Vinculação dinâmica

Vinculação dinâmica (1)

- Vinculação é uma associação possivelmente entre um atributo e uma entidade.
- Variáveis de programa são entidades com atributos de nome, tipo e área de armazenamento.
- Tempo de vinculação é o tempo em que ocorre a vinculação entre a entidade e seus atributos.

Vinculação dinâmica (2)

- Vinculação estática ocorre antes do tempo de execução,
- e permanece inalterado durante a execução do programa.

Vinculação dinâmica (3)

- Vinculação dinâmica ocorre durante o tempo de execução,
- e muda no curso da execução do programa.
- Na vinculação dinâmica, as associações podem ser alteradas em tempo de execução.

Vinculação dinâmica (4)

- Em POO, vinculação está relacionado com o mapeamento entre o nome de um método e sua implementação.
- Uma mensagem enviada em tempo de execução é dinamicamente vinculada a uma implementação dependendo do tipo do objeto recebedor da mensagem.

Tipo estático e dinâmico (1)

- Uma hierarquia de tipos mais complexa requer mais conceitos para descrevê-la.
- Alguns novos termos:
  - **tipo estático;**
  - **tipo dinâmico; e**
  - **encaminhamento e pesquisa de método (method dispatch/lookup).**

Tipo estático e dinâmico (2)

- O tipo declarado de uma variável é seu *tipo estático*.
- O tipo de objeto que uma variável referencia é seu *tipo dinâmico*.
- O trabalho do compilador é verificar violações nos tipos estáticos.

```
Item item = (Item) iter.next();
item.print(); // Erro em tempo de compilação.
```

Tipo estático e dinâmico (3)

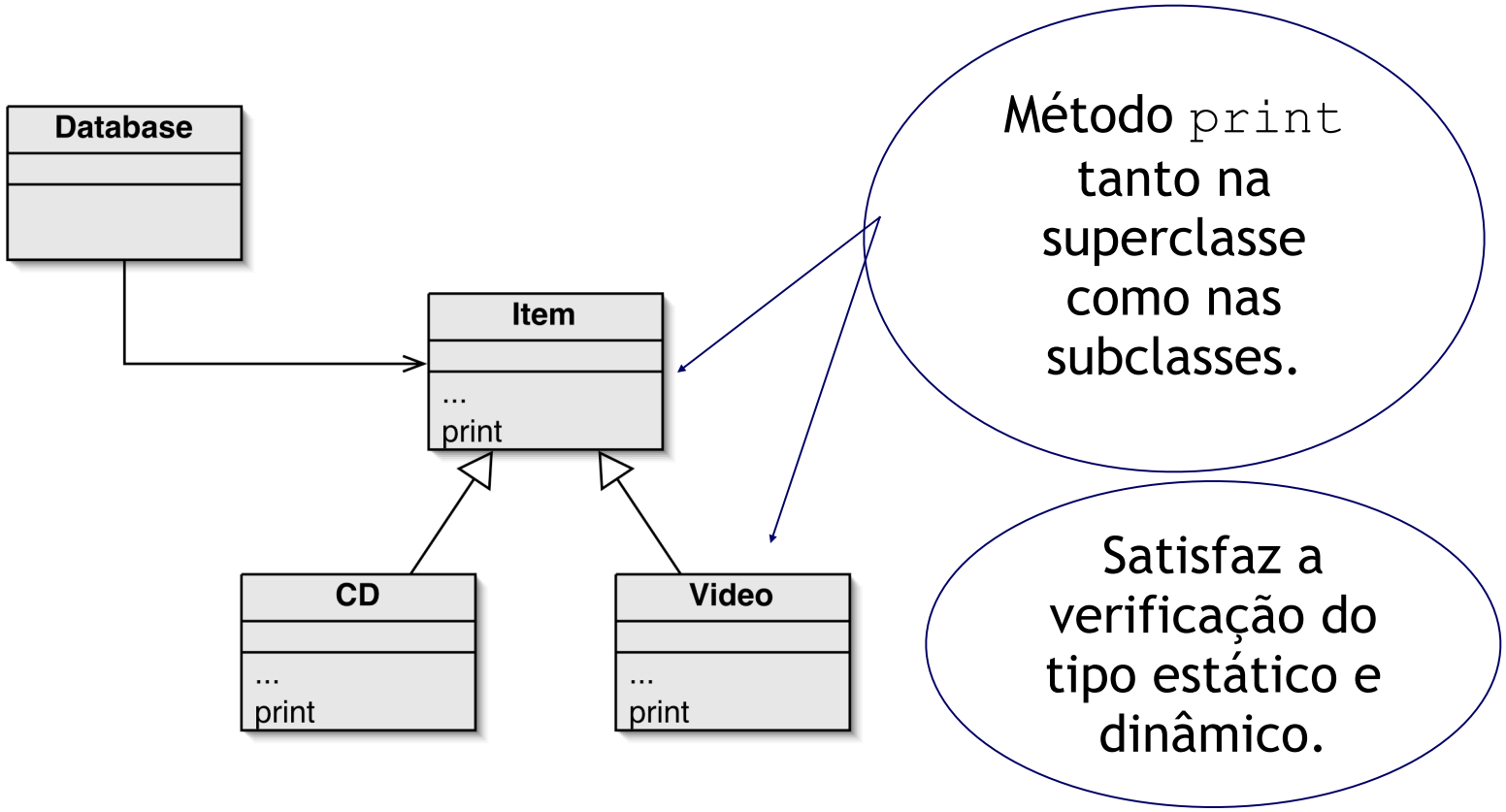
Qual é o tipo de c1?

```
c1: Car
c1 = Car()
```

