ΠΡΟΓΡΑΜΜΑΤΙΣΤΙΚΕΣ ΤΕΧΝΙΚΕΣ

https://helios.ntua.gr/course/view.php?id=869

Διδάσκοντες:

Γιώργος Γκούμας
Άρης Παγουρτζής
Γιώργος Στάμου
Βασίλης Βεσκούκης
Θάνος Βουλόδημος
Γιώργος Αλεξανδρίδης
Γιώργος Σιόλας
Παρασκευή Τζούβελη

progtech@cslab.ece.ntua.gr

1/4/22

Διαφάνειες παρουσιάσεων

- Η γλώσσα προγραμματισμού C++
- Αρχές αντικειμενοστρεφούς προγραμματισμού
- Δομές δεδομένων

C++ Templates

- Γενικός προγραμματισμός (generic programming)
- Function templates: πράξεις που εφαρμόζονται σε (σχεδόν) οποιονδήποτε τύπο δεδομένων
- Class templates: τύποι δεδομένων που περιέχουν ή εξαρτώνται από (σχεδόν) οποιονδήποτε τύπο δεδομένων

Function templates

```
template <typename T>
void bubble_sort(int n, T a[]) {
  for (int i = 0; i < n-1; ++i)
    for (int j = n-2; j >= i; --j)
      if (a[j] > a[j+1]) {
        T t = a[j];
        a[j] = a[j+1];
        a[j+1] = t;
      }
}
```

- Απαιτείται `operator>` (σύγκριση)
  και `operator=` (ανάθεση)

Class complex, συμπληρωμένη

```
class complex {
private:
     double re, im;
public:
     complex(double, double);     // constructors
     complex(const complex &c);
     complex();
     ~complex();                  // destructor
     complex add(complex c);      // behaviour, exposed
     void print(ostream &out);
     void printN(ostream &out);
     double norm();
     double Re();                 // getters
     double Im();
     void setIm(double);          // setters
     void setRe(double);
     complex& operator=(const complex &y);   // overloaded
     bool operator>(complex c);
     friend ostream& operator<<(ostream &out, complex c);
};
```

δημιουργία

ανάθεση

σύγκριση

εκτύπωση

Function templates

```
...
// needed by the constructor that
// generates random complex numbers
srand(time(NULL));

complex c[10];

cout << "Unsorted random complex numbers\n";
for (int i=0; i<10; ++i)
      c[i].printN(cout);

bubble_sort(10, c);

cout << "Sorted\n";
for (int i=0; i<10; ++i)
      c[i].printN(cout);
```

```
Unsorted random complex numbers
24+25i  34.6554
59+59i  83.4386
48+52i  70.7672
78+81i  112.45
22+19i  29.0689
13+1i  13.0384
74+25i  78.1089
12+52i  53.3667
49+33i  59.0762
91+63i  110.68
```

```
Sorted
13+1i  13.0384
22+19i  29.0689
24+25i  34.6554
12+52i  53.3667
49+33i  59.0762
48+52i  70.7672
74+25i  78.1089
59+59i  83.4386
91+63i  110.68
78+81i  112.45
```

Class templates

- Μυστικό <ο,τιδήποτε>

```
template <typename T>
class secret {
private:
    string password;
    T secretData;
public:
    secret(const string &pwd, const T &d):
        password(pwd), secretData(d) {};

    T get(const string &pwd) {
        if (pwd == password) return secretData;
        cout << "Wrong password!" << endl;
        exit(0);
    }
};
```

Class templates

```
template <typename T>
class secret {
private:
    string password;
    T secretData;
public:
    secret(const string&, const T&);
    T get(const string&);
};

template <typename T>
secret<T>::secret(const string &pwd, const T &d):
    password(pwd), secretData(d) {};

template <typename T>
T secret<T>::get(const string &pwd) {
    if (pwd == password) return secretData;
    cout << "Wrong password!" << endl;
    exit(0); }
```

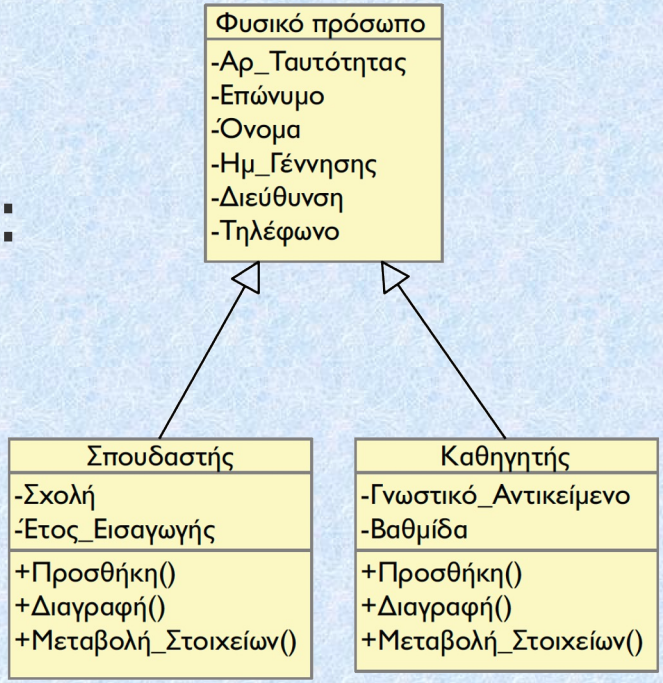
Ορισμός εκτός της κλάσης

Class templates

```
int main() {
  string secretPassw = "ntuaece4ever";
  string enteredPassw, secretString;
  int secretInteger;

  cout << "a secret int: "; cin >> secretInteger;
  cout << "a secret string: "; cin >> secretString;
  cout << "passw: "; cin >> enteredPassw;

  secret<int> s1(secretPassw, secretInteger);
  cout << s1.get(enteredPassw);

  secret<string> s2(secretPassw, secretString);
  cout << s2.get(enteredPassw);
}
```

Κληρονομικότητα (inheritance)

- Η δυνατότητα ορισμού κλάσεων που αποκτούν (κληρονομούν) όλα τα κληροδοτούμενα χαρακτηριστικά μιας ή περισσοτέρων κλάσεων
  - Δεν κληροδοτούνται όλα τα μέλη μιας κλάσης
  - Απλή κληρονομικότητα:
    1 κλάση-γονέας
  - Πολλαπλή κληρονομικότητα:
    >1 κλάσεις-γονείς



Κληρονομικότητα

- Επαναχρησιμοποίηση πηγαίου κώδικα
  - Οι δηλώσεις των πεδίων και των μεθόδων ξαναχρησιμοποιούνται, χωρίς να χρειάζεται να επαναληφθούν
  - Ο κώδικας μπορεί να έχει κατασκευαστεί σε κάποιο άλλο έργο

Υπενθύμιση!

