Könyv.hpp

```cpp
#pragma once
#include "konyvtar.h"
#include "konyv.h"
#include "kolcson.h"
#include "mufaj.h"
#include "szemely.h"
#include <string>
#include <iostream>
class Kolcson;
class Konyvtar;
class Mufaj;
class Szemely;

using namespace std;

class Konyv{
    private:
        int azon, oldal;
        string cim, szerzo;
        bool kinn;
        Mufaj*mufaj;
        Kolcson* fej;

    public:
        Konyv()=default;
        Konyv(string cim, string szerzo,int oldal, Mufaj* mufaj):azon(0), szerzo(szerzo),oldal(oldal),mufaj(mufaj),cim(cim), fej(0),kinn(false){}
        int Dij(int ma);
        ~Konyv();

        void set_azon(int azon){this->azon=azon;}
        int get_azon(){ return azon;}
        bool get_kinn(){ return kinn;}
        void set_kinn(bool k) {kinn=k; }
        Kolcson* get_fej(){ return fej;}
        void set_fej(Kolcson*k ){fej=k;}
        string get_cim(){return cim;}
        string get_szerzo(){return szerzo;}
        int get_oldal(){return oldal;}
        Mufaj* get_mufaj(){return mufaj;}
};
```

Könyv.cpp

```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

```cpp
#pragma once
#include "konyvtar.h"
#include "konyv.h"
#include "kolcson.h"
#include "mufaj.h"
#include "szemely.h"
#include <string>
#include <iostream>

using namespace std;

class Kolcson;
class Konyvtar;
class Konyv;
class Szemely;

class Mufaj{

    public:
        virtual int Dij()=0;
        virtual ~Mufaj()=default;
};
```

```
class Termeszettudomanyos : public Mufaj{
    private:
        static Termeszettudomanyos* _instance;
        Termeszettudomanyos(){}
        Kolcson* fej;
    public:
        static Termeszettudomanyos* instance(){
            if(_instance==nullptr)
            {
                _instance= new Termeszettudomanyos();
            }
            return _instance;
        }
        static void destroy(){
            if(_instance!=nullptr)
            {
                delete _instance;
            }
            _instance=nullptr;
        }
        int Dij() override{
            return 100;
        }
};

class Szepirodalmi : public Mufaj{
    private:
        static Szepirodalmi* _instance;
        Szepirodalmi(){}
    public:
        static Szepirodalmi* instance(){
            if(_instance==nullptr)
            {
                _instance= new Szepirodalmi();
            }
            return _instance;
        }
        static void destroy(){
            if(_instance!=nullptr)
            {
                delete _instance;
            }
            _instance=nullptr;
        }
        int Dij() override{
            return 50;
        }
};


class Ifjusagi : public Mufaj{
    private:
        static Ifjusagi* _instance;
        Ifjusagi(){}
        Kolcson* fej;
    public:
        static Ifjusagi* instance(){
            if(_instance==nullptr)
            {
                _instance= new Ifjusagi();
            }
            return _instance;
        }
        static void destroy(){
            if(_instance!=nullptr)
            {
                delete _instance;
            }
            _instance=nullptr;
        }
        int Dij() override{
            return 20;
        }
};
```

Műfaj.cpp

```cpp
#include "mufaj.h"

Termeszettudomanyos* Termeszettudomanyos::_instance=nullptr;
Ifjusagi* Ifjusagi::_instance=nullptr;
Szepirodalmi* Szepirodalmi::_instance=nullptr;
```

Kölcsön.hpp

```cpp
#pragma once
#include "konyvtar.h"
#include "konyv.h"
#include "kolcson.h"
#include "mufaj.h"
#include "szemely.h"
#include <string>
#include <iostream>
#include <vector>

using namespace std;
class Mufaj;
class Konyvtar;
class Konyv;
class Szemely;

class Kolcson{
    private:
        int datum;
        Konyvtar* konyvtar;
        Szemely* tag;
        vector<Konyv*> tetelek;

    public:
        Kolcson(Konyvtar* k, Szemely* sz, int d): datum(d),konyvtar(k),tag(sz){}
        ~Kolcson(){}

