FATEC – ADS – PROGRAMAÇÃO ORIENTADA A OBJETOS
Introdução

Prof. Cristóvão Cunha

Objetivos de aprendizagem

- Conhecer a história do paradigma de programação orientada a objetos.
- Aprender o conceito de Classe e Objeto.
- Implementar uma Classe e Criar objetos a partir dela.

Para início de conversa

A orientação a objetos é uma maneira (paradigma) de se pensar e desenvolver programas. Ao desenvolver um programa orientado a objeto, você deve pensar em como a situação que o programa está automatizando funciona no mundo real, ou seja, quais os objetos envolvidos na situação e como eles se relacionam.

A partir disso você deve se preocupar em modelar e implementar o programa de maneira que ele espelhe a situação existente no mundo real, ou seja, deve criar e manipular as representações de objetos existentes no mundo real.

Você aprenderá um pouco da história, dos objetivos da orientação a objetos e aprenderá dois conceitos mais importantes da orientação a objetos: Classe e Objeto.

Paradigma de Programação Orientada a Objeto

Paradigma é um conjunto de regras que estabelecem fronteiras e descrevem como resolver os problemas dentro dessas fronteiras. Os paradigmas influenciam nossa percepção; ajudam-nos a organizar e coordenar a maneira como olhamos para o mundo. (Definição literal).

Entenda paradigma de programação como um estilo, uma maneira de se desenvolver um programa. Existem outros paradigmas de programação como veremos adiante.

Um pouco de história

O conceito de programação orientada a objeto não é algo novo. No final da década de 60, a linguagem Simula67, desenvolvida na Noruega, introduzia conceitos hoje encontrados nas linguagens orientadas a objetos. Em meados de 1970, o Centro de Pesquisa da Xerox (PARC) desenvolveu a linguagem Smalltalk, a primeira totalmente orientada a objetos. No início da década de 80, a AT&T lançou a Linguagem C++, uma evolução da linguagem de programação C em direção à orientação a objetos.

Atualmente, um grande número de linguagens incorpora características de orientação a objeto, tais como Java, Object Pascal, Python , etc.

Orientação a objeto – Conceitos básicos

O paradigma de programação orientada a objeto é baseado em alguns conceitos que definem uma forma de criar programas para computadores.

A filosofia principal desse paradigma de programação é solucionar um problema (via programação) através da representação computacional dos objetos existentes nesse problema (objetos do mundo real), usando para isso os conceitos mencionados acima. Com isso, essa maneira de se desenvolver programas fica mais próxima das situações, dos problemas como eles acontecem no mundo real.

Será um pouco difícil entender todas as vantagens de OO (Orientação a Objetos). Agora, portanto vamos mencionar apenas duas:

- É mais fácil conversar com o cliente que pediu o software se falarmos com objetos que existem no mundo dele (o mundo do software fica mais perto do mundo real).
- O software feito com OO pode ser feito com maior qualidade e pode ser mais fácil de escrever e, principalmente, alterar no futuro.

Já que a programação orientada a objeto se baseia no conceito de objetos vamos entender um pouco mais sobre objetos?

O nosso mundo está repleto de objetos, sejam eles concretos ou abstratos. Qualquer tipo de objeto possui **atributos** (características) e **comportamentos** que são inerentes a esses objetos.

Exemplos de objetos **CONCRETOS**

1) CANETA

Atributos de qualquer Caneta

- altura
- espessura
- cor

Nesse caso, o nosso objeto **Caneta** tem os seguintes dados em cada característica:

- altura: 10 cm
- espessura: 2 cm
- cor: rosa

Comportamentos de qualquer Caneta

- desenhar
- etc.

2) PESSOA

Atributos de qualquer Pessoa

- altura
- peso
- idade

Nesse caso, o nosso objeto **Pessoa** tem os seguintes dados em cada característica:

- altura: 1,80
- peso: 70
- idade: 25

Comportamentos de qualquer Pessoa

- andar
- sorrir
- ler
- etc.

Exemplos de objetos que **NÃO SÃO CONCRETOS**:

1) RETÂNGULO
Atributos de qualquer Retângulo
- base
- altura

Nesse caso, o nosso objeto Retângulo tem os seguintes dados em cada característica:
- base: 10
- altura: 5

Comportamentos de qualquer Retângulo
- calcular área
- calcular perímetro
- etc.

2) DISCIPLINA
Atributos de qualquer Disciplina
- código da disciplina
- nome da disciplina
- carga horária

Nesse caso, o nosso objeto Disciplina tem os seguintes dados em cada característica:
- código da disciplina: 1
- nome da disciplina: Linguagem de Programação
- carga horária: 40

