Теми:

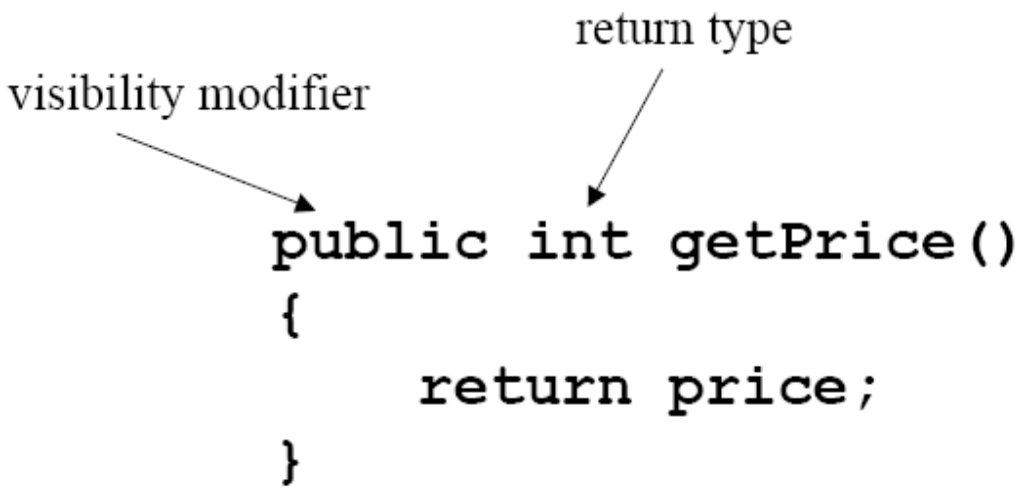
- **Предаване на данни като параметри. Accessor методи. Mutator метод.**
- **Общи понятия:функции; декларация на прототип; дефиниране на функция; повикване на функция; работа с локални и глобални типове; дефиниране и използване на overloaded функции.**
- **Функции –методи. Глобални функции**
- **Припокриване на функции**
- **Локални променливи. Локални и/или глобални елементи на клас.**

Accessor methods

- Methods implement the behavior of objects.
- Accessors provide information about an object.
- Methods have a structure consisting of a header and a body.
- The header defines the method's *signature*.

- The body encloses the method's statements.

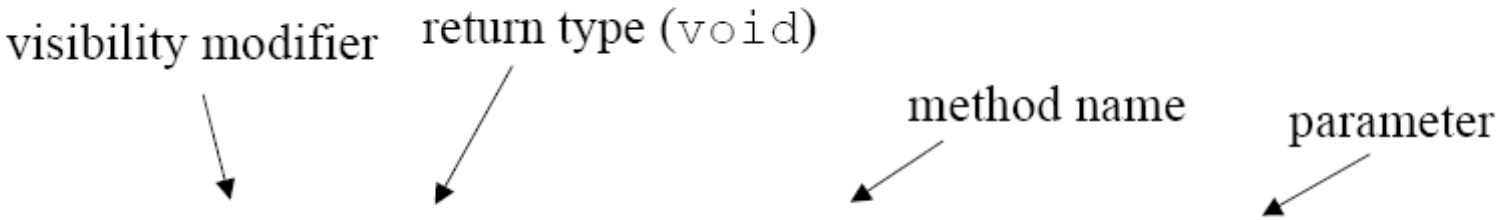
Accessor methods



Mutator methods

- Have a similar method structure: header and body.
- Used to *mutate* (i.e., change) an object's state.
- Achieved through changing the value of one or more fields.
  - Typically contain assignment statements.
  - Typically receive parameters.

