Curso Java Básico

Uma introdução prática usando BlueJ

Entendendo as definições de classe

Examinando as classes por dentro

Principais conceitos a serem abordados

- Campos
- Construtores
- Métodos
- Parâmetros
- Variáveis
- Instrução de atribuição
- Instrução condicional

Máquinas de vender bilhetes: um exame externo

- Explorando o comportamento de uma máquina ingênua de vender bilhetes.
  - Utilize o projeto "*naive-ticket-machine*".
  - Máquinas que fornecem bilhetes de preço fixo.
    - Como esse preço é determinado?
  - Como o 'dinheiro' é inserido na máquina?
  - Como uma máquina monitora o dinheiro que é inserido?

Exercício

- **Máquina simples de bilhetes**
  - Inicie o **BlueJ**, abra o projeto *naive-ticket-machine* e crie uma instância de *TicketMachine* informando 500 (centavos) para valor do bilhete
  - Use *insertMoney* para simular a inserção de uma quantia na máquina
  - Use *getBalance* para verificar se a máquina registrou a quantia inserida
  - Use *printTicket* para simular a impressão do bilhete

Exercício

- **Máquina simples de bilhetes**
  - Qual o valor do saldo após imprimir o bilhete ?
  - O que acontece se você não inserir dinheiro suficiente e tentar imprimir um bilhete ?
  - O que acontece se você imprimir um bilhete após inserir mais dinheiro que o seu preço ? Você recebe algum reembolso ?

Máquinas de vender bilhetes: um exame interno

- Interagir com um objeto fornece dicas sobre seu comportamento.
- Examinar internamente permite determinar como esse comportamento é fornecido ou implementado.
- Todas as classes Java têm uma visualização interna semelhante.

Estrutura de uma classe básica

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

O empacotador externo da TicketMachine

```
public class ClassName
{
    Campos
    Construtores
    Métodos
}
```

O conteúdo de uma classe

Obs1:conteúdo básico
Obs2:definiremos nessa ordem
Obs3:comentários são opcionais

Comentários

- São inseridos no código-fonte para fornecer explicações para leitores humanos.
- Não têm nenhum efeito na funcionalidade da classe.

Comentários

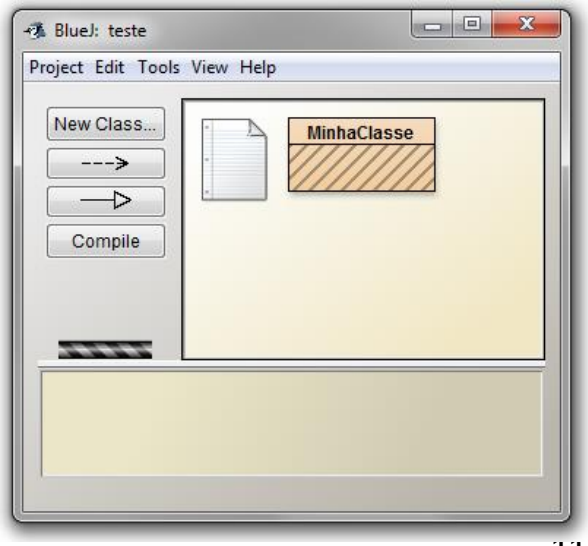
Comentário de múltiplas linhas (inicia com /* e finaliza com */)

```
/**
 * TicketMachine models a naive ticket machine
 * that issues flat-fare tickets.
 * The price of a ticket is specified
 * via the constructor.
 */
public class TicketMachine
{
    // The price of a ticket from this machine.
    private int price;
    ...
 }
```

Comentário de uma linha (inicia com //)

Exercício

- **Criando uma definição de classe**
  - Inicie o **BlueJ**, abra um projeto novo, crie uma classe e edite sua definição.
  - Quais são os campos, construtores e métodos gerados ?