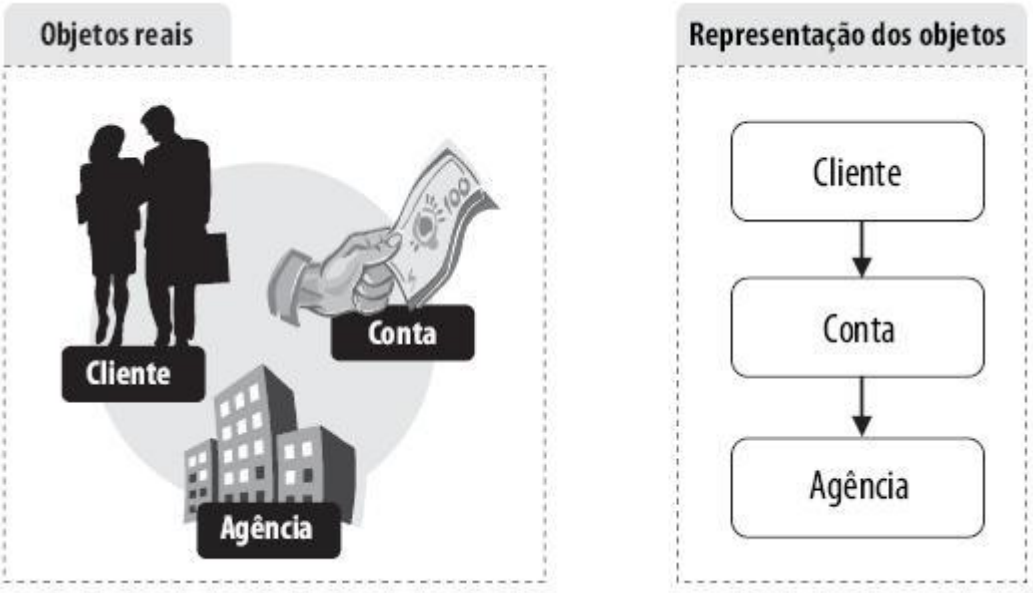
Comportamentos de qualquer Disciplina
- listar o nome da disciplina
- etc.

Para usar objetos na programação, primeiro precisamos distinguir entre um **objeto real** e a **representação de um objeto**.

No desenvolvimento de programas, sempre trabalhamos com representações de objetos. Essa representação precisa refletir os objetos reais, ou seja, objetos existentes no problema do mundo real que queremos solucionar.

Por exemplo, um sistema de conta corrente não manipula diretamente os clientes, contas e cheques (esses são os objetos reais do problema).

Em vez disso, o software deve criar representações desses objetos, com os mesmos atributos e comportamentos dos objetos do mundo real. Essa representação é chamada de **ABSTRAÇÃO**.



Vamos aprender mais sobre representação de objetos?

Para entender isso, vamos voltar ao exemplo do sistema de conta de corrente.

No mundo real existem vários objetos Cliente, vários objetos Conta e vários objetos Agência (pode ser somente um), ou seja, muito provavelmente não existirá somente um cliente, uma conta e uma agência.

Todos os objetos cliente possuem o mesmo conjunto de atributos, por exemplo, todos os objetos cliente possuem nome e endereço, assim como todos os objetos conta possuem um número, um saldo, um histórico de transações e todos os objetos agência possui um número e um nome.

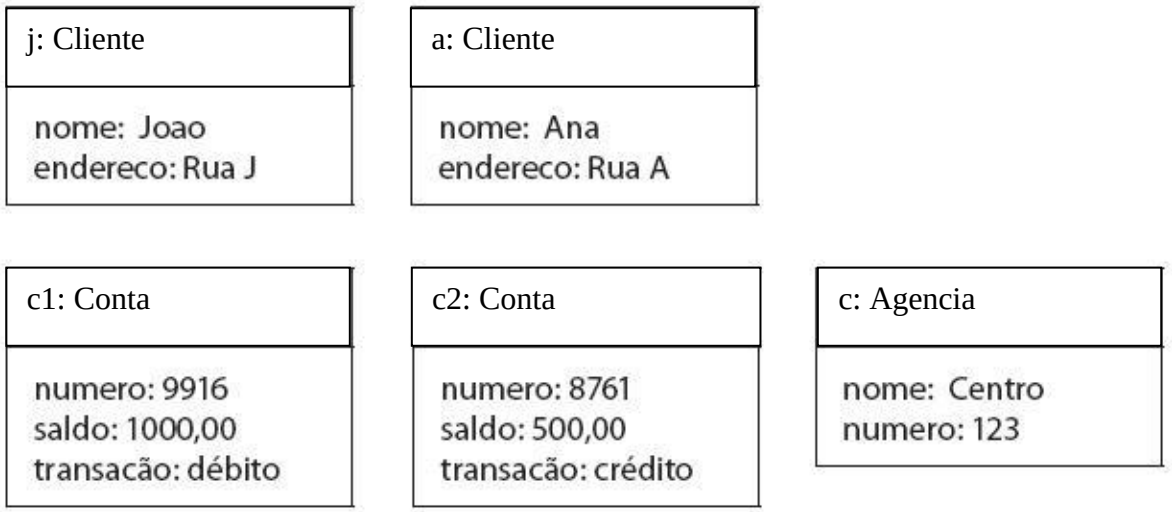
Assim como os atributos, todos os objetos de um mesmo tipo compartilham do mesmo conjunto de **comportamentos**, por exemplo, todos os objetos Cliente fazem depósitos e saques.

Apesar de cada objeto ter o mesmo conjunto de atributos, cada objeto possui valores próprios para cada atributo, ou seja, cada objeto é único.

Segundo o dicionário Aurélio **abstração** significa: “Ato de separar mentalmente um ou mais elementos de uma totalidade complexa (coisa, representação, fato), os quais só mentalmente podem subsistir fora dessa totalidade”.

