Curso de Ciência da Computação

Algoritmos e Programação de Computadores 2per

Programação Orientada a Objetos POO

Profa. Fernanda dos Santos Cunha

Abstração

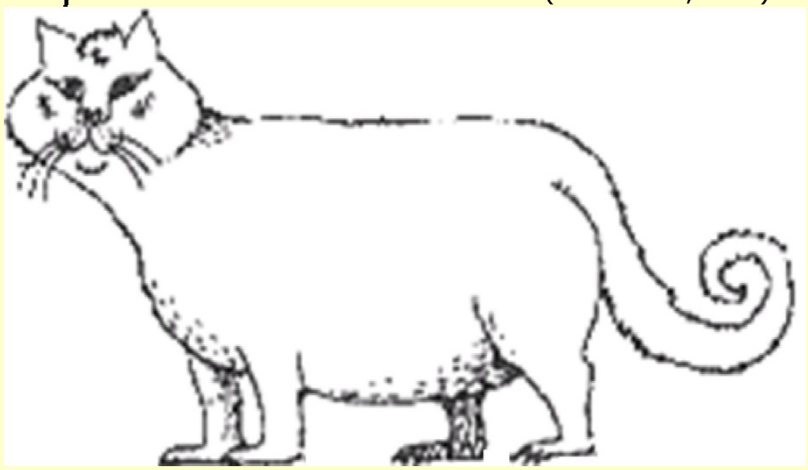
- Processo de **identificar os aspectos essenciais** de um contexto qualquer, ignorando características menos importantes ou acidentais.
- **"Uma abstração"** é o resultado deste processo.
- Não se analisa o "todo", em POO é importante analisar as partes para entender o todo.

Abstração

- Abstração é uma das formas fundamentais para lidar com a complexidade.
  - Quando se quer diminuir a complexidade de alguma coisa, ignora-se detalhes sobre as partes para concentrar a atenção no nível mais alto de um problema.

Abstração

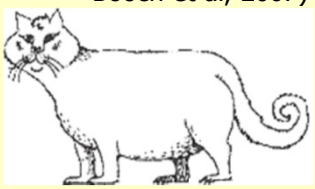
- O que se vê ??? (Booch et al, 2007)



Abstração

- Níveis de abstração
  - Animal
    - Mamífero
      - Gato
        - »Gato Siamês

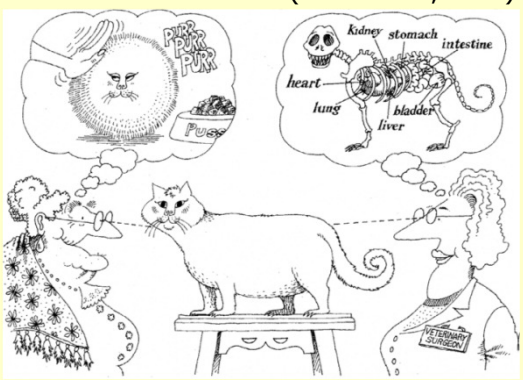
Booch et al, 2007)



Visões diferentes

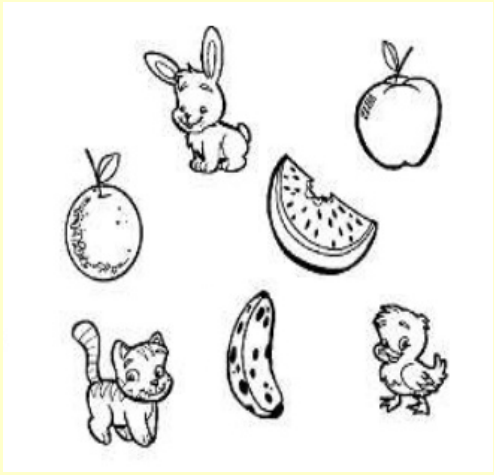
- Identificar os aspectos essenciais depende **do observador e do fenômeno observado**.
- Diferentes **observadores**
- Diferentes **necessidades**

(Booch et al, 2007)