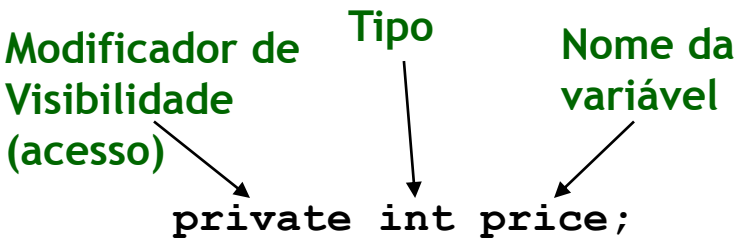
Campos

- Campos armazenam valores para um objeto.
- Eles também são conhecidos como **variáveis de instância.**
- Os valores dos campos definem o **estado** de um objeto.
- Utilize a opção *Inspect* para visualizar os campos de um objeto.

```
public class TicketMachine
{
    private int price;
    private int balance;
    private int total;

    Construtor e métodos omitidos.

}
```



Exercício

- **Campos**
  - Edite o código-fonte de *TicketMachine* e defina um campo *status* do tipo int e um campo isInitialized do tipo boolean.
  - Compile a classe *TicketMachine*.
  - Crie uma instância de *TicketMachine* e verifique o estado do objeto usando a função *Inspect*. Qual são os valores dos campos novos ? Por que ?
  - Edite o código-fonte de *TicketMachine* e retire os campos novos.

Construtores

- Construtores são chamados pelo operador **new**.
- Eles armazenam valores iniciais nos campos.
- Eles frequentemente recebem valores de parâmetros externos.
- Eles têm o mesmo nome das suas classes.
- Eles não têm retorno.
- Em Java, os campos não inicializados recebem um valor padrão.

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

Atribuição

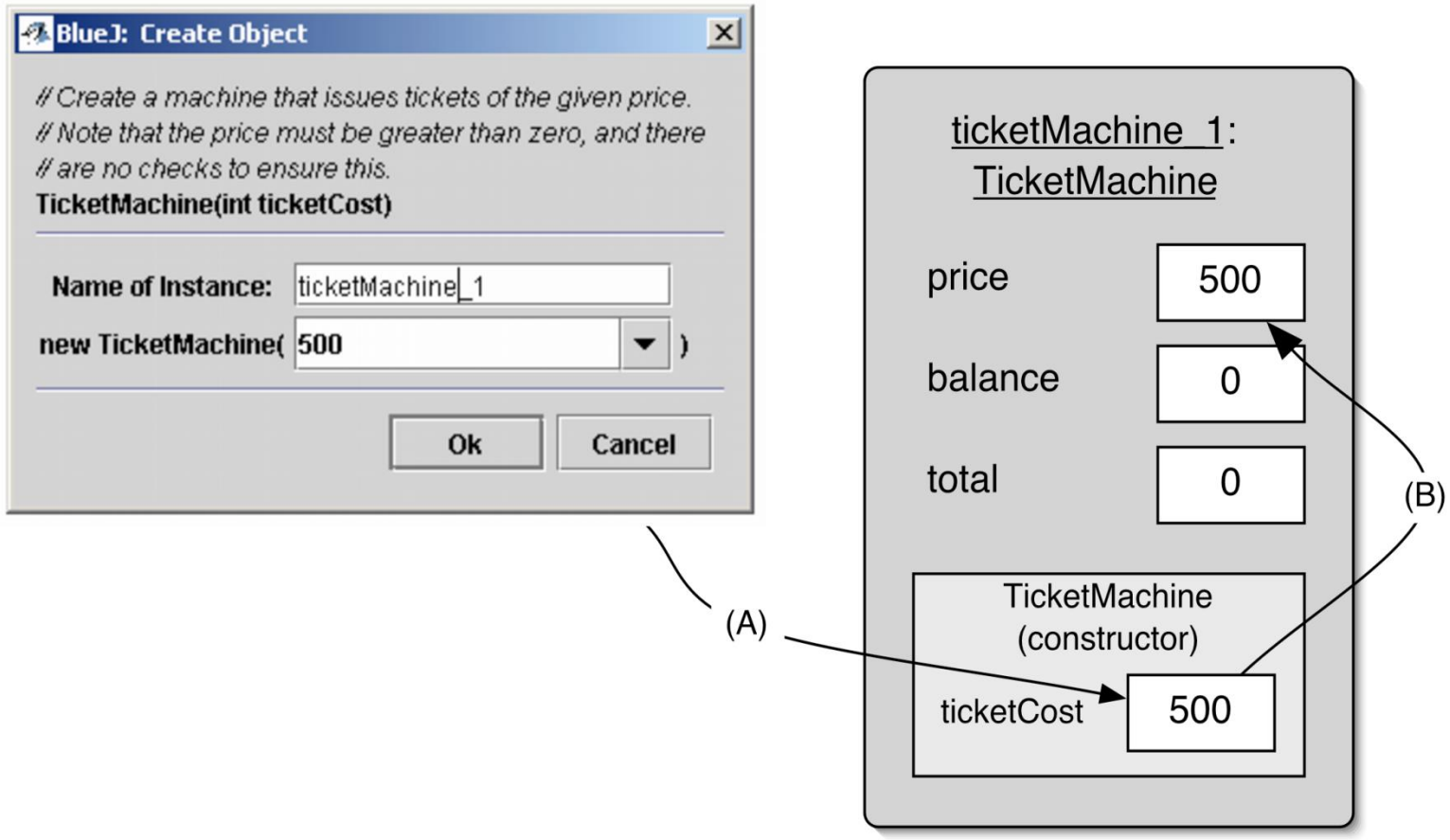
- Valores são armazenados em campos (e outras variáveis) via instruções de atribuição:

  variável = *expressão*;

  ```
  price = ticketCost;
  ```

- O tipo da expressão deve corresponder ao tipo da variável

Transmitindo dados via parâmetros



Parâmetros

- Os nomes dos parâmetros são chamados parâmetros formais e seus valores são chamados parâmetros reais (ou argumentos) :
  - *ticketCost* ← parâmetro formal
  - 500 ← parâmetro real

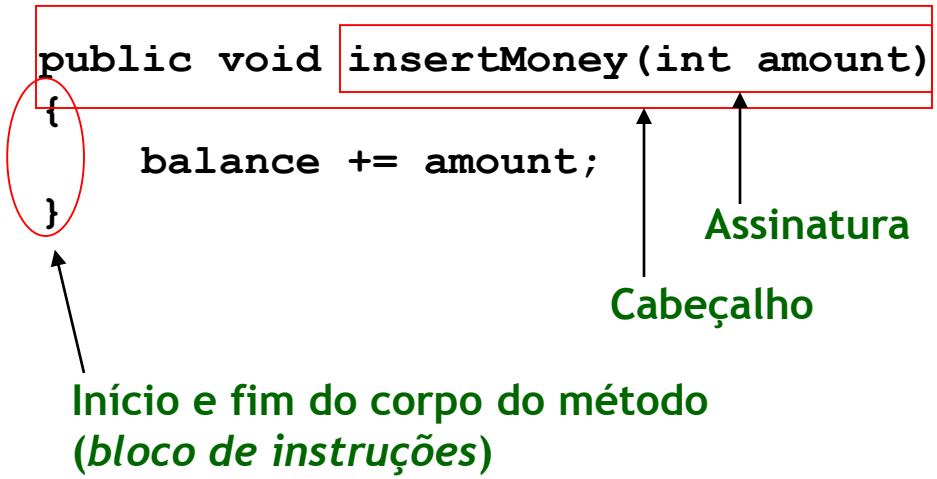
- Como são capazes de armazenar valores, os parâmetros formais são um tipo de variável
  - seu tempo de vida é o da execução do método ou construtor

Exercício

- **Construtores**
  - Edite o código-fonte de *TicketMachine*
  - Defina em *TicketMachine* um outro construtor com uma lista vazia de parâmetros e que configure o preço do bilhete para um valor de sua escolha.
  - Compile a classe *TicketMachine*, crie uma instância e teste o construtor.

