
57
58      check(konyv1->get_mufaj()->Dij() == 100);
59      check(konyv2->get_mufaj()->Dij() == 50);
60
61      Termesztudomanyos::destroy();
62      Szepirodalmi::destroy();
63      Ifjusagi::destroy();
64
65      if (!jo)
66      {
67          return 1;
68      }
69      else
70      {
71          std::cout << "Ok " << testCounter << std::endl;
72          return 0;
73      }
74
75  }
```

## Kehely.hpp

```
1  #pragma once
2  #include "rendeles.h"
3  #include "kivansag.h"
4  #include "meret.h"
5  #include "iz.h"
6  #include "fagyizo.h"
7  #include "cukraszda.h"
8
9  #include <vector>
10 #include <string>
11 class Fagyizo;
12 class Cukraszda;
13 class Kivansag;
14 class Meret;
15 class Rendeles;
16
17 using namespace std;
18
19 class Kehely{
20
21     private:
22         vector<Iz> gomboc;
23         Meret* meret;
24
25     public:
26         Kehely(Meret* m, vector<Iz> i);
27         Meret* get_meret()
28         {
29             return meret;
30         }
31 };
```

## Kehely.cpp

```
1  #include "kehely.h"
2
3  Kehely::Kehely(Meret* m, vector<Iz> i){
4      if(i.size()<3 || i.size()>5)
5      {
6          throw exception();
7      }
8      gomboc=i;
9      meret=m;
10 }
```

## Íz.hpp

```
1  #pragma once
2  #include "rendeles.h"
3  #include "kivansag.h"
4  #include "meret.h"
5  #include "kehely.h"
6  #include "fagyizo.h"
7  #include "cukraszda.h"
8
9  #include <vector>
10 #include <string>
11
12 class Meret;
13 class Fagyizo;
14 class Cukraszda;
15 class Kivansag;
16 class Kehely;
17 class Rendeles;
18
19 using namespace std;
20
21 typedef enum Iz{
22     eper, csoki, vanilia
23 } Iz;
```

## Méret.hpp

```
2
3 #include "rendeles.h"
4 #include "kivansag.h"
5 #include "kehely.h"
6 #include "iz.h"
7 #include "fagyizo.h"
8 #include "cukraszda.h"
9
10 #include <vector>
11 #include <string>
12
13 class Fagyizo;
14 class Cukraszda;
15 class Kivansag;
16 class Kehely;
17 class Rendeles;
18
19 using namespace std;
20
21
22 class Meret{
23     protected:
24         int suly;
25     public:
26         static Meret* Atalakit(string s);
27         virtual int get_suly() {return suly;}
28         virtual ~Meret() {}
29 };
30
31 class Kicsi: public Meret{
32     private:
33         Kicsi()
34         {
35             suly=55;
36         }
37         static Kicsi* _instance;
38     public:
39         static Kicsi* instance() {if(_instance==nullptr){_instance=new Kicsi();}return _instance;}
40         static void destroy() {if(_instance!=nullptr){delete _instance;}_instance=nullptr;}
41 };
42
43
44 class Kozepes : public Meret{
45     private:
46         Kozepes()
47         {
48             suly=70;
49         }
50         static Kozepes* _instance;
51     public:
52         static Kozepes* instance() {if(_instance==nullptr){_instance=new Kozepes();}return _instance;}
53         static void destroy() {if(_instance!=nullptr){delete _instance;}_instance=nullptr;}
54 };
55
56
57 class Nagy : public Meret{
58     private:
59         Nagy(){
60             suly=90;
61         }
62         static Nagy* _instance;
63     public:
64         static Nagy* instance() {if(_instance==nullptr){_instance=new Nagy();}return _instance;}
65         static void destroy() {if(_instance!=nullptr){delete _instance;}_instance=nullptr;}
66 };
```



## Méret.cpp

```
1 #include "meret.h"
2 Kicsi* Kicsi::_instance = nullptr;
3 Nagy* Nagy::_instance = nullptr;
4 Kozepes* Kozepes::_instance=nullptr;
5
6 Meret* Meret::Atalakit(string s)
7 {
8     if(s=="kicsi")
9     {
10         return Kicsi::instance();
11     }
12     else if(s=="kozepes")
13     {
14         return Kozepes::instance();
15     }
16     else{
17         return Nagy::instance();
18     }
19 }
20
21
22 }
```

## Kívánság.hpp

