

OO Programozás

Konstansok

Dr. Hatwagner F. Miklós

Széchenyi István Egyetem, Győr

https://github.com/wajzy/GKxB_INTM085

2023. október 11.

- *Konstans változó (paradoxon! → elnevezett konstans)*
- Memóriában helyezik el,
- azonnali inicializálást igényel,
- értéke megjelenik a nyomkövető programokban (debugger),
- csak olvasható (fordító biztosítja),
- van típusa (vö. #define).
- Láthatóságuk, hatókörük jobban szabályozható.

Tömbök definiálása, méret megadása

- konstans kifejezéssel,
- konstans kifejezéssel inicializált elnevezett konstanssal,
- vagy tetszőleges kifejezéssel C99 / C++14-től

```
#include <iostream>
#include "constHeader.h"
#define MEANING_M 42 // no type, cumbersome debugging

void array(int);

int main() {
    const int MEANING = 42;
    int meaningCopy = MEANING; // read
    // MEANING++; // error
    // MEANING = -42; // error
}
```

```

12 // size expressed with constant expression
13 int oldArray1[MEANING_M * sizeof(double)]; // OK
14 int oldArray2[MEANING * sizeof(double)]; // OK
15 array(21);
16 // std::cout << i << std::endl; // error
17 std::cout << ci << std::endl; // OK
18 std::cout << si << std::endl; // OK
19 }
20
21 void array(int size) {
22     // C99+ / C++14+ -> variable size is OK, allocated on stack
23     int newArray[size * sizeof(double)];
24 }

```

```
1 // int i = 1; // error, multiple definition of 'i'
2 const int ci = 2; // OK, visible only in defining file scope
3 static int si = 3; // OK, file scope
```

```
1 #include "constHeader.h"
```

- „Értékelj ki fordításkor!” → gyorsabb programok, lassabb fordítás
- Viszonylag egyszerű függvények készíthetők csak vele (C++11: csak 1 utasítás)
- Csak olyan globális változókra hivatkozhat, melyek konstansok
- Csak `constexpr` függvényt hívhat, akár önmagát is
- Értékkel kell visszatérnie
- Nem csak konstansokkal „hívható”, de akkor az eredmény nem használható konstans inicializálására
- C++11-ben még tiltott volt a prefix `++` operátor használata

```

3  constexpr double PI = 3.14159;
4  constexpr double deg2rad(double degree) {
5      return degree * PI / 180.;
6  }
7
8  int main() {
9      constexpr double rightAngle = deg2rad(90.);
10     std::cout << rightAngle << std::endl;
11     int deg180 = 180;
12     // the value of 'deg180' is not usable in a constant expression
13     // constexpr double rad180 = deg2rad(deg180);
14     double rad180 = deg2rad(deg180); // OK
15     std::cout << rad180 << std::endl;
16     return 0;
17 }

```

1000

```

3  constexpr int fibonacci1(int n) {
4      return (n <= 1) ? n : fibonacci1(n-1) + fibonacci1(n-2);
5  }
6
7  int main() {
8      constexpr int i = fibonacci1(32);
9      std::cout << i << std::endl;
10 }

```

```
$ time ./fibonacci1
2178309

real 0m0,005s
user 0m0,005s
sys 0m0,000s
```


[illegible]

- Az adattagok kvázi konstansok lesznek, amik inicializálása túl későn van a konstruktorban → *taginicializáló lista*
- Konstans adattagok és referenciák csak taginicializáló listával hozhatók létre.
- Ez egyébként használható lett volna a nem konstans adattagok inicializálására is.
- Példányosításkor a paramétereknek konstansnak kell lenniük.
- A tagfüggvények implicit inline-ok lesznek.

```

3  class Rectangle {
4      double mWidth;
5      double mHeight;
6  public:
7      constexpr Rectangle(double width, double height) :
8          mWidth(width), mHeight(height) {}
9
10     constexpr double getArea() const {
11         return mWidth * mHeight;
12     }
13
14     constexpr double getPerimeter() const {
15         return 2. * (mWidth + mHeight);
16     };
17 };

```

```
int main() {
    const int width = 5.;
    constexpr Rectangle r1(width, 3.);
    std::cout << "Area:_" << r1.getArea() << std::endl;
    std::cout << "Perimeter:_" << r1.getPerimeter() << std::endl;
}
```

- Maga a mutató nem konstans, nem *kell* inicializálni.
- Mutathat változóra, csak olvashatóvá téve azt.