        //**********


        vector<Konyv*> &get_tetelek(){ return tetelek; }
        int get_datum(){return datum;}

};
```

Személy.hpp

```cpp
#pragma once
#include "konyvtar.h"
#include "konyv.h"
#include "kolcson.h"
#include "mufaj.h"
#include "szemely.h"
#include <string>
#include <iostream>
#include <vector>

using namespace std;
class Kolcson;
class Konyvtar;
class Mufaj;
class Konyv;



class Szemely{
    private:
        string nev;
        vector<Kolcson*> kolcs;
        Konyvtar* konyvtar;

    public:
        Szemely(string n):nev(n){}
        void Rogzit(Kolcson*k);


        void set_konyvtar(Konyvtar* k){ konyvtar = k;}
        string get_nev(){ return nev;}
        vector<Kolcson*> get_kolcs(){return kolcs;}
        Konyvtar* get_konyvtar(){return konyvtar;}


};
```

Személy.cpp

```cpp
#include "szemely.h"

void Szemely::Rogzit(Kolcson*k){
    kolcs.push_back(k);
}
```

Könyvtár.hpp

```cpp
#pragma once
#include "konyvtar.h"
#include "konyv.h"
#include "kolcson.h"
#include "mufaj.h"
#include "szemely.h"
#include <string>
#include <iostream>
#include <vector>
class Mufaj;
class Kolcson;
class Konyv;
class Szemely;

using namespace std;

class Konyvtar{

    private:
        vector<Kolcson*> kolcs;
        vector<Konyv*> konyvek;
        vector<Szemely*> tagok;
        int c;
    public:
        Konyvtar():c(1){}
        void Bevetelez(Konyv* k);
        void Belep(Szemely* &sz);
        bool Tag(Szemely* sz);
        bool Keres(int az,Konyv* &k);
        void Kolcsonoz(Szemely* &sz, vector<int> lista, int ma);



        vector<Konyv*> get_konyvek(){return konyvek;}
        vector<Kolcson*> get_kolcs(){return kolcs;}
        vector<Szemely*> get_tagok(){return tagok;}


};
```

Könyvtár.cpp

```cpp
#include "konyvtar.h"

void Konyvtar::Bevetelez(Konyv* k)
{
    k->set_azon(c);
    c++;
    konyvek.push_back(k);

}

void Konyvtar::Belep(Szemely* &sz){
    if(!Tag(sz))
    {
        tagok.push_back(sz);
        sz->set_konyvtar(this);
    }

}

bool Konyvtar::Tag(Szemely* sz)
{
    for(Szemely*  e : tagok)
    {
        if(e->get_nev()==sz->get_nev())
        {
            return true;
        }

    }
    return false;
}

bool Konyvtar::Keres(int az,Konyv* &k)
{
    bool van=false;
    for(Konyv* e : konyvek)
    {
        if(e->get_azon()==az)
        {
            k=e;
            van=true;
            return true;
        }
    }
    return van;
}

void Konyvtar::Kolcsonoz(Szemely* &sz, vector<int> lista, int ma)
{
    if(!Tag(sz) && lista.size()>5) { throw exception();}
    Kolcson* kg = new Kolcson(this,sz,ma);
    Konyv* k;

    for(int az : lista)
    {
        bool van=false;
        van=this->Keres(az,k);
        if(van && !k->get_kinn())
        {
            kg->get_tetelek().push_back(k);
            k->set_kinn(true);
            k->set_fej(kg);
        }
    }
    kolcs.push_back(kg);
    sz->Rogzit(kg);

}
```

main.cpp

```cpp
#include "konyvtar.h"
#include "konyv.h"
#include "kolcson.h"
#include "mufaj.h"
#include "szemely.h"

#include <iostream>


int testCounter = 1;
bool jo = true;

void check(bool l)
{
    if(!l)
    {
        jo = false;
        std::cerr<<testCounter<<". teszt sikertelen."<<std::endl;
    }
    testCounter++;
}

int main()
{

    Konyvtar* konyvtar = new Konyvtar();