Αρχή της μοναδικής ευθύνης (Single Responsibility Principle)

- Μια κλάση πρέπει να κάνει σωστά κάτι, για το οποίο να φέρει την αποκλειστική ευθύνη μέσα σε ένα πρόγραμμα
- Δηλαδή να μην επαναλαμβάνεται κώδικας, όταν αυτό μπορεί να αποφευχθεί

Ορατότητα μελών και κληρονομικότητα

```
class [όνομα] {
```

public:

Μέλη ορατά εντός και εκτός της κλάσης

private: `// default!`

Μέλη ορατά μέσα στην κλάση και σε φίλες συναρτήσεις και κλάσεις

protected:

Μέλη ορατά μέσα στην κλάση, σε φίλες συναρτήσεις και κλάσεις και σε κλάσεις που τα κληρονομούν (κλάσεις-παιδιά)

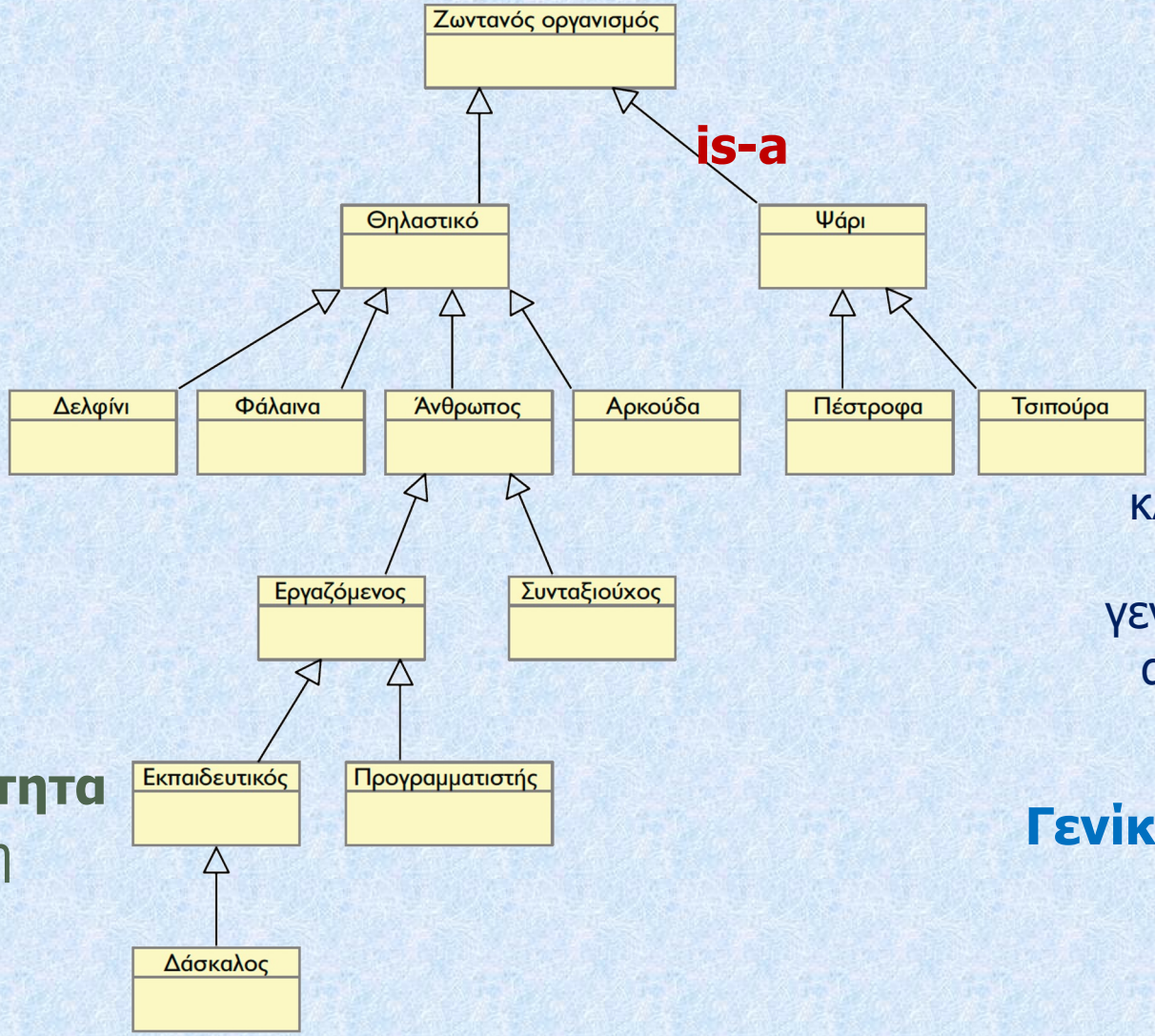
```
};
```

Τύποι κληρονομικότητας

- **public**: Η κλάση-παιδί κληρονομεί τα public και protected μέλη της κλάσης-γονέα ως public και protected μέλη, αντίστοιχα, ισχύει το "is-a"
- **protected**: Η κλάση-παιδί κληρονομεί τα public και protected μέλη της κλάσης-γονέα ως protected μέλη, δεν ισχύει το "is-a"
- **private**: Η κλάση-παιδί κληρονομεί τα public και protected μέλη της κλάσης-γονέα ως private μέλη, δεν ισχύει το "is-a"

- Τα private μέλη ΔΕΝ κληρονομούνται

Κληρονομικότητα και γενίκευση



Οι εξειδικευμένες κλάσεις κληρονομούν τα χαρακτηριστικά των γονέων τους

Κληρονομικότητα

Εξειδίκευση

Κάθε κλάση που κληρονομείται είναι γενικότερη από αυτή που την κληρονομεί

Γενίκευση (is-a)

Υλοποίηση κληρονομικότητας

```
class parent_class
{
  public:
        ...
  protected:
        ...
  private:
        ...
};
```

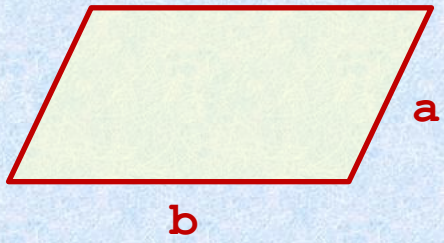
Αφορά την κληρονομικότητα, όχι την ορατότητα μελών της κλάσης!

`public, protected, private`

```
class child_class : access_modifier parent_class
{
  public:
        ...
  protected:
        ...
  private:
        ...
};
```

Κληρονομικότητα, παράδειγμα

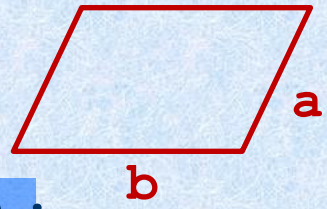
```
class myRectangle {
private:
    double a, b;
public:
    myRectangle();
    myRectangle(double, double);
    double A();
    double B();
    void setA(double);
    void setB(double);
    double perimeter();
    double area();
    void print(ostream&);
};
```



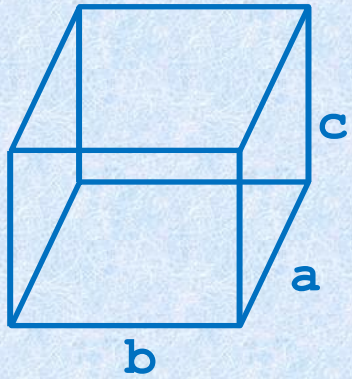
Κληρονομικότητα, παράδειγμα

re-use change new

```
class myCuboid {
private:
    double a, b, c;
public:
    myCuboid();
    myCuboid(double, double, double);
    double A(); double B(); double C();
    void setA(double);
    void setB(double);
    void setC(double);
    double perimeter();
    double aArea();
    double volume();
    void print(ostream&);
};
```



is-a



Κληρονομικότητα, παράδειγμα

```
class myRectangle {
protected:
    double a, b;
public:
    myRectangle() ;
    myRectangle(double,double);
    double A();
    double B();
    void setA(double A);
    void setB(double B);
    double perimeter();
    double area();
    void print(ostream &out );
};
```

κληρονομημένα

επισκιάζοντα

επισκιασμένα

```
class myCuboid : public myRectangle {
protected:
    double c;
public:
    myCuboid();
    myCuboid(double,double,double);
    double perimeter();
    double area();
    double volume();
    double C();
    void setC(double);
    void print(ostream&);
};
```