```
1 #pragma once
2 #include "rendeles.h"
3 #include "kehely.h"
4 #include "meret.h"
5 #include "iz.h"
6 #include "fagyizo.h"
7 #include "cukraszda.h"
8
9 #include <vector>
10 #include <string>
11
12 class Meret;
13 class Fagyizo;
14 class Cukraszda;
15 class Kehely;
16 class Rendeles;
17
18 using namespace std;
19
20 class Kivansag{
21 public:
22     string meret;
23     vector<Iz> i;
24     Kivansag()=default;
25     ~Kivansag(){}
26
27     vector<Iz> get_iz(){return i;}
28     string get_meret(){return meret;}
29
30 };
31
```

## Rendelés.hpp

```
1  #pragma once
2  #include "kehely.h"
3  #include "kivansag.h"
4  #include "meret.h"
5  #include "iz.h"
6  #include "fagyizo.h"
7  #include "cukraszda.h"
8
9  #include <vector>
10 #include <string>
11
12 class Meret;
13 class Fagyizo;
14 class Kivansag;
15 class Kehely;
16 class Cukraszda;
17
18 using namespace std;
19
20 class Rendeles{
21     private:
22         vector<Kehely*> kelyhek;
23         string idopont;
24     public:
25         Rendeles(string t):idopont(t){}
26         void Hozzavesz(Kehely* k){
27             kelyhek.push_back(k);
28         }
29         vector<Kehely*> get_kelyhek(){return kelyhek;}
30         string get_idopont()
31         {
32             return idopont;
33         }
34
35 };
36
```

## Fagyizó.hpp

```
1  #pragma once
2  #include "rendeles.h"
3  #include "kivansag.h"
4  #include "meret.h"
5  #include "iz.h"
6  #include "kehely.h"
7  #include "cukraszda.h"
8
9  #include <vector>
10 #include <string>
11
12 class Meret;
13 class Kivansag;
14 class Cukraszda;
15 class Kehely;
16 class Rendeles;
17
18 using namespace std;
19
20 class Fagyizo{
21     private:
22         string nev,cim;
23         int ar;
24         vector<Rendeles*> napi;
25     public:
26         Fagyizo(string n, string c,int ar):nev(n),cim(c),ar(ar){}
27         ~Fagyizo(){}
28
29         void Felvesz(vector<Kivansag> /*&*/list,string t);
30
31         vector<Rendeles*> get_napi(){return napi;}
32         string get_cim(){ return cim;}
33         string get_nev(){return nev;}
34         int get_ar(){return ar;}
35
36 };
37
```

## Fagyizó.cpp

```
1 #include "fagyizo.h"
2
3 void Fagyizo::Felvesz(vector<Kivansag> /*&*/list,string t){
4     Rendeles* r = new Rendeles(t);
5     for(Kivansag e : list)
6     {
7         Meret* m = Meret::Atalakit(e.get_meret());
8         r->Hozzavesz(new Kehely(m,e.get_iz()));
9     }
10    napi.push_back(r);
11 }
```

## Cukrászda.hpp

```
1 #pragma once
2 #include "rendeles.h"
3 #include "kivansag.h"
4 #include "meret.h"
5 #include "iz.h"
6 #include "fagyizo.h"
7 #include "kehely.h"
8
9 #include <vector>
10 #include <string>
11
12 class Meret;
13 class Fagyizo;
14 class Kivansag;
15 class Kehely;
16 class Rendeles;
17
18 using namespace std;
19
20 class Cukraszda{
21 private:
22     vector<Fagyizo*> uzletek;
23 public:
24     void Nyit(Fagyizo* u);
25     vector<Fagyizo*> get_uzletek(){return uzletek;}
26
27
28 };
29
```

## Cukrászda.cpp

```
1 #include "cukraszda.h"
2
3 void Cukraszda::Nyit(Fagyizo* u){
4     for(Fagyizo* e : uzletek)
5     {
6         if(e->get_cim()==u->get_cim())
7         {
8             throw exception();
9         }
10    }
11    uzletek.push_back(u);
12
13 }
```

## main.cpp

