Programação Orientada a Objetos em

Flávio Miguel Varejão

Departamento de Informática

UFES

Polimorfismo

- Mesmo código aplicado a variáveis (objetos) de diferentes tipos (classes)
- Uso de Ampliação ou Upcast
  - Referência a superclasse passa a denotar objeto de subclasse

Ampliação ou Upcast

```
class Instrumento {
  public void tocar() {
    System.out.println("Instrumento.tocar()");
  }
}
class Sopro extends Instrumento {
  public void tocar() {
    System.out.println("Sopro.tocar()");
  }
}
```

Ampliação ou Upcast

```
public class Musica {
  public static void melodia(Instrumento i) {
    i.tocar();
  }
  public static void main(String[] args) {
    Sopro flauta = new Sopro();
    melodia(flauta); // Upcasting
  }
}
```

Sobrecarga x Upcast

```
class Instrumento {
  public void tocar() {
    System.out.println("Instrumento.tocar()");
  }
}
class Sopro extends Instrumento {
  public void tocar() {
    System.out.println("Sopro.tocar()");
  }
}
```

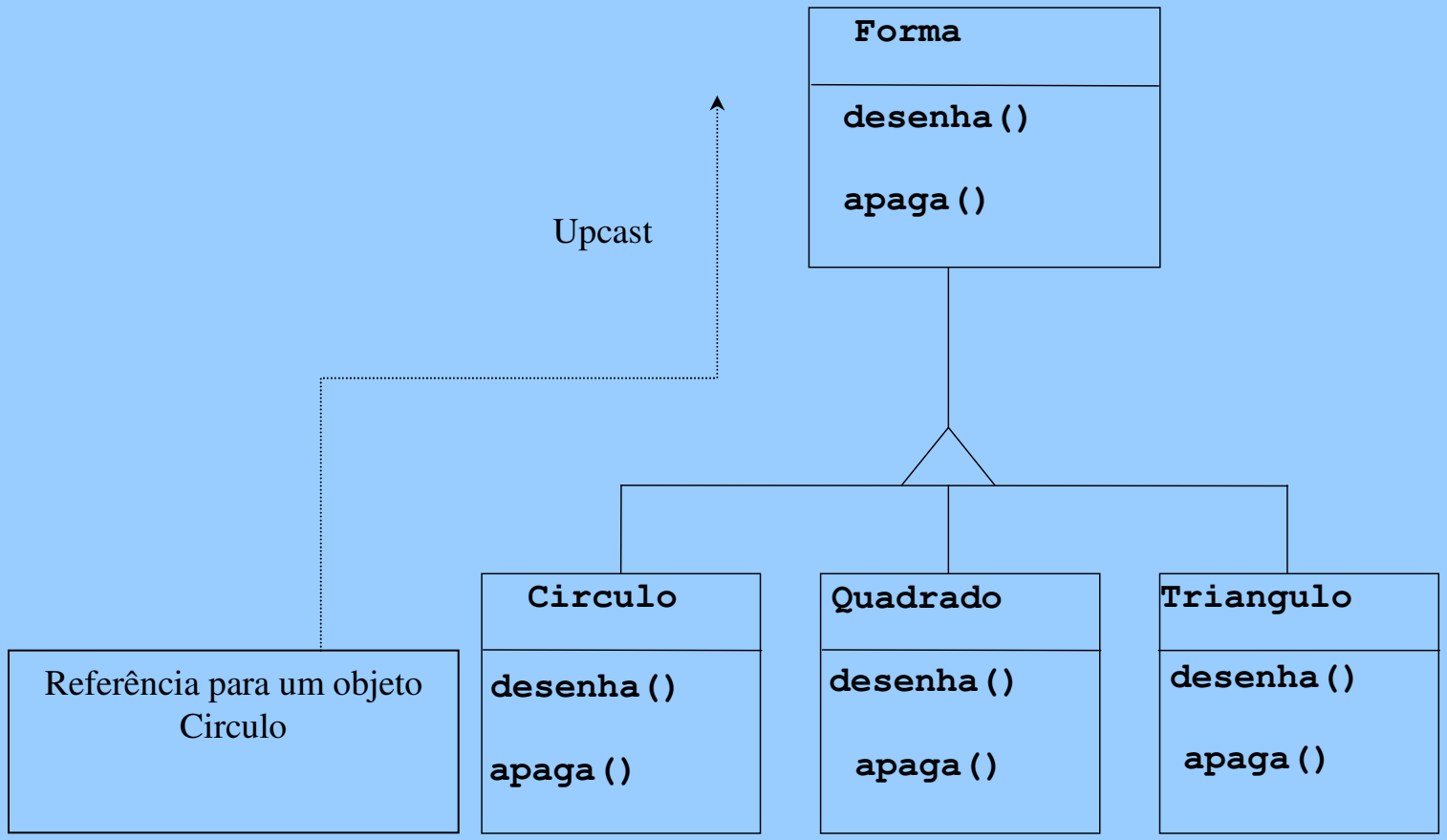
Sobrecarga x Upcast

```
class Corda extends Instrumento {
  public void tocar() {
    System.out.println("Corda.tocar()");
  }
}
class Percussão extends Instrumento {
  public void tocar() {
    System.out.println("Percussão.tocar()");
  }
}
```