Métodos

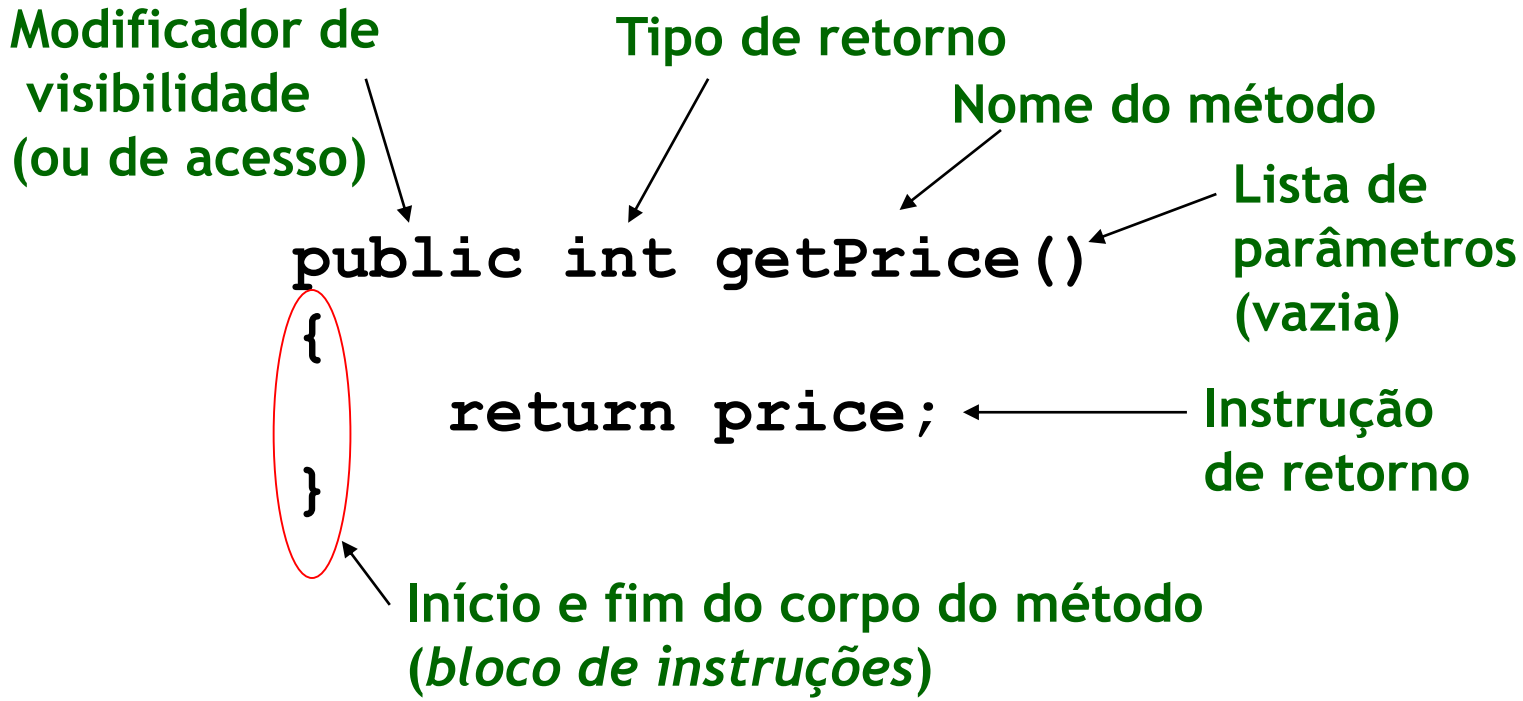
- Métodos implementam o ***comportamento*** dos objetos.
- Métodos (e construtores) têm uma estrutura que consiste em um ***cabeçalho*** e um ***corpo***.
- O cabeçalho compreende a ***assinatura*** *do* método (ou construtor).
- O corpo engloba as instruções.



Métodos de acesso

- Métodos de acesso fornecem informações sobre um objeto.
  - Geralmente contêm instruções de retorno.
  - Geralmente não recebem parâmetros.

