

# Programação Orientada a Objetos – Aula 05

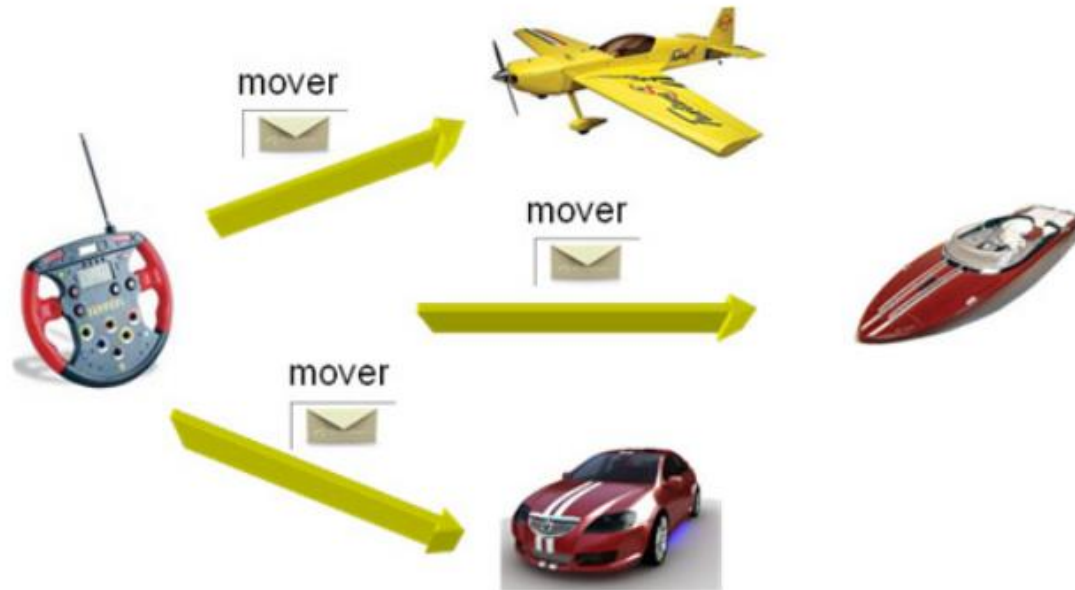
Prof. Dr. Eduardo Takeo Ueda  
*[eduardo.tueda@sp.senac.br](mailto:eduardo.tueda@sp.senac.br)*

# Polimorfismo

- **Polimorfismo** (“muitas formas”) é a habilidade de objetos de classes diferentes responderem a mesma mensagem de diferentes maneiras
- Em outras palavras é a capacidade de um objeto decidir qual método aplicar em si mesmo
- Mecanismo que permite **reutilização**

# Polimorfismo

- Controle remoto “universal”



- O Polimorfismo permite que diferentes objetos (avião, barco, automóvel) respondam uma mesma mensagem (mover) de formas diferentes (voar, navegar e correr)

# Tipos de Polimorfismo

- *Polimorfismo de Sobrecarga (Overloading)*
  - Também chamado Polimorfismo Estático
- *Polimorfismo de Sobreposição (Overriding)*
  - Também chamado Polimorfismo Dinâmico

