PROGRAMOZÁS

Gömb-pont modellezés

Gregorics Tibor
http://people.inf.elte.hu/gt/prog

Feladat

Hány UFO tartózkodik az űrállomás közelében?

Programterv

$A = (\ v{:}UFO^n\ ,\ s{:}SpaceStation,\ \ c{:}\mathbb{N}\ )$

$Ef = (\ v = v' \wedge s = s'\ )$

$$Uf = (\ v = v' \wedge s = s' \wedge c = \sum_{i=1}^{n} 1$$

v[i] az s közelében van

c := 0	
i = 1 .. n	
v[i] az s közelében van	
c:= c + 1	–

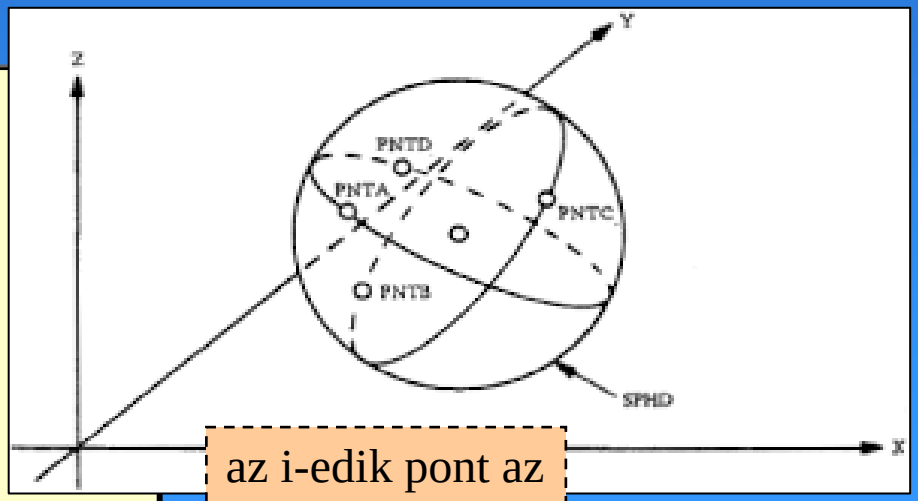
absztrakció

$A = (\ v{:}Point^n\ ,\ s{:}Sphere,\ \ c{:}\mathbb{N}\ )$

$Ef = (\ v = v' \wedge s = s'\ )$

$$Uf = (\ v = v' \wedge s = s' \wedge \ \ c = \sum_{i=1}^{n} 1$$

$v[i] \in s$



az i-edik pont az s gömbben van

Fő program

```
Sphere s;
...  // s tulajdonságainak beállítása

vector<Point> v(n);
...  // a v-beli pontok tulajdonságainak beállítása

int c = 0;
for(int i=0; i<n; ++i){
    if(s.contains(v[i])) ++c;
}

cout << "result: " << c << endl;
```

A C++ számára ismeretlen típusok ismeretlen művelettel.

Gömb típusa

a típus értékhalmaza:
egy Sphere típusú adat lehetséges értékei

$$\text{Sphere} = \{h \in 2^{\text{Point}} \mid \exists c \in \text{Point} \wedge \exists r \in \mathbb{R}: \forall p \in h: \text{distance}(c,p) \leqq r \}$$

a típus műveletei:
a gömbökkel végezhető művelet

gömbök	Benne van-e egy pont a gömbben: *l:=contains(s, p)* s : Sphere, p : Point, l : $\mathbb{L}$
centre : Point *radius* : $\mathbb{R}$	*l:= distance(s.centre, p)* $\leqq$ *s.radius*

Invariáns: radius >= 0.0

a típus értékeinek reprezentációja:
egy értéknek (gömbnek) a számítógép memóriájában történő ábrázolásához szükséges adatok

a típus műveleteinek implementációja:
a művelet működését leíró program, amely az eredeti típus érték (gömb) helyett az azt helyettesítő középponttal és sugárral dolgozik.

Pont típusa

a típus értékhalmaza:
egy Point típusú adat lehetséges értékei

Point = {térbeli pontok}

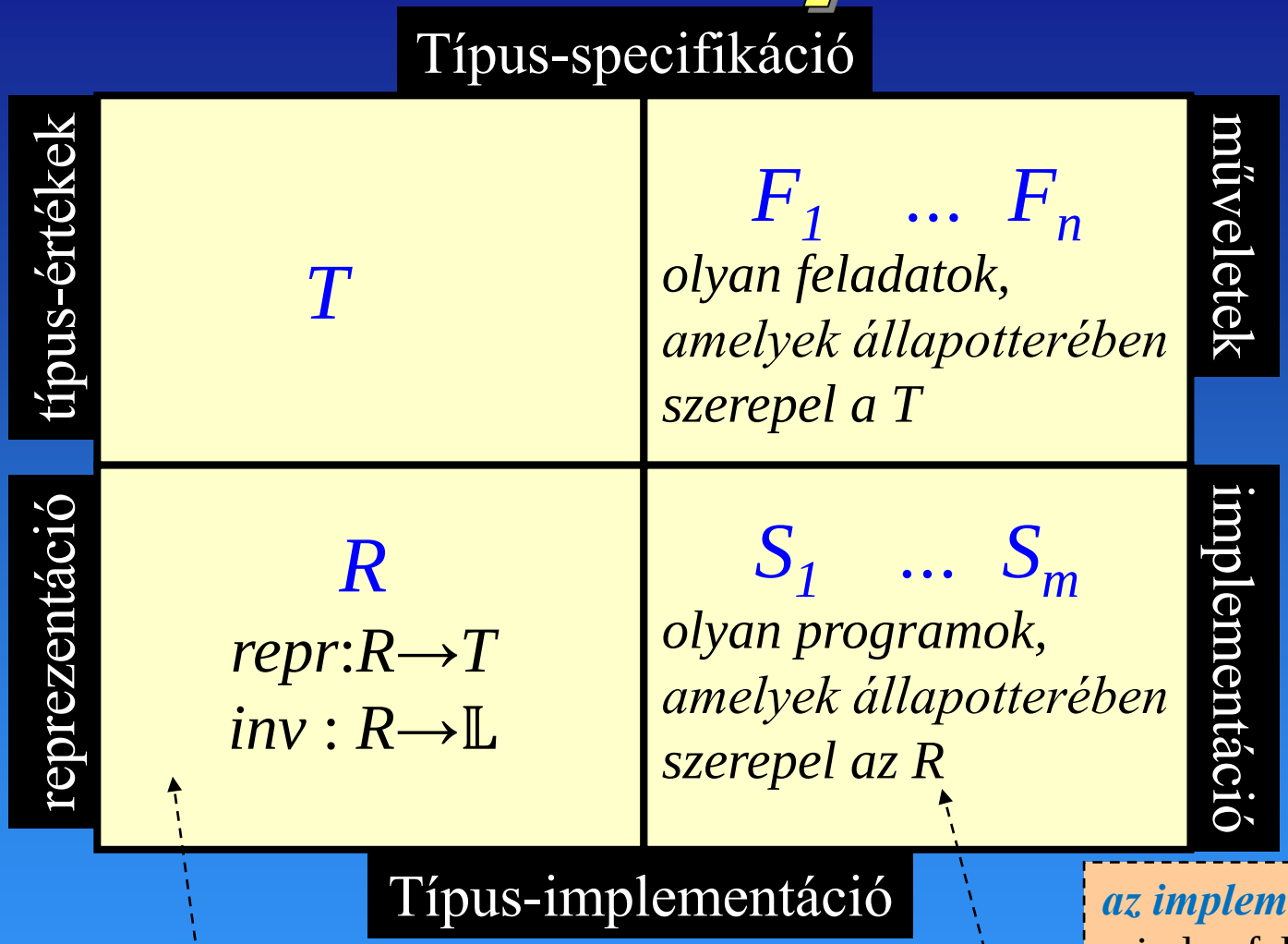
a típus műveletei:
a pontokkal végezhető művelet

