Nunca existiu uma grande inteligência sem uma veia de loucura.

Aristóteles

1 Classes

- Uma classe é um modelo de um objeto.
- Quando definimos uma classe estamos definindo somente o comportamento de um objeto e não um objeto.
- A definição da classe é a descrição dos atributos e métodos de um objeto.
- A classe passa a ser tratada como um tipo para a Linguagem C++.
- A definição da classe não reserva nem cria nenhum espaço em memória.

A classe é um modelo de um objeto da realidade. A classe irá possuir os atributos e os métodos que o objeto da realidade possui. Quando definimos uma classe, estamos definindo um modelo e não há alocação de nenhum espaço de memória. A partir da definição de uma classe teremos então um tipo a mais na linguagem.

Um objeto é uma variável de determinada classe. A variável aloca um espaço de memória e possui os atributos e métodos definidos pela classe. Uma variável de uma determinada classe também é chamada de instância de uma classe. Podemos ter várias variáveis definidas de uma única classe, sendo que cada variável é uma instância da classe e tem o seu próprio conjunto de atributos. Um objeto é considerado pela linguagem como se fosse uma variável comum definida no programa.

Como exemplo, considere o tipo *int* que é predefinido na linguagem. Podemos declarar quantas variáveis do tipo *int* forem necessárias ao programa. Da mesma forma, podemos declarar quantas variáveis quisermos de uma classe já definida.

Por exemplo, a descrição de uma espaçonave é uma classe, que contém atributos (nível de energia ou quantidade de munição restantes) e métodos (disparar tiros ou levantar o escudo, por exemplo). A figura 1 exemplifica os atributos e métodos de uma espaçonave.

Figura 1: Exemplo de objeto.

2 Definição de Classe

- A definição possui três partes distintas: **private**, **protected** e **public**
- Em cada parte podemos definir dois componentes: Atributos e métodos

Sintaxe:

```
class nome_classe {
  [private:]   /* Atributos e métodos */
  [protected:] /* atributos e métodos */
  [public:]    /* atributos e métodos */
};
```

A definição de uma classe é muito parecida com a definição de uma estrutura em C++. Coloca-se a palavra reservada class seguida do nome da classe. Após coloca-se a definição dos atributos e métodos da classe entre chaves ({ }). Obrigatoriamente ao final da definição da classe deve-se colocar o ponto e virgula (;).

Dentro das chaves pode-se ter três sessões definidas, cada uma delas apresentando as suas características. A definição das sessões pode seguir qualquer ordem, podendo inclusive ter mais de uma sessão do mesmo tipo dentro da definição da classe. Cada sessão deve tem um cabeçalho indicando o seu tipo seguido de dois pontos (:).

Uma classe pode não ter as três seções podendo ser omitida aquela que não é necessária.

Programa 1: Definição de classe.

```
/* programa_001.cpp */
#include "biblaureano.h"

class Espaconave{
   protected : // Definição de membros protegidos
   private:    // Definição de membros privados
   public :    // Definição de membros publicos
}; //não esquecer do ; ao final

int main(){
  cout << "Game Over!!!" << endl;
  return 0;
}
```

3 Definição de Objetos

- Para se definir um objeto devemos criar uma variável do tipo de uma classe previamente definida.
- Esta variável será uma instância da classe.

Sintaxe:

```
  class nome_classe {
     private : ... // membros privados
     public: ....  // membros públicos
  };
  nome_classe      objeto_1;
  nome_classe      objeto_2;
```

Uma vez definida uma classe podemos criar variáveis, ou melhor dizendo, objetos desta classe.

A definição de objetos é bastante parecida com a definição de uma variável comum na linguagem C++. Basta-se colocar o nome da classe como sendo o tipo da variável que estaremos definindo um objeto.

Podemos definir mais de uma variável para uma determinada classe, criando diversas instâncias da classe. Cada variável possuirá o seu próprio conjunto de atributos e somente compartilharão o código dos métodos entre si.

Programa 2: Definição de objeto.

```
/* programa_002.cpp */
#include "biblaureano.h"

//aqui é a definição do modelo
class Espaconave{
   protected : // Definição de membros protegidos
   private:    // Definição de membros privados
```

```
8     public :     // Definição de membros publicos
9  }; //não esquecer do ; ao final
10
11 int main(){
12    Espaconave TieFighter; //aqui criamos 2 objetos
13    Espaconave XWing;      //a partir do modelo
14    cout << "Game Over!!!" << endl;
15    return 0;
16 }
```

4 Membros públicos

- Todos os atributos e métodos definidos na seção **public** podem ser acessados e usados por qualquer programa que declare a classe.
- Para se acessar um atributo de um objeto, colocamos o nome do objeto seguido do nome do atributo separados por ponto.
- Apesar de possível, não convém colocar-se atributos na seção **public** pois isto fere o conceito de encapsulamento.

A sessão pública, definida pela palavra **public**, indica que tanto os atributos definidos nela como os métodos, podem ser chamados e usados por qualquer programa que defina esta classe.

