

# Programação Orientada a Objetos - POOS3

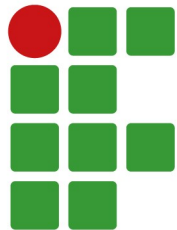
1

Tecnologia em Análise e Desenvolvimento de Sistemas

## Aula 3

Introdução à Programação Orientada a  
Objetos - Continuação

2º semestre de 2018



**INSTITUTO FEDERAL**

São Paulo

Câmpus Araraquara

# Momento da Revisão

---



# Exercício Avaliativo 4

---

- Crie uma classe **USMoney** com dois atributos inteiros: **dollars** e **cents**.
- Adicione um construtor com dois parâmetros para a inicialização do objeto USMoney. O construtor deve verificar se o valor de cents está entre 0 e 99 e, se não estiver, transferir alguns dos cents para o atributo dollars para que ela passe a ter entre 0 e 99.
- Implemente um método **plus** que recebe um objeto USMoney como argumento. Esse método deve criar e retornar um novo objeto USMoney representando a soma dos objeto cujo método plus() está sendo chamado mais o argumento, sem modificar os valores dos dois objetos já existentes.
- Deve-se assegurar que o valor do atributo cents do novo objeto esteja entre 0 e 99. Por exemplo, se x for um objeto USMoney com 5 dollars e 80 cents e se y for um objeto USMoney de 1 dollar e 90 cents, **x.plus(y)** retornará um novo objeto USMoney com 7 dollars e 70 cents.

```
public class USMoney {
    private int dollar;
    private int cents;

    public USMoney(int argDollar, int argCents) {
        if(argDollar <= 0 || argCents <= 0) {
            dollar = 0;
            cents = 0;
        } else {
            dollar = argDollar;
            cents = argCents;
        }
        simplifica();
    }

    private void simplifica() {
        int esquerda;
        esquerda = cents / 100;
        cents = cents % 100;
        dollar += esquerda;
    }

    public String toString() {
        String txt;
        txt = "U$ " + dollar + ".";
        if(cents < 10)
            txt += "0";
        txt += cents;
        return txt;
    }
}
```

```
public USMoney plus(USMoney money) {
    USMoney novo = new USMoney(getDollar() +
        money.getDollar(), getCents() + money.getCents());
    return novo;
}

public int getDollar() {
    return dollar;
}

public void setDollar(int dollar) {
    this.dollar = dollar;
}

public int getCents() {
    return cents;
}

public void setCents(int cents) {
    this.cents = cents;
    simplifica();
}
}
```

```
public class Main {  
  
    public static void main(String[] args) {  
        USMoney valor1, valor2, soma;  
  
        valor1 = new USMoney(10, 3);  
        valor2 = new USMoney(3, 71);  
  
        soma = valor1.plus(valor2);  
  
        System.out.println(soma.toString());  
    }  
}
```



De sobrecarga,  
falar iremos.

Muito legal, mas utilizo  
com frequência valores  
inteiros, ou seja, sem os  
centavos. Terei que  
construir o objeto sempre  
com zero cents?



---

# Sobrecarga

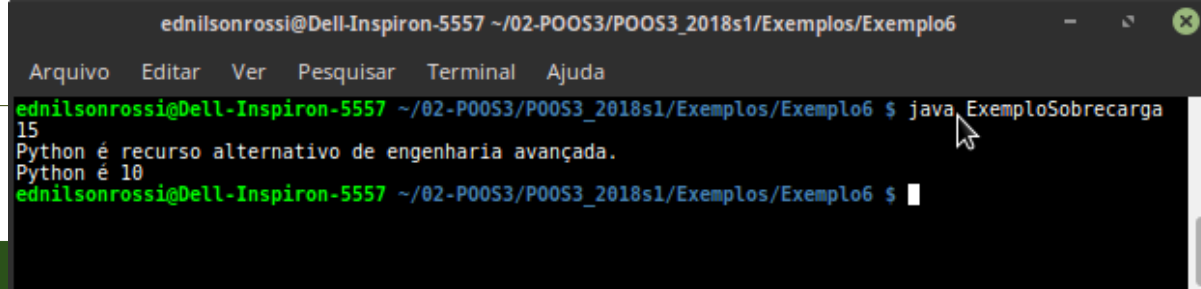
# Sobrecarga (overloading)

```
public class ExemploSobrecarga{
    public static void main(String args[]){
        int n1, n2, n3;
        String str1, str2, str3;

        n1 = 5;
        n2 = 10;
        str1 = "Python é ";
        str2 = "recurso alternativo de engenharia avançada.";

        n3 = n1 + n2;
        str3 = str1 + str2;

        System.out.println(n3);
        System.out.println(str3);
        System.out.println(str1 + n2);
    }
}
```

A screenshot of a terminal window with a dark background. The title bar shows the user 'ednilsonrossi' on a 'Dell-Inspiron-5557' machine, in the directory '~/02-POOS3/POOS3\_2018s1/Exemplos/Exemplo6'. The terminal has a menu bar with 'Arquivo', 'Editar', 'Ver', 'Pesquisar', 'Terminal', and 'Ajuda'. The command 'java ExemploSobrecarga' has been executed, resulting in three lines of output: '15', 'Python é recurso alternativo de engenharia avançada.', and 'Python é 10'. The prompt is ready for the next command.

```
ednilsonrossi@Dell-Inspiron-5557 ~/02-POOS3/POOS3_2018s1/Exemplos/Exemplo6
Arquivo  Editar  Ver  Pesquisar  Terminal  Ajuda
ednilsonrossi@Dell-Inspiron-5557 ~/02-POOS3/POOS3_2018s1/Exemplos/Exemplo6 $ java ExemploSobrecarga
15
Python é recurso alternativo de engenharia avançada.
Python é 10
ednilsonrossi@Dell-Inspiron-5557 ~/02-POOS3/POOS3_2018s1/Exemplos/Exemplo6 $
```

