Tipos de dados e operadores

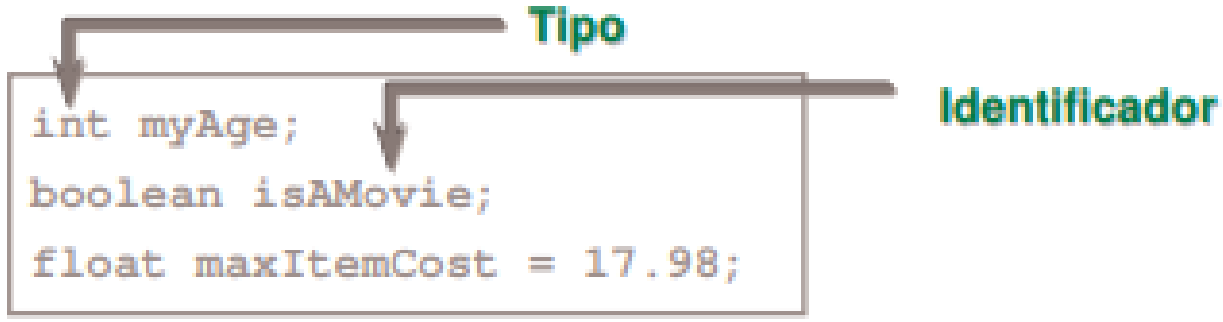
- O Java oferece primitivas para os tipos de dados básicos.
- As primitivas são a fundação para armazenar e utilizar informação
- Declarar e inicializar primitivas é a base da construção de tipos definidos pelo utilizador

Tipos de dados e operadores

- Os operadores manipulam dados e objetos
- Aceitam um ou mais argumentos e produzem um valor
- Java oferece 44 operadores diferentes.
- Alguns operadores alteram o valor do operando

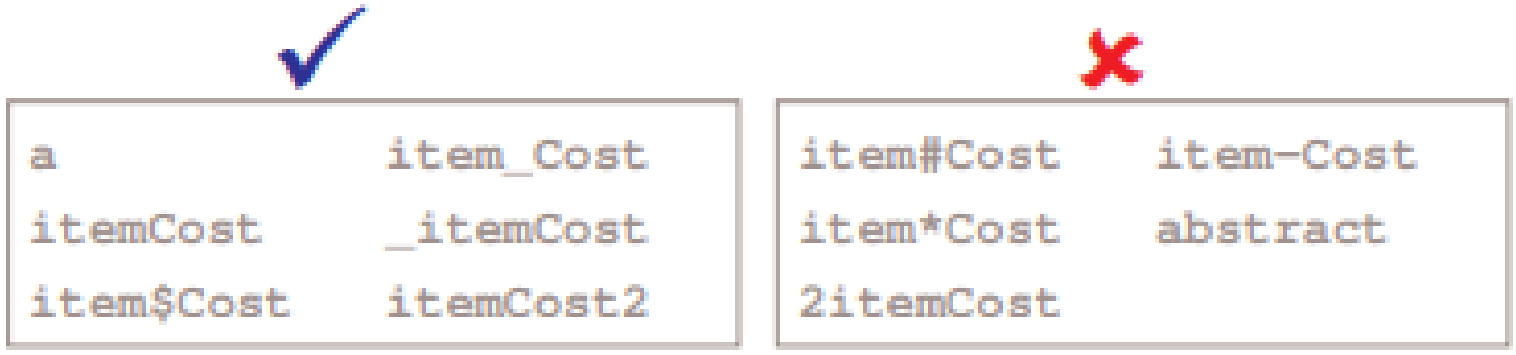
Variáveis

- Uma variável é a unidade básica de armazenamento
- As variáveis devem ser declaradas explicitamente.
- Cada variável tem um tipo, um identificador, e um âmbito.
- As variáveis podem ser inicializadas.



Nomes de Variáveis

- Os nomes das variáveis devem começar por uma letra do alfabeto, um underscore ou um $.
- Os outros caracteres podem incluir dígitos
- Deve-se utilizar nomes elucidativos para variáveis, por exemplo, customerFirstName.