O acesso aos métodos pode ser feito em qualquer função do programa, sem restrição. Para se acessar um atributo diretamente, coloca-se o nome do objeto (variável) seguida do nome do atributo, separado por um ponto. A sintaxe é parecida com a sintaxe de acesso à campos de uma estrutura.

Os atributos definidos na sessão pública podem ser alterados indistintamente também por qualquer função em qualquer parte do programa, desde que esta função tenha acesso a instância da classe (variável). Esta característica vai contra a filosofia de encapsulamento da Orientação a Objeto que diz que todos os atributos devem ser acessados por métodos da própria classe e não devem ser alterados por nenhum módulo fora da classe.

Com o encapsulamento garante-se sempre a integridade de valores dos atributos, pois os métodos de alteração, criados na própria classe, não devem colocar nenhum valor inválido.

Programa 3: Acesso público dentro da classe.

```
1  /* programa_003.cpp */
2  #include "biblaureano.h"
3
4  //aqui é a definição do modelo
5  class Espaconave{
6     protected : // Definição de membros protegidos
7     private:    // Definição de membros privados
8     public :    // Definição de membros publicos
9        int energia;
```

```
10      int municao;
11  }; //não esquecer do ; ao final
12
13  int main(){
14    Espaconave TieFighter; //aqui criamos 2 objetos
15    Espaconave XWing;      //a partir do modelo
16
17    //acessando atributos públicos dos objetos
18    XWing.energia = 100;
19    XWing.municao = 900;
20
21    TieFighter.energia = 200;
22    TieFighter.municao = 1200;
23
24    cout << "TieFighter tem " << TieFighter.energia
25         << " de energia e " << TieFighter.municao
26         << " tiros." << endl;
27
28    cout << "XWing tem " << XWing.energia
29         << " de energia e " << XWing.municao
30         << " tiros." << endl;
31
32    cout << "Game Over!!!" << endl;
33    return 0;
34  }
```

5 Definição de métodos

- Juntamente com os atributos podemos definir funções dentro da classe.
- Estas funções são chamadas de funções-membro ou de métodos.
- Os métodos possuem acesso a todos os atributos da classe e podem chamar qualquer outro método da classe.
- Funções definidas dentro da definição do corpo da classe serão consideradas como uma função **inline**.

Além de atributos, uma classe possui funções que implementam um processo qualquer. O nome técnico destas funções é função-membro ou simplesmente método.

A sintaxe de definição de um método é idêntica a definição de uma função em C++, aceitando parâmetros de entrada e saída, podendo ter um valor de retorno e implementando o conceito de polimorfismo.

A definição de um método pode ser feito dentro da definição da classe, ou seja, entre chaves ({ }) ou fora da definição da classe. Os métodos definidos dentro da classe serão considerados automaticamente como funções inline para a Linguagem C++.

Um método pode acessar qualquer atributo definido da classe, indiferente do local da definição do atributo, seja

público, privado ou protegido. Um método também pode acionar qualquer outro método definido na classe, indiferente do tipo do mesmo.

6 Chamadas de Métodos

Para se ativar um método de um objeto coloca-se o nome da instância (variável) seguida de um ponto e do nome do método com os parâmetros necessários.

Sintaxe:

```
    nome_classe   objeto;
    int           valor;

    // É executado o método le_atributos do objeto
    valor = objeto.le_atributos();
```

Para se chamar um método procede-se de maneira análoga ao acesso a atributos. Deve-se colocar o nome do objeto, seguido do nome do métodos com todos os seus atributos, separados por um ponto.

Caso o método retorne um valor deve-se atribuir todo o conjunto a uma variável ou parâmetro conforme o padrão de uma função.

Um método de uma classe é nada mais que uma função para a linguagem C++, portanto deve-se pensar que onde a linguagem aceita uma função, pode-se colocar um método de uma classe desde que sempre se identifique a sobre qual instância o método se aplica.

Programa 4: Chamada de um método da classe.

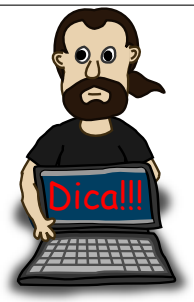
```
/* programa_004.cpp */
#include "biblaureano.h"

//aqui é a definição do modelo
class Espaconave{
   protected : // Definição de membros protegidos
   private:    // Definição de membros privados
   public :    // Definição de membros publicos
      int energia;
      int municao;

   void imprimeDados(){ //definido um método da classe
      cout << "Esta nave tem " << energia
           << " de energia e " << municao
           << " tiros." << endl;
      return;
   }
}; //não esquecer do ; ao final

int main(){
  Espaconave TieFighter; //aqui criamos 2 objetos
  Espaconave XWing;      //a partir do modelo

  //acessando atributos públicos da classe
  XWing.energia = 100;
  XWing.municao = 900;

  TieFighter.energia = 200;
  TieFighter.municao = 1200;

  cout << "TieFighter:";
  TieFighter.imprimeDados(); //chama o método

  cout << "XWing:";
  XWing.imprimeDados(); //chama o método
```

```
36
37    cout << "Game Over!!!" << endl;
38    return 0;
39  }
```

7 Membros Privados

- Os atributos definidos na seção **private** só podem ser acessados pelos métodos da própria classe.
- Nenhuma outra função fora da classe pode alterar os atributos.
- Um método definido na seção **private** só pode ser chamado por métodos da própria classe.
- Esta característica básica permite o conceito de encapsulamento.