# Sobrecarga (overloading)

```
public class USMoney {  
  
    private int dollar;  
    private int cents;  
  
    public USMoney() {  
        dollar = 0;  
        cents = 0;  
    }  
  
    public USMoney(int argDollar) {  
        if(argDollar < 0) {  
            dollar = 0;  
            cents = 0;  
        }else {  
            dollar = argDollar;  
            cents = 0;  
        }  
    }  
}
```

```
    public USMoney(int argDollar, int argCents) {  
        if(argDollar <= 0 || argCents <= 0) {  
            dollar = 0;  
            cents = 0;  
        }else {  
            dollar = argDollar;  
            cents = argCents;  
        }  
        simplifica();  
    }  
    //...  
}
```

Três construtores?  
Como isso  
funciona?





# Sobrecarga (overloading)

---

- Na programação orientada a objetos, um método aplicado a um objeto é selecionado para execução através da sua assinatura e da verificação a qual classe o objeto pertence.
- Através do mecanismo de sobrecarga (overloading), dois métodos de uma mesma classe podem ter o mesmo nome, desde que suas listas de argumentos sejam diferentes, constituindo assim uma assinatura diferente.
- Tal situação não gera conflito pois o compilador é capaz de detectar qual método deve ser escolhido a partir da análise dos tipos de argumentos do método.

# Sobrecarga (overloading)

---

- Por fim, sobrecarregar um método (construtor ou não) é permitir que se utilize o mesmo nome de chamada para operar sobre uma variedade de tipos de dados.
- Podemos ter vários métodos com o mesmo nome em uma mesma classe, porém os tipos dos argumentos devem ser diferentes.

# Sobrecarga (overloading)

- Em uma classe qualquer podemos ter dois métodos setValor():
  - `public void setValor(int varA){ }`
  - `public void setValor(float varB){ }`
- Observe que o tipo dos argumentos é diferente.
- Diferente deste exemplo:
  - `public void setValor(int varA){ }`
  - `public void setValor(int varB){ }`

No segundo exemplo há um erro, pois o nome dos argumentos é diferente mas o tipo é o mesmo.



# Material adicional

- **Leitura Obrigatória**

- Capítulo 6.12 → Deitel, P.; Deitel, H. Java como programar. 10 ed. São Paulo: Pearson Education do Brasil, 2017.

- **Videoaulas**

- [https://www.youtube.com/watch?v=ZpssJov\\_5\\_A](https://www.youtube.com/watch?v=ZpssJov_5_A)
  - Aula que trata sobre sobrecarga
- <https://www.youtube.com/watch?v=YvTiyjeXJZU>
  - Aula que trata sobre sobrecarga
- <https://www.devmedia.com.br/sobrecarga-e-sobreposicao-de-metodos-em-orientacao-a-objetos/33066>
  - Artigo que trata sobre sobrecarga e reescrita de métodos. Reescrita será assunto futuro!
- [https://www.youtube.com/watch?v=uq4O\\_\\_CGPdo](https://www.youtube.com/watch?v=uq4O__CGPdo)
  - Aula que trata sobre sobrecarga de construtor



---

this

# this

---

- Quando um método é chamado, ele recebe automaticamente um argumento implícito, que é a referência ao objeto chamador.
- Esta referência se chama **this**.

# this

```
public class USMoney {  
  
    private int dollar;  
    private int cents;  
  
    public USMoney() {  
        dollar = 0;  
        cents = 0;  
    }  
  
    public USMoney(int dollar) {  
        if(dollar < 0) {  
            this.dollar = 0;  
            this.cents = 0;  
        }else {  
            this.dollar = dollar;  
            this.cents = 0;  
        }  
    }  
}
```

```
    public USMoney(int dollar, int cents) {  
        if(dollar <= 0 || cents <= 0) {  
            this.dollar = 0;  
            this.cents = 0;  
        }else {  
            this.dollar = dollar;  
            this.cents = cents;  
        }  
        simplifica();  
    }  
}
```

Nesta implementação sempre que nos referimos aos atributos da classe utilizamos o **this** antes do atributo. Não precisamos ficar inventando nomes para os argumentos dos métodos.

No futuro outras utilidades :-D



# Material adicional

---

- Videoaula
  - <https://www.youtube.com/watch?v=RLzR--Pwvcs>
  - <https://www.youtube.com/watch?v=f6zbLCwq71w>