```
1  #include "cukraszda.h"
2  #include <iostream>
3  #include <string>
4
5  #include <cassert>
6
7
8  int main()
9  {
10     Cukraszda* c = new Cukraszda();
11     c->Nyit(new Fagyizo("Fagyizo", "Varos-Utca-Hazsam", 100));
12
13     Fagyizo* f = c->get_uzletek()[0];
14
15     std::cout << f->get_nev() << std::endl;
16     std::cout << f->get_cim() << std::endl;
17     std::cout << f->get_ar() << std::endl;
18
19     Kivansag ki1, ki2, ki3;
20     ki1.i = {Iz::csoki, Iz::csoki, Iz::vanilia};
21     ki1.meret = "kicsi";
22
23     ki2.i = {Iz::csoki, Iz::eper, Iz::vanilia, Iz::eper};
24     ki2.meret = "nagy";
25
26     ki3.i = {Iz::csoki, Iz::csoki, Iz::vanilia, Iz::vanilia, Iz::eper};
27     ki3.meret = "kozepes";
28
29
30
31     std::vector<Kivansag> kList1;
32     kList1.push_back(ki1);
33     kList1.push_back(ki2);
34     kList1.push_back(ki3);
35
36     f->Felvesz(kList1, "10:00");
37     std::cout << f->get_napi()[0]->get_idopont() << std::endl;
38
39     Rendeles* r1 = f->get_napi()[0];
40
41     Kehely* ke1 = r1->get_kelyhek()[0];
42     Kehely* ke2 = r1->get_kelyhek()[1];
43     Kehely* ke3 = r1->get_kelyhek()[2];
44
45
46     assert(ke1->get_meret()->get_suly() == 55);
47     assert(ke2->get_meret()->get_suly() == 90);
48     assert(ke3->get_meret()->get_suly() == 70);
49
50     std::cout << ke1->get_meret()->get_suly() << " - " << (ke1->get_meret() == Kicsi::instance()) << std::endl;
51     std::cout << ke2->get_meret()->get_suly() << " - " << (ke2->get_meret() == Nagy::instance()) << std::endl;
52     std::cout << ke3->get_meret()->get_suly() << " - " << (ke3->get_meret() == Kozepes::instance()) << std::endl;
53
54     Kicsi::destroy();
55     Kozepes::destroy();
56     Nagy::destroy();
57     return 0;
58 }
```

## Plant.hpp

```
1  #pragma once
2
3  #include <fstream>
4  #include <string>
5  #include "Radiant.hpp"
6
7  class Plant{
8  protected:
9      std::string name;
10     int nutrient;
11     Plant(const std::string &s, int n = 0) : name(s), nutrient(n) {}
12
13 public:
14     std::string getName() const {return name; }
15     void changeNut(int n) {nutrient += n; }
16     int getNut() {return nutrient; }
17     virtual bool isAlive() const {return false;}
18     virtual bool isPuffancs() const {return false;}
19     virtual bool isDeltafa() const {return false;}
20     virtual bool isParabokor() const {return false;}
21     virtual int transMute(int need, Radiant* rad) = 0;
22     virtual ~Plant () {}
23     static Plant* create(const std::string name, char type, int food);
24 };
25
26 class Puffancs : public Plant {
27 public:
28     Puffancs(const std::string &s, int n = 0) : Plant(s, n){}
29     bool isAlive() const {return (nutrient > 0) & (nutrient < 10);}
30     bool isPuffancs() const override {return true;}
31     int transMute(int need, Radiant* rad) override {
32         return rad->transForm(need, this);
33     }
34 };
35
36 class Deltafa : public Plant {
37 public:
38     Deltafa(const std::string &s, int n = 0) : Plant(s, n){}
39     bool isAlive() const {return nutrient > 0;}
40     bool isDeltafa() const override {return true;}
41     int transMute(int need, Radiant* rad) override {
42         return rad->transForm(need, this);
43     }
44 };
45
46 class Parabokor : public Plant {
47 public:
48     Parabokor(const std::string &s, int n = 0) : Plant(s, n){}
49     bool isAlive() const {return nutrient > 0;}
50     bool isParabokor() const override {return true;}
51     int transMute(int need, Radiant* rad) override{
52         return rad->transForm(need, this);
53     }
54 };
55
```

## Plant.cpp