    Konyv* konyv1 = new Konyv("Cim1","Szerzo1",100,Termeszettudomanyos::instance());
    Konyv* konyv2 = new Konyv("Cim2","Szerzo2",200,Szepirodalmi::instance());

    Szemely* szemely1 = new Szemely("Szemely1");

    konyvtar->Bevetelez(konyv1);
    check(konyvtar->get_konyvek().size() == 1);
    check(konyvtar->get_konyvek()[0] == konyv1);
    konyvtar->Bevetelez(konyv2);

    check(konyvtar->get_konyvek()[0]->get_azon() == 1);
    check(konyvtar->get_konyvek()[0]->get_cim() == "Cim1");
    check(konyvtar->get_konyvek()[0]->get_szerzo() == "Szerzo1");
    check(konyvtar->get_konyvek()[0]->get_oldal() == 100);
    check(konyvtar->get_konyvek()[0]->get_kinn() == false);
    check(konyvtar->get_konyvek()[0]->get_mufaj() == Termeszettudomanyos::instance());

    konyvtar->Belep(szemely1);
    check(szemely1->get_konyvtar() == konyvtar);
    check(konyvtar->get_tagok().size() == 1);
    check(konyvtar->get_tagok()[0]->get_nev() == "Szemely1");

    konyvtar->Kolcsonoz(szemely1,{1,2},0);
    check(konyv1->get_kinn() == true);
    check(szemely1->get_kolcs()[0]->get_tetelek()[0] == konyv1);
    check(szemely1->get_kolcs()[0]->get_tetelek().size() == 2);
    check(konyv1->get_fej() == konyvtar->get_kolcs()[0]);


    check(konyv1->get_mufaj()->Dij() == 100);
    check(konyv2->get_mufaj()->Dij() == 50);

    Termeszettudomanyos::destroy();
    Szepirodalmi::destroy();
    Ifjusagi::destroy();

    if (!jo)
    {
        return 1;
    }
    else
    {
        std::cout << "Ok "  << testCounter << std::endl;
        return 0;
    }

}
```

Kehely.hpp

```cpp
#pragma once
#include "rendeles.h"
#include "kivansag.h"
#include "meret.h"
#include "iz.h"
#include "fagyizo.h"
#include "cukraszda.h"

#include <vector>
#include <string>
class Fagyizo;
class Cukraszda;
class Kivansag;
class Meret;
class Rendeles;

using namespace std;

class Kehely{

    private:
        vector<Iz> gomboc;
        Meret* meret;

    public:
        Kehely(Meret* m, vector<Iz> i);
        Meret* get_meret()
        {
            return meret;
        }
};
```

Kehely.cpp

```cpp
#include "kehely.h"

Kehely::Kehely(Meret* m, vector<Iz> i){
    if(i.size()<3 ||i.size()>5)
    {
        throw exception();
    }
    gomboc=i;
    meret=m;
}
```

Íz.hpp

```cpp
#pragma once
#include "rendeles.h"
#include "kivansag.h"
#include "meret.h"
#include "kehely.h"
#include "fagyizo.h"
#include "cukraszda.h"

#include <vector>
#include <string>

class Meret;
class Fagyizo;
class Cukraszda;
class Kivansag;
class Kehely;
class Rendeles;

using namespace std;

typedef enum Iz{
    eper,csoki,vanilia
} Iz;
```

Méret.hpp

```cpp

#include "rendeles.h"
#include "kivansag.h"
#include "kehely.h"
#include "iz.h"
#include "fagyizo.h"
#include "cukraszda.h"

#include <vector>
#include <string>

class Fagyizo;
class Cukraszda;
class Kivansag;
class Kehely;
class Rendeles;

using namespace std;


class Meret{
    protected:
        int suly;
    public:
        static Meret* Atalakit(string s);
        virtual int get_suly(){return suly;}
        virtual ~Meret(){}
};
```