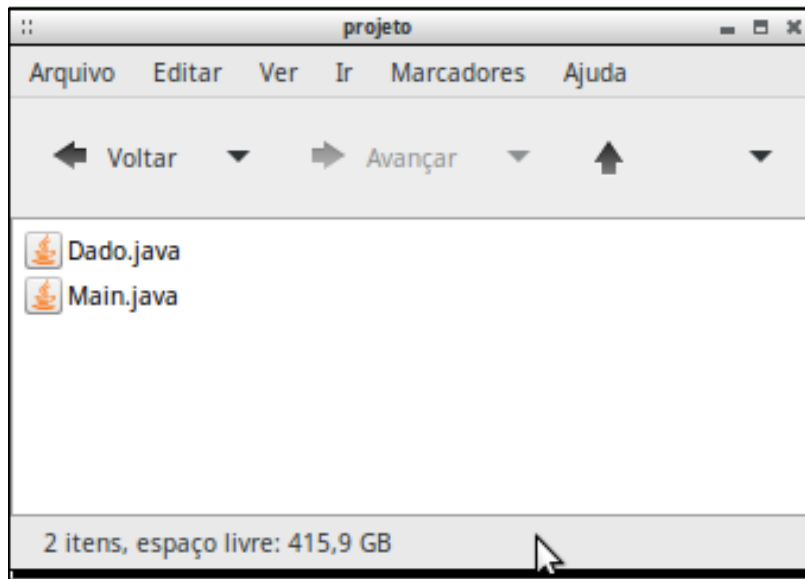




---

# Organização de pacotes

# Organização de pacotes



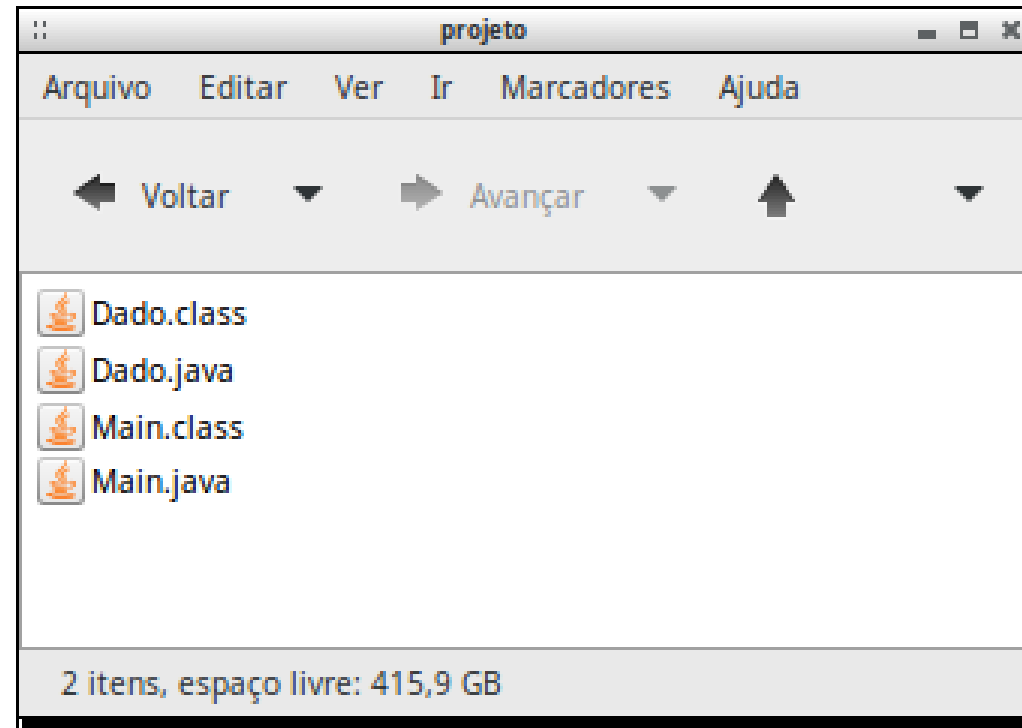
```
ednilsonrossi@Dell-Inspiron-5557 ~/Documentos/IFSP/01-Disciplinas/2018/2/POOS3/POOS3_2s2018/Exemplos/Exemplo3Dados/projeto
$ javac *.java
ednilsonrossi@Dell-Inspiron-5557 ~/Documentos/IFSP/01-Disciplinas/2018/2/POOS3/POOS3_2s2018/Exemplos/Exemplo3Dados/projeto
$ java Main
Face: 2
Face: 3
Face: 2
Face: 6
Face: 3
ednilsonrossi@Dell-Inspiron-5557 ~/Documentos/IFSP/01-Disciplinas/2018/2/POOS3/POOS3_2s2018/Exemplos/Exemplo3Dados/projeto
$
```



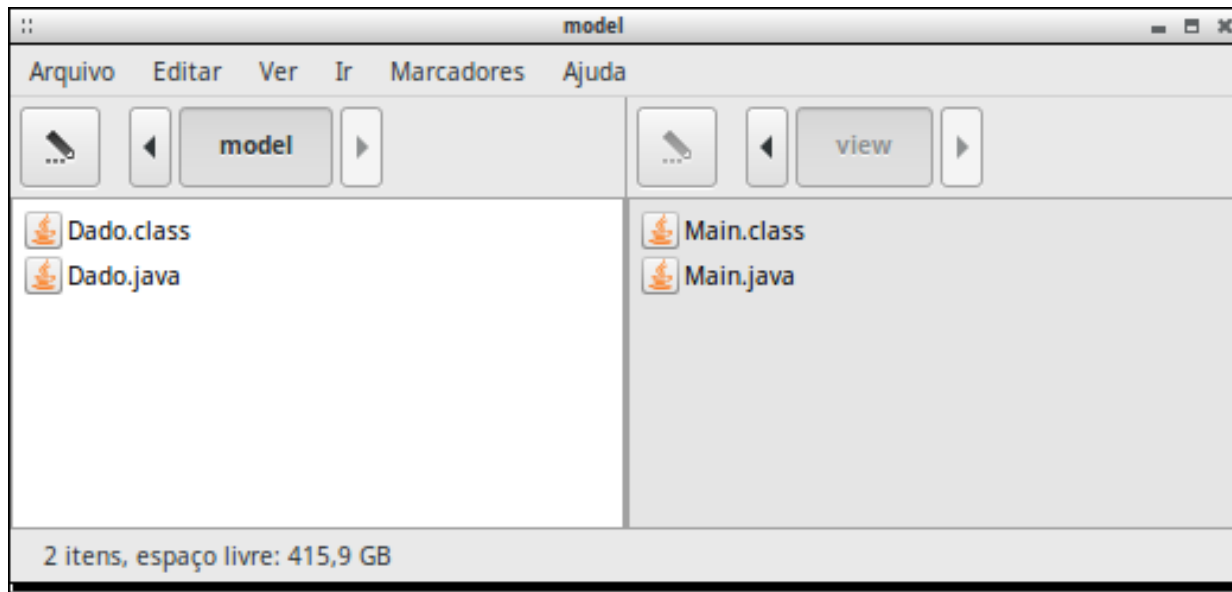
No exemplo do projeto de dados, você deve ter criado um diretório para o projeto e inserido nele as classes do projeto. Após compilar, usando javac, obtive os bytecodes e foi possível executar o programa.