```
1  #include "Plant.hpp"
2
3  Plant* Plant::create(const std::string name, char c, int n)
4  {
5      switch (c)
6      {
7          case 'p': return new Puffancs(name, n);
8          case 'd': return new Deltafa(name, n);
9          case 'b': return new Parabokor(name, n);
10     }
11
12     return nullptr;
13 }
14
```

# Radiant.hpp

```
1  #pragma once
2
3  #include <string>
4
5  class Puffancs;
6  class Deltafa;
7  class Parabokor;
8
9  class Radiant{
10 public:
11     virtual int transForm(int need, Puffancs *p) = 0;
12     virtual int transForm(int need, Deltafa *p) = 0;
13     virtual int transForm(int need, Parabokor *p) = 0;
14     virtual bool isAlpha() const {return false;}
15     virtual bool isDelta() const {return false;}
16     virtual bool isNoRad() const {return false;}
17     virtual ~Radiant() {}
18     static Radiant* create();
19 };
20
21 class Alpha : public Radiant{
22 private:
23     Alpha(){}
24     static Alpha* _instance;
25
26 public:
27     static Alpha* Instance();
28     int transForm(int need, Puffancs *p) override;
29     int transForm(int need, Deltafa *p) override;
30     int transForm(int need, Parabokor *p) override;
31     bool isAlpha() const override {return true;}
32
33     void static destroy() {
34         if ( nullptr!=_instance ) {
35             delete _instance;
36             _instance = nullptr;
37         }
38     }
39 };
40
41 class Delta : public Radiant{
42 private:
43     Delta(){}
44     static Delta* _instance;
45
46 public:
47     static Delta* Instance();
48     int transForm(int need, Puffancs *p) override;
49     int transForm(int need, Deltafa *p) override;
50     int transForm(int need, Parabokor *p) override;
51     bool isDelta() const override {return true;}
52
53     void static destroy() {
54         if ( nullptr!=_instance ) {
55             delete _instance;
56             _instance = nullptr;
57         }
58     }
59 };
60
61 class NoRad : public Radiant{
62 private:
63     NoRad(){}
64     static NoRad* _instance;
65
66 public:
67     static NoRad* Instance();
68     int transForm(int need, Puffancs *p) override;
69     int transForm(int need, Deltafa *p) override;
70     int transForm(int need, Parabokor *p) override;
71     bool isNoRad() const override {return true;}
72
73     void static destroy() {
74         if ( nullptr!=_instance ) {
75             delete _instance;
76             _instance = nullptr;
77         }
78     }
79 };
80
```

# Radiant.cpp

```
1  #include "Radiant.hpp"
2  #include "Plant.hpp"
3
4  Radiant* Radiant::create()
5  {
6      return NoRad::Instance();
7  }
8
9  Alpha* Alpha::_instance = nullptr;
10 Alpha* Alpha::Instance()
11 {
12     if(_instance == nullptr) {
13         _instance = new Alpha();
14     }
15     return _instance;
16 }
17
18 int Alpha::transForm(int need, Puffancs *p)
19 {
20     p->changeNut(2);
21
22     if(p->isAlive()) {
23         return need + 10;
24     } else {
25         return need;
26     }
27
28     return need;
29 }
30
31 int Alpha::transForm(int need, Deltafa *d)
32 {
33     d->changeNut(-3);
34
35     if(d->isAlive()) {
36         if(d->getNut() < 5) {
37             return need - 4;
38         } else if(d->getNut() >= 5 && d->getNut() <= 10) {
39             return need - 1;
40         } else {
41             return need;
42         }
43     }
44
45     return need;
46 }
47
48 int Alpha::transForm(int need, Parabokor *b)
49 {
50     b->changeNut(1);
51     return need;
52 }
53
54 Delta* Delta::_instance = nullptr;
55 Delta* Delta::Instance()
56 {
57     if(_instance == nullptr) {
58         _instance = new Delta();
59     }
60     return _instance;
61 }
62
63 int Delta::transForm(int need, Puffancs *p)
64 {
65     p->changeNut(-2);
66
67     if(p->isAlive()) {
68         return need + 10;
69     } else {
70         return need;
71     }
72
73     return need;
74 }
75
```