Para se implementar o conceito de encapsulamento em toda a sua extensão, deve-se definir todos os atributos de uma classe dentro da sessão **private** da classe. Fazendo-se isto, a linguagem só permite acesso aos atributos através de métodos da própria classe. Um método definido dentro da sessão privada, também só poderá ser chamado por outro método da mesma classe.



Os membros definidos após ***private:*** são chamadas de parte privada da classe e podem ser acessados pela classe inteira, mas não fora dela. Normalmente, na parte privada da classe são colocados os membros que conterão dados. Logo, esses dados estarão *escondidos*. O conceito de *esconder* dados significa que estarão confinados e não poderão ser alterados, por descuido, por funções que não pertençam a própria classe.

Programa 5: Uso de atributos privados.

```cpp
/* programa_005.cpp */
#include "biblaureano.h"

//aqui é a definição do modelo
class Espaconave{
   private:    // Definição de membros privados
      int energia;
      int municao;

   public :    // Definição de membros publicos

   void imprimeDados(){ //definido um método da classe
      cout << "Esta nave tem " << energia
           << " de energia e " << municao
           << " tiros." << endl;
      return;
   }

   void setEnergia( int pEnergia ){
      energia = pEnergia;
      return;
   }

   void setMunicao( int pMunicao ){
      municao = pMunicao;
      return;
   }
}; //não esquecer do ; ao final

int main(){
  Espaconave TieFighter; //aqui criamos 2 objetos
  Espaconave XWing;      //a partir do modelo

  //utilizando os métodos públicos para acessar
  //atributos privados da classe
  TieFighter.setEnergia(900);
  TieFighter.setMunicao(1500);

  XWing.setEnergia(1200);
  XWing.setMunicao(2000);

  cout << "TieFigher:";
  TieFighter.imprimeDados(); //chama o método

  cout << "XWing:";
  XWing.imprimeDados(); //chama o método

  cout << "Game Over!!!" << endl;
  return 0;
}
```

8 Método Construtor

- Pode-se criar um método com o mesmo nome da classe.
- Este método é chamado de método construtor.
- O método construtor será chamado automaticamente para cada definição de uma instância da classe.
- Basicamente no construtor coloca-se a inicialização dos atributos da classe.

As vezes é necessário que se faça uma inicialização dos atributos de um objeto para que o mesmo não contenha valores indevidos. Uma solução seria a criação de um método específico para isto, que faria a inicialização dos atributos de maneira correta. Esta abordagem obrigaria o programador a sempre chamar este método após a definição de uma instância do mesmo. Caso o método não seja chamado a instância poderá conter valores indefinidos.

Na Linguagem C++ podemos definir um método com o mesmo nome da classe. Este método passa a ser considerado o construtor da classe, ou seja, ele será automaticamente chamado quando ocorrer a definição de uma instância do objeto.

Se o método possuir parâmetros, será obrigatório na definição do objeto colocar-se os valores destes parâmetros já na definição. A definição um método construtor possibilita a inicialização automática.

Um *construtor* é uma função-membro que tem o mesmo nome da classe e é executada automaticamente toda vez que um objeto é criado.

Programa 6: Definição do método construtor.

```
1  /* programa_006.cpp */
2  #include "biblaureano.h"
3
4  //aqui é a definição do modelo
5  class Espaconave{
6     private:     // Definição de membros privados
7        int energia;
8        int municao;
9        string nome;
10
11    public :     // Definição de membros publicos
12
13    Espaconave( string pNome, int pEnergia, int pMunicao){
14       cout << "Executando método construtor" << endl;
15       nome = pNome;
```

```
16        energia = pEnergia;
17        municao = pMunicao;
18     }
19
20     void imprimeDados(){ //definido um método da classe
21        cout << "Esta nave se chama " << nome
22             << " e tem " << energia
23             << " de energia e " << municao
24             << " tiros." << endl;
25        return;
26     }
27
28     void setEnergia( int pEnergia ){
29        energia = pEnergia;
30        return;
31     }
32
33     void setMunicao( int pMunicao ){
34        municao = pMunicao;
35        return;
36     }
37 }; //não esquecer do ; ao final
38
39 int main(){
40   Espaconave TieFighter("TieFighter", 90,100);
41   Espaconave XWing("XWing", 500,200);
42
43   TieFighter.imprimeDados();
44   XWing.imprimeDados();
45
46   //ajustando parâmetros
47   TieFighter.setEnergia(900);
48   TieFighter.setMunicao(1500);
49
50   XWing.setEnergia(1200);
51   XWing.setMunicao(2000);
52
53   TieFighter.imprimeDados(); //chama o método
54   XWing.imprimeDados(); //chama o método
55
56   cout << "Game Over!!!" << endl;
57   return 0;
58 }
```

Observe que um construtor não deve retornar nenhum valor. Como o método é chamado diretamente pelo sistema, não é possível recuperar um valor de retorno. Da mesma forma, não faria nenhum sentido um construtor retornar um valor. Caso seja definido um tipo de retorno para o método construtor, o compilador apresentará um erro.