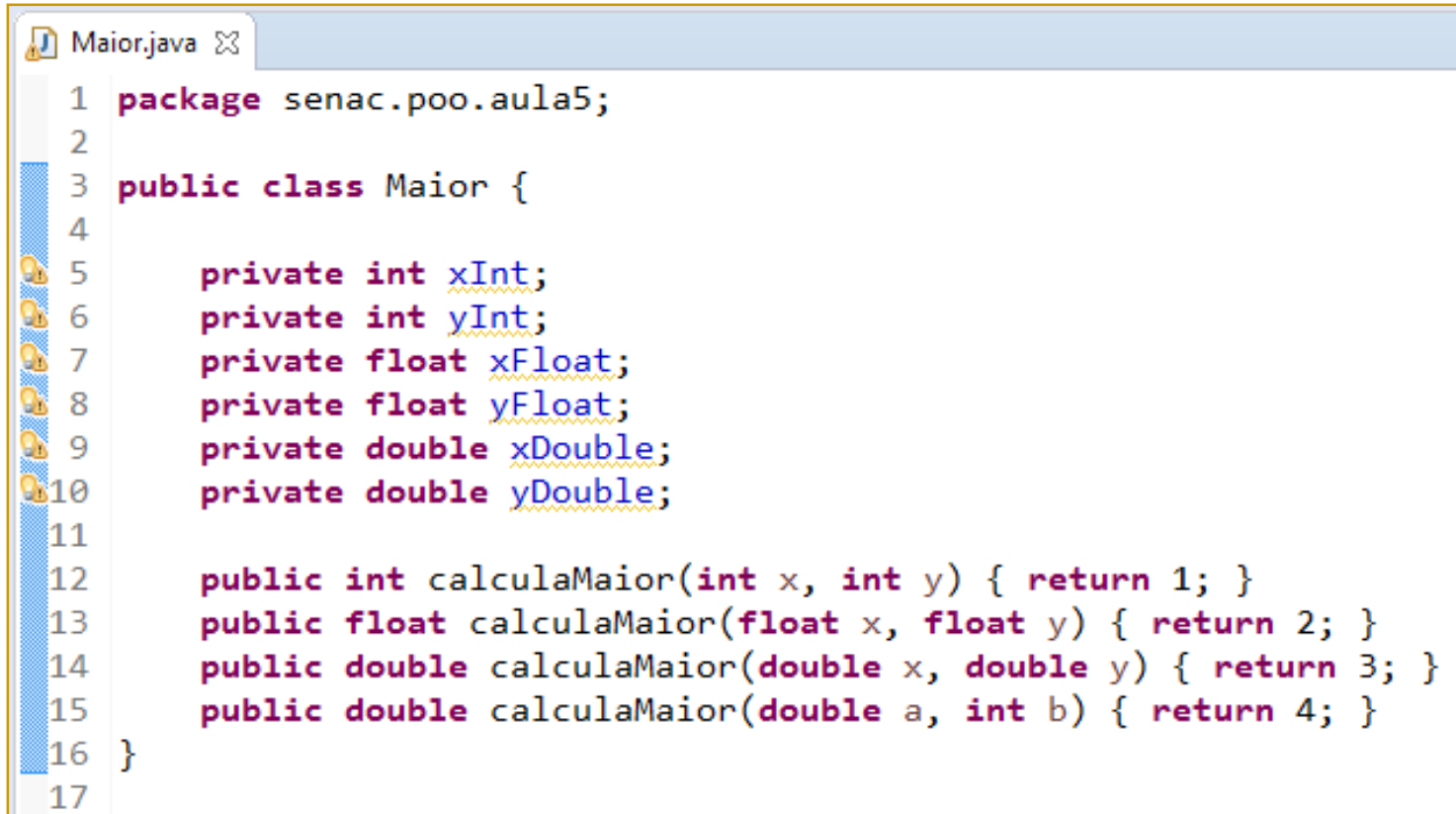
# Polimorfismo de Sobrecarga

- Polimorfismo de sobrecarga permite que um método de determinado nome tenha **comportamentos distintos**, em função de diferentes parâmetros que ele recebe
- Cada método difere apenas no **número** e no **tipo** de parâmetros

# Polimorfismo de Sobrecarga

```
Maior.java
1 package senac.poo.aula5;
2
3 public class Maior {
4
5     private int xInt;
6     private int yInt;
7     private float xFloat;
8     private float yFloat;
9     private double xDouble;
10    private double yDouble;
11
12    public int calculaMaior(int x, int y) { return 1; }
13    public float calculaMaior(float x, float y) { return 2; }
14    public double calculaMaior(double x, double y) { return 3; } // erro
15    public double calculaMaior(double a, double b) { return 4; } // erro
16 }
17
```

# Polimorfismo de Sobrecarga



```
1 package senac.poo.aula5;
2
3 public class Maior {
4
5     private int xInt;
6     private int yInt;
7     private float xFloat;
8     private float yFloat;
9     private double xDouble;
10    private double yDouble;
11
12    public int calculaMaior(int x, int y) { return 1; }
13    public float calculaMaior(float x, float y) { return 2; }
14    public double calculaMaior(double x, double y) { return 3; }
15    public double calculaMaior(double a, int b) { return 4; }
16 }
17
```

# Sobrecarga e Construtores

- Polimorfismo de sobrecarga é amplamente aplicado sobre os métodos construtores de classes
- É comum para uma classe ter várias maneiras de instanciar objetos