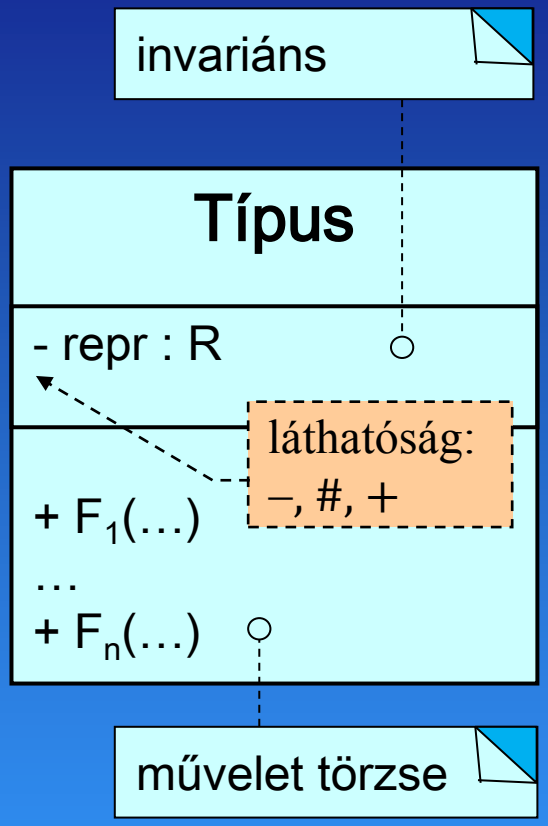
térbeli pontok	Két pont távolsága: $d:=distance(p_1, p_2)$ $\quad p_1, p_2$: Point, $d : \mathbb{R}$
$x, y, z : \mathbb{R}$	$d:=sqrt((p_1.x-p_2.x)^2+(p_1.y-p_2.y)^2+(p_1.z-p_2.z)^2)$

a típus műveleteinek implementációja:
a művelet működését leíró program, amely az eredeti típus érték (pont) helyett az azt helyettesítő koordinátákkal dolgozik.

a típus értékeinek reprezentációja:
egy értéknek (térbeli pontnak) a számítógép memóriájában történő ábrázolásához szükséges adatok