9 Polimorfismo do Construtor

- O método construtor é uma função que é chamada automaticamente.
- Por ser uma função, pode-se definir mais de uma função com o mesmo nome, usando o polimorfismo.
- Conforme o objeto é definido será escolhido o construtor desejado, baseado nos parâmetros usados.

O construtor nada mais é que um método, somente tendo a característica de ser chamado automaticamente na definição do objeto.

A linguagem C++ permite que se tenha mais de uma função com o mesmo nome. A chamada da função correta dependerá dos atributos colocados na definição do objeto. Pode-se ter tantos construtores quantos necessários. A linguagem C++ decide, através do polimorfismo, qual função será chamada.

Como o construtor é um método e a linguagem aceita que se tenha mais de uma função com o mesmo nome, pode-se ter mais de um construtor em uma determinada classe. Baseado em como a definição da instância seja feita será escolhido o construtor adequado, conforme os parâmetros.

A única restrição referente aos métodos construtores é que eles não devem retornar nenhum valor, pois sendo a chamada automática, fica impossível de se processar este valor retornado.

Programa 7: Polimorfismo do método construtor.

```
1  /* programa_007.cpp */
2  #include "biblaureano.h"
3
4  //aqui é a definição do modelo
5  class Espaconave{
6     private:    // Definição de membros privados
7        int energia;
8        int municao;
9        string nome;
10
11    public :    // Definição de membros publicos
12
13    Espaconave( string pNome, int pEnergia, int pMunicao){ //construtor
14       cout << "Executando método construtor com 3 parâmetros" << endl;
15       nome = pNome;
16       energia = pEnergia;
17       municao = pMunicao;
```

```
18      return;
19    }
20
21    Espaconave(string pNome){  //construtor
22       cout << "Executando método construtor com 1 parâmetro" << endl;
23       nome = pNome;
24       municao = 0;
25       energia = 0;
26       return;
27    }
28
29    Espaconave(int pMunicao, int pEnergia ){  //construtor
30       cout << "Executando método construtor com 2 parâmetros" << endl;
31       nome = "unknown";
32       municao = pMunicao;
33       energia = pEnergia;
34       return;
35    }
36
37    Espaconave(){
38       cout << "Executando método construtor sem parâmetros" << endl;
39       nome = "unknown";
40       municao = 0;
41       energia = 0;
42       return;
43    }
44
45    void imprimeDados(){ //definido um método da classe
46       cout << "Esta nave se chama " << nome
47            << " e tem " << energia
48            << " de energia e " << municao
49            << " tiros." << endl;
50       return;
51    }
52
53    void setEnergia( int pEnergia ){
54       energia = pEnergia;
55       return;
56    }
57
58    void setMunicao( int pMunicao ){
59       municao = pMunicao;
60       return;
61    }
62 }; //não esquecer do ; ao final
63
64 int main(){
65   Espaconave TieFighter("TieFigher", 90,100);
66   Espaconave Unknown(100,200);
67   Espaconave YWing("YWing");
68   Espaconave Unknown2;
69
70   TieFighter.imprimeDados();
71   Unknown.imprimeDados();
72   YWing.imprimeDados();
73   Unknown2.imprimeDados();
74
75   cout << "Game Over!!!" << endl;
76   return 0;
77 }
```

10 Construtores de Cópia

Quando é criado uma variável na linguagem C++, pode-se inicializá-la através da colocação da atribuição junto com o valor na própria linha de definição da variável.

Como para se definir uma instância de um objeto nada mais é que a definição de uma variável na linguagem,

pode-se portanto, ao se criar uma instância atribuir um valor para ela através da colocação do sinal de = seguido de um valor.

A definição de uma instância irá sempre executar um construtor, portanto é necessário que se tenha um construtor para este tipo de atribuição na definição. Este construtor é chamado de construtor de cópia, mas na realidade, é um construtor como qualquer outro.

A exigência sintática do mesmo é que ele deve ter um único parâmetro e este parâmetro deve ser do mesmo tipo do valor constante ou variável colocada na definição após =.