Κληρονομικότητα - constructors

- Κατασκευαστές (constructors)

```
myRectangle::myRectangle() :a(0),b(0) {};
```

default constructor κλάσης-γονέα

```
myCuboid::myCuboid() : c(0) {};
```

default constructor κλάσης-παιδί

```
myCuboid c;
c.print(cout);    //a=0, b=0, c=0, area=0, perimeter=0, volume=0
```

Κληρονομικότητα - constructors

- Κατασκευαστές (constructors)

```
myRectangle::myRectangle() :a(0), b(0) {};
```

constructor κλάσης-γονέα

```
myCuboid::myCuboid(double A, double B, double C) :
myRectangle(A, B), c(C) {};
```

constructor κλάσης-παιδί

```
myCuboid c(1,2, 3);
c.print(cout);
//a=1, b=2, c=3, area=22, perimeter=24, volume=6
```

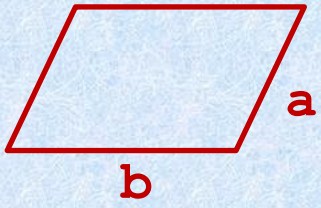
Κληρονομικότητα, παράδειγμα

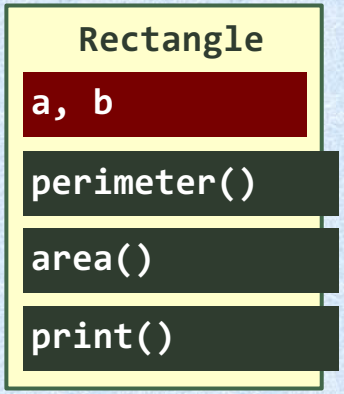
Subtyping

```
myCuboid c(2, 3, 4);
c.print(cout);

myRectangle *mr = &c;
mr->print(cout);            // τι τυπώνει;;;
```

Πολλαπλή κληρονομικότητα

- Η κλάση Rectangle

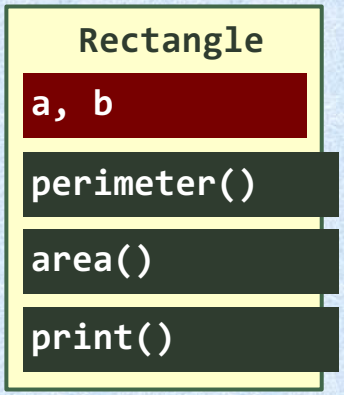





```
class Rectangle {
protected:
    double a, b;
public:
    Rectangle;
    Rectangle(double A, double B);

    double A();
    double B();
    void setA(double A);
    void setB(double B);

    double perimeter();
    double area();
    void print(ostream &out);

    friend ostream& operator<<(ostream &out, Rectangle r
};
```

Πολλαπλή κληρονομικότητα

- Η κλάση Rectangle

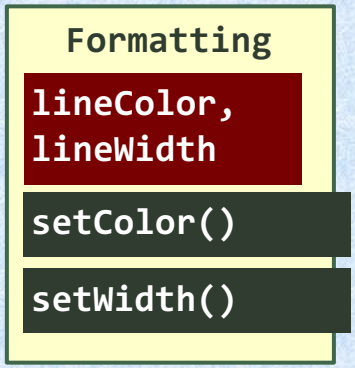



```
class Rectangle {      // The parent-class
protected:
    double a, b;
public:
  Rectangle() : a(0), b(0) {} ;
  Rectangle(double A, double B) : a(A), b(B) {}
  double A() {return a;}
  double B() {return b;}
  void setA(double A) { a = A; }
  void setB(double B) { b = B; }
  double perimeter() { return 2*(a+b); }
  double area() { return a*b; }
  void print(ostream &out) {out << "[a=" << a
        << " b=" << b << " a=" << area()
        << " p=" << perimeter() << "]\n";}
  friend ostream& operator<<(ostream &out, Rectangle r) {
    out << "[a=" << r.a << " b=" << r.b << " a="
        << r.area() << " p=" << r.perimeter() << "]";
    return out; }
};
```

```
int main (){
  Rectangle r(3, 10);
  r.print(cout);
  r.setA(15);
  r.print(cout);
}

[a=3 b=10 a=30 p=26]
[a=15 b=10 a=150 p=50]
```

Πολλαπλή κληρονομικότητα

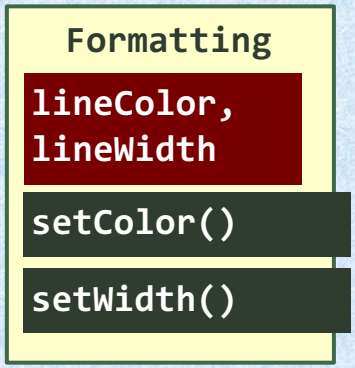
- Η κλάση Formatting




```
class Formatting {
protected:
  int lineWidth;
  string lineColor;
public:
  Formatting();
  Formatting(int W, string C);

  void setColor(string c);
  void setWidth(int w);
  friend ostream& operator<<(ostream &out, Formatting d);
};
```

Πολλαπλή κληρονομικότητα

- Η κλάση Formatting



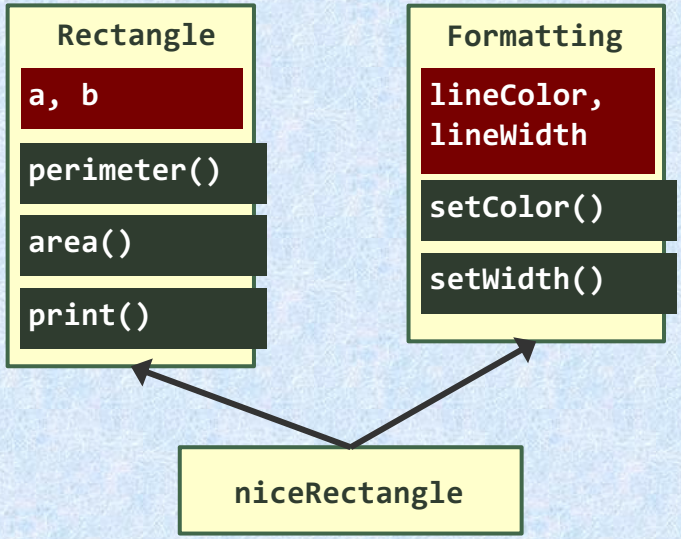

```
class Formatting {
protected:
  int lineWidth;
  string lineColor;
public:
  Formatting(): lineWidth(0), lineColor("transparent") {};
  Formatting(int W, string C): lineWidth(W), lineColor(C) {};

  void setColor(string c) { lineColor = c;}
  void setWidth(int w) { lineWidth = w; }

  friend ostream& operator<<(ostream &out, Formatting d) {
    out << "[color="<<d.lineColor<<" width="<< d.lineWidth<<"]";
    return out;
  }
};
```

```
int main (){
  Formatting f1(1,"blue");
  cout << f1 << endl;
  f1.setColor("red");
  cout << f1; }
```

```
[color=blue width=1]
[color=red width=1]
```

Πολλαπλή κληρονομικότητα

- Μια κλάση κληρονομεί από περισσότερες




```
class niceRectangle : public Formatting, public Rectangle {
public:
  niceRectangle();
  niceRectangle(double, double, int, string);
  void print(ostream &out );
  friend ostream& operator <<(ostream&, niceRectangle); };
```

Πολλαπλή κληρονομικότητα

```
class niceRectangle : public Formatting, public Rectangle {
public:
  niceRectangle(): Rectangle(), Formatting() {} ;

  niceRectangle(double A, double B, int W, string C):
                Rectangle(A, B), Formatting(W, C) {};
    void print(ostream &out )  {
      out <<"(nr) "<< a << " by " << b << ": ar="
          << area() << ", per=" << perimeter()
          << ", linecol=" << lineColor << ", linewidth="
          << lineWidth << endl;
    }

  friend ostream& operator <<(ostream &out,niceRectangle NR) {
        NR.print(out);
    }
};
```

