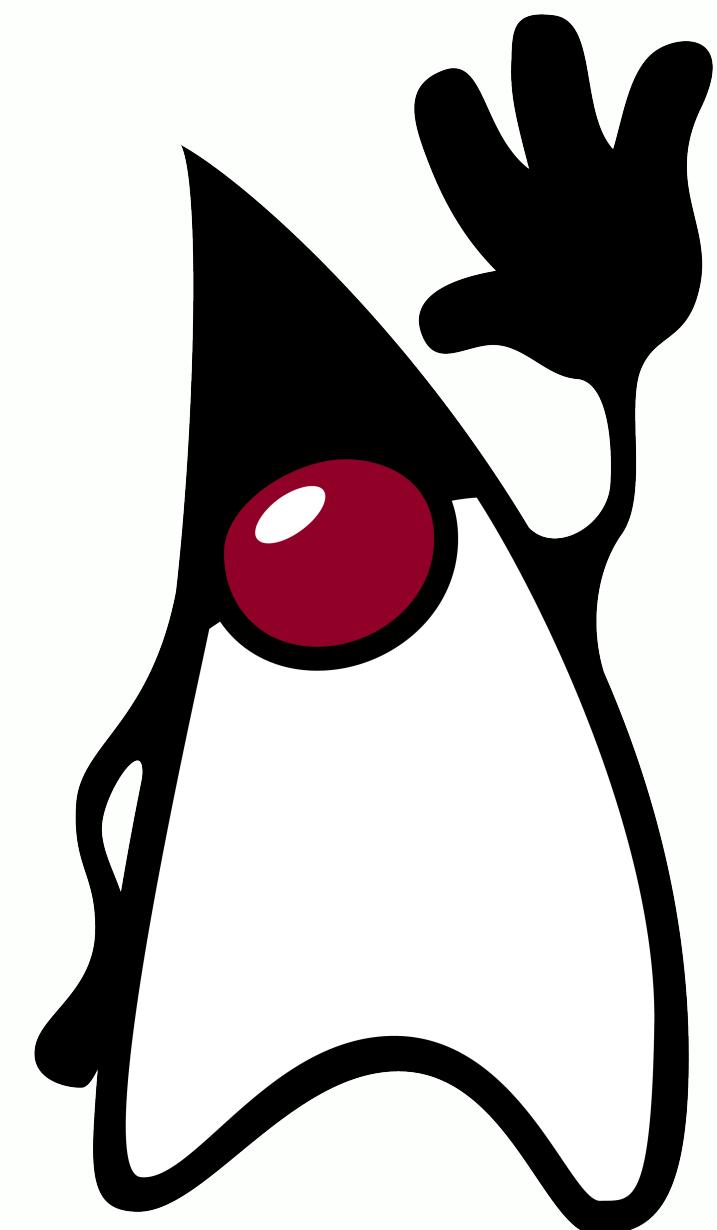


# Herança

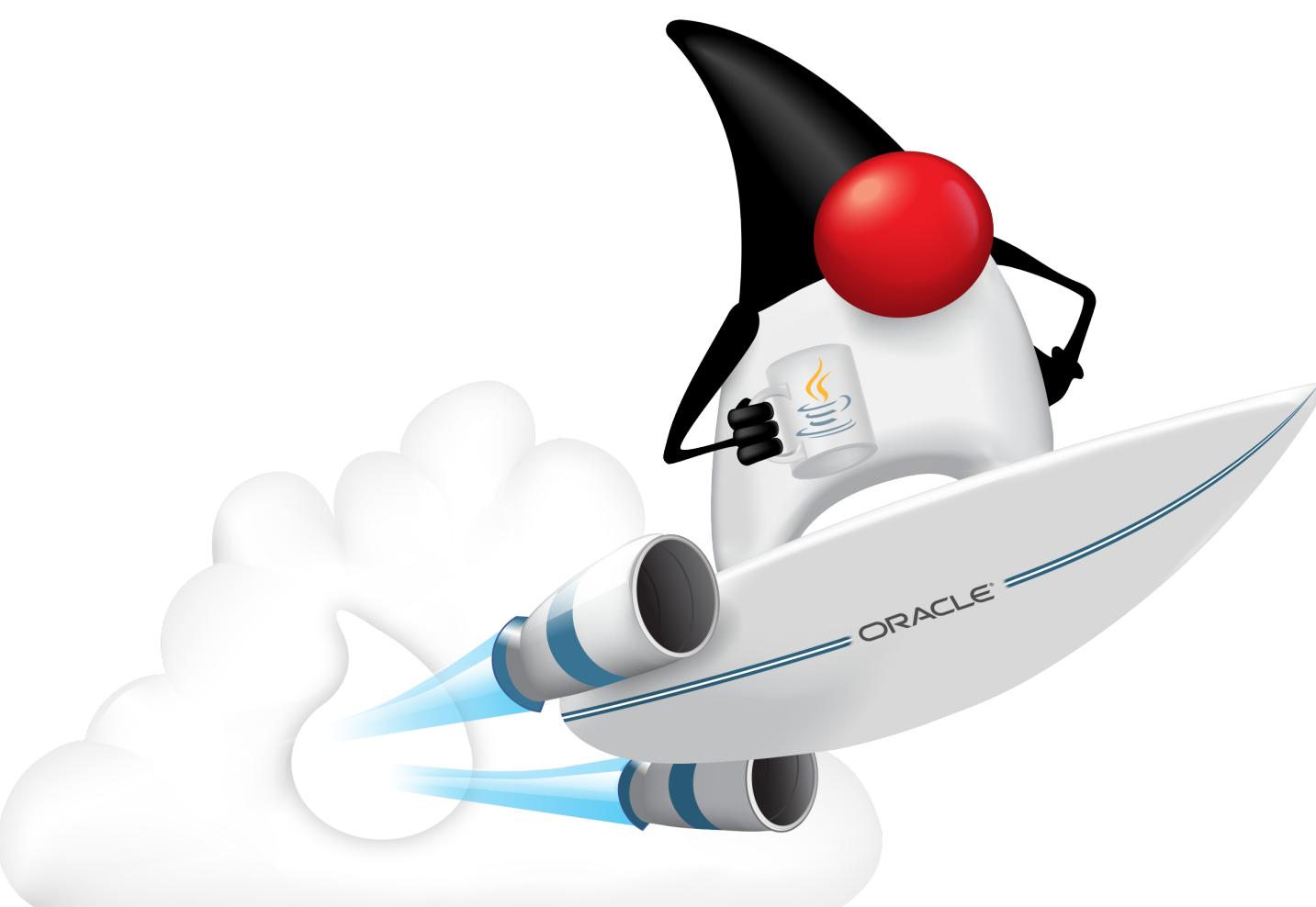
QXD0007 - Programação Orientada a Objetos



Prof. Bruno Góis Mateus ([brunomateus@ufc.br](mailto:brunomateus@ufc.br))