Programa 8: Método construtor de cópia.

```
1  /* programa_008.cpp */
2  #include "biblaureano.h"
3
4  //aqui é a definição do modelo
5  class Espaconave{
6     private:    // Definição de membros privados
7        int energia;
8        int municao;
9        string nome;
10
11    public :    // Definição de membros publicos
12
13    Espaconave( string pNome, int pEnergia, int pMunicao){ //construtor
14       cout << "Executando método construtor com 3 parâmetros" << endl;
15       nome = pNome;
16       energia = pEnergia;
17       municao = pMunicao;
18       return;
19    }
20
21    Espaconave(string pNome){  //construtor (ao mesmo tempo construtor de cópia)
22       cout << "Executando método construtor com 1 parâmetro" << endl;
23       nome = pNome;
24       municao = 0;
25       energia = 0;
26       return;
27    }
28
29    Espaconave(int pMunicao, int pEnergia ){  //construtor
30       cout << "Executando método construtor com 2 parâmetros" << endl;
31       nome = "unknown";
32       municao = pMunicao;
33       energia = pEnergia;
34       return;
35    }
36
37    Espaconave( int pMunicao ){ //outro construtor de cópia
38       cout << "Executando método construtor de cópia" << endl;
39       nome = "unknown";
40       municao = pMunicao;
41       energia = 0;
42    }
43
44    void imprimeDados(){ //definido um método da classe
45       cout << "Esta nave se chama " << nome
46            << " e tem " << energia
47            << " de energia e " << municao
48            << " tiros." << endl;
49       return;
```

```
50      }
51
52      void setEnergia( int pEnergia ){
53          energia = pEnergia;
54          return;
55      }
56
57      void setMunicao( int pMunicao ){
58          municao = pMunicao;
59          return;
60      }
61  }; //não esquecer do ; ao final
62
63  int main(){
64    Espaconave YWing = string("YWing");
65    YWing.imprimeDados();
66
67    Espaconave Unknown = 200;
68    Unknown.imprimeDados();
69
70    cout << "Game Over!!!" << endl;
71    return 0;
72  }
```

11 Método Destrutor

- Pode-se definir uma função que seja executada assim que o fim do escopo de um objeto é atingido.
- Este método é chamado de **destrutor**. Deve ser definido sempre que haja alocação de memória, fazendo a liberação da mesma.
- O destrutor deve ser definido com o nome da classe precedido por ~. O destrutor não pode ter valor de retorno e nem parâmetros na sua definição.

De modo similar ao construtor, podemos definir um método destrutor. Este método será chamado ao final do bloco onde o objeto foi definido.

A sintaxe da linguagem exige que o nome do método destrutor seja o mesmo da classe precedido do sinal ~.

Podemos definir somente um método destrutor para cada classe. Este método não pode ter nenhum parâmetro e também não pode retornar nenhum valor, pois a chamada será automática.

Devemos **obrigatoriamente** definir um método destrutor quando no método construtor ou em algum outro método é *alocado memória dinâmica*.

Programa 9: Classe com método destrutor.

```
1  /* programa_009.cpp */
2  #include "biblaureano.h"
```

```
3
4  //aqui é a definição do modelo
5  class Espaconave{
6     private:     // Definição de membros privados
7        int energia;
8        int municao;
9        string nome;
10
11    public :     // Definição de membros publicos
12
13    Espaconave( string pNome, int pEnergia, int pMunicao){ //construtor
14       cout << "Executando método construtor com 3 parâmetros" << endl;
15       nome = pNome;
16       energia = pEnergia;
17       municao = pMunicao;
18       return;
19    }
20
21    Espaconave(string pNome){  //construtor (ao mesmo tempo construtor de cópia)
22       cout << "Executando método construtor com 1 parâmetro" << endl;
23       nome = pNome;
24       municao = 0;
25       energia = 0;
26       return;
27    }
28
29    Espaconave(int pMunicao, int pEnergia ){  //construtor
30       cout << "Executando método construtor com 2 parâmetros" << endl;
31       nome = "unknown";
32       municao = pMunicao;
33       energia = pEnergia;
34       return;
35    }
36
37    Espaconave( int pMunicao ){ //outro construtor de cópia
38       cout << "Executando método construtor de cópia" << endl;
39       nome = "unknown";
40       municao = pMunicao;
41       energia = 0;
42    }
43
44    ~Espaconave(){ //método destrutor
45       cout << "Espaçonave " << nome
46            << " foi destruída..." << endl;
47    }
48
49    void imprimeDados(){ //definido um método da classe
50       cout << "Esta nave se chama " << nome
51            << " e tem " << energia
52            << " de energia e " << municao
53            << " tiros." << endl;
54       return;
55    }
56
57    void setEnergia( int pEnergia ){
58       energia = pEnergia;
59       return;
60    }
61
62    void setMunicao( int pMunicao ){
63       municao = pMunicao;
64       return;
65    }
66 }; //não esquecer do ; ao final
67
68 int main(){
69   Espaconave YWing = string("YWing");
70   YWing.imprimeDados();
71
72   Espaconave Unknown = 200;
73   Unknown.imprimeDados();
74
75   cout << "Game Over!!!" << endl;
```

```
76    return 0;
77  }
```

12 Objetos como parâmetros e retornando objetos

- Pode-se definir um objeto para ser usado como parâmetro em qualquer função ou método.
- Dentro da função pode-se ter acesso a todos os métodos e atributos públicos.
- Deve-se usar o nome do parâmetro para se acessar os atributos e métodos.
- Para o parâmetro não será chamado o construtor pois é somente passado uma cópia do objeto na pilha.

Deve-se sempre considerar uma instância de um objeto como sendo uma variável, mais explicitamente uma variável do tipo estrutura. Sendo assim é possível passar objetos como parâmetros de uma função ou de um método.