# Organização de pacotes

- Contudo, a organização do projeto não fica adequada.
- Observe que temos uma classe que define a **lógica de negócio** e outra que faz a **interação com o usuários**.
  - Lógica de negócio
    - pacote model
  - Interação com usuário
    - pacote view



# Organização de pacotes



Não basta separar em diretório, é preciso informar qual o **package** de cada classe e também realizar os **imports** adequados.



```
package model;
```

```
import java.util.Random;
```

```
public class Dado{  
    private int face;  
  
    public Dado(){  
        sortear();  
    }  
  
    private void sortear(){  
        Random random = new Random();  
        face = random.nextInt(6) + 1;  
    }  
  
    public int getFace(){  
        return face;  
    }  
  
    public void jogar(){  
        sortear();  
    }  
}
```

```
package view;
```

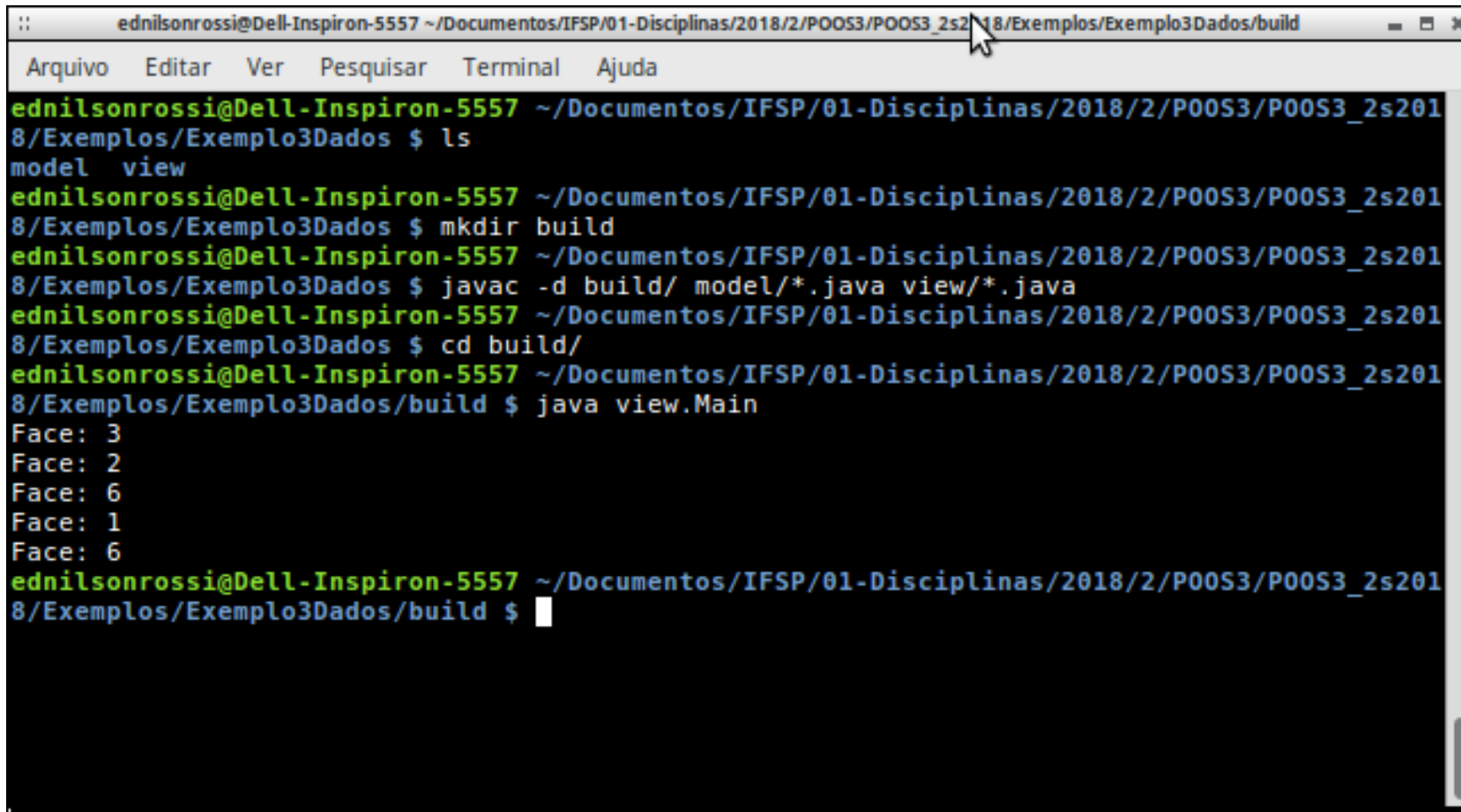
```
import model.Dado;
```

```
public class Main{  
  
    public static void main(String args[]){  
        int i;  
        Dado dado;  
  
        dado = new Dado();  
  
        for(i=0; i<5; i++){  
            System.out.printf("Face: %d\n", dado.getFace());  
            dado.jogar();  
        }  
    }  
}
```