# Conteúdo

- Introdução
- Herança
- Verificação dinâmica de tipos
- Reescrita



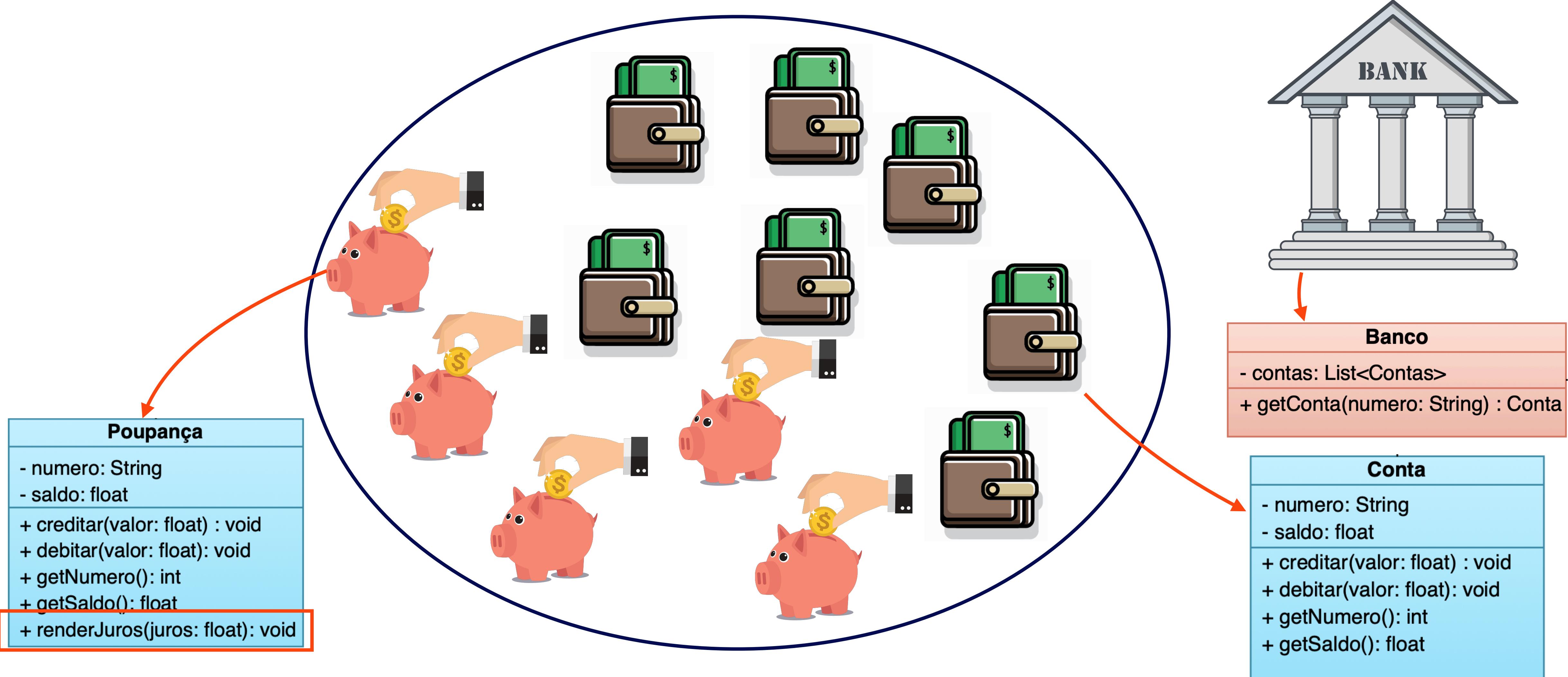
# Introdução



# Introdução

- Talvez um dos recursos mais importante da programação orientada a objeto seja o **reuso**
- Podemos promover o **reuso** de várias formas, incluído a definição de relacionamentos entre classes
  - Organizando classes a partir de pontos comuns entre elas
  - **Herança** é o mecanismo responsável por prover tal funcionalidade