Típus

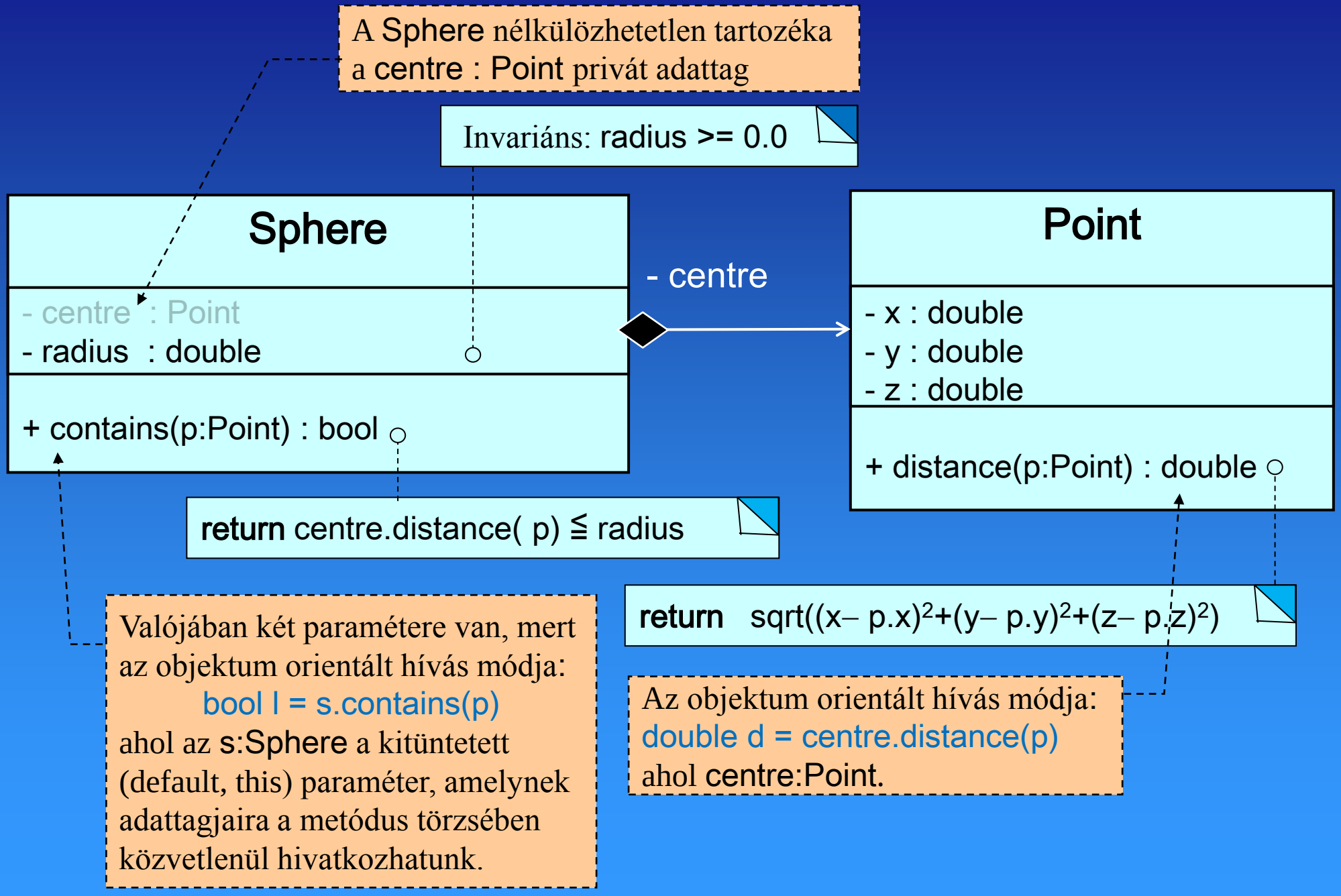




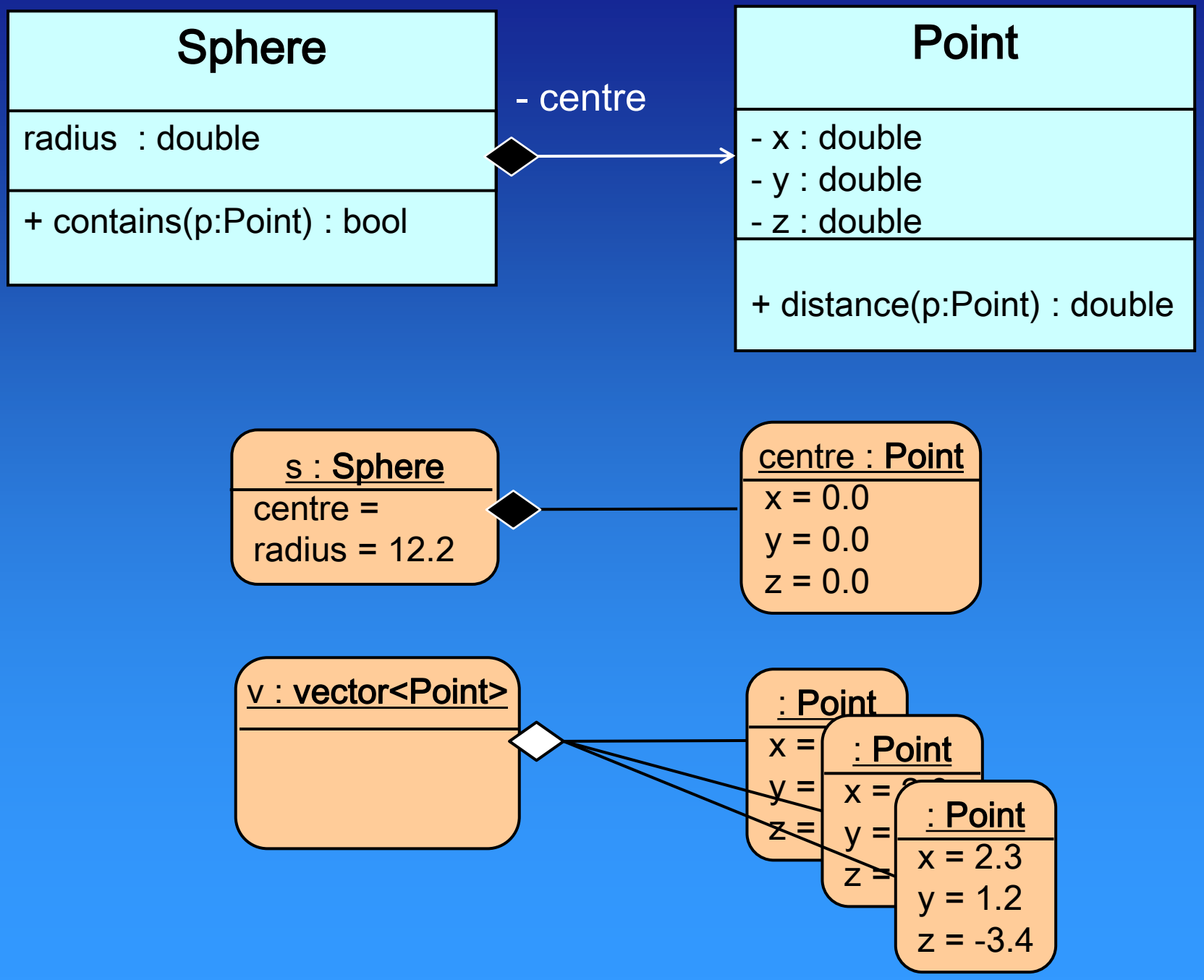
a reprezentáció helyessége:
minden *T*-beli érték helyettesíthető *inv*-et kielégítő *R*-beli adattal, és minden *inv*-et kielégítő *R*-beli adat helyettesít *T*-beli értéket.

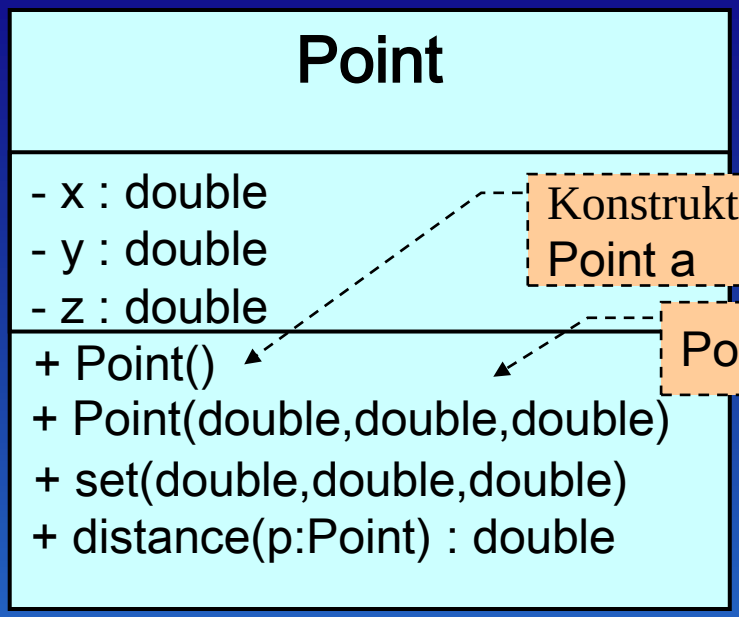
az implementáció helyessége:
minden feladatot megold egy program abban az értelemben, amilyen értelemben a típusértékeket *inv*-et kielégítő *R*-beli adatokkal lehet helyettesíteni.

Típusok osztálydiagramja



Kompozíciós modell





Konstruktor: Point a

a : Point
x =
y =
z =

Point b(2.3, 1.2, -3.4)

b : Point
x = 2.3
y = 1.2
z = -3.4

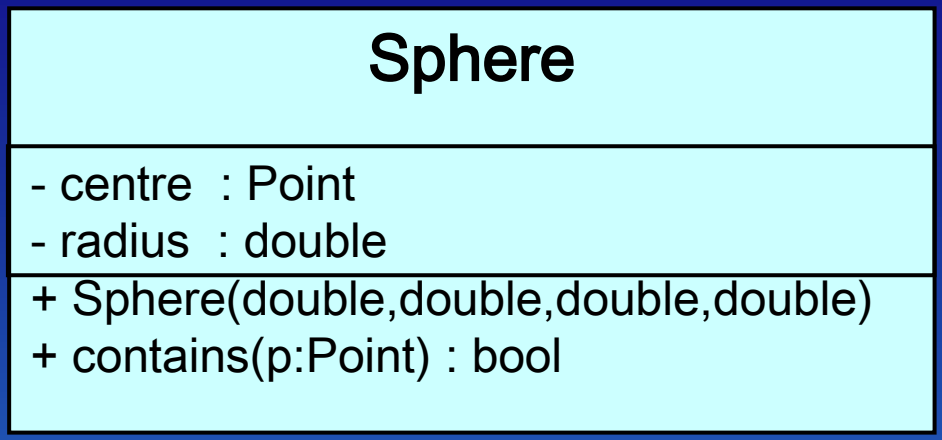
C++ osztály implicit módon mindig rendelkezik egy Pont(){} üres (default) konstruktorral, amíg nem definiálunk mást explicit módon.

```
class Point{
public:
    Point():_x(0.0), _y(0.0), _z(0.0) {}

    Point(double x, double y, double z) :_x(x),_y(y),_z(z) {}

    void set(double x, double y, double z){_x = x; _y = y; _z = z;}

    double distance(const Point &p) const{
        return sqrt(pow(_x-p._x,2)+pow(_y-p._y,2)+pow(_z-p._z,2));
    }
private:
    double _x, _y, _z;
};
```

másképp: Point() {set(0.0,0.0,0.0);}

Point(double x,double y,double z) {set(x,y,z);}

konstans metódus: nem változtatja meg az adattagokat




```
class Sphere {
public:
    enum Errors { NEGATIVE_RADIUS };

    Sphere(double x, double y, double z, double r) :_centre(x,y,z) {
        if(r<0) throw NEGATIVE_RADIUS; _radius = r;
    }

    bool contains(const Point &p) const {
        return _centre.distance(p)<= _radius;
    }
private:
    Point _centre;
    double _radius;
};
```

beágyazott felsorolás típus

Sphere(…) { _centre.set(x,y,z); … }

Point metódusait hívják

A konstruktor ellenőrzi az invariánst: hibás input esetén kivételt dob.