Declaração de Variáveis

- A forma básica de declaração de uma variável:

tipo *identifier* [= valor]

```
public static void main(String[] args) {
    int itemsRented;
    float itemCost;
    int i, j, k;
    double interestRate;
}
```

- As variavés podem ser inicializadas quando declaradas.

Declaração de Variáveis

- As variáveis locais estão contidas apenas num método ou bloco de código.
- As variáveis locais devem ser inicializadas antes de ser usadas.

```
class Rental {
  private int instVar;      // instance variável
  public void addItem() {
    float itemCost = 3.50; // local variável
    int numOfDays  = 3;    // local variável
  }
}
```

Declaração de variáveis

- Encontra os erros no código abaixo e corrigi-los;

```
1  byte sizeof = 200;
2  short mom = 43;
3  short hello mom;
4  int big = sizeof * sizeof * sizeof;
5  long bigger = big + big + big     // ouch
6  double old = 78.0;
7  double new = 0.1;
8  boolean consequence = true;
9  boolean max = big > bigger;
10 char maine = "New England state";
11 char ming = 'd';
```

Operadores

Cinco tipos de operadores:

- Atribuição;
- Aritméticos;
- Manipulação de bits;
- Relacionais;
- Booleanos

Operadores: Atribuição

- A expressão da direita é atribuída à variável da esquerda.

```
int var1 = 0, var2 = 0;
var1 = 50;          // var1 now equals 50
var2 = var1 + 10; // var2 now equals 60
```

- A expressão da direita é sempre avaliada antes da atribuição
- As atribuições podem ser agrupadas:

```
var1 = var2 = var3 = 50;
```

Operadores: Aritméticos

- A maioria das operações resultam num int ou long:

```
byte b1 = 1, b2 = 2, b3;
b3 = b1 + b2;      // error: result is an int
                   // b3 is byte
```

- Valores byte, char, e short são promovidos a int antes da operação.
- Se algum argumento for long, o outro é promovido a long, e o resultado é long.

Incrementar e Decrementar

- O operador ++ incrementa 1 unidade:

```
int var1 = 3;
var1++;             // var1 now equals 4
```

- O operador ++ pode ser usados de duas maneiras:

```
int var1 = 3, var2 = 0;
var2 = ++var1;    // Prefix:  Increment var1 first,
                  //          then assign to var2.
var2 = var1++;    // Postfix: Assign to var2 first,
                  //          then increment var1.
```

- O operador -- decrementa 1 unidade.

Operadores Lógicos

- Os resultados de expressões Booleanas podem ser combinados usando operadores lógicos:

&&	&	e (with / without short-circuit evaluation)
||	|	or (with / without short-circuit evaluation)
^		exclusive or
!		not

```
int var0 = 0, var1 = 1, var2 = 2;
boolean res = true;
res = (var2 > var1) & (var0 == 3);    // now false
res = !res;                           // now true
```

Atribuição Composta

- O operador de atribuição pode ser combinado com qualquer operador binário convencional

```
double total=0, num = 1;
double percentage = .50;
…
total  = total + num;     // total is now  1
total += num;             // total is now  2
total -= num;             // total is now  1
total *= percentage;      // total is now .5
```

Precedência de Operadores

Order	Operadores	Comments	Assoc.
1	++ -- + - ~ !(tipo)	Unary operadores	R
2	* / %	Multiply, divide, remainder	L
3	+ - +	Add, subtract, add string	L
4	<< >> >>>	Shift (>>> is zero-fill shift)	L
5	< > <= >= instanceof	Relational, tipo compare	L
6	== !=	Equality	L
7	&	Bit/logical e	L
8	^	Bit/logical exclusive OR	L
9	|	Bit/logical inclusive OR	L
10	&&	Logical e	L
11	||	Logical OR	L
12	?:	Conditional operador	R
13	= op=	Assignment operadores	R

