



UPskill – JAVA + .NET

Programação Orientada a Objetos - Classes e Objetos

Adaptado de Donald W. Smith (TechNeTrain)

- Programação Orientada ao Objeto
- Implementação de uma Classe Simples
- Definição da Interface Pública de uma Classe
- Conceção da Representação de Dados
- Implementação de Métodos de Instância
- Construtores
- Teste de uma Classe
- Referências de Objetos
- Variáveis e Métodos Estáticos

Programação Orientada ao Objeto

- Até agora, aprenderam programação estruturada
 - Decompor tarefas em sub-tarefas
 - Escrita de métodos reutilizáveis para processar tarefas
- A partir de agora estudaremos Classes e Objetos
 - Para construir programas de maior complexidade e dimensão
 - Para modelar objetos usados no mundo real



Uma classe descreve objetos com o mesmo comportamento. Por exemplo, a classe Carro descreve todos os veículos de passageiros que têm uma determinada marca e marca, entre outras características.

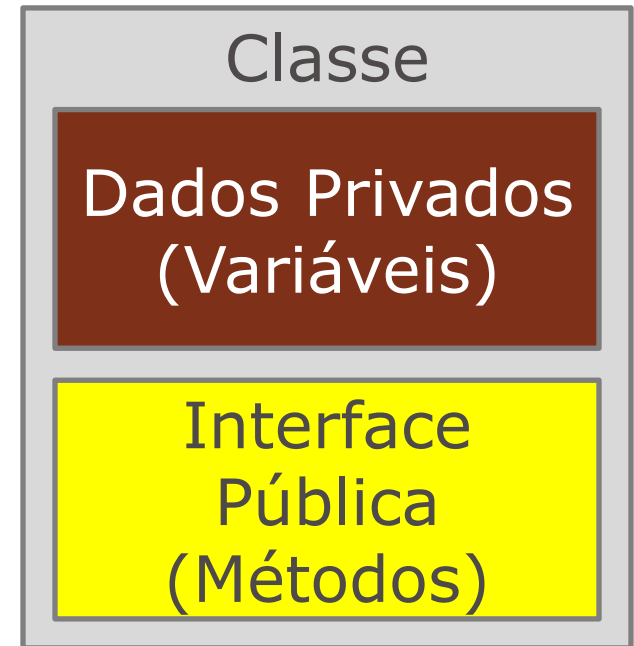
- Os programas Java são formados por objetos que interagem entre eles
 - Cada objeto é baseado numa classe
 - Uma classe descreve um conjunto de objetos com o mesmo comportamento
- Cada classe define um conjunto específico de métodos para usar com os seus objetos
 - Por exemplo, a classe String disponibiliza os métodos `length()` e `charAt()`

```
String saudacao = "Hello World";  
int len = saudacao.length();  
char c1 = saudacao.charAt(0);
```

Estrutura de uma Classe

■ Dados Privados

- Cada objeto tem os seus próprios dados privados aos quais outros objetos não podem aceder diretamente
- Métodos da interface pública podem aceder a dados privados, escondendo detalhes de implementação:
- A isto chama-se **Encapsulamento**



■ Interface Pública

- Cada objeto tem um conjunto de métodos disponíveis para uso de outros objetos

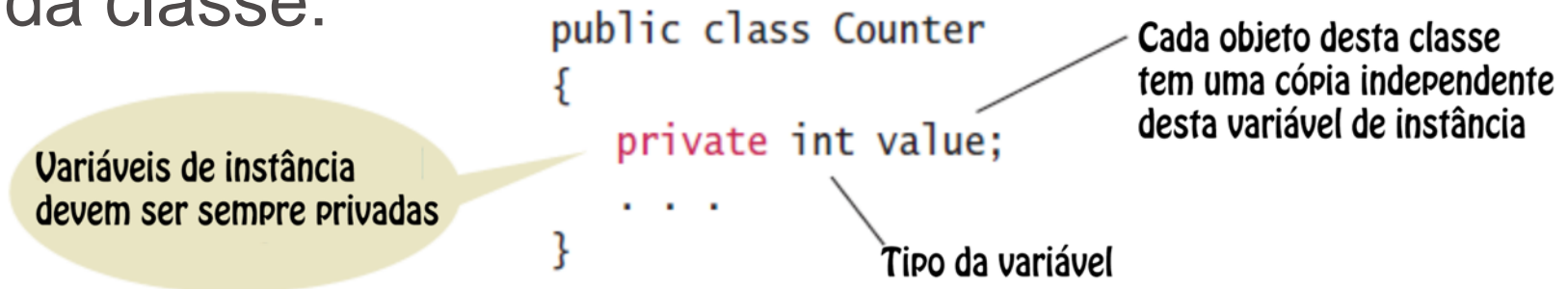
Implementação de uma Classe Simples

- Exemplo:
 - Contador de Visitas — Uma classe que modela um dispositivo mecânico que é usado para contar pessoas
 - Por exemplo, para determinar o número de pessoas num concerto ou num autocarro
- O que deve fazer?
 - Incrementar o contador
 - Obter o valor total atual