```cpp
class Kicsi: public Meret{
    private:
        Kicsi()
        {
            suly=55;
        }
        static Kicsi* _instance;
    public:
        static Kicsi* instance(){if(_instance==nullptr){_instance=new Kicsi();}return _instance;}
        static void destroy(){if(_instance!=nullptr){delete _instance;}_instance=nullptr;}

};

class Kozepes : public Meret{
    private:
        Kozepes()
        {
            suly=70;
        }
        static Kozepes* _instance;
    public:
        static Kozepes* instance(){if(_instance==nullptr){_instance=new Kozepes();}return _instance;}
        static void destroy(){if(_instance!=nullptr){delete _instance;}_instance=nullptr;}

};

class Nagy : public Meret{
    private:
        Nagy(){
            suly=90;
        }
        static Nagy* _instance;
    public:
        static Nagy* instance(){if(_instance==nullptr){_instance=new Nagy();}return _instance;}
        static void destroy(){if(_instance!=nullptr){delete _instance;}_instance=nullptr;}
};
```

Méret.cpp

```cpp
#include "meret.h"
Kicsi* Kicsi::_instance = nullptr;
Nagy* Nagy::_instance = nullptr;
Kozepes* Kozepes::_instance=nullptr;

Meret* Meret::Atalakit(string s)
{
    if(s=="kicsi")
    {
        return Kicsi::instance();
    }
    else if(s=="kozepes")
    {
        return Kozepes::instance();
    }
    else{
        return Nagy::instance();

    }


}
```

Kívánság.hpp

```cpp
#pragma once
#include "rendeles.h"
#include "kehely.h"
#include "meret.h"
#include "iz.h"
#include "fagyizo.h"
#include "cukraszda.h"

#include <vector>
#include <string>

class Meret;
class Fagyizo;
class Cukraszda;
class Kehely;
class Rendeles;

using namespace std;

class Kivansag{

    public:
        string meret;
        vector<Iz> i;
        Kivansag()=default;
        ~Kivansag(){}

        vector<Iz> get_iz(){return i;}
        string get_meret(){return meret;}

};
```

Rendelés.hpp

```cpp
#pragma once
#include "kehely.h"
#include "kivansag.h"
#include "meret.h"
#include "iz.h"
#include "fagyizo.h"
#include "cukraszda.h"

#include <vector>
#include <string>

class Meret;
class Fagyizo;
class Kivansag;
class Kehely;
class Cukraszda;

using namespace std;

class Rendeles{
    private:
        vector<Kehely*> kelyhek;
        string idopont;
    public:
        Rendeles(string t):idopont(t){}
        void Hozzavesz(Kehely* k){
            kelyhek.push_back(k);
        }
        vector<Kehely*> get_kelyhek(){return kelyhek;}
        string get_idopont()
        {
            return idopont;
        }


};
```

Fagyizó.hpp

```cpp
#pragma once
#include "rendeles.h"
#include "kivansag.h"
#include "meret.h"
#include "iz.h"
#include "kehely.h"
#include "cukraszda.h"

#include <vector>
#include <string>

class Meret;
class Kivansag;
class Cukraszda;
class Kehely;
class Rendeles;

using namespace std;

class Fagyizo{
    private:
        string nev,cim;
        int ar;
        vector<Rendeles*> napi;
    public:
        Fagyizo(string n, string c,int ar):nev(n),cim(c),ar(ar){}
        ~Fagyizo(){}

        void Felvesz(vector<Kivansag> /*&*/list,string t);

        vector<Rendeles*> get_napi(){return napi;}
        string get_cim(){ return cim;}
        string get_nev(){return nev;}
        int get_ar(){return ar;}

};
```

Fagyizó.cpp

```cpp
#include "fagyizo.h"

void Fagyizo::Felvesz(vector<Kivansag> /*&*/list,string t){
            Rendeles* r = new Rendeles(t);
            for(Kivansag e : list)
            {
                Meret* m = Meret::Atalakit(e.get_meret());
                r->Hozzavesz(new Kehely(m,e.get_iz()));
            }
            napi.push_back(r);
        }
```

Cukrászda.hpp