Precedências

- A precedência de um operador determina a ordem pela qual os operadores são executados:

```
int var1 = 0;
var1 = 2 + 3 * 4;     // var1 now equals 14
```

- Os operadores com a mesma precedência são executados da esquerda para a direita:

```
int var1 = 0;
var1 = 12 - 6 + 3;    // var1 now equals 9
```

- Os parêntesis permitem alterar a ordem definida

Concatenação de String's

- O operador + cria e concatena strings:

```
String name = "Jane ";
String lastName = "Hathaway";
String fullName;
name = name + lastName;      // name is now
                             //"Jane Hathaway"
                             //     OR
name += lastName;            // same result
fullName = name;
```

PROGRAMAÇÃO ORIENTADA A OBJETOS

Hoje em dia, a maioria das linguagens de programação são orientadas a objetos como Java, C#, Python e C++ e, apesar de terem algumas diferenças na implementação, todas seguem os mesmos princípios e conceitos.

Muitos programadores, apesar de utilizarem linguagens orientadas a objetos, não sabem utilizar alguns dos principais conceitos desse paradigma orientado a objetos e, por isso, desenvolvem sistemas com alguns erros conceituais e acabam escrevendo mais código que o necessário, não conseguindo reutilizar o código como seria possível.

Vamos começar por fazer uma revisão dos principais conceitos de orientação a objetos e ver diversos exemplos de implementação dos conceitos de orientação a objetos em Java.

Os principais conceitos do paradigma orientado a objetos são Classes e Objetos, Associação,Encapsulamento, Herança e Polimorfismo.

A programação orientada a objetos tem o propósito principal de aproximar o mundo lógico da programação e o mundo em que vivemos. À vista disso, ela parte do princípio de que tudo é objeto

No entanto, eles necessitam ser criados (tal qual no mundo real).

Para isso, eles precisam de uma espécie de forma, de um invólucro, algo que possa dar-lhes ao menos seu aspeto inicial: um ponto de partida!

Imaginem uma unidade fabril a realizar a fabricação de um determinado produto.

Esse item (objeto) pode ser qualquer coisa: um carro, uma televisão ou mesmo ovos de Páscoa.

Em todos os casos, no início desta produção, haverá o uso de uma forma para dar o aspeto inicial dos objetos finais.

Todas estas formas são equivalentes ao conceito de classes em POO.

Após terminado esse processo é que os produtos passam por personalizações: os carros ganham cores e acessórios, as televisões recebem diferentes programações internas e os ovos de Páscoa serão de embalagens e recheios diversos.

Esse quadro completa nosso comparativo de POO: todas essas características individuais dos objetos são os seus atributos, que pertencem somente a eles. Já suas funcionalidades (caso tenham alguma) são os seus métodos. Por exemplo, acelerar e travar são funcionalidades de um carro.

São estas quatro características — objetos, classes, atributos e métodos — que definem o paradigma de programação orientada a objetos. Lembrem-se de que algumas particularidades podem (e são) compartilhadas entre diferentes objetos, conferindo grande versatilidade a esse tipo de paradigma.

Quais as vantagens de usar a POO?

Confiável
A geração de código baseado no conceito de objetos e classes fornece uma grande independência ao programa. Assim, qualquer intervenção que seja necessária não afetará outros pontos do sistema. Isso confere robustez e confiabilidade.

Oportuno
A criação paralela de código torna todo o processo bastante ágil. Ganha-se em tempo e eficiência, tornando um software com paradigma **POO** oportuno. Várias equipas podem trabalhar no mesmo projeto de forma independente.

Ajustável
Esta característica diz respeito ao processo de manutenção do código-fonte. Ao atualizar uma parte pequena, o conceito de herança garante que, automaticamente, todas as partes que utilizarem tal método sejam beneficiadas.
Esta característica torna especial o paradigma de POO, pois é muito comum que equipes de desenvolvimento de softwares sejam escaladas para trabalhar com softwares já existentes. Dessa forma, torna-se mais fácil executar o trabalho de manutenção.