```

76 int Delta::transForm(int need, Deltafa *d)
77 {
78     d->changeNut(4);
79
80     if(d->isAlive()){
81         if(d->getNut() < 5){
82             return need - 4;
83         } else if(d->getNut() >=5 && d->getNut() <=10){
84             return need -1;
85         } else{
86             return need;
87         }
88     }
89
90     return need;
91 }
92
93 int Delta::transForm(int need, Parabokor *b)
94 {
95     b->changeNut(1);
96     return need;
97 }
98
99
100 NoRad* NoRad::_instance = nullptr;
101 NoRad* NoRad::Instance()
102 {
103     if(_instance == nullptr) {
104         _instance = new NoRad();
105     }
106     return _instance;
107 }
108
109 int NoRad::transForm(int need, Puffancs *p)
110 {
111     p->changeNut(-1);
112
113     if(p->isAlive()){
114         return need + 10;
115     } else{
116         return need;
117     }
118
119     return need;
120 }
121
122 int NoRad::transForm(int need, Deltafa *d)
123 {
124     d->changeNut(-1);
125
126     if(d->isAlive()){
127         if(d->getNut() < 5){
128             return need - 4;
129         } else if(d->getNut() >=5 && d->getNut() <=10){
130             return need -1;
131         } else{
132             return need;
133         }
134     }
135
136     return need;
137 }
138
139 int NoRad::transForm(int need, Parabokor *b)
140 {
141     b->changeNut(-1);
142     return need;
143 }
144

```



## main.cpp

```
1  #include <iostream>
2  #include <cstdlib>
3  #include <fstream>
4  #include <sstream>
5  #include <vector>
6  #include <string>
7  #include "Plant.hpp"
8  #include "Radiant.hpp"
9  using namespace std;
10
11 int createPlanet(const std::string fileName, std::vector<Plant*> &plants){
12     ifstream f(fileName);
13     if(f.fail()){
14         std::cout << "Sowwy but I can't find the file o.o" << std::endl;
15         exit(1);
16     }
17
18     int plantDB, dayDB, nut;
19     std::string name;
20     char type;
21
22     f >> plantDB;
23     plants.resize(plantDB);
24
25     for(int i=0; i<plantDB; ++i){
26         f >> name >> type >> nut;
27         plants[i] = Plant::create(name, type, nut);
28     }
29
30     f >> dayDB;
31     return dayDB;
32 }
33
34 void destroyRad(Radiant* &rad){
35     if(rad->isAlpha()){
36         Alpha::destroy();
37     } else if(rad->isDelta()){
38         Delta::destroy();
39     } else if(rad->isNoRad()){
40         NoRad::destroy();
41     }
42 }
43
44 void daysLater(vector<Plant*> &plants, Radiant* &rad, int dayDB, std::string &strongestSurvivor, vector<std::string>& report){
45     try{
46         report.clear();
47         report.push_back("NO RADIATION");
48         while(dayDB > 0){
49             int need = 0;
50
51             for(Plant* p:plants){
52                 if(p->isAlive()){
53                     need = p->transMute(need, rad);
54
55                     std::string plant = "";
56                     stringstream ss;
57
58                     plant = plant + p->getName() + " ";
59                     ss << p->getNut();
60
61                     if(p->isPuffancs()){
62                         plant = plant + "| puffancs \t|" + ss.str();
63                     } else if(p->isDeltafa()){
64                         plant = plant + "| deltafa \t|" + ss.str();
65                     } else if(p->isParabokor()){
66                         plant = plant + "| parabokor \t|" + ss.str();
67                     }
68
69                     report.push_back(plant);
70                 }
71             }
72         }
73     }
```

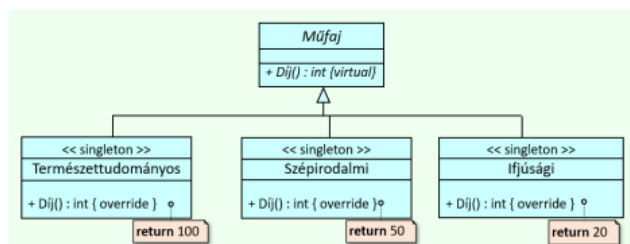
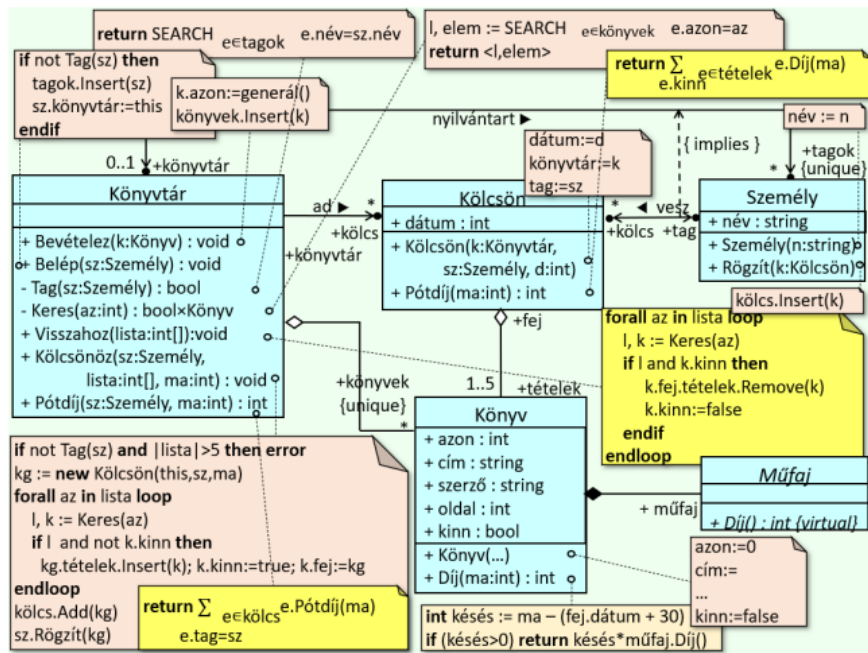
```