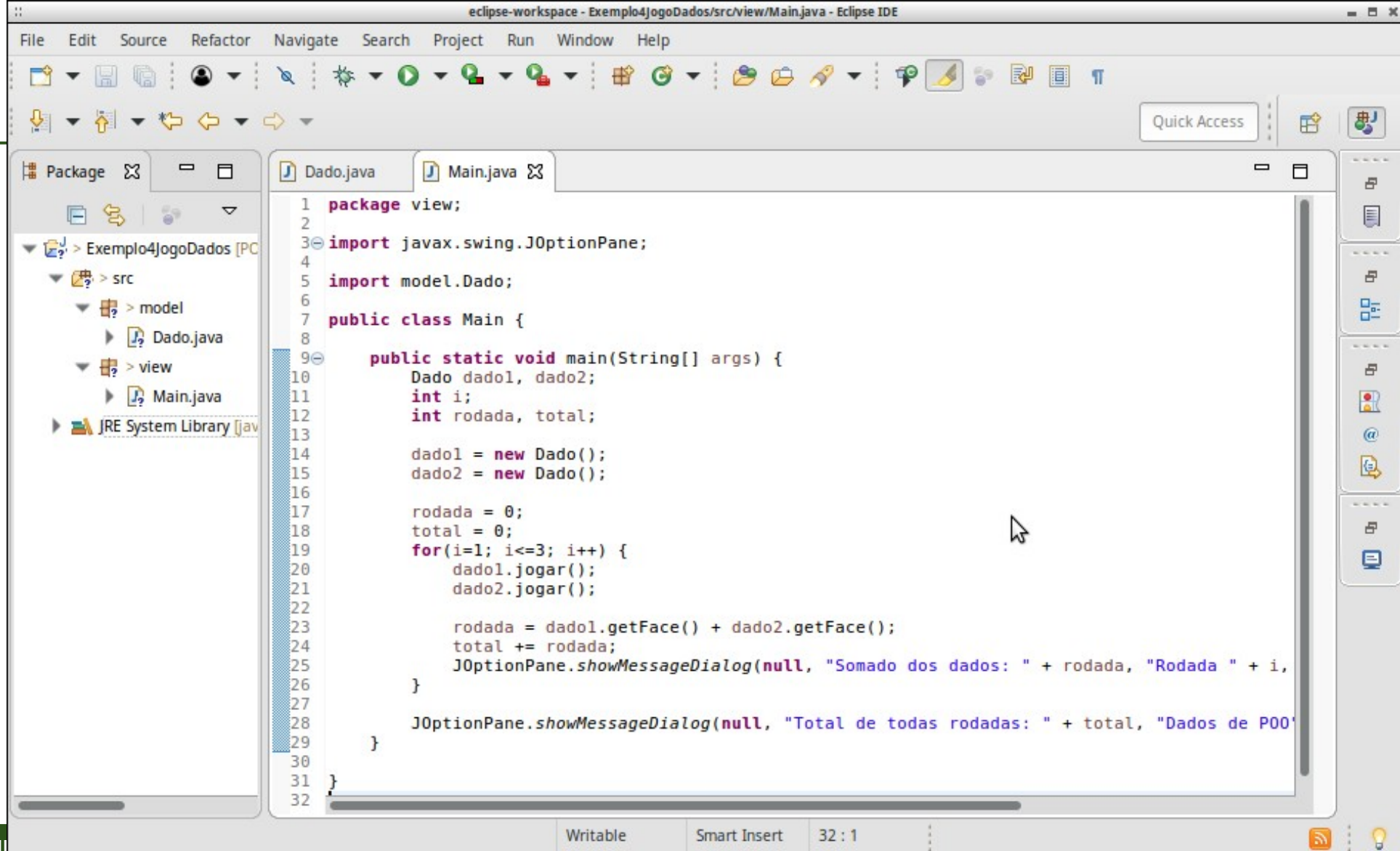
Como compilar?

# Organização de pacotes

A screenshot of a terminal window on a Linux system. The window title is "ednilsonrossi@Dell-Inspiron-5557 ~/Documentos/IFSP/01-Disciplinas/2018/2/POOS3/POOS3\_2s2018/Exemplos/Exemplo3Dados/build". The menu bar includes "Arquivo", "Editar", "Ver", "Pesquisar", "Terminal", and "Ajuda". The terminal shows the following commands and output:

```
ednilsonrossi@Dell-Inspiron-5557 ~/Documentos/IFSP/01-Disciplinas/2018/2/POOS3/POOS3_2s2018/Exemplos/Exemplo3Dados $ ls
model  view
ednilsonrossi@Dell-Inspiron-5557 ~/Documentos/IFSP/01-Disciplinas/2018/2/POOS3/POOS3_2s2018/Exemplos/Exemplo3Dados $ mkdir build
ednilsonrossi@Dell-Inspiron-5557 ~/Documentos/IFSP/01-Disciplinas/2018/2/POOS3/POOS3_2s2018/Exemplos/Exemplo3Dados $ javac -d build/ model/*.java view/*.java
ednilsonrossi@Dell-Inspiron-5557 ~/Documentos/IFSP/01-Disciplinas/2018/2/POOS3/POOS3_2s2018/Exemplos/Exemplo3Dados $ cd build/
ednilsonrossi@Dell-Inspiron-5557 ~/Documentos/IFSP/01-Disciplinas/2018/2/POOS3/POOS3_2s2018/Exemplos/Exemplo3Dados/build $ java view.Main
Face: 3
Face: 2
Face: 6
Face: 1
Face: 6
ednilsonrossi@Dell-Inspiron-5557 ~/Documentos/IFSP/01-Disciplinas/2018/2/POOS3/POOS3_2s2018/Exemplos/Exemplo3Dados/build $
```