```cpp
#pragma once
#include "rendeles.h"
#include "kivansag.h"
#include "meret.h"
#include "iz.h"
#include "fagyizo.h"
#include "kehely.h"

#include <vector>
#include <string>

class Meret;
class Fagyizo;
class Kivansag;
class Kehely;
class Rendeles;

using namespace std;

class Cukraszda{

    private:
        vector<Fagyizo*> uzletek;
    public:
        void Nyit(Fagyizo* u);
        vector<Fagyizo*> get_uzletek(){return uzletek;}


};
```

Cukrászda.cpp

```cpp
#include "cukraszda.h"

void Cukraszda::Nyit(Fagyizo* u){
        for(Fagyizo* e : uzletek)
        {
            if(e->get_cim()==u->get_cim())
            {
                throw exception();
            }
        }
        uzletek.push_back(u);

    }
```

main.cpp

```cpp
#include "cukraszda.h"
#include <iostream>
#include <string>

 #include <cassert>


int main()
{
    Cukraszda* c = new Cukraszda();
    c->Nyit(new Fagyizo("Fagyizo", "Varos-Utca-Hazszam",100));

    Fagyizo* f = c->get_uzletek()[0];

    std::cout << f->get_nev() << std::endl;
    std::cout << f->get_cim() << std::endl;
    std::cout << f->get_ar() << std::endl;

    Kivansag ki1, ki2, ki3;
    ki1.i = {Iz::csoki,Iz::csoki,Iz::vanilia};
    ki1.meret = "kicsi";

    ki2.i = {Iz::csoki,Iz::eper,Iz::vanilia,Iz::eper};
    ki2.meret = "nagy";

    ki3.i = {Iz::csoki,Iz::csoki,Iz::vanilia,Iz::vanilia,Iz::eper};
    ki3.meret = "kozepes";



    std::vector<Kivansag> kList1;
    kList1.push_back(ki1);
    kList1.push_back(ki2);
    kList1.push_back(ki3);

    f->Felvesz(kList1,"10:00");
    std::cout << f->get_napi()[0]->get_idopont() << std::endl;

    Rendeles* r1 = f->get_napi()[0];

    Kehely* ke1 = r1->get_kelyhek()[0];
    Kehely* ke2 = r1->get_kelyhek()[1];
    Kehely* ke3 = r1->get_kelyhek()[2];


    assert(ke1->get_meret()->get_suly() == 55);
    assert(ke2->get_meret()->get_suly() == 90);
    assert(ke3->get_meret()->get_suly() == 70);

    std::cout << ke1->get_meret()->get_suly() << " - " << (ke1->get_meret() == Kicsi::instance()) << std::endl;
    std::cout << ke2->get_meret()->get_suly() << " - " << (ke2->get_meret() == Nagy::instance()) << std::endl;
    std::cout << ke3->get_meret()->get_suly() << " - " << (ke3->get_meret() == Kozepes::instance()) << std::endl;

    Kicsi::destroy();
    Kozepes::destroy();
    Nagy::destroy();
    return 0;
}
```

Plant.hpp

```cpp
#pragma once

#include <fstream>
#include <string>
#include "Radiant.hpp"

class Plant{
protected:
    std::string name;
    int nutrient;
    Plant(const std::string &s, int n = 0) : name(s), nutrient(n) {}

public:
    std::string getName() const {return name; }
    void changeNut(int n) {nutrient += n; }
    int getNut() {return nutrient; }
    virtual bool isAlive() const {return false;}
    virtual bool isPuffancs() const {return false;}
    virtual bool isDeltafa() const {return false;}
    virtual bool isParabokor() const {return false;}
    virtual int transMute(int need, Radiant* rad) = 0;
    virtual ~Plant () {}
    static Plant* create(const std::string name, char type,  int food);
};