Sobrecarga x Upcast

```
public class Musica1 {
  public static void melodia(Sopro i) {
    i.tocar(); }
  public static void melodia(Corda i) {
    i.tocar(); }
  public static void melodia(Percussão i) {
    i.tocar(); }
  public static void main(String[] args) {
    Sopro saxofone = new Sopro();
    Corda guitarra = new Corda();
    Percussão bateria = new Percussão();
    melodia(saxofone); // Não realiza upcasting
    melodia(guitarra);
    melodia(bateria);
  }
}
```

Amarração Tardia



Polimorfismo

```
class Forma {
  void desenha() {}
  void apaga() {}
}
class Circulo extends Forma {
  void desenha() {
    System.out.println("Circulo.desenha()");
  }
  void apaga() {
    System.out.println("Circulo.apaga()");
  }
}
```

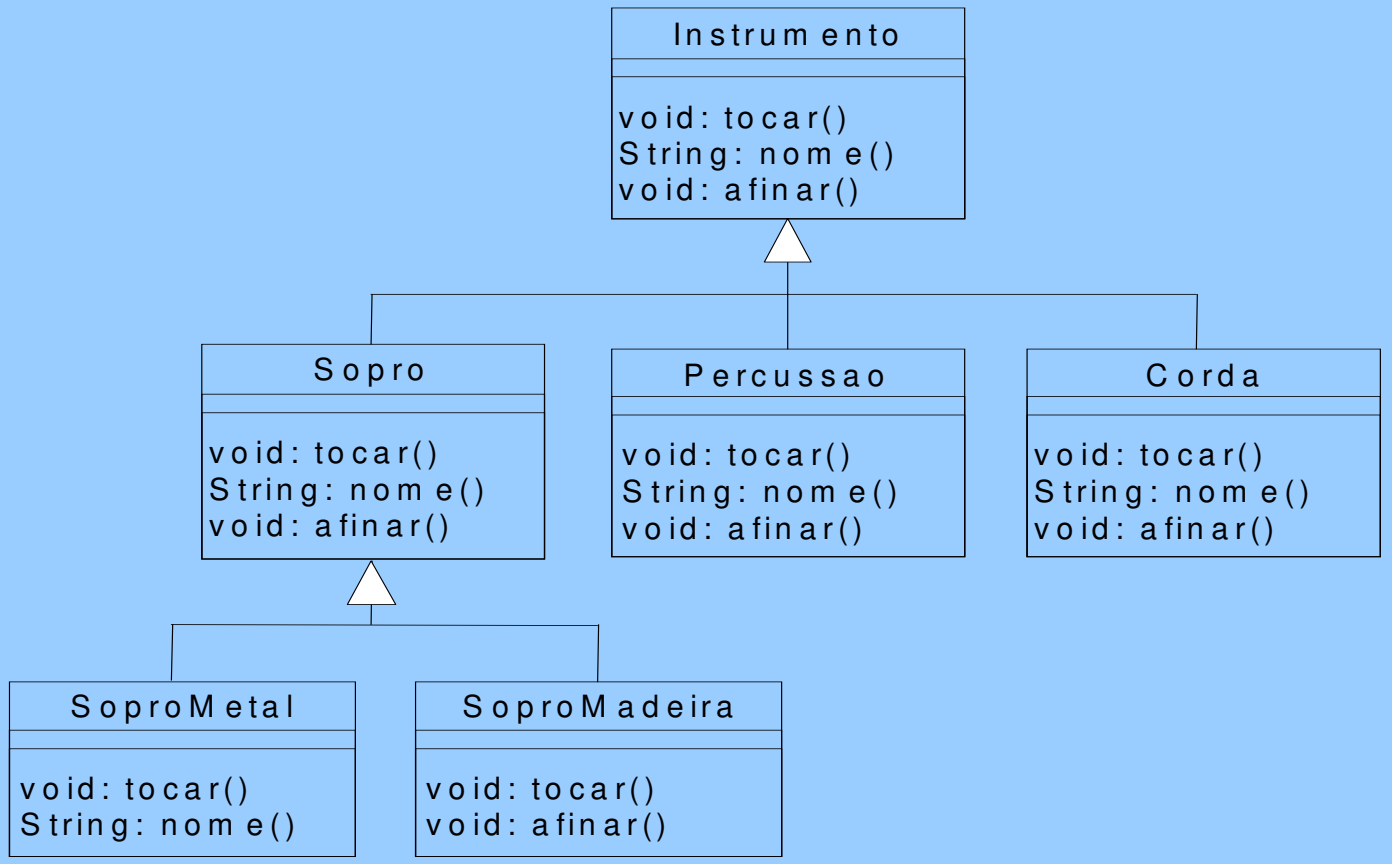
Polimorfismo

```
class Quadrado extends Forma {
  void desenha() {
    System.out.println("Quadrado.desenha()");
  }
  void apaga() {
    System.out.println("Quadrado.apaga()");
  }
}
class Triangulo extends Forma {
  void desenha() {
    System.out.println("Triangulo.desenha()");
  }
  void apaga() {
    System.out.println("Triangulo.apaga()");
  }
}
```