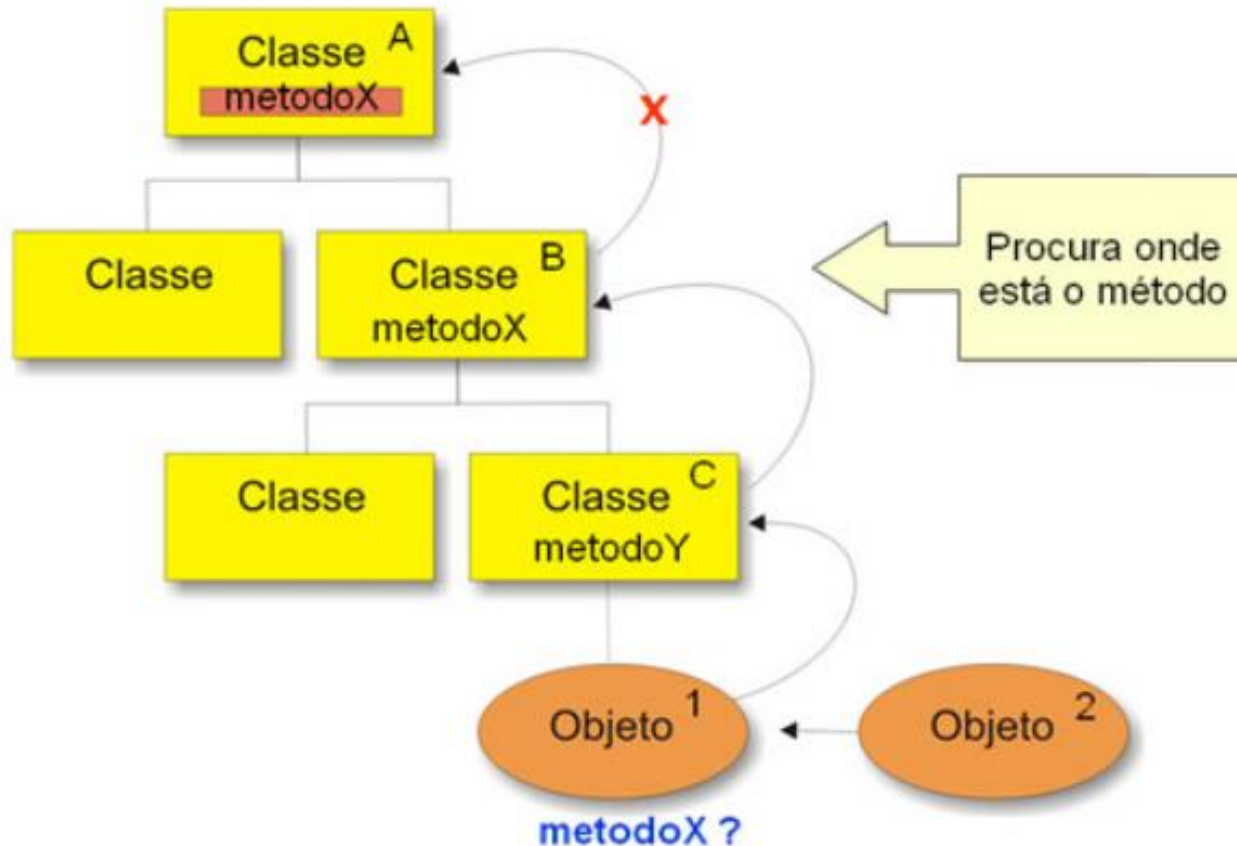
# Sobrecarga e Construtores

```
Pessoa.java ✕
1 package senac.poo.aula5;
2
3 public class Pessoa {
4
5     private String nome;
6     private int rg;
7
8     Pessoa() {
9         //construtor default
10    }
11
12    Pessoa(String nome) {
13        this.nome = nome;
14    }
15
16    Pessoa(int rg) {
17        this.rg = rg;
18    }
19 }
20
```

# Polimorfismo de Sobreposição

- Polimorfismo de sobreposição é a **redefinição** de métodos em classes descendentes, i. é, quando trabalhamos com herança
- Um método de uma subclasse (classe filha) com o mesmo nome de um método de uma superclasse (classe pai/mãe) irá **sobrepôr** esse último

# Polimorfismo de Sobreposição



# Polimorfismo de Sobreposição

- Considere que a classe **Brinquedo** possui 3 classes descendentes diretas: **Carro**, **Avião** e **Barco**
- As subclasses (classes filhas) sobrepõem o método **mover()** da superclasse (classe pai/mãe) **Brinquedo**