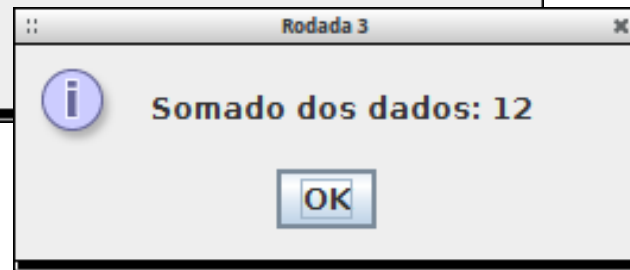
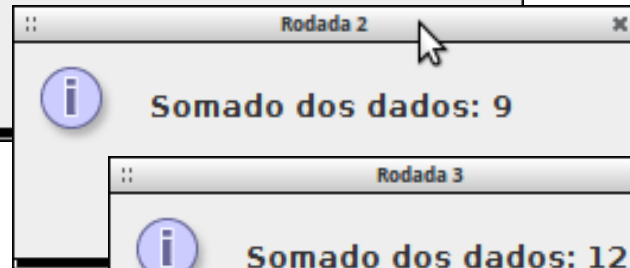
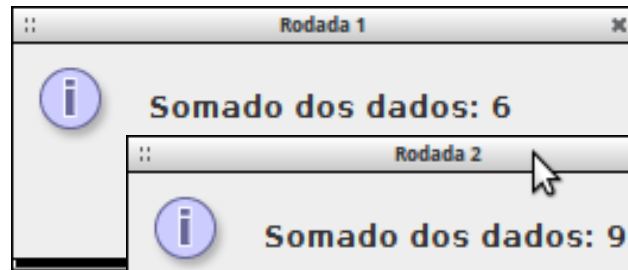
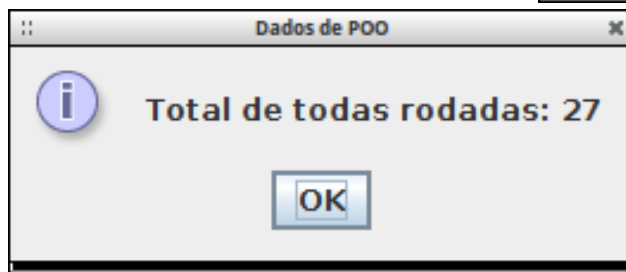






# Jogo de dois dados

```
package view;
import javax.swing.JOptionPane;
import model.Dado;
public class Main {
    public static void main(String[] args) {
        Dado dado1, dado2;
        int i;
        int rodada, total;
        dado1 = new Dado();
        dado2 = new Dado();
        rodada = 0;
        total = 0;
        for(i=1; i<=3; i++) {
            dado1.jogar();
            dado2.jogar();
            rodada = dado1.getFace() + dado2.getFace();
            total += rodada;
            JOptionPane.showMessageDialog(null, "Somado dos dados: " + rodada, "Rodada "
                + i, JOptionPane.INFORMATION_MESSAGE);
        }
        JOptionPane.showMessageDialog(null, "Total de todas rodadas: " + total,
            "Dados de P00", JOptionPane.INFORMATION_MESSAGE);
    }
}
```



# Material adicional

---

- **Apostila Caelum:**
  - <https://www.caelum.com.br/apostila-java-orientacao-objetos/pacotes-organizando-suas-classes-e-bibliotecas/#import>
- **Videoaula**
  - <https://www.youtube.com/watch?v=aRQHjfYBpM8>
  - <https://www.youtube.com/watch?v=jlfPvg2s-dg>
  - <https://www.youtube.com/watch?v=u1Nd4UIGJel>
- **Entrada e saída por JOptionPane**
  - <https://pt.scribd.com/document/32741678/Entrada-e-Saida-de-dados-por-JOptionPane>

# Trabalhando

---

- Exercício de Fixação
  - 1



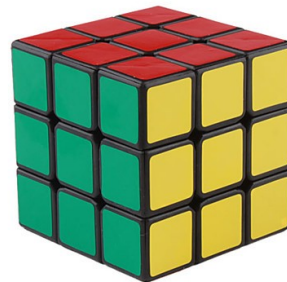
---

# Membros estáticos

# Problema

---

- Vamos implementar um programa que permita inserir o nome de alunos em um objeto.



```
package model;

public class Estudante {
    private String nome;

    public Estudante(String nome) {
        this.nome = nome;
    }

    public String getNome() {
        return nome;
    }

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

```
package view;

import model.Estudante;

public class Main {

    public static void main(String[] args) {
        Estudante pio;
        Estudante gustavo;

        pio = new Estudante("Gustavo Pio");
        gustavo = new Estudante("Gustavo F.");

        System.out.println(pio.getNome());
        System.out.println(gustavo.getNome());
    }
}
```

## Memória

**pio**

pio.nome = "Gustavo Pio"

**gustavo**

gustavo.nome = "Gustavo F."

Cada um dos objetos possui seus atributos e métodos.

Invocar `pio.getNome()` executa um método enquanto `gustavo.getNome()` executa outro método, ou seja, cada instância é independente da outra. Dados e métodos não são compartilhados.