Qual é o tipo de v1?

```
v1: Vehicle
v1 = Car()
```

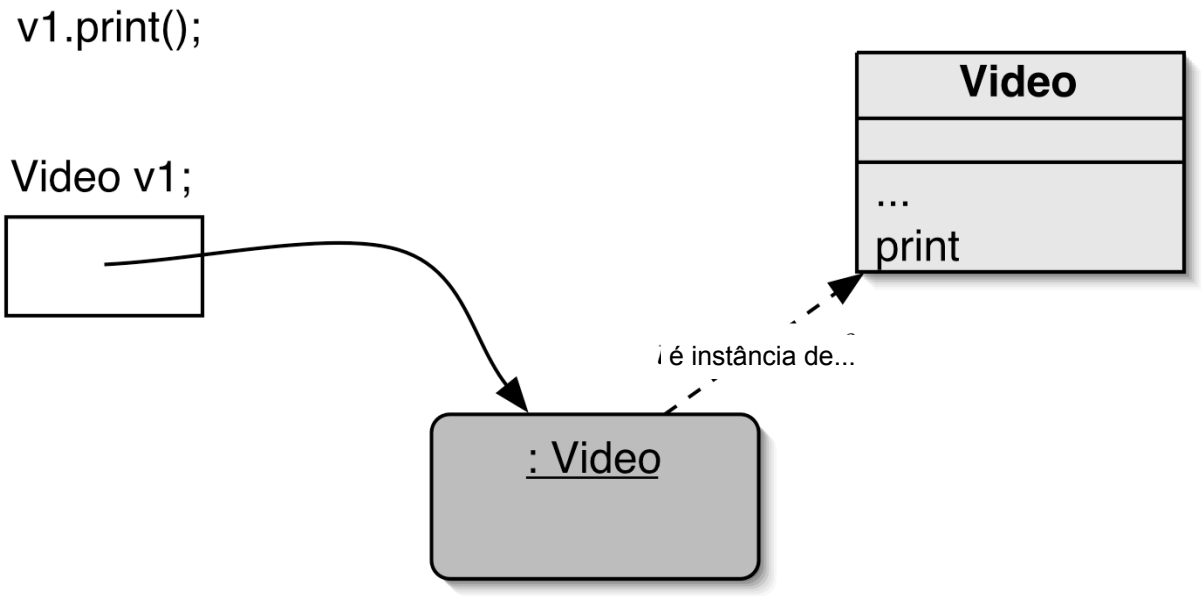
Sobrescrever: a solução



Sobrescrição

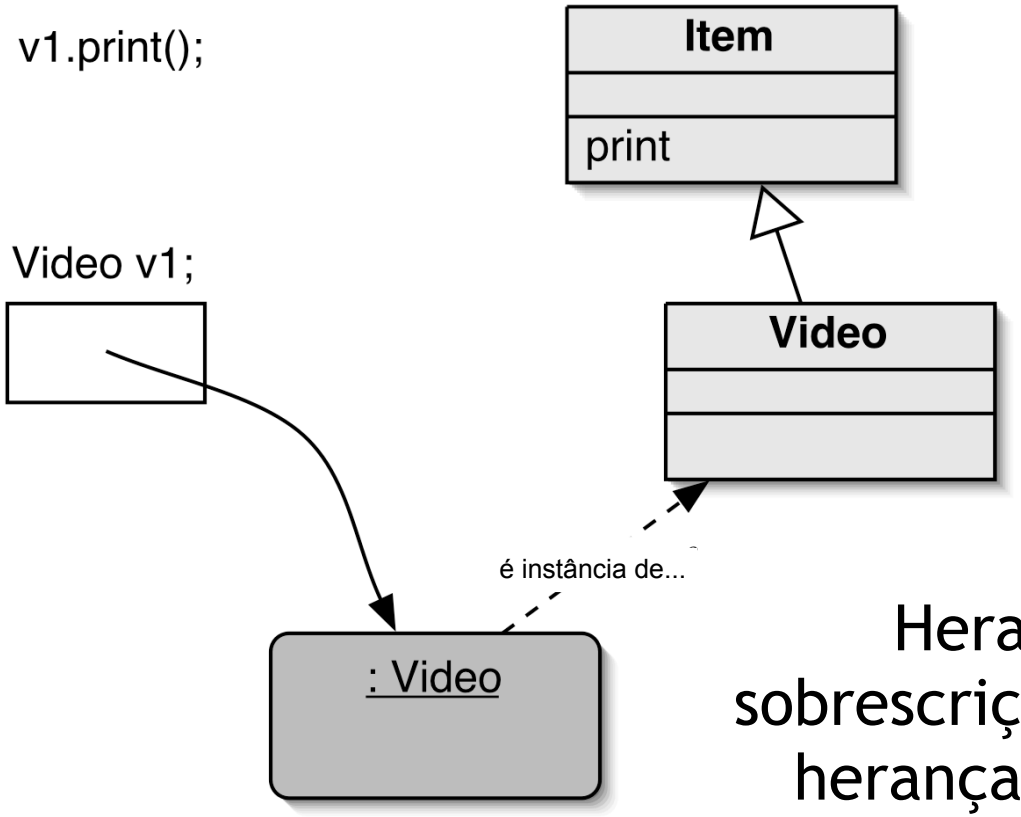
- Superclasse e subclasse definem métodos com a mesma assinatura.
- Cada uma tem acesso aos atributos da sua classe.
- A superclasse satisfaz a verificação do tipo estático.
- O método da subclasse é chamado em tempo de execução — ele *sobrescreve* a versão da superclasse.

Pesquisa de método (1)