Essa explicação para objetos do mundo real vale para quando trabalhamos com a representação desses objetos no contexto de desenvolvimento desse sistema (programação).

Portanto, podemos perceber que o sistema de conta corrente (programa de conta corrente) será composto de vários objetos, cada qual com um conjunto de atributos e comportamento em comum (se forem do mesmo tipo) e com valores próprios nos seus atributos. A figura abaixo ilustra os vários objetos cliente, conta e agência.



O termo **instância** é frequentemente usado para descrever um objeto com valores próprios em seus atributos. Por exemplo, observe a figura acima:

- **"a"** é uma instância de **Cliente**
- **"c1"** é uma instância de **Conta**

Todo objeto deve ter:

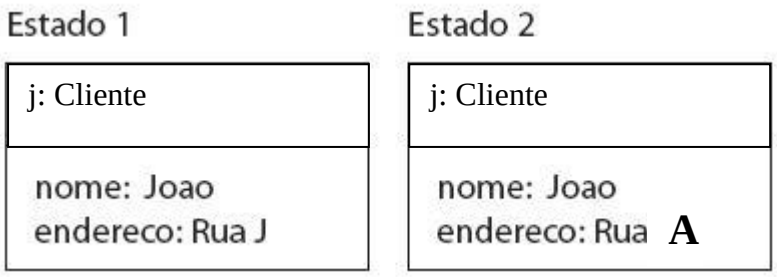
1. **Estado**
2. **Comportamento**
3. **Identidade**

1) Estado

É representado pelos valores dos atributos de um objeto em um determinado instante do tempo. O estado do objeto usualmente muda ao longo da existência do objeto.

Exemplo

O valor dos atributos nome e endereço do objeto João podem mudar ao longo da existência desse objeto. As figuras abaixo ilustram isso: no estado 1 do objeto João, o valor do atributo endereço era Rua J, no estado 2 o valor desse atributo é Rua XX.



2) Comportamento

Determina como um objeto age e reage: suas modificações de estado e interações com outros objetos.

O comportamento é determinado pelo *conjunto de operações* ou *métodos* que o objeto pode realizar. Operações ou métodos são algo que você pede para o objeto fazer, como fazer um depósito (comportamento de qualquer objeto do tipo Conta).

Você deve estar achando estranho: fazer um depósito é um comportamento de qualquer objeto do tipo Conta? Isso mesmo!

Fazer depósito é um comportamento de qualquer objeto do tipo Conta e não de qualquer objeto do tipo Cliente, como você deve ter pensado a princípio.

Isso acontece porque há diferenças grandes quando pensamos no comportamento de representações de objetos, ou seja, objetos do mundo do software.

Ao identificar comportamentos de objetos do mundo do software devemos levar em conta os seguintes princípios:

- **Primeiro princípio**: não vamos modelar todo o comportamento dos objetos. Exemplo: Clientes tomam café no mundo real, mas provavelmente não no software.
- **Segundo princípio**: os comportamentos freqüentemente são assumidos por objetos diferentes. Exemplo: No mundo real, quem faz um depósito? Um Cliente.

No software, quais objetos assumiriam a responsabilidade de fazer um depósito? Objetos do tipo Conta.

Por que isso?

A primeira grande regra de programação Orientada a Objeto é a seguinte:

"Comportamentos são associados aos objetos que possuem os atributos afetados pelo comportamento".

Um depósito afeta duas coisas, do ponto de vista de um sistema bancário:

- O saldo de uma conta.
- O histórico de transações feitas a uma conta.

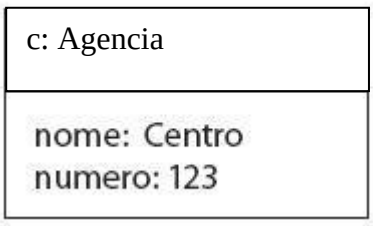
Como se pode ver, o depósito afeta apenas atributos de uma Conta. Portanto, esse comportamento está associado à Conta e não ao Cliente, embora, no mundo físico, seja o Cliente que faz o depósito.

Outros comportamentos associados aos objetos do tipo Conta: fazer um saque, informar seu saldo, etc.

3) Identidade

Refere-se à identificação do objeto. Como cada objeto é único, ou seja, têm seus próprios valores nos atributos, ele deve ser identificado por algum nome. Mesmo que seu estado seja idêntico ao de outro objeto ele tem identificação própria.

A identificação do objeto do tipo Agência mostrado na figura abaixo é **"c"**. Esse objeto poderia ter qualquer nome identificando-o.



Agora você já sabe que qualquer programa orientado a objeto deve trabalhar com representações de objetos que reflitam as características e comportamento de objetos do mundo real e já aprendeu os principais conceitos relacionados a objetos.

Falta agora aprender como CRIAR a representação de um objeto dentro de um programa.

No sistema de conta corrente foram identificados, a princípio, 3 tipos ou grupos de objetos: *Cliente*, *Agência* e *Conta*.

Sabemos que poderão existir vários objetos de cada um desses tipos e todos eles irão possuir o mesmo conjunto de atributos e comportamentos do seu tipo.