Extensível
O uso do princípio da reutilização do software adiciona funcionalidades ao sistema já existente. Não é preciso “reinventar a roda”, reescrevendo o código. Isso confere maior capacidade de estender um sistema já existente.

Quais as vantagens de usar a POO?

Reutilizável
Um mesmo objeto pode ser utilizado em aplicações diferentes, desde que sejam compatíveis. Se tivéssemos um objeto "aluno", por exemplo, ele poderia ser utilizado em sistemas de empresas diferentes, desde que elas contassem com alunas e alunos na sua estrutura.
Assim, tanto uma escola de música como uma academia poderiam fazer uso do objeto "aluno" em um possível software para uso próprio, pois ambas as empresas têm essas pessoas como seu público.

Natural
O conceito de **POO** que traz para a programação o mundo concreto, tal qual vemos no dia-a-dia, faz com que se ganhe naturalidade. O ponto de vista humano se torna mais próximo do virtual. Assim, pensa-se no problema geral, em vez de concentrar o foco em pormenores.

Fundamentos da programação orientada a objetos

A programação orientada a objetos é um paradigma de programação em que tudo é representado como um objeto.

Os objetos passam mensagens um para o outro. Cada objeto decide o que fazer com uma mensagem recebida.

POO concentra-se nos estados e comportamentos de cada objeto.

O que são objetos?

Um objeto é uma entidade que possui estados e comportamentos.

Por exemplo cão, gato e veículo.

Para ilustrar, o cão tem seus estados como idade, cor, nome e comportamentos como comer, dormir e correr.
O estado diz-nos como o objeto se parece ou quais propriedades ele possui.
O comportamento diz-nos o que o objeto faz.
Na verdade, podemos representar um cão do mundo real num programa como um objeto de software, definindo os seus estados e comportamentos.
Objetos de software são a representação real de objetos do mundo real. A memória é alocada na RAM sempre que criar um objeto lógico.
Um objeto também é referido a uma instância de uma classe. Instanciar uma classe significa a mesma coisa que criar um objeto.

O importante a lembrar ao criar um objeto é: o tipo de referência deve ser o **mesmo tipo** ou um **super tipo** do tipo de objeto. Veremos o que é um tipo de referência posteriormente.

Classe

Uma classe é uma forma de definir um tipo de dado numa **linguagem orientada a objeto**.

Ela é formada por dados e comportamentos.

Para definir os dados são utilizados os atributos, e para definir o comportamento são utilizados métodos. Depois que uma classe é definida podem ser criados diferentes objetos que utilizam a classe.

Uma classe é um modelo ou blueprint a partir do qual os objetos são criados.
Imaginem uma classe como um cortador de biscoitos e objetos como cookies.

Classe

Classes definem estados como variáveis de instância e comportamentos como métodos de instância.

As variáveis de instância também são conhecidas como variáveis de membro.

A classe não consome nenhum espaço.

Para dar-vos uma ideia sobre classe e objetos, vamos criar uma classe Cat que represente estados e comportamentos do mundo real Cat.

```
public class Cat
{
/* Instance variables: states of Cat */
String name; int age; String color; String breed;

/* Instance methods: behaviors of Cat */
void sleep()
{
System.out.println("Sleeping");
}
void play()
{
System.out.println("Playing");
}
void feed()
{
System.out.println("Eating");
}
}
```

Classe

Agora, definimos com sucesso um modelo para o gato. Digamos que temos dois gatos chamados Thor e Rambo.
Como podemos defini-los no nosso programa?
Primeiro, precisamos criar dois objetos da classe Cat.