Mutator methods




```
public void insertMoney(int amount)
{
    balance = balance + amount;
}
```

Method calls (2)

Syntax :

object . methodName (parameter-list)

Functions

- declare function prototypes
- define function bodies
- call functions
- deal with local and global variable scope
- define and use overloaded functions

Member functions

Constructor and destructor are member functions that are always present.
You can add your special-purpose class member functions – declaring them in class declaration
And defining them in class definition:

```
class XY {
public :
        double x,y;
        XY();
        XY(double xarg, double yarg);
        double Getx() const {return x;}
        double GetY() const { return y;}
};
```

Function overloading

- more than 1 function with same name and different parameter list

- example:

```
double average(double nmber1, double number2);
double average( int array[], int arraysize);
```

- if difference is return type only – you receive a compiler error

Example with classes

```
// Function prototypes

void DisplayWelcome();

void DisplayProjectedValue(double amount, int years, double
rate);

void DisplayProjectedValue(double amount, int years);

double GetInvestmentAmount();

int GetInvestmentPeriod(int min=10, int max=25);


// Define and initialize a global integer variable

int numberOfYourFunctionsCalled = 0;
```

Programming tips

```
// This is the entry point for this application
int main(void)
{       DisplayWelcome();
        DisplayProjectedValue(10000, 25, 6.0);

        Console::WriteLine(S"\nEnter details for your investment:");
        double sum = GetInvestmentAmount();
        int period = GetInvestmentPeriod(5, 25);

        Console::WriteLine(S"\nYour plan...");
        DisplayProjectedValue(sum, period, 6.0);

        return 0;}
// Display a welcome message to the user
void DisplayWelcome()
{       numberOfYourFunctionsCalled++;
        Console::WriteLine(S"Welcome to your friendly Investment Planner");
        return;
}
```

Programming tips

```
// Calculate and display the projected value of the investment
void DisplayProjectedValue(double amount, int years, double rate)
{           numberOfYourFunctionsCalled++;

            double rateFraction = 1 + (rate/100);
            double finalAmount = amount * Math::Pow(rateFraction, years);
            finalAmount = Math::Round(finalAmount, 2);

            Console::Write(S"Investment amount: ");
            Console::WriteLine(amount);

            Console::Write(S"Growth rate [%]: ");
            Console::WriteLine(rate);

            Console::Write(S"Period [years]: ");
            Console::WriteLine(years);
            return;
}
```

Programming tips

```
// Ask the user how much money they want to invest
double GetInvestmentAmount()
{   numberOfYourFunctionsCalled++;

    Console::Write(S"How much money do you want to invest? ");
   String __gc * input = Console::ReadLine();
   double amount = input->ToDouble(0);

   return amount;
}
```

Programming tips

```
// Ask the user how long they want to invest their money (between min and max)
int GetInvestmentPeriod(int min, int max)
{
        numberOfYourFunctionsCalled++;

        Console::Write(S"Over how many years [");
        Console::Write(S"min=");
        Console::Write(min);
        Console::Write(S", max=");
        Console::Write(max);
        Console::Write(S"] ? ");

        String __gc * input = Console::ReadLine();
        int years = input->ToInt32(0);

        return years;
}
```

Local variables

- Fields are one sort of variable.
  - They store values through the life of an object.
  - They are accessible throughout the class.
- Methods can include shorter-lived variables.
  - They exist only as long as the method is being executed.
  - They are only accessible from within the method.

Local variables

```
public int refundBalance()
{
    int amountToRefund;
    amountToRefund = balance;
    balance = 0;
    return amountToRefund;
}
```

Public vs private

- Public attributes (fields, constructors, methods) are accessible to other classes.
- Fields should not be public.
- Private attributes are accessible only within the same class.
- Only methods that are intended for other classes should be public.

Information hiding

- Data belonging to one object is hidden from other objects.
- Know what an object can do, not how it does it.
- Information hiding increases the level of *independence*.

public and private elements

```
class XY {
        private:
        double x,y;
        public:
                XY();
                XY(double a, double b);
                double GetX() const;
                double GetY() const;
};
```

- class members are private by default
- object declaration:

  XY bottom(4.0, 10.0);

- we can access functions from outside

Global functions

- A function declared outside any class declaration
- Example:

```
void Show(XY xy) // global function,working with
                 // class methods
{
  printf("x=%f, y =%f\n", xy.GetX(), xy.GetY());
}
```

- The global function can have name, matching the name of a class member function if differs in parameter list;

Свойства на класа в C++/CLI

Основни характеристики

- Името на свойството извиква функция;
- Свойството има get() и set() функция;
  - **read-only property** – дефиниция само на get() функция;
  - **write-only property** - дефиниция само на set() функция.