Para criar uma representação de cada objeto dentro do programa, devemos antes criar uma estrutura, uma espécie de **molde** ou ***template*** que represente os atributos e comportamentos do tipo do objeto. Essa estrutura é chamada de **classe.**

Portanto, devemos ter uma classe Cliente, uma classe Agência e uma classe Conta.

É a partir dessa estrutura conhecida como classe que iremos criar as representações de objetos que precisamos para desenvolver o programa, ou seja, precisamos criar a estrutura (classe) apenas uma vez e, a partir dela, serão criadas as representações dos objetos.

Atenção!

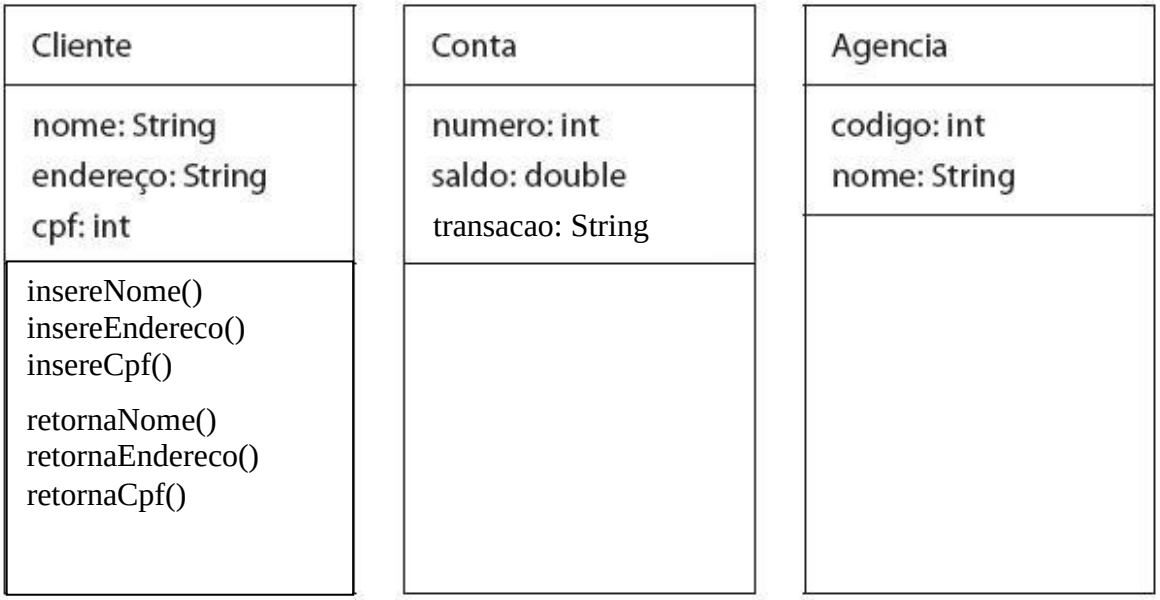
Os conceitos de classe e objeto são os mais básicos e importantes da orientação a objetos. O correto entendimento desses conceitos é imprescindível para o entendimento dos demais conceitos da OO.

Vamos aprender mais conceitos sobre Classes e Objetos?

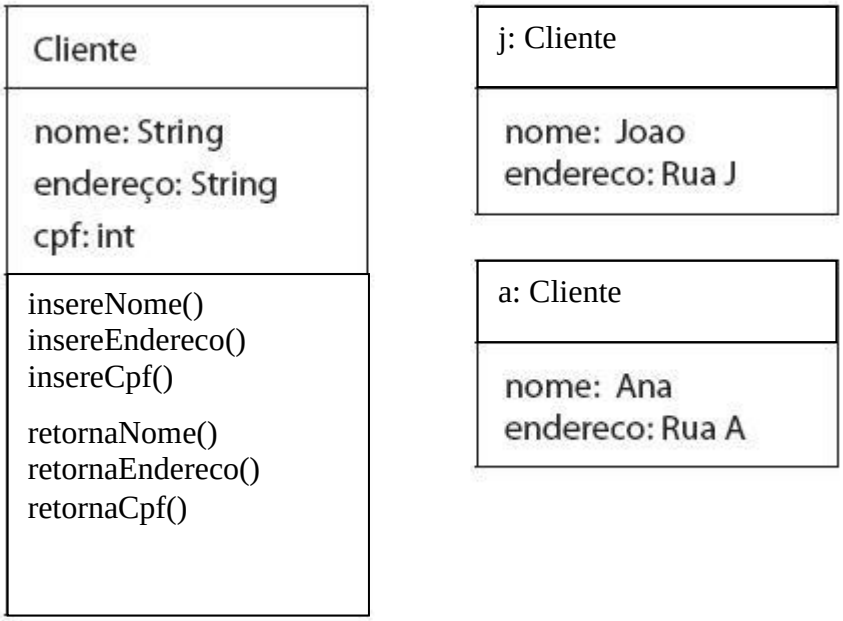
Uma **classe** é a descrição de atributos e comportamentos de um grupo de objetos com propriedades similares (atributos) e comportamento comum (operações ou métodos).

Por representar atributos e comportamento de um determinado tipo de objeto que existe no mundo real, uma classe é uma abstração da realidade, pois nela estão especificados somente parte dos atributos e comportamentos de um objeto do mundo real. Somente parte, porque não são todos os atributos e comportamentos dos objetos do mundo real que interessam ao sistema.