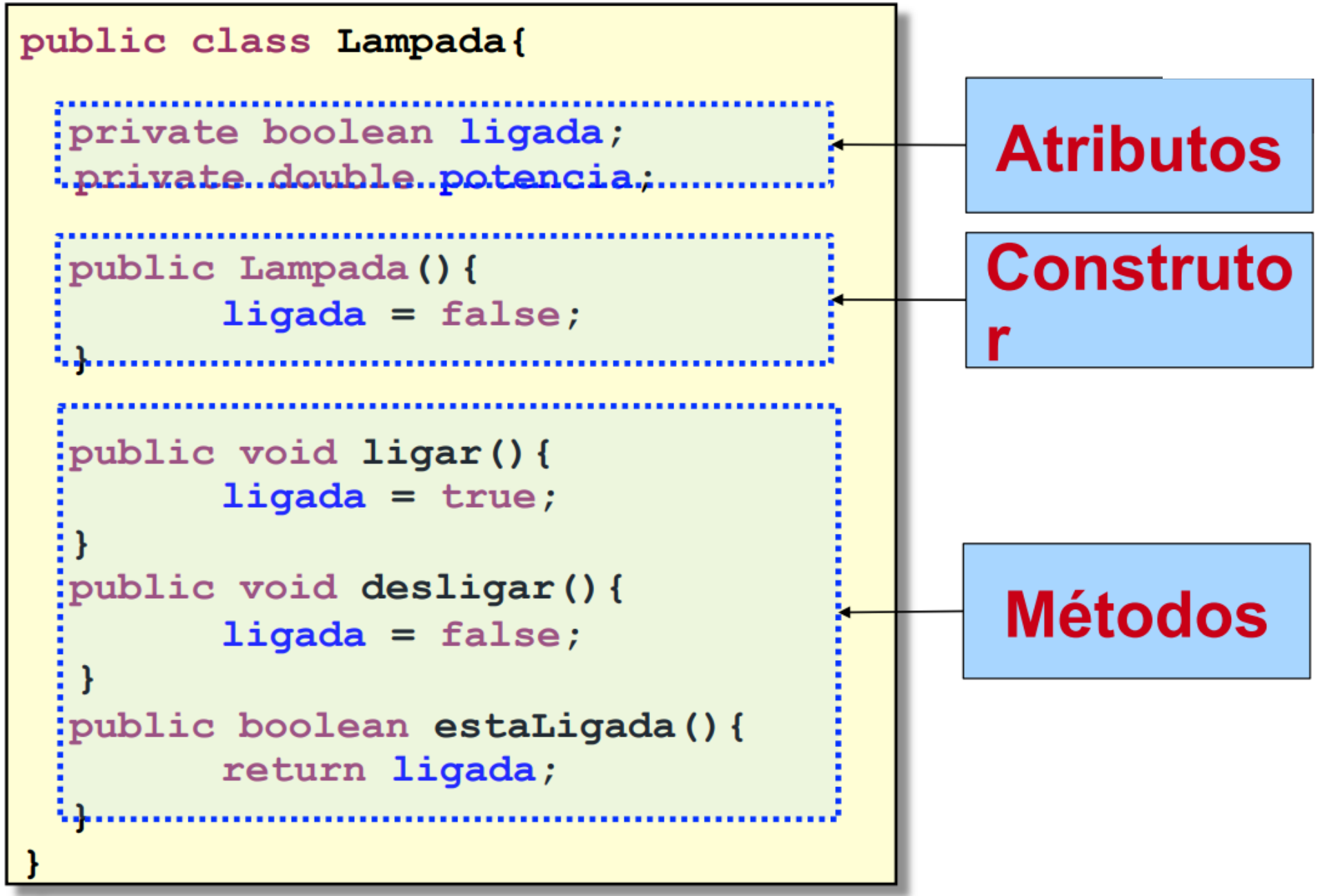
```
public class Main
{
public static void main(String[] args)
{
Cat thor = new Cat();
Cat rambo = new Cat();
}
}
```

Em seguida, definiremos os seus estados e comportamentos..

```
public class Main
{ public static void main(String[] args)
{

/* Creating objects */
Cat1 thor = new Cat();
Cat2 rambo = new Cat();

/* Defining Thor cat */
thor.name = "Thor";
thor.age = 3;
thor.breed = "Russian Blue";
thor.color = "Brown";
thor.sleep();

/* Defining Rambo cat */
rambo.name = "Rambo";
rambo.age = 4;
rambo.breed = "Maine Coon";
rambo.color = "Brown";
rambo.play();
}
}
```

Como nos exemplos de código acima, podemos definir a nossa classe, instanciar (criar objetos) e especificar os estados e comportamentos desses objetos.