Dentro da função o objeto é acessado através do nome do parâmetro colocado na função. Pode-se realizar qualquer operação com o objeto.

Na entrada da função o objeto será copiado para o parâmetro e sendo assim não será executado nenhum construtor para o parâmetro. Mas ao término da função será executado o método destrutor do objeto copiado.

- Um instância de um objeto pode ser retornado de uma função ou método.
- Na definição da função é colocado a classe como o tipo de retorno da função.
- No uso desta função ou método pode-se atribuir a saída da função para um outro objeto da mesma classe.
- Os atributos são copiados para o objeto que recebe o valor da função.

De maneira análoga à parâmetros, pode-se definir que uma função retornará um objeto. Para indicar isto deve ser colocado o nome da classe como o tipo de retorno de uma função.

Na chamada desta função, o seu retorno será atribuído para alguma outra instância da mesma classe, ocorrendo a copia dos atributos da instância retornada para a instância atribuída.

Este procedimento pode ser efetuado sem problemas desde que não se tenha alocação de memória dinâmica no construtor ou em algum outro método e foi feito a liberação no destrutor desta memória dinâmica.

Programa 10: Objetos como parâmetros de funções e como retorno de funções.

```
/* programa_010.cpp */
#include "biblaureano.h"

//aqui é a definição do modelo
class Espaconave{
   private:    // Definição de membros privados
      int energia;
      int municao;
      string nome;

   public :    // Definição de membros publicos

   Espaconave( string pNome, int pEnergia, int pMunicao){//construtor
      cout << "Executando método construtor com 3 parâmetros" << endl;
      nome = pNome;
      energia = pEnergia;
      municao = pMunicao;
      return;
   }

   Espaconave(string pNome){  //construtor (ao mesmo tempo construtor de cópia)
      cout << "Executando método construtor com 1 parâmetro" << endl;
      nome = pNome;
      municao = 0;
      energia = 0;
      return;
   }

   Espaconave(int pMunicao, int pEnergia ){  //construtor
      cout << "Executando método construtor com 2 parâmetros" << endl;
      nome = "unknown";
      municao = pMunicao;
      energia = pEnergia;
      return;
   }

   Espaconave( int pMunicao ){ //outro construtor de cópia
      cout << "Executando método construtor de cópia" << endl;
      nome = "unknown";
      municao = pMunicao;
      energia = 0;
   }

   ~Espaconave(){ //método destrutor
      cout << "Espaçonave " << nome
           << " foi destruída..." << endl;
   }

   void imprimeDados(){ //definido um método da classe
      cout << "Esta nave se chama " << nome
           << " e tem " << energia
           << " de energia e " << municao
           << " tiros." << endl;
      return;
   }

   void setEnergia( int pEnergia ){
      energia = pEnergia;
      return;
   }

   void setMunicao( int pMunicao ){
      municao = pMunicao;
      return;
   }

   int getMunicao(){
      return municao;
   }

   int getEnergia(){
```

```
72        return energia;
73     }
74
75     string getNome(){
76        return nome;
77     }
78  }; //não esquecer do ; ao final
79
80  //definição da função
81  Espaconave Attack( Espaconave Nave01, Espaconave Nave02);
82
83  int main(){
84    Espaconave YWing = string("YWing");
85    YWing.imprimeDados();
86
87    Espaconave Unknown = 200;
88    Unknown.imprimeDados();
89
90    cout << "Antes de chamar Attack..." << endl;
91    Espaconave Vencedor = Attack( YWing, Unknown );
92    Vencedor.imprimeDados();
93    cout << "Depois de chamar Attack..." << endl;
94
95    cout << "Game Over!!!" << endl;
96    return 0;
97  }
98
99  Espaconave Attack( Espaconave Nave01, Espaconave Nave02){
100    cout << "Dentro da função attack..." << endl;
101    if( Nave01.getEnergia() > Nave02.getEnergia() ){
102       return Nave01;
103    }
104    else{
105       return Nave02;
106    }
107 }
```

13 Objetos e referência

- Uma instância passada por valor para uma função é uma cópia do objeto.
- Caso se altere algum atributo da instância por valor, não será alterado o atributo do parâmetro colocado na chamada da função.
- Dentro da função, pode-se alterar os atributos do objeto colocado no parâmetro bastando definir esse parâmetro como sendo uma referência para o objeto.

Um objeto passado como parâmetro sempre será passado como um cópia do objeto. Se dentro da função se alterar algum atributo, está sendo alterando o atributo da cópia e não do objeto original passado como parâmetro.

Em caso que exista a necessidade de que dentro de uma função se altere o valor de um atributo e esta alteração

reflita no atributo do objeto colocado como parâmetro, deve-se definir este parâmetro como se fosse uma referência ao objeto.