A figura a seguir, ilustra a representação gráfica das classes do sistema de conta corrente. Na primeira parte da figura está o nome da classe, na parte do meio, estão os atributos e, na parte final, está o comportamento (métodos ou operações). Observe que a classe Conta e Agencia ainda não estão com os comportamentos definidos. Faremos isso na medida em que avançarmos nos conceitos de OO.



Um **objeto** é uma instância de uma classe. Essa instância possui valores próprios em cada atributo definido na classe. As figuras a seguir ilustram a classe Cliente e os vários objetos (instâncias) da classe Cliente.



A notação que estamos utilizando para representar classes e objetos é a utilizada pela **UML** (*Unified Modelling Language*) ou Linguagem de Modelagem Unificada. A UML não é uma linguagem de programação e sim, uma linguagem composta de vários diagramas (cada qual com seu conjunto de símbolos gráficos) que expressam como um sistema orientado a objetos deve funcionar. Ela é utilizada nas fases iniciais do desenvolvimento de um software, ou seja, antes da programação propriamente dita, com o objetivo de especificar e documentar o funcionamento do sistema. O diagrama que iremos utilizar é o **Diagrama de Classes** que especifica as classes do sistema e o relacionamento entre elas.

Agora que você já estudou a diferença entre CLASSE e OBJETO, vamos partir para a parte prática, ou seja, você vai aprender a implementar (programar) uma classe e a criar objetos a partir dela. Você continuará aprendendo outros conceitos nas próximas aulas, porém, para que os conceitos não fiquem muito abstratos para você, vamos ver como isso funciona na prática de maneira paralela.

Para finalizar essa seção vamos fazer uma pequena revisão:

Um programa orientado a objeto é composto por vários objetos do mesmo tipo e de outros tipos que se relacionam. Chamamos esses objetos do mundo do software de representações de objetos.

Como num programa muito provavelmente existirão vários objetos do mesmo tipo, devemos criar uma estrutura ou *template* que represente os atributos e comportamentos desses objetos do mesmo tipo, essa estrutura é chamada de classe. Portanto classe é uma estrutura a partir da qual os objetos (do software) são criados.

Importante!

O entendimento dos conceitos de orientação a objetos que já vimos e, dos demais que estão por vir, são de vital importância para o desenvolvimento de programas orientado a objetos e sua utilização ***independe*** *da linguagem utilizada para implementação dos programas. Quer dizer que, para desenvolver qualquer programa OO, é preciso aplicar esses conceitos na implementação do programa, utilizando para isso, qualquer linguagem de programação orientada a objetos.*

Implementando uma Classe e Criando Objetos a partir dela

A implementação de uma classe só tem sentido se, no contexto do desenvolvimento de um sistema orientado a objeto, for identificada a necessidade de existência de um conjunto de objetos com atributos e comportamentos em comum. A partir do momento que esses objetos com atributos e comportamento em comum são identificados, já existe a necessidade de criar a estrutura (classe) que os defina.

Você irá aprender como implementar na linguagem Java uma classe e como criar objetos (instâncias) a partir dela. Pode-se falar também no termo “instanciar objetos” a partir de uma classe.

Não iremos neste momento demonstrar o processo de identificação de objetos num dado problema, isso será feito nas próximas aulas.

A visão prática desses dois conceitos (Classe e Objeto) facilitará o entendimento dos demais conceitos.

Implementação da Classe Cliente

A seguir será apresentada uma implementação resumida e mais **didática** da classe Cliente.

Didática porque tem como objetivo facilitar o entendimento nesse primeiro momento.

```
Linha 1 public class Cliente{
Linha 2    private String nome, endereco;
Linha 3    private int cpf;
Linha 4
Linha 5    public Cliente( ){
Linha 6       nome = "";
Linha 7       endereco = "";
Linha 8       cpf = 0;
Linha 9    }
Linha 10   public void insereNome(String sNome){
Linha 11      nome = sNome;
Linha 12   }
Linha 13
Linha 14   public void insereEndereco(String sEndereco){
Linha 15      endereco = sEndereco;
Linha 16   }
Linha 17
Linha 18   public void insereCpf(int iCpf){
Linha 19      cpf = iCpf;
Linha 20   }
Linha 21   public String retornaNome(){
Linha 22      return nome;
Linha 23   }
Linha 24
Linha 25   public String retornaEndereco(){
Linha 26      return endereco;
Linha 27   }
Linha 28
Linha 29   public int retornaCpf(){
Linha 30      return cpf;
Linha 31   }
Linha 32}
```

Vamos analisar a implementação da classe Cliente linha a linha.

Relembrando: para desenvolver qualquer programa você deve editar o programa num editor de texto, salvá-lo num local conhecido e, logo após, compilá-lo. O mesmo processo deve ser feito com as classes que você desenvolver daqui para frente. Detalhe: classes que representam atributos e comportamento de objetos NÃO SÃO executadas.

Linha 1: definição inicial da classe. Começa com a palavra **public** (veremos mais adiante qual o significado dessa palavra) seguida da palavra **class** e seguida do **nome da classe**.

