

Programação Orientada a Objetos

Introdução à Orientação a Objetos

- Interfaces e Encapsulamento

Objetos e Interfaces

- A partir do instante que uma classe é criada...
 - pode-se criar (instanciar) quantos objetos da classe quanto forem necessários e manipular estes objetos como se fossem elementos que existissem no problema sendo solucionado.
- Como fazemos um objeto realizar trabalho útil?
 - Cada objeto tem operações (funções-membro) para desempenhar suas atividades;
 - Cada objeto somente pode responder a determinadas requisições;
 - O conjunto de métodos de um objeto é conhecido como sua *interface*.

Objetos e Interfaces

- Exemplo de Interface:



Nome da classe: —————→

Lampada
liga(); desliga() aumenta_luminosidade(); diminui_luminosidade();

Interface: →

```
Lampada luz;  
luz.liga();
```

Objetos e Interfaces

- Exemplo de Interface:

```
class Lampada {  
    private:  
        ...  
    public:  
        void liga();  
        void desliga();  
        void aumenta_luminosidade(double quanto);  
        void diminui_luminosidade(double quanto);  
};
```

Lampada

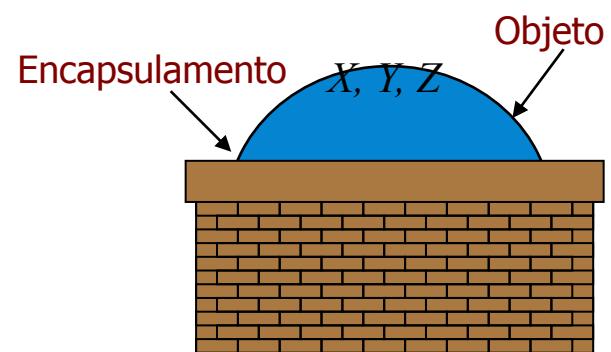
liga();
desliga()
aumenta_luminosidade();
diminui_luminosidade();

Objetos e Interfaces

- Que métodos (funções) podemos chamar?
 - Isto é a **interface**.
- Onde estes métodos estão codificados, como são implementados?
 - Isto é a **implementação**.
- Geralmente não interessa a quem usa o objeto conhecer os detalhes da implementação ...

Interface, Implementação e Encapsulamento

- O cliente usa os serviços de um objeto através de sua interface
- O criador da classe expõe o mínimo da interface para o cliente, escondendo o resto. Por quê?
 - Porque se está escondido, pode ser mudado sem **quebrar** o programa do cliente;
 - Se fosse dado total acesso à implementação interna, o cliente poderia **quebrar a integridade do objeto**.