Métodos de acesso



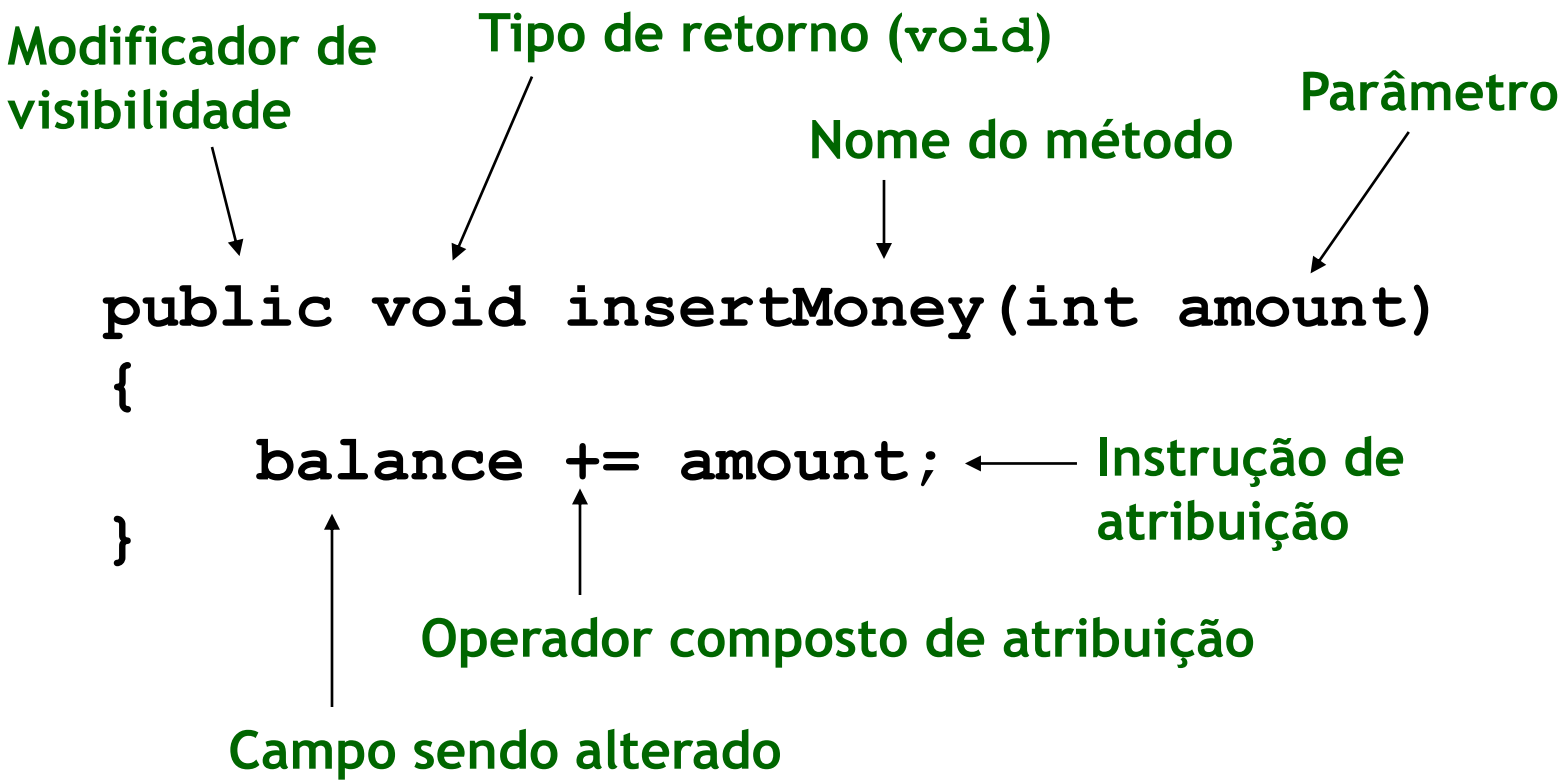
Exercício

- **Métodos de acesso**
  - Edite o código-fonte de *TicketMachine*
  - Os métodos *insertMoney* e *printTicket* têm instruções de retorno ? Você nota algo em seus cabeçalhos que possa sugerir por que eles não requerem instruções de retorno ?
  - Defina em *TicketMachine* um método de acesso chamado *getTotal* que retorne o valor do campo *total*.
  - Compile a classe *TicketMachine*, crie uma instância e teste o método.

Métodos modificadores

- Utilizados para *modificar* o estado de um objeto (alterando o valor de um ou mais campos).
  - Geralmente contêm instruções de atribuição.
  - Geralmente recebem parâmetros.

Métodos modificadores



Exercício

- **Métodos modificadores**
  - Como podemos afirmar que o método abaixo não é um construtor ?

    public void setPrice (ticketCost) {}
  - Edite o código-fonte de *TicketMachine* e defina o método *setPrice* de modo que atribua o valor do parâmetro *ticketCost* ao campo *price* (use a opção ***Insert method*** do menu ***Edit*** do ***Editor*** do **BlueJ**).
  - Compile a classe *TicketMachine*, crie uma instância e teste o método.

Classes imutáveis

- Classes sem métodos modificadores são chamadas *classes imutáveis*.
- Classes imutáveis têm uma vantagem importante: é seguro dar referências a seus objetos, pois seu valor não pode ser alterado inesperadamente.
- A classe **`String`** é imutável.