Programa 11: Passagem de objetos como referência para funções

```
/* programa_011.cpp */
#include "biblaureano.h"

//aqui é a definição do modelo
class Espaconave{
   private:    // Definição de membros privados
      int energia;
      int municao;
      string nome;

   public :    // Definição de membros publicos

   Espaconave( string pNome, int pEnergia, int pMunicao){ //construtor
      cout << "Executando método construtor com 3 parâmetros" << endl;
      nome = pNome;
      energia = pEnergia;
      municao = pMunicao;
      return;
   }

   Espaconave(string pNome){  //construtor (ao mesmo tempo construtor de cópia)
      cout << "Executando método construtor com 1 parâmetro" << endl;
      nome = pNome;
      municao = 0;
      energia = 0;
      return;
   }

   Espaconave(int pMunicao, int pEnergia ){  //construtor
      cout << "Executando método construtor com 2 parâmetros" << endl;
      nome = "unknown";
      municao = pMunicao;
      energia = pEnergia;
      return;
   }

   Espaconave( int pMunicao ){ //outro construtor de cópia
      cout << "Executando método construtor de cópia" << endl;
      nome = "unknown";
      municao = pMunicao;
      energia = 0;
   }

   ~Espaconave(){ //método destrutor
      cout << "Espaçonave " << nome
           << " foi destruída..." << endl;
   }

   void imprimeDados(){ //definido um método da classe
      cout << "Esta nave se chama " << nome
           << " e tem " << energia
           << " de energia e " << municao
           << " tiros." << endl;
      return;
   }

   void setEnergia( int pEnergia ){
      energia = pEnergia;
      return;
   }

   void setMunicao( int pMunicao ){
      municao = pMunicao;
      return;
   }

   int getMunicao(){
```

```
68        return municao;
69     }
70
71     int getEnergia(){
72        return energia;
73     }
74
75     string getNome(){
76        return nome;
77     }
78  }; //não esquecer do ; ao final
79
80  //definição da função, observe que agora é por referência
81  Espaconave & Attack( Espaconave & Nave01, Espaconave & Nave02);
82
83  int main(){
84    Espaconave YWing = string("YWing");
85    YWing.imprimeDados();
86
87    Espaconave Unknown = 200;
88    Unknown.imprimeDados();
89
90    cout << "Antes de chamar Attack..." << endl;
91    Espaconave & Vencedor = Attack( YWing, Unknown );
92    Vencedor.imprimeDados();
93    cout << "Depois de chamar Attack..." << endl;
94
95    cout << "Game Over!!!" << endl;
96    return 0;
97  }
98
99  Espaconave & Attack( Espaconave & Nave01, Espaconave & Nave02){
100    cout << "Dentro da função attack..." << endl;
101    if( Nave01.getEnergia() > Nave02.getEnergia() ){
102       return Nave01;
103    }
104    else{
105       return Nave02;
106    }
107 }
```

14 Vetores de Objetos

- Como qualquer tipo em C++, pode-se criar um vetor ou matriz de objetos.
- Será chamado um construtor para cada ocorrência definida no vetor ou matriz.
- Para se acessar um atributo ou método de uma única ocorrência de um objeto deve-se usar a sintaxe: `nome_objeto[indice].atributo_publico` ou `nome_objeto[indice].metodo(parâmetro)`

Como qualquer tipo da linguagem pode-se construir um vetor ou matriz de objetos. A definição é a mesma usada

para um vetor ou matriz de um tipo básico ou de estrutura.

Como na definição de um vetor está se definindo várias instâncias, a linguagem C++ chamará tantos construtores quantos objetos definidos, ou seja, caso seja definido um vetor com 100 ocorrências, será ativado o método construtor 100 vezes.

O acesso à uma única ocorrência de um objeto se dará através da utilização de índices, conforme padrão para vetores e matrizes. Caso seja preciso o acesso a algum método deve-se indicar a ocorrência desejada e o método indicado através da seguinte sintaxe: `nome_objeto[indice].nome_metodo(parâmetros...)`

O acesso a atributos públicos segue a mesma regra sintática. Coloca-se o nome do atributo logo após a identificação do objeto através de índices. Também é possível utilizar vetores dinâmicos (*vector*) de classes:

Programa 12: Vetores de Objetos.

```
/* programa_012.cpp */
#include "biblaureano.h"

//aqui é a definição do modelo
class Espaconave{
   protected : // Definição de membros protegidos
   private:    // Definição de membros privados
      int energia;
      int municao;
      string nome;

   public :    // Definição de membros publicos

   Espaconave( string pNome, int pEnergia, int pMunicao){ //construtor
      cout << "Executando método construtor com 3 parâmetros" << endl;
      nome = pNome;
      energia = pEnergia;
      municao = pMunicao;
      return;
   }

   Espaconave(string pNome){  //construtor (ao mesmo tempo construtor de cópia)
      cout << "Executando método construtor com 1 parâmetro" << endl;
      nome = pNome;
      municao = 0;
      energia = 0;
      return;
   }

   Espaconave(int pMunicao, int pEnergia ){  //construtor
      cout << "Executando método construtor com 2 parâmetros" << endl;
      nome = "unknown";
      municao = pMunicao;
      energia = pEnergia;
      return;
   }

   Espaconave( int pMunicao ){ //outro construtor de cópia
      cout << "Executando método construtor de cópia" << endl;
      nome = "unknown";
      municao = pMunicao;
      energia = 0;
      return;
   }

   Espaconave(){ //construtor vazio
      nome = "unknown";
      municao = 0;
      energia = 0;
      return;
   }

   ~Espaconave(){ //método destrutor
      cout << "Espaçonave " << nome
```

```
55              << " foi destruída..." << endl;
56      }
57
58      void imprimeDados(){ //definido um método da classe
59          cout << "Esta nave se chama " << nome
60               << " e tem " << energia
61               << " de energia e " << municao
62               << " tiros." << endl;
63          return;
64      }
65
66      void setEnergia( int pEnergia ){
67          energia = pEnergia;
68          return;
69      }
70
71      void setMunicao( int pMunicao ){
72          municao = pMunicao;
73          return;
74      }
75
76      void setNome( string pNome ){
77          nome = pNome;
78          return;
79      }
80
81      int getMunicao(){
82          return municao;
83      }
84
85      int getEnergia(){
86          return energia;
87      }
88
89      string getNome(){
90          return nome;
91      }
92  }; //não esquecer do ; ao final
93
94  int main(){
95    //vetor de classe
96    Espaconave minhasNaves[10];
97    for( int i = 0; i<10; ++i){
98        string nome = "Nave " + numeroToString(i);
99
100       minhasNaves[i].setNome( nome );
101       minhasNaves[i].setEnergia(i*300);
102       minhasNaves[i].setMunicao(i*200);
103   }
104
105   for( int i = 0; i<10; ++i){
106       minhasNaves[i].imprimeDados();
107   }
108
109   //vetor dinâmico
110   vector <Espaconave> outrasNaves;
111   for( int i=0; i<10;++i){
112       //a fim facilitar a inclusão no vector
113       //cria-se uma variável auxiliar para o push_back
114       Espaconave e;
115       string nome = "OutraNave " + numeroToString(i);
116
117       e.setNome( nome );
118       e.setEnergia(i*300);
119       e.setMunicao(i*200);
120       outrasNaves.push_back(e);
121   }
122
123   for( int i = 0; i<10; ++i){
124       outrasNaves[i].imprimeDados();
125   }
126
127
```

```
128   cout << "Game Over!!!" << endl;
129   return 0;
130 }
```

15 A importância dos métodos get e set

A utilização de atributos **private** ou **protected** (no caso de herança) garante a proteção do conteúdo dados a estes atributos. Cada classe é responsável pelo correto tratamento dos valores atribuídos a seus atributos. É possível imaginar uma espaçonave com energia ou munição negativas? Nos programas anteriores, os métodos **setXXXX** não realizavam nenhum tratamento adequado para os valores de entrada, no programa 13 esta validação é realizada.

Definir todos os atributos como privados e métodos públicos para implementar as lógicas de acesso e alteração é quase uma regra da orientação a objetos. O intuito é ter sempre um controle centralizado do dados dos objetos para facilitar a manutenção do sistema e a detecção de erros.

Programa 13: Validando dados nos métodos

```
1  /* programa_013.cpp */
2  #include "biblaureano.h"
3
4  //aqui é a definição do modelo
5  class Espaconave{
6     private:    // Definição de membros privados
7        int energia;
8        int municao;
9        string nome;
10
11    public :    // Definição de membros publicos
12
13    Espaconave(string pNome){
14       nome = pNome;
15       municao = 0; //é importante inicializar todos os atributos
16       energia = 0; //com valores padrões
17       return;
18    }
19
20    ~Espaconave(){ //método destrutor
21       cout << "Espaçonave " << nome
22            << " foi destruída..." << endl;
23    }
24
25    void imprimeDados(){ //definido um método da classe
26       cout << "Esta nave se chama " << nome
27            << " e tem " << energia
28            << " de energia e " << municao
29            << " tiros." << endl;
30       return;
31    }
32
33    void setEnergia( int pEnergia ){
34       if( pEnergia >= 0){ //validação da entrada
35          energia = pEnergia;
36       }
37       return;
38    }
```

```
39
40    void setMunicao( int pMunicao ){
41       if( pMunicao >= 0 ){ //validação da entrada
42          municao = pMunicao;
43       }
44       return;
45    }
46
47    int getMunicao(){
48       return municao;
49    }
50
51    int getEnergia(){
52       return energia;
53    }
54
55    string getNome(){
56       return nome;
57    }
58 }; //não esquecer do ; ao final
59
60 int main(){
61   Espaconave YWing = string("YWing");
62
63   YWing.imprimeDados();
64   YWing.setMunicao(-1);
65   YWing.setEnergia(-1);
66   YWing.imprimeDados();
67   YWing.setMunicao(300);
68   YWing.setEnergia(500);
69   YWing.imprimeDados();
70
71   cout << "Game Over!!!" << endl;
72   return 0;
73 }
```

Sempre declare os seus atributos como **private** ou **protected** (no caso de herança) e crie métodos **setXXXX** e **getXXXX** para alterar ou ler os valores dos atributos.