Classe Contador de Visitas

- Especificar as variáveis de instância na declaração da classe:



- Cada objeto instanciado a partir da classe tem o seu próprio conjunto de variáveis de instância
 - Cada contador de visitas tem a sua própria contagem
- Especificadores de Acesso:
 - Classes (e métodos de interface) são públicos (**public**)
 - Variáveis de instância são sempre privadas (**private**)

Instanciação de Objetos

- Os objetos são criados a partir de classes
 - Usar o operador **new** para construir objetos
 - Atribuir a cada objeto um nome único (são variáveis)
- O operador **new** foi usado anteriormente:

```
Scanner in = new Scanner(System.in);
```

- Exemplo: Criação de duas instâncias de objetos Counter:

Nome da classe Nome do objeto Nome da classe

```
Counter concertCounter = new Counter();  
Counter boardingCounter = new Counter();
```

Usar o operador **new** para
construir objetos de uma classe

Counter

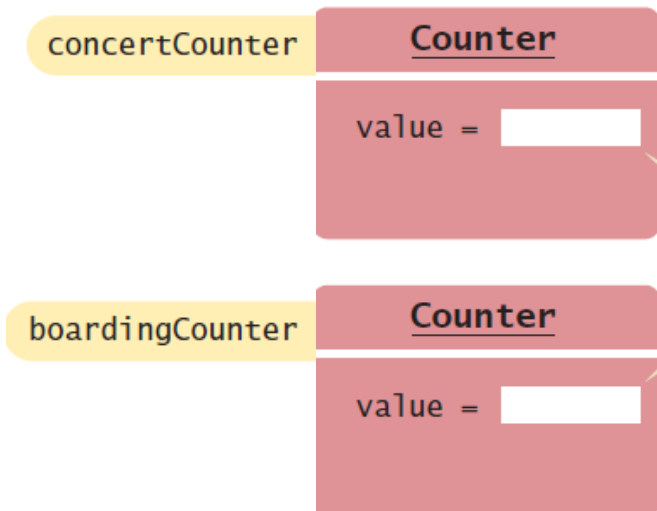
value =

Counter

value =

Métodos do Contador de Visitas

- Conceber um método chamado count que adiciona 1 à variável de instância
- A que variável de instância?
 - Usar o nome do objeto:
 - concertCounter.count()
 - boardingCounter.count()



```
public class Counter
{
    private int value;

    public void count()
    {
        value = value + 1;
    }

    public int getValue()
    {
        return value;
    }
}
```

Interface Pública de uma Classe

- Quando se define uma classe, começamos por especificar a sua interface pública
 - Exemplo: Classe Caixa Registradora
 - Que tarefas irá a classe desempenhar?
 - Que métodos serão necessários?
 - Que parâmetros necessitarão os métodos de receber?
 - O que irão os métodos devolver?

Tarefa	Método	Retorno
Adicionar o preço de um item	<code>addItem(double)</code>	<code>void</code>
Obter a quantia total devida	<code>getTotal()</code>	<code>double</code>
Obter o número de itens comprados	<code>getCount()</code>	<code>int</code>
Limpar o registo de caixa para uma nova compra	<code>clear()</code>	<code>void</code>

Definição da Interface Pública

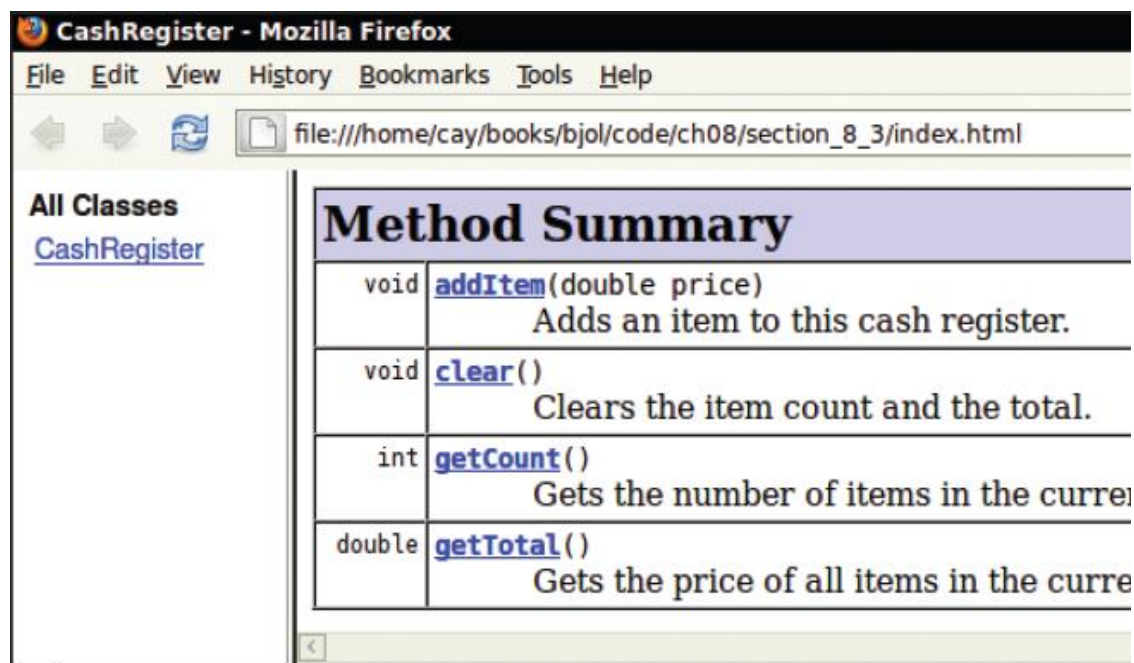
```
/**
 * Simulação de uma caixa registadora que mantém a contagem de itens
 * e a quantia total devida.
 */
public class CashRegister
{
    /**
     * Adiciona um item a esta caixa registadora.
     * @param price: o preço deste item
     */
    public void addItem(double price)
    {
        // Corpo do método
    }
    /**
     * Obtém o preço de todos os itens da venda atual.
     * @return o preço total
     */
    public double getTotal() ...
```

Comentários ao estilo Javadoc documentam a classe e o comportamento de cada método

As declarações de métodos públicos formam a *interface pública* da classe

Os corpos de dados e métodos privados formam a *implementação privada* da classe

- Javadoc – ferramenta que gera documentação em formato HTML a partir de comentários no estilo Javadoc presentes no código fonte
 - Métodos documentam parâmetros e tipos de retorno:
 - @param
 - @return




The screenshot shows a Mozilla Firefox browser window titled "CashRegister - Mozilla Firefox". The address bar displays the file path: `file:///home/cay/books/bjol/code/ch08/section_8_3/index.html`. On the left side, under the heading "All Classes", there is a link for [CashRegister](#). The main content area is titled "Method Summary" and contains a table listing the methods of the CashRegister class.

Method Summary	
void	addItem (double price) Adds an item to this cash register.
void	clear () Clears the item count and the total.
int	getCount () Gets the number of items in the current register.
double	getTotal () Gets the price of all items in the current register.

Métodos Não Estáticos

- Até agora, definiram métodos de *classe* usando o modificador **static**:

```
public static void addItem(double val)
```
- No caso de métodos não estáticos (de *instância*), é necessário instanciar um objeto da classe, para poderem ser invocados

register1 = 

CashRegister

```
public void addItem(double val)
```

```
public static void main(String[] args)
{
    // Criação de um objeto CashRegister
    CashRegister register1 = new CashRegister();
    // Invocação de um método não estático do objeto
    register1.addItem(1.95);
}
```

Métodos de Acesso e Modificação

- Alguns métodos enquadram-se nas categorias seguintes:

1. Métodos de Acesso (métodos **get**)

- Pede ao objeto informação sem a alterar
- Normalmente devolve um valor

```
public double getTotal() { }  
public int getCount() { }
```

2. Métodos de Modificação (**nomeadamente** métodos **set**)

- Modifica valores no objeto
- Habitualmente recebem um parâmetro que mudará uma variável de instância
- Normalmente devolvem void

```
public void addItem(double price) { }  
public void clear() { }
```

Conceção da Representação de Dados

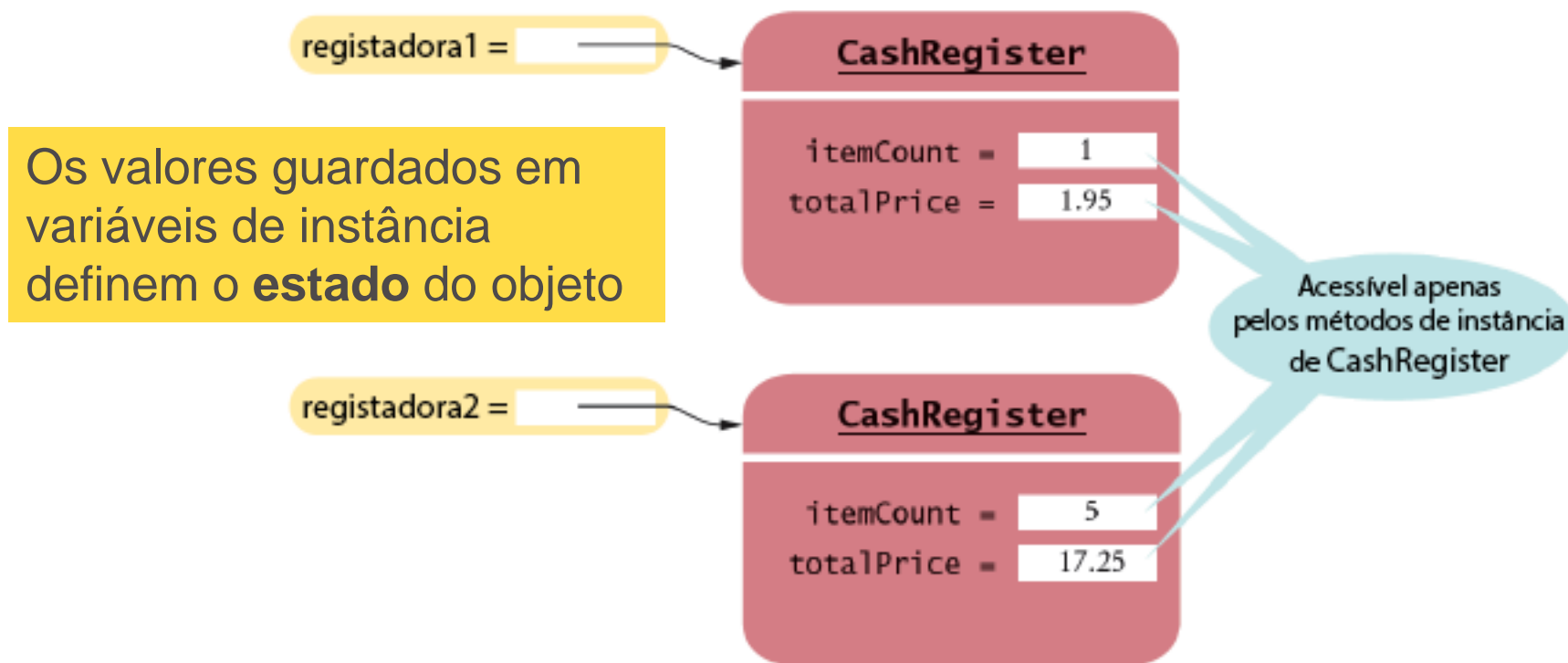
- Um objeto armazena dados nas variáveis de instância
 - Variáveis declaradas no interior da classe
 - Todos os métodos no interior da classe têm acesso às variáveis de instância, podendo por isso alterá-las ou aceder-lhes
 - Quais os dados necessários aos métodos da classe CashRegister?

Tarefa	Método	Dados necessários
Adiciona o preço de um item	addItem()	total, count
Obtém a quantia total devida	getTotal()	total
Obtém a contagem (count) de itens adquiridos	getCount()	count
Limpa a caixa registadora para uma nova venda	clear()	total, count

Um objeto contém variáveis de instância que são acedidos por métodos

Variáveis de Instância de Objetos

- Cada objeto de uma classe tem um conjunto independente de variáveis de instância



Acesso a Variáveis de Instância

- Variáveis de instância **private** não podem ser acedidas a partir de métodos fora da classe

```
public static void main(String[] args)
{
    ...
    System.out.println(register1.itemCount); // Erro
    ...
}
```

O compilador não permitirá esta violação de privacidade

- O acesso tem que ser feito através de métodos de acesso da classe

```
public static void main(String[] args)
{
    ...
    System.out.println( register1.getCount() ); // OK
    ...
}
```

O encapsulamento oferece uma interface pública e esconde os detalhes de implementação

Implementação de Métodos de Instância

- Implementação de métodos de instância que usarão as variáveis de instância privadas

```
public void addItem(double price)
{
    itemCount++;
    totalPrice = totalPrice + price;
}
```

Tarefa	Método	Retorno
Adicionar o preço de um item	addItem(double)	void
Obter a quantia total devida	getTotal()	double
Obter o número de itens comprados	getCount()	int
Limpar o registo de caixa para uma nova compra	clear()	void

Métodos de Instância

- Uso de variáveis de instância dentro dos métodos da classe
 - Não há necessidade de especificar o parâmetro implícito (nome do objeto) quando se usam variáveis de instância no interior da classe
 - Parâmetros explícitos devem ser definidos na declaração do método

```
public class CashRegister
{
    . . .
    public void addItem(double price)
    {
        itemCount++;
        totalPrice = totalPrice + price;
    }
    . . .
}
```

Parâmetro
explícito

Variáveis de instância de
parâmetro implícito

Parâmetros Implícitos e Explícitos

- Quando um item é adicionado, são afetadas as variáveis de instância do objeto sobre o qual o método é invocado

1 Before the method call.

register1 =

CashRegister

itemCount =

totalPrice =

2 After the method call register1.addItem(1.95).

The implicit parameter references this object.

The explicit parameter is set to this argument.

register1 =

CashRegister

itemCount =

totalPrice =

O objeto sobre o qual um método é aplicado é o parâmetro *implícito*

- Um *construtor* é um método que inicializa as variáveis de instância de um objeto
 - É chamado automaticamente quando um objeto é criado
 - Tem exatamente o mesmo nome da classe

```
public class CashRegister
{
    ...
    /**
     Constrói uma caixa registadora com o número de itens e total
     inicializados a zero.
    */
    public CashRegister() // Construtor
    {
        itemCount = 0;
        totalPrice = 0;
    }
}
```

Os construtores nunca devolvem valores, mas não se usa **void** na sua declaração

Múltiplos Construtores

- Uma classe pode ter mais do que um construtor
- Cada construtor tem um conjunto único de parâmetros

```
public class BankAccount
{
    ...
    /**
     Constrói uma conta bancária com balanço igual a zero.
    */
    public BankAccount() { ... }
    /**
     Constrói uma conta bancária com um dado balanço.
     @param initialBalance balanço inicial
    */
    public BankAccount(double initialBalance) { ... }
}
```

O compilador escolhe o construtor cujos parâmetros correspondem aos parâmetros de construção

```
BankAccount joesAccount = new BankAccount();
BankAccount lisasAccount = new BankAccount(499.95);
```

Construtores (sintaxe)

- Um construtor é invocado quando o objeto é criado com a palavra reservada **new**

```
public class BankAccount
{
    private double balance;

    public BankAccount()
    {
        balance = 0;
    }

    public BankAccount(double initialBalance)
    {
        balance = initialBalance;
    }
    . . .
}
```

Um construtor não devolve qualquer valor

Um construtor partilha o nome com a classe

Este construtor é selecionado para a expressão `new BankAccount(499.95)`

Construtor por omissão

- Se não for declarado qualquer construtor, o compilador constrói automaticamente um construtor
 - Não tem qualquer parâmetro
 - Inicializa todas as variáveis de instância

```
public class CashRegister
{
    ...
    /**
     * Faz exatamente o que o construtor gerado pelo compilador faria.
     */
    public CashRegister()
    {
        itemCount = 0;
        totalPrice = 0;
    }
}
```

Por omissão, números são inicializados a 0, booleanos a false e objetos a null

CashRegister.java

```
1  /**
2   * Modelação de uma caixa registadora que regista
3   * o número de itens e a quantia devida
4   */
5  public class CashRegister
6  {
7      private int itemCount;
8      private double totalPrice;
9
10     /**
11      * Constrói uma caixa registadora com número
12      * de itens e quantia total inicializadas a zero
13      */
14     public CashRegister()
15     {
16         itemCount = 0;
17         totalPrice = 0;
18     }
19
20     /**
21      * Adiciona um item a esta caixa registadora
22      * @param price o preço deste item
23      */
24     public void addItem(double price)
25     {
26         itemCount++;
27         totalPrice = totalPrice + price;
```

```
28
29     /**
30      * Obtém o preço de todos os itens na venda atual
31      * @return a quantia total
32      */
33     public double getTotal()
34     {
35         return totalPrice;
36     }
37
38     /**
39      * Obtém o número de itens na venda atual
40      * @return o número de itens
41      */
42     public int getCount()
43     {
44         return itemCount;
45     }
46
47     /**
48      * Inicializa o número de itens e o total
49      */
50     public void clear()
51     {
52         itemCount = 0;
53         totalPrice = 0;
54     }
55 }
```



Erro Comum (1)

- Não inicialização de referências de objetos num construtor
 - Referências são inicializadas a **null**
 - A chamada de um método sobre uma referência null resulta num erro de *runtime*: **NullPointerException**
 - O compilador deteta variáveis locais não inicializadas

```
public class BankAccount
{
    private String name;    // default constructor will set to null

    public void showStrings()
    {
        String localName;
        System.out.println(name.length());
        System.out.println(localName.length());
    }
}
```

Runtime Error:
java.lang.NullPointerException

**Compiler Error: variable localName
might not have been initialized**



Erro Comum (2)

■ Chamada a um construtor

- Não é possível invocar um construtor como outros métodos
- É 'invocado' automaticamente através de new

```
CashRegister register1 = new CashRegister();
```

- Não é possível invocar o construtor sobre um objeto existente:

```
register1.CashRegister(); // Erro
```

- Mas é possível criar um novo objeto usando uma referência existente

```
CashRegister register1 = new CashRegister();  
register1.newItem(1.95);  
CashRegister register1 = new CashRegister();
```



Erro Comum (3)

- Declarar um construtor como void
 - Os construtores não possuem retorno
 - No exemplo seguinte é criado um método com retorno do tipo void, mas **NÃO É** um construtor!
 - O compilador Java não considera isto como erro

```
public class BankAccount
{
    /**
     Não se trata de um construtor.
    */
    public void BankAccount( )
    {
        ...
    }
}
```

Não é um construtor...
É apenas um método sem retorno (void)

- Vimos já que podemos ter vários construtores com o mesmo nome
 - Requerem listas diferentes de parâmetros
- Na verdade qualquer método pode ser sobrecarregado (*overloaded*)
 - Métodos com o mesmo nome com diferentes parâmetros:

```
void print(CashRegister register)    { ... }  
void print(BankAccount account)     { ... }  
void print(int value)                { ... }  
void print(double value)             { ... }
```

Teste de uma Classe

- A classe CashRegister foi declarada mas...
 - não é possível executá-la – não possui o método main
- Contudo, pode fazer parte de um programa
- Devemos, no entanto, testá-la previamente através de:
 1. Testes unitários
 2. Uma classe de teste – possui o método **main**

```
public class CashRegisterTester
{
    public static void main(String[] args)
    {
        CashRegister c1 = new CashRegister();
        ...
    }
}
```

CashRegisterTester.java

```
1  /**
2   * This program tests the CashRegister class.
3   */
4  public class CashRegisterTester
5  {
6      public static void main(String[] args)
7      {
8          CashRegister register1 = new CashRegister();
9          register1.addItem(1.95);
10         register1.addItem(0.95);
11         register1.addItem(2.50);
12         System.out.println(register1.getCount());
13         System.out.println("Expected: 3");
14         System.out.printf("%.2f\n", register1.getTotal());
15         System.out.println("Expected: 5.40");
16     }
17 }
```

- Testar todos os métodos
 - Imprimir os resultados esperados
 - Imprimir os resultados obtidos
 - Comparar resultados

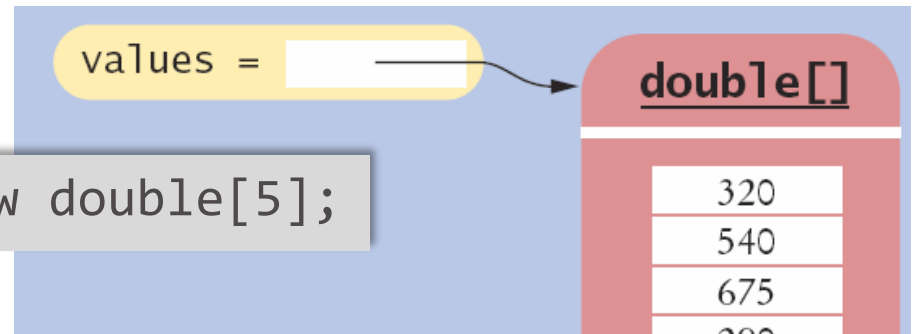
Program Run

```
3
Expected: 3
5.40
Expected: 5.40
```

Referências de Objetos

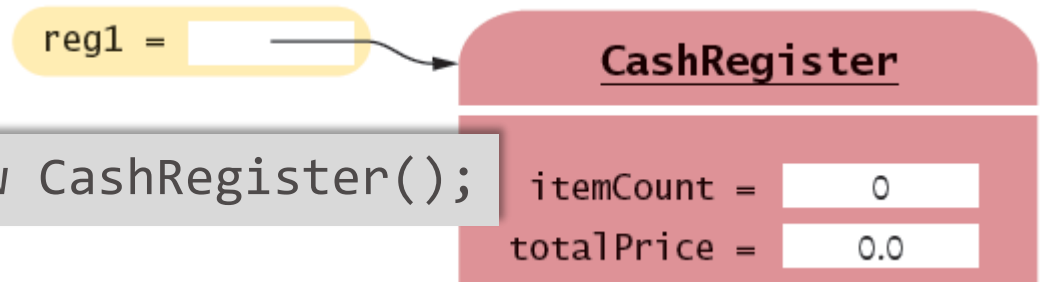
- Os objetos são semelhantes aos arrays porque possuem sempre variáveis de referência
 - Referência de um array

```
double[] values = new double[5];
```



- Referência de um objeto

```
CashRegister reg1 = new CashRegister();
```




A referência de um objeto especifica
a localização de memória do objeto

Referências Partilhadas


- Diferentes variáveis de objetos podem conter referências para o mesmo objeto


- Referência única

reg1 = 

CashRegister


```
CashRegister reg1 = new CashRegister();
```


itemCount = 

totalPrice = 


- Referências partilhadas


```
CashRegister reg2 = reg1;
```

reg1 = 

reg2 = 

CashRegister

itemCount = 

totalPrice = 

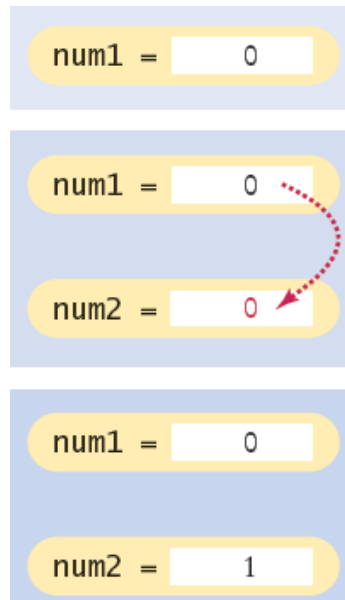
Os valores internos do objeto podem ser alterados através de qualquer uma das referências

Cópia de Referências vs. Tipos Primitivos

- Variáveis de tipos primitivos podem ser copiadas, mas funcionam de forma diferente das referências de objetos

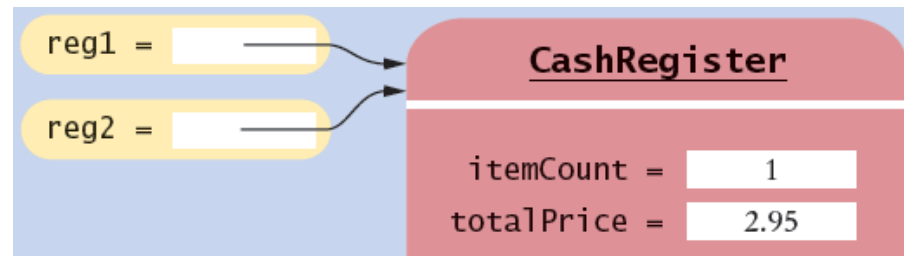
cópia primitiva (dois locais)

```
int num1 = 0;  
int num2 = num1;  
num2++;
```



cópia de referência (um único local)

```
CashRegister reg1 = new CashRegister;  
CashRegister reg2 = reg1;  
reg2.addItem(2.95);
```



Porquê? Tipos primitivos ocupam muito menos espaço do que objetos!

Referência null

- Uma referência pode apontar para nenhum objeto
 - Não é possível invocar métodos de um objeto através de uma referência null – originará um *run-time error*

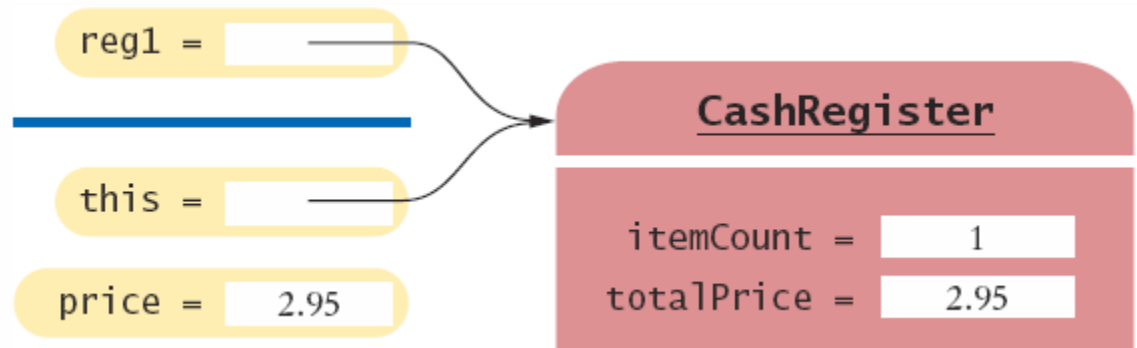
```
CashRegister reg = null;  
System.out.println(reg.getTotal());    // Runtime Error!
```

- Testar se uma referência é null antes de a usar:

```
String middleInitial = null; // A string não existe  
  
if (middleInitial == null)  
    System.out.println(firstName + " " + lastName);  
else  
    System.out.println(firstName + " " + middleInitial + ". "  
        + lastName);
```

Referência this

- Os métodos recebem o “parâmetro implícito” numa variável de referência chamada **this**
 - É uma referência ao objeto sobre o qual o método foi invocado:



- Ajuda a clarificar a utilização das variáveis de instância

```
void addItem(double price)
{
    this.itemCount++;
    this.totalPrice = this.totalPrice + price;
}
```

Referência this num Construtor

- Por vezes a referência **this** é usada nos construtores
 - Torna mais clara a intenção de alterar uma variável de instância

```
public class Student
{
    private int id;
    private String name;
    public Student(int id, String name)
    {
        this.id = id;
        this.name = name;
    }
}
```

Variáveis e Métodos static

- As variáveis podem ser declaradas como **static** na declaração da classe
 - Existirá apenas uma cópia de uma variável **static** que será partilhada por todos os objetos da classe

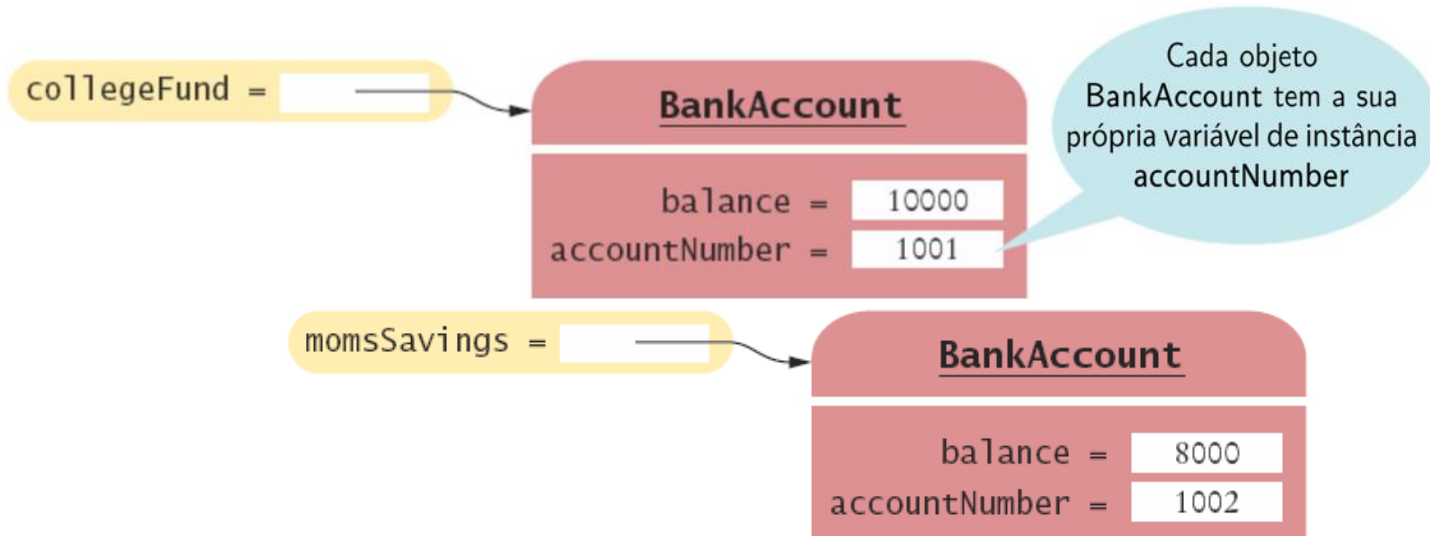
```
public class BankAccount
{
    private double balance;
    private int accountNumber;
    private static int lastAssignedNumber = 1000;

    public BankAccount()
    {
        lastAssignedNumber++;
        accountNumber = lastAssignedNumber;
    }
    ...
}
```

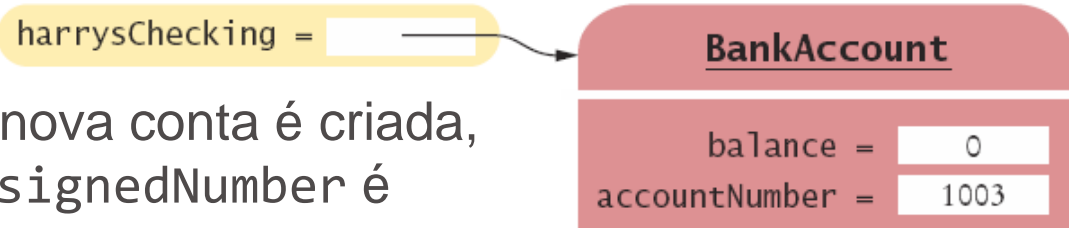


Os métodos de qualquer objeto da classe podem usar ou alterar o valor de uma variável **static**

Utilização de Variáveis static



- Exemplo:
 - Sempre que uma nova conta é criada, a variável `lastAssignedNumber` é incrementada pelo construtor
- Acesso à variável **static**:
 - `NomeClasse.nomeVariavel`



Existe apenas uma variável estática `lastAssignedNumber` para a classe `BankAccount`

`BankAccount.lastAssignedNumber = 1003`

- As APIs Java dispõem de várias classes que fornecem métodos que são usados sem instanciar objetos
 - A classe `Math` é um destes exemplos
 - `Math.sqrt(value)` é um método **static** que devolve a raiz quadrada de um valor
 - Não é necessário instanciar a classe `Math`
- Acesso aos métodos **static**:
 - `NomeClasse.nomeMetodo()`

Definição de Métodos `static`

- Definição de métodos `static`

```
public class Financial
{
    /**
     * Calcula a percentagem de uma quantia.
     * @param percentage a percentagem a aplicar
     * @param amount a quantia à qual a percentagem é aplicada
     * @return a percentagem calculada da quantia
     */
    public static double percentOf(double percentage, double amount)
    {
        return (percentage / 100) * amount;
    }
}
```

Os métodos `static` normalmente devolvem um valor. Apenas podem aceder a variáveis e métodos `static`.

- Invocar o método sobre a Classe, não sobre um objeto

```
double tax = Financial.percentOf(taxRate, total);
```

- Uma classe descreve um conjunto de objetos com o mesmo comportamento
 - Todas as classes têm uma interface pública: uma coleção de métodos através dos quais os objetos da classe podem ser manipulados
 - Encapsulamento é o ato de fornecer uma interface pública e esconder os detalhes de implementação
 - O encapsulamento permite mudanças na implementação sem afetar os utilizadores da classe

Resumo: Variáveis e Métodos

- As variáveis de instância de um objeto guardam os dados necessários à execução dos seus métodos
- Cada objeto de uma classe tem o seu próprio conjunto de variáveis de instância
- Um método de instância pode aceder às variáveis de instância do objeto sobre o qual atua
- Uma variável de instância `private` apenas pode ser acedida pelos métodos da sua própria classe
- As classes que contêm a declaração de variáveis `static` possuem uma única cópia da variável, partilhada entre todas as instâncias da classe

Resumo: Assinatura dos Métodos

■ Assinatura dos métodos

- Os cabeçalhos dos métodos e respetivos comentários podem ser usados para especificar a interface pública de uma classe
- Um método de modificação (**set**) altera o objeto sobre o qual opera
- Um método de acesso (**get**) não altera o objeto sobre o qual opera

■ Parâmetros dos métodos

- O objeto sobre o qual um método é aplicado é o parâmetro implícito
- Os parâmetros explícitos de um método são definidos na declaração do método

■ Construtores

- Um construtor inicializa as variáveis de instância de um método
- Um construtor é invocado quando um objeto é criado através do operador **new**
- O nome do construtor coincide com o da classe
- Uma classe pode ter vários construtores
- O compilador escolhe o construtor cujo conjunto de parâmetros corresponde aos argumentos de construção do objeto