class Puffancs : public Plant {
public:
    Puffancs(const std::string &s, int n = 0) : Plant(s, n){}
    bool isAlive() const {return (nutrient > 0) & (nutrient < 10);}
    bool isPuffancs() const override {return true;}
    int transMute(int need, Radiant* rad) override {
        return rad->transForm(need, this);
    }
};

class Deltafa : public Plant {
public:
    Deltafa(const std::string &s, int n = 0) : Plant(s, n){}
    bool isAlive() const {return nutrient > 0;}
    bool isDeltafa() const override {return true;}
    int transMute(int need, Radiant* rad) override {
        return rad->transForm(need, this);
    }
};

class Parabokor : public Plant {
public:
    Parabokor(const std::string &s, int n = 0) : Plant(s, n){}
    bool isAlive() const {return nutrient > 0;}
    bool isParabokor() const override {return true;}
    int transMute(int need, Radiant* rad) override{
        return rad->transForm(need, this);
    }
};

```

Plant.cpp

```cpp
#include "Plant.hpp"

Plant* Plant::create(const std::string name, char c, int n)
{
    switch (c)
    {
        case 'p': return new Puffancs(name, n);
        case 'd': return new Deltafa(name, n);
        case 'b': return new Parabokor(name, n);
    }

    return nullptr;
}

```

Radiant.hpp

```hpp
#pragma once

#include <string>

class Puffancs;
class Deltafa;
class Parabokor;

class Radiant{
public:
    virtual int transForm(int need, Puffancs *p) = 0;
    virtual int transForm(int need, Deltafa *p) = 0;
    virtual int transForm(int need, Parabokor *p) = 0;
    virtual bool isAlpha() const {return false;}
    virtual bool isDelta() const {return false;}
    virtual bool isNoRad() const {return false;}
    virtual ~Radiant() {}
    static Radiant* create();
};

class Alpha : public Radiant{
private:
    Alpha(){}
    static Alpha* _instance;

public:
    static Alpha* Instance();
    int transForm(int need, Puffancs *p) override;
    int transForm(int need, Deltafa *p) override;
    int transForm(int need, Parabokor *p) override;
    bool isAlpha() const override {return true;}

    void static destroy() {
        if ( nullptr!=_instance ) {
            delete _instance;
            _instance = nullptr;
        }
    }
};

class Delta : public Radiant{
private:
    Delta(){}
    static Delta* _instance;

public:
    static Delta* Instance();
    int transForm(int need, Puffancs *p) override;
    int transForm(int need, Deltafa *p) override;
    int transForm(int need, Parabokor *p) override;
    bool isDelta() const override {return true;}

    void static destroy() {
        if ( nullptr!=_instance ) {
            delete _instance;
            _instance = nullptr;
        }
    }
};

class NoRad : public Radiant{
private:
    NoRad(){}
    static NoRad* _instance;

public:
    static NoRad* Instance();
    int transForm(int need, Puffancs *p) override;
    int transForm(int need, Deltafa *p) override;
    int transForm(int need, Parabokor *p) override;
    bool isNoRad() const override {return true;}

    void static destroy() {
        if ( nullptr!=_instance ) {
            delete _instance;
            _instance = nullptr;
        }
    }
};

```

Radiant.cpp

```cpp
#include "Radiant.hpp"
#include "Plant.hpp"

Radiant* Radiant::create()
{
    return NoRad::Instance();
}

Alpha* Alpha::_instance = nullptr;
Alpha* Alpha::Instance()
{
    if(_instance == nullptr) {
        _instance = new Alpha();
    }
    return _instance;
}

int Alpha::transForm(int need, Puffancs *p)
{
    p->changeNut(2);

    if(p->isAlive()){
        return need + 10;
    } else{
        return need;
    }

    return need;
}

int Alpha::transForm(int need, Deltafa *d)
{
    d->changeNut(-3);

    if(d->isAlive()){
        if(d->getNut() < 5){
            return need - 4;
        } else if(d->getNut() >=5 && d->getNut() <=10){
            return need - 1;
        } else{
            return need;
        }
    }

    return need;
}

int Alpha::transForm(int need, Parabokor *b)
{
    b->changeNut(1);
    return need;
}