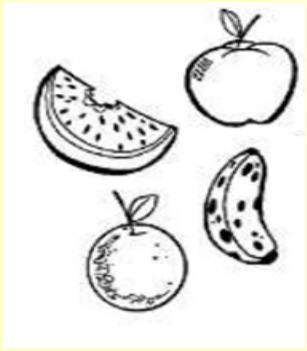


Classificação

- Como classificar estas "coisas" ??
- Classificar é uma forma de abstração



Classificação



ANIMAL **FRUTA**

ABSTRAÇÕES

Classificação

- O que foi observado em cada "coisa"?
- Similaridades?
- Características comuns
- Comportamento comum
- Porém cada "coisa" é um indivíduo dentro do grupo
- "Coisa" = **objeto !!!!**

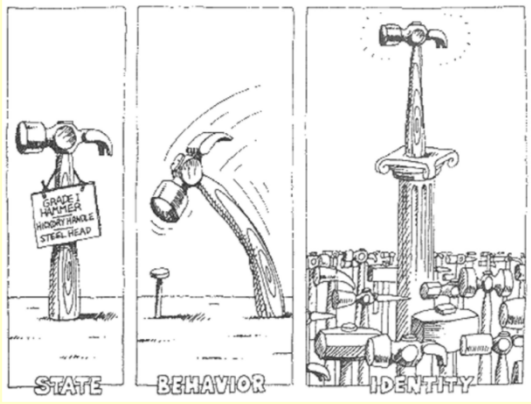
Objeto

- Entidade com **estado** e **comportamento** específico
- Objetos de software modelam objetos reais
- Estado de um objeto
  - Objeto se caracteriza por um conjunto de **atributos**
    - Características (propriedades) presentes em todos os objetos de uma mesma classe
  - O **conjunto dos valores** de cada atributo em um determinado momento representa o **estado de um objeto**

Objeto

- **Estado**
  - Martelo tipo 1
  - Cabo de nogueira
  - Cabeça de aço
- **Comportamento**
  - Martelar
- **Identidade**
  - Meu martelo

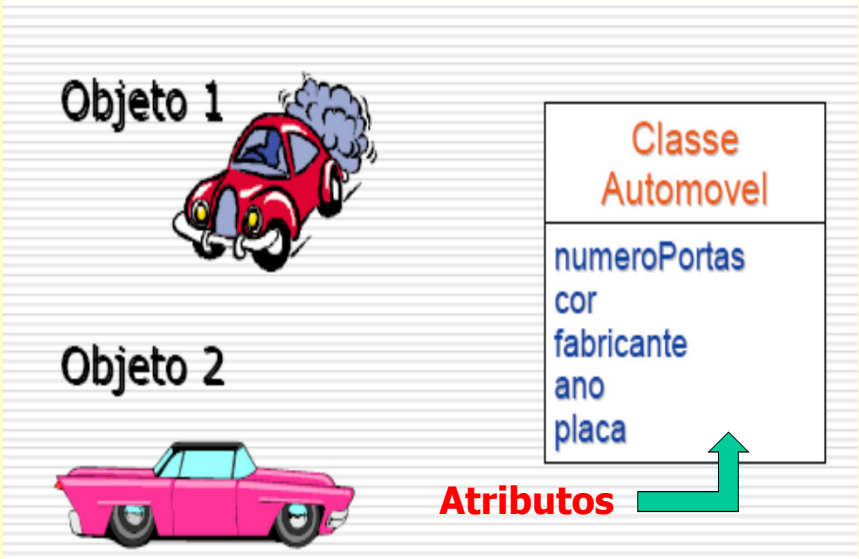
(Booch et al, 2007)



Atributos X Var. locais

- Um **atributo** é uma característica relevante para o objeto.
  - Não faz sentido falar do objeto sem esse atributo.
- Variáveis temporárias não devem ser declaradas como atributos. Ex.: variável para controle de laço, para armazenar cálculos intermediários, etc.