# Polimorfismo de Sobreposição

```
Brinquedo.java
1 package senac.poo.aula5;
2
3 public class Brinquedo {
4     //...
5     public void mover() {
6         System.out.println("Mover brinquedo");
7     }
8     //...
9 }
10
```

```
Carro.java
1 package senac.poo.aula5;
2
3 public class Carro extends Brinquedo {
4     //...
5     public void mover() {
6         System.out.println("Correr");
7     }
8     //...
9 }
10
```

# Polimorfismo de Sobreposição

```
Aviao.java ✕
1 package senac.poo.aula5;
2
3 public class Aviao extends Brinquedo {
4     //...
5     public void mover() {
6         System.out.println("Voar");
7     }
8     //...
9 }
10
```

```
Barco.java ✕
1 package senac.poo.aula5;
2
3 public class Barco extends Brinquedo {
4     //...
5     public void mover() {
6         System.out.println("Navegar");
7     }
8     //...
9 }
10
```

# Polimorfismo de Sobreposição

```
ControleRemoto.java ✕
1 package senac.poo.aula5;
2
3 public class ControleRemoto {
4
5     // declaração de referência
6     private Brinquedo brinquedo;
7
8     // construtor
9     public ControleRemoto(Brinquedo b) {
10         brinquedo = b;
11     }
12
13     // método
14     public void mover() {
15         brinquedo.mover();
16     }
17 }
18
```

# Polimorfismo de Sobreposição

```
Programa.java X
1 package senac.poo.aula5;
2
3 public class Programa {
4
5     public static void main(String[] args) {
6
7         Aviao aviao = new Aviao();
8         ControleRemoto controle1 = new ControleRemoto(aviao);
9         controle1.mover();
10
11         Carro carro = new Carro();
12         ControleRemoto controle2 = new ControleRemoto(carro);
13         controle2.mover();
14
15         Barco barco = new Barco();
16         ControleRemoto controle3 = new ControleRemoto(barco);
17         controle3.mover();
18     }
19 }
20
```