Fő program

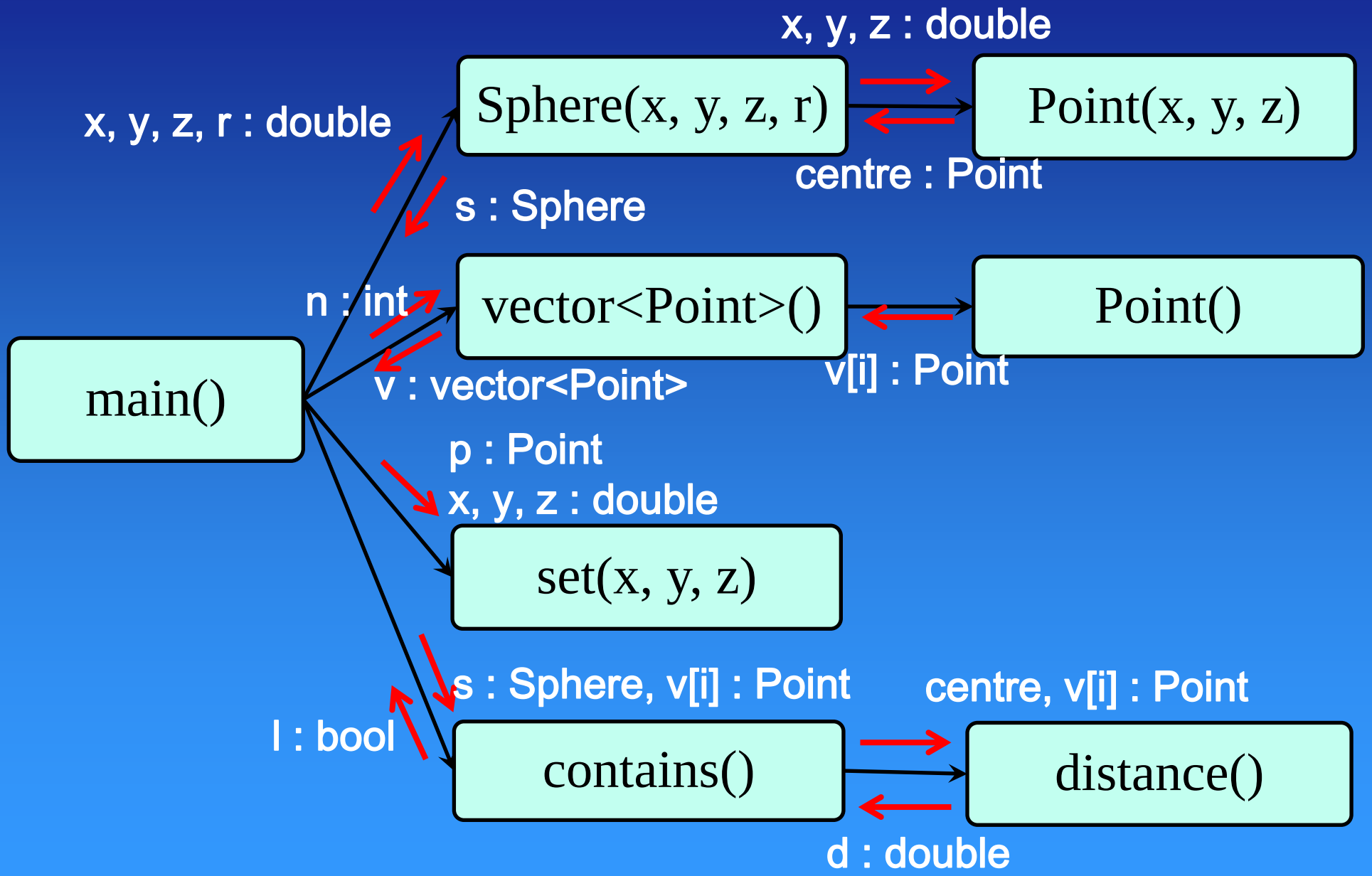
```
int main()
{
    double x,y,z,r;
    ...  // x,y,z,r  beolvasása
    try{
        Sphere s(x,y,z,r);
    }catch(Sphere::Errors err){
        if(err==Sphere::NEGATIVE_RADIUS) cout << ... ;
    }
    int n; cin >> n;
    // n ellenőrzése
    vector<Point> v(n);

    for(int i=0; i<n; ++i){
        cout << i+1 << ". points:" << endl;
        ... // x,y,z beolvasása
        v[i].set(x,y,z);
    }

    int c = 0;
    for(int i=0; i<n; ++i){
        if(s.contains(v[i])) ++c;
    }
    cout << "result: " << c << endl;
    return 0;
}
```

Sphere(x,y,z,r) azon belül Point(x,y,z)