Regras para Nomeação de Classes

Para se nomear classes, existem alguns requisitos que devem ser observados:

- Classes são nomeadas com um substantivo no singular;
- O nome de uma classe deve iniciar com a **primeira letra maiúscula;**
- Não devem ser utilizados símbolos de sublinhado ("_") - nomes compostos por múltiplas palavras são organizados com todas as palavras juntas, onde a primeira letra de cada uma fica em maiúscula.

  Exemplos: Aluno, Professor, FolhaPagamento.

Linhas 2 e 3: definição dos atributos definidos para a classe. Na definição dos atributos deve-se utilizar a palavra **private** (veremos mais adiante o significado dessa palavra) seguido do tipo de dado de cada atributo e do nome do atributo. No caso da classe Cliente, os dois atributos nome e endereço devem **armazenar** dados do tipo String (literal) e o atributo cpf deve armazenar valores do tipo int (numéricos inteiros).

Regras para Definição de Atributos

- Normalmente, atributos são colocados no início da definição da classe, depois do primeiro { . Também pode ser colocados bem no final, antes do último }.
- Devem começar com letra minúscula.

Linha 5: depois da definição do nome da classe e dos atributos da classe, segue-se a definição dos comportamentos, no contexto da implementação, conhecidos como **métodos**. Por enquanto, os comportamentos ou métodos definidos para a classe Cliente são simples. Na medida em que formos avançando, mais comportamentos ou métodos poderão ser implementados. No momento, implementamos quatro métodos: Cliente( ), insereNome( ), insereEndereco( ), retornaNome( ), retornaEndereco( );

O primeiro método implementado se chama Cliente( ) e se enquadra na categoria dos **métodos construtores**. Um método construtor SEMPRE deve ter o mesmo nome da classe e não deve

especificar no cabeçalho nenhum tipo de retorno, nem mesmo deve ser utilizada a palavra void (o que acontece nos demais métodos que não retornam nada – métodos do tipo procedimento). Todo o método construtor só é executado no momento em que um objeto dessa classe é criado (instanciado). Vamos saber mais sobre métodos construtores no momento da explicação da criação de objetos. Saiba por enquanto que é importante (não obrigatório) que cada classe tenha ao menos um método construtor implementado.

Linhas 6 a 8: linhas de código do método construtor. O que se pode programar dentro de um método construtor?

Qualquer coisa que se quer que aconteça quando um objeto do tipo Cliente, no caso, for criado, porque o código que está dentro de um método construtor é executado sempre que um objeto desse tipo for criado.

Porém, normalmente se programa para que os **atributos do objeto** sejam inicializados. No caso do nosso método construtor, os atributos nome, endereço foram inicializados com Strings vazias (“”) e o atributo cpf com o valor 0.

Linha 10: nessa linha está o comportamento ou método chamado insereNome( ). Esse tipo de comportamento é típico de qualquer objeto do tipo Cliente, pois todo o cliente pode ter seu nome alterado (ou inserido). No caso de um sistema OO, quem executa esse comportamento é o próprio objeto Cliente.

A implementação desse método ou comportamento é feita através de um procedimento (veja que o tipo de retorno é void, ou seja, essa subrotina não retorna nenhum valor) que recebe como **parâmetro** o novo nome do cliente. Esse valor é recebido em uma variável de memória do tipo String que chamamos de sNome. O nome da variável de memória que receberá o novo valor do atributo nome pode ser qualquer um, porém essa variável deve ser declarada como sendo do tipo String, já que o valor dela será atribuído ao atributo nome que também é do tipo String (próxima linha).

Linha 11: atribuição do novo nome do cliente ao atributo nome. O atributo nome recebe o conteúdo da variável sNome, que contém o novo nome do cliente.

Linha 12: fim do método insereNome( );

Linhas 14 a 16: implementação do método ou comportamento insereEndereco(). Esse método altera (ou insere) o endereço de qualquer objeto Cliente.

Linhas 18 a 20: implementação do método ou comportamento insereCpf(). Esse método altera (ou insere) o cpf de qualquer objeto Cliente.

Linha 21: nessa linha está o comportamento ou método chamado retornaNome( ). Esse tipo de comportamento é típico de qualquer objeto do tipo Cliente, pois todo o cliente pode ser capaz de fornecer (retornar) seu nome. No caso de um sistema OO, quem executa esse comportamento é o próprio objeto Cliente.

A implementação desse método ou comportamento é feita através de uma **função** (veja que o tipo de retorno é String) que retorna o nome do cliente, ou seja, o valor armazenado no atributo nome do objeto Cliente.

Linha 22: retorno (usar palavra return) do conteúdo do atributo nome.

Linhas 25 a 27: implementação do método ou comportamento chamado retornaEndereco() .

Linhas 29 a 31: implementação do método ou comportamento chamado retornaCpf( ).

Linha 32: final da implementação da classe.

Depois de implementar a classe, você deve compilar, mas não deve executar. Esse tipo de classe não é executável, ela não possui **método main ( )**. Essa classe só possui a finalidade de ser uma estrutura ou *template (modelo ou "receita de bolo")* para a criação de objetos desse tipo.

Criar objetos a partir da classe Cliente

Depois que a classe Cliente for implementada e compilada, ela está pronta para ser "usada", ou seja, é possível criar objetos do tipo Cliente a partir dela.

Abaixo está a implementação de uma classe que possui um método main( ), ou seja, é uma classe executável. Dentro desse método estamos criando um objeto do tipo Cliente e manipulando esse objeto.