Estado de um objeto

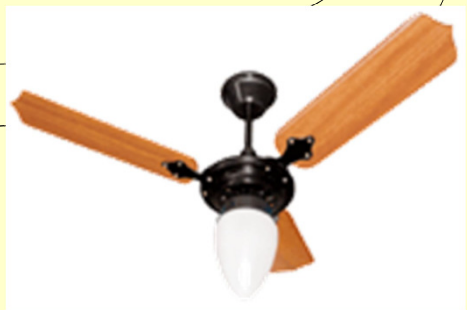


Estado de um objeto

- Objeto **"ventilador 1"**
- Número de pás: **3**
- Material da pá: **MADEIRA**
- Número de velocidades: **3**
- Cor: **MOGNO**
- Tem exaustor: **SIM**
- Tem lustre: **SIM**

Estado atual

Atributos



Estado de um objeto

- Objeto **"ventilador 2"**
- Número de pás: **2**
- Material da pá: **PLÁSTICO**
- Número de velocidades: **4**
- Cor: **VERDE**
- Tem exaustor: **NÃO**
- Tem lustre: **SIM**

Comportamento de um objeto

- Como pedir ao objeto que ele faça alguma ação?
- Quais as ações sobre um ventilador?
  - Alterar número de pás
  - Obter número de pás ?
  - Alterar cor
  - Obter a cor ?
  - ...

Comportamento de um objeto

- Outras ações ?
  - Ligar / desligar modo ventilador
  - Ligar / desligar modo ventilador
  - Acender / desligar a luz
  - Aumentar a velocidade
  - ...

Comportamento de um objeto

- Mais ações ?
  - Está ligado ou não ?
  - Em qual modo está ligado ?
  - A luz está acesa ?
  - Qual a velocidade atual ?
  - ...

Comportamento de um objeto

- Como você identificou o comportamento ?
  - Qual a relação com a abstração ?
  - Qual seu ponto de vista?
    - Desenvolver um sistema de controle de estoque ?
    - Desenvolver um sistema de controle automático de ventiladores em um prédio inteligente ?
  - **O ponto de vista (necessidades) afeta a abstração !!**

Comportamento de um objeto

- O comportamento é representado por um conjunto de operações
  - Elas são as responsabilidades da classe, determinando os que os objetos podem fazer (ações)

Identidade de um objeto

ventilador 1 ventilador 33

Objetos podem ter um mesmo estado...

Mas eles ainda são objetos diferentes

Classe

- **Modelo** a partir do qual se criam objetos
- Define **dados** (atributos) e **comportamento** (operações) comuns a todos os objetos criados a partir dela

Classes x objetos

- Objetos são criados a partir de uma classe
- Cada objeto é uma instância de sua classe
- Criação = instanciação

Criar novo objeto

Instanciando e destruindo objetos

- Deve-se solicitar a classe que "instancie" um objeto – **Construtor:** função que é chamada sempre que é criado um objeto de uma classe, faz a inicialização
- O próprio objeto resolve quando deve ser "destruído" – **Destrutor:** função que é chamada sempre que o escopo de duração do objeto de uma classe encerra-se - faz a "limpeza".

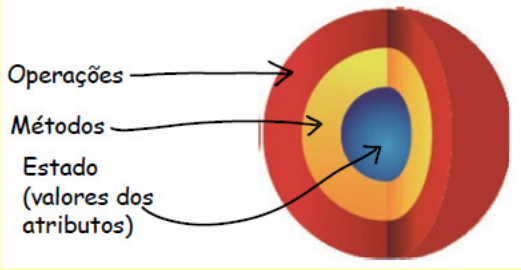
Encapsulamento

- Mecanismo para agrupar os atributos e as operações que manipulam estes atributos

- **Esconder o estado** do objeto
  - A única forma de acessar ou modificar um objeto são as operações
  - Proteção ao acesso descontrolado

Encapsulamento

- **Esconder a implementação das operações** do objeto
  - Método é a implementação de uma operação