```
package view;

import model.Estudante;

public class Main {

    public static void main(String[] args) {
        Estudante pio;
        Estudante gustavo;

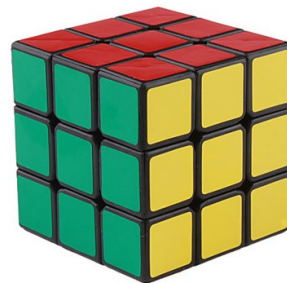
        pio = new Estudante("Gustavo Pio");
        gustavo = new Estudante("Gustavo F.");

        System.out.println(pio.getNome());
        System.out.println(gustavo.getNome());
    }
}
```

# Problema

---

- Vamos implementar um programa que permita inserir o nome de alunos em um objeto.
- Deve-se saber quantos alunos foram instanciados. Detalhe, a quantidade de objetos instanciados faz parte da lógica de negócio, como resolver?





```
package model;

public class Aluno {
    private static int quantidade = 0;
    private String nome;

    public Aluno(String nome) {
        this.nome = nome;
        Aluno.quantidade += 1;
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public static int getQuantidadeInstancias() {
        return quantidade;
    }
}
```

```
package view;

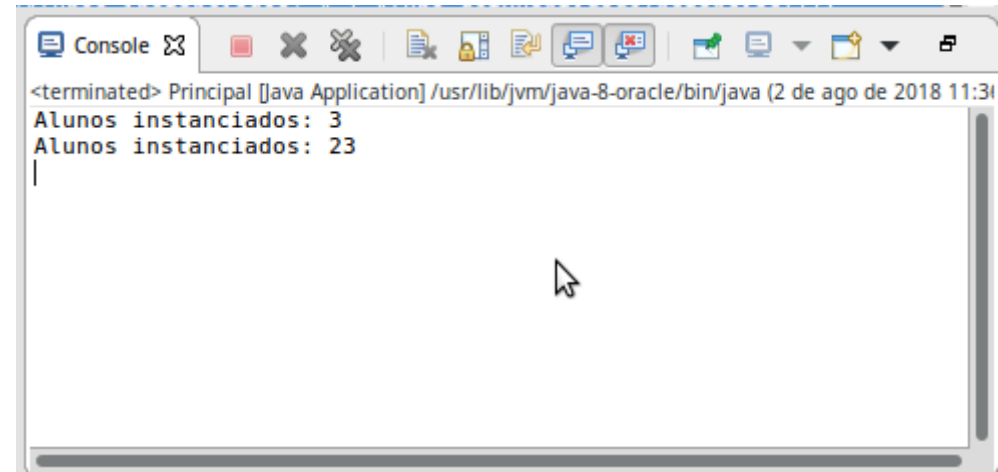
import model.Aluno;