Κλήση των constructors των κλάσεων - γονέων

Από Rectangle

Από Formatting

Πολλαπλή κληρονομικότητα

```
niceRectangle nr;
nr.print(cout);
```
```
(nr) 0 by 0: ar=0, per=0, linecol=transparent, linewidth=0
```

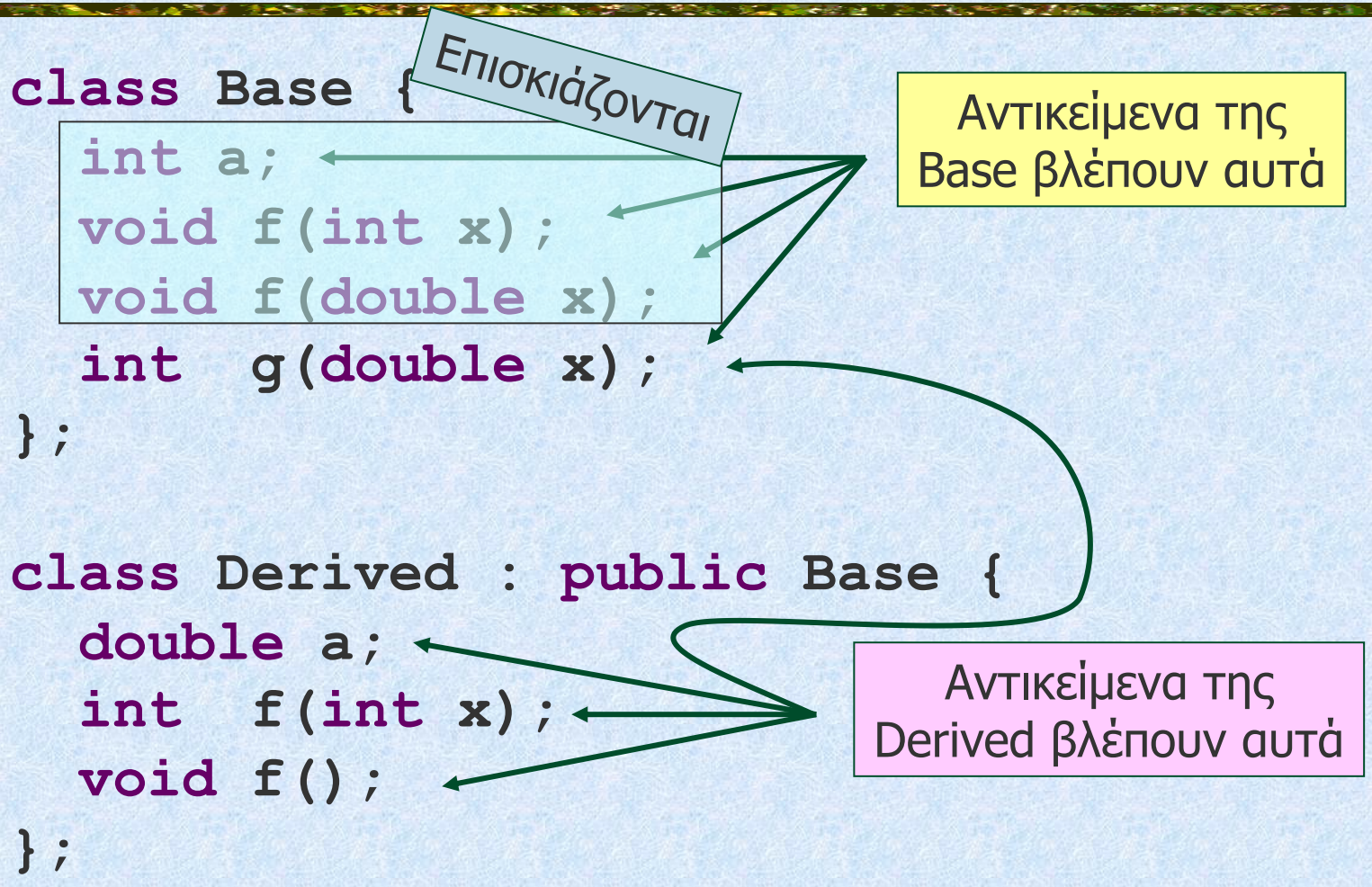
```
niceRectangle nr2(2, 3, 1, "blue");
nr2.print(cout);
```
```
(nr) 2 by 3: ar=6, per=10, linecol=blue, linewidth=1
```

```
cout<<"(nr2) area= "<<nr2.area()<<endl;
```
```
(nr2) area= 6
```
```
nr2.setA(3);
cout<<"(nr2) per="<<nr2.perimeter()<<endl;
```
```
(nr2) per=12
```

```
Rectangle *rr = &nr2;  // ?
rr->print();
```
```
[a=3 b=3 ar=9 per=12]
```

```
Formatting *ff = &nr2; // ?
ff->print();
```
```
[color=blue width=1]
```

Επισκίαση μεθόδων




```
class Base {
  int a;
  void f(int x);
  void f(double x);
  int  g(double x);
};

class Derived : public Base {
  double a;
  int  f(int x);
  void f();
};
```

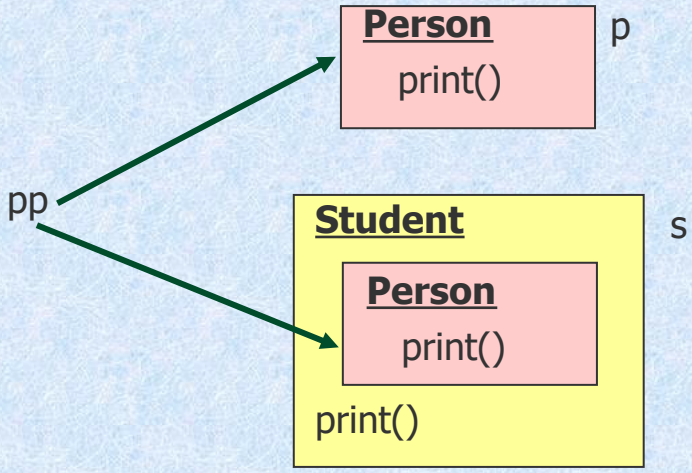
Επισκίαση μεθόδων

```
class Person {
protected:
  string name;
public:
  ...
  void print() const { cout << "a person called "
            << name << " sleeps\n";}
};

class Student : public Person {
private:
  string enrolledIn;
public:
  void print() const { cout << "a student called "
    << name << " reads about " << enrolledIn << "\n"; }
};
```

Δεν μεταβάλλει το αντικείμενο

Επισκίαση μεθόδων

```
Person p, *pp;
Student s;
p.print(); // person John Doe sleeps
s.print(); // student John Doe reads about maths

pp = &p;
pp->print(); // person John Doe sleeps

pp = &s;
pp->print(); // person John Doe sleeps
```