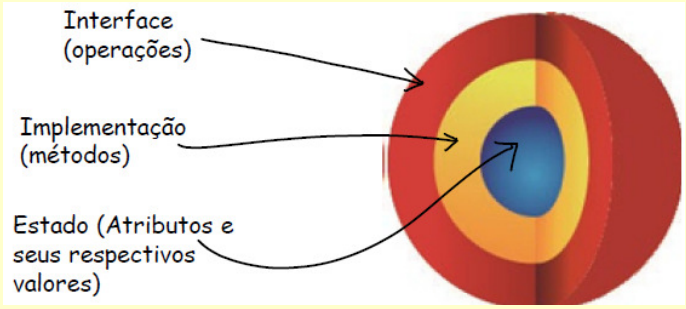
Ex.: a operação *imprimir* pode ser implementada de forma distinta, dependendo se o arquivo a ser impresso contém apenas texto ASCII, é um arquivo de um processador de texto ou binário.

Encapsulamento e Interface

- Toda classe implementa (realiza) uma **interface**
- Interface define conjunto de operações visíveis (públicas) de uma classe
  - Quais operações podem ser invocadas nos objetos de uma determinada classe

Encapsulamento e Interface

- Encapsulamento permite separação entre a interface (operações) e sua implementação (métodos)



Nomear uma Classe

- O nome da classe deve estar relacionado com o **principal objetivo** dela
- Qual abstração a classe representa ?

- **Princípio da Responsabilidade Única**
  - Uma classe deve ter uma única responsabilidade, que deve ser totalmente encapsulada, e todas as operações devem estar fortemente alinhadas a esta responsabilidade.

Nomear uma Classe

- Utilizar um substantivo que represente a abstração:
  - Professor, Aluno, Cliente, Produto, Automovel, ...
  - NotaFiscal, Contracheque, ContaBancaria, ...
- Nome expresso no singular e adote um estilo de escrita (UpperCamelCase[1])
- Utilize a relação "é um/uma" do objeto para a classe.
  - Ex.: João é um Aluno

1. http://pt.wikipedia.org/wiki/CamelCase

Nomear uma Classe

- Coesão
  - Medida originária do paradigma estruturado, relacionada a Modularidade, que indica o quanto os elementos de um módulo fazem sentidos juntos
- Métrica interna a classe
- Manter ALTA COESÃO
  - Qto maior a coesão, melhor foi aplicado o Princípio da Responsabilidade Única

Nomear um Atributo

- Utilizar nomes significativos, com relação ao problema
- Deve mostrar claramente o que representa
- Evitar mnemônicos e nomes muito longos
- Adote um estilo de escrita
  - LowerCamelCase[1]
  - dataNascimento, nomeCompleto, enderecoResidencial, ...

1. http://pt.wikipedia.org/wiki/CamelCase

Nomear uma Operação

- Utilizar nomes significativos relacionados diretamente a ação realizada
- Evitar mnemônicos e nomes muito longos
- Utilize o domínio do problema para identificar nomes adequados
- Evite muitos parâmetros, mantenha legibilidade na **assinatura da operação**
- Adote um estilo de escrita (LowerCamelCase[1])
  - setEnderecoResidencial, transmitirArquivoContas, getSaldo, ...

Assinatura de uma Operação

- Nome da operação
- Visibilidade (p.ex. pública, ...)
- Lista de parâmetros
- Tipo do retorno

```
public int fazAlgo(float nomeParametro)
```

Nomear uma Operação

- Cada operação deve realizar um serviço
  - Dificuldade em nomear => existe mais de um serviço oferecido pela operação ?
  - Muitos parâmetros => muitos serviços?
  - Lembrar da **coesão**

Exercício

- Quais características estão presentes em qualquer pessoa ??
  Considere o contexto de um sistema bancário.

Exercício

- Pense agora sobre o que você precisará fazer com uma pessoa no sistema bancário.
- Todas as características são necessárias ??