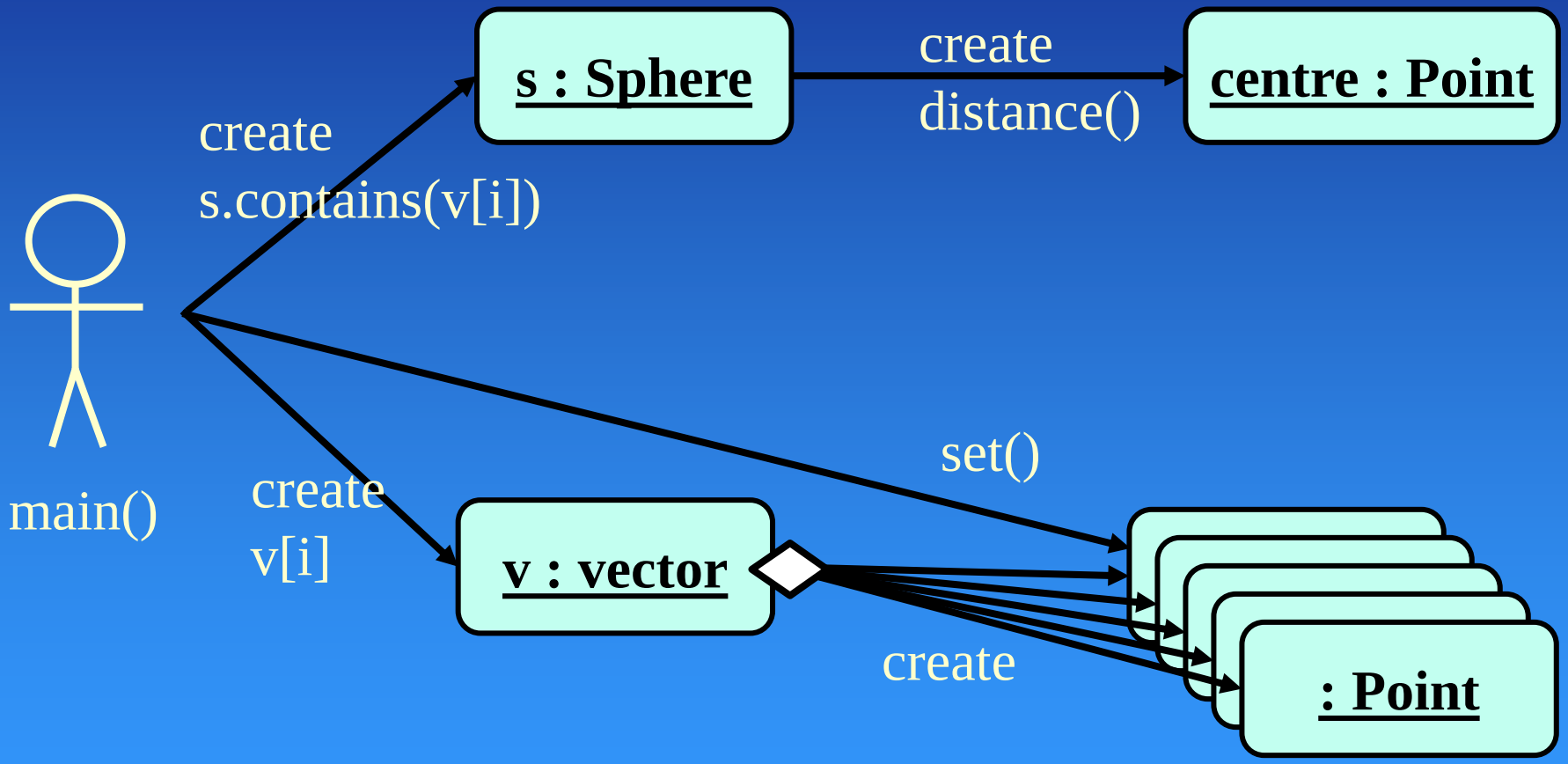
kivétel lekezelése

Point() *n*-szer

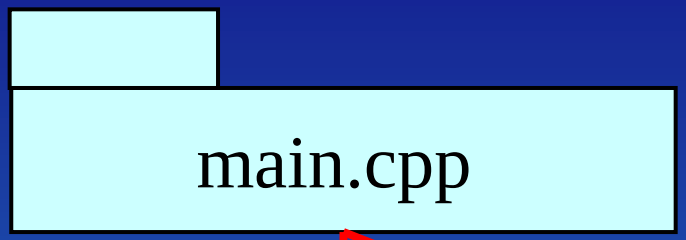
Modulszerkezet (procedurális szemlélet)

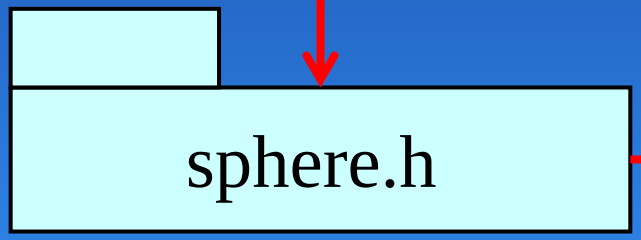


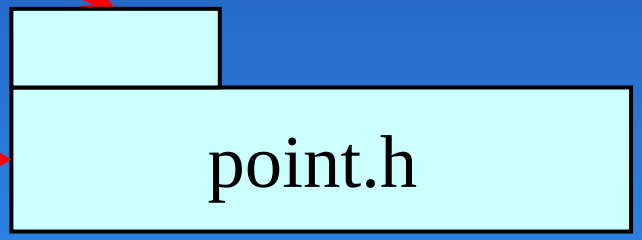
Együttműködési diagram (objektum orientált szemlélet)



Csomag szerkezet








```cpp
#include <iostream>
#include <vector>
#include "sphere.h"
#include "point.h"

using namespace std;

int main(){ … }
```

```cpp
#ifndef _SPHERE_H
#define _SPHERE_H

#include "point.h"
class Sphere{…};

#endif  // _SPHERE_H
```

```cpp
#ifndef _POINT_H
#define _POINT_H

class Point{…};

#endif  // _POINT_H
```

Fekete doboz tesztelés vázlata

Számlálás tesztje:

intervallum hossza: nulla, egy vagy több pont a tömbben
intervallum (1..n) eleje: csak a tömb legelső pontja esik bele a gömbbe
intervallum (1..n) vége: csak a tömb utolsó pontja esik bele a gömbbe
intervallum (1..n) túlindexelése: nagyon nagy sugarú gömb n=0 mellett
funkció szerint: 0, 1 vagy több pont is legyen a gömbben

Osztályok tesztje

- Egyenként teszteljük a metódusokat.
- Külön ügyelünk arra, hogy a metódusok (köztük a konstruktorok) nem sértik-e meg a típus-invariánst (ezt a konstans metódusoknál nen kell vizsgálni).
- Integrációs teszt: a metódusok tetszőleges variációinak végrehajtásait kell vizsgálni.

