Linguagem de Programação II

Conceitos de programação orientada a objetos (POO)

Universidade do Estado do Rio de Janeiro-UERJ
Instituto de Matemática e Estatística-IME
Ciência da Computação
Professor: Alexandre Sztajnberg

Classes e objetos

- Em linguagens orientadas a objetos
  - classes são projetos, "receitas", de um objeto
  - possuem suas características (atributos) e comportamentos (métodos)
- Características básicas de uma classe:
  - Possui um nome;
  - Visibilidade: public, private, protected Também vale para os atributos e métodos;
  - Atributos e métodos para definir suas características e comportamentos;
- Construção de uma classe

```
[Modificadores] class NomeDaClasse { [bloco] }
```

- Representar objetos do mundo real de maneira abstrata e geral
  - Uma classe **gato.** O que todo gato tem?
    - 4 patas, rabo, bigode, pelos, dentes afiados.
    - Nome, idade, pelagem
    - Movimentos, etc.
    - Todos possuem a mesma "estrutura" geral, mas cada gato é único.
  - Um objeto **meuGato**
    - Tem nome? *Tigre*
    - Idade? *3 anos,* Pelagem? *malhado*

Classes e objetos

```
public class Gato{
 public String nome;
 public String cor;
 public int idade;

 public void miar(){
  //Código do método miar()
 }
 public void comer(){
  //Código do método comer()
 }
 public void andar(){
  //Código do método andar()
 }
...
}
```

Nome da classe

Gato.java

Classes e objetos

```
public class Gato{
 public String nome;
 public String cor;
 public int idade;

 public void miar(){
  //Código do método miar()
 }
 public void comer(){
  //Código do método comer()
 }
 public void andar(){
  //Código do método andar()
 }
...
}
```

Nome da classe

Atributos da classe

Gato.java

Classes e objetos

```
public class Gato{
 public String nome;
 public String cor;
 public int idade;

 public void miar(){
  //Código do método miar()
 }
 public void comer(){
  //Código do método comer()
 }
 public void andar(){
  //Código do método andar()
 }
...
}
```

Nome da classe

Atributos da classe

Métodos da classe

Classes e objetos

- Objetos são as concretizações dos projetos, "receitas" especificadas pelas classes. É necessário instanciar esse objetos para inicializar os atributos e invocar os métodos. São como variáveis em *C*
- Atributos são as propriedades de um objeto, também são conhecidos como campos. Eles definem o estado de um objeto, podendo ser alterados conforme a execução do programa.
- Métodos são ações ou procedimentos, sendo possível a comunicação ou interação com outros objetos.
- Criação de uma instância de um objeto

```
NomeDaClasse NomeDaVariável = new NomeDaClasse([lista de argumentos])
```

Classes e objetos

```
public class TesteGato{

  public static void main(String[] args){

    Gato gato1 = new Gato();

    gato1.nome = "Faisca" ;
    gato1.cor = "Amarelo" ;
    gato1.idade = 5 ;

    System.out.println(gato1.nome);

    ...

  }

}
```

Criação da instância

Classes e objetos

```
public class TesteGato{

  public static void main(String[] args){

    Gato gato1 = new Gato();

    gato1.nome = "Faisca" ;
    gato1.cor = "Amarelo" ;
    gato1.idade = 5 ;

    System.out.println(gato1.nome);

    ...

  }

}
```

Criação da instância

Inicializando os atributos

TesteGato.java

Classes e objetos

```
public class TesteGato{

  public static void main(String[] args){

    Gato gato1 = new Gato();

    gato1.nome = "Faisca" ;
    gato1.cor = "Amarelo" ;
    gato1.idade = 5 ;

    System.out.println(gato1.nome);

    ...

  }

}
```

Criação da instância

Inicializando os atributos

Exibindo o nome

TesteGato.java

Classes e objetos

- As variáveis de tipos compostos
  - referências a um objeto criado na memória quando usamos a palavra reservada `new`.
  - São "ponteiros", contendo o endereço de memória do objeto.

- Importante: A criação de um novo objeto ocorre em três etapas
- Um *assign*, atribuição, NÃO, é uma operação matemática ou símbolo de igualdade!
  1. Criação da referência (variável), apontando para "`null`";
  2. Criação do objeto sem nome na memória;
  3. Após a criação, a referência passa a apontar para o endereço de memória onde foi criado o objeto;

- E como fica todo esse processo na JVM?

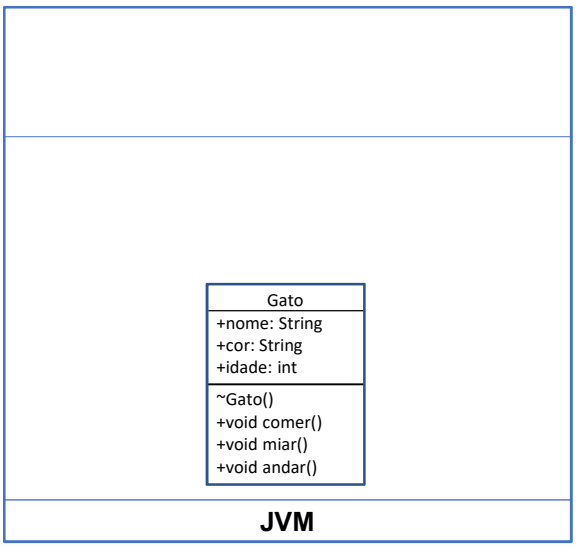
Classes e objetos

```
public class TesteObjetosPonteiros{

  public static void main(String[] args){

    Gato gato1 = new Gato();
    Gato gato2 = new Gato();

    ...
```

Criação de duas instâncias de Gato

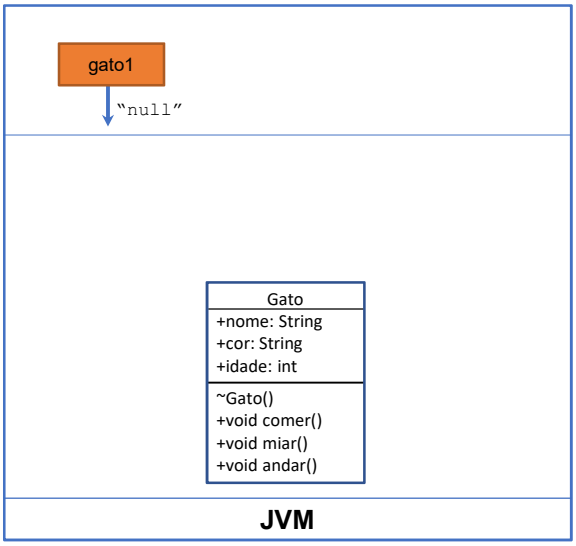
Classes e objetos

```
public class TesteObjetosPonteiros{

  public static void main(String[] args){

    Gato gato1 = new Gato();
    Gato gato2 = new Gato();
```



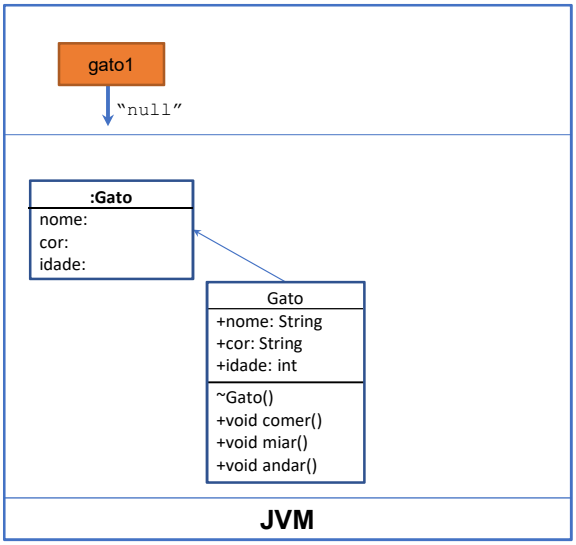
Classes e objetos

```
public class TesteObjetosPonteiros{

  public static void main(String[] args){

    Gato gato1 = new Gato();
    Gato gato2 = new Gato();
```



Classes e objetos

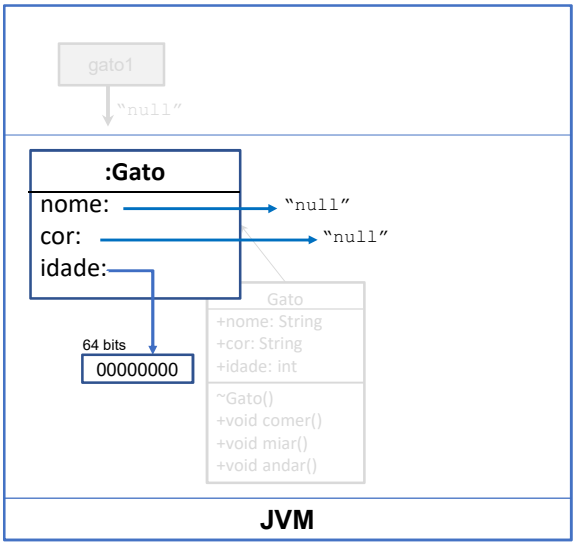
```
public class TesteObjetosPonteiros{

  public static void main(String[] args){

    Gato gato1 = new Gato();
    Gato gato2 = new Gato();
```



Classes e objetos

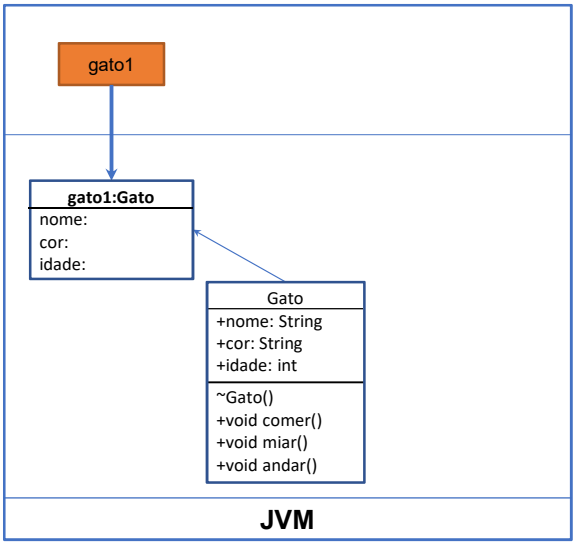
```
public class TesteObjetosPonteiros{

  public static void main(String[] args){

    Gato gato1 = new Gato();
    Gato gato2 = new Gato();
```

Atenção! Zoom no objeto *:Gato*



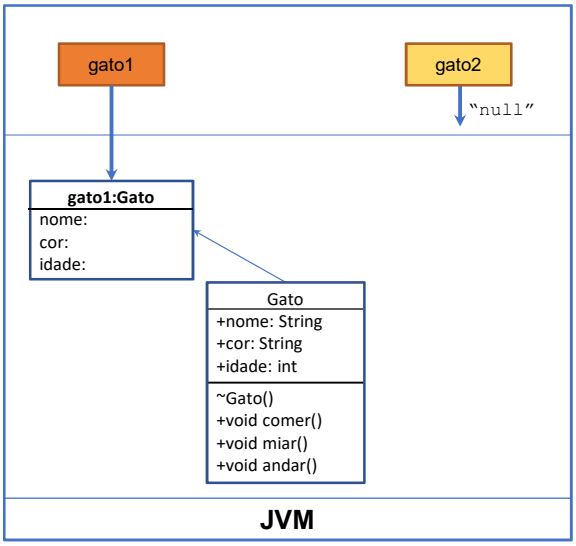
Classes e objetos

```
public class TesteObjetosPonteiros{

  public static void main(String[] args){

    Gato gato1 = new Gato();
    Gato gato2 = new Gato();
```



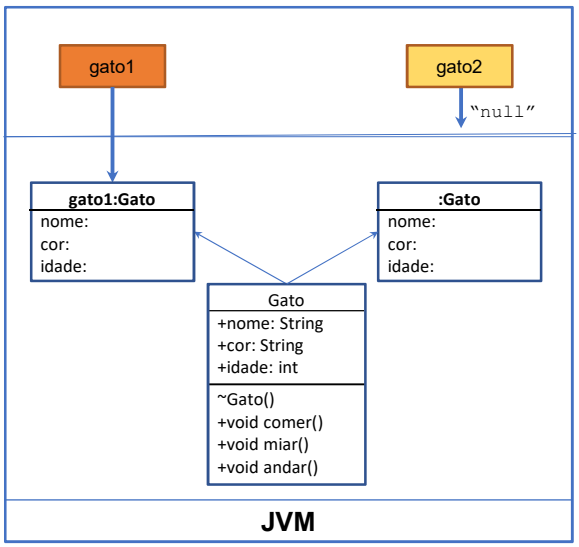
Classes e objetos

```
public class TesteObjetosPonteiros{

  public static void main(String[] args){

    Gato gato1 = new Gato();
    Gato gato2 = new Gato();
```



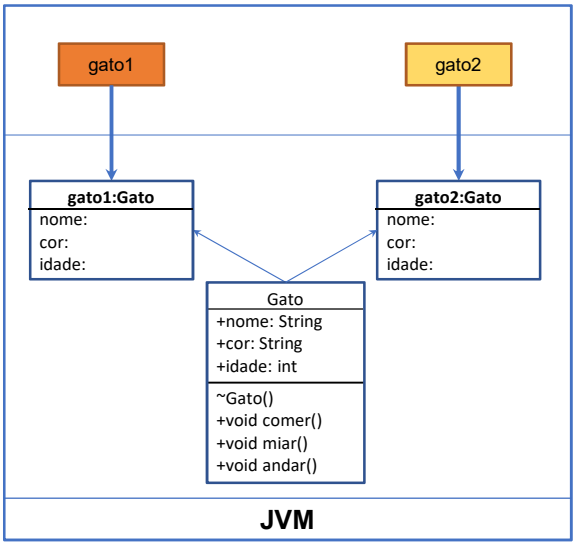
Classes e objetos

```
public class TesteObjetosPonteiros{

  public static void main(String[] args){

    Gato gato1 = new Gato();
    Gato gato2 = new Gato();
```



Classes e objetos

```
public class TesteObjetosPonteiros{

  public static void main(String[] args){

    Gato gato1 = new Gato();
    Gato gato2 = new Gato();
```



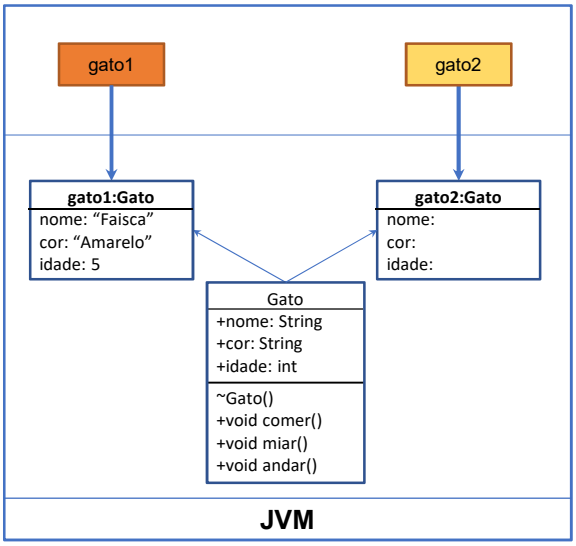
Classes e objetos

```
...
    gato1.nome = "Faisca" ;
    gato1.cor = "Amarelo" ;
    gato1.idade = 5 ;
    System.out.println(gato1.nome);
    System.out.println(gato1.cor);
    System.out.println(gato1.idade);
    System.out.println("\n");

    gato2.nome = "Chiclete" ;
    gato2.cor = "Preto" ;
    gato2.idade = 6 ;
    System.out.println(gato2.nome);
    System.out.println(gato2.cor);
    System.out.println(gato2.idade);
    System.out.println("\n");
...
```

Inicialização e exibição dos atributos

Classes e objetos

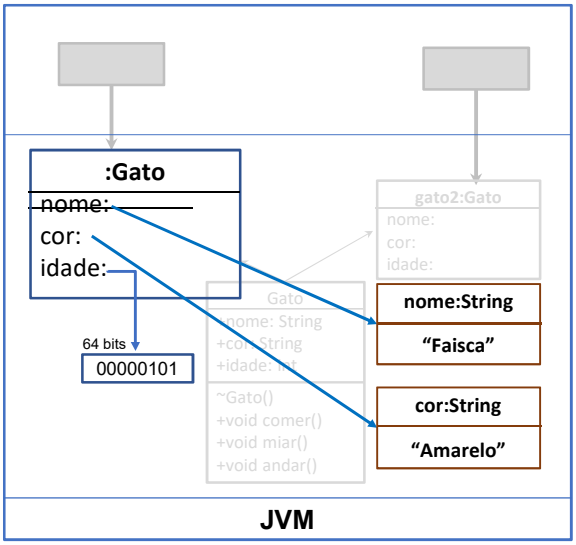
```
...

    gato1.nome = "Faisca" ;
    gato1.cor = "Amarelo" ;
    gato1.idade = 5 ;
    System.out.println(gato1.nome);
    System.out.println(gato1.cor);
    System.out.println(gato1.idade);
    System.out.println("\n");

    gato2.nome = "Chiclete" ;
    gato2.cor = "Preto" ;
    gato2.idade = 6 ;
    System.out.println(gato2.nome);
    System.out.println(gato2.cor);
    System.out.println(gato2.idade);
    System.out.println("\n");

...
```



Classes e objetos

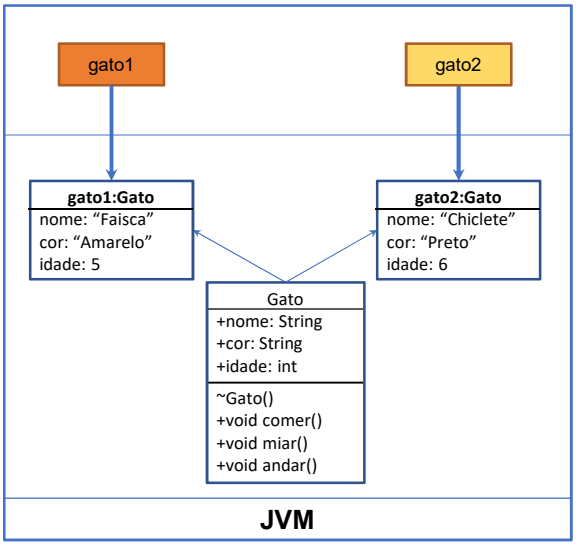
```
...

    gato1.nome = "Faisca" ;
    gato1.cor = "Amarelo" ;
    gato1.idade = 5 ;
    System.out.println(gato1.nome);
    System.out.println(gato1.cor);
    System.out.println(gato1.idade);
    System.out.println("\n");

    gato2.nome = "Chiclete" ;
    gato2.cor = "Preto" ;
    gato2.idade = 6 ;
    System.out.println(gato2.nome);
    System.out.println(gato2.cor);
    System.out.println(gato2.idade);
    System.out.println("\n");

...
```

Atenção! Zoom na criação de objetos da classe String e na inicialização de variáveis de tipo primitivo.



Classes e objetos

```
...

    gato1.nome = "Faisca" ;
    gato1.cor = "Amarelo" ;
    gato1.idade = 5 ;
    System.out.println(gato1.nome);
    System.out.println(gato1.cor);
    System.out.println(gato1.idade);
    gato1.miar();
    System.out.println("\n");

    gato2.nome = "Chiclete" ;
    gato2.cor = "Preto" ;
    gato2.idade = 6 ;
    System.out.println(gato2.nome);
    System.out.println(gato2.cor);
    System.out.println(gato2.idade);
    System.out.println("\n");
```



Classes e objetos

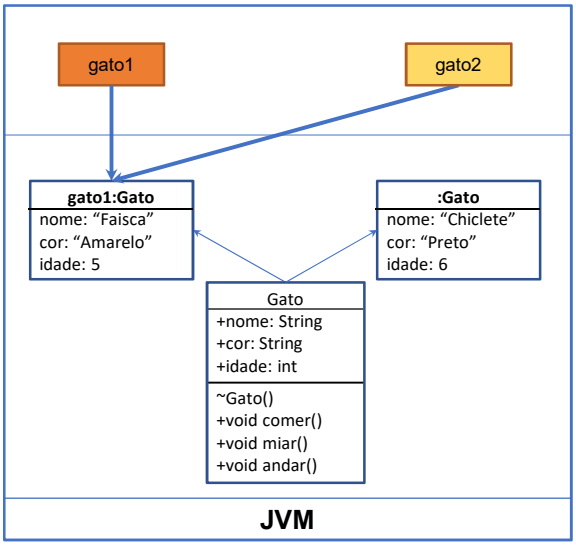
```
        gato2 = gato1;

        System.out.println(gato2.nome);
        System.out.println(gato2.cor);
        System.out.println(gato2.idade);
        System.out.println("\n");
    }
}
```

Agora, a variável gato2 passa a apontar para o objeto que gato1 está apontando

TesteObjetosPonteiros.java

Classes e objetos

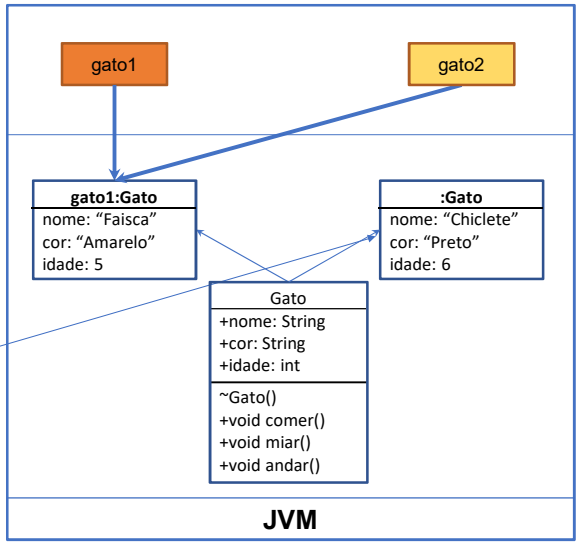
```
        gato2 = gato1;

        System.out.println(gato2.nome);
        System.out.println(gato2.cor);
        System.out.println(gato2.idade);
        System.out.println("\n");
    }
}
```



Classes e objetos

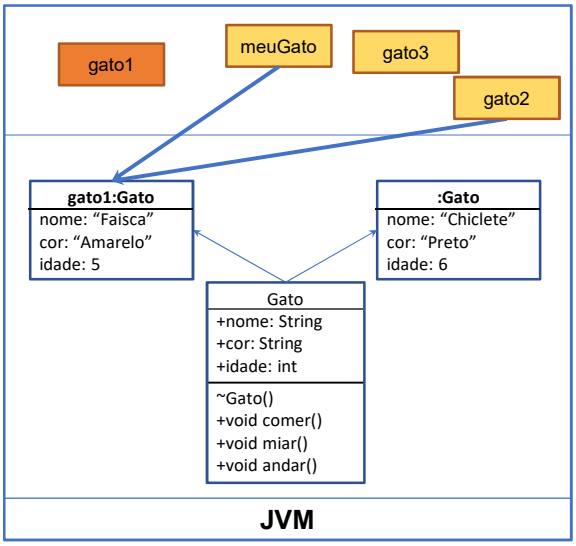
```
        gato2 = gato1;

        System.out.println(gato2.nome);
        System.out.println(gato2.cor);
        System.out.println(gato2.idade);
        System.out.println("\n");
    }
}
```

O que acontece com este objeto?



Classes e objetos

```
                gato2 = gato1;

        System.out.println(gato2.nome.charAt(0));
        System.out.println(gato2.cor);
        System.out.println(gato2.idade);
        System.out.println("\n");

        Gato meuGato = gato1;

        Gato gato3 = "null";
        gato1 = gato3;

    }
}
```



Classes e objetos

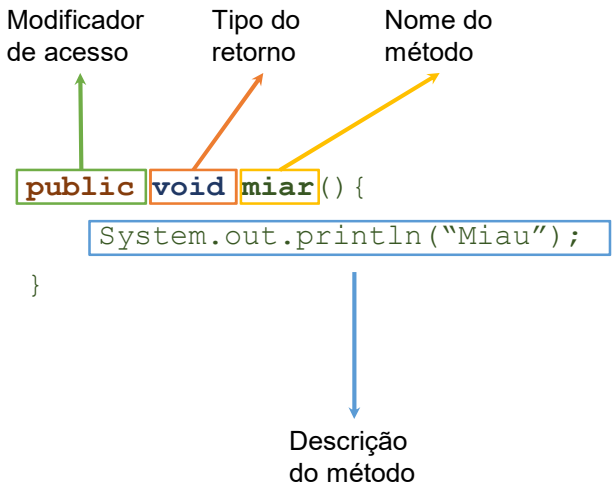
```
Faisca
Amarelo
5


Chiclete
Preto
6


Faisca
Amarelo
5
```

Métodos

- Métodos são ações/procedimentos/comportamentos
  - Permitem comunicação e/ou interação com outros objetos.
  - São equivalentes à funções em outras linguagens de programação (C ,por exemplo).
- Em Java não existe o conceito de “métodos globais”
  - Todos devem ser definidos dentro de uma classe.
  - Um conceito que se aproxima disso é usar métodos e variáveis “de classe”, com o modificador *static*.
- Sintaxe da **assinatura**

```
<Modificadores> [TipoRetorno]|void nomeDoMetodo(<lista de argumentos>)
{
      <corpo do método>
}
```

Métodos: assinatura



Métodos: modificadores

- Existem 3 modificadores de acesso que podem ser usados ao declarar métodos e atributos:
  - **`public`** : Menos restritivo de todos, permite acesso pelos métodos da própria classe, de classes derivadas desta e por qualquer outra classe externa dentro ou fora do pacote
  - **`protected`** : Permite acesso pelos métodos da própria classe e classes herdeiras
  - **`private`** : É o mais restritivo, permite acesso somente pelos métodos da própria classe.
- E se não for usado nenhum modificador de acesso na declaração?
  - É definido acesso do tipo *default/package* , permite acesso pelos métodos da própria classe, classes derivadas e outra classe dentro do pacote
    - **Vamos ver isso depois ...**
- Outros modificadores
  - **`synchonized`**, por exemplo

Métodos: retorno

- A conclusão da execução do método por ou não retornar uma informação
  - Sem retorno: palavra reservada `void` é usada na assinatura do método
  - Com retorno
    - `return` é usado no código para retornar o objeto ou valor do tipo especificado.
    - Tipos primitivos
    - Tipo composto (referência a objeto ou array)
- O nome do método deve começar com uma letra, _ , ou $. Os caracteres subsequentes são de escolha do desenvolvedor. Por convenção:
  - Geralmente usam-se verbos como nome;
  - Primeira letra do nome minúscula, letras iniciais internas maiúsculas.
    Ex: `getNome()`.

Métodos: argumentos de entrada

- Um método pode ter zero ou mais argumentos
  - Caso não tenha
    - ( ), parênteses vazios.
      - Declaração: `public void miar(){}`
      - Chamada: `gato1.miar();`
  - Caso tenha
    - Cada argumento deve ser declarado como define-se uma variável, especificando nome e tipo
    - Separados por vírgula
      - Declaração: `public int fazAlgo(String nome, [Tipo][Nome da variável], ...){}`
      - Chamada: `int a = gato1.fazAlgo("Felix", "dia", "mia", 3);`

Métodos: *getters* e *setters*

```
...
 String nome;
...
  public void setNome(String n){
    this.nome = n;
  }
...
  public String getNome(){
    return this.nome;
  }
...
```

Método setter

Método getter

Métodos: argumentos primitivo *versus* composto

- Valores primitivos (`char`, `int`, `float`, ...)
  - É criado uma cópia do valor que é passada ao método como argumento.
  - Se houver alguma mudança no valor feita dentro do método, somente a cópia sofrerá essas mudanças, mantendo a variável original inalterada.
- Instâncias de classe (objetos), arrays ou interfaces
  - É passado uma referência com o endereço de memória do elemento
  - Assim, qualquer mudança feita no objeto na execução do do método afetará o elemento

❑ Atenção! Ao se criar uma instância de objeto

- Ainda não temos a referência associada (podemos até não precisar de uma)
- O objeto fica na memória da JVM, "junto com todos os outros objetos"
- Mesmo tendo sido criado por outro objeto (aliás, não tem outra forma de se criarem objetos)
- Resumindo: **um objeto criado "normalmente" NÃO fica DENTRO de outro!**
- **Um objeto passado como parâmetro nunca "vai" para o método. Apenas a sua referência!**

Métodos: *de instância* versus *de classe*

De instância

- É o padrão
- A partir da classe, criam-se instâncias de objeto. É o caso da classe *Gato* do exemplo
- Cada objeto tem o seu conjunto de atributos e métodos
- São invocados desta forma:
  - `Objeto.Método`.
    Ex: `gato1.miar()`

De classe

- Precisa ser modificado.
- Os métodos/atributos de classe são marcados pelo uso da palavra reservada `static`
- Usamos os atributos e métodos de classe diretamente da classe (não precisamos instanciar um objeto para usar)
- São chamados diretamente da classe (mas podem ser chamados da instância se você quiser)
- Pode encarar como variáveis ou métodos "globais" para todos os objetos daquela classe
- Úteis, por exemplo, quando se quer apenas executar um método e gerenciar a criação de instâncias de objetos com seus estados.
- Exemplo clássico: a classe Math do JDK
  - Veja esta classe ... Quero apenas saber o seno de 30º
  - `double seno30 = Math.sin(30)`

Métodos: construtor

- Automaticamente chamado ao se criar uma instância de objeto (`new`)
- Bom para inicializar o objeto, ativar recursos, configurações iniciais, limpeza de buffer, etc.
- Tem o mesmo nome da classe e sem nenhum tipo de retorno
  - Nenhum mesmo, nem `void`
- Se nenhum construtor por explicitamente definido
  - um construtor padrão, que não receber nenhum argumento, é incluído por herança, pelo compilador *Java*.
    - **Toda classe herda da classe *Object***. Guarde isso para mais tarde ...
  - No entanto, se for definido, o construtor padrão não será incluído na classe automaticamente, tendo que ser colocado manualmente na classe
    - `public NomeDaClasse(){}`

Métodos: declaração de construtores

- Mais de um construtor pode ser definido por classe, usando o conceito de *sobrecarga*
  - Por exemplo, construtores que iniciam todos os atributos de um objeto
  - ... outros que só iniciam alguns
  - Ou que aceitem argumentos de tipos diferentes.
  - Seu uso depende da situação. Procure os construtores da classe String na documentação.

```java
//Construtor Padrão
public Gato(){ }

//Construtor com 2 parâmetros
public Gato(String n, String c){
  this.nome = n;
  this.cor = c;
}

//Construtor com 3 parâmetros
public Gato(String n, String c, int i){
  this.nome = n;
  this.cor = c;
  this.idade = i;
}
```

Gato.java

Métodos: chamada de construtores

```
...
Gato gato1 = new Gato();
Gato gato2 = new Gato("Faisca", "Vermelho");
Gato gato3 = new Gato("Luna", "Preto", 5);
...
```

Invocação ou chamada de método construtor

esteConstrutor.java

Exercícios

- Continue a construção da classe Pessoa, agora com métodos construtores, *setters, getters*, e outros como `falar()`.
- Se, por algum motivo, for necessário o controle do número de objetos em um programa, qual modificador eu devo usar?
  - Conte o número de instâncias criadas da classe Pessoa
- Continue avançando no tutorial da *Oracle*