Polimorfismo

```
public class Formas {
  public static Forma randForma() {
    switch((int)(Math.random() * 3)) {
      default: // para que o compilador aceite
      case 0: return new Circulo();
      case 1: return new Quadrado();
      case 2: return new Triangulo();
    }
  }
  public static void main(String[] args) {
    Forma[] s = new Forma[9];
    // preenche o vetor com formas
    for(int i = 0; i < s.length; i++)
      s[i] = randForma();
    // chamadas polimorficas de metodos
    for(int i = 0; i < s.length; i++)
      s[i].desenha();
  }
}
```

Extensibilidade



Extensibilidade

```
class Instrumento {
  public void tocar() {
    System.out.println("Instrumento.tocar()");
  }
  public String nome() {
    return "Instrumento";
  }
  public void afinar() {}
}
class Sopro extends Instrumento {
  public void tocar() {
     System.out.println("Sopro.tocar()");
  }
  public String nome() { return "Sopro"; }
  public void afinar() {}
}
```

Extensibilidade

```
class Percussao extends Instrumento {
  public void tocar() {
    System.out.println("Percussao.tocar()");
  }
  public String nome() { return "Percussao"; }
  public void afinar() {}
}
class Corda extends Instrumento {
  public void tocar() {
    System.out.println("Corda.tocar()");
  }
  public String nome() { return "Corda"; }
  public void afinar() {}
}
```

Extensibilidade

```
    class SoproMetal extends Sopro {
      public void tocar() {
        System.out.println("SoproMetal.tocar()");
      }
      public void afinar() {
        System.out.println("SoproMetal.afinar()");
      }
    }
    class SoproMadeira extends Sopro {
      public void tocar() {
        System.out.println("SoproMadeira.tocar()");
      }
      public String nome() { return "SoproMadeira"; }
}
```

Extensibilidade

```
public class Musica3 {
// Também funcionará com os novos tipos adicionados!
  static void melodia(Instrumento i) {
    // ...
    i.tocar();
  }
  static void sinfonia(Instrumento[] e) {
    for(int i = 0; i < e.length; i++) melodia(e[i]);
  }
  public static void main(String[] args) {
    Instrumento[] orquestra = new Instrumento[5];
    int i = 0;
    // Upcasting:
    orquestra[i++] = new Sopro();
    orquestra[i++] = new Percussao();
    orquestra[i++] = new Corda();
    orquestra[i++] = new SoproMetal();
    orquestra[i++] = new SoproMadeira();
    sinfonia(orquestra);
  }
}
```

Sobrescrita x Sobrecarga

```
class NotaX {
   public static final int
     DO = 0, RE = 1, MI = 2;
}
class InstrumentoX {
   public void tocar(int NotaX) {
     System.out.println("InstrumentoX.tocar()");
   }
}
class SoproX extends InstrumentoX {
 // Mudança na interface do método:
  public void tocar(NotaX n) {
    System.out.println("SoproX.tocar(NotaX n)");
  }
}
```

Sobrescrita x Sobrecarga

```
public class SoproErro {
  public static void melodia(InstrumentoX i) {
    // ...
    i.tocar(NotaX.DO);
  }
  public static void main(String[] args) {
    SoproX flauta = new SoproX();
    melodia(flauta); // Não eh como o esperado!
  }
}
```

