FACAMP
FACULDADES DE CAMPINAS

Laboratório de Programação

Aula 4

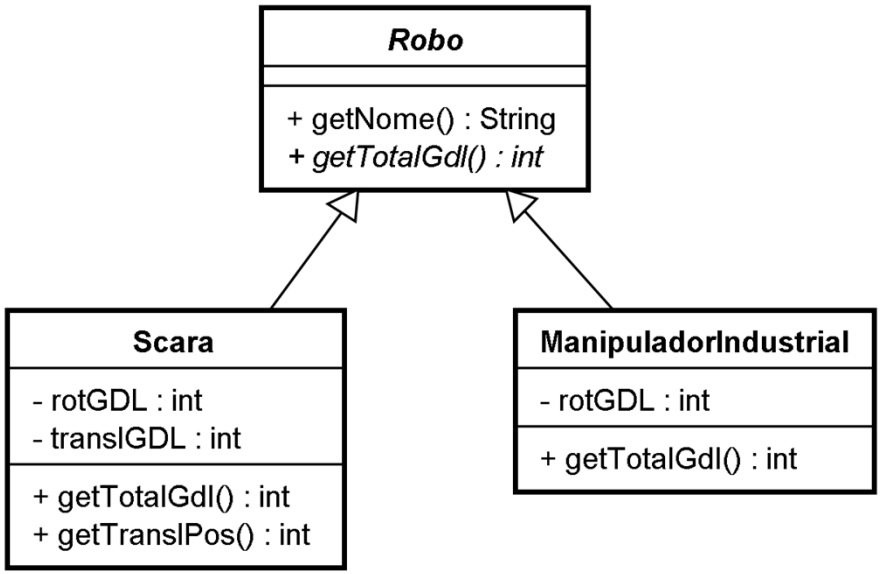
Java e Orientação a Objetos

2º semestre de 2019

Prof José Martins Junior

Herança

- Indica especialização de uma classe genérica
  - Relação: “é uma” ou “é um tipo de”
  - Classe-filha (subclasse) **estende** classe-pai (superclasse)
  - Atributos e métodos são herdadas pela classe-filha
  - Classe-filha pode incluir modificar métodos (sobrescrever) ou incluir novos atributos e métodos
  - Ex.:



Super e subclasses

- O operador `extends`
  - Indica que uma classe-filha (subclasse) é definida a partir de uma classe-pai (superclasse)
    - Ex.: `public class Scara extends Robo`
- A referência `super`
  - Utilizada pela subclasse para referenciar sua superclasse
    - Ex.: referência ao método **getNome()** de **Robo**, em uma classe filha

      `super.getNome()`
- A referência `this`
  - Referência que um objeto faz a sua própria instância
    - Ex.: referência ao método **getTotalGdl()** dentro de uma instância da classe **Scara** (neste caso, o uso é opcional)

      `this.getTotalGdl()`

Atribuindo subclasses

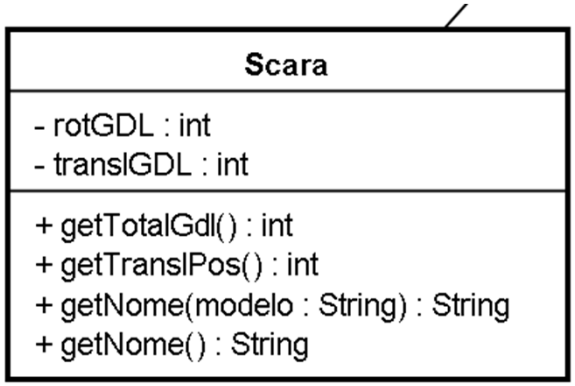
- A instância de uma subclasse pode ser utilizada no lugar de um objeto da superclasse
  - Ou seja, pode-se atribuir uma instância da subclasse a uma "variável" declarada com um tipo mais genérico
    - Ex.: `Robo r = new Scara();`
  - O acesso aos campos da instância será realizado pelos métodos definidos pela superclasse (tipo mais genérico)
  - Isso é porque um tipo mais genérico é sempre atendido por um objeto mais especialista (no mínimo, o especialista "é um tipo do" genérico)
- A recíproca normalmente é falsa
  - Motivo: o subtipo pode prever novos atributos e métodos que não serão providos pela instância da superclasse