- **Vamos abstrair a classe Pessoa**

Curso de Ciência da Computação

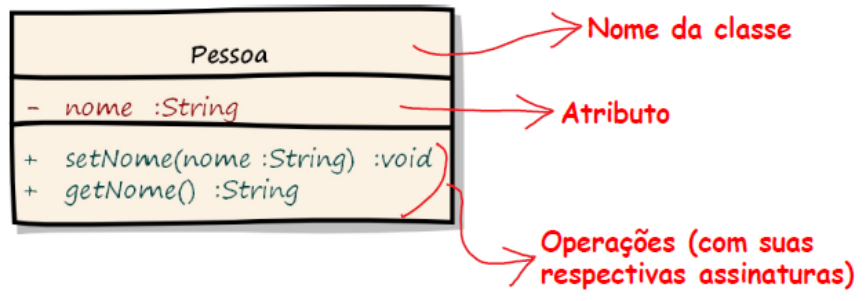
Algoritmos e Programação de Computadores 2per

Programação Orientada a Objetos POO

Profa. Fernanda dos Santos Cunha

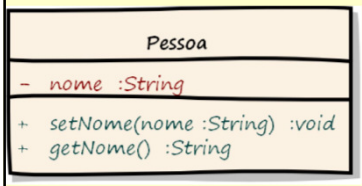
Exemplo

- Em UML



Exemplo

- Em C++



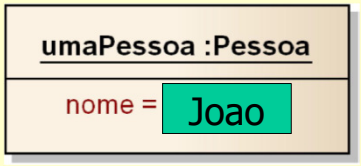

```
class Pessoa{
  public:
    void setNome(string nome);
    string getNome();
  private:
    string nome;
};
void Pessoa::setNome(string nome){
  this->nome = nome;
}
string Pessoa::getNome(){
  return this->nome;
  // OU return nome;
}
```

A palavra reservada "this"

- Dentro de uma operação de objeto ou construtor, **this** é uma referência ao objeto corrente
- Muitas vezes a razão mais comum para o uso do **this** é para diferenciar um atributo de um parâmetro de mesmo nome
  - Exemplo: construtor Data com parâmetros
- O ponteiro **this** é implícito para todas as funções membro de uma classe

Instanciando Objetos


```
class Pessoa{
      string nome; // visibilidade padrão private
public:
    void setNome(string nome);
    string getNome();
};
...
int main (){
    Pessoa umaPessoa;
    umaPessoa.setNome("Joao");
    Pessoa outraPessoa;
    outraPessoa.setNome("Maria");
    cout << umaPessoa.getNome() << endl;
    cout << outraPessoa.getNome();
    return 1;
}
```



Construtor e Destrutor

Construtor: basicamente método de inicialização de uma classe, invocado no momento da criação de objetos. Permite inicializar campos internos da classe e alocar recursos que um objeto da classe possa necessitar. Pode-se definir mais de um construtor (sobrecarga de método). Construtor pode ser usado para suportar a inicialização de valores internos da classe durante a declaração de objetos - os seus argumentos são os valores que deverão ser inicializados para os dados do objeto.

```
Pessoa();
```

Default: retorna um objeto Pessoa

Construtor e Destrutor

Destrutor: realiza a função inversa do construtor, é um método invocado automaticamente quando o objeto está para "morrer". Caso um objeto tenha recursos alocados, destrutores devem liberar tais recursos.
Por exemplo, se o construtor de uma classe alocou um atributo dinamicamente com new, o destrutor correspondente deve liberar este espaço com o operador delete.

```
~Pessoa();
```

Default: "destroi" o objeto

Instanciando Objetos


```
#include <iostream>
using namespace std;
class Ponto {
    int x, y;
public:
    Ponto(){ x = y = 0; } // Construtor
    ~Ponto() { } // Destrutor, neste caso espec. não faz nada
    void setXY(int n){ x = y = n; }
    int getX(){ return x; }
    int getY(){ return y; }
};
int main (){
    Ponto p1; // chama o construtor default imediatamente
    Ponto p2(); // chama o construtor imediatamente
    p2.setXY(5);
    cout << p1.getX() << " " << p2.getX();
    return 1;
}
```