- Nem tartalmazhatja elnevezett állandó címét, mert a védelem nem kerülhető meg.

constptr.cpp

```
14 int* pi = &i; // ok
15 // pi = &ci; // invalid conversion from 'const int*' to 'int*'
```

Konstans mutató egy változóra:

- Mivel a mutató konstans, inicializálni kell.
- A mutatott változó módosítható, de a mutató nem mutathat máshova.

constptr.cpp

```
18  int* const cpi = &i; // must be initialized
19  *cpi = 3; // OK
20  int i2 = 4;
21  // cpi = &i2; // assignment of read-only variable 'cpi'
```

```
24  const int* const cpci = &ci;
25  // *cpci = 4; // error
26  // cpci = &ci; // error
```



```
5  const int ci = 1;
6  int i = 2;
7  const int& rci = ci; // references must be initialized
8  // rci = 3; // assignment of read-only reference 'rci'
9  const int& rci2 = i; // read-only access to 'i'
10 // rci2 = 3; // assignment of read-only reference 'rci2'
11 const int& rci3 = 3; // OK
```

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡

Ha a függvény paramétere konstans, akkor a fv.-en belül sem változtatható meg az értéke, de a hívót ez nem érdekli (érték szerinti paraméterátadás).

De ha a paraméter mutató vagy referencia, akkor a fv. megváltoztathatná a változó értékét!

constfn.cpp

```
3 void pfn(int* pi) {  
4     *pi *= 2;  
5 }  
6  
7 void rfn(int& ri) {  
8     ri *= 2;  
9 }
```

constfn.cpp

```
11 void pcfnc(const int* pi) {  
12     // *pi *= 2; // assignment of read-only location '* pi'  
13 }  
14  
15 void rcfnc(const int& ri) {  
16     // ri *= 2; // assignment of read-only reference 'ri'  
17 }  
  
39 int main() {  
40     int i = 1;  
41     pcfnc(&i);  
42     rcfnc(i);  
43     std::cout << i << std::endl;  
44     const int ci = 5;  
45     // rcfnc(ci); // binding reference of type 'int&' to 'const int' discards ...
```

Ha a visszatérési érték típusa alaptípus, nincs haszna a `const`-nak (nem balérték).
De ha mutató vagy referencia, a visszatérési érték megváltoztatása tiltható `const`-tal.

constfn.cpp

```
19 int* fnp() {  
20     static int si = 10;  
21     return &si;  
22 }  
23  
24 const int* fncp() {  
25     static int si = 20;  
26     return &si;  
27 }
```

constfn.cpp

```
29 int& fnr() {  
30     static int si = 30;  
31     return si;  
32 }  
33  
34 const int& fncr() {  
35     static int si = 40;  
36     return si;  
37 }
```

```
39  int main() {
46      rcfn(ci); rcfn(5); pcf(&i); pcf(&ci); // ok
47      *fnp() = 11;
48      // *fncp() = 21; // assignment of read-only location '*fncp()'
49      fnr() = 31;
50      // fnr() = 41; // assignment of read-only location 'fnr()'
}
```

Példány is lehet konstans. Az adattagok többnyire eleve rejtettek, ezért kívülről nem írhatók. Nyilvános, konstans tag: getter elhagyható.

Konstans tagfüggvény konstans objektumon is hívható!

Rectangle11.cpp

```
3 class Rectangle {  
4     public:  
5         double mWidth; // bad idea  
6         const double mHeight;  
7  
8         Rectangle(double width, double height) : mHeight(height) {  
9             mWidth = width;  
10        }
```

Rectangle11.cpp

```
12     double getArea() const {
13         return mWidth * mHeight;
14     }
15
16     double getPerimeter() /* const */ {
17         return 2. * (mWidth + mHeight);
18     };
19 };
20
21 int main() {
22     const Rectangle r1(5., 3.);
23     // r1.mWidth = 55.; // assignment of member 'Rectangle::mWidth' in
24     std::cout << "Area: " << r1.getArea() << std::endl; // read-only object
25     // std::cout << "Perimeter: " << r1.getPerimeter() << std::endl;
26     // passing 'const Rectangle' as 'this' argument discards qualifiers
27 }
```


Konstans tagfüggvény is módosíthat egy adattagot, ha az `mutable`. Cél: érdemi állapotváltozást nem jelentő változások, pl. gyorsítótárak menedzselése.