Exemplo


public class Lampada{
private boolean ligada;
private double potencia;
public Lampada(){
ligada = false;
}
public void ligar(){
ligada = true;
}
public void desligar(){
ligada = false;
}
public boolean estaLigada(){
return ligada;
}
}
Atributos
Construtor
Métodos

O que fazer

Edite o arquivo fonte

• Salve com a extensão .java

– Se a classe for **public** o nome do arquivo deve ser o mesmo nome da classe com a extensão .java

• Compile com o javac

• Será criado o arquivo **.class** que contém a classe a ser usada

– Este arquivo deverá estar no CLASSPATH da JVM

• CLASSPATH é o lugar onde a JVM procura as classes

• Variável de ambiente CLASSPATH

• Crie um programa que use a classe

– Objetos dessa classe podem ser criados e manipulados

O que fazer

```
public class Main {
    public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      Lampada l = new Lampada();
      int opcao = 0;
      do
      {
        System.out.println("0) Sair 1) Ligar 2) desligar 3) Mostrar ");
        System.out.print("Digite opção: "); opcao = sc.nextInt();
        switch (opcao){
          case 1: l.ligar(); break;
          case 2: l.desligar() ; break;
          case 3:
            if (l.estaLigada())
              System.out.println("Ligada");
            else
              System.out.println("Desligada");
            break;
        }

      } while (opcao != 0);
      System.out.println("Fim do programa!");
    }
}
```

Atributos

São as variáveis de instância
– Fazem parte de cada objeto (instância)

• Declarada fora dos métodos

• "Vivem" enquanto o objeto "viver"

• São pré-inicializadas
– boolean ==> false
– tipo aritmético ==> 0
– char ==> ‘\u0000’
– class ==> null 7

Obs: Todo objeto possui um identificador chamado **this**, que é uma referência para o próprio objeto (próximo slide).

this

Todo objeto possui um atributo que é uma referência a ele mesmo
– Usado para acesso a membros do proprio objeto

- this.membro

– Evita conflito
- Com parâmetros de metodos, por exemplo
- Exemplo:

```
class Qualquer
{    int x, y;
public void mover(int x,int y){
          this.x = x;
          this.y = y;
}
}
```

Métodos

Declaração de método:

– <opcoes> tipoRetorno nomeMetodo (parâmetros)

• public int metodo(float x)

• Passagem de parâmetros:

– Deve ser informados o tipo e identificador dos parâmetros.

– Funciona no método como uma variável normal

– Passam o valor do identificador

Corpo do método:

– Implementa as operações do método

– Fica entre chaves ({})

– Variáveis podem ser criadas

• Ela é dita local

• Não é pré-inicializada.

• Só existe enquanto o método está em execução

Construtores

Mesmo nome da classe

- **Não possui retorno**

- **Podem ser vários**
  – Diferença na quantidade e tipo dos parâmetros

- **Construtor padrão é fornecido**
  – Se não houver pelo menos um definido
  – Não possui parâmetros

- É chamado na execução do **new**

Outro exemplo

Ponto

– Plano cartesiano
– Coordenadas X e Y
– Pode ser movido de lugar
– Podemos saber sua distância da origem

```
public class Ponto{
 private double x,y;
 public Ponto(){
   x = 0; y = 0;
 }
 public Ponto(double x, double y)
 {this.x = x, this.y = y;}

 public void moverPara(double x, double y){
  this.x = x;  this.y = y;
 }

 public double getX(){ return x;}

 public double getY(){ return y;}

 public double distanciaOrigem(){
  double distancia;
  distancia = (double)Math.sqrt(x*x +y*y);
  return distancia;
 }
 public String toString(){
  return "Ponto ("+x+","+y+")";
 }
}
```