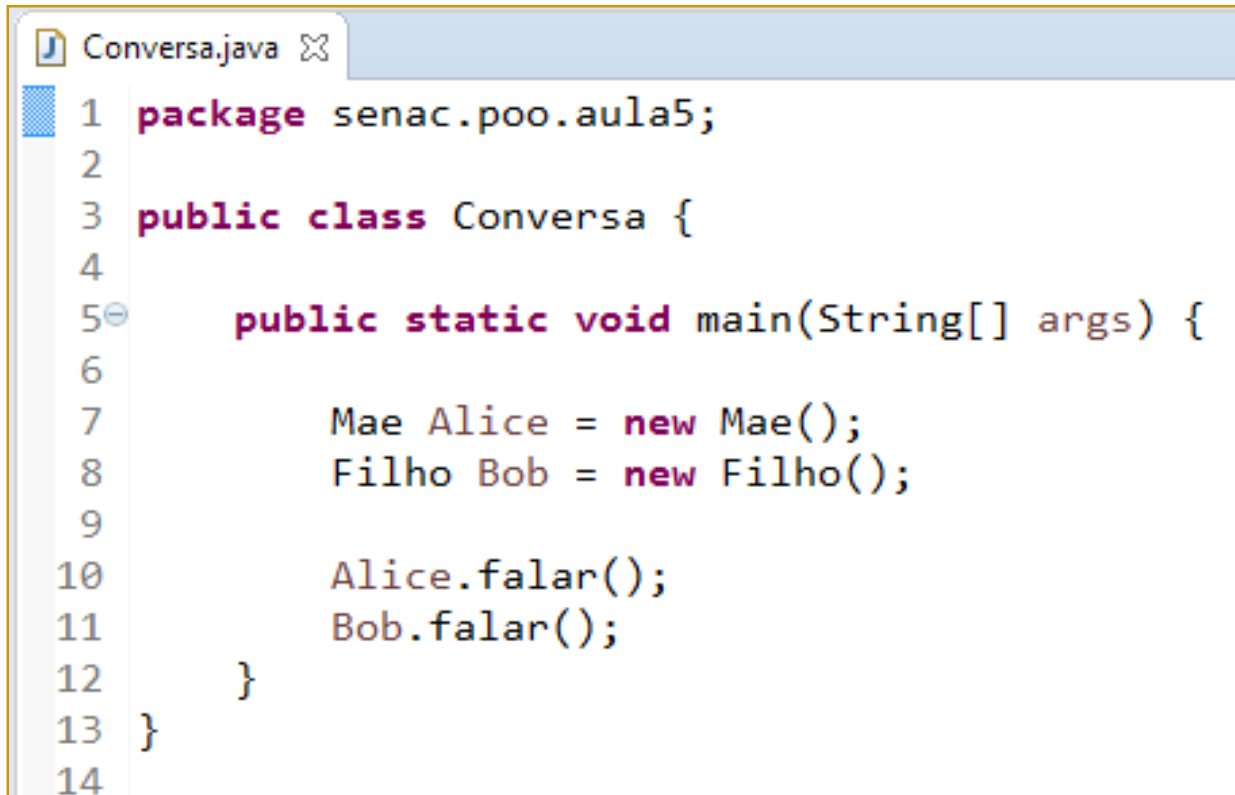


# Chamando métodos da superclasse

```
Mae.java ✕
1 package senac.poo.aula5;
2
3 public class Mae {
4
5     public void falar() {
6
7         System.out.println("Mãe falando...");
8     }
9 }
10
```

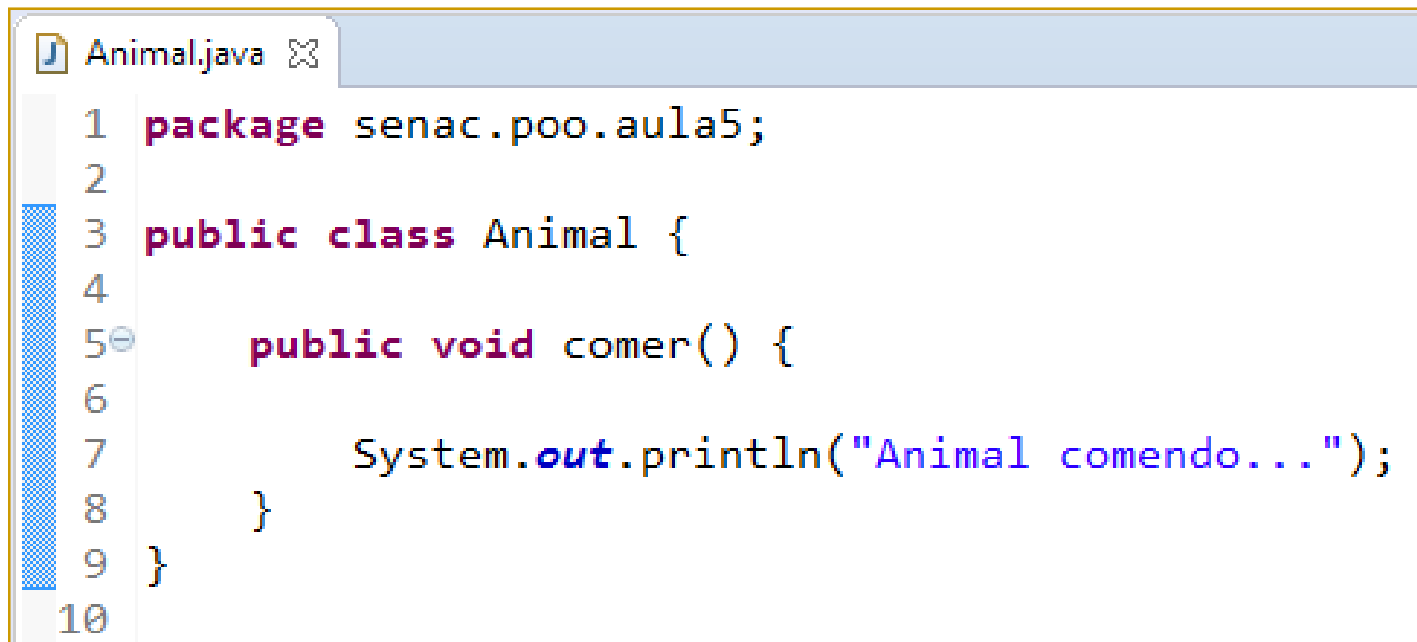
```
Filho.java ✕
1 package senac.poo.aula5;
2
3 public class Filho extends Mae {
4
5     public void falar() {
6
7         super.falar(); // invoca método da superclasse
8         System.out.println("Filho falando...");
9     }
10 }
11
```

# Chamando métodos da superclasse



```
Conversa.java X
1 package senac.poo.aula5;
2
3 public class Conversa {
4
5     public static void main(String[] args) {
6
7         Mae Alice = new Mae();
8         Filho Bob = new Filho();
9
10        Alice.falar();
11        Bob.falar();
12    }
13 }
14
```