Rectangle12.cpp

```
3 class Rectangle {  
4     mutable bool areaCached;  
5     mutable double area;  
6     mutable bool perimeterCached;  
7     mutable double perimeter;  
8 public:  
9     const double mWidth;  
10    const double mHeight;
```

Rectangle12.cpp

```
12     Rectangle(double width, double height) : mWidth(width), mHeight(height) {  
13         areaCached = perimeterCached = false;  
14     }  
15  
16     double getArea() const {  
17         if(not areaCached) {  
18             area = mWidth * mHeight;  
19             areaCached = true;  
20         }  
21         return area;  
22     }
```

(Sajnos) néhány további kivételes helyzetben is módosíthatja a `const` tagfüggvény a példány állapotát, pl. a tag struktúrát nem lehet lecserélni, de annak tagját már lehet módosítani (tranzitívan).

Egy objektumnak lehet beágyazott objektuma, ami szintén a taginicializáló listán keresztül inicializálható. Ha ezt nem tesszük meg → alapértelmezett konstruktor hívása, ha van ilyen.

Rectangle13.cpp

```
4  class Point {  
5      public:  
6          const double x;  
7          const double y;  
8  
9          Point(double x, double y) : x(x), y(y) {}  
10 };
```

Rectangle13.cpp

```
12  class Rectangle {
13      Point ul;
14      Point br;
15
16  public:
17      Rectangle(Point ul, Point br) : ul(ul), br(br) {}
18
19      double getArea() const {
20          return abs(ul.x - br.x) * abs(ul.y - br.y);
21      }
22
23      double getPerimeter() const {
24          return 2. * (abs(ul.x - br.x) + abs(ul.y - br.y));
25      };
26  };
```

Rectangle13.cpp

```
28  int main() {
29      Rectangle r1(Point(0, 3), Point(5, 0));
30      std::cout << "Area:_" << r1.getArea() << std::endl;
31      std::cout << "Perimeter:_" << r1.getPerimeter() << std::endl;
32  }
```

Feladat: készítsünk osztályokat egy átlagos alkalmazott, és egy programozó bérének kiszámítására!

Employee
- baseSalary : int
+ Employee(salary : int) «constructor»
+ getSalary() : int
+ setSalary(salary : int)

Programmer
- baseSalary : int
- bonusMult : double
+ Programmer(salary : int, bonus : double) «constructor»
+ getSalary() : int
+ setSalary(salary : int)
+ setBonus(bonus : double)

A program UML osztálydiagramja.

Employee

```
3 class Employee {
4     int baseSalary;
5     public:
6         Employee(int salary) {
7             baseSalary = salary;
8         }
9         int getSalary() const {
10             return baseSalary;
11         }
12         void setSalary(int salary) {
13             baseSalary = salary;
14         }
15 };
```

Programmer

```
17 class Programmer {
18     int baseSalary;
19     double bonusMult;
20     public:
21         Programmer(int salary, double bonus) {
22             baseSalary = salary;
23             bonusMult = bonus;
24         }
25         int getSalary() const {
26             return (int)(baseSalary * bonusMult);
27         }
28         void setSalary(int salary) {
29             baseSalary = salary;
30         }
31         void setBonus(double bonus) {
32             bonusMult = bonus;
33         }
34 };
```

main()

```
36  int main() {  
37      Employee e(300000);  
38      Programmer p(300000, 2.5);  
39      std::cout << e.getSalary() << std::endl;  
40      std::cout << p.getSalary() << std::endl;  
41  }
```

Kimenet

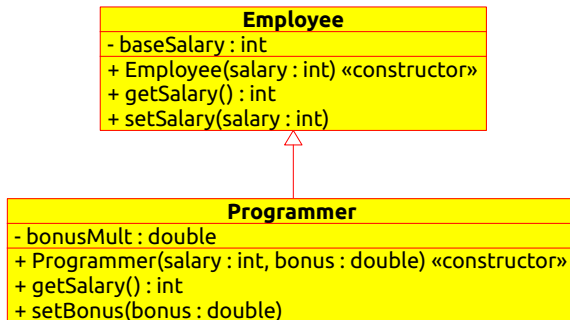
```
300000  
750000
```


Probléma:

- Hasonló feladatok következménye: kódismétlés (a programozó az alkalmazott egy speciális esete).