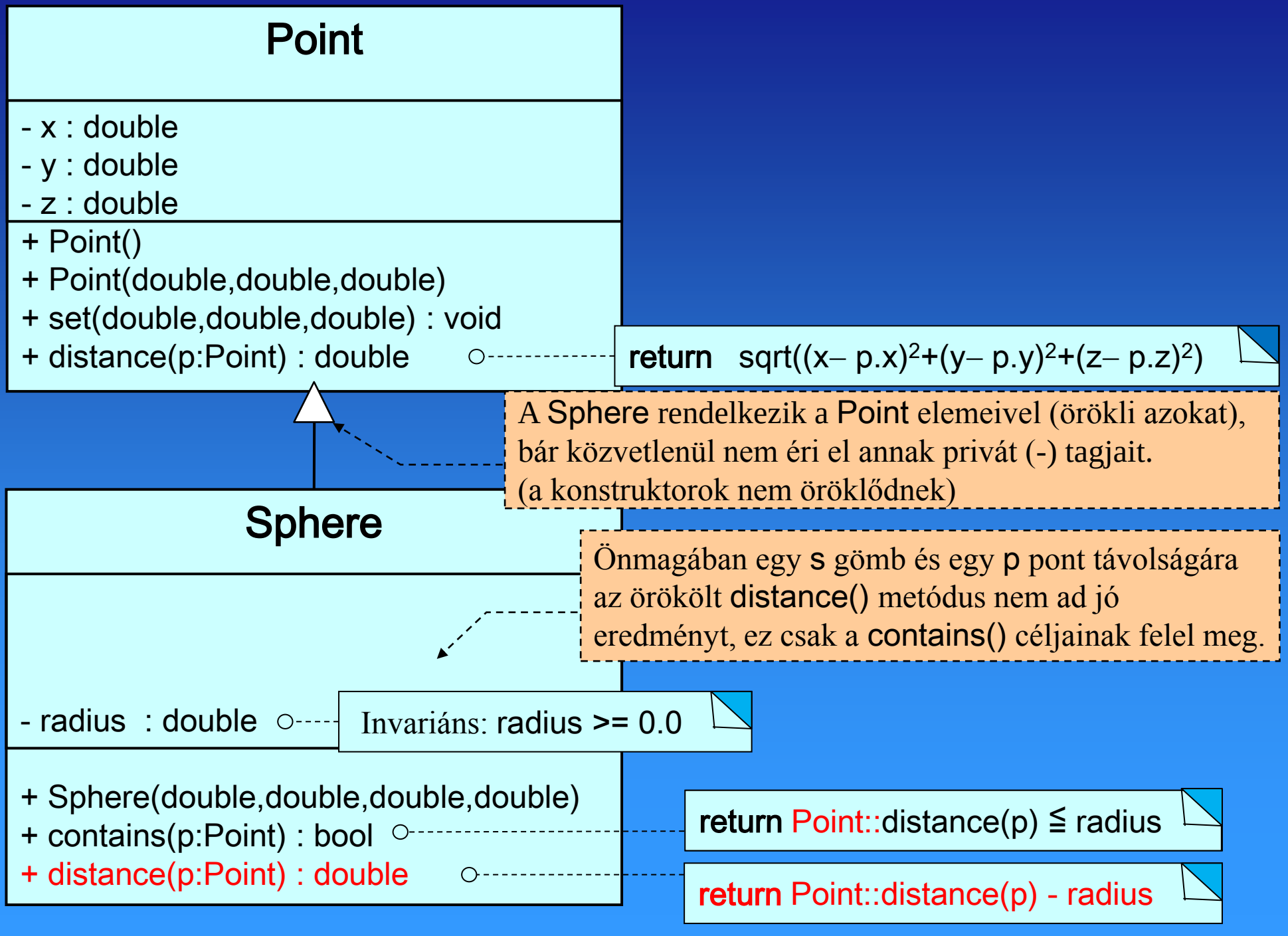
Egy másik modellje a feladat típusainak

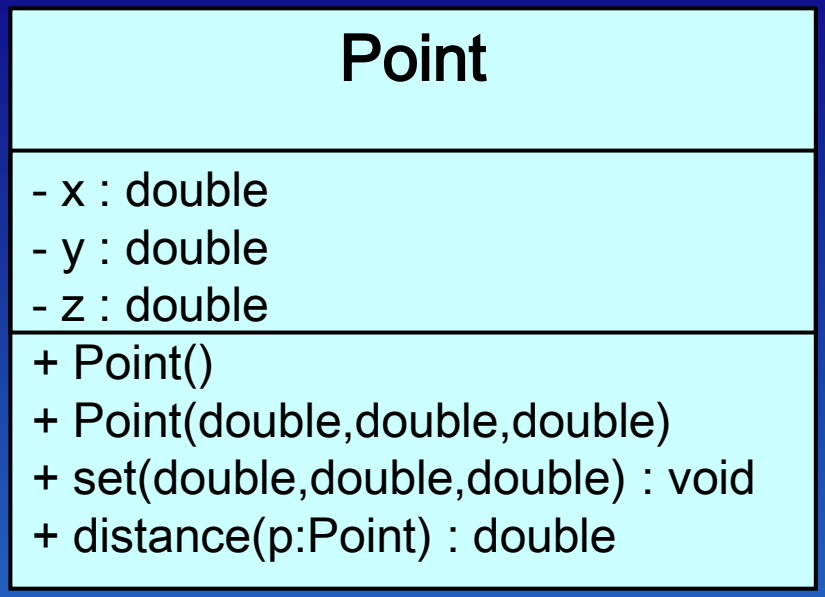
Point	Két pont távolsága: $d:=distance(p_1, p_2)$ p_1, p_2 : Point, d : $\mathbb{N}$
$x, y, z : \mathbb{R}$	$d:=sqrt((p_1.x-p_2.x)^2+(p_1.y-p_2.y)^2+(p_1.z-p_2.z)^2)$

Egy gömb legyen olyan, mint egy pont, csak legyen még neki sugara is és legyen egy contains() művelete.

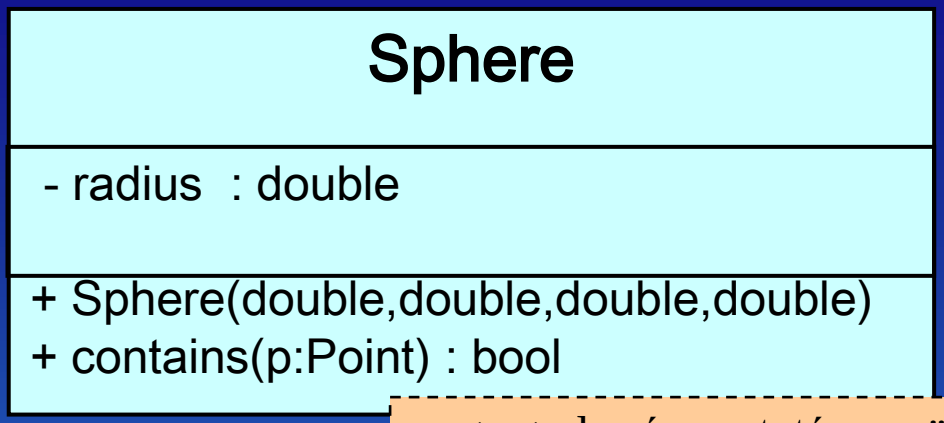
Sphere	Benne van-e egy pont a gömbben: $l:=contains(s, p)$ s : Sphere, p : Point, l : $\mathbb{L}$
az x, y, z : $\mathbb{R}$ -t örökli $x, y, z : \mathbb{R}$ $radius : \mathbb{R}$ Invariáns: radius >= 0.0	$l:= distance(s, p) \leqq s.radius$ a distance()-t örökli, ami egy gömbre és egy pontra a gömb középpontja és a pont közötti távolságot adja meg.

Származtatásos modell






```
class Point{
public:
    Point():_x(0.0), _y(0.0), _z(0.0) {}

    Point(double x, double y, double z) :_x(x),_y(y),_z(z) {}

    void set(double x, double y, double z){_x = x; _y = y; _z = z;}

    double distance(const Point &p) const{
        return sqrt(pow(_x-p._x,2)+pow(_y-p._y,2)+pow(_z-p._z,2));
    }
private:
    double _x, _y, _z;
};
```



protected származtatás: az öröklött publikus tagokat védetté teszi, így a distance() metódus is védett lesz, azaz kívülről gömbökre nem használható.

származtatás

Az ősosztály konstruktora hívódik meg az öröklött adattagok beállítására.

```
class Sphere : protected Point {
public:

    enum Errors { NEGATIVE_RADIUS };

    Sphere(double x, double y, double z, double r) : Point(x,y,z){
        if( r<0) throw NEGATIVE_RADIUS; _radius = r;
    }

    bool contains(const Point &p) const {
        return distance(p) <= _radius;
    }
private:
    double _radius;
};
```

Ugyanaz a fő program

```
int main()
{
    double x,y,z,r;
    ...  // x,y,z,r  beolvasása
    try{
        Sphere s(x,y,z,r);
    catch(Sphere::Errors err){
        if(err==Sphere::NEGATIVE_RADIUS) cout << ... ;
    }

    int n; cin >> n;
    vector<Point> v(n);
    for(int i=0; i<n; ++i){
        cout << i+1 << ". points:" << endl;
        ... // x,y,z beolvasása
        v[i].set(x,y,z);
    }

    int c = 0;
    for(int i=0; i<n; ++i){
        if(s.contains(v[i])) ++c;
    }
    cout << "result: " << c << endl;
    return 0;
}
```

Egy újabb modellje a feladat típusainak

Sphere

Két gömb távolsága:

$d:=distance(s_1, s_2)$ $\quad s_1, s_2$: Sphere, $d : \mathbb{N}$

Benne van-e egy pont a gömbben:

$l:=contains(s_1, s_2)$ $\quad s_1, s_2$: Sphere, $l : \mathbb{L}$

$x, y, z : \mathbb{R}$

$radius : \mathbb{R}$

$$d:=sqrt((s_1.x-s_2.x)^2+(s_1.y-s_2.y)^2+(s_1.z-s_2.z)^2)-$$
$$-s_1.radius-s_2.radius$$

$$l:= distance(s_1, s_2) - 2\cdot s_2.radius \leqq 0$$

Invariáns: radius >= 0.0

A pont egy speciális gömb, aminek a sugara nulla.