Видове свойства

- **Скаларни свойства**

  Клас String - свойството Length е скаларно и е read-only защото е дефинирана само функцията get():

  ```
  str->Length
  ```

- **Индексни свойства**

  Класът String ви дава достъп до отделен символ от низа, което е индексно свойство:

  ```
  str[2]
  ```

Дефиниране на скаларни свойства

```
ref class Weight
{
      private:
            int lbs;
      public:
            property int pounds
            {
              int get() { return lbs; }
              void set(int value) {lbs = value;}
            }

};
```

Примери

- **read-only property**

```
property double meters
{
        double get();
}
```

- **write-only property**

```
property double meters
{
        void set(int x);
}
```

Тривиални скаларни свойства

```
value class Point
{
  public:
        property int x;
        propecrty int y;
  };
```

Използване на свойства

```
Weight^ wt = gcnew Weight;
wt->pounds = 162;

Console::WriteLine(L"Weight is {0} lbs.",
                   wt->pounds);
```

Когато е **ref class** , винаги се достъпва свойството с оператора ->

Дефиниране на индексни свойства

```
ref class Name
{
    private:
        array<String^>^ Names;
    public:
      property String^ default[int]

      {
          String^ get(int index)
          {
            if(index < Names->Length)
                return Names[index];
          }
          void set(int index, String^ name)
          {
                if(index < Names->Length)
                      Names[index] = name;
          }
    }
};
```

Работа с индексни свойства

```
Name^ myName = gcnew Name(L”Ebenezer”, L”Isaiah”);

// List the names
for(int i = 0 ; i < myName->NameCount ; i++)
      Console::WriteLine(L”Name {0} is {1}”, i+1,
                              myName[i]);
```

Статични свойства

```
value class Length
{
        // Code as before...
        public:
        static property String^ Units
        {
                String^ get()
                {
                        return L"feet and inches";
                }
        }
};
```

- Console::WriteLine(L"Class units are {0}.", Length::Units);

Review

- Class bodies contain fields, constructors and methods.
- Fields store values that determine an object's state.
- Constructors initialize objects.
- Methods implement the behavior of objects.

Review (variables)

- Fields, parameters and local variables are all variables.
- Fields persist for the lifetime of an object.
- Parameters are used to receive values into a constructor or method.
- Local variables are used for short-lived temporary storage.

Object interrelationships
1. Friend classes and friend functions

- for closely related classes
- declaring someone is a friend give him some rights (including on private members)
- if I want to use some elements of Planet in Moon objects, I have to declare like this:

```
class Planet : public Orbiter
{
friend class Moon;    // no prior declaration required. Moon can now use Planet
      elements

public:
// constructors and other member functions
};
…
Now is possible in all Moon class member functions:
int Moon::NewMass()
{
      return GetPlanet()->m_mass * m_mass;          // planet's mass * moon's
      mass
}
```

Пример

```
class Storage
{
private:
    int m_nValue;
    double m_dValue;
public:
    Storage(int nValue, double dValue)
    {
        m_nValue = nValue;
        m_dValue = dValue;
    }

    // Make the Display class a friend of Storage
    friend class Display;
};
```

```
class Display
{
private:
    bool m_bDisplayIntFirst;

public:
    Display(bool bDisplayIntFirst) { m_bDisplayIntFirst = bDisplayIntFirst; }

    void DisplayItem(Storage &cStorage)
    {
        if (m_bDisplayIntFirst)
            std::cout << cStorage.m_nValue << " " << cStorage.m_dValue << std::endl;
        else // display double first
            std::cout << cStorage.m_dValue << " " << cStorage.m_nValue << std::endl;
    }
};
```

Friend functions - example

```
class Accumulator
{
private:
    int m_nValue;
public:
    Accumulator() { m_nValue = 0; }
    void Add(int nValue) { m_nValue += nValue; }

    // Make the Reset() function a friend of this class
    friend void Reset(Accumulator &cAccumulator);
};

// Reset() is now a friend of the Accumulator class
void Reset(Accumulator &cAccumulator)
{
//And can access the private data of Accumulator objects
    cAccumulator.m_nValue = 0;
}
```