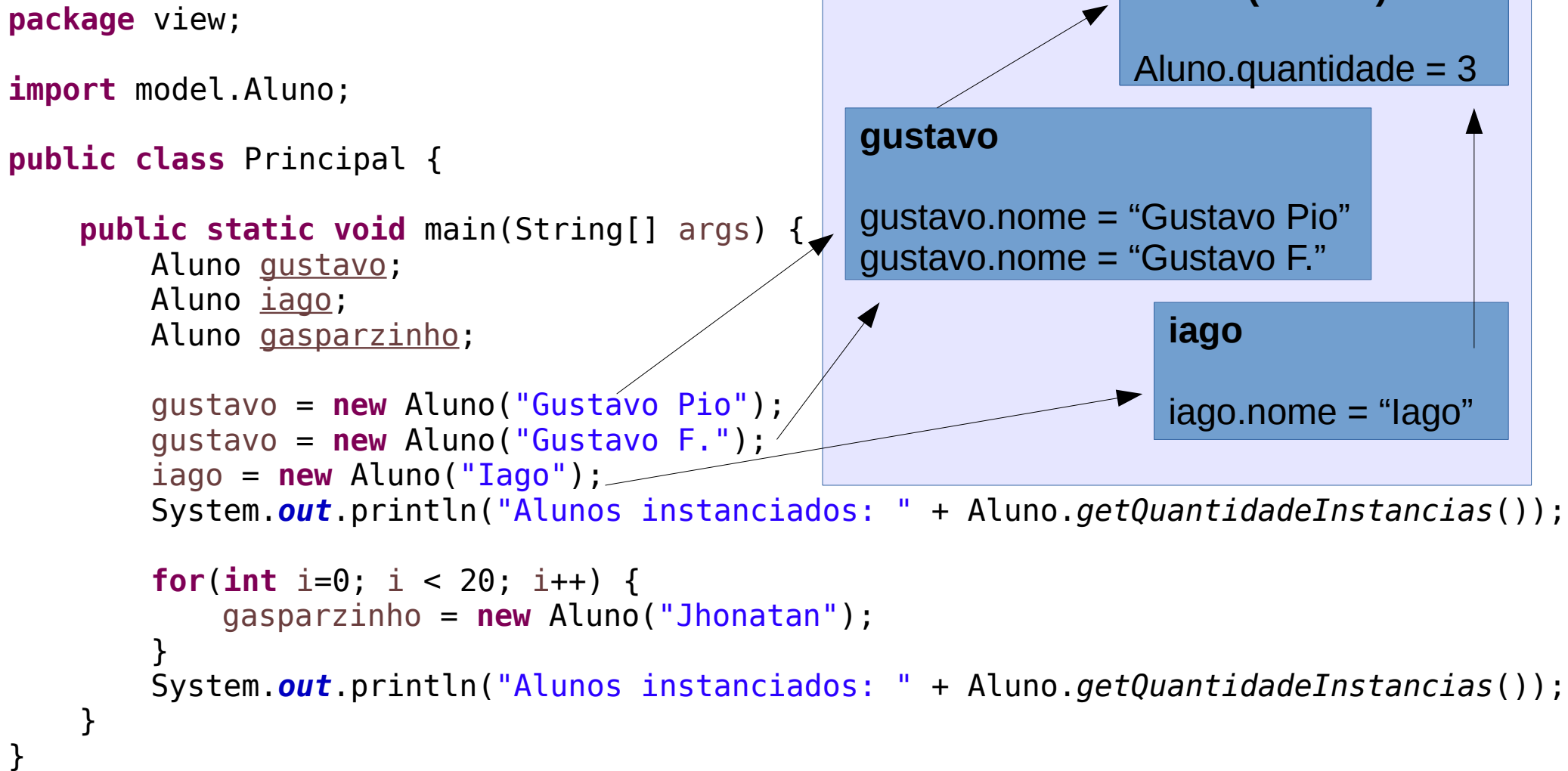
public class Principal {

    public static void main(String[] args) {
        Aluno gustavo;
        Aluno iago;
        Aluno gasparzinho;

        gustavo = new Aluno("Gustavo Pio");
        gustavo = new Aluno("Gustavo F.");
        iago = new Aluno("Iago");
        System.out.println("Alunos instanciados: " + Aluno.getQuantidadeInstancias());

        for(int i=0; i < 20; i++) {
            gasparzinho = new Aluno("Jhonatan");
        }
        System.out.println("Alunos instanciados: " + Aluno.getQuantidadeInstancias());
    }
}
```





# Membros estáticos

- Java permite a criação de membros de classe que são comuns efetivamente a todos os objetos da classe.
- Só existe uma cópia de um atributo **estático** que pode ser usada por todos objetos da classe (variável global)

Cuidado com o uso de membros estáticos.

Variáveis globais continuam desencorajadas.



# Quando usar membros estáticos?

- Um bom uso dos membros estáticos é a declaração de constantes.
- Ao criar a classe Calculadora, definiu-se as operações como constantes estáticas, isso permite que todo o sistema utilize essas constantes.

```
public class Calculadora {  
    public static final char OPERADOR_SOMA = '+';  
    public static final char OPERADOR_SUBTRACAO = '-';  
    public static final char OPERADOR_MULTIPLICACAO = '*';  
    public static final char OPERADOR_DIVISAO = '/';  
  
    private int memory;  
    public int calcular(int operando, char operador){  
        switch (operador){  
            case OPERADOR_SOMA:  
                memory = soma(operando);  
                break;  
            case OPERADOR_SUBTRACAO:  
                memory = subtrai(operando);  
                break;  
            case OPERADOR_MULTIPLICACAO:  
                memory = multiplica(operando);  
                break;  
            case OPERADOR_DIVISAO:  
                memory = divisao(operando);  
        }  
        return memory;  
    } ... }  
}
```

# () - Constantes

Volte aos membros estáticos!



Constantes são declaradas com o operador **final** antes do tipo de dado. Isso indica que o valor daquela variável só é atribuído na declaração.

Também pode-se implementar métodos com o operador final. Isso define que o método não poderá ser rescrito. Assunto para um breve futuro.

Constantes são variáveis que não variam!

```
public class Calculadora {  
    public static final char OPERADOR_SOMA = '+';  
    public static final char OPERADOR_SUBTRACAO = '-';  
    public static final char OPERADOR_MULTIPLICACAO = '*';  
    public static final char OPERADOR_DIVISAO = '/';  
}
```



# Quando usar membros estáticos?

- Métodos acessados sem a instância do objeto.

```
public static Aluno lerAluno(){
    Aluno estudante;
    String nome;
    int prontuario;
    double n1, n2, n3, n4;
    nome = JOptionPane.showInputDialog("Nome do aluno: ");
    prontuario = Integer.parseInt(JOptionPane.showInputDialog("Prontuário: "));
    n1 = Double.parseDouble(JOptionPane.showInputDialog("Prova 1:"));
    n2 = Double.parseDouble(JOptionPane.showInputDialog("Prova 2:"));
    n3 = Double.parseDouble(JOptionPane.showInputDialog("Exercícios:"));
    n4 = Double.parseDouble(JOptionPane.showInputDialog("Projeto:"));
    estudante = new Aluno(nome, prontuario, n1, n2, n3, n4);
    return estudante;
}
```

Quais métodos estáticos foram invocados e implementados nesse método?



# Material adicional

- **Leitura Obrigatória**

- Winder, R.; Roberts, G. Desenvolvendo Software em Java. 3 ed. Rio de Janeiro: LTC, 2009.
  - Capítulo 6 completo.
  - **Capítulo 7.3**
- Programação orientada a objetos / organizador Rafael Félix. - São Paulo: Pearson, 2016
  - Unidade 2 – Herança e Polimorfismo

Laboratório da próxima aula depende da leitura deste capítulo.

- **Videoaula**

- [https://www.youtube.com/watch?v=Dz\\_w4YpFL80](https://www.youtube.com/watch?v=Dz_w4YpFL80)
- <https://www.youtube.com/watch?v=UMkftZ2hKxQ>
- <https://www.youtube.com/watch?v=-dQ0yDTHcS0>





# Trabalhando

---

- Exercício de Fixação
  - 2
- Exercício Avaliativo
  - 5

