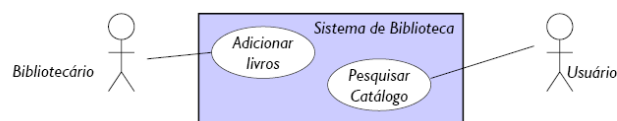


# Introdução a ORIENTAÇÃO A OBJETOS

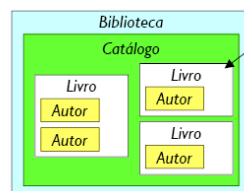
Mário Meireles Teixeira  
DEINF - UFMA  
mario@deinf.ufma.br

## Visão OO (1) e Visão Procedural (2)



(1) Trabalha no **espaço do problema** (casos de uso simplificados em objetos)

- Abstrações mais simples e mais próximas do mundo real

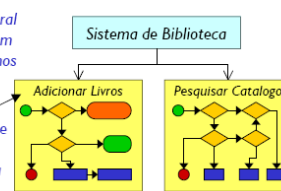


Lógica procedural encapsulada em objetos pequenos

Lógica exposta e espalhada por todo o sistema

(2) Trabalha no **espaço da solução** (casos de uso decompostos em procedimentos algorítmicos)

- Abstrações mais próximas do mundo do computador



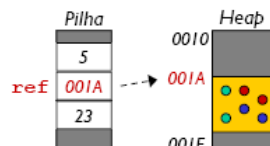
## O que é um objeto?

- Um **objeto** é uma entidade (física, conceitual ou do domínio do problema) que possui:
  - identidade (implícita)
  - estado
  - comportamento
- Estado:
  - Características ou propriedades do objeto
  - Expresso pelos **atributos**
    - programação estruturada → variáveis
- Comportamento:
  - O que o objeto sabe fazer ou se pode fazer com ele
  - Definido através dos **métodos**
    - prog. estruturada → funções, procedimentos, ...

3

## O que é um objeto?

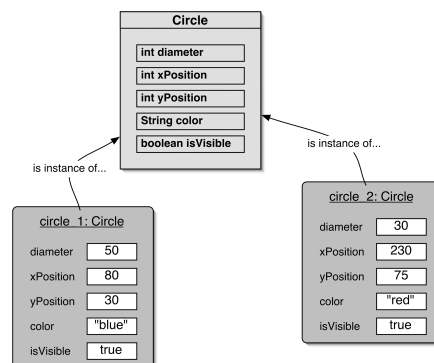
- Em uma linguagem orientada a objetos pura (Smalltalk):
  - Um número, uma letra, uma palavra, um valor booleano, uma data, um registro, um botão da interface são objetos
- Valores unidimensionais (tipos primitivos) não são objetos em Java:
  - Números, booleanos, caracteres são armazenados na pilha
  - Têm apenas identidade (nome da variável) e estado (valor armazenado na variável)
  - - dinâmicos; + rápidos
- Em Java, objetos são armazenados no *heap* e manipulados através de uma referência (variável), guardada na pilha:
  - Têm **identidade** (a referência)
  - Têm **estado** (seus atributos)
  - Têm **comportamento** (seus métodos)



4

## Exemplo: dois objetos Círculo

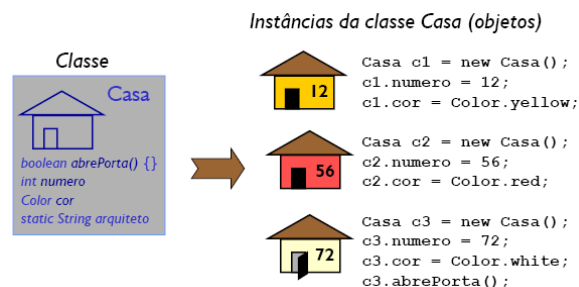
- A classe **Circle** define um tipo
- Os objetos são criados de acordo com as especificações da classe
- Cada objeto **circle1**, **circle2** é independente (e distinto) do outro
- **Classe**: tipo, especificação  
**Objeto**: instância da classe



5

## O que é uma classe?

- Classes são uma especificação para objetos com **propriedades** semelhantes (atributos), **comportamentos** semelhantes (métodos) e **relacionamentos** comuns com outros objetos
- Uma classe representa um tipo de dados complexo
- Classes descrevem:
  - Tipos dos dados que compõem o objeto (o que podem armazenar)
  - Procedimentos que o objeto pode executar (o que podem fazer)



6

## O que é uma classe?

- Classes definem uma especificação (lógica) estática. Objetos são dinâmicos e podem mudar durante a execução do programa
- Classes não existem no contexto de execução:
  - Uma classe representa vários objetos que ocupam espaço na memória, mas ela não existe nesse domínio
  - A classe tem papel na criação dos objetos, mas a execução do programa se dá pela interação entre os objetos
- A classe é a “planta”, o objeto é a “casa” construída:
  - Muitas casas podem ser feitas a partir da mesma planta, mas cada uma preserva sua própria identidade

7

## Definição de classe em Java

```
class Conta {  
    String numero;  
    double saldo;  
    void creditar (double valor) {  
        saldo = saldo + valor;  
    }  
    void debitar (double valor) {  
        saldo = saldo - valor;  
    }  
}
```

Atributos

Métodos

8

## Exemplo: Programa que cria e manipula Conta

```
class CriaConta {  
    /* Criando um objeto do tipo Conta */  
    public static void main (String [] args) {  
        Conta conta1 = new Conta ();  
        conta1.numero = "21.342-7";  
        conta1.saldo = 0;  
        conta1.creditar (500.87);  
        conta1.debitar (45.00);  
        System.out.println(conta1.saldo);  
    }  
}
```

9

## Usando membros estáticos

- Classes podem declarar membros (campos e métodos) que sejam comuns a todas as instâncias, ou seja, membros compartilhados por todos os objetos da classe
- Tais membros são comumente chamados de 'membros de classe' (versus 'de objetos')
- Em Java, declara-se um membro de classe usando o qualificador **static**. Daí, o nome 'membros estáticos' usado em Java.

```
class Conta {  
    static String nomeBanco = "Itaú";  
    String numero;  
    double saldo;  
    ...  
    void creditar(double valor) {saldo += valor; }  
    void debitar(double valor) {saldo -= valor; }  
}
```

10

## Construtores

- São procedimentos executados na criação do objeto, uma única vez
- Têm o mesmo nome da classe. São similares a métodos, mas não têm tipo de retorno. Não fazem parte da definição do tipo do objeto (interface)

```
class Conta {  
    String numero;  
    double saldo;  
    Conta(String n) {  
        numero = n;  
        saldo = 0;  
    }  
    void creditar(double valor) {saldo += valor; }  
    void debitar(double valor) {saldo -= valor; }  
}
```

11

## Criando Objetos com Construtores

```
. . .  
Conta conta1;  
conta1 = new Conta("21.342-7");  
conta1.creditar(500.87);  
conta1.debitar(45.00);  
System.out.println(conta1.saldo);  
. . .
```

12

## Membros de uma classe: atributos e métodos

- Uma classe define uma estrutura de dados não-ordenada, podendo conter componentes em qualquer ordem
- Os componentes de uma classe são seus **membros**
- Uma classe pode conter três tipos de componentes
  - Membros estáticos ou de classe: não fazem parte do “tipo” do objeto
  - Membros de instância: definem o tipo de um objeto
  - Procedimentos de inicialização
- **Membros estáticos ou de classe**
  - Podem ser usados através da classe mesmo quando não há objetos
  - Não se replicam quando novos objetos são criados (“variáveis globais” da classe)
- **Membros de instância**
  - Cada objeto, quando criado, aloca espaço para eles
  - Só podem ser usados através de objetos
- **Procedimentos de inicialização** (construtores)
  - Usados para inicializar objetos ou classes

13

## Terminologia

```
public class Casa {  
    private Porta porta;  
    private int numero;  
    public java.awt.Color cor;  
  
    public Casa() {  
        porta = new Porta();  
        numero = ++contagem * 10;  
    }  
  
    public void abrePorta() {  
        porta.abre();  
    }  
  
    public static String arquiteto = "Zé";  
    private static int contagem = 0;  
  
    static {  
        if ( condição ) {  
            arquiteto = "Og";  
        }  
    }  
}
```

**Atributos de instância:** cada objeto poderá armazenar valores diferentes nessas variáveis.

**Procedimento de inicialização de objetos (Construtor):** código é executado após a criação de cada novo objeto. Cada objeto terá um número diferente.

**Método de instância:** só é possível chamá-lo se for através de um objeto.

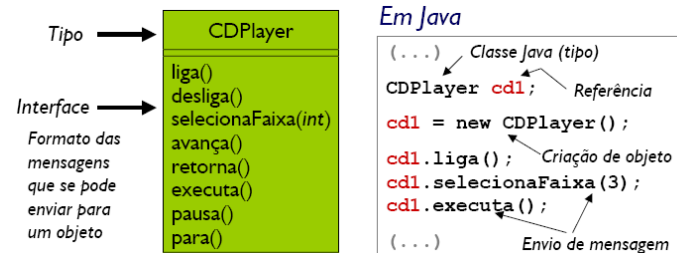
**Atributos estáticos:** não é preciso criar objetos para usá-los. Todos os objetos os compartilham.

**Procedimento de inicialização estático:** código é executado uma única vez, quando a classe é carregada. O arquiteto será um só para todas as casas: ou Zé ou Og.

14

## Objetos possuem uma interface...

- Através da interface é possível utilizá-lo
  - Não é preciso saber dos detalhes de sua implementação
- O tipo (Classe) de um objeto determina sua interface
  - O tipo determina quais mensagens lhe podem ser enviadas



Ao interagir com um objeto, levamos em conta a interface (o serviço) e não a implementação. Chamamos a isso de **Abstração**.

15

## ...e uma implementação (oculta)

- Separação de interface e implementação:
  - Os programas usuários de classes veem apenas sua interface
  - a implementação fica escondida dentro dos métodos (a isso chamamos de **Encapsulamento**)
- Papel do usuário da classe:
  - não precisa saber como a classe foi escrita, apenas o nome dos métodos, seus parâmetros (quantidade, ordem e tipo) e valor de retorno – **assinatura** do método
  - utiliza apenas a **interface** (pública) da classe
- Papel do desenvolvedor da classe:
  - expõe, através de métodos, as funcionalidades necessárias ao usuário da classe e oculta o restante da implementação
  - tem a liberdade de mudar a implementação da classe, desde que isso não comprometa sua interface

16



## Desenvolvendo um exemplo

*Uma máquina de venda de bilhetes*

### Máquina de venda de bilhetes: visão externa

- Explorando o comportamento de uma máquina de venda de bilhetes:
  - A máquina fornece bilhetes de preço fixo
  - Os clientes inserem dinheiro na máquina e depois solicitam que um bilhete seja impresso
  - A máquina mantém um total geral do dinheiro coletado durante sua operação

## Estrutura de uma classe básica em Java

```
public class TicketMachine  
{  
    Parte interna da classe omitida.  
}
```

O empacotador externo da TicketMachine

```
public class NomeDaClasse  
{  
    Campos/Atributos  
    Construtores  
    Métodos  
}
```

O conteúdo de uma classe

19

## Atributos ou campos

- Atributos armazenam valores para um objeto
- Eles também são conhecidos como variáveis de instância
- Atributos definem o estado de um objeto

```
Public class TicketMachine  
{  
    private int price;  
    private int balance;  
    private int total;  
  
    Construtor e métodos omitidos  
}
```

Modificador de visibilidade      Tipo      Nome da variável

↓                                      ↓                                      ↓

private int price;

20

## Métodos

```
public class TicketMachine
{
    Atributos omitidos

    public TicketMachine(int ticketCost){ }

    public int getPrice(){ return price; }

    public int getBalance(){ return balance; }

    public void insertMoney(int amount) { }

    public void printTicket(){ }

}
```

21

## Construtores

- Construtores inicializam um objeto
- Têm o mesmo nome de sua classe
- Inicializam os atributos
- Frequentemente recebem valores de parâmetros externos à classe

```
public class TicketMachine
{
    public TicketMachine(int ticketCost)
    {
        price = ticketCost;
        balance = 0;
        total = 0;
    }
}
```

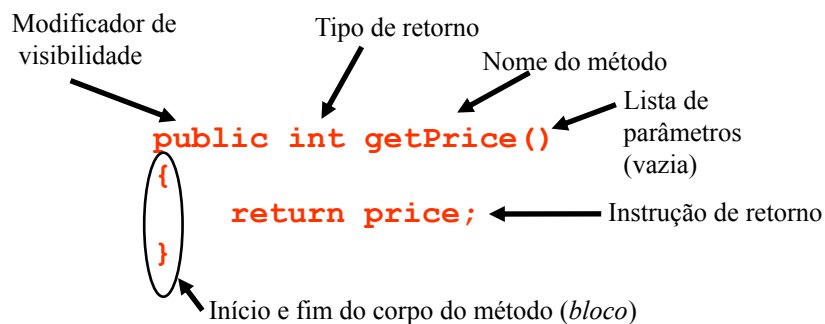
22

## Métodos de acesso

- Métodos implementam o comportamento dos objetos
- Métodos de acesso **fornecem informações** sobre um objeto
- A classe `TicketMachine` possui os seguintes métodos:
  - `getPrice`, `getBalance`, `insertMoney`, `printTicket`
- Métodos têm uma estrutura que consiste em um cabeçalho e um corpo
- O cabeçalho define a *assinatura do método*:  
`public int getPrice()`
- O corpo engloba as instruções do método

23

## Métodos de acesso (getters)



24

## Métodos modificadores (*setters*)

- Utilizados para **modificar** o estado de um objeto
  - Geralmente contêm instruções de atribuição
  - Geralmente recebem parâmetros
  - Geralmente seu tipo de retorno é void

```
public void insertMoney(int amount)
{
    balance += amount;
}
```

Modificador de visibilidade

Tipo de retorno (void)

Nome do método

Parâmetro

Instrução de atribuição

Campo sendo alterado

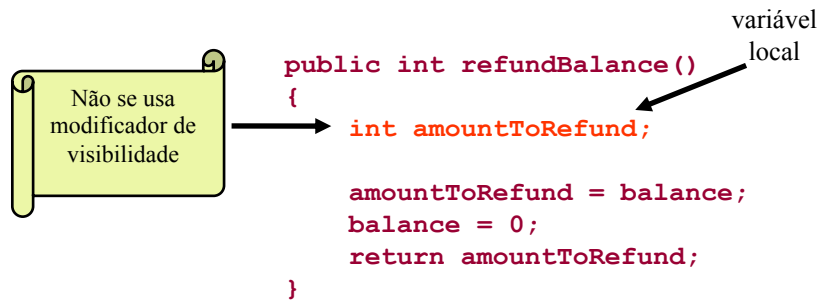
25

## Variáveis locais

- Atributos são um tipo de variável:
  - armazenam valores por toda a vida de um objeto
  - são acessíveis por meio da classe (seu escopo)
  - uma cópia do atributo por objeto instanciado
- Os métodos também podem declarar variáveis locais:
  - existem apenas enquanto o método está em execução
  - são acessíveis somente dentro do método

26

## Variáveis locais



27

## Resumo

- O corpo das classes pode conter atributos (campos), construtores e métodos
- Atributos armazenam o estado de um objeto
- Construtores inicializam objetos
- Métodos implementam o comportamento dos objetos
- Atributos, parâmetros e variáveis locais são variáveis
- Atributos persistem pelo tempo de vida de um objeto
- Parâmetros são utilizados para receber valores em um construtor ou método
- Variáveis locais são utilizadas para armazenamento temporário de curta duração e existem apenas durante a execução do método

28

## Métodos estáticos

- **Métodos estáticos** (métodos de classe)
  - Aplicam-se à classe como um todo
  - Não é necessário criar um objeto para utilizá-los
  - São chamados especificando-se o nome da classe onde o método é declarado:  
`NomeClasse.nomeMétodo()`
  - Todos os métodos da classe Math são **static**:
    - EX: `Math.sqrt(144.0)`

29

## Atributos estáticos

- **Campos estáticos** (variáveis de classe)
  - A mesma cópia é compartilhada entre todas as instâncias da classe (objetos)
  - Uma espécie de “variável global” da classe
- **Constantes**
  - Declaradas com a palavra-chave **final**. Não pode ser alterada depois da inicialização
  - Classe Math:
    - Campos `Math.E` e `Math.PI` são `final static`

30

## A classe Math

<code>abs( x )</code>	valor absoluto de $x$	<code>abs( 23.7 )</code> é 23.7 <code>abs( 0.0 )</code> é 0.0 <code>abs( -23.7 )</code> é 23.7
<code>ceil( x )</code>	arredonda $x$ para o menor inteiro não menor que $x$	<code>ceil( 9.2 )</code> é 10.0 <code>ceil( -9.8 )</code> é -9.0
<code>cos( x )</code>	co-seno trigonométrico de $x$ ( $x$ em radianos)	<code>cos( 0.0 )</code> é 1.0
<code>exp( x )</code>	método exponencial $e^x$	<code>exp( 1.0 )</code> é 2.71828 <code>exp( 2.0 )</code> é 7.38906
<code>floor( x )</code>	arredonda $x$ para o maior inteiro não maior que $x$	<code>floor( 9.2 )</code> é 9.0 <code>floor( -9.8 )</code> é -10.0
<code>log( x )</code>	logaritmo natural de $x$ (base $e$ )	<code>log( Math.E )</code> é 1.0 <code>log( Math.E * Math.E )</code> é 2.0
<code>max( x, y )</code>	maior valor de $x$ e $y$	<code>max( 2.3, 12.7 )</code> é 12.7 <code>max( -2.3, -12.7 )</code> é -2.3
<code>min( x, y )</code>	menor valor de $x$ e $y$	<code>min( 2.3, 12.7 )</code> é 2.3 <code>min( -2.3, -12.7 )</code> é -12.7
<code>pow( x, y )</code>	$x$ elevado à potência de $y$ (isto é, $x^y$ )	<code>pow( 2.0, 7.0 )</code> é 128.0 <code>pow( 9.0, 0.5 )</code> é 3.0
<code>sin( x )</code>	seno trigonométrico de $x$ ( $x$ em radianos)	<code>sin( 0.0 )</code> é 0.0
<code>sqrt( x )</code>	raiz quadrada de $x$	<code>sqrt( 900.0 )</code> é 30.0
<code>tan( x )</code>	tangente trigonométrica de $x$ ( $x$ em radianos)	<code>tan( 0.0 )</code> é 0.0

31

## Cooperação entre objetos

*Um relógio digital*

1.0



## Abstração e modularização

- **Abstração** é a habilidade de ignorar detalhes sobre as partes para concentrar a atenção no nível mais alto de um problema
- **Modularização** é o processo de dividir um todo em partes bem definidas, que podem ser construídas e examinadas separadamente e que interagem de uma maneira pré-determinada

33

## Modularização no exemplo do relógio

11:03

Um mostrador de número de quatro dígitos?

Ou um mostrador de número de dois dígitos?

11

03

34

## Implementação — NumberDisplay

```
public class NumberDisplay
{
    private int limit;
    private int value;

    Construtor e métodos omitidos
}
```

Uma classe de exibição de um dígito,  
com um **limite** superior e um **valor** atual

35

## Implementação — ClockDisplay

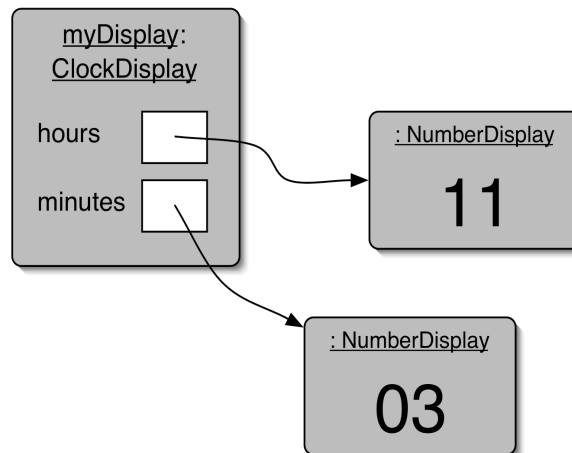
```
public class ClockDisplay
{
    private NumberDisplay hours;
    private NumberDisplay minutes;

    Construtor e métodos omitidos
}
```

Um mostrador de relógio completo que contém,  
internamente, dois **mostradores de números**

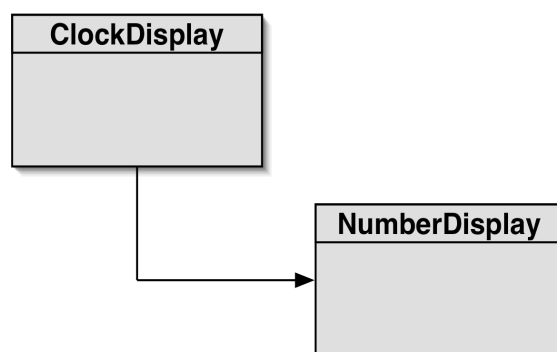
36

## Diagrama de objetos



37

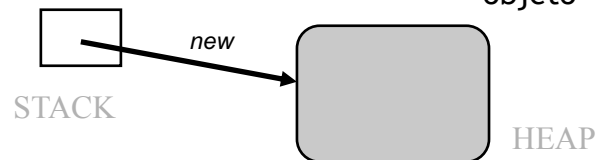
## Diagrama de classes



38

## Tipos primitivos vs. Objetos

```
SomeObject obj;
```



Uma referência para o objeto é armazenada na variável

```
int i;
```



tipo primitivo

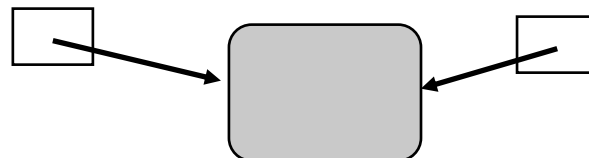
Os tipos primitivos são armazenados na própria variável

39

## Tipos primitivos vs. Objetos

```
SomeObject a;
```

```
SomeObject b;
```



\_\_\_\_\_ b = a; \_\_\_\_\_

```
int a;
```

```
int b;
```



40

## NumberDisplay (1)

```
// Construtor
public NumberDisplay(int rollOverLimit)
{
    limit = rollOverLimit;
    value = 0;
}

public void increment()
{
    value = (value + 1) % limit;
}
```

41

## NumberDisplay (2)

```
public String getDisplayValue()
{
    if(value < 10)
        return "0" + value;
    else
        return "" + value;
}
```

42

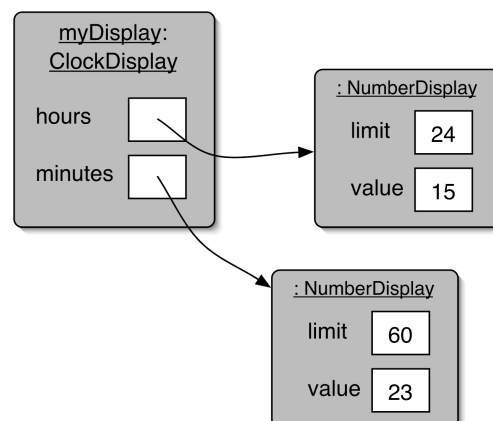
## ClockDisplay: Objetos criando objetos

```
public class ClockDisplay
{
    private NumberDisplay hours;
    private NumberDisplay minutes;
    private String displayString;

    public ClockDisplay()
    {
        hours = new NumberDisplay(24);
        minutes = new NumberDisplay(60);
        updateDisplay();
    }
}
```

43

## Diagrama do objeto ClockDisplay



44

## Objetos criando objetos

Na classe NumberDisplay:

```
public NumberDisplay(int rolloverLimit);
```

*parâmetro formal*

Na classe ClockDisplay:

```
hours = new NumberDisplay(24);
```

*parâmetro real*

45

## ClockDisplay: Métodos externos

```
public void timeTick()
{
    minutes.increment();
    if(minutes.getValue() == 0) {
        // retornou a zero
        hours.increment();
    }
    updateDisplay();
}
```

46

## ClockDisplay: Método interno

```
// Atualiza a string interna que
// representa o mostrador

private void updateDisplay()
{
    displayString =
        hours.getDisplayValue() + ":" +
        minutes.getDisplayValue();
}
```

47

## ClockDisplay: Múltiplos construtores

```
public ClockDisplay() {
    hours = new NumberDisplay(24);
    minutes = new NumberDisplay(60);
    updateDisplay(); // Hora inicial 00:00
}

public ClockDisplay(int hour, int minute) {
    hours = new NumberDisplay(24);
    minutes = new NumberDisplay(60);
    setTime(hour, minute);
}
```

É possível inicializar um objeto ClockDisplay de duas maneiras diferentes -- *sobrecarga de construtor ou método*

48



## A palavra-chave *this*

```
public class MailItem
{
    private String from;
    private String to;
    private String message;

    public MailItem(String from, String to,
                    String message)
    {
        this.from = from;
        this.to = to;
        this.message = message;
    }
}
```

A expressão *this* referencia o objeto atual, diferenciando o atributo do parâmetro

49

## Relacionamento entre Objetos

Associação, Agregação e Composição

## Tipos de relacionamentos

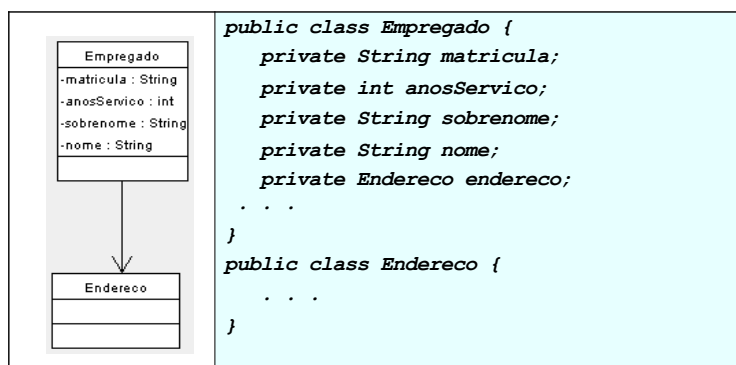
- Objetos **não** são ilhas isoladas. Pelo contrário, podem existir diferentes tipos de relacionamentos entre os objetos ao longo do ciclo de vida do sistema
- **Reuso de classe:** **fornece menos flexibilidade**
  - **Herança pura** (sobreposição): b “é” a
  - **Herança com extensão:** b “é um tipo de” a
- **Uso e reuso de objetos:** **fornece mais flexibilidade**
  - **Associação:** a “é usado por” b
  - **Agregação:** a “é parte de” b
  - **Composição:** a “é parte essencial de” b

O grau de coesão entre os objetos foi colocado do mais fraco para o mais forte

51

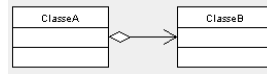
## Associação

- Representa relacionamentos mais fortes entre instâncias de classes (objetos)

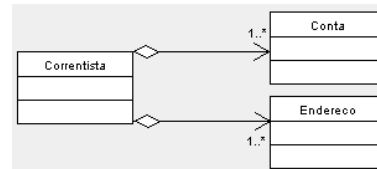


52

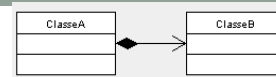
## Agregação



- A agregação é um relacionamento do tipo **todo/parte**:
  - É usado para mostrar uma relação de inclusão lógica, ou seja, um todo formado por partes
  - Embora as partes possam existir **independentemente** do todo, sua existência é basicamente para formar o todo
  - Exemplo: um correntista **precisa** de pelo menos uma conta e um endereço. Sem uma conta não se pode dizer que ele é correntista
  - A **agregação** é frequentemente representada como uma simples **associação**

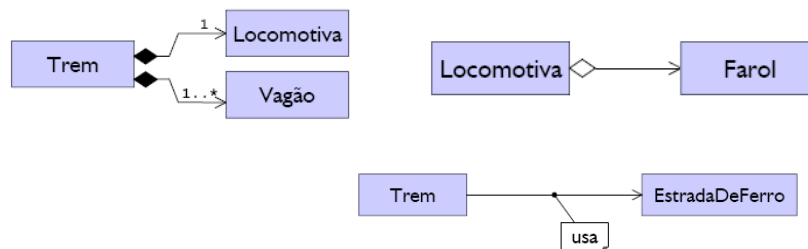


## Composição



- Uma composição é um tipo de agregação. A diferença é que na **composição** o objeto composto faz parte de **somente** um relacionamento (um todo), enquanto que na agregação isso não é obrigatório
- Exemplo 1: um trem é formado por uma locomotiva e vagões. Uma locomotiva ainda tem um farol
- Exemplo 2: um Quadrado é formado por dois pontos e um Círculo tem um ponto e um raio. Cada uma dessas formas geométricas ainda tem um estilo

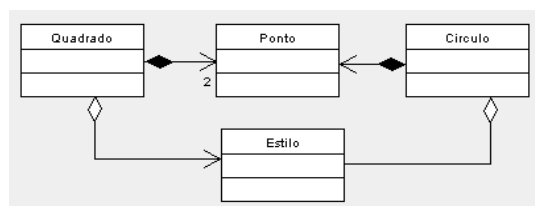
## Composição X Agregação X Associação



- Um trem **não existe** sem a locomotiva e os vagões. Por sua vez, os dois últimos até “assumem” uma nova identidade ao formarem o trem
- Uma locomotiva **possui** um farol (mas não vai deixar de ser uma locomotiva se não o tiver)
- Um trem **usa** uma estrada de ferro (ela não faz parte do trem, mas ele depende dela)

55

## Composição X Agregação



- A relação de **Ponto** com **Circulo** e **Quadrado** é uma **composição**, pois os mesmos não podem ser compartilhados
- Enquanto que o mesmo objeto de **Estilo** pode ser compartilhado por **Circulo** e **Ponto** (**agregação**)

56

# Composição X Agregação

- Tradução de composição e agregação na linguagem Java

```
public class Quadrado {  
    // p1 e p2 são composição - new  
    // estilo é agregação - atribuição  
    private Ponto p1, p2;  
    private Estilo estilo;  
    public Quadrado(int x1, int y1,  
                    int x2, int y2, Estilo e){  
        p1 = new Ponto(x1, y1);  
        p2 = new Ponto(x2, y2);  
        estilo = e;  
    }  
}
```

57