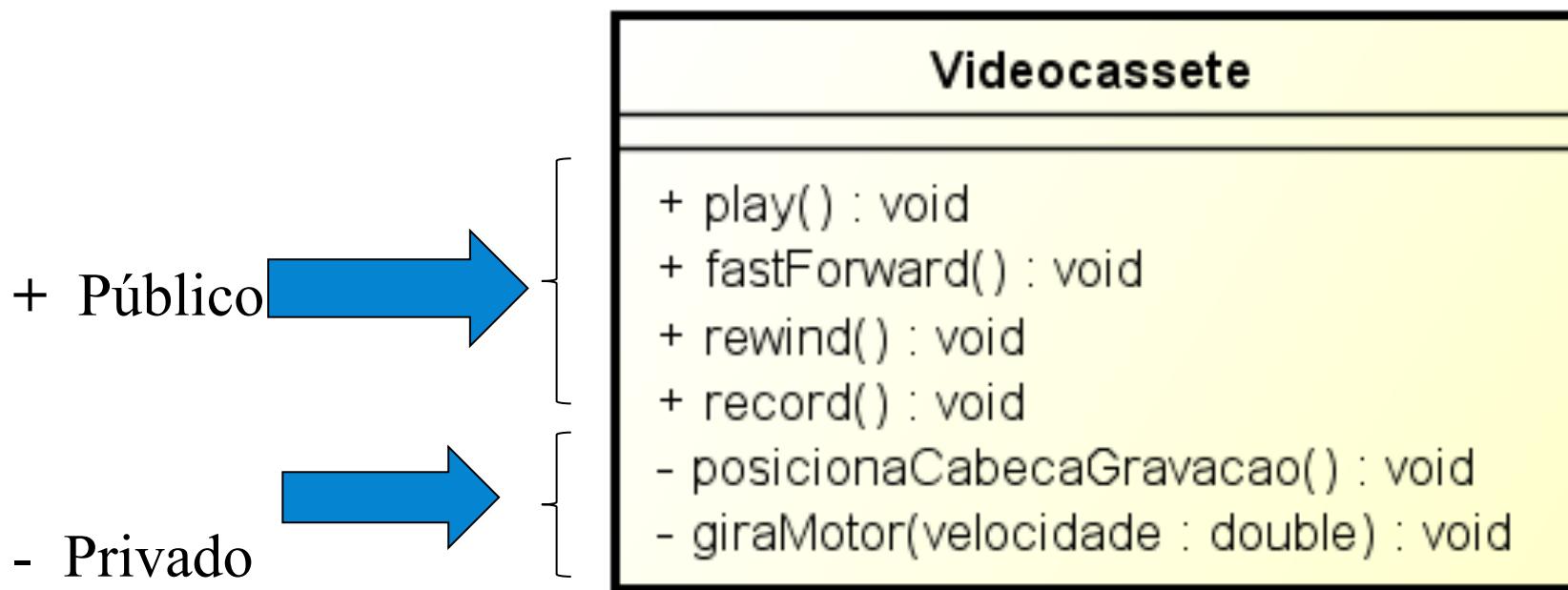


Encapsulamento

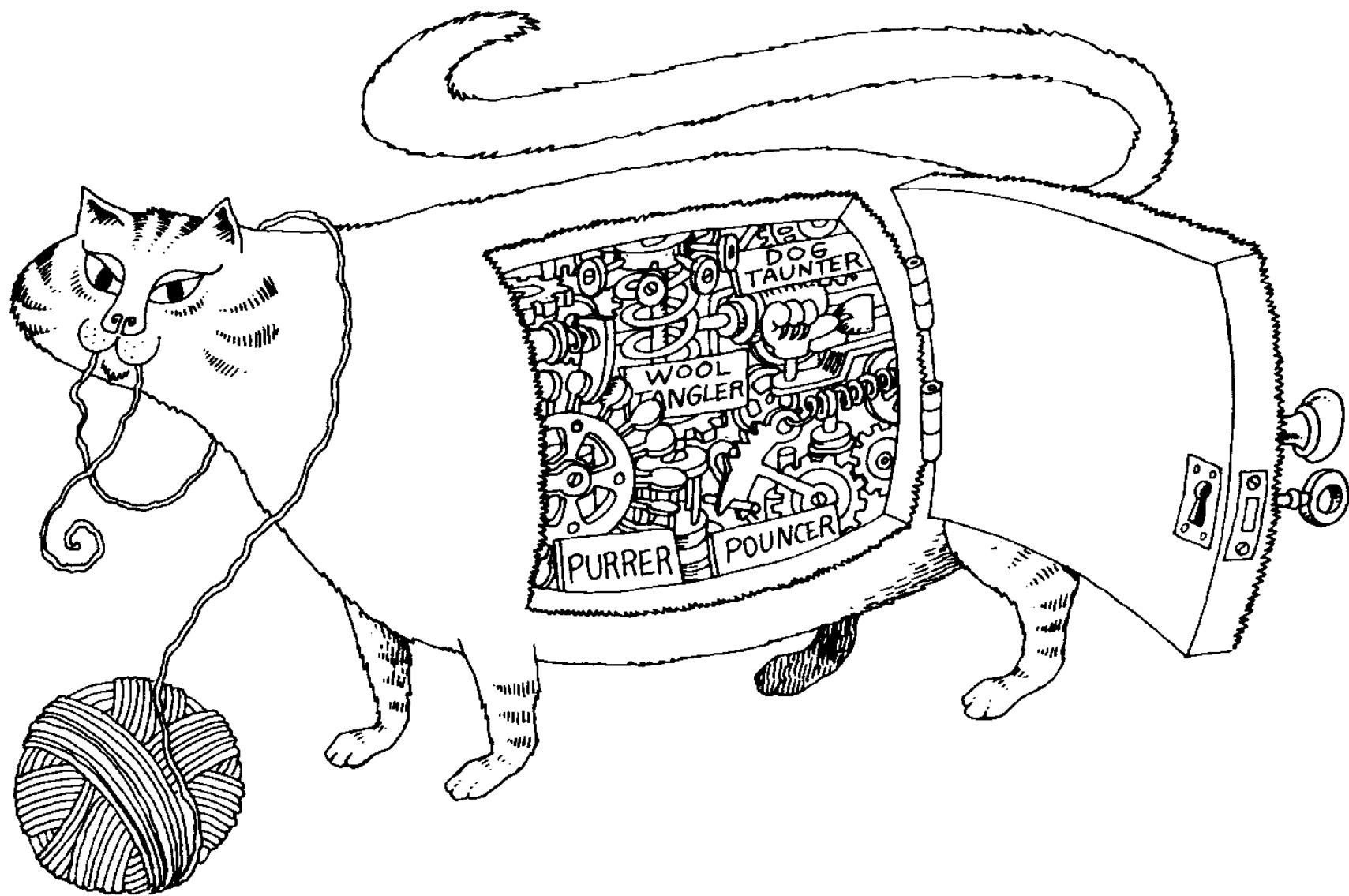
- A estratégia utilizada para garantir que determinadas partes de uma classe não são acessíveis por seus clientes é denominada **controle de acesso**.
- A interface não apresenta necessariamente todos os métodos de um objeto, mas somente aqueles que podem ser acessados pelo público em geral, os chamados **métodos públicos**. Existem métodos internos aos objetos: **os métodos privados**.