Imprimindo a partir de métodos

```
public void printTicket()
{
    // Simula a impressão de um bilhete.
    System.out.println("####################");
    System.out.println("# The BlueJ Line");
    System.out.println("# Ticket");
    System.out.println("# " + price + " cents.");
    System.out.println("####################");
    System.out.println();

    // Atualiza o total coletado com o saldo.
    total += balance;
    // Limpa o saldo.
    balance = 0;
}
```

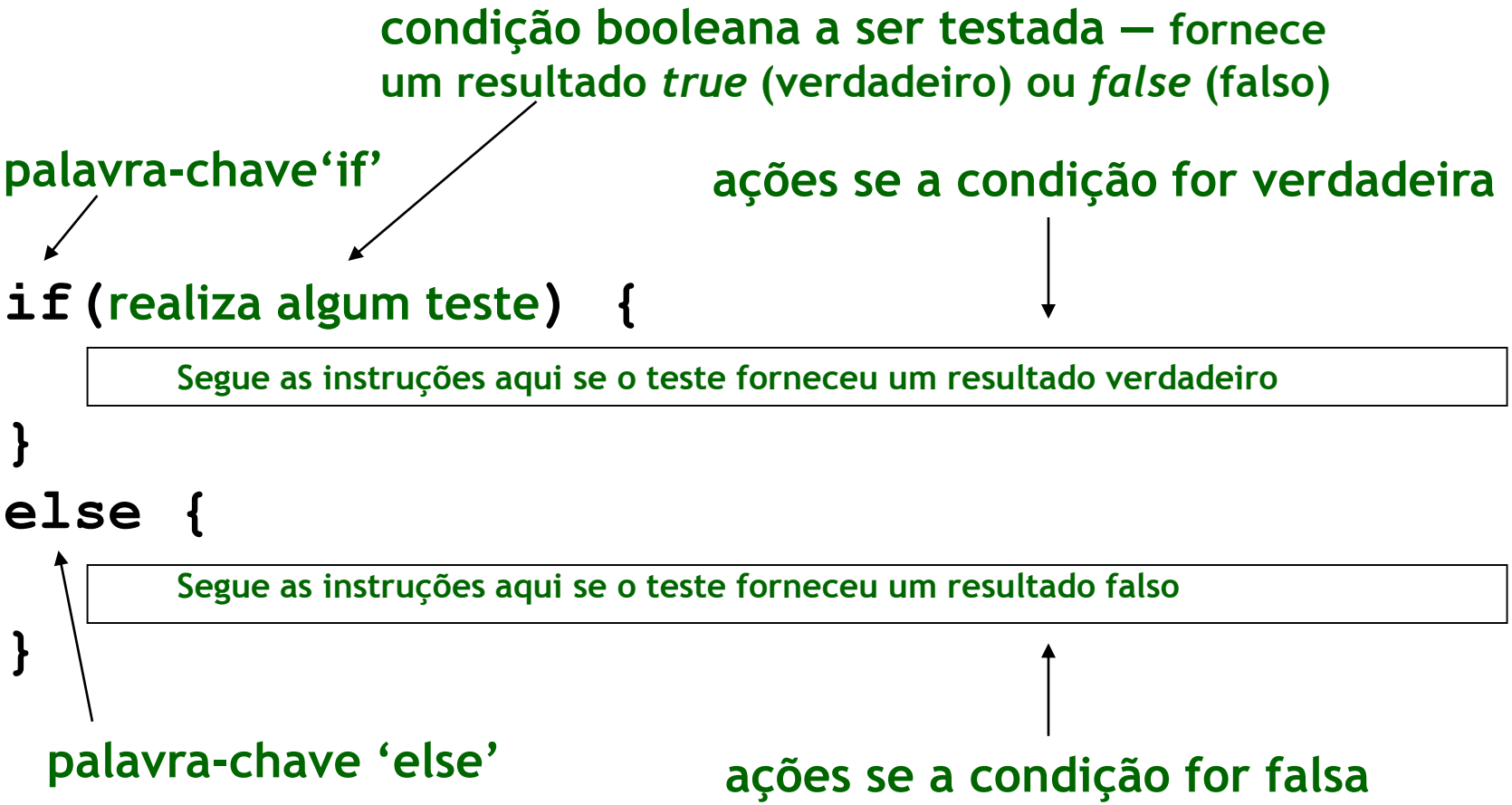
Refletindo sobre o projeto da máquina de vender bilhetes

- Seus comportamentos não são adequados por várias razões:
  - nenhuma verificação dos valores inseridos;
  - nenhum reembolso;
  - nenhuma verificação quanto a uma inicialização sensata.
- Como podemos melhorar isso?
  - Precisamos de um comportamento mais sofisticado.