Conversão

- *Casting* de objetos assemelha-se ao de tipos básicos
- Exemplo de uso
  - Suponha que a instância de uma subclasse foi atribuída a uma variável de tipo mais genérico (superclasse)
    - Isso limitará o acesso ao objeto, mas não o modificará
    - Ex.: `Robo r = new Scara();`
    - Neste caso, só serão acessíveis atributos e métodos de Robo
  - Para acessar as partes mais especialistas do objeto Scara como, por exemplo, o método **getTranslPos()**
    - Ex.: `(Scara) r.getTranlsPos();`
  - Se o objeto não for compatível com a associação, pode ocorrer um erro em tempo de execução
    - Pode-se testar a instância com o operador `instanceof`
    - Ex.: `if ( r instanceof Scara) ...`

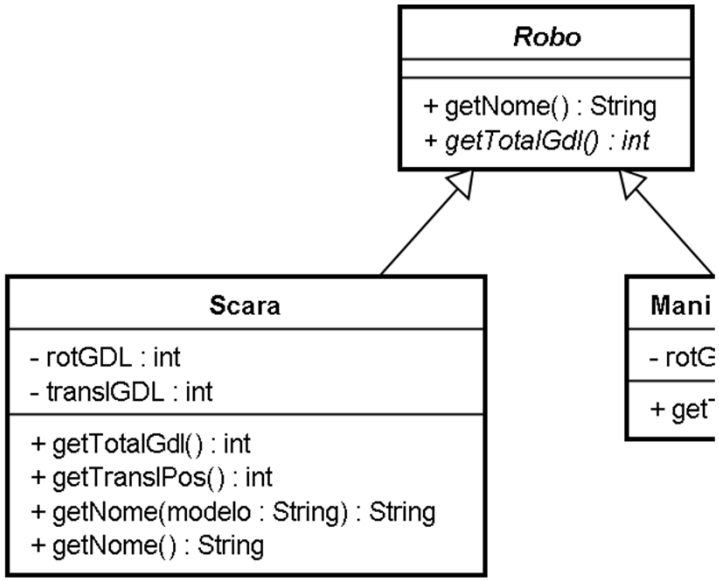
Sobrecarga *(overloading)*

- Reimplementação de um método em uma classe ou em uma subclasse dela
  - Muda-se a assinatura do método
    - Mantém-se o mesmo nome e tipo de retorno
    - Alteram-se tipos e/ou quantidades dos parâmetros de entrada
  - Ligação estática
    - Feita em tempo de compilação (*Static binding*), pela assinatura distinta do método
    - Ex.: inclusão de dois métodos getNome() na classe Scara, com diferentes assinaturas



Sobrescrita *(overriding)*

- Redefinição (ou reimplementação) de um método de uma classe, por uma subclasse sua
  - A assinatura do método não é alterada
  - A subclasse pode reescrever o método para refletir um comportamento mais especialista
    - Ex.: sobrescrita do método getNome() pela subclasse Scara




```
public class Robo {
   public String getNome() {
        return "Robô genérico";
   }
}
public class Scara extends Robo {
   public String getNome() {
        return "Robô Scara";
   }
}
```

Polimorfismo

- Capacidade de um objeto adequar o seu comportamento ao tipo específico da instância
  - Permite que uma instância obtenha e aplique a implementação mais especialista de um método que dispõe
  - Ligação tardia
    - Decidida em tempo de execução (*Dynamic binding*)
    - Ex.: atribuição de uma instância da classe Scara a uma variável do tipo Robo e, em seguida, a chamada ao método getNome()

```
Robo r = new Scara();
r.getNome();
```