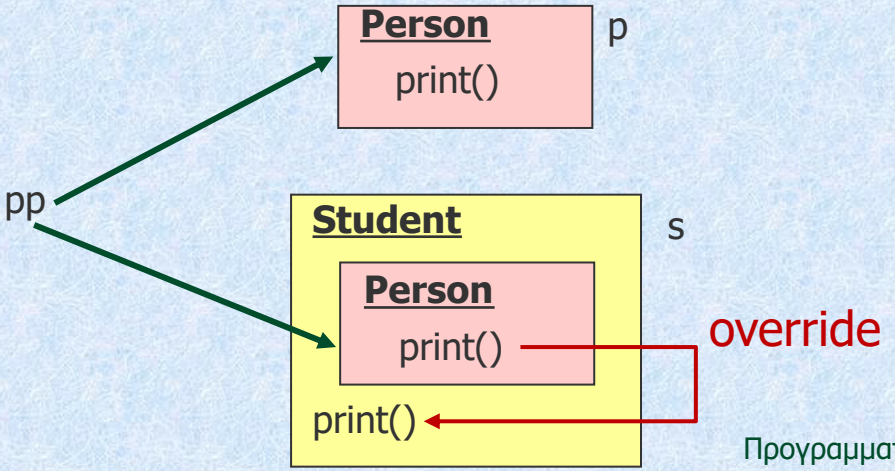


Εικονικές μέθοδοι

```
class Person {
protected:
  string name;
public:
  ...
  virtual void print() const {
     cout << "person " << name << " sleeps\n";}
};

class Student : public Person {
private:
  string enrolledIn;
public:
  void print() const override{ cout << "student "
    << name << " reads about " << enrolledIn << "\n"; }
};
```

Εικονικές μέθοδοι

```
Person p, *pp;
Student s;
p.print(); // person John Doe sleeps
s.print(); // student John Doe reads about maths

pp = &p;
pp->print(); // person John Doe sleeps

pp = &s;
pp->print();// student John Doe reads about maths
```



Πολυμορφισμός

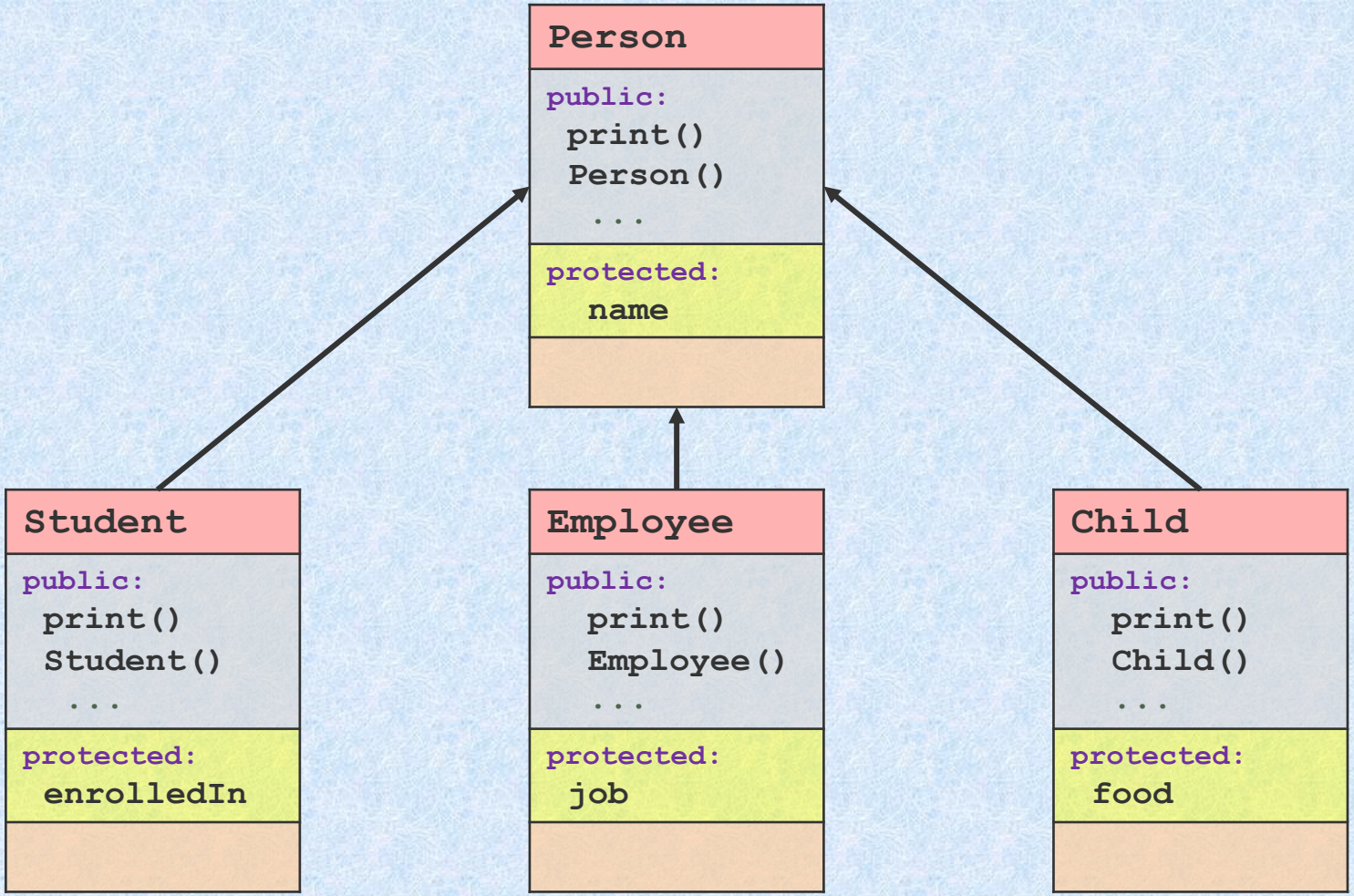
- Η ίδια μέθοδος υλοποιείται διαφορετικά σε διαφορετικές εξειδικεύσεις των αντικειμένων
- Υλοποιείται με χρήση:
  - εικονικών μεθόδων
  - κλήση αυτών μέσω δεικτών ή αναφορών

Αφηρημένες μέθοδοι και κλάσεις

```
class Person {
   ...
   virtual void freeTime() const = 0;
};
```

- Εικονικές (αφηρημένες) μέθοδοι:
  δεν υλοποιούνται στην κλάση Person αλλά οι κλάσεις που την κληρονομούν αναλαμβάνουν την υποχρέωση να τις κάνουν override
- Εικονικές (αφηρημένες) κλάσεις:
  κλάσεις που περιέχουν αφηρημένες μεθόδους

Πολυμορφισμός



Πολυμορφισμός

```
class Person {
  ...
  virtual void print() const { cout <<"person "<<name
                             <<" sleeps\n";} };

class Student : public Person {
  ...
  void print() const override { cout<<"student "<< name
                        <<" studies "<<enrolledIn<<"\n";} };

class Child : public Person {
  ...
  void print() const override { cout <<"child "<<name
                             <<" eats "<<food<<"\n"; } };

class Employee : public Person {
  ...
  void print() const override { cout<< "employee "<<name
                             <<" works as a "<<job<<"\n"; } };
```

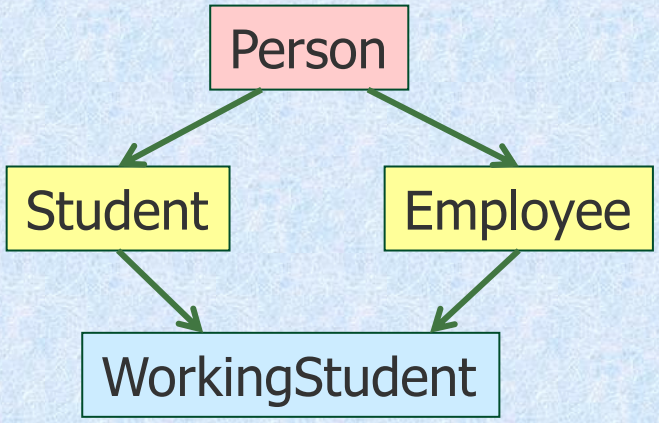
Πολυμορφισμός

```
void allInfo(int n, Person *p[]) {
    for (int i = 0; i < n; i++)
        p[i] -> print();
}

int main() {
  Person *p[] = { new Person, new Student,
                  new Child,  new Employee };
  allInfo(4, p);
}
```

```
person John Doe sleeps
student John Doe studies maths
child John Doe eats Milk
employee John Doe works as a c++ programmer
```

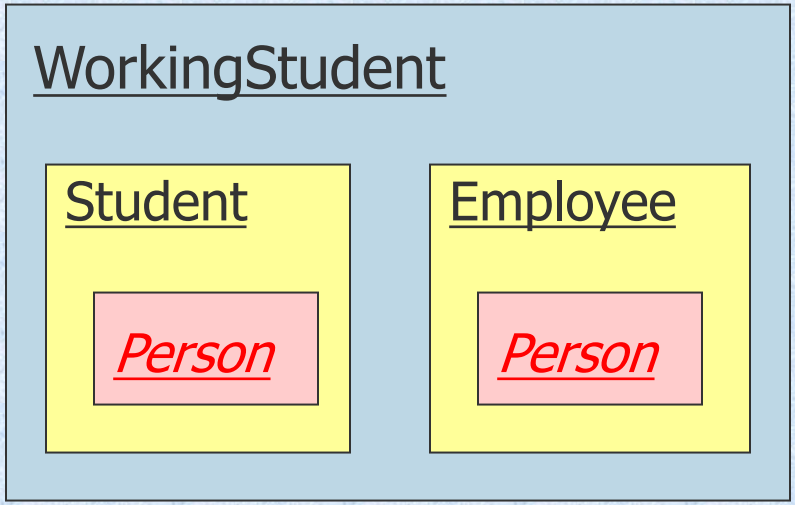
Εικονικές κλάσεις

```
class Person { … }

class Student :
  public Person { … }

class Employee :
  public Person { … }

class WorkingStudent :
  public Student,
  public Employee { … }
```



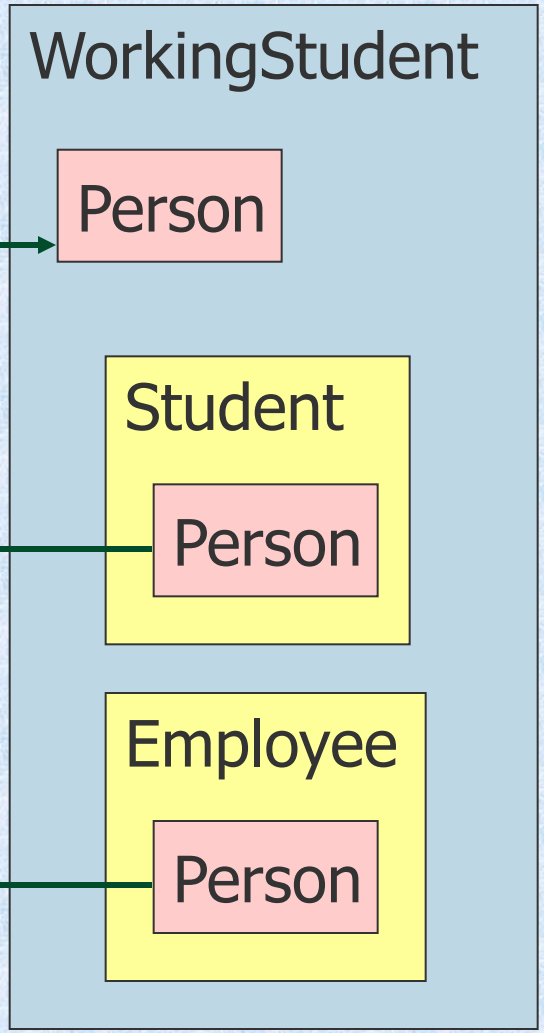
Non-static member 'name' found in multiple base-class subobjects of type 'Person':
class WorkingStudent -> class Student -> class Person
class WorkingStudent -> class Employee -> class Person
member found by ambiguous name lookup

```
Declared in: main.cpp

protected:
basic_string<char, char_traits<char>, allocator<char>> Person::name
```



Εικονικές κλάσεις

```
class Person { … }

class Student :
  virtual public Person { … }

class Employee :
  virtual public Person { … }

class WorkingStudent :
  public Student,
  public Employee { … }
```



virtual: δεν θα δημιουργηθεί στιγμιότυπο μελών δηλωμένων ως virtual

Πολυμορφισμός

```
void allInfo(int n, Person *p[]) {
    for (int i = 0; i < n; i++)
        p[i] -> print();
}

int main() {
  Person * p[] = {new Person, new Student, new Child,
                  new Employee, new WorkingStudent };
allInfo(5, p);
}
```

```
person John Doe sleeps
student John Doe studies maths
child John Doe eats Milk
employee John Doe works as a c++ programmer
c++ programmer working student John Doe studies 4 and works
6 hours per day
```

ΕΞΑΙΡΕΣΕΙΣ

Χειρισμός εξαιρέσεων

- Τι είναι εξαιρέσεις (exceptions)
  - Κάθε είδους ανωμαλίες ή σφάλματα που προκύπτουν κατά την εκτέλεση του προγράμματος και πρέπει να ξεπερασθούν με ειδικό τρόπο
- Εξαιρέσεις προκαλούνται
  - Από το σύστημα, π.χ. εξάντληση μνήμης
  - Οπως αποφασίζουμε εμείς, σύμφωνα με τη λογική του προγράμματός μας

Χειρισμός εξαιρέσεων

- Χειρισμός εξαιρέσεων στη C++
  - Τι θέλουμε να γίνει όταν προκύψει μια εξαίρεση:
    - τερματισμός του προγράμματος;
    - μεταβολή που αναιρεί την εξαίρεση;
    - άλλο;
- Μηχανισμός: **throw / try - catch**
  - η εξαίρεση προκαλείται με **throw** είτε αυτόματα, είτε ρητά στο πρόγραμμα
  - ανιχνεύεται μέσα σε block εντολών **try**
  - αντιμετωπίζεται μέσα σε block εντολών **catch**

Χειρισμός εξαιρέσεων: τυπικό παράδειγμα

- Αιτούμαστε μνήμη: αν υπάρχει παίρνουμε ένα δείκτη, διαφορετικά παίρνουμε NULL
- Δυναμική δέσμευση σε C: **`malloc`**

```
int *p;
...
p = (int *) malloc(sizeof(int));
if (p == NULL) {
  printf("Out of memory!\n");
  exit(1);
}
...
```

type casting

Ζητούμενο μέγεθος μνήμης

Τι κάνουμε σε περίπτωση μη διαθέσιμης μνήμης;

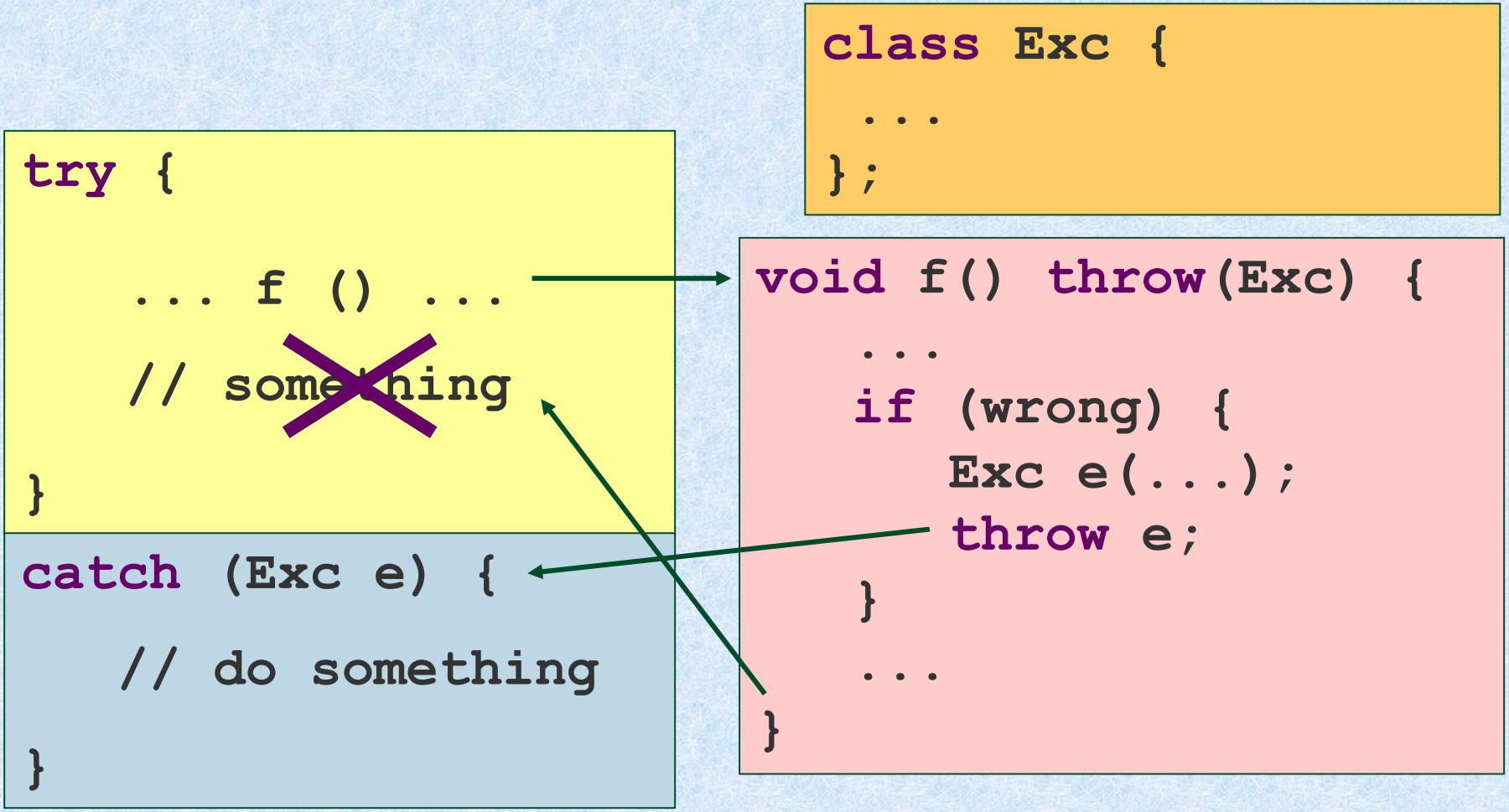
Χειρισμός εξαιρέσεων

- Δυναμική δέσμευση σε C++
  - Έλεγχος με ανίχνευση εξαίρεσης (exception) ή με ανίχνευση null, δηλώνοντας nothrow

```
int *p = new(nothrow) int;
if (!p) {
   cout << "Out of memory!";
   exit(1); }
```

Τι κάνουμε σε περίπτωση μη διαθέσιμης μνήμης;

Χειρισμός εξαιρέσεων



Χειρισμός εξαιρέσεων - παράδειγμα

1. Γνωστή εξαίρεση (ορισμένη στη γλώσσα ή από τον προγραμματιστή)

2. Πρόκληση εξαίρεσης (από τον προγραμματιστή, ενίοτε από τη γλώσσα)

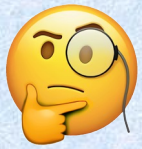
```
class runtime_error: public exception
double division(double num, double den) {
    if (den) return num/den;
    throw runtime_error("Division by zero!\n");
}
int main() {
    double a = 1, b = 0, c;
    try {
        c = division(a, b);
        cout << a << "/" << b << "= " << c << endl; }
    catch(runtime_error &e) {
        cout << "exception: " << e.what();
    }
}
```

3. Χειρισμός εξαίρεσης (από τον προγραμματιστή)

`what()`:
Ορίζεται στην `runtime_error`

Χειρισμός εξαιρέσεων - παράδειγμα

```
class myRectangle {
private:
    double a, b;
public:
    myRectangle();
    myRectangle(double, double);
    double A();
    double B();
    void setA(double);
    void setB(double);
    double perimeter();
    double area();
    void print(ostream&);
};
```



Τι κάνουμε σε περίπτωση αρνητικού μήκους;

Ερώτηση:
Είναι υποχρεωτικό να γίνει με exception η αντιμετώπιση της ανεπιθύμητης αυτής συνθήκης;
Hint: όχι

Χειρισμός εξαιρέσεων - στρατηγική

- ορισμός κλάσεων εξαιρέσεων
- συγγραφή κώδικα **throw ... try ... catch**

ΠΡΟΚΛΗΣΗ

ΧΕΙΡΙΣΜΟΣ

```
class exc_1: public exception {...}
class exc_2: public exception {...}
...
try {
   ...
   if (error condition 1) throw exc_1()
   if (error condition 2) throw exc_2()
   ...
}
catch(exc_1 &e) { ...do_something 1...}
catch(exc_2 &e) { ...do_something 2...}
```

εκτελείται ο constructor της exc_1

πρόσβαση στο αντικείμενο

Χειρισμός εξαιρέσεων - παραδείγματα

```
class unreal_rectangle : public exception {};

class Rectangle {
...
public:
  ...
  Rectangle(double A, double B) {
    try {
      a = A; b = B;
      if ((a<0) || (b<0)) throw unreal_rectangle();
    }
    catch (unreal_rectangle &e) {
      cout << "illegal value of a or b\n";
      a = a < 0 ? 0 : a;
      b = b < 0 ? 0 : b;
    }
```

Πρόκληση εξαίρεσης

Χειρισμός εξαίρεσης

Χειρισμός εξαιρέσεων - παραδείγματα

```
class unreal_rectangle_detailed : public runtime_error {
public:
  unreal_rectangle_detailed():
    runtime_error("exception: invalid rectangle") {};
};
class Rectangle {
...
public:
  ...
  Rectangle(double A, double B) {
    try {
      a = A; b = B;
      if ((a<0) || (b<0)) throw unreal_rectangle_detailed();
    }
    catch (unreal_rectangle_detailed &e) {
      cout << e.what() << "\n";
      a = a < 0 ? 0 : a;
      b = b < 0 ? 0 : b;
    }
}
```

Ορισμός εξαίρεσης από τον προγραμματιστή

Χειρισμός εξαιρέσεων - παραδείγματα

```
class negative_length_a : public exception {};
class negative_length_b : public exception {};

class Rectangle {
...
public:
  Rectangle(double A, double B) {
    try {
      a = A; b = B;
      if (a<0) throw(negative_length_a());
      if (b<0) throw(negative_length_b());
    }
    catch(negative_length_a) { a = -a;}
    catch(negative_length_b) { b = -b;}
  }
```

Ορισμός εξαιρέσεων από τον προγραμματιστή (εναλλακτικό σενάριο)

Χειρισμός εξαιρέσεων

Χειρισμός εξαιρέσεων

- Δυναμική δέσμευση μνήμης σε C++
  - Έλεγχος με ανίχνευση εξαίρεσης (exception)

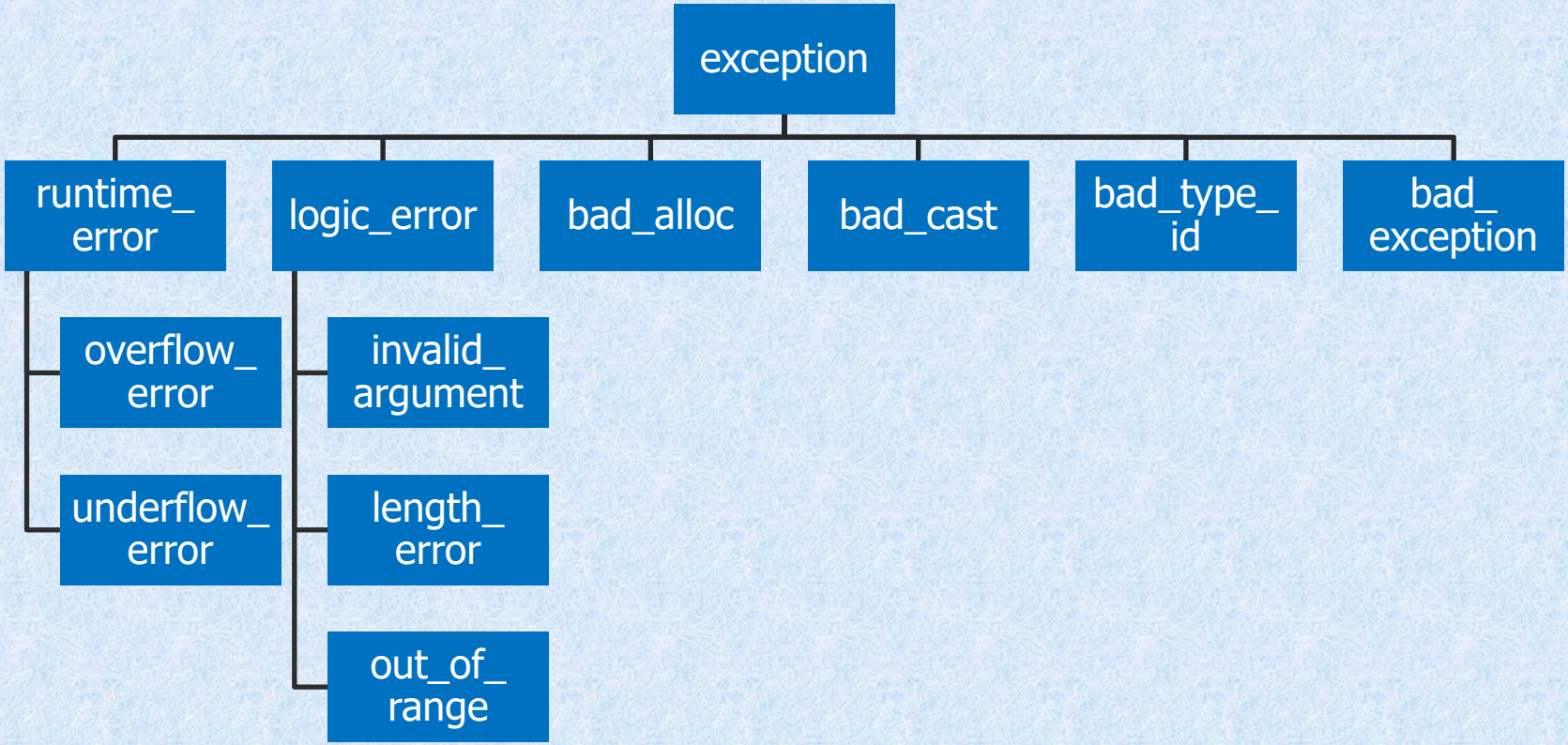
```
try {
   int *p = new int[10000000000000000];
}
catch(bad_alloc) {
   cout << "Are you serious?!";
   exit(1);
}
```

Ορισμός εξαίρεσης (@STL)

Πρόκληση εξαίρεσης από την γλώσσα

Χειρισμός εξαίρεσης

Εξαιρέσεις στη C++



Χειρισμός εξαιρέσεων

```
class stack {
  ...
  void push (int x) {
    a[top++] = x;
  }
  int pop () {
    return a[--top];
  }
  ...
  int mysize;
  int top;
  int *a;
};
```

Αν η στοίβα είναι γεμάτη:
top == mysize

Αν η στοίβα είναι άδεια:
top == 0

Χειρισμός εξαιρέσεων

```
#include <exception>
using namespace std;
class full_stack : public exception {};
class stack {
   ...
stack(int NN) {
   top=0; mysize=NN;
   try {
     data = new int[mysize]; }
   catch(bad_alloc) {
     mysize = 5; data = new int[mysize];
     cerr<<"not enough memory, size set to 5";
   }
};
```

το new κάνει throw

Δεν είναι υποχρεωτικό να διακόπτουμε το πρόγραμμα...

Χειρισμός εξαιρέσεων

```
#include <exception>
using namespace std;
class empty_stack : public exception {};
class stack {
  ...
  int pop() throw(empty_stack) {
    if (top > 0)
      return a[--top];
    else
      throw empty_stack();
  }
};
```

Χειρισμός εξαιρέσεων

```
#include <exception>
using namespace std;
class full_stack : public exception {};
class stack {
  ...
  void push(int x) throw(full_stack) {
    if (top < mysize)
      a[top++] = x;
    else
      throw full_stack();
  }
};
```

Πιθανή αντιμετώπιση:

```
catch (full_stack) {
    myStack.pop();
    myStack.push(data);
}
```

Χειρισμός εξαιρέσεων

```
int main() {
  try {
    ...
    if (...) throw 42;
    if (...) throw "hello";
    if (...) throw out_of_memory();
    ...
  }
  catch (int n)                { ... }
  catch (const char *s)        { ... }
  catch (out_of_memory &e)     { ... }
  catch (exception &e)         { ... }
};
```

Πολλαπλοί handlers, δοκιμάζονται με τη σειρά

Χειρισμός εξαιρέσεων

```
int main() {
  int choice;
  do {
    cin >> choice;
    try {
      if (choice == 1) throw 1.0;
      if (choice == 2) throw 2;
      if (choice == 3) throw bad_alloc();
      if (choice == 0) throw 'e';
    }
    catch (const double s) { cout<< "your choice is "<<s; }
    catch (const int s) { cout<<"hello, your choice is "<<s; }
    catch (const char s) { cout <<s<< " means bye!"; break;}
    catch (bad_alloc) { cout<<"not really bad_alloc..."; }
  } while (true);
}
```

Κακή πρακτική!

Χειρισμός εξαιρέσεων - αρχεία

```
ifstream inFile;
char c;
try {
  inFile.open("mydata.txt");
  if (inFile.fail()) throw "cannot open 'mydata.txt'";
  inFile >> c;
  while (inFile.good()) {
    cout << c;
    inFile >> c;
    if (inFile.fail()) throw "file error";
  }
}
catch(const char *e) {
  cout << "file error: " << e << endl;
  // ...
}
```

Χειρισμός εξαιρέσεων

```
class red_ball: public exception {};
class green_ball: public exception {};

void children_playing()
  throw(red_ball, green_ball) { ... }

void little_field() throw(green_ball) {
  try {
    children_playing();
  }
  catch (red_ball &e) {
    cout << "Caught the red one!" << endl;
  }
  cout << "Little finished" << endl;
}
```

Χειρισμός εξαιρέσεων

```
void bigger_field() {
  try {
    little_field();
  }
  catch (green_ball &e) {
    cout << "Caught the green one!" << endl;
  }
  cout << "Bigger finished" << endl;
}
```