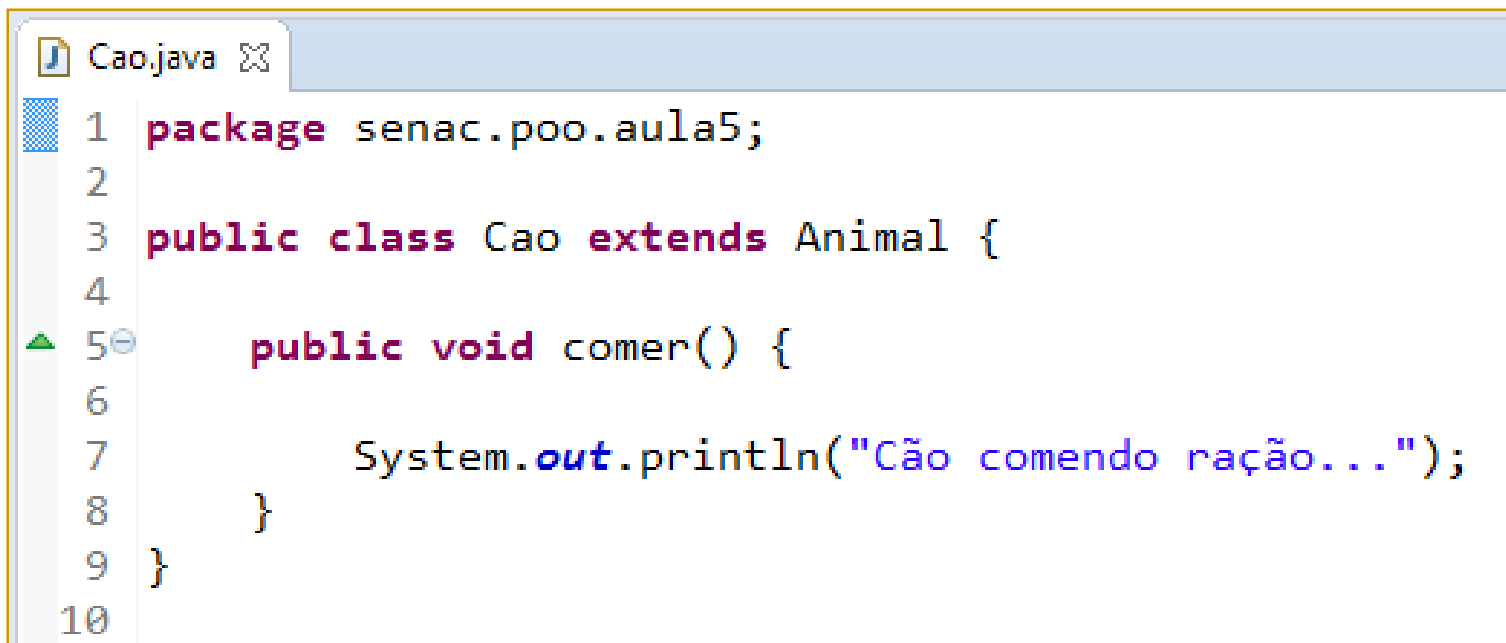
# Mais um Exemplo de Polimorfismo



The screenshot shows a code editor window titled "Animal.java". The code is as follows:

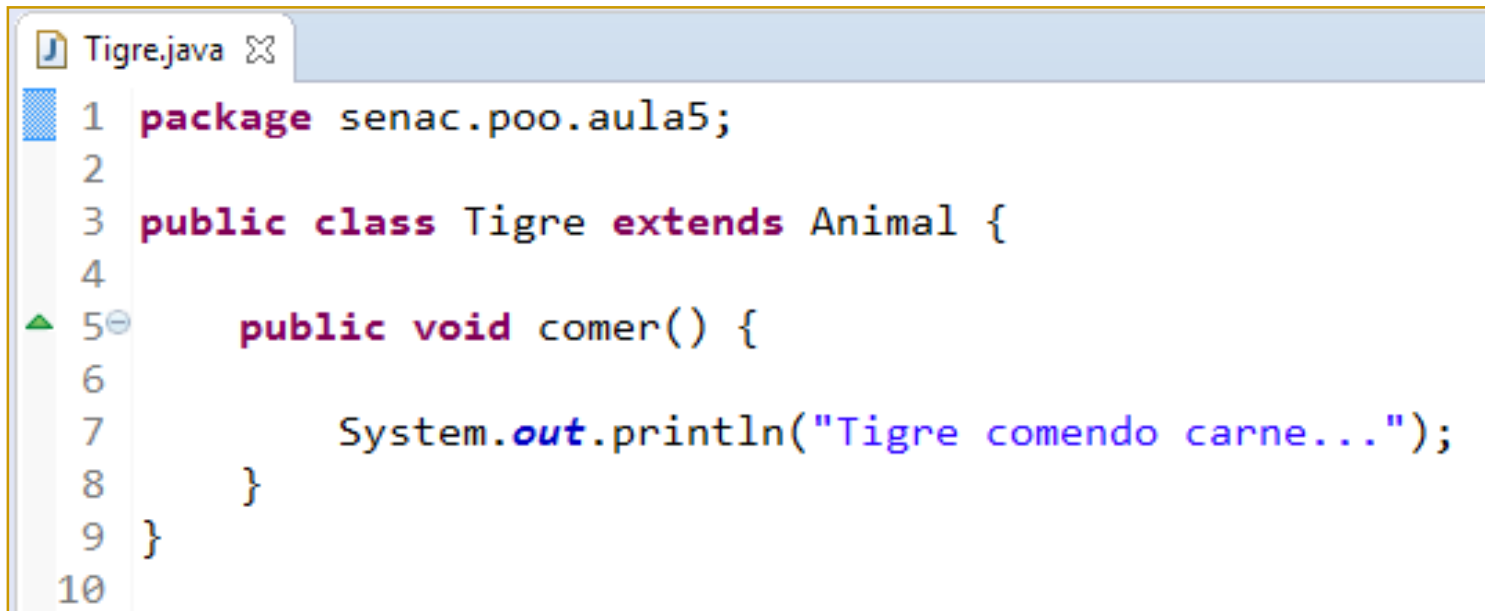
```
1 package senac.poo.aula5;
2
3 public class Animal {
4
5     public void comer() {
6
7         System.out.println("Animal comendo...");
8     }
9 }
10
```

# Mais um Exemplo de Polimorfismo

A screenshot of a Java IDE window titled 'Cao.java'. The code defines a package 'senac.poo.aula5' and a public class 'Cao' that extends 'Animal'. The 'Cao' class has a public void method 'comer()' which prints 'Cão comendo ração...' to the console. The code is as follows:

```
1 package senac.poo.aula5;
2
3 public class Cao extends Animal {
4
5     public void comer() {
6
7         System.out.println("Cão comendo ração...");
8     }
9 }
10
```

# Mais um Exemplo de Polimorfismo



```
Tigre.java ✕
1 package senac.poo.aula5;
2
3 public class Tigre extends Animal {
4
5     public void comer() {
6
7         System.out.println("Tigre comendo carne...");
8     }
9 }
10
```

# Mais um Exemplo de Polimorfismo

```
Refeicao.java
1 package senac.poo.aula5;
2
3 public class Refeicao {
4
5     public void fazerAnimalComer( Animal animal ) {
6
7         animal.comer();
8     }
9
10    public static void main(String[] args) {
11
12        Refeicao refeicao = new Refeicao();
13
14        refeicao.fazerAnimalComer( new Animal() );
15        refeicao.fazerAnimalComer( new Cao() );
16        refeicao.fazerAnimalComer( new Tigre() );
17    }
18 }
19
```

# Modificador final

- **Atributos:** não permite alteração do valor depois de uma declaração, mas **cuidado!**
  - `public final String nome;`
  - `public final String nome = "Takeo";`
- **Métodos:** não permite a sobreposição em subclasses
  - `public final void comer() {...`
- **Classes:** não permite aplicar herança em uma classe
  - `public final class Animal {...`

# Atributos Estáticos

- Atributo estático  $\neq$  Atributo constante (modificador final)
- A ideia por trás do conceito de atributos (e métodos) estáticos é a de que objetos de uma mesma classe podem e, em algumas situações devem, compartilhar valores em comum
- Caso contrário, teríamos que criar atributos que precisariam ser atualizados todos ao mesmo tempo, em cada objeto criado