    - O método retornará a String “Robô Scara”
    - Ou seja, o método executado é o implementado por Scara
    - A compilação só foi permitida porque existe implementação (ou, pelo menos, declaração) do método getNome() em Robo

Classes finais

- Quando deseja-se que uma classe não seja estendida utiliza-se o modificador final
  - Ex.: `public final class Quadrado extends Poligono`
- O modificador final pode ser aplicado a métodos de uma classe, informando que não poderão ser modificados por subclasses
- Um método declarado como final é tratado como uma ligação estática, economizando tempo de processamento

Classes abstratas

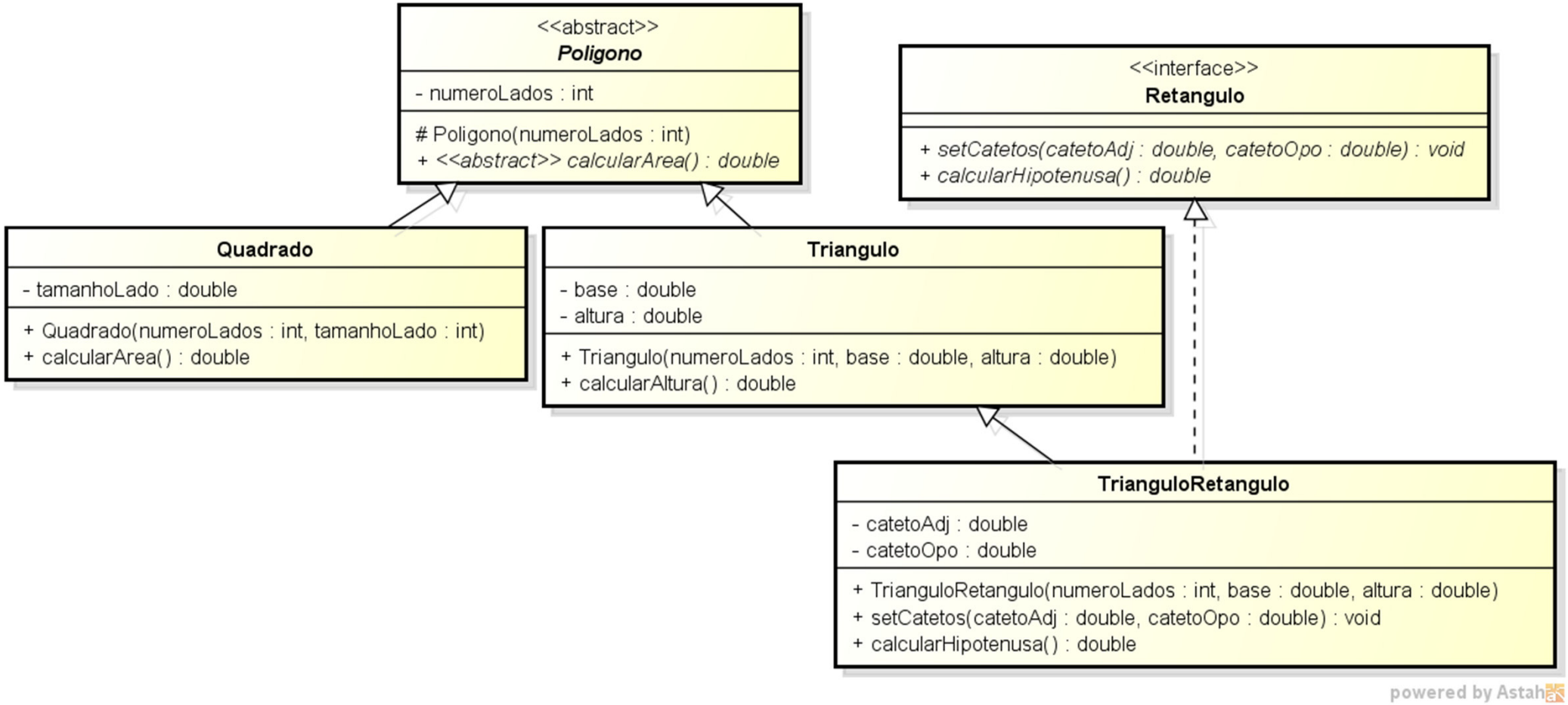
- Tratam da generalização de mais alto nível de uma hierarquia. Úteis para definição do modelo de implementação de classes que venham estendê-la
  - Toda classe que possuir (pelo menos) um método abstrato (abstract) deve ser declarada como abstrata
  - Pode conter campos de dados concretos e métodos não abstratos, como qualquer superclasse
  - Métodos abstratos de uma classe devem ser implementados por qualquer subclasse que a estenda
  - A classe abstrata somente fará a declaração do método (visibilidade, parâmetros de entrada e retorno)
  - Exemplo: uma classe abstrata Poligono que define um método abstrato `calcularArea()`

```
public abstract class Poligono { ...
    public abstract double calcularArea();  ... }
```