Megoldás:

- Szármasztatás/öröklés (inheritance)
- Ősosztály/szülő osztály (superclass, base class) → Employee
- Leszármasztott/szármasztott/gyerek osztály (subclass, derived class) → Programmer



- A leszármaszottakban (felül)definiáljuk azokat a függvényeket, amik másképpen fognak viselkedni (`getSalary()`), mint az ősből vagy teljesen hiányoztak (`setBonus()`).
- Minden más átöröklődik, kivéve a konstruktort. (Ettől még nem feltétlenül elérhetők ezek a leszármaszottban! → `private`)
- Ősosztálytól örökölt tagok inicializálása a leszármaszott dolga, pl. az ősből konstruktorának felhasználásával a taginicializáló listán. Ennek hiányában a fordító az ősből alapértelmezett konstruktorát hívja.

Employee

```
3 class Employee {
4     int baseSalary;
5     public:
6         Employee(int salary) {
7             baseSalary = salary;
8         }
9         int getSalary() const {
10             return baseSalary;
11         }
12         void setSalary(int salary) {
13             baseSalary = salary;
14         }
15 };
```

Programmer

```
class Programmer : public Employee {
    double bonusMult;
    public:
        Programmer(int salary, double bonus)
        : Employee(salary) {
            bonusMult = bonus;
        }
        int getSalary() const {
            // 'int Employee::baseSalary' is private
            // within this context
            // return (int)(baseSalary * bonusMult);
            return (int)(Employee::getSalary()
                * bonusMult);
        }
        void setBonus(double bonus) {
            bonusMult = bonus;
        }
};
```

17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34

main()

```
36  int main() {  
37      Employee e(300000);  
38      Programmer p(300000, 2.5);  
39      std::cout << e.getSalary() << std::endl;  
40      std::cout << p.getSalary() << std::endl;
```

Kimenet

```
300000  
750000
```

main()

```
42 // Using base class is OK: Programmer inherits from Employee
43 Employee* eArray[] = { &e, &p };
44 for(unsigned i=0; i<sizeof(eArray)/sizeof(eArray[0]); i++) {
45     // Ooops! Early / static binding!!!
46     std::cout << eArray[i]->getSalary() << std::endl;
47 }
48 }
```

Kimenet

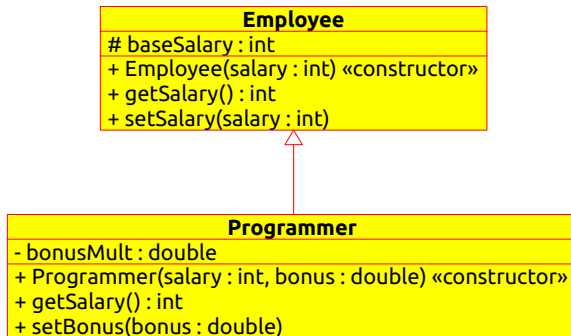
```
300000
300000
```

Probléma:

- Az ős privát tagjainak elérése körülményes a leszármazottból.

Megoldás:

- A védett (protected) tagok elérhetők a saját osztályukban és minden leszármazottban.
- Származtatásnál is használható ez és a `private` kulcsszó, bár szinte mindig `public`-ot használunk, azaz nem korlátozzuk tovább az öröklött láthatósági kategóriákat.



Probléma:

- Az ősrre utaló típusú, de a leszármazott egyik példányát címző mutatóval csak az ős függvényét tudjuk hívni.

Megoldás:

- Amikor egy objektum tagfüggvényét hívjuk, mindig eldönthető, hogy az ős vagy a leszármazott felüldefiniált függvényéről van-e szó, és fordítási időben meghatározható annak címe (*early/static bindig, korai/statikus kötés*).
Pl. `Programmer p(300000, 2.5); p.getSalary(); → 750000`
- Viszont egy objektum elérhető az ős típust címző mutatóval is (biztonságos, mert a leszármazott mindennel rendelkezik, amivel az őse). A korai kötés miatt az ősen definiált függvényt tudjuk csak elérni.