Delta* Delta::_instance = nullptr;
Delta* Delta::Instance()
{
    if(_instance == nullptr) {
        _instance = new Delta();
    }
    return _instance;
}

int Delta::transForm(int need, Puffancs *p)
{
    p->changeNut(-2);

    if(p->isAlive()){
        return need + 10;
    } else{
        return need;
    }

    return need;
}

```

```
int Delta::transForm(int need, Deltafa *d)
{
    d->changeNut(4);

    if(d->isAlive()){
        if(d->getNut() < 5){
            return need - 4;
        } else if(d->getNut() >=5 && d->getNut() <=10){
            return need -1;
        } else{
            return need;
        }
    }

    return need;

}

int Delta::transForm(int need, Parabokor *b)
{
    b->changeNut(1);
    return need;
}

NoRad* NoRad::_instance = nullptr;
NoRad* NoRad::Instance()
{
    if(_instance == nullptr) {
        _instance = new NoRad();
    }
    return _instance;
}

int NoRad::transForm(int need, Puffancs *p)
{
    p->changeNut(-1);

    if(p->isAlive()){
        return need + 10;
    } else{
        return need;
    }

    return need;
}

int NoRad::transForm(int need, Deltafa *d)
{
    d->changeNut(-1);

    if(d->isAlive()){
        if(d->getNut() < 5){
            return need - 4;
        } else if(d->getNut() >=5 && d->getNut() <=10){
            return need -1;
        } else{
            return need;
        }
    }

    return need;
}

int NoRad::transForm(int need, Parabokor *b)
{
    b->changeNut(-1);
    return need;
}
```

main.cpp

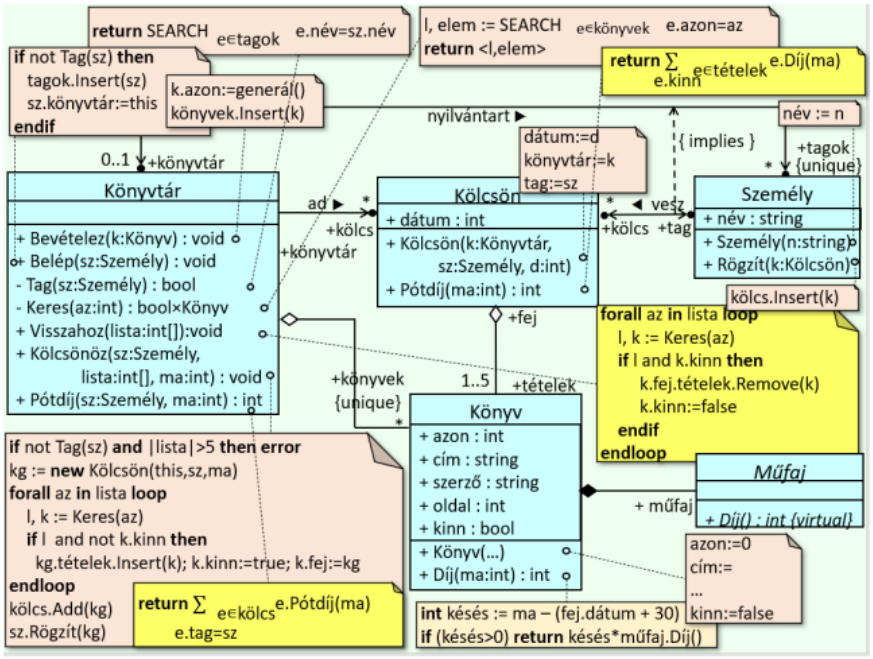
```cpp
#include <iostream>
#include <cstdlib>
#include <fstream>
#include <sstream>
#include <vector>
#include <string>
#include "Plant.hpp"
#include "Radiant.hpp"
using namespace std;

int createPlanet(const std::string fileName, std::vector<Plant*> &plants){
    ifstream f(fileName);
    if(f.fail()){
        std::cout << "Sowwy but I can't find the file o.o" << std::endl;
        exit(1);
    }