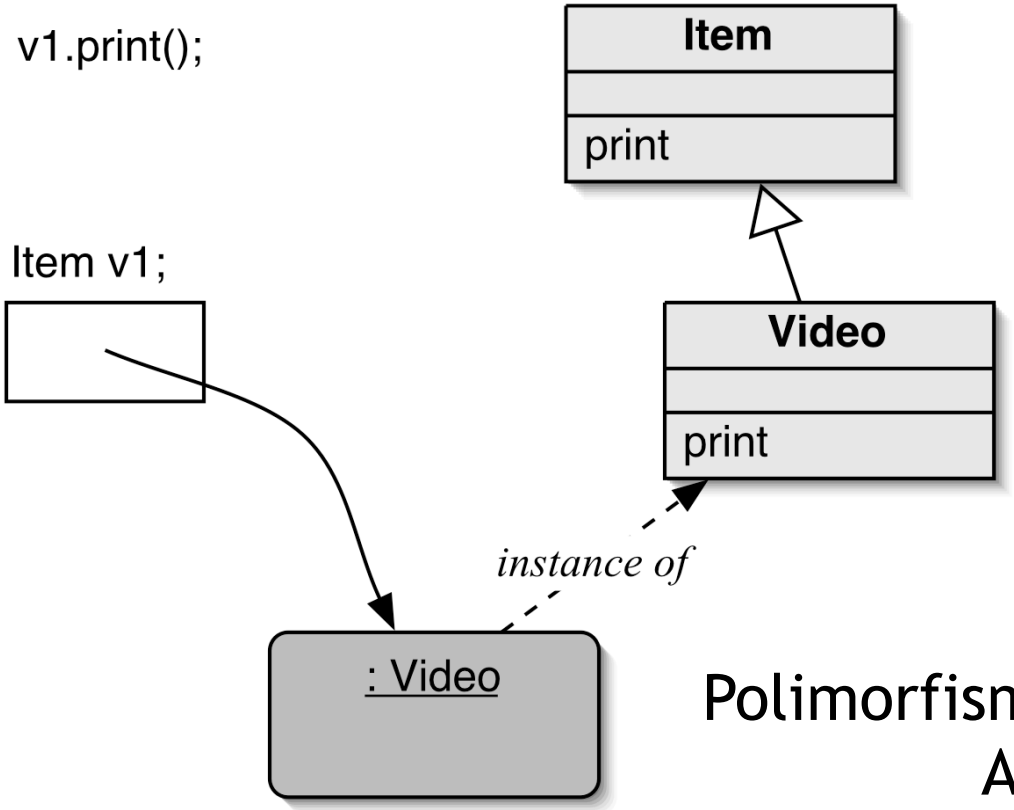
Nenhuma herança ou polimorfismo.
O método óbvio é selecionado.

Pesquisa de método (2)



Herança, mas nenhuma sobrescrição. A hierarquia da herança sobe, pesquisando uma correspondência.

Pesquisa de método (3)



Polimorfismo e sobrescrição.
A ‘primeira’ versão encontrada é utilizada.

Resumo da pesquisa de método

- A variável `v1` é acessada.
- O objeto armazenado na variável é encontrado.
- A classe do objeto é encontrada.
- É pesquisada na classe uma correspondência de método.
- Se nenhuma correspondência for encontrada, a superclasse é pesquisada.
- Isso é repetido até que uma correspondência seja encontrada ou a hierarquia da classe seja exaurida.
- Métodos sobrescritos têm precedência.

Chamadas super em métodos

- Métodos sobrescritos são ocultados...
- ... mas, freqüentemente, queremos ser capazes de chamá-los.
- Um método sobrescrito *pode* ser chamado a partir do método que o sobrescreve.

```
super().method(…)
```

  - **Compare com a utilização de super nos construtores.**

Chamando um método sobrescrito

```
class CD(Item):

    ...
    def imprime(self):

        super().imprime()
        print("    " + self._artist)
        print("    tracks: " +
                              self._numberOfTracks)

    ...
```

Polimorfismo de método

- Discutimos o encaminhamento do *método polimórfico*.
- Uma variável polimórfica pode armazenar objetos de diferentes tipos.
- Chamadas de método são polimórficas.
  - **O método real chamado depende do tipo do objeto dinâmico.**

Os métodos da classe `Object`

- Métodos em `Object` são herdados por todas as classes.
- Qualquer um desses pode ser sobrescrito.

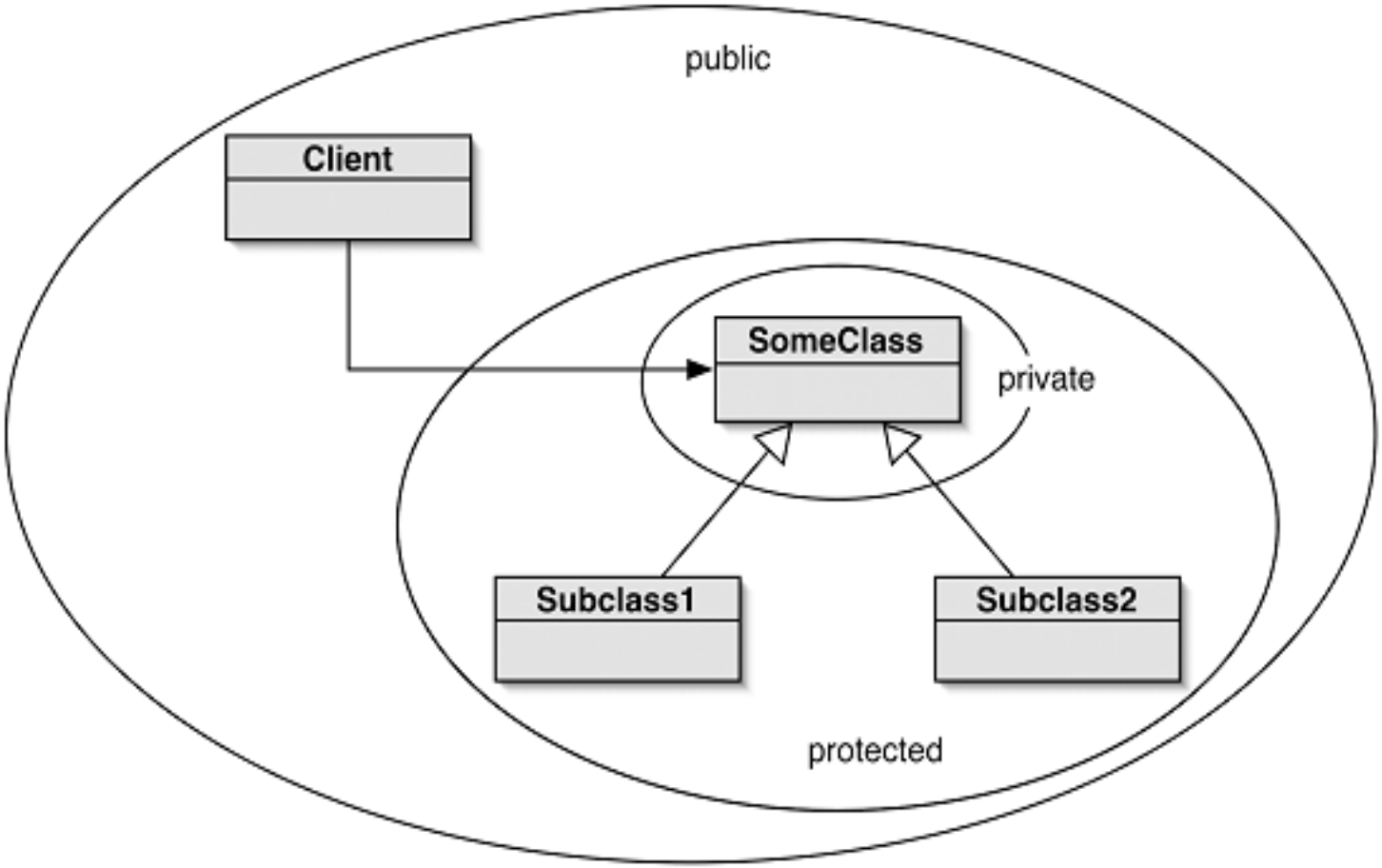
Acesso protegido (1)

- Acesso *privado* na superclasse pode ser bem restritivo para uma subclasse.
- O relacionamento de herança mais próximo é suportado pelo acesso *protegido*.
- O acesso *protegido* é mais restritivo que o acesso *público*.

Acesso protegido (2)

- Ainda recomendamos manter os atributos como privados.
  - Define métodos de acesso e métodos modificadores protegidos.
- Em Python, dois underlines (__) os atributos/métodos são privados
- Um underline (_), os atributos/métodos são protegidos.

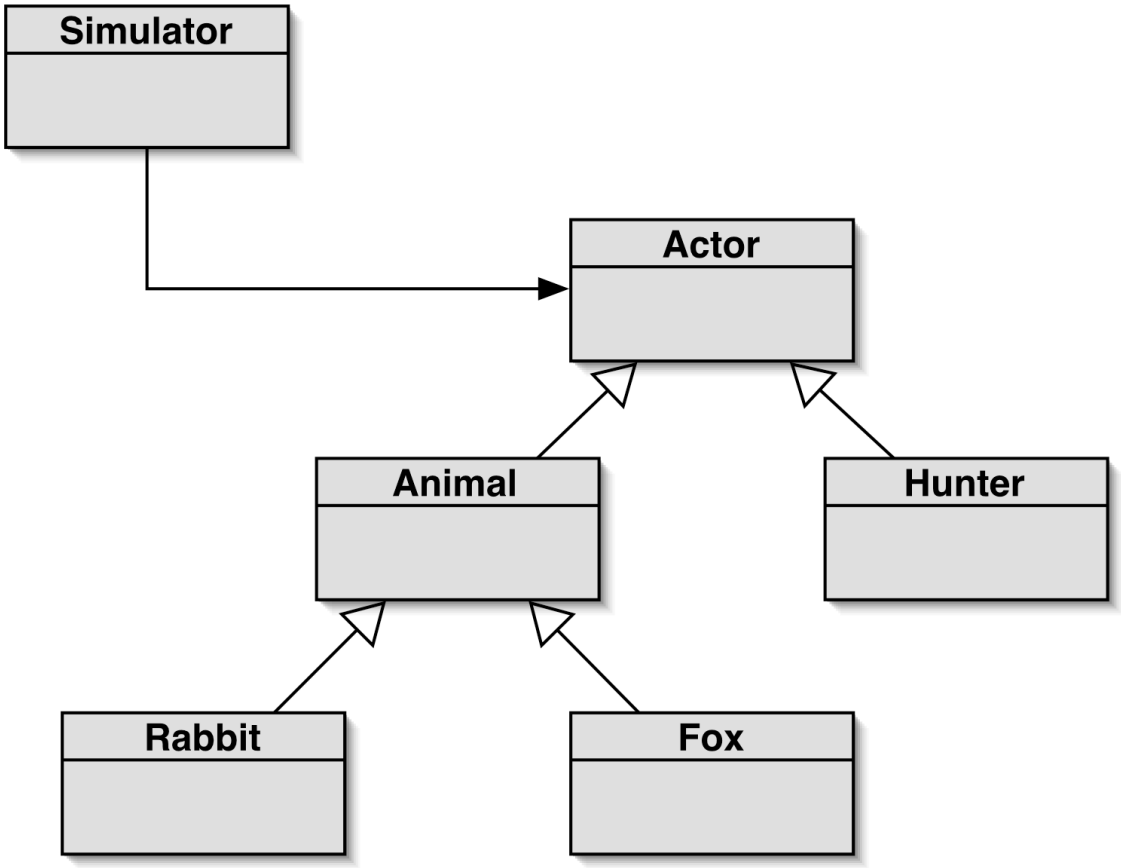
Níveis de acesso



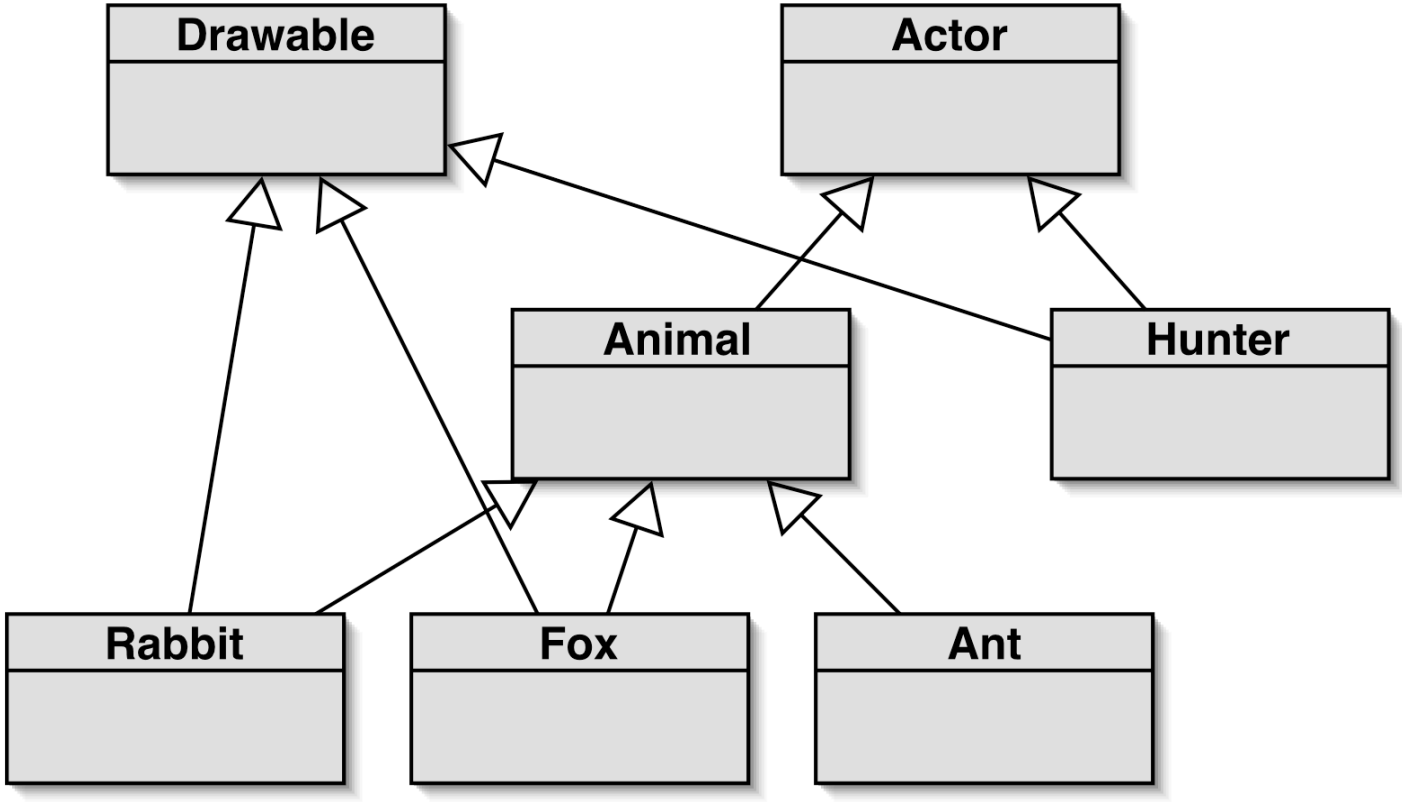
Métodos e classes abstratas

- Métodos abstratos têm `abstract` na assinatura.
- Métodos abstratos não têm nenhum corpo.
- Métodos abstratos tornam a classe abstrata.
- Classes abstratas não podem ser instanciadas.
- Subclasses concretas completam a implementação.

Mais abstração



Desenho seletivo (herança múltipla)



Herança múltipla

- Faz com que uma classe herde diretamente de múltiplos ancestrais.
- Cada linguagem tem suas próprias regras.
  - **Como resolver as definições de concorrência?**
- O Java proíbe isso para classes.
- O Python permite isso para classes.
- O Java permite isso para interfaces.
  - **Nenhuma implementação concorrente.**

Uma interface `actor`

```
public interface Actor
{
    /**
     * Realiza o comportamento diário do ator.
     * Transfere o ator para updatedField se ele for
     * participar das etapas posteriores        da simulação.
     * @param currentField O estado atual do campo.
     * @param location A posição do ator no campo.
     * @param updatedField O estado atualizado do campo.
     */
    void act(Field currentField, Location location,
             Field updatedField);
}
```

Classes implementam uma interface

```
public class Fox extends Animal implements Drawable
{
    ...
}
```

```
public class Hunter implements Actor, Drawable
{
    ...
}
```

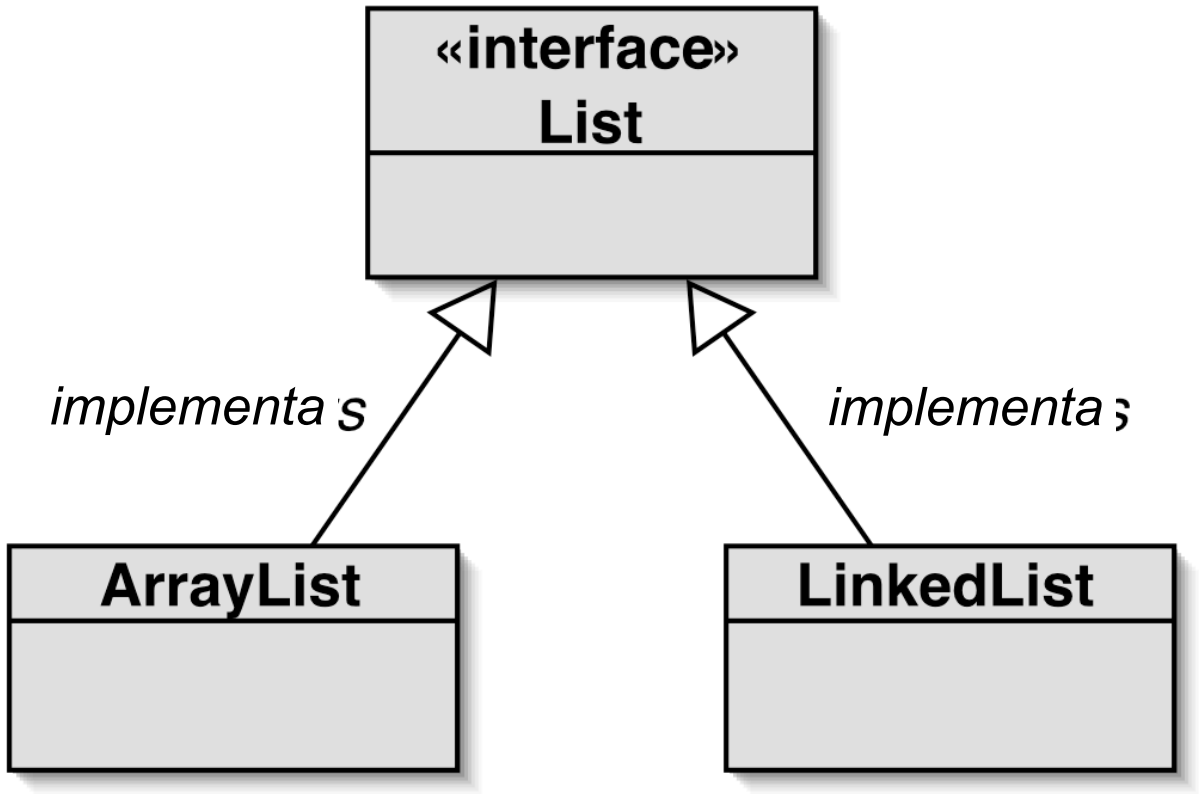
Interfaces como tipos

- Classes de implementação não herdam o código, mas...
- ... classes de implementação são subtipos do tipo de interface.
- Portanto, o polimorfismo está disponível para interfaces e classes.

Interfaces como especificações

- Forte separação entre funcionalidade e implementação.
  - **Embora parâmetros e tipos de retorno sejam obrigatórios.**
- Clientes interagem independentemente da implementação.
  - **Mas os clientes podem escolher implementações alternativas.**

Implementações alternativas



Interfaces

- Interfaces fornecem uma especificação sem uma implementação.
  - **Interfaces são totalmente abstratas.**
- Interfaces suportam polimorfismo.
- Interfaces Java suportam herança múltipla.

Revisão (1)

- Herança pode fornecer implementação compartilhada.
  - **Classes concretas e abstratas.**
- Herança fornece informações sobre o tipo compartilhado.
  - **Classes e interfaces.**

Revisão (2)

- Métodos abstratos permitem a verificação do tipo estático sem exigir uma implementação.
- Classes abstratas funcionam como superclasses incompletas.
  - **Nenhuma instância.**
- Classes abstratas suportam o polimorfismo.

Revisão (3)

- O tipo declarado de uma variável é seu tipo estático.
  - **Compiladores verificam os tipos estáticos.**
- O tipo de um objeto é seu tipo dinâmico.
  - **Tipos dinâmicos são utilizados em tempo de execução.**
- Métodos podem ser sobrescritos em uma subclasse.
- A pesquisa de método inicia com o tipo dinâmico.
- O acesso protegido suporta herança.