CENIN

Curso Java Básico

Uma introdução prática usando BlueJ

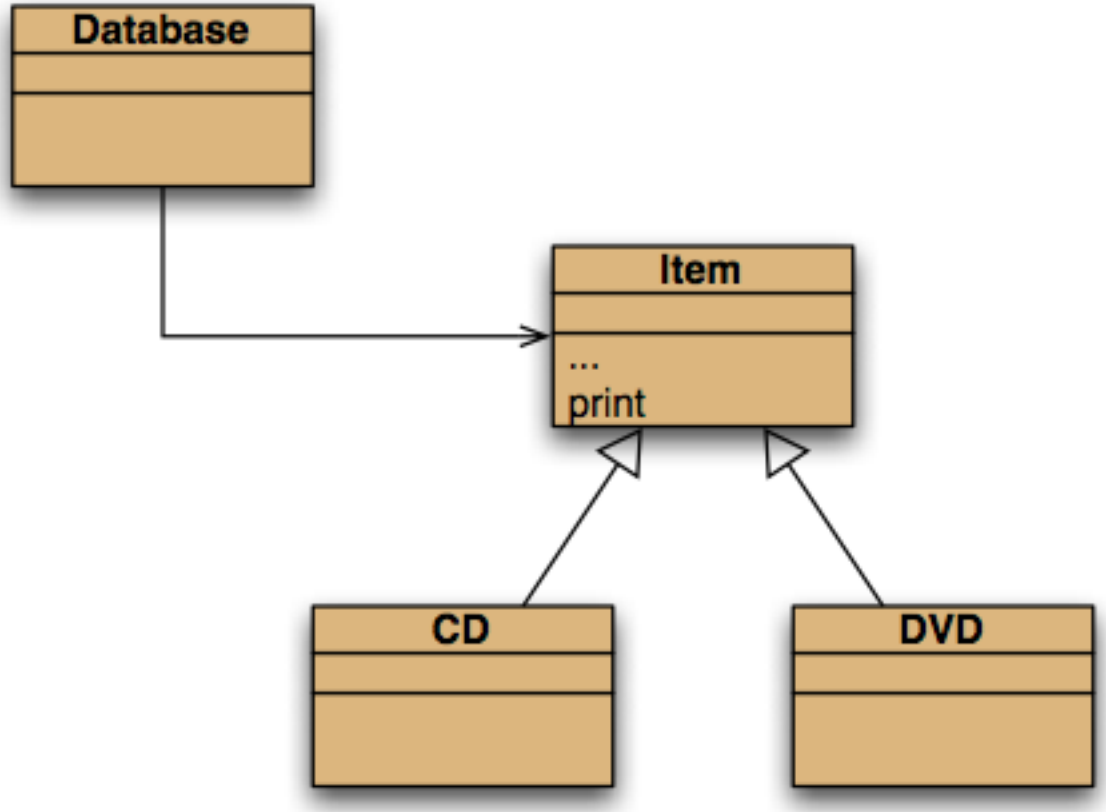
Mais sobre herança

Explorando o polimorfismo

Principais conceitos a serem abordados

- Tipo estático e dinâmico
- Sobrescrição de métodos
- Escolha dinâmica de método
- Polimorfismo de método
- Controle de acesso a membros de classe
- A classe Object

A hierarquia da herança



Saída conflitante

O que queremos

```
CD: A Swingin' Affair (64 mins)*
    Frank Sinatra
    tracks: 16
    my favourite Sinatra album

video: The Matrix (136 mins)
    Andy & Larry Wachowski
    must see if interested in virtual reality!
```