73     if(need >= 3){
74         destroyRad(rad);
75         rad = Alpha::Instance();
76         report.push_back("ALPHA RADIATION");
77     } else if(need <= -3){
78         destroyRad(rad);
79         rad = Delta::Instance();
80         report.push_back("DELTA RADIATION");
81     } else {
82         destroyRad(rad);
83         rad = NoRad::Instance();
84         report.push_back("NO RADIATION");
85     }
86
87     dayDB--;
88 }
89
90 int nutMax = 0;
91 std::string strongest;
92
93 for(Plant* p : plants){
94     if(p->isAlive() && p->getNut()>nutMax){
95         strongest = p->getName();
96         nutMax = p->getNut();
97     }
98 }
99
100 if(nutMax == 0){
101     strongestSurvivor = "There are no survivors..";
102 } else{
103     strongestSurvivor = "The strongest survivor is the " + strongest;
104 }
105 } catch(exception e){
106     std::cout << e.what() << std::endl;
107 }
108 }
109
110 void destroyAll(vector<Plant*> &plants, vector<std::string>& report){
111
112     for(Plant* p : plants){
113         delete p;
114     }
115
116     Alpha::destroy();
117     Delta::destroy();
118     NoRad::destroy();
119 }
120
121 std::string check(bool test){
122     if(!test){
123         return "oopsie";
124     }
125
126     return "allGood";
127 }
128
129 bool AllTestPassed(vector<std::string>& result, int testDB){
130     int tests = 0;
131     for(std::string s : result){
132         if(s=="allGood"){
133             tests++;
134         }
135     }
136
137     if(tests == testDB){
138         return true;
139     }
140
141     return false;
142 }
143
144 int main()
145 {
146     //PLANT LIFE SIMULATION
147     Radiant* rad = Radiant::create();
148
149     std::vector<Plant*> plants;
150     int days = createPlanet("input.txt", plants);
151
152     std::string ss;
153     vector<std::string> report;
154     daysLater(plants, rad, days, ss, report);
155
156     for(std::string s : report){
157         std::cout << s << std::endl;
158     }
159
160     std::cout << "-----" << std::endl;
161     std::cout << ss << std::endl;
162     destroyAll(plants, report);

```

# KÖNYVTÁR

A terv osztálydiagramja:



# FAGYIZÓ