# Introdução



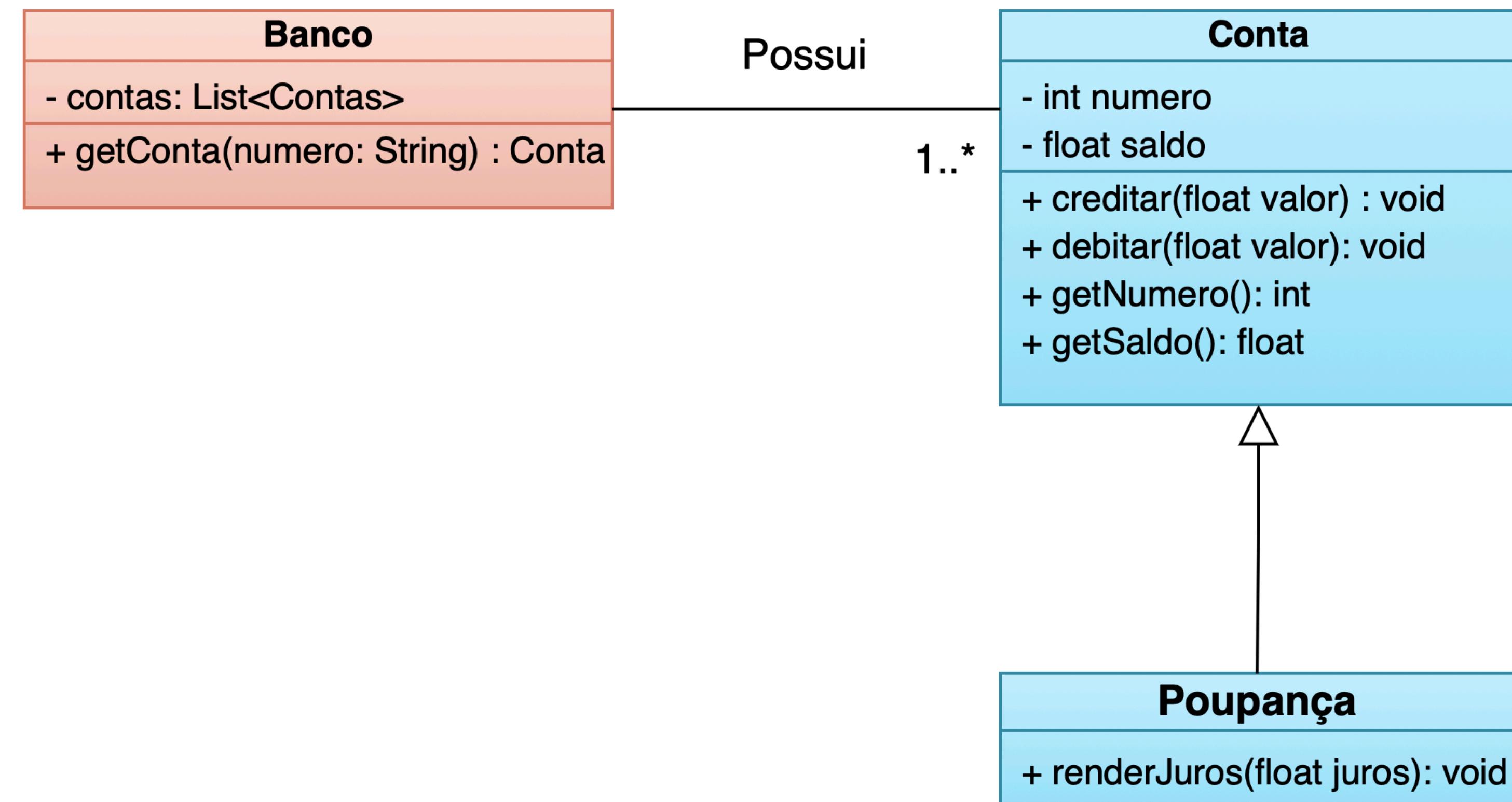
# Herança



# Herança

- Podemos relacionar **duas classes** de forma que uma **herda características** de outra classe
- Criando uma relação de **mãe e filha**
  - **Superclasse e subclasse**

# Herança



# Herança

## Reuso

- A descrição da **superclasse** pode ser usada para definir a **subclasse**

## Extensibilidade

- Algumas operações da **superclasse** podem ser redefinidas na **subclasse**

# Herança

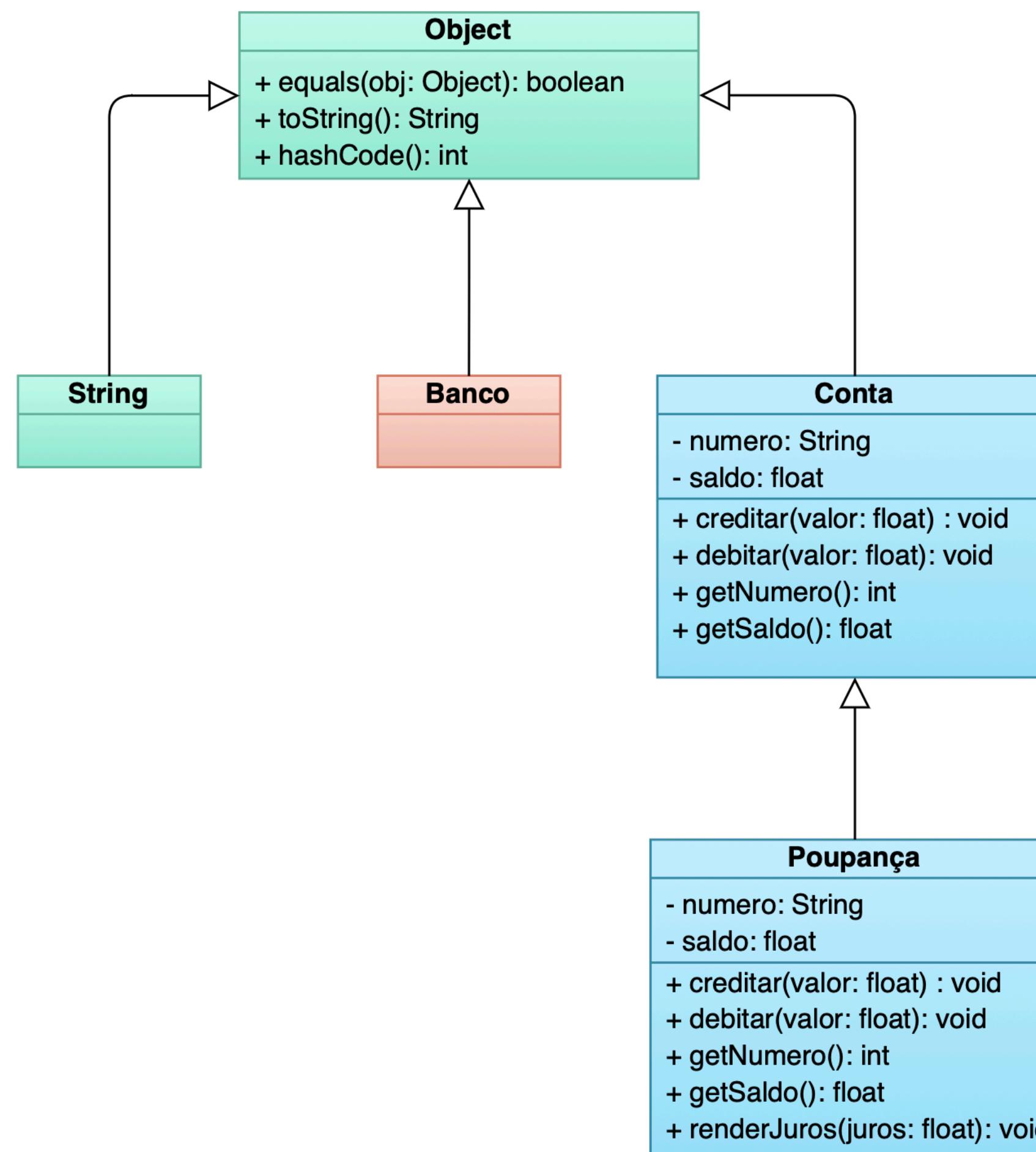
## Comportamento

- Objetos **subclasse** comportam-se como os objetos da **superclasse**

## Substituição

- Objetos da **subclasse** podem ser usados no lugar de objetos da **superclasse**

# Herança



## Herança em Java

- Todas as classes herdam da classe **Object**
- Implementa o mecanismo de **Herança Simples**
  - Só é possível herdar de **uma classe**
  - Atributos e métodos **privados** são herdados, mas **não podem ser acessados diretamente**
  - **Construtores** não são herdados

# Verificação dinâmica de tipos



# Verificação dinâmica de tipos

## Subclasses e Subtipos

- Classes definem “tipos”
- Subclasses definem “subtipos”

Objetos das subclasses podem ser utilizados onde os objetos dos superclasse são requeridos

# Verificação dinâmica de tipos

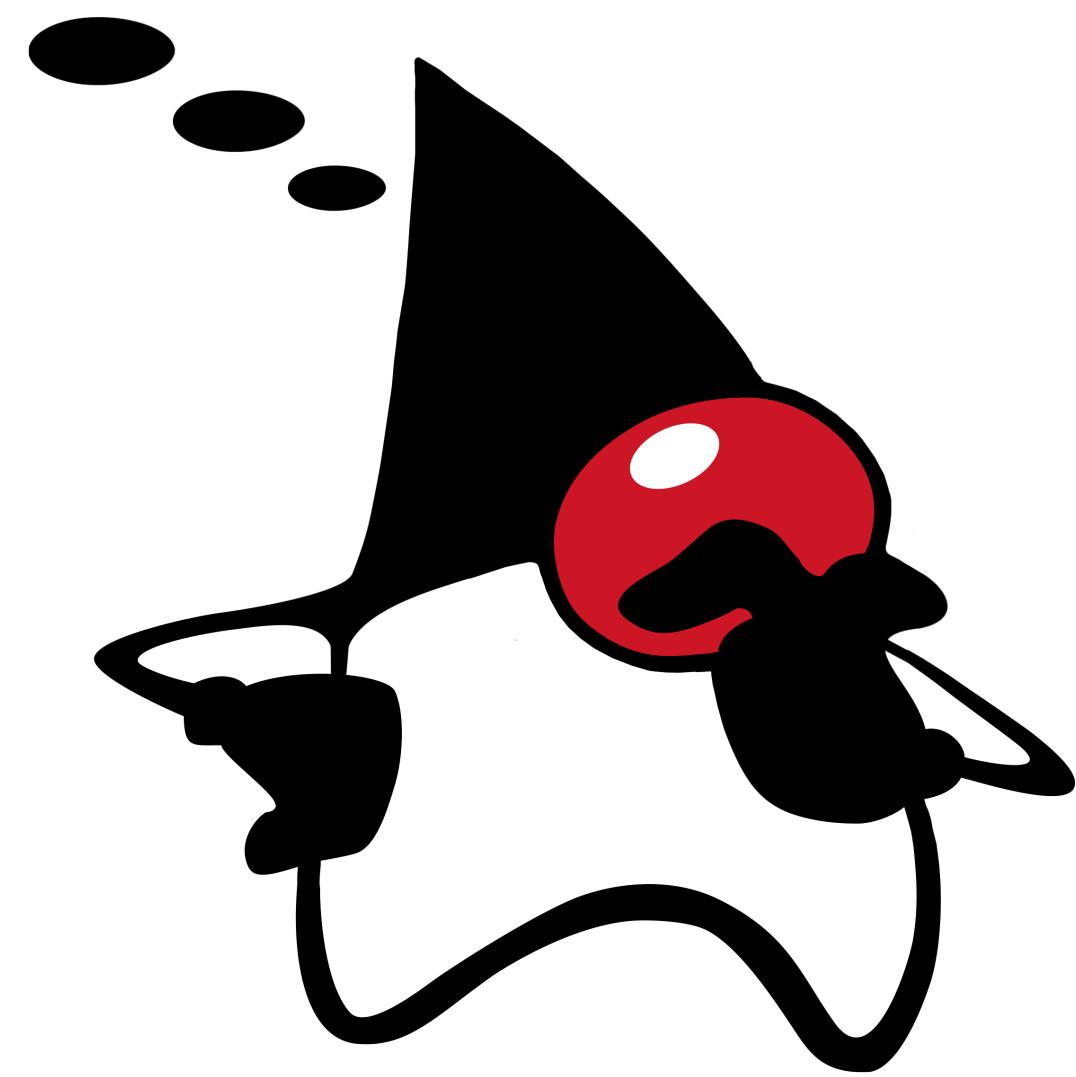
- Em casos onde há **substituição**, podemos forçar o compilador a “enxergar” o tipo dinâmico da variável
- A **coerção** facilita a verificação estática de tipos
  - Em tempo de execução, pode levar a erros de tipagem
- Em Java, por precaução, usamos **instanceof** para verificar o tipo dinâmico de uma variável antes de fazermos uma coerção
- O **instanceof** retorna **false** quando a operação não casa ou **true** caso contrário

# Verificação dinâmica de tipos

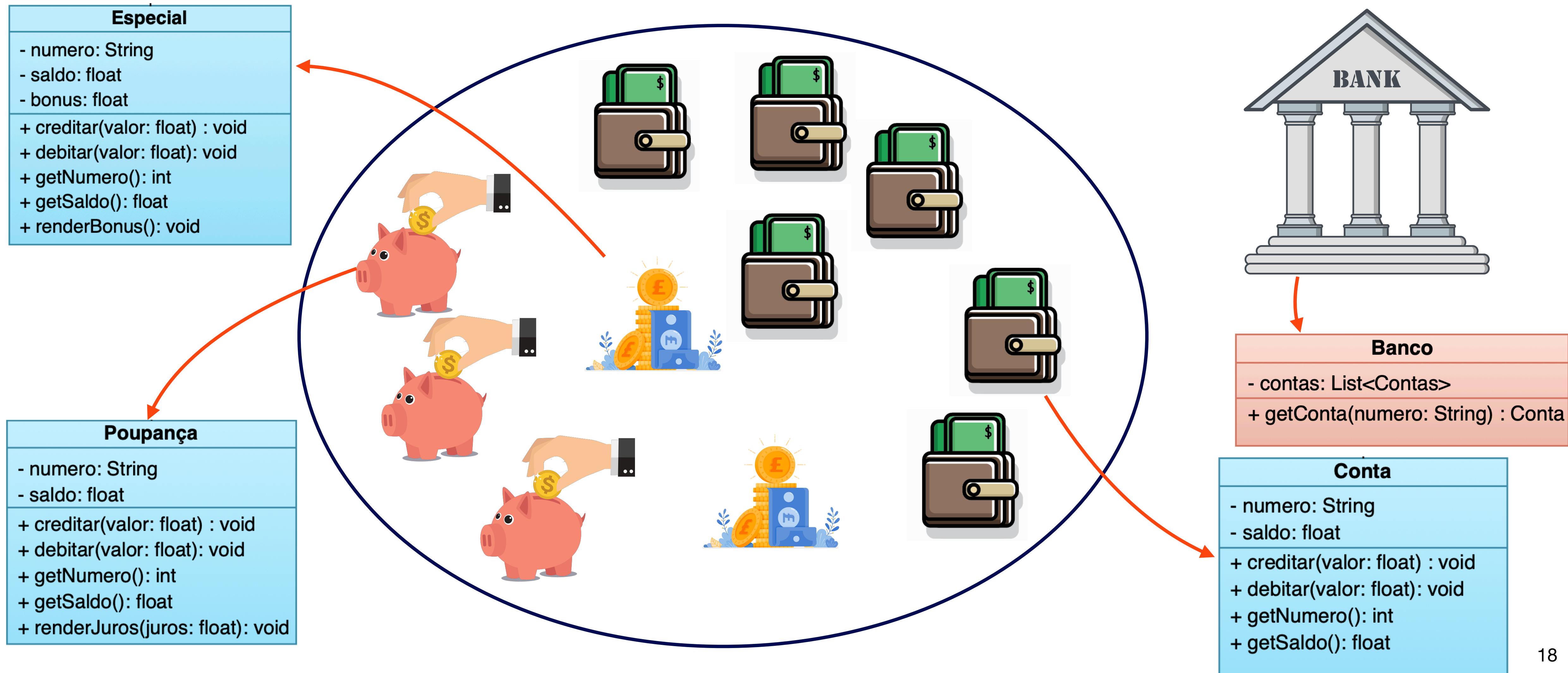
## Variáveis Polimórficas

- Elas podem armazenar objetos do tipo (classe) declarado ou dos subtipos (subclasses) do tipo (classe) declarado!
- Chamamos de **polimorfismo de tipo**

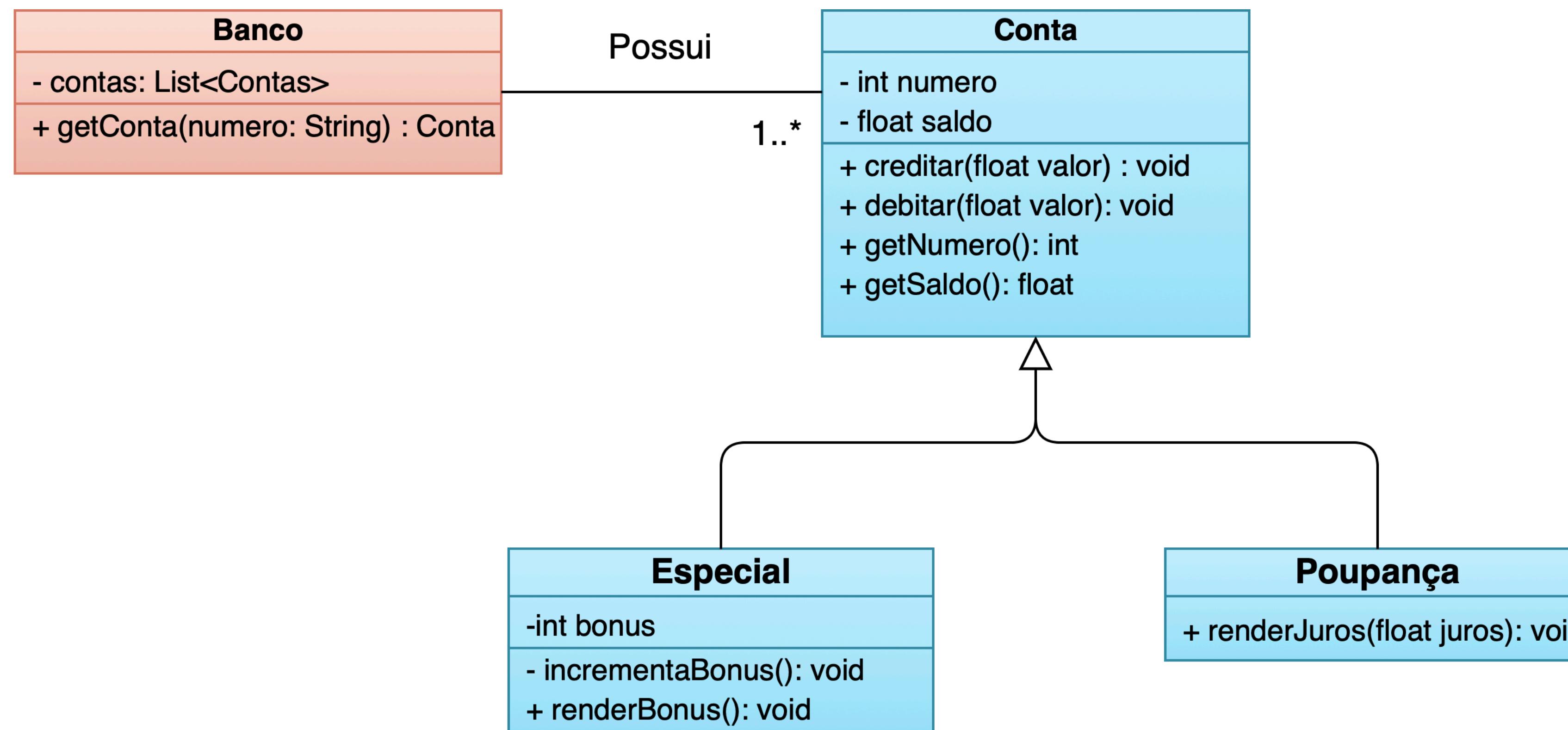
# Reescrita



# Reescrita



# Reescrita



# Reescrita

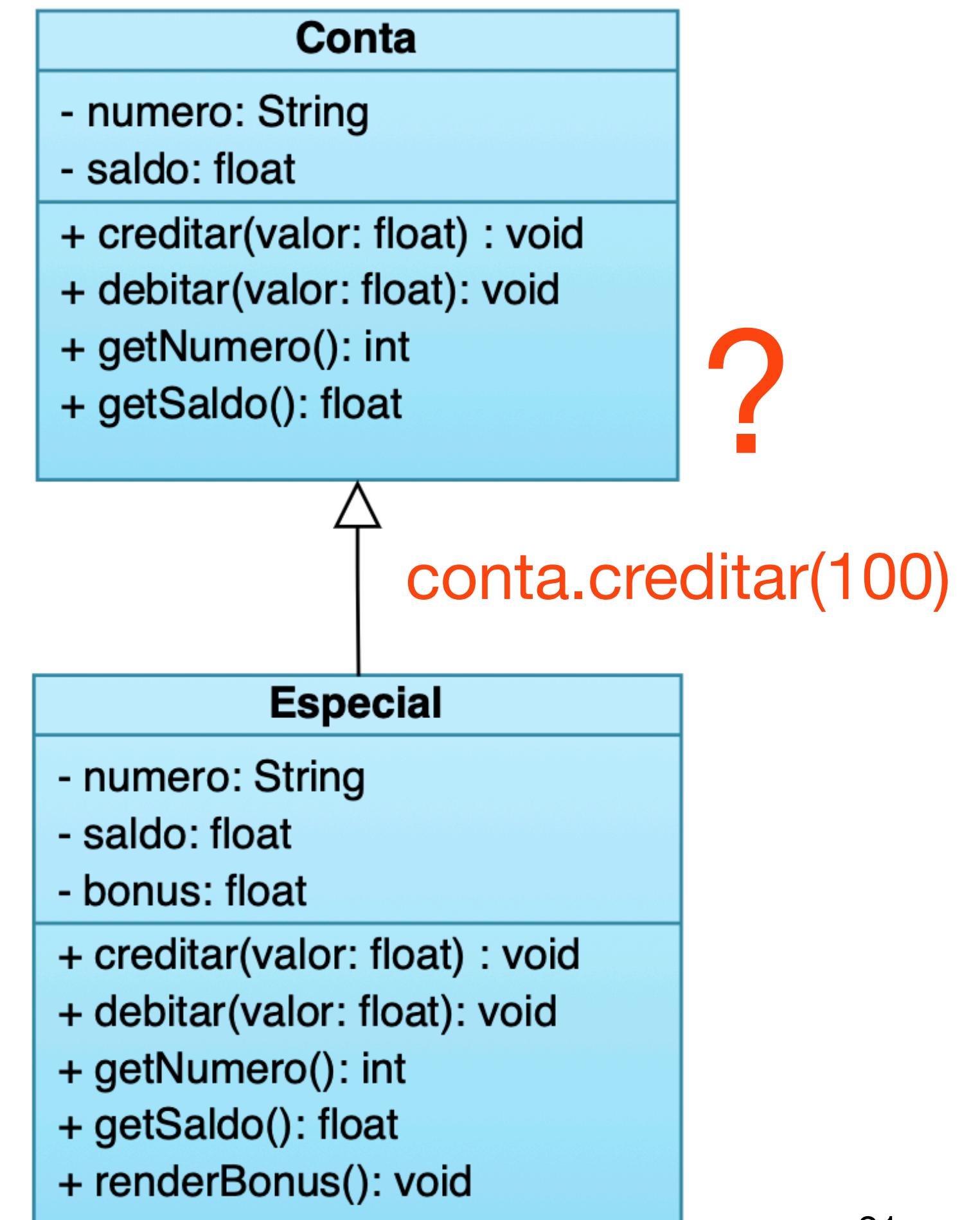
## Reescrita

- A **subclasse** redefine **métodos da superclasse com a mesma assinatura**
- Cada método tem **acesso aos atributos da classe na qual foi definido**
- A **superclasse** satisfaz a verificação do **tipo estático**
- O método da **subclasse** é chamado **em tempo de execução**, ele sobrescreve a versão da superclasse
- Só é possível acessar a definição dos métodos da superclasse imediata via **super**

# Reescrita

## Ligaçāo Tardia/Dinâmica de método

- O código é escolhido dinamicamente (em **tempo de execução**), não estaticamente (em **tempo de compilação**)
- Escolha é feita com base no **tipo dinâmico** do objeto associado à variável de acesso ao método



Por hoje é só