O que temos

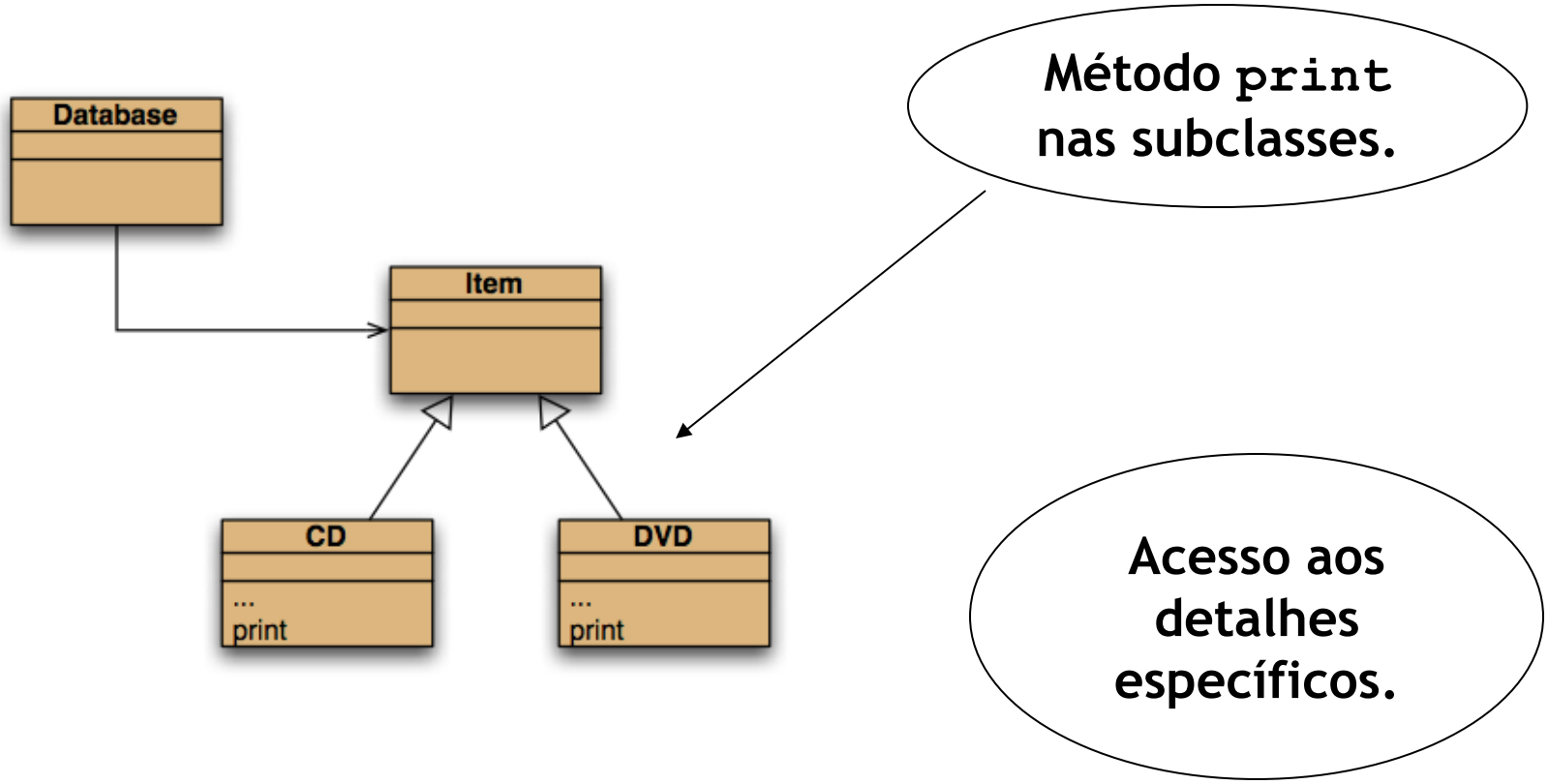
```
title: A Swingin' Affair (64 mins)*
    my favourite Sinatra album

title: The Matrix (136 mins)
    must see if interested in virtual reality!
```

O problema

- O método `print` em `Item` imprime apenas os campos comuns.
- Herança é uma avenida de mão única:
  - uma subclasse herda os campos da superclasse; e
  - a superclasse não sabe nada sobre os campos da sua subclasse.

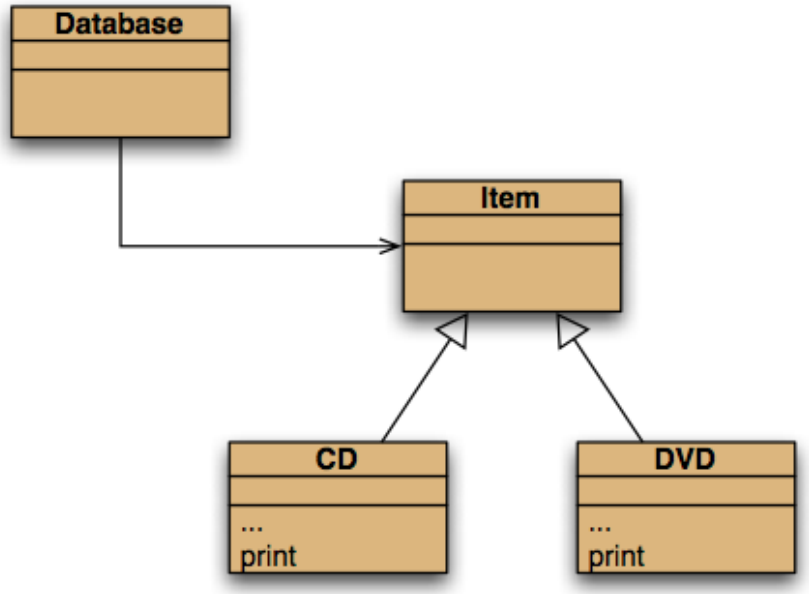
Tentando resolver o problema



Exercício

- **Database multimídia**
  - Feche o projeto anterior, abra o projeto *dome-v3*.
  - Edite a classe *Item* e comente o método *print*.
  - Edite as classes *CD* e *DVD* e copie o método *print* de *Item* para elas.
  - Compile o projeto. O que ocorre ?

Tentando resolver o problema



- Cada subclasse tem sua própria versão do método **`print`**.
- Mas campos de **`Item`** são privados (1) e
- **`Database`** não pode encontrar um método **`print`** em **`Item`** (2),
- apesar de que todo **`Item`** é um **`CD`** ou **`DVD`**.