Point

Invariáns: radius = 0.0

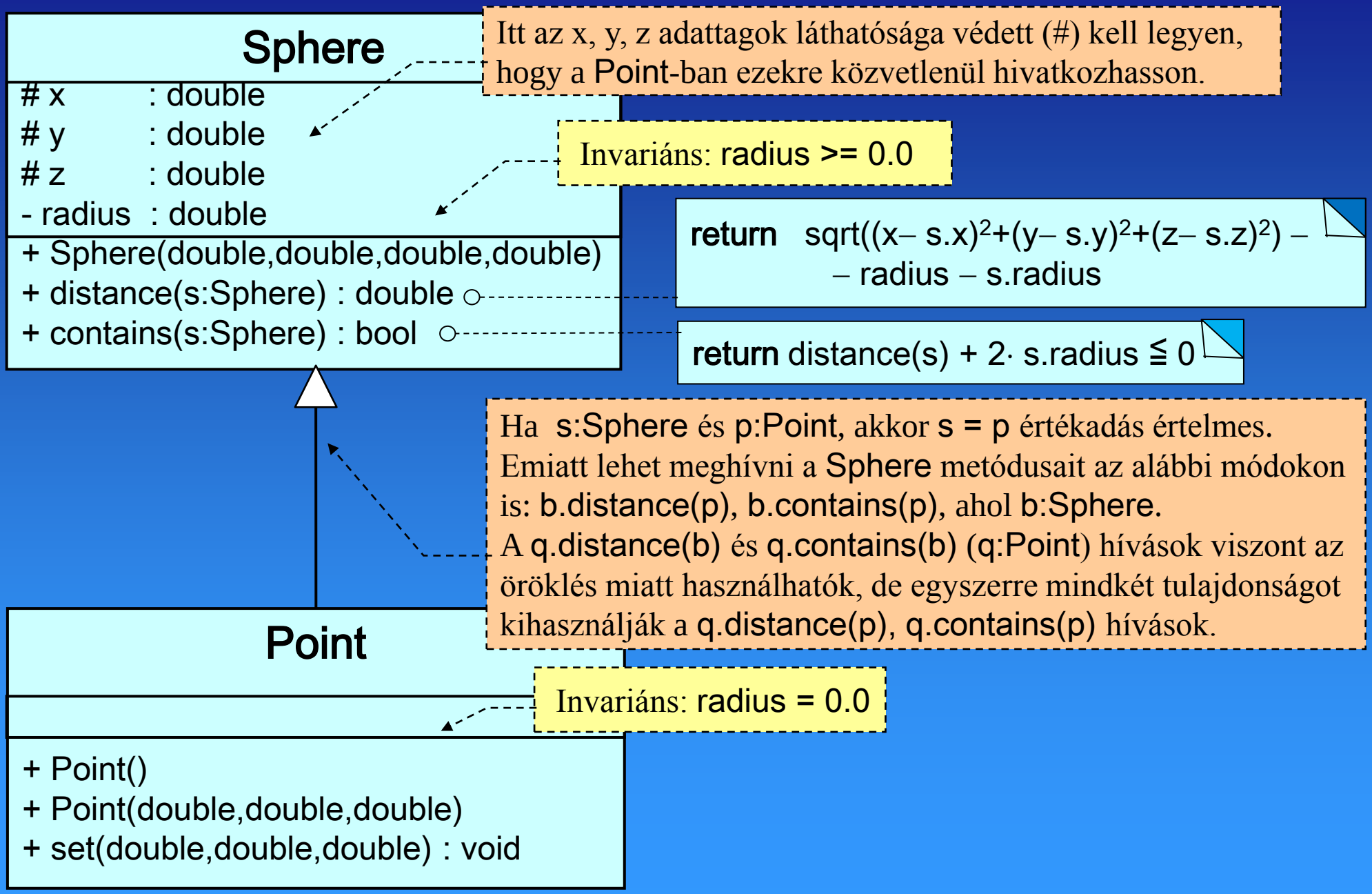
Az alábbi műveletek mind értelmesek:

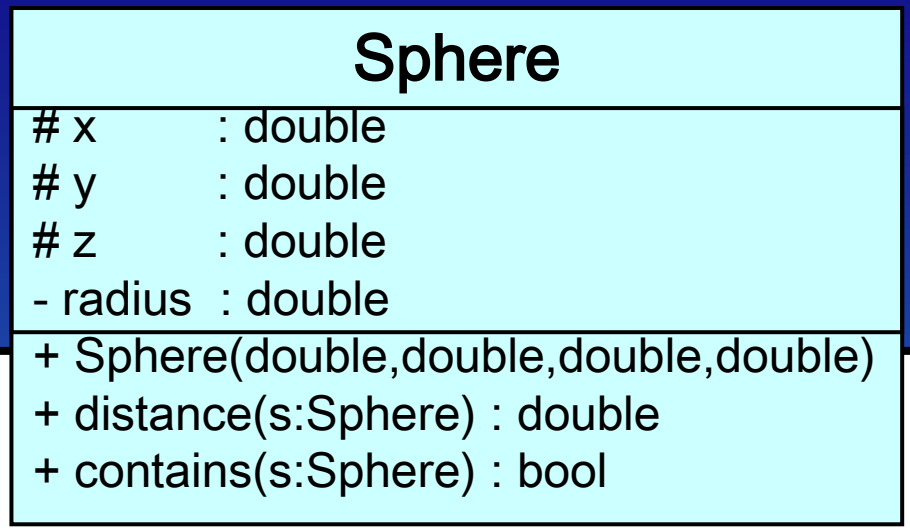
distance(s_1, s_2), distance(s_1, p_2), distance(p_1, s_2), distance(p_1, p_2)

contains(s_1, s_2), contains(s_1, p_2), contains(p_1, s_2), contains(p_1, p_2)

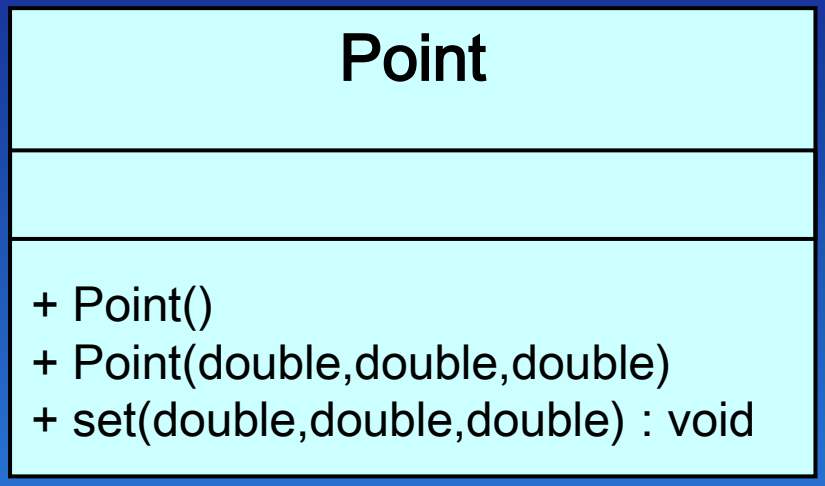
ahol s_1, s_2 : Sphere, p_1, p_2 : Point

Altípusos modell






```
class Sphere {
public:
    enum Errors { NEGATIVE_RADIUS };

    Sphere(double x, double y, double z, double r) : _x(x), _y(y), _z(z){
        if( r<0) throw NEGATIVE_RADIUS; _radius = r;
    }
    double distance(const Sphere &s)const {
        return sqrt(pow(_x-s._x,2) + pow(_y-s._y,2) + pow(_z-s._z,2))
                    - _radius - s._radius;
    }
    bool contains(const Sphere &s) const {
        return distance(s) + 2*s._radius <= 0;
    }
protected:
    double _x, _y, _z;
private:
    double _radius;
};
```