Exercício

- **Máquina melhorada de bilhetes**
  - Abra o projeto *better-ticket-machine* e crie uma instância de *TicketMachine* informando 500 (centavos) para valor do bilhete
  - Verifique se esta versão da classe *TicketMachine* apresenta comportamentos adequados. Por exemplo,
    - Há verificação dos valores inseridos ?
    - Há reembolso de valores maiores que o bilhete ?

Fazendo escolhas



Testando quantia inserida

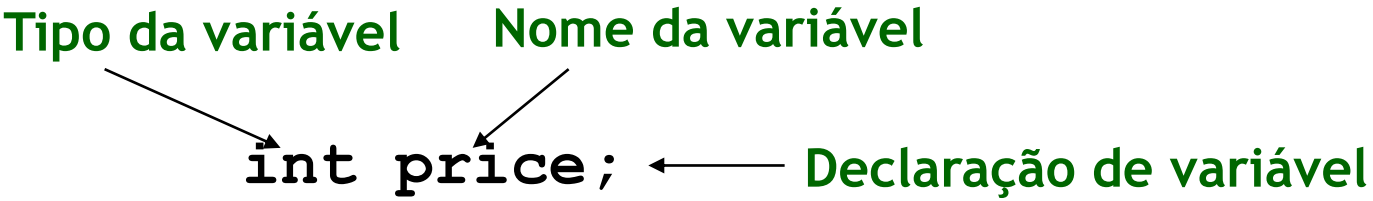
```
public void insertMoney(int amount)
{
    if(amount > 0) {
        balance += amount;
    }
    else {
        System.out.println("Use uma quantia positiva: "
            + amount);
    }
}
```

Testando saldo atual

```
public void printTicket()
{
    if(balance >= price) {
        // Detalhes da impressão omitidos.
        // Atualiza o total com o preço.
        total = total + price;
        // Reduz o saldo do preço.
        balance = balance - price;
    }
    else {
        System.out.println("Você deve inserir mais: "
        +  price - balance + " more cents.");
    }
 }
```

Variáveis

- Uma variável é uma área de memória.
- Essa área é alocada através de uma declaração de variável, que contém:



- Variáveis só assumem valores do tipo declarado.
- Variáveis armazenam um único valor; qualquer valor anterior é perdido.

Variáveis

- O ***escopo*** de uma variável é a seção do código-fonte onde a variável pode ser acessada.
- O ***tempo de vida*** de uma variável descreve quanto tempo a variável continuará a existir antes de ser destruída.

Variáveis locais

- Campos são um tipo de variável. Eles:
  - existem por toda a vida de um objeto;
  - são declaradas fora dos métodos*;
  - são inicializadas automaticamente (por omissão); e
  - são acessíveis dentro dos métodos*.
- Métodos podem incluir variáveis de vida mais curta (variáveis locais). Elas:
  - existem durante a execução do método*;
  - são declaradas dentro de um método*;
  - não são inicializadas automaticamente; e
  - são acessíveis de dentro do método*.

** (ou construtores)*

Variáveis locais

Declaração de uma variável local

```
public int refundBalance()
{
    int amountToRefund;
    amountToRefund = balance;
    balance = 0;
    return amountToRefund;
}
```

Exercício

- **Variáveis locais**
  - Edite o código-fonte do construtor de *TicketMachine*
  - Substitua a primeira linha por

    int price = ticketCost;
  - Verifique o valor do campo *price* com a função ***Inspect***. Porque ele foi inicializado com o valor default ?
  - Desfaça a alteração.

Revisão (1)

- O corpo das classes contém campos, construtores e métodos.
- Campos armazenam valores que determinam o estado de um objeto.
- Construtores inicializam objetos.
- Métodos implementam o comportamento dos objetos.

Revisão (2)