Encapsulamento

- Exemplo:
 - Videocassete:



Encapsulamento - Idéia



Encapsulamento

- Em C++, se não for especificado o controle de acesso, os atributos e métodos são ...
 - Em uma classe: **privados** por *default*
 - Em uma *struct*: **públicos** por *default*

Encapsulamento

```
// B.h
class B {
    private:
        char j; float f;
    public:
        int i;
        void func();
    };
// B.cpp
void B::func() {
    i = 0; // OK!
    j = '0'; // OK!
    f = 0.0; // OK!
};
```

```
// main.cpp
int main() {
    B b;
    b.i = 1; // OK, public
    b.func(); // OK, public
    // b.j = '1'; // Illegal, private
    // b.f = 1.0; // Illegal, private
}
```

Encapsulamento

```
4 #define EMPLOYEE_H
5
6 class Employee
7 {
8     public:
9         void inititalize(string n, double s);
10        void cleanup();
11        string getName();
12        double getSalary();
13        void raiseSalary(double byPercent);
14
15    private:
16        string name;
17        double salary;
18
19 };
```

```
1 #include "Employee.h"
2
3     void Employee::initialize(string n, double s) {
4         name = n;
5         salary = s;
6     }
7
8     void Employee::cleanup() {
9
10    }
11
12    string Employee::getName() {
13        return name;
14    }
15
16    double Employee::getSalary() {
17        return salary;
18    }
19
20    void Employee::raiseSalary(double byPercent) {
21        double raise = salary * byPercent/100;
22        salary += raise;
23    }
24
```

```
1 #include <iostream>
2 #include<vector>
3 #include "Employee.h"
4
5 using namespace std;
6
7 int main()
8 {
9
10     // cria e inicializa os funcionarios
11     Employee e1, e2, e3;
12     e1.initialize("Carl Cracker", 75000);
13     e2.initialize("Harry Hacker", 50000);
14     e3.initialize("Tony Tester", 40000);
15
16     // preenche uma lista com os funcionarios
17     vector<Employee> v;
18     v.push_back(e1);
19     v.push_back(e2);
20     v.push_back(e3);
21
22     // aumenta os salarios de todos os funcionario em 5%
23     for(int i=0; i<v.size(); i++){
24         v[i].raiseSalary(5);
25     }
26
27     //imprime as informacoes de todos os funcionarios
28     for(int i=0; i<v.size(); i++){
29         cout << "name = " << v[i].getName() << " - salary = " << v[i].getSalary() << endl;
30     }
31
32     return 0;
33 }
34
```

Alguns Benefícios do Encapsulamento:

1) Garantia de que o atributo não será mudado !!!

- Considere a função-membro:

```
string Employee::getName() {  
    return name;  
}
```

- atributo *name* é um campo de leitura somente.
- Após inicializá-lo (configurá-lo) com o *initialize()*, não há nenhum método disponível na classe para alterá-lo. Assim, tem-se a garantia de que *name* nunca será corrompido.

Alguns Benefícios do Encapsulamento:

2) Facilidade de Depuração:

- Considere o método:

```
void Employee::raiseSalary(double byPercent){  
    double raise = salary * byPercent / 100;  
    salary += raise;  
}
```

- atributo *salary* só pode ser alterado pelo método *raiseSalary*. Caso, alguma vez, seu valor esteja inconsistente, somente esse método precisa ser depurado.
- Se *salary* fosse *public*, o culpado por bagunçar o valor poderia estar em qualquer lugar!

Alguns Benefícios do Encapsulamento

3) Possibilidade de Alteração da implementação interna da classe sem afetar nenhum código cliente (que faz uso da classe).

Exemplo: O atributo de string *name* na classe *Employee* poderia ser substituído por dois outros atributos:

```
string firstName;  
string lastName;
```

Assim, o método *getName()* poderá ser alterado para retornar:

```
return firstName + “ ” + lastName;
```

Essa mudança é completamente invisível para os clientes da classe!!!

Encapsulamento

- Em POO, para obter e configurar (alterar) o valor de um atributo, é recomendado fornecer 3 itens:
 - atributo de dados privado;
 - Função-membro de acesso público; *// getter*
 - Funcao-membro modificador público; *// setter*

Leitura

- Thinking in C++. Vol. 1. 2nd Edition
 - Caps. 1, 4 e 5.