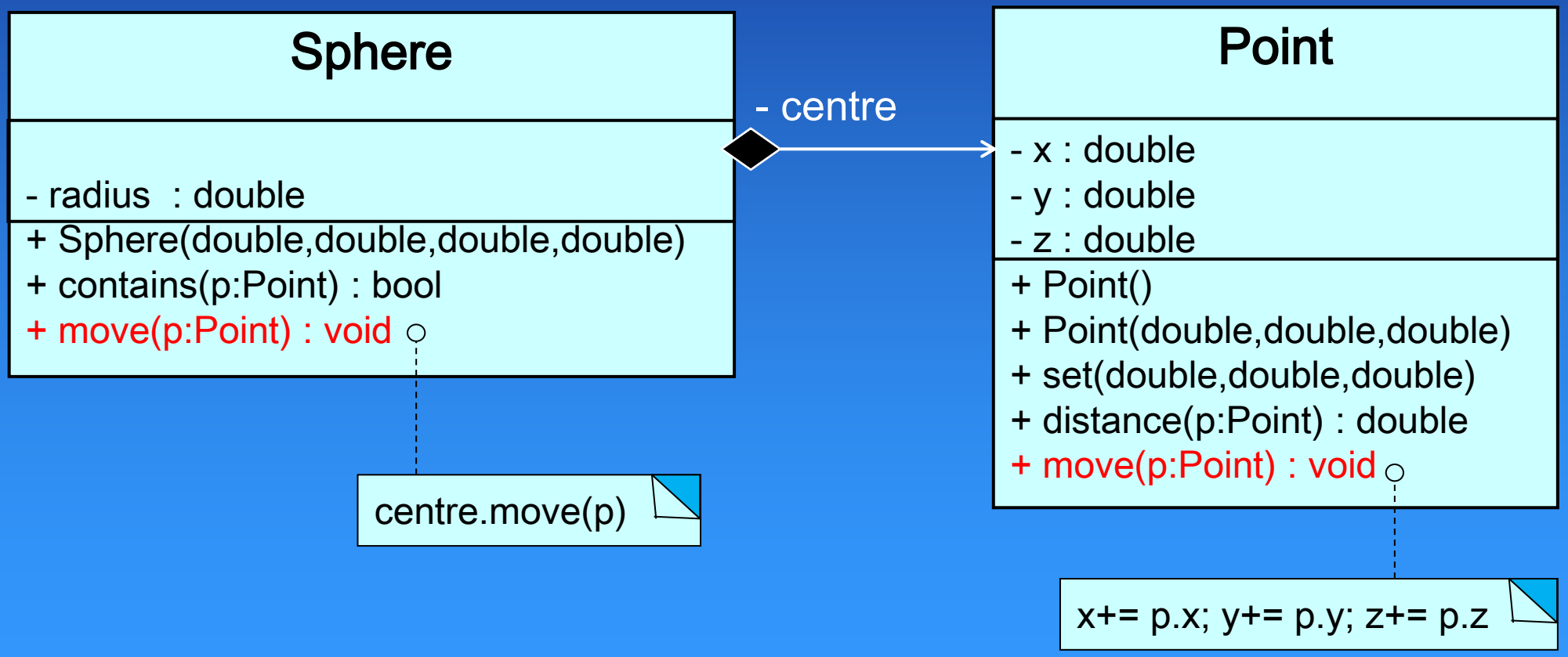



```
class Point : public Sphere {
public:
    Point():Sphere(0.0,0.0,0.0,0.0){}
    Point(double x, double y, double z):Sphere(x,y,z,0.0){}
    void set(double x, double y, double z){_x = x; _y = y; _z = z;}
};
```

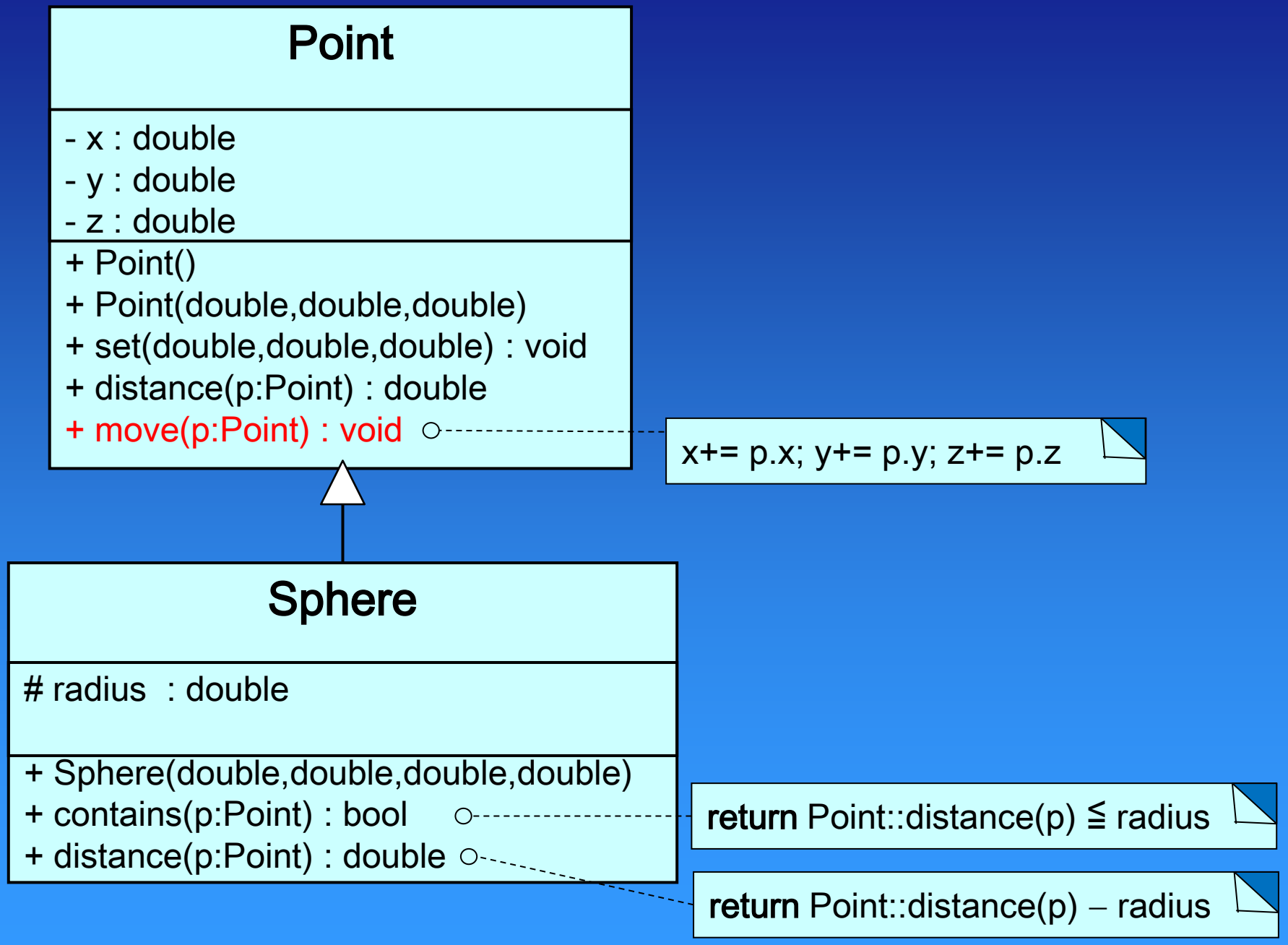
Ugyanaz a fő program

```
int main()
{
    double x,y,z,r;
    ...  // x,y,z,r  beolvasása
    try{
        Sphere s(x,y,z,r);
    catch(Sphere::Errors err){
        if(err==Sphere::NEGATIVE_RADIUS) cout << ... ;
    }

    int n; cin >> n;
    vector<Point> v(n);
    for(int i=0; i<n; ++i){
        cout << i+1 << ". points:" << endl;
        ... // x,y,z beolvasása
        v[i].set(x,y,z);
    }

    int c = 0;
    for(int i=0; i<n; ++i){
        if(s.contains(v[i])) ++c;
    }
    cout << "result: " << c << endl;
    return 0;
}
```

Kompozíciós modell eltolás művelettel kiegészítve



Származtatásos modell eltolás művelettel kiegészítve



Altípusos modell eltolás művelettel kiegészítve