Mas o que é de fato "criar um objeto"? Lembre-se sempre: agora estamos falando de objetos no contexto de um software. Você entenderá isso nas linhas seguintes.

```
Linha 1 public class UsaCliente{
Linha 2  public static void main(String args[]) {
Linha 3       Cliente a = new Cliente();
Linha 4       a.insereNome("Ana");
Linha 5       a.insereEndereco("Rua A");
Linha 6       a.insereCpf(12352241123);
Linha 7       System.out.println(a.retornaNome());
Linha 8  }
Linha 9 }
```

Vamos analisar a implementação da classe UsaCliente linha a linha:

Relembrando: para desenvolver qualquer programa você deve editar o programa num editor de texto, salvá-lo num local conhecido e logo após compilá-lo. O mesmo processo deve ser feito com as classes

que você desenvolver daqui para frente. Detalhe: classes que representam atributos e comportamento de objetos NÃO SÃO executadas.

Linha 1: definição inicial da classe. Começa com a palavra **public** seguida da palavra **class** e seguida do **nome da classe**.

Linha 2: definição do método main( ). Essa definição é sempre a mesma utilizada nos programas anteriores. Todo o programa que iremos implementar estará dentro do método main( ).

Linha 3: essa linha cria um objeto do tipo Cliente, chamado "a".

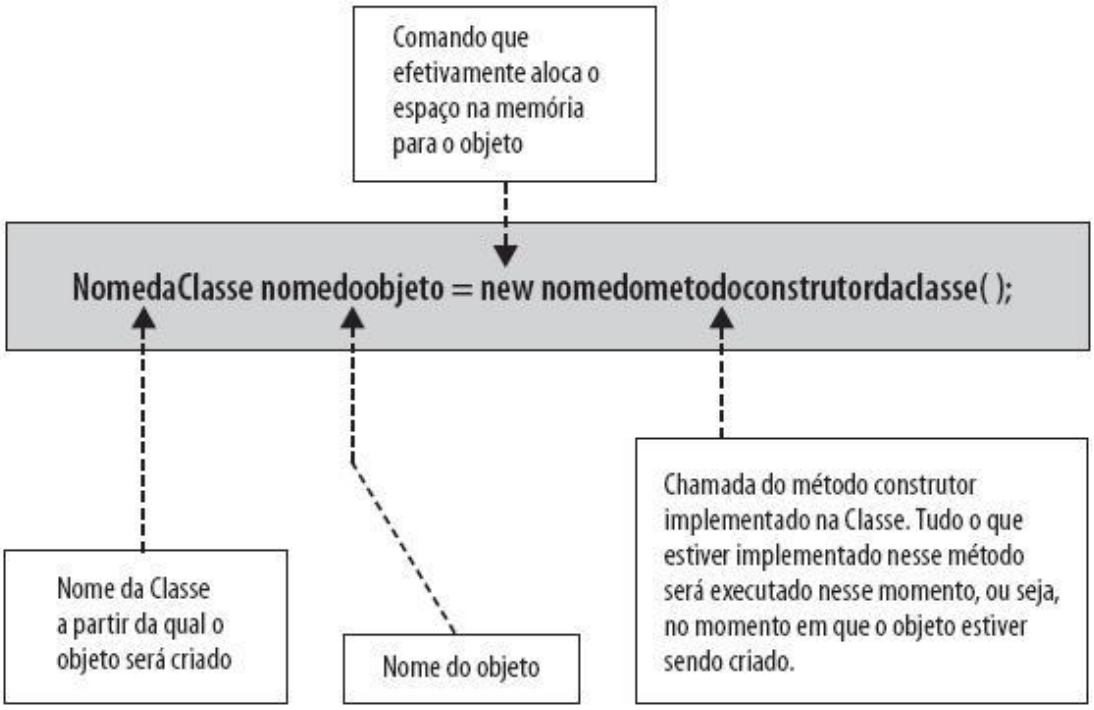
Criar um objeto é alocar (reservar) um espaço na memória do computador para armazenar valores para os atributos definidos na classe desse objeto.

A explicação de criar um objeto é similar à criação (declaração) de uma variável de memória. Quando declaramos uma variável de memória também estamos reservando espaço na memória para armazenar algum valor. Essa variável de memória deve ter um nome e deve ter um tipo de dados e nela só poderá ser armazenado valores desse tipo.

O mesmo acontece com a criação de um objeto. Ao criar um objeto, precisamos dar um nome (identificação) para ele e, nesse espaço de memória que estamos declarando, só poderão ser armazenados valores correspondentes aos atributos definidos no tipo (classe) desse objeto.

Essa explicação para criação de objeto é simples e será utilizada no momento para facilitar seu entendimento. Posteriormente, essa explicação será refinada.