- A késői/dinamikus kötés (late/dynamic binding) hatására a fordító *virtuális metódustáblát* (VMT, vtable) hoz létre, melyben a leszármazottak felüldefiniált függvényeinek címei is benne vannak, melynek segítségével az aktuális objektum típusának megfelelő változat is hívható, típuskényszerítés nélkül.
Pl.: `Employee* pe = new Programmer(); pe->getSalary();` → 300000
- C++-ban késői kötést *virtuális tagfüggvényekkel* (`virtual`) hozhatunk létre.
- *Polimorfizmus*: mindig az ős típusát címző mutatót használjuk, de a vele hívott tagfüggvény osztályonként eltérően működik

Employee

```

3 class Employee {
4     protected: // private —> protected
5         int baseSalary;
6     public:
7         Employee(int salary) {
8             baseSalary = salary;
9         }
10        virtual int getSalary() const {
11            return baseSalary;
12        }
13        void setSalary(int salary) {
14            baseSalary = salary;
15        }
16 };

```

Programmer

```

class Programmer : public Employee {
    double bonusMult;
    public:
        Programmer(int salary, double bonus)
            : Employee(salary) {
            bonusMult = bonus;
        }
        // implicitly virtual
        /* virtual */ int getSalary() const {
        // OK to access protected member
            return (int)(baseSalary * bonusMult);
        }
        void setBonus(double bonus) {
            bonusMult = bonus;
        }
};

```

17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32

main()

```
35  int main() {  
  
41      Employee* eArray[] = { &e, &p };  
42      for(unsigned i=0; i<sizeof(eArray)/sizeof(eArray[0]); i++) {  
43          // OK! Late / dynamic binding  
44          std::cout << eArray[i]->getSalary() << std::endl;  
45      }  
46  }
```

Kimenet

```
300000  
750000
```

Induljunk ki **Rectangle12**-ből, majd

- a kerület/terület lekérdezést (minden síkidomnál hasonlóan elvégezhető tevékenység) válasszuk külön a számítások elvégzésétől,
- a méretek legyenek módosíthatóak, és
- készítsünk háromszög (**Triangle**) osztályt is!

Rectangle14.h

```
4  class Rectangle {  
5      mutable bool areaCached;  
6      mutable double area;  
7      mutable bool perimeterCached;  
8      mutable double perimeter;  
9      double mWidth;  
10     double mHeight;
```

Triangle14.h

```
class Triangle {  
    mutable bool areaCached;  
    mutable double area;  
    mutable bool perimeterCached;  
    mutable double perimeter;  
    double mA, mB, mC;
```

6
7
8
9
10
11

Rectangle14.h

```
11 public:
12     Rectangle(
13         double width, double height) {
14         mWidth = width;
15         mHeight = height;
16         invalidateCache();
17     }
18
19     void setWidth(double width);
20     void setHeight(double height);
21     double getArea() const;
22     double getPerimeter() const;
```

Triangle14.h

```
12 public:
13     Triangle(
14         double a, double b, double c) {
15         mA = a; mB = b; mC = c;
16         invalidateCache();
17     }
18
19     void setA(double a);
20     void setB(double b);
21     void setC(double c);
22     double getArea() const;
23     double getPerimeter() const;
```

Rectangle14.h

```
23 private:
24     // called from getArea(),
25     // must be const
26     void calcArea() const;
27     void calcPerimeter() const;
28
29     void invalidateCache() {
30         areaCached = perimeterCached
31             = false;
32     }
33 };
```

Triangle14.h

```
24 private:
25     // called from getArea(),
26     // must be const
27     void calcArea() const;
28     void calcPerimeter() const;
29
30     void invalidateCache() {
31         areaCached = perimeterCached
32             = false;
33     }
34 };
```


Rectangle14.cpp

```

3 void Rectangle::setWidth(double width) {
4     mWidth = width;
5     invalidateCache();
6 }

13 double Rectangle::getArea() const {
14     if(not areaCached) {
15         calcArea();
16         areaCached = true;
17     }
18     return area;
19 }

29 void Rectangle::calcArea() const {
30     area = mWidth * mHeight;
31 }

```

Triangle14.cpp

```

3 void Triangle::setA(double a) {
4     mA = a;
5     invalidateCache();
6 }

18 double Triangle::getArea() const {
19     if(not areaCached) {
20         calcArea();
21         areaCached = true;
22     }
23     return area;
24 }

34 void Triangle::calcArea() const {
35     double s = (mA + mB + mC) / 2.;
36     area = sqrt(s * (s - mA) * (s - mB)
37               * (s - mC));
38 }

```

main14.cpp

```
2 #include "Rectangle14.h"
3 #include "Triangle14.h"
4
5 int main() {
6     Rectangle rArray[] = {
7         Rectangle(1., 2.), Rectangle(2., 3.), Rectangle(3., 4.)
8     };
9     const int n = sizeof(rArray)/sizeof(rArray[0]);
10    for(int i=0; i<n; i++) {
11        std::cout << "Rectangle_#" << (i+1)
12                << "_Area:" << rArray[i].getArea()
13                << "_Perimeter:" << rArray[i].getPerimeter()
14                << std::endl;
15    }
```

Kimenet

```
Rectangle #1 Area: 2 Perimeter: 6
Rectangle #2 Area: 6 Perimeter: 10
Rectangle #3 Area: 12 Perimeter: 14
```

main14.cpp

```
16  const Triangle tArray[] = {
17      Triangle(3., 4., 5.), Triangle(5., 12., 13.), Triangle(7., 24., 25.)
18  };
19  const int m = sizeof(tArray)/sizeof(tArray[0]);
20  for(int i=0; i<m; i++) {
21      std::cout << "Triangle_#" << (i+1)
22                  << "_Area:" << tArray[i].getArea()
23                  << "_Perimeter:" << tArray[i].getPerimeter()
24                  << std::endl;
25  }
26  // const Shape sArray[] = { ... }; // ?!
27 }
```

Kimenet

```
Triangle #1 Area: 6 Perimeter: 12
Triangle #2 Area: 30 Perimeter: 30
Triangle #3 Area: 84 Perimeter: 56
```

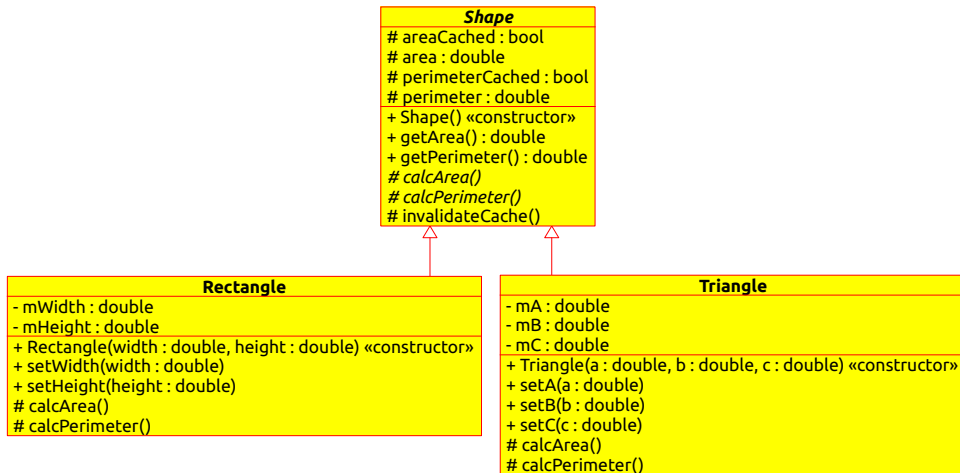
Probléma:

- Sok ismétlődő kódrészlet → szükség lenne egy általános síkidomra a származtatáshoz, de az milyen adatokkal írható le? Hogyan számolható ki pl. a kerülete?

Megoldás:

- *Absztrakt* (abstract) osztály: csak öröklési céllal hozzák létre, nem példányosítható a nem implementált, ún. *pure virtual* tagfüggvények (absztrakt metódus) miatt
- Ha egy absztrakt osztály kizárólag pure virtual tagfüggvényeket tartalmaz: *interfész* (interface) → elvárt szolgáltatáscsomag előírására

Absztrakt osztályok



Shape15.h

```
4  class Shape {
5      protected: // private —> protected
6          mutable bool areaCached;
7          mutable double area;
8          mutable bool perimeterCached;
9          mutable double perimeter;
10     public:
11         Shape() {
12             invalidateCache();
13         }
```

Shape15.h

```
15     double getArea() const;
16     double getPerimeter() const;
17     protected: // private —> protected
18         virtual void calcArea() const = 0;           // pure
19         virtual void calcPerimeter() const = 0;      // virtual
20         void invalidateCache() {
21             areaCached = perimeterCached = false;
22         }
23     };
```

Shape15.cpp

```
1  #include "Shape15.h"
2
3  double Shape::getArea() const {
4      if(not areaCached) {
5          calcArea();
6          areaCached = true;
7      }
8      return area;
9  }
```


Rectangle15.h

```
4 #include "Shape15.h"
5
6 // inheritance; base class -> subclass / derived class
7 class Rectangle : public Shape {
8     double mWidth;
9     double mHeight;
10 public:
11     // implicit call of base class's default constructor
12     Rectangle(double width, double height) {
13         mWidth = width;
14         mHeight = height;
15     }
```

Rectangle15.h

```
17     void setWidth(double width);
18     void setHeight(double height);
19 protected:
20     virtual void calcArea() const;
21     virtual void calcPerimeter() const;
22 };
```

Rectangle15.cpp

```
1 #include "Rectangle15.h"
2
3 void Rectangle::setWidth(double width) {
4     mWidth = width;
5     invalidateCache();
6 }
13
14 void Rectangle::calcArea() const {
15     area = mWidth * mHeight;
16 }
```

Triangle15.cpp

```
1 #include "Triangle15.h"
2
3 void Triangle::setA(double a) {
4     mA = a;
5     invalidateCache();
6 }
18
19 void Triangle::calcArea() const {
20     double s = (mA + mB + mC) / 2.;
21     area = sqrt(s * (s - mA)
22             * (s - mB)
23             * (s - mC));
24 }
```

main15.cpp

```
1 #include <iostream>
2 #include <typeinfo> // typeid() uses RTTI
3 #include "Shape15.h"
4 #include "Rectangle15.h"
5 #include "Triangle15.h"
6
7 int main() {
8     // cannot create array of abstract objects
9     Shape* sArray[] = {
10         // new calls constructor
11         new Rectangle(1., 2.), new Rectangle(2., 3.),
12         new Triangle(3., 4., 5.), new Triangle(5., 12., 13.)
13     };
14     const int n = sizeof(sArray)/sizeof(sArray[0]);
```

main15.cpp

```
15   for(int i=0; i<n; i++) {  
16       // name mangling  
17       std::cout << typeid(*sArray[i]).name() << "_#" << (i+1)  
18           << "_Area:_" << sArray[i]->getArea()  
19           << "_Perimeter:_" << sArray[i]->getPerimeter()  
20           << std::endl;  
21   }
```

Kimenet

```
9Rectangle #1 Area: 2 Perimeter: 6  
9Rectangle #2 Area: 6 Perimeter: 10  
8Triangle #3 Area: 6 Perimeter: 12  
8Triangle #4 Area: 30 Perimeter: 30
```

main15.cpp

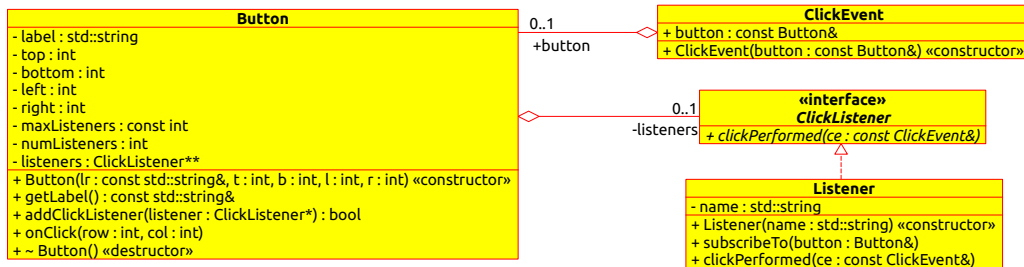
```
22     std::cout << (std::is_abstract<Shape>())
23         ? "Shape is abstract."
24         : "Shape is NOT abstract.")
25     << std::endl;
26     std::cout << (std::is_abstract<Rectangle>())
27         ? "Rectangle is abstract."
28         : "Rectangle is NOT abstract.")
29     << std::endl;
30 }
```

Kimenet

```
Shape is abstract.
Rectangle is NOT abstract.
```

Grafikus felületű, eseményvezérelt program modellje interfésszel

- Button: ha rákattintanak, ClickEvent eseményt küld egy ClickListenernek
- A ClickListenert megvalósító objektumnak fel kell iratkozni a Button eseményére



ClickEvent.h

```
4  class Button;  
5  
6  class ClickEvent {  
7      public:  
8          const Button& button;  
9          ClickEvent(const Button& button) : button(button) {}  
10 };
```

ClickListener.h

```
4  #include "ClickEvent.h"  
5  
6  class ClickListener {  
7      public:  
8          virtual void clickPerformed(const ClickEvent& ce) = 0;  
9  };
```


Listener.h

```
4 #include <string>
5 #include "ClickListener.h"
6 #include "Button.h"
7
8 class Listener : public ClickListener {
9     const std::string name;
10 public:
11     Listener(const std::string& name) : name(name) {}
12     void subscribeTo(Button& button) {
13         button.addClickListener(this);
14     }
15     virtual void clickPerformed(const ClickEvent& ce) {
16         std::cout << name << " received a click event from "
17             << ce.button.getLabel() << std::endl;
18     }
19 };
```

```
class Button {
    const std::string label;
    int top, bottom, left, right;
    const int maxListeners;
    int numListeners;
    ClickListener** listeners;
public:
    Button(const std::string& label, int t, int b, int l, int r)
        : label(label), maxListeners(10) {
        top = t;
        bottom = b;
        left = l;
        right = r;
        numListeners = 0;
        listeners = new ClickListener*[maxListeners];
    }
}
```

```
const std::string& getLabel() const {
    return label;
}
bool addClickListener(ClickListener* listener);
void onClick(int row, int col) const;
~Button() {
    delete[] listeners;
}
};
```

```
#include "Button.h"

bool Button::addClickListener(ClickListener* listener) {
    if (numListeners < maxListeners) {
        listeners[numListeners++] = listener;
        return true;
    }
    return false;
}

void Button::onClick(int row, int col) const {
    if (row >= top and row <= bottom and col >= left and col <= right) {
        for (auto i=0; i<numListeners; i++) { // auto-detected type
            listeners[i] -> clickPerformed(ClickEvent(*this));
        }
    }
}
```

eventMain.cpp

```
1 #include <iostream>
2 #include <string>
3 #include "Listener.h"
4 #include "Button.h"
5
6 int main() {
7     Button b1("Button1", 0, 100, 0, 100);
8     Button b2("Button2", 0, 100, 150, 250);
9     Listener l1("Listener1");
10    l1.subscribeTo(b1);
11    l1.subscribeTo(b2);
12    Listener l2("Listener2");
13    l2.subscribeTo(b1);
14    std::string row, col;
```

eventMain.cpp

```
15     while(std::cout << "row: ", std::getline(std::cin, row), !row.empty()) {
16         int r = stoi(row);
17         std::cout << "col: "; getline(std::cin, col);
18         int c = stoi(col);
19         b1.onClick(r, c);
20         b2.onClick(r, c);
21     }
22 }
```

Kimenet

```
row: 0
```

```
col: 0
```

```
Listener1 received a click event from Button1
```

```
Listener2 received a click event from Button1
```

```
row: 0
```

```
col: 150
```

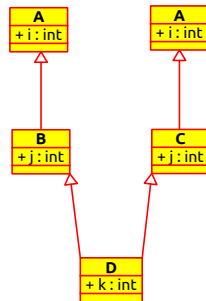
```
Listener1 received a click event from Button2
```

```
row: 0
```

```
col: 300
```

```
row:
```

- C++-ban létezik többszörös öröklődés (a legtöbb nyelv csak interfészekből enged többet megvalósítani, melyekre külön kulcsszó is van, pl. Java)
- Ha egy őstől több útvonalon is örököl egy leszármazott, nem lesz egyértelmű, melyiket kellene használni → virtuális öröklés (vagy a probléma elodázása `::-ral`)



diamond1.cpp

```
1  class A {
2      public:
3          int i;
4  };
5
6  class B : public A {
7      public:
8          int j;
9  };
10
11 class C : public A {
12     public:
13         int j;
14 };
```

```
// multiple inheritance
class D : public B, public C {
    public:
        int k;
};

int main(){
    D obj;
    obj.i = 42;
    return 0;
}
```

16
17
18
19
20
21
22
23
24
25
26

Kimenet

```
diamond1.cpp: In function 'int main()':
diamond1.cpp:26:9: error: request for member 'i' is ambiguous
    obj.i = 42;
        ^
diamond1.cpp:5:9: note: candidates are: int A::i
    int i;
        ^
diamond1.cpp:5:9: note:                  int A::i
```

diamond2.cpp

```
6 class B : virtual public A {  
7     public:  
8         int j;  
9 };  
10  
11 class C : virtual public A {  
12     public:  
13         int j;  
14 };
```