Classes e Métodos Abstratos

```
abstract class Instrumento {
  int i;
  abstract public void tocar();
  public String nome() {
    return "Instrumento";
  }
  abstract public void afinar();
}
class Sopro extends Instrumento {
  public void tocar() {
     System.out.println("Sopro.tocar()");
  }
  public String nome() { return "Sopro"; }
  public void afinar() {}
}
```

Classes e Métodos Abstratos

```
class Percussao extends Instrumento {
  public void tocar() {
    System.out.println("Percussao.tocar()");
  }
  public String nome() { return "Percussao"; }
  public void afinar() {}
}
class Corda extends Instrumento {
  public void tocar() {
    System.out.println("Corda.tocar()");
  }
  public String nome() { return "Corda"; }
  public void afinar() {}
}
```

Classes e Métodos Abstratos

```
    class SoproMetal extends Sopro {
      public void tocar() {
        System.out.println("SoproMetal.tocar()");
      }
      public void afinar() {
        System.out.println("SoproMetal.afinar()");
      }
    }
    class SoproMadeira extends Sopro {
      public void tocar() {
        System.out.println("SoproMadeira.tocar()");
      }
      public String nome() { return "SoproMadeira"; }
}
```

Classes e Métodos Abstratos

```
public class Musica4 {
  static void melodia(Instrumento i) {
    // ...
    i.tocar();
  }
  static void sinfonia(Instrumento[] e) {
    for(int i = 0; i < e.length; i++) melodia(e[i]);
  }
  public static void main(String[] args) {
    Instrumento[] orquestra = new Instrumento[5];
    int i = 0;
    // Upcasting:
    orquestra[i++] = new Sopro();
    orquestra[i++] = new Percussao();
    orquestra[i++] = new Corda();
    orquestra[i++] = new SoproMetal();
    orquestra[i++] = new SoproMadeira();
    sinfonia(orquestra);
  }
}
```

Inicialização e Amarração Tardia

```
abstract class Simbolo {
  abstract void desenha();
  Simbolo() {
    System.out.println("Simbolo() antes");
    desenha();
    System.out.println("Simbolo() depois");
  }
}
class SimboloGrego extends Simbolo {
  int tamanho = 1;
 SimboloGrego (int r) {
    tamanho = r;
    System.out.println(
      " SimboloGrego.SimboloGrego(), tamanho = "
      + tamanho);
  }
```

Inicialização e Amarração Tardia

```
    void desenha() {
      System.out.println(
        " SimboloGrego.desenha (),tamanho = " + tamanho);
    }
  }
  public class PoliConstrutor {
    public static void main(String[] args) {
      new SimboloGrego(5);
    }
}
```

```
Simbolo() antes
SimboloGrego.desenha(), tamanho = 0
Simbolo() depois
SimboloGrego.SimboloGrego(), tamanho = 5
```

Object e equals

```
class Valor {
  int i;
}
public class Equivalencia {
    public static void main(String[ ] args) {
      Integer n1 = new Integer(47);
      Integer n2 = new Integer(47);
      System.out.println(n1 == n2);
      System.out.println(n1 != n2);
      System.out.println(n1.equals (n2));
      Valor v1 = new Valor();
      Valor v2 = new Valor();
      v1.i = v2.i = 100;
      System.out.println(v1.equals(v2));
   }
}
```

O Método toString

- Qualquer objeto não primitivo tem toString
  - herdado de Object
- Chamada pelo compilador pode ser implícita

  System.out.println (“fonte = “ + fonte);
- Classes devem implementar sua própria toString

toString de Object

```
// HMS.java
class A { int i; }
public class HMS {
  static void f(Object[] x) {
    for(int i = 0; i < x.length; i++)
      System.out.println(x[i]);
  }
  public static void main(String[] args) {
    f(new Object[] { new Float(3.14), new A(),
       new Integer(47), new HMS(), new Double(11.11) });
    f(new Object[] {new A(), new A(), new A()});
  }
}
```

Exercícios

1 Crie uma subclasse Empregado Público de Empregado. Os atributos específicos desta subclasse são número do PASEP e estabilidade.

2 Crie uma subclasse Empregado Privado de Empregado. Os atributos específicos desta subclasse são número do PIS e valor do FGTS.

Exercícios

3 Faça um programa que leia um número n de Empregados (Públicos e Privados), ordene-os por nome e imprima todos os seus dados. Em seguida, ordene por aniversário e imprima novamente todos os seus dados.

4 O que deve ser alterado no programa 3 para que possa se ordenar também por salário e matrícula?