Portanto, sempre que se quer criar um objeto deve-se usar a seguinte sintaxe:



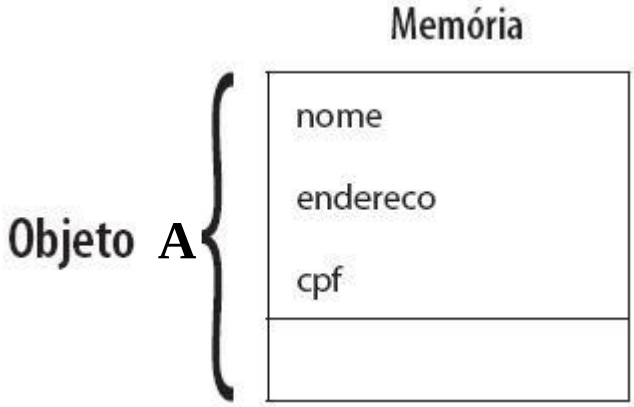
Após a execução dessa linha, um objeto, ou seja, uma área de memória é alocada.

Na instrução da linha 3, estamos criando um objeto chamado "a". Note cuja a sintaxe é igual a definida acima para criação de objetos. Sempre que um objeto é criado, o método construtor definido na classe desse objeto é chamado (invocado) e, tudo o que está programado nesse método, é executado.

O que foi programado no método construtor da classe Cliente?

A inicialização de todos os atributos do objeto Cliente, portanto ao criar o objeto cliente, todos os seus atributos já estarão inicializados.

A figura abaixo ilustra de maneira simples o objeto alocado na memória e seus atributos inicializados.



A partir desse momento, poderá ser chamado qualquer comportamento ou método definido na classe cliente. Isso será feito nas linhas seguintes.

Linha 4: depois que o objeto é criado, podemos invocar seus comportamentos ou métodos. Nessa linha estamos chamando o método insereNome("Ana") e passando como parâmetro o novo nome do cliente.

Observe que o método está **precedido** do nome do objeto e SEMPRE deve ser assim. Qualquer comportamento ou método deve ser chamado precedido do nome do objeto.

Quando se chama qualquer método definido em uma classe, o código implementando nesse método é executado. Nesse caso, o método irá alterar o nome do objeto "a".

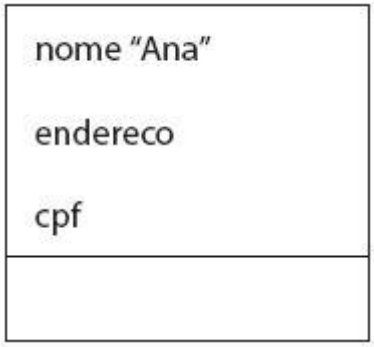
Qual o valor do atributo nome do objeto "a" ANTES da chamada desse método?

"" (String vazia) porque quando o objeto foi criado, esse valor foi armazenado lá. Isso aconteceu no momento que o método construtor foi chamado.

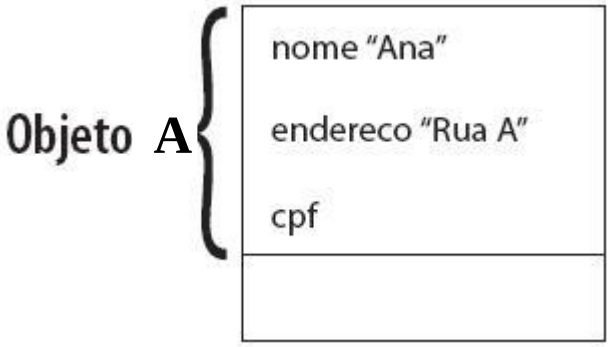
Qual o valor do atributo nome do objeto "a" DEPOIS da chamada desse método?

"Ana" porque foi esse o valor passado para o método. Lembre-se que o método insereNome(String sNome) recebe um valor como parâmetro e, esse valor, é armazenado no atributo nome.

Logo, após a execução dessa linha o objeto está com o seguinte estado:

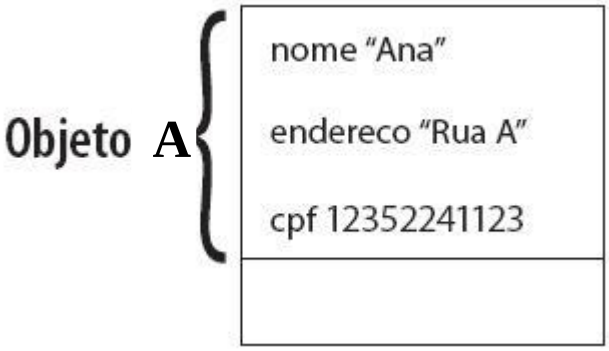


Linha 5: chamada do método insereEndereco( ) que irá inserir (alterar) o endereço do objeto "a". Logo após a execução dessa linha, o objeto está com o seguinte estado:



Linha 6: chamada do método insereCpf( ) que irá inserir (alterar) o CPF do objeto "a".

Logo, após a execução dessa linha, o objeto está com o seguinte estado:



Linha 7: impressão do nome do cliente. Para isso chamamos o método retornaNome( ) do objeto "a". Como esse método é uma função e toda a função retorna valor, ele não pode ser chamado isolado numa linha de instrução. Deve ser feito alguma coisa com o valor retornado do método, ou imprimir ou armazenar em outra variável de memória.

Linha 8: fim do método main().

Linha 9: fim da classe UsaCliente.

Observação: Essa classe utiliza apenas 1 objeto, mas normalmente trabalharemos com programas que envolvem a criação de vários objetos. O procedimento explicado para criação e utilização de 1 objeto é o mesmo para vários objetos.

Boa Diversão!
Ótimos Estudos!