# Atributos Estáticos

```
ReligioPonto.java
1 package senac.poo.aula5;
2
3 public class ReligioPonto {
4     public static int horas; // atributo estático
5     public static int minutos; // atributo estático
6     public static int segundos; // atributo estático
7     private int id;
8     private String nomeFilial;
9
10    public ReligioPonto(int id, String nome) {
11        this.id = id;
12        this.nomeFilial = nome;
13    }
14    public void setId(int id) {
15        this.id = id;
16    }
17    public void setNomeFilial(String nomeFilial) {
18        this.nomeFilial = nomeFilial;
19    }
20    public int getId() {
21        return id;
22    }
23    public String getNomeFilial() {
24        return nomeFilial;
25    }
26 }
```

# Atributos Estáticos

```
Expediente.java
1 package senac.poo.aula5;
2
3 public class Expediente {
4
5     public static void main(String[] args) {
6
7         RelogioPonto.horas = 07; // atributo estático da classe
8         RelogioPonto.minutos = 30; // idem
9         RelogioPonto.segundos = 00; // idem
10
11         RelogioPonto filialBSB = new RelogioPonto(1, "Brasilia");
12         RelogioPonto filialSP = new RelogioPonto(2, "São Paulo");
13         RelogioPonto filialRJ = new RelogioPonto(3, "Rio de Janeiro");
14
15         System.out.println("Filial Brasília: "+filialBSB.horas+": "+
16                             filialBSB.minutos+": "+filialBSB.segundos);
17         System.out.println("Filial São Paulo: "+filialSP.horas+": "+
18                             filialSP.minutos+": "+filialSP.segundos);
19         System.out.println("Filial Rio de Janeiro: "+filialRJ.horas+": "+
20                             filialRJ.minutos+": "+filialRJ.segundos);
21     }
22 }
23
```

# Atributos Estáticos

- Observe que não é necessário criar um objeto da classe RelogioPonto para que se possa acessar seus atributos estáticos

```
5 public static void main(String[] args) {  
6  
7     RelogioPonto.horas = 07; // atributo estático da classe  
8     RelogioPonto.minutos = 30; // idem  
9     RelogioPonto.segundos = 00; // idem  
10 }
```

- Atributo estático = Atributo comum (ou compartilhado) entre objetos de uma classe

# Métodos Estáticos

- A mesma ideia dos atributos estáticos pode ser ampliada e aplicada para os métodos, pois, se quisermos alterar o modo de acesso dos atributos hora, minuto e segundo para private, por exemplo, teremos que criar métodos que sejam capazes de alterá-los
- Não é uma boa prática modificar valores de atributos estáticos através de referências para seus objetos
- Os métodos estáticos surgiram basicamente para operarem sobre os atributos estáticos, ou que não realizam operação alguma sobre os atributos dos objetos

# Métodos Estáticos

```
ReligioPonto.java
9
10 public ReligioPonto(int id, String nome) {
11     this.id = id;
12     this.nomeFilial = nome;
13 }
14 public static void setHoras(int horas) {
15     ReligioPonto.horas = horas;
16 }
17 public static int getHoras() {
18     return horas;
19 }
20 public static void setMinutos(int minutos) {
21     ReligioPonto.minutos = minutos;
22 }
23 public static int getMinutos() {
24     return minutos;
25 }
26 public static void setSegundos(int segundos) {
27     ReligioPonto.segundos = segundos;
28 }
29 public static int getSegundos() {
30     return segundos;
31 }
32 }
33
```

# Atributos Estáticos

- Assim como no caso de atributos, também não é necessário criar o objeto da classe para que se possa usar o método estático

```
5 public static void main(String[] args) {  
6  
7     RelogioPonto.setHoras(07); // invocação de método estático  
8     RelogioPonto.setMinutos(30); // idem  
9     RelogioPonto.setSegundos(00); // idem  
10  
11     RelogioPonto filialBSB = new RelogioPonto(1, "Brasilia");  
12     RelogioPonto filialSP = new RelogioPonto(2, "São Paulo");  
13     RelogioPonto filialRJ = new RelogioPonto(3, "Rio de Janeiro");  
14  
15     System.out.println("Filial Brasília: "+filialBSB.getHoras()+":"+  
16         filialBSB.getMinutos()+":"+filialBSB.getSegundos());  
17     System.out.println("Filial São Paulo: "+filialSP.getHoras()+":"+  
18         filialSP.getMinutos()+":"+filialSP.getSegundos());  
19     System.out.println("Filial Rio de Janeiro: "+filialRJ.getHoras()+":"+  
20         filialRJ.getMinutos()+":"+filialRJ.getSegundos());  
21 }
```