

Relacionamentos entre objetos

4



Associação, composição e agregação
Construtores e sobrecarga
Variáveis de classe e de instância

Reinaldo Gomes
reinaldo@cefet-al.br

Relacionamentos entre objetos

- Objetos do mundo real relacionam-se uns com os outros de diversas formas:
 - Um objeto motor *é parte de* um objeto carro
 - Um objeto turma *tem vários* objetos alunos
 - Um objeto botão *tem um* objeto tratador de eventos
- Tipos de associações entre objetos de software:
 - *Agregação*: estabelecem um vínculo entre objetos
 - *Composição*: relacionamento do tipo *todo/parte*.
 - *Uso*: um objeto usa a funcionalidade de outro sem estabelecer vínculo duradouro (referências)

Relacionamentos entre objetos



Copyright 1991, Grady Booch

3

Relacionamentos entre objetos

■ *Agregação*

- Forma de composição em que o objeto composto apenas usa ou tem conhecimento da existência do(s) objeto(s) componente(s)
- Os objetos componentes podem existir sem o agregado e vice-versa.

■ *Composição*

- Forma de associação em que o objeto composto é responsável pela existência dos componentes
- O componente não tem sentido fora da composição

4

Relacionamentos entre objetos

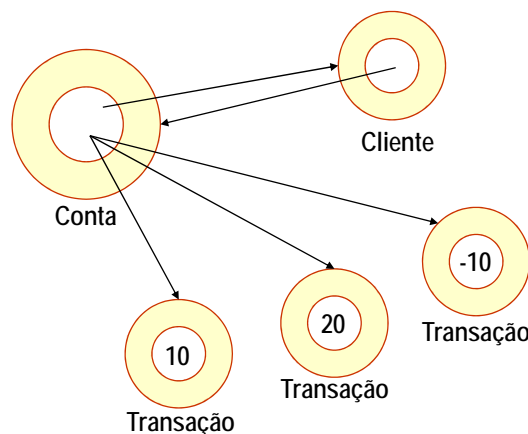
■ Exemplos:

- Uma conta corrente é formada por várias transações de crédito e débito → *composição*
- Um cadastro de clientes é formado por vários clientes → *agregação*
- Um cliente tem uma conta-corrente → *agregação*
- Um documento possui um conjunto de parágrafos → *composição*
- Uma turma é um conjunto de alunos → *composição*

5

Relacionamentos entre objetos

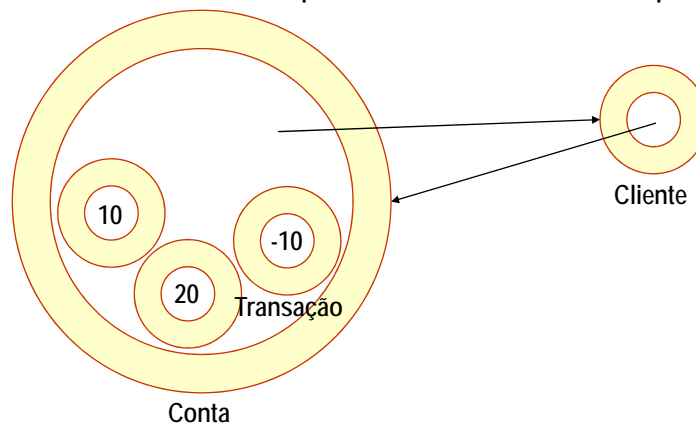
■ Conta bancária implementada como agregação:



6

Relacionamentos entre objetos

- Conta bancária implementada como composição:



7

Relacionamentos entre objetos

- Em Java:

- Java não possui uma forma declarativa para implementar **agregações** nem **associações**
- Java apenas cria **associações unidirecionais** através de referências. Exemplo:
 - Uma Conta tem uma propriedade do tipo Cliente → referência de conta para cliente, **mas não** de cliente para conta

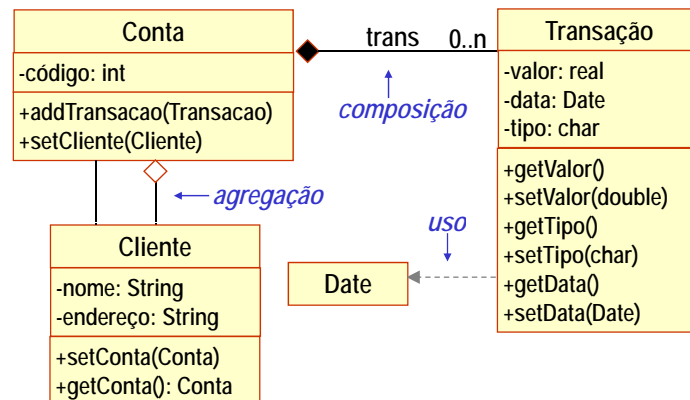
Associação
unidirecional

```
public class Conta {  
    private int codigo;  
    private Cliente cliente;  
    //...  
}
```

8

Relacionamentos entre objetos

- Notação UML para definir composições, agregações e uso:



9

Relacionamentos entre objetos

- Implementação em Java:

```

public class Conta {
    private int codigo;
    private Cliente cliente;
    private Transacao[] trans;
    private int qTransacoes;
    //...
}

public class Cliente {
    private String nome;
    private Conta conta;
    private String endereço;
    //...
}

public class Transacao {
    private double valor;
    private char tipo;
    private Date data;
    //...
}
    
```

10

Relacionamentos entre objetos

■ Em Java:

- Estabelece-se uma associação *unidirecional* entre objetos definindo, na classe do objeto originário, uma propriedade cujo tipo é a classe do objeto destinatário
- **Lembre-se!!!** Ao instanciar um objeto da classe de onde parte a associação você não estará criando uma instância do objeto destinatário da associação
 - Ao criar um objeto da classe Conta, você não criará um objeto da classe Cliente ou vários objetos da classe Transação
 - Métodos em ambas as classes deverão concretizar a agregação/composição em tempo de execução

11

Relacionamentos entre objetos

■ Em alguma classe de aplicação:

```
public static void main(String[] args) {  
    Conta cc5 = new Conta();  
    Cliente cliente = new Cliente();  
    Transacao credito = new Transacao();  
  
    cc5.setCodigo(100);  
    cc5.setCliente(cliente);  
    cliente.setConta(cc5);  
    credito = new Transacao();  
    credito.setValor(10000000.0);  
    credito.setTipo('C');  
    credito.setData(new Date());  
    cc5.addTransacao(credito);  
}
```

Concretiza a **agregação**
entre cliente e conta

Concretiza a **composição**
entre transação e conta

12

Relacionamentos entre objetos

■ Na classe Conta:

```
public class Conta {  
    private int codigo;  
    private Cliente cliente;  
    private Transacao[] trans;  
    private int qTransacoes=0;  
  
    public void setCliente(Cliente c) {  
        cliente = c;  
    }  
  
    public void addTransacao(Transacao t) {  
        trans[qTransacoes++] = t;  
    }  
}
```

Implementação muito simplista!

Outros possíveis métodos da classe foram omitidos.

13

Relacionamentos entre objetos

■ Na classe Cliente:

```
public class Cliente {  
    private String nome;  
    private Conta conta;  
    private String endereco;  
  
    public void setConta(Conta c) {  
        conta = c;  
    }  
}
```

- Como o relacionamento entre Cliente e Conta é bidirecional, cada uma das classes possui uma propriedade que é do tipo da outra!

14

Relacionamentos entre objetos

- Um método para devolver o saldo da conta:

```
public class Conta {  
    //...  
    public double getSaldo() {  
        double total = 0.0;  
        for (int i=0; i<qTransacoes ;i++) {  
            if (trans[i].getTipo() == 'C')  
                total += trans[i].getValor();  
            if (trans[i].getTipo() == 'D')  
                total -= trans[i].getValor();  
        }  
        return total;  
    }  
}
```

15

Relacionamentos entre objetos

- A composição do mundo real é conseguida por meio de um processo chamado de montagem
 - Os objetos são unidos por meio de interfaces físicas bem definidas (o cano do guidon deve encaixar perfeitamente no tubo do quadro, o parafuso do pneu dianteiro deve encaixar no garfo do quadro, etc..)
- As operações enviadas ao objeto composto disparam operações dos seus objetos componentes

16

Relacionamentos entre objetos

■ Exemplo de relacionamento de uso:

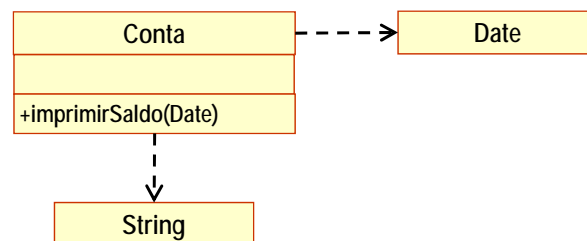
```
public class Conta {  
    private int codigo;  
    private Cliente cliente;  
    private Transacao[] trans;  
    private int qTransacoes=0;  
  
    public void imprimeSaldo(Date data) {  
        String s;  
        ...  
    }  
}
```

Conta **usa** as classes String e Date

17

Relacionamentos entre objetos

■ Exemplo de relacionamento de uso:



18

Relacionamentos entre objetos

■ Mais um exemplo:

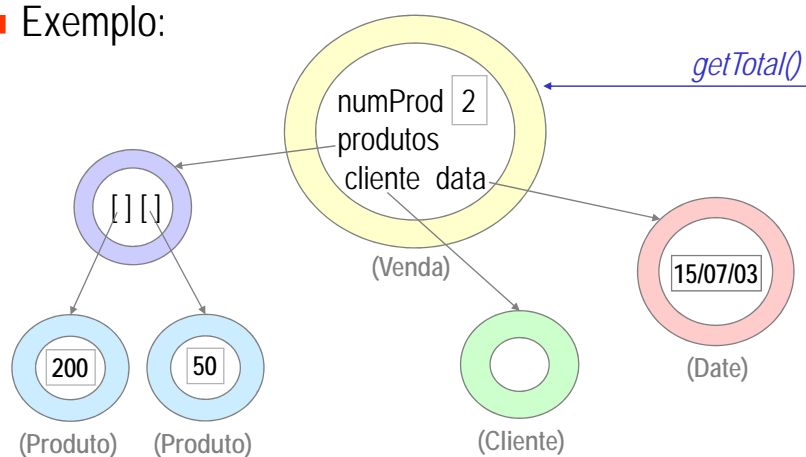
```
class Venda {  
    private Cliente cliente;  
    private Produto[] produtos;  
    private Date data;  
    private int numProd;  
    public float getTotal() {...}  
}
```

```
class Produto {  
    private float preco;  
    public float getTotal() {return preco;}  
}
```

19

Relacionamentos entre objetos

■ Exemplo:



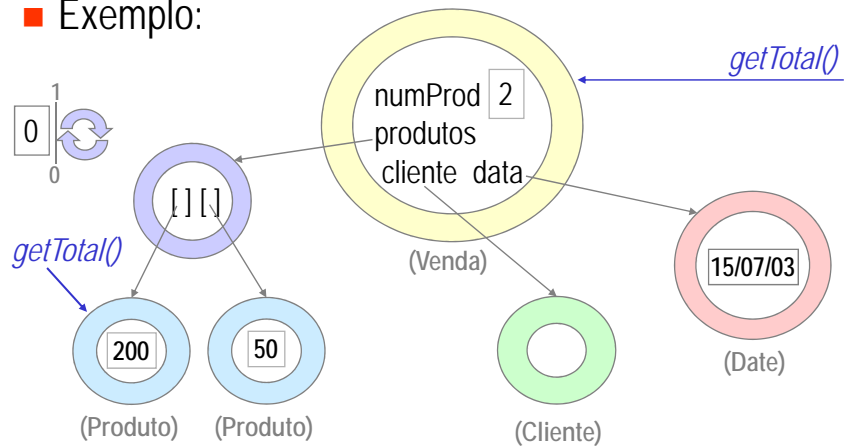
Em `getTotal()` →

```
total += produtos[i].getTotal();
```

20

Relacionamentos entre objetos

Exemplo:



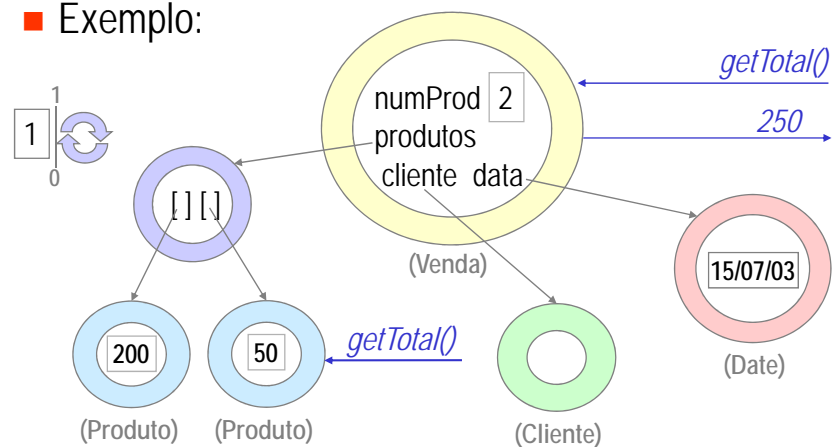
Em `getTotal()` →

```
total += produtos[0].getTotal();
```

21

Relacionamentos entre objetos

Exemplo:



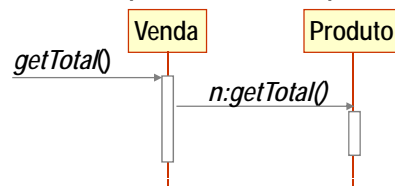
Em `getTotal()` →

```
total += produtos[1].getTotal();
```

22

Relacionamentos entre objetos

- O objeto Venda *delega* para os objetos Produto a tarefa de informar o preço para que a operação getTotal() possa ser implementada.



- De forma análoga, poderia ser implementada a operação getMesVenda(); só que desta vez, o objeto delegado seria a data.

23

Relacionamentos entre objetos

- Resumo:
 - Os relacionamentos que *objetos* assumirão em *tempo de execução* são definidos nas suas respectivas *classes* em *tempo de compilação*

24

Construtores

- A instanciação de um objeto:
 - Aloca área em memória para as propriedades do objeto
 - Põe valores iniciais para as propriedades, de acordo com seus tipos (0 para números, string vazio, null, etc)
- *Construtores* de objetos
 - Método especial que define valores iniciais para as propriedades de um objeto recém-instanciado
 - Só é usado no momento da instanciação do objeto
 - Em Java, os construtores devem ter o mesmo nome da classe e não possuem tipo de retorno

25

Construtores

- Um construtor para a classe Cliente:

```
public class Cliente {  
    private String nome;  
    private Conta conta;  
    private String endereço;  
  
    public Cliente(String n, String e, Conta c) {  
        nome = n;  
        endereço = e;  
        conta = c;  
    }  
  
    //outros métodos de Cliente  
}
```

26

Construtores

■ Instanciando um cliente "sem construtor":

```
public static void main(String[] args) {  
    Cliente c;  
    Conta cc5;  
  
    cc5 = new Conta();  
    cc5 = setCodigo(100);  
  
    [c = new Cliente();  
    c.setNome("Malug");  
    c.setConta(cc5);  
    c.setEndereco("São Paulo");  
    ]  
}
```

O método main acima está em alguma classe de aplicação.

27

Construtores

■ Instanciando um cliente com o construtor anterior:

```
public static void main(String[] args) {  
    Cliente c;  
    Conta cc5;  
  
    cc5 = new Conta();  
    cc5 = setCodigo(100);  
  
    [c = new Cliente("Malug", cc5, "São Paulo");  
    ]  
}
```

O método main acima está em alguma classe de aplicação.

28

Construtores

■ Um construtor para a classe Conta:

```
public class Conta {  
    private int codigo;  
    private Cliente cliente;  
    private Transacao[] trans;  
    private int qTransacoes;  
  
    public Conta(int cod, Cliente c, int maxTran) {  
        codigo = cod;  
        cliente = c;  
        trans = new Transacao[maxTran];  
        qTransacoes = 0;  
    }  
    //outros métodos de Conta  
}
```

29

Resumo de construtores

■ Construtores em Java:

- Construtores não possuem tipo de retorno
- Construtores devem ter o mesmo nome da classe
- Construtores podem ser públicos ou privados
- Construtores só são usados durante a instanciação
- Java já oferece para todas as classes um *construtor padrão* sem parâmetros e com implementação vazia
 - Se você não definir nenhum construtor na classe poderá usar esse construtor vazio na instanciação de objetos

30

Sobrecarga

- Métodos em Java podem ser *sobrecarregados*
 - Métodos com mesmo identificador mas com listas de parâmetros diferentes (quantidade ou tipos)
- Vantagens:
 - Permitir que uma mesma operação possa ser executada com diferentes parâmetros mas mantenha o mesmo nome
 - Melhora a legibilidade do código, pois o programador não precisa aprender os diferentes nomes que uma operação pode ter

31

Sobrecarga

- Sobrecarregando o método `getSaldo()` em Conta:

```
// calcula o saldo com todas as transacoes
public double getSaldo() {
    double total = 0.0;
    for (int i=0; i<qTransacoes ;i++) {
        if (trans[i].getTipo() == 'C')
            total += trans[i].getValor();
        if (trans[i].getTipo() == 'D')
            total -= trans[i].getValor();
    }
    return total;
}
```

32

Sobrecarga

■ Segunda implementação de getSaldo():

```
// calcula o saldo até uma determinada data
public double getSaldo(Date data) {
    double total = 0.0;
    for (int i=0; i<qTransacoes ;i++) {
        if (trans[i].getTipo() == 'C' &&
            trans[i].getDate().before(data))
            total += trans[i].getValor();
        if (trans[i].getTipo() == 'D' &&
            trans[i].getDate().before(data))
            total -= trans[i].getValor();
    }
    return total;
}
```

33

Sobrecarga

■ Sobrecarregando o construtor de Cliente:

```
public class Cliente {
    // propriedades omitidas

    [ public Cliente(String n, String e, Conta c) {
        nome = n;
        endereço = e;
        conta = c;
    }

    [ public Cliente(String n, String e) {
        nome = n;
        endereço = e;
    }
    // outros métodos
}
```

34

Resumo de sobrecarga

■ Sobrecarga em Java:

- Métodos sobrecarregados devem possuir o mesmo identificador, rigorosamente
- O tipo de retorno de um método *não pode* ser levado em conta para diferenciar métodos sobrecarregados
- A lista de parâmetros reais da mensagem define qual versão do método sobrecarregado será chamada
- Construtores podem ser sobrecarregados
- A sobrecarga de métodos também é conhecida por *polimorfismo paramétrico*

35

Variáveis de instância e de classe

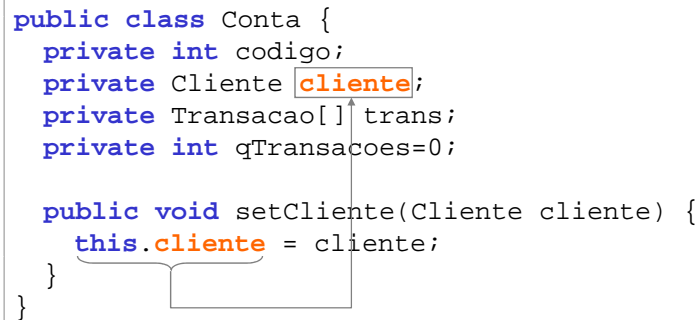
- Todas as propriedades definidas em uma classe serão criadas quando objetos desta classe forem instanciados
 - São também chamadas de *variáveis de instância*
 - Cada objeto tem suas próprias cópias particulares das variáveis de instância
 - Elas não são compartilhadas entre objetos distintos (a menos que seu encapsulamento seja **public**)
 - Nos métodos, a palavra *this* é usada para referenciar as variáveis (e métodos) de instância do objeto atual

36

Variáveis de instância e de classe

■ Uso do *this* em Java:

```
public class Conta {  
    private int codigo;  
    private Cliente cliente;  
    private Transacao[] trans;  
    private int qTransacoes=0;  
  
    public void setCliente(Cliente cliente) {  
        this.cliente = cliente;  
    }  
}
```



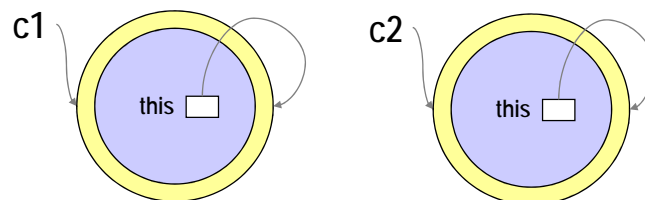
- O *this* é uma forma genérica de referenciar o objeto que receberá a chamada setCliente()

37

Variáveis de instância e de classe

■ Em algum método de alguma classe que usa Conta:

```
Conta c1 = new Conta();  
Conta c2 = new Conta();  
  
c1.setCliente(new Cliente());  
c2.setCliente(new Cliente());
```



38

Variáveis de instância e de classe

- Algumas informações, entretanto, não pertencem a uma instância particular, mas a todas elas.
 - Variáveis compartilhadas entre todas as instâncias são chamadas de *variáveis de classe*
 - Existe apenas uma cópia dela que fica na classe (em memória)
 - Todos os objetos da classe podem acessar/modificar as variáveis de classe

39

Variáveis de instância e de classe

- Exemplo:
 - A alíquota da CPMF é a mesma para todas as contas
 - Define-se esta informação numa *variável de classe*

```
public class Conta {  
    private int codigo;  
    private Cliente cliente;  
    private Transacao[] trans;  
    private int qTransacoes=0;  
  
    [ public static double CPMF = 0.0038;  
      ↑  
    ]  
}
```

Outros possíveis métodos da classe foram omitidos.

40

Variáveis de instância e de classe

■ Variáveis de classe em Java:

- São declaradas com a palavra reservada **static**
- Na própria classe, são acessadas diretamente
- Fora da classe, são acessadas antepondo-se a ela o nome da classe e o operador "." (ponto)

Exemplos de acessos fora da classe:



```
Conta cc5 = new Conta();  
double cpmf;  
cpmf = cc5.getSaldo() * cc5.CPMF;
```

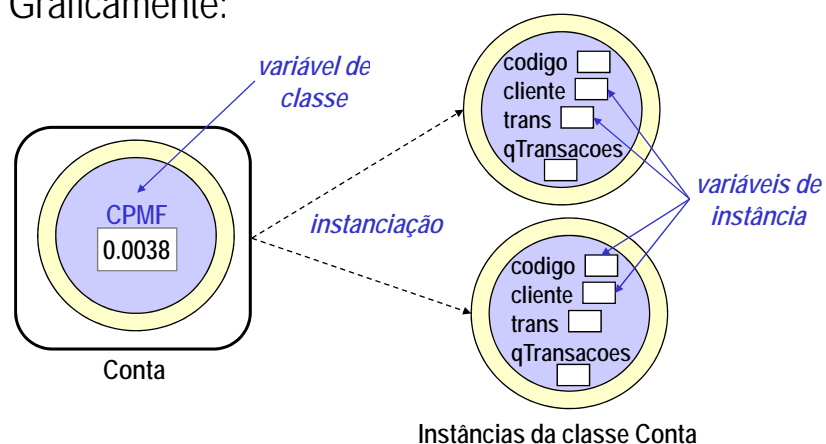


```
Conta cc5 = new Conta();  
double cpmf;  
cpmf = cc5.getSaldo() * Conta.CPMF;
```

41

Variáveis de instância e de classe

■ Graficamente:



42

Variáveis de classe

■ Em Java:

- Variáveis de classe são muito utilizadas para declarar constantes comuns a todas as instâncias
- Se uma variável de classe for pública, todas as outras classes podem acessá-la através da notação geral **Classe.variável**
- Dentro da classe onde está declarada, a forma de acessar a variável de classe é indiferente
 - Deve-se sempre preferir a notação acima

43

Relacionamentos entre objetos

4



Associação, composição e agregação
Construtores e sobrecarga
Variáveis de classe e de instância