

# Métodos e variáveis de instância

Maurício Linhares – [mauricio.linhares@gmail.com](mailto:mauricio.linhares@gmail.com)

# Visibilidade, pacotes e “importando” objetos

- Todas as classes Java vivem dentro de “pacotes” (o diretório aonde elas se encontram);
- Dividir as classes em pacotes é necessário para organizar o código, assim, classes relacionadas ficam juntas, nas mesmas pastas;
- O pacote aonde a classe está é definido pela palavra reservada “**package**”, que aparece no início de um arquivo .java;
- O nome do pacote sempre fica fora da definição de classe;

# Exemplo de definição de pacote

```
package locadora.cadastro;
```

```
import java.util.List;
```

```
import locadora.util.Paginador;
```

```
public class CadastroDeClientes {  
    //métodos e atributos...  
}
```

# Packages

- O “package locadora.cadastro” avisa que essa classe pertence ao pacote locadora.cadastro e que deve estar dentro da pasta “locadora/cadastro” ;
- O ponto ( “.” ) no nome de um pacote avisa a definição de uma pasta, então cada nome separado por ponto simboliza uma pasta no sistema de arquivos do sistema operacional, o arquivo .java que referencia o pacote sempre deve estar na última pasta do pacote;
- A definição de pacotes não é obrigatória, quando eles não são definidos, o compilador assume que o pacote é o diretório atual;

# Importando código

- Uma classe não pode acessar classes fora do seu pacote, a não ser que a classe do outro pacote esteja visível (pública) e que a classe que a deseja usar faça a importação dessa classe;
- Para importar uma classe é necessário utilizar a palavra reservada “import” seguida do nome completo da classe (o nome do pacote junto do nome da classe, como em `java.util.Date`);
- Depois de importada, a classe externa pode ser utilizada normalmente.

# Exemplo do uso de imports

```
import java.util.Random;
```

```
public class Sorteador {
```

```
    public static void main(String[]  
    args) {
```

```
        Random random = new Random();
```

```
        System.out.println(  
            random.nextInt(7) + 1 );
```

```
    }
```

```
}
```

# Níveis de visibilidade em Java

- `private` – apenas quem estiver na mesma classe pode ver;
- `protected` – apenas quem estiver no mesmo pacote ou em subclasses pode ver;
- `default` (sem definição) – apenas quem estiver no mesmo pacote pode ver;
- `public` – todos podem ver;

# Invocando métodos

- Para se invocar um método, nós utilizamos o operador “.” (ponto);
- É possível passar informações para os métodos através dos parâmetros, mas a definição de parâmetros não é obrigatória;
- Métodos podem ou não retornar valores;



# Exemplo de definição de métodos

```
public class Cachorro {  
  
    public void latir() {  
        System.out.println("Hu! Hu!");  
    }  
  
    public boolean comFome() {  
        return true;  
    }  
  
}
```

# Métodos que acessam variáveis de instância

```
public class Conta extends Object {  
  
    public int numero;  
    private double saldo;  
  
    public void depositar( double valor ) {  
        this.saldo = this.saldo + valor;  
    }  
  
    public boolean sacar( double valor ) {  
        boolean efetuado = false;  
        if ( this.saldo >= valor ) {  
            this.saldo = this.saldo - valor;  
            efetuado = true;  
        }  
        return efetuado;  
    }  
}
```

# Usando o objeto Conta

```
Conta minhaConta = new Conta();
```

```
Conta suaConta = new Conta();
```

```
minhaConta.depositar(400);
```

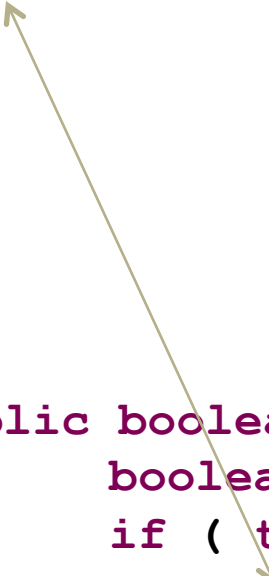
```
suaConta.depositar( 1000 );
```

```
suaConta.sacar(200);
```

```
minhaConta.depositar(200);
```

# Anatomia da chamada

```
suaConta.sacar(200);
```



```
public boolean sacar( double valor ) {  
    boolean efetuado = false;  
    if ( this.saldo >= valor ) {  
        this.saldo = this.saldo - valor;  
        efetuado = true;  
    }  
    return efetuado;  
}
```

# Em java tudo se passa por cópia...

- Todos os parâmetros são passados por cópia de valor;
- Para primitivos, o que é copiado é o valor do primitivo;
- Para objetos, o que é copiado é a referência;

# Passagem de primitivos

```
public void testeDeCopiaDeParametros () {  
    int x = 9;  
    int y = 5;  
  
    System.out.println(soma( x, y ));  
    System.out.println(x);  
}
```

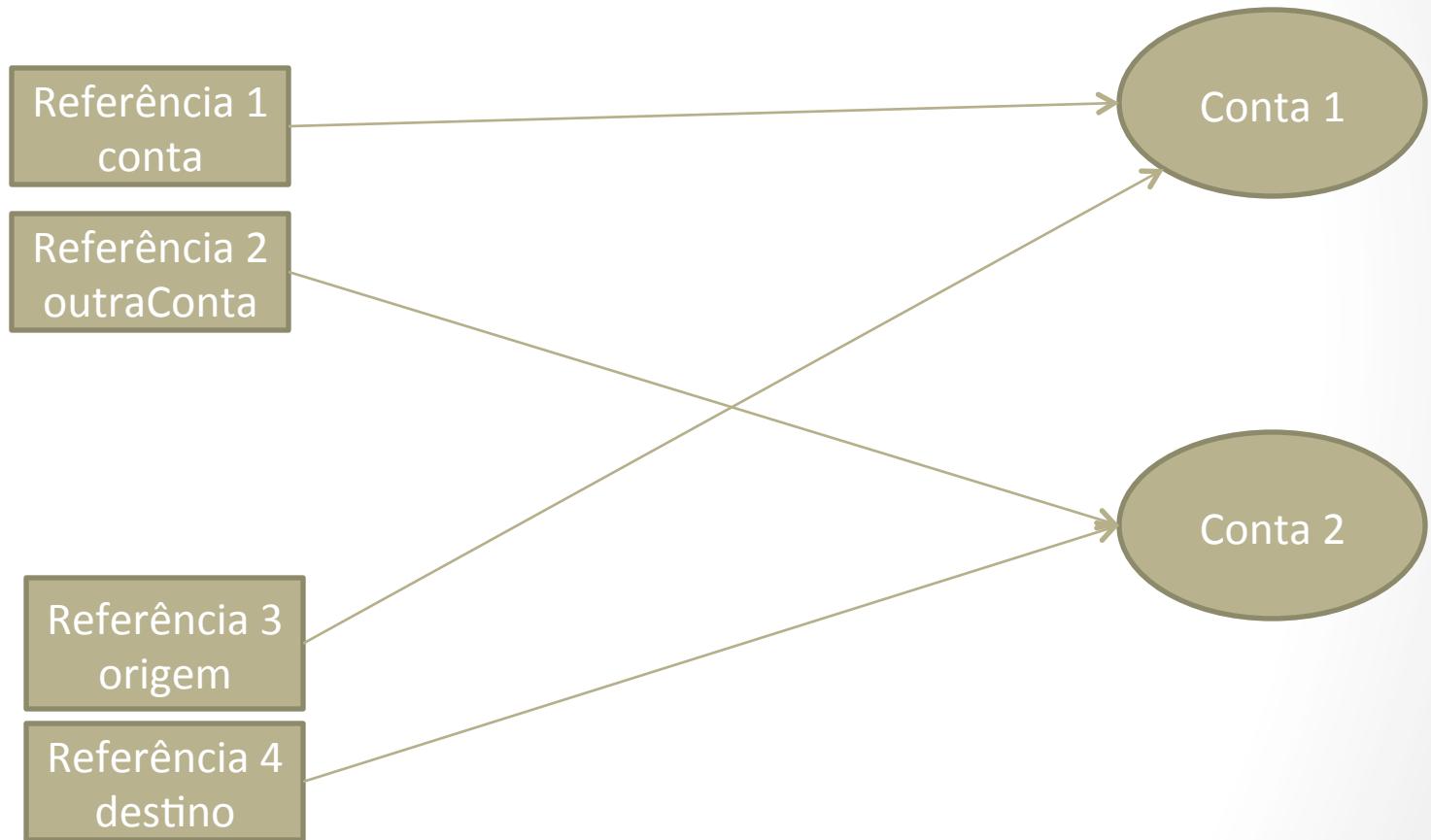
```
public int soma( int primeiro, int  
    segundo ) {  
    primeiro = segundo;  
    return primeiro + segundo;  
}
```

# Teste de passagem de objetos

```
public void testeDePassagemDeObjetos() {  
    Conta conta = new Conta(300);  
    Conta outraConta = new Conta(400);  
    transferir( conta, outraConta, 200);  
}
```

```
public void transferir( Conta origem,  
    Conta destino, double valor ) {  
    origem = new Conta(500);  
    origem.sacar( valor );  
    destino.depositar(valor);  
}
```

# Dissecando o código





# Voltando a orientação a objetos

- Isso não é uma boa idéia:
  - `conta.numero = 2;`
- Qualquer um pode alterar o valor do meu objeto;
- Se um dia eu precisar mudar o jeito de se colocar o número numa conta, vou quebrar todo o código;
- Não existe nenhuma validação para o que está sendo passado;

# Então lá vem o encapsulamento

- As informações de um objeto devem ser “escondidas” dos agentes externos;
- Ninguém deve ter acesso direto as variáveis de instância de um objeto;
- O objeto deve definir métodos para que seja possível alterar o seu estado de forma segura e considerando as suas invariantes (o seu estado correto);

# Jeito simples de se implementar o encapsulamento em Java

- Coloque todas as variáveis de instância como sendo `private`;
- Crie métodos para alterar e/ou ler o valor dessas variáveis;
- Apenas as variáveis que **REALMENTE** precisam ser acessadas por objetos externos devem ter métodos para acesso e alteração, **NÃO** crie métodos de acesso e alteração pra todo mundo;
- Cada par de métodos `get/set` determina uma “propriedade virtual” do seu objeto, ele não precisa ter essa propriedade como variável de instância;

# Em java...

- Os métodos de acesso são chamados “getters” ou “accessors” (os pegadores):

```
public double getSaldo() {  
    return this.saldo;  
}
```

- Os métodos de alteração são chamados de “setters” ou “mutators” (os colocadores):

```
public void setSaldo( double saldo ) {  
    this.saldo = saldo;  
}
```

# Por que fazer isso?

- Quando você usa os gets e sets você está escondendo o funcionamento da sua classe de quem a usa;
- Escondendo as variáveis que a sua classe tem através de métodos torna possível que você possa alterar a implementação do seu objeto sem que as pessoas e o código que o usam percebam;
- Você pode adicionar regras de validação ou transformação nos métodos para evitar que o usuário envie informações erradas;

# Um exemplo....

- Usar datas em Java é um martírio, as classes Date e Calendar são tristes;
- Na classe calendar, alguém teve a brilhante idéia de colocar os meses começando do 0 em vez do 1, então dezembro é o mês 11;
- Não existe um meio simples e direto de se trabalhar com Java...

# Até surgir a nossa classe Data!

```
public class Data {  
    private GregorianCalendar calendar = new GregorianCalendar();  
  
    public int getDia() {  
        return this.calendar.get( Calendar.DAY_OF_MONTH );  
    }  
  
    public void setDia( int diaDoMes ) {  
        this.calendar.set( Calendar.DAY_OF_MONTH , diaDoMes);  
    }  
  
    public void setMes( int mes ) {  
        if ( mes >= 1 && mes <= 12 ) {  
            this.calendar.set( Calendar.MONTH , mes - 1);  
        }  
    }  
  
    public int getMes() {  
        return this.calendar.get( Calendar.MONTH ) + 1;  
    }  
}
```

# O que é que nós temos?

- Um objeto com uma única variável de instância, mas com três propriedades (o ano não aparece no slide);
- Na propriedade “mes” nós estamos fazendo uma transformação para que os meses sejam contados corretamente, de 1 a 12;
- Ainda na propriedade “mes” nós estamos fazendo uma validação quando só aceitamos meses de 1 a 12;



# Caso especial dos gets/sets

- Para variáveis booleanas o método get vira um método “is”:

```
public class Cliente {  
  
    boolean pagador;  
  
    public boolean isPagador() {  
        return pagador;  
    }  
}
```

# Variáveis de instância e inicialização

- As variáveis de instância são inicializadas com um valor padrão se você não der nenhum valor a elas;
  - Numeros são inicializados com 0
  - Booleans são inicializados com “**false**”
  - Referências para objetos são inicializados com “**null**”

# Exemplo...

```
public class Cliente {  
  
    int numero;  
    boolean pagador;  
    String nome;  
  
    public static void main(String[] args) {  
        Cliente cliente = new Cliente();  
        System.out.println( cliente.numero );  
        System.out.println( cliente.pagador );  
        System.out.println( cliente.nome );  
    }  
  
}
```

# Mas e as variáveis locais?

- Variáveis locais são aquelas que são definidas dentro de um método;
- Variáveis locais não tem valores de inicialização padrão, você precisa definir explicitamente o valor delas;
- Se você não definir explicitamente o valor, a variável não vai poder ser utilizada;

# Exemplo de variáveis locais

```
public static void main(String[]  
    args) {  
    int x;  
    System.out.println( x ); //erro  
    de compilação  
    int z = 20;  
    System.out.println(z);  
}
```

# Exercício - 1

- Implemente um objeto pessoa que tenha nome e data de nascimento como propriedades e um método getIdade() que retorne a idade atual da pessoa através do cálculo da data de nascimento pela data atual.
  - Dicas:
    - Lembre-se que a data de nascimento pode ser antes ou depois da data atual no ano atual, então substituir os anos não funciona;
    - Cálculos de datas em Java são sempre feitos através da classe `java.util.Calendar`

# Exercício - 2

- O sistema deve definir duas contas bancárias;
- O sistema deve fornecer ao usuário na linha de comando as opções de depositar ou sacar das contas, além de transferir dinheiro entre as duas;
- O sistema deve avisar ao usuário quando ele tentar sacar ou transferir mais dinheiro do que há na conta;
- O sistema deve ter uma opção para “fechar” (o usuário não deve fechar o sistema “no braço”);

# Dicas de implementação

- Um switch pode ser uma boa escolha para definir o que o cliente quer fazer;
- Todos os dados das contas devem ficar nos seus respectivos objetos conta;
- Scanners são a forma mais fácil de ler da linha de comando;