Adicionando Comportamento

- Algumas informações podem ser calculadas, não precisando ser representadas por atributos reais
- Revisando a classe Pessoa
  - Idade = atributo virtual (calculado)
- É mais útil ter a **data de nascimento** da pessoa, pois a partir dela calcula-se a idade e pode-se verificar outras situações
  - **Criar a classe Data !!!!**

Classe Data - Deitel
Arquivo Date.h


```
#ifndef DATE_H_INCLUDED
#define DATE_H_INCLUDED
class Date {
        int month, day, year;
public:
        Date();                         //Construtores
        Date(int, int, int);
        ~Date(){}                       //Destrutor
        void setDay(int);               //métodos
        void setMonth(int);
        void setYear(int);
        void setToday();
        int getDay();
        int getMonth();
        int getYear();
        void showDate();
        };
#endif // DATE_H_INCLUDED
```

Classe Data
Arquivo Date.cpp


```
#include <iostream>
#include "Date.h"
using namespace std;

Date::Date(){  // construtor sem parametros
        month = 0, day = 0, year = 0;
}
Date::Date(int month, int day, int year) { // construtor com parametros
        this->month = month;
        this->day = day;
        this->year = year;
}
void Date::setDay(int d) {
        if (d < 1 && d > 31) cout << "The day is invalid" << endl;
        else day = d;
}
```

Classe Data
Arquivo Date.cpp


```
void Date::setMonth(int m) {
        if (m < 1 && m > 12) cout << "The month is invalid" << endl;
        else month = m;
}
void Date::setYear(int y) {
        if (y < 1950 && y > 2020) cout << "The year is invalid" << endl;
        else year = y;
}
int Date::getDay() {
    return day;
}
int Date::getMonth() {
    return month;
}
int Date::getYear() {
    return year;
}
```

Classe Data
Arquivo Date.cpp


```
void Date::setToday() {
    string monthName[] = { "January", "February", "March", "April",
        "May", "June", "July","August", "September", "October",
        "November", "December" };
    time_t     rawtime;
    struct tm* timeinfo;
    time( &rawtime );
    timeinfo = localtime( &rawtime );
    this->day = timeinfo->tm_mday;
    this->month = timeinfo->tm_mon+1;
    this->year = (timeinfo->tm_year + 1900);
    cout <<"Today's date is " << day <<" "<< month <<" "<< year <<endl;
}
void Date::showDate() {
    string monthName[] = { "January", "February", "March", "April",
        "May", "June", "July","August", "September", "October",
        "November", "December" };
        cout << monthName[month-1] <<"  "<< day <<"  "<< year << endl;
}
```

Revisando a modelagem


```
#ifndef PESSOA_H_INCLUDED
#define PESSOA_H_INCLUDED

#include <iostream>
#include <string>
#include "Date.h"
using namespace std;

class Pessoa{
    string nome;
    Date dataNascimento;
  public:
    void setNome(string nome);
    string getNome();
    void setDataNascimento(Date data);
    Date getDataNascimento();
    int getIdade();
};
#endif // PESSOA_H_INCLUDED
```

Inserindo o atributo dataNascimento e o atributo virtual idade

Arquivo Pessoa.h

Revisando a modelagem


```
#include <iostream>
#include "Date.h"
#include "Pessoa.h"
using namespace std;

void Pessoa::setNome(string nome){
  this->nome = nome;
}
string Pessoa::getNome(){
  return this->nome;
}
void Pessoa::setDataNascimento(Date data){
    dataNascimento = data;
}
Date Pessoa::getDataNascimento(){
    return dataNascimento;
}
```

Arquivo Pessoa.cpp

Revisando a modelagem


```
#include <iostream>
#include "Date.h"
#include "Pessoa.h"

...
int Pessoa::getIdade(){
    Date hoje;
    hoje.setToday();
    int idade = hoje.getYear() - dataNascimento.getYear();
    if(hoje.getDay() < dataNascimento.getDay())
        idade--;
    return idade;
}
```

Arquivo Pessoa.cpp

Método para calcular o atributo virtual idade