    int plantDB, dayDB, nut;
    std::string name;
    char type;

    f >> plantDB;
    plants.resize(plantDB);

    for(int i=0; i<plantDB; ++i){
        f >> name >> type >> nut;
        plants[i] = Plant::create(name, type, nut);
    }

    f >> dayDB;
    return dayDB;
}

void destroyRad(Radiant* &rad){
    if(rad->isAlpha()){
        Alpha::destroy();
    } else if(rad->isDelta()){
        Delta::destroy();
    } else if (rad->isNoRad()){
        NoRad::destroy();
    }
}
```

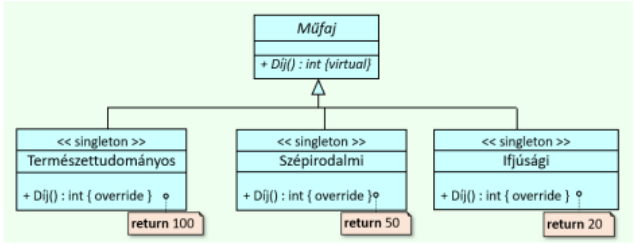
```cpp
void daysLater(vector<Plant*> &plants, Radiant* &rad, int dayDB, std::string &strongestSurvivor, vector<std::string>& report){
    try{
        report.clear();
        report.push_back("NO RADIATION");
        while(dayDB > 0){
            int need = 0;

            for(Plant* p:plants){
                if(p->isAlive()){
                    need = p->transMute(need, rad);

                    std::string plant = "";
                    stringstream ss;

                    plant = plant + p->getName() + " ";
                    ss << p->getNut();

                    if(p->isPuffancs()){
                        plant = plant + "| puffancs \t|" + ss.str();
                    } else if(p->isDeltafa()){
                        plant = plant + "| deltafa \t|" + ss.str();
                    } else if(p->isParabokor()){
                        plant = plant + "| parabokor \t|" + ss.str();
                    }

                    report.push_back(plant);
                }
            }
```

```
            if(need >= 3){
                destroyRad(rad);
                rad = Alpha::Instance();
                report.push_back("ALPHA RADIATION");
            } else if(need <= -3){
                destroyRad(rad);
                rad = Delta::Instance();
                report.push_back("DELTA RADIATION");
            } else {
                destroyRad(rad);
                rad = NoRad::Instance();
                report.push_back("NO RADIATION");
            }

            dayDB--;
        }

        int nutMax = 0;
        std::string strongest;

        for(Plant* p : plants){
            if(p->isAlive() && p->getNut()>nutMax){
                strongest = p->getName();
                nutMax = p->getNut();
            }
        }

        if(nutMax == 0){
            strongestSurvivor = "There are no survivors..";
        } else{
            strongestSurvivor = "The strongest survivor is the " + strongest;
        }
    } catch(exception e){
        std::cout << e.what() << std::endl;
    }
}

void destroyAll(vector<Plant*> &plants, vector<std::string>& report){

    for(Plant* p : plants){
        delete p;
    }

    Alpha::destroy();
    Delta::destroy();
    NoRad::destroy();
}

std::string check(bool test){
    if(!test){
        return "oopsie";
    }

    return "allGood";
}

bool AllTestPassed(vector<std::string>& result, int testDB){
    int tests = 0;
    for(std::string s : result){
        if(s=="allGood"){
            tests++;
        }
    }

    if(tests == testDB){
        return true;
    }

    return false;
}

int main()
{
    //PLANT LIFE SIMULATION
    Radiant* rad = Radiant::create();

    std::vector<Plant*> plants;
    int days = createPlanet("input.txt", plants);

    std::string ss;
    vector<std::string> report;
    daysLater(plants, rad, days, ss, report);

    for(std::string s : report){
        std::cout << s << std::endl;
    }
    std::cout << "--------------------" << std::endl;
    std::cout << ss << std::endl;
    destroyAll(plants, report);
```

KÖNYVTÁR

A terv osztálydiagramja:





FAGYIZÓ