Tipo estático e dinâmico

- 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

Qual é o tipo de c1?

```
Car c1 = new Car();
```

Qual é o tipo de v1?

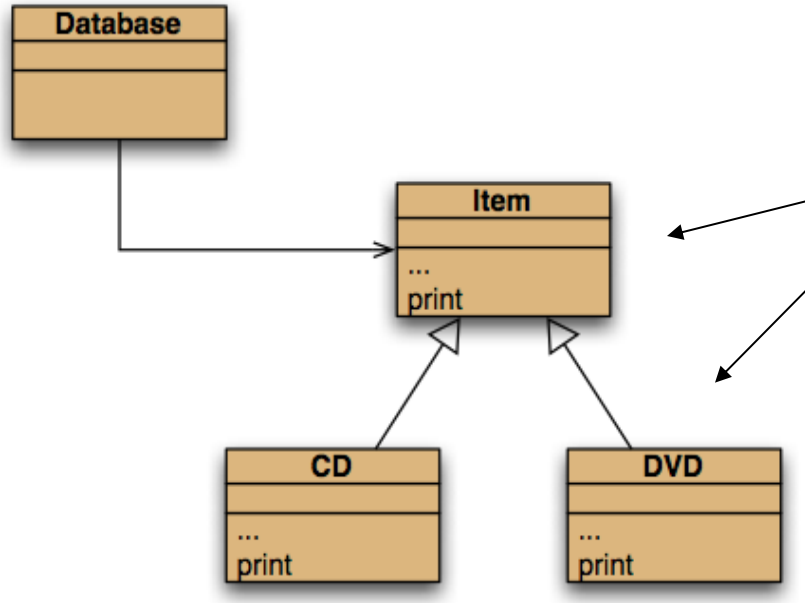
```
Vehicle v1 = new Car();
```

Tipo estático e dinâmico

- 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.

```
for(Item item : items) {
    item.print();   // Erro compilação.
}
```

Sobrescrever: a solução



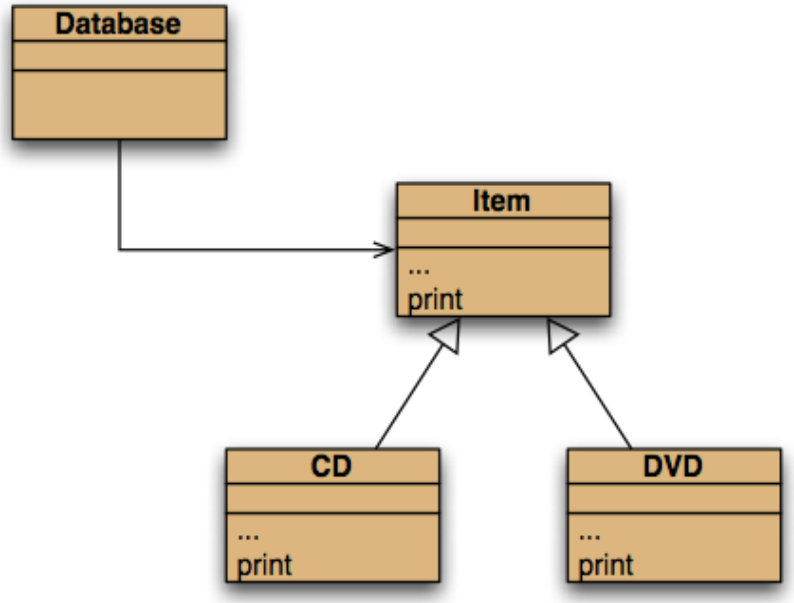
Método `print` tanto na superclasse como nas subclasses.

Satisfaz a verificação do tipo estático e o acesso dinâmico aos detalhes.

Exercício

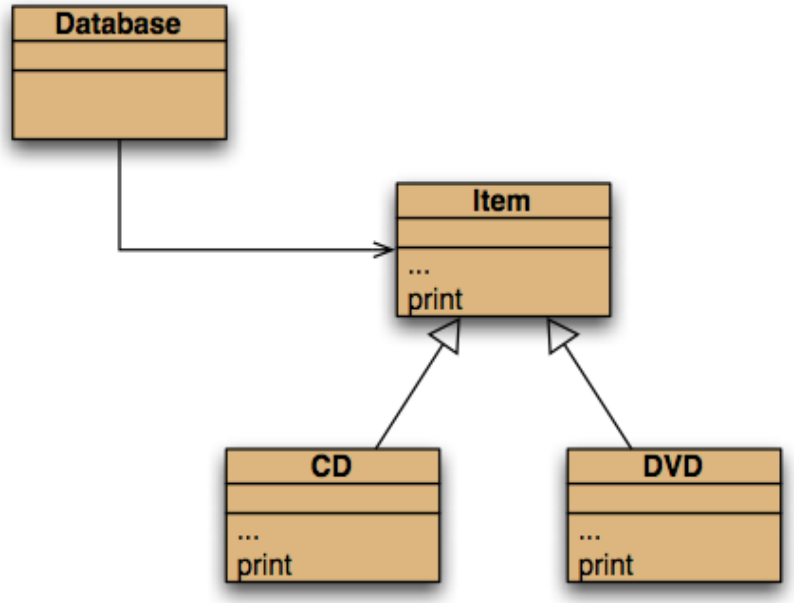
- **Database multimídia**
  - Edite a classe *Item* e desfaça o comentário do método *print*.
  - Edite as classes *CD* e *DVD* e deixe no método *print* apenas a impressão dos campos específicos.
  - Crie uma instância de *Database* e insira um *CD* e um *DVD*.
  - Quais métodos *print* serão chamados ao executarmos o *list* de *Database* ? Execute este método. Sua previsão estava correta ?

Sobrescrever: a solução



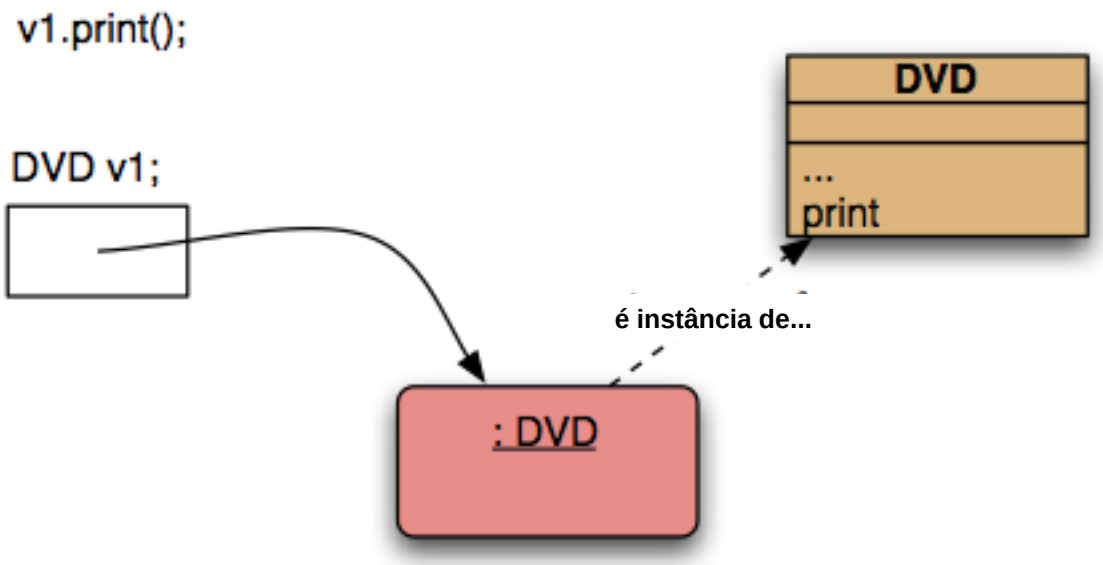
- A superclasse e cada subclasse tem sua própria versão do método **`print`**.
- Cada método tem acesso aos campos da sua classe.
- A superclasse satisfaz a verificação do tipo estático.
- As subclasses têm o método chamado na execução.

Sobrescrever: a solução



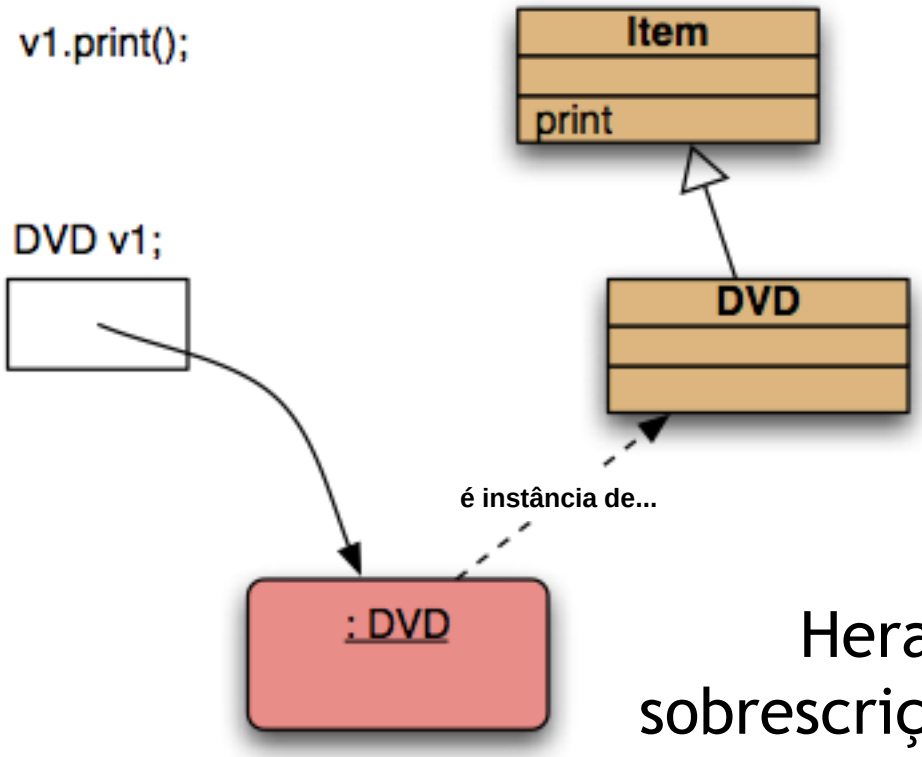
- O método da subclasse é chamado em tempo de execução — ele *sobrescreve* a versão da superclasse.
- Assim, o método da superclasse *não* é executado.

Pesquisa de método



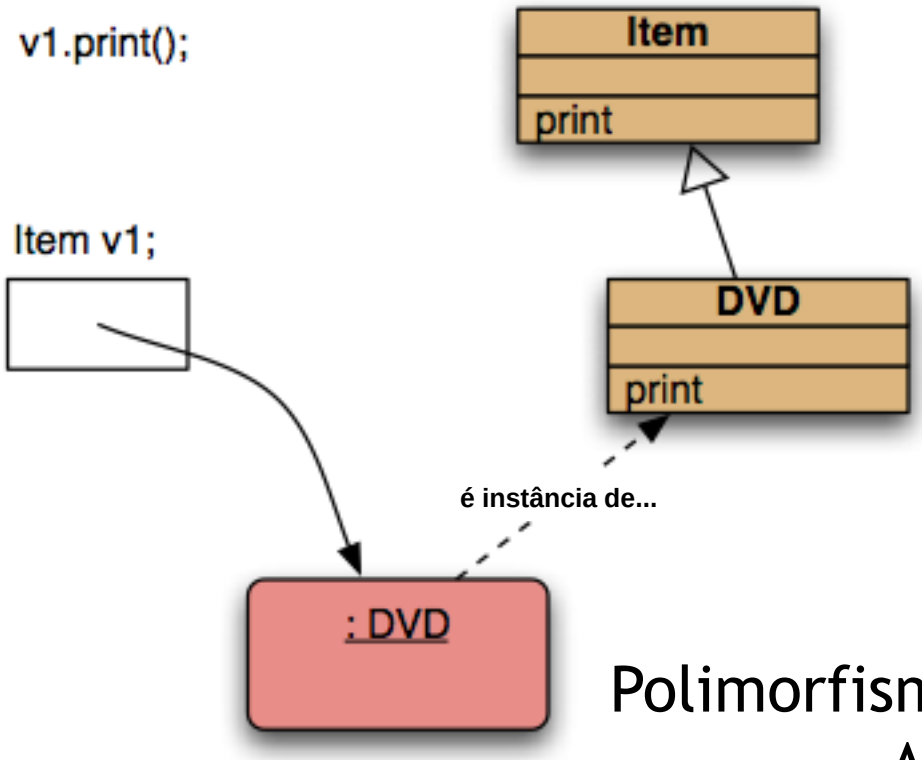
Nenhuma herança ou polimorfismo.
O método óbvio é selecionado.

Pesquisa de método



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

Pesquisa de método



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

Pesquisa de método

1. A variável **`v1`** é acessada.
2. O objeto armazenado na variável é encontrado.
3. A classe do objeto é encontrada.
4. É pesquisada na classe uma correspondência de método.
5. Se nenhuma correspondência for encontrada, a superclasse é pesquisada.
6. Isso é repetido até que uma correspondência seja encontrada ou a hierarquia da classe seja exaurida.

Chamando um método sobrescrito

- Métodos sobrescritos são ocultados...
- ... mas, frequentemente, queremos ser capazes de chamá-los.
- Um método sobrescrito *pode* ser chamado a partir do método que o sobrescreve:
  - `super.nomeDoMetodo( parâmetros )`
- Compare com a utilização de `super()` nos construtores.

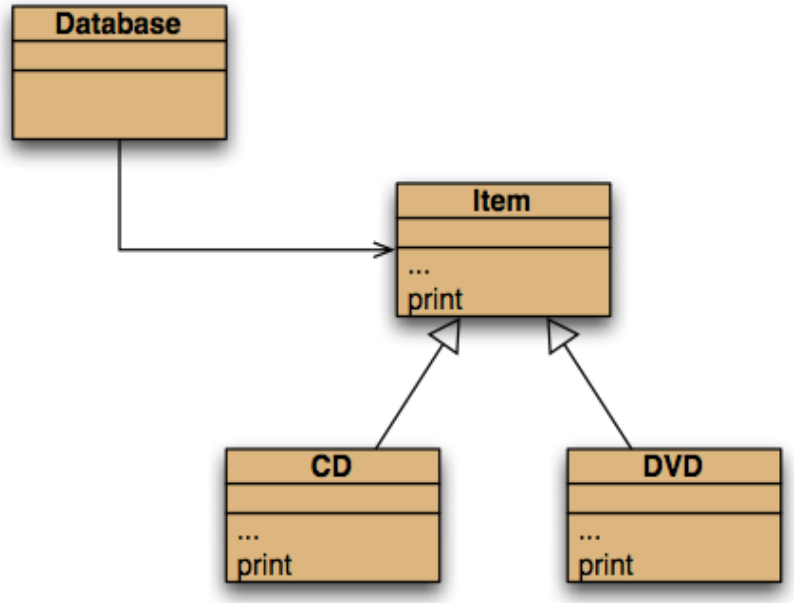
Chamando um método sobrescrito

```
public class CD
{
    ...
    public void print()
    {
        super.print();
        System.out.println("    " + artist);
        System.out.println("    tracks: " +
                           numberOfTracks);
    }
    ...
}
```

Exercício

- **Database multimídia**
  - Edite as classes *CD* e *DVD* e inclua nos seus métodos *print* uma chamada ao método sobrescrito.
  - Crie uma instância de *Database* e insira um *CD* e um *DVD*.
  - Quais métodos *print* serão chamados ao executarmos o *list* de *Database* ? Execute este método. Sua previsão estava correta ?
  - Compare o estágio atual do projeto em uso com *dome-v4*.

Sobrescrever: a solução



- O método da subclasse é chamado em tempo de execução — ele *sobrescreve* a versão da superclasse, mas chama o método sobrescrito.
- Assim, o método da superclasse é executado.

Polimorfismo de método

- Discutimos o *encaminhamento de método polimórfico*.
- Uma variável polimórfica pode armazenar objetos de diferentes tipos.
- Chamadas de método são polimórficas:
  - o método realmente chamado depende do tipo dinâmico do objeto;
  - `item.print();` // pode chamar print de CD ou DVD

Controle de acesso

- Dicas para controle de acesso de membros:
  - Use o mais restritivo nível de acesso que faça sentido para o membro;
  - Evite campos públicos, exceto para constantes.

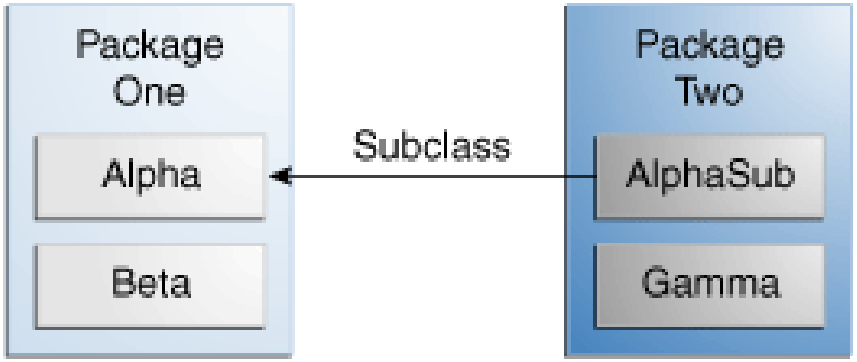
Modificadores de acesso

- Em Java, há dois níveis de controle de acesso:
  - Top level:
    - *public*, or *package-private* (no explicit modifier);
  - Member level:
    - *public*, *private*, *protected*, or *package-private*:

Access Levels

Modifier	Class	Package	Subclass	World
`public`	Y	Y	Y	Y
`protected`	Y	Y	Y	N
no modifier	Y	Y	N	N
`private`	Y	N	N	N

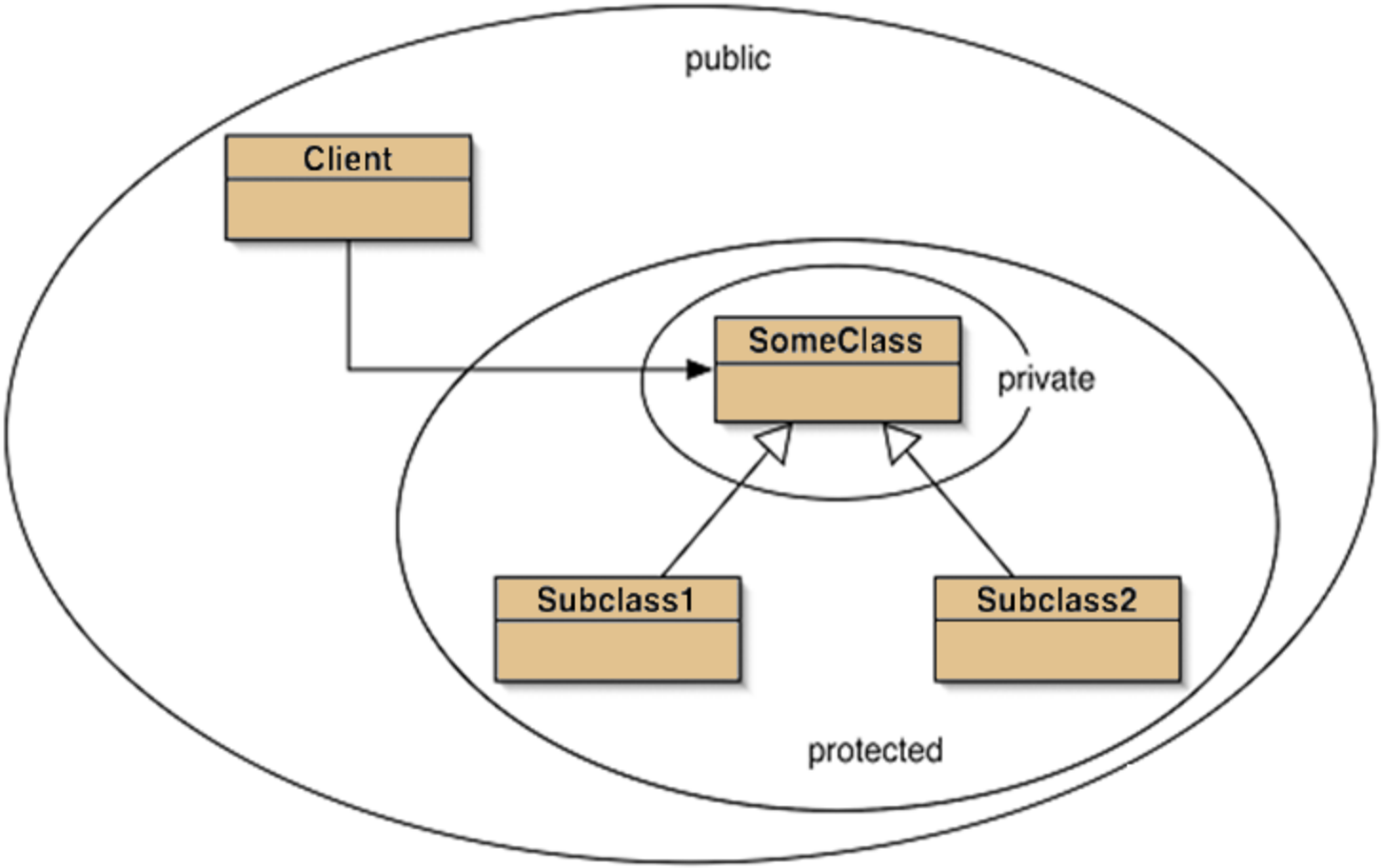
Modificadores de acesso



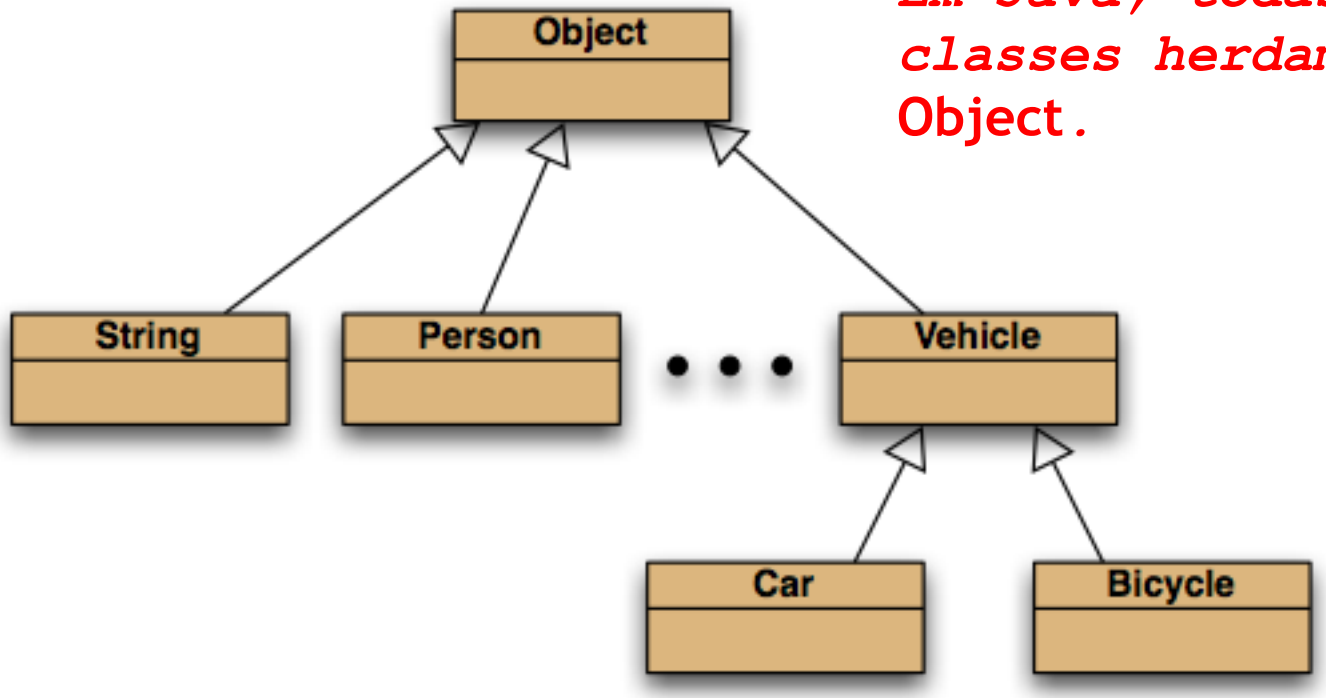
Visibility

Modifier	Alpha	Beta	Alphasub	Gamma
`public`	Y	Y	Y	Y
`protected`	Y	Y	Y	N
no modifier	Y	Y	N	N
`private`	Y	N	N	N

Níveis de acesso



A classe Object



***Em Java, todas as classes herdam de* Object.**

Os métodos da classe `Object`

- Métodos em `Object` são herdados por todas as classes.
- Qualquer um dos não finais pode ser sobrescrito.
- O método `toString` retorna uma representação de *string* do objeto e é comumente sobrescrito.

Exercício

- **Database multimídia**
  - Experimente chamar o método *toString*, herdado de Object, em instâncias de *CD* e de *DVD*. O que você observa na implementação padrão de *toString* ?
  - Como poderíamos sobrescrever *toString* para torná-lo útil ao DoME ? Qual seria a utilidade ? Quais classes deveriam sobrescrevê-lo ?

Sobrescrevendo `toString`

```
public class Item
{
    ...
    public String toString()
    {
        String line1 = title +
                       " (" + playingTime + " mins)";
        if(gotIt) {
            return line1 + "*\n" + "    " +
                   comment + "\n";
        } else {
            return line1 + "\n" + "    " +
                   comment + "\n";
        }
    }
    ...
}
```

Usando `toString`

- Frequentemente, métodos `print` explícitos podem ser omitidos de uma classe:
  - `System.out.println(item.toString());`
- Chamadas a `println` com apenas um objeto como parâmetro fazem com que `toString` seja chamado automaticamente:
  - `System.out.println(item);`

Usando `toString`

```
public class Database
{
    ...

    public void list()
    {
        for (Item item : itens) {
            System.out.println(item);
        }
    }
    ...
}
```

Exercício

- **Database multimídia**
  - Substitua os métodos *print* em *Item*, *CD* e *DVD* por métodos *toString*.
  - Altere a classe Database para que chame *toString* ao invés de *print*.
  - Crie uma instância de *Database* e insira um *CD* e um *DVD*.
  - Chame o método *list* de *Database*. O comportamento é o esperado ?
  - Compare o estágio atual do projeto em uso com *dome-v5*.

Exercício

- **Database multimídia**
  - Como poderíamos ter o comentário de cada item apresentado na última linha ? Faça as alterações necessárias.
  - Compare o estágio atual do projeto em uso com *dome-v6*.

Os métodos da classe `Object`

- Muitas vezes é necessário determinar se dois objetos são '*os mesmos*', o que pode ser duas coisas diferentes:
  - A igualdade de referência ou identidade (mesmo objeto) é testada usando o operador ==.
  - A igualdade lógica (mesmo conteúdo) é testada pelo método `equals` da classe `Object`, que é comumente sobrescrito.

Sobrescrevendo `equals`

```
public class Student
{
    private String name;
    private String id;
    private int credits;

    // Construtores e métodos omitidos.
}
```

Sobrescrevendo `equals`

```
public boolean equals(Object obj)
{
    if (this == obj) {
        return true;
    }
    if (!(obj instanceOf Student)) {
        return false;
    }
    Student other = (Student) obj;
    return name.equals(other.name) &&
        id.equals(other.id) &&
        credits == other.credits;
    }
    ...
}
```

Igualdade de referência

Classes diferentes

Conversão segura

Compara conteúdo dos campos

Sobrescrevendo `equals`

```
public boolean equals(Object obj)
{
    if (this == obj) {
        return true;
    }
    if (!(obj instanceOf Student)) {
        return false;
    }
    Student other = (Student) obj;
    return id.equals(other.id);
    }
    ...
}
```

Id é único

Os métodos da classe `Object`

- O método `hashCode` retorna uma representação *numérica* do objeto.
- Sempre que o método `equals` é sobrescrito, o método `hashCode` também deve ser sobrescrito:
  - dois objetos idênticos, a partir de uma chamada a `equals`, também devem retornar valores idênticos a partir de chamadas a `hashCode`.

Sobrescrevendo `hashCode`

```
public int hashCode()
{
    final int prime = 31;
    int result = 1;
    result = prime * result + name.hashCode();
    result = prime * result + id.hashCode();
    result = prime * result + credits;
    return result;
}
```

Sobrescrevendo `hashCode`

```
public int hashCode()
{
    final int prime = 31;
    int result = 1;
    result = prime * result + id.hashCode();
    return result;
}
```


Id é único

Revisão

- O tipo declarado de uma variável é seu tipo estático.
  - Tipos estáticos são utilizados em tempo de compilação.
- 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.

Contatos

Câmara dos Deputados

CENIN - Centro de Informática

Carlos Renato S. Ramos

carlosrenato.ramos@camara.gov.br