Interfaces

- Interface define um modelo para a implementação de certos métodos que pode ser feita por outras classes
  - Em Java, uma classe só pode ter uma classe-pai
    - Para a implementação de herança múltipla, ou seja, para adicionar características adicionais paralelas à linhagem oficial
    - Uma subclasse pode implementar (cláusula implements) uma interface (ou várias), mesmo que sua superclasse não o faça
  - A interface somente faz a declaração do método (visibilidade, parâmetros de entrada e retorno)
    - A classe que implementa uma interface deve implementar os métodos que ela declarou
  - Interfaces não podem ser instanciadas
    - Atribui-se a um tipo declarado como interface uma instância da classe que a implementa
  - Métodos estáticos não podem ser colocados em interfaces
  - Não deve conter campos de instância, mas pode definir constantes

Exemplo


<<abstract>>
Poligono
- numeroLados : int
Poligono(numeroLados : int)
+ <<abstract>> calcularArea() : double
<<interface>>
Retangulo
+ setCatetos(catetoAdj : double, catetoOpo : double) : void
+ calcularHipotenusa() : double
Quadrado
- tamanhoLado : double
+ Quadrado(numeroLados : int, tamanhoLado : int)
+ calcularArea( ) : double
Triangulo
- base : double
- altura : double
+ Triangulo(numeroLados : int, base : double, altura : double)
+ calcularAltura( ) : double
TrianguloRetangulo
- catetoAdj : double
- catetoOpo : double
+ TrianguloRetangulo(numeroLados : int, base : double, altura : double)
+ setCatetos(catetoAdj : double, catetoOpo : double) : void
+ calcularHipotenusa( ) : double
powered by Astah

A superclasse Object

- É o ancestral mais básico em Java
- Toda classe em Java estende a classe Object
- Principais métodos herdados
  - `Class getClass()` - retorna um objeto Class que contém informações sobre o objeto instanciado
  - `boolean equals(Object obj)` - compara se dois objetos são iguais (se apontam para o mesmo lugar na memória)
  - `Object clone()` - cria um clone (cópia do objeto na memória)
  - `String toString()` - retorna uma String que representa o valor do objeto (normalmente as classes redefinem tal método para retornarem um valor mais significativo)

A classe Class

- Java mantém a identificação de tipo em tempo de execução (RTTI) sobre todos os objetos
- Um objeto da classe Class, retornado pela chamada do método `getClass()` (de Object) pode fornecer tais informações
- Principais métodos
  - `String getName()` - retorna o nome da classe
  - `static Class forName(c)` - retorna um objeto Class para um nome de "variável" de instância
  - `Object newInstance()` - obtém uma instância para o objeto Class definido

Bibliografia

DEITEL, P.; DEITEL, H. Java TM: como programar. 8ª edição. São Paulo: Pearson Prentice Hall, 2012. 1144p.

GOSLING, J.; ARNOLD, K.; HOLMES, D. A Linguagem de Programação Java. 4ª edição, Porto Alegre: Bookman, 2007.