Visibilidade

- Proteção de acesso
  - Proteger o interior da classe
- Explicitar o que usuários (da classe) precisa saber
- pode ser:
  - **private**: Apenas membros da classe têm acesso
  - **protected**: Membros da classe e subclasses
  - **public**: Todos têm acesso
  - **default**: Apenas membros do mesmo pacote

Proteção de acesso

Interface

– Visão externa da classe
– O que os objetos da classes fazem
– Definem o “contrato” da classe
    • O que o cliente precisa conhecer da classe

• **Implementação**

– Visão interna da classe
– Como os objetos fazem as operação
– Representação interna
    • cliente não precisa (nem deve) conhecer a implementação
– Realizam o contrato definido pela interface

Elementos do modelo de objetos

Abstração

– Uma abstração denota as características essenciais de um objeto que o distingue de todas as outras espécies de objetos e assim provê limites conceituais bem definidos, sempre relativos à perspetiva de um observador.

Encapsulamento

– Encapsulamento é o processo de esconder todos os detalhes de um objeto que não contribuem para suas características essenciais

Proteção de acesso

Atributos fazem parte a implementação
– Declare-os como **private**

• Nem todos os métodos fazem parte da interface
– Métodos que servem para auxiliar outros métodos

• Declare-os **private**

• Deixe **public** apenas o que o cliente deve saber
– Métodos da interface

Convenção de nomes

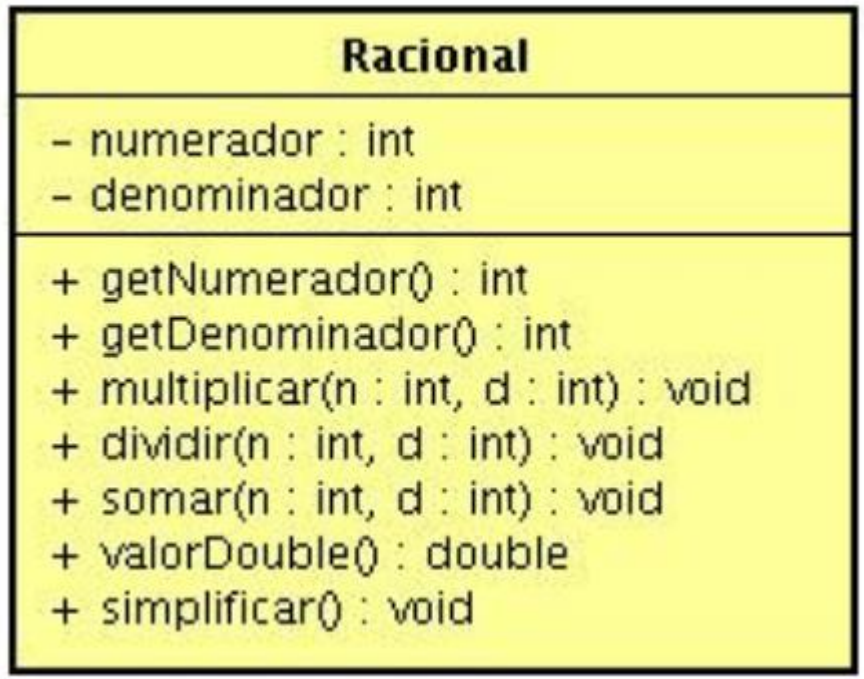
Variáveis e Métodos:

Use minúsculas.
Se o nome consiste de várias palavras, concatene-as e use a primeira letra de cada uma delas em maiúsculo.
Variáveis: **raio** e **area**
Método: **calcularArea**

Nomes de Classes:

Use a Primeira letra de cada palavra em maiúscula
Classe: **Circulo, Ponto, NumeroComplexo**

Exercício



Implemente a classe Racional
– Representam uma fração
– Dois construtores

• Sem parâmetros
– 1/1

• Com dois parâmetros
– numerador e denominador
– Métodos que realizam as operações

• Recebem numerador e denominador que representam a fração da operação
• modificam a fração

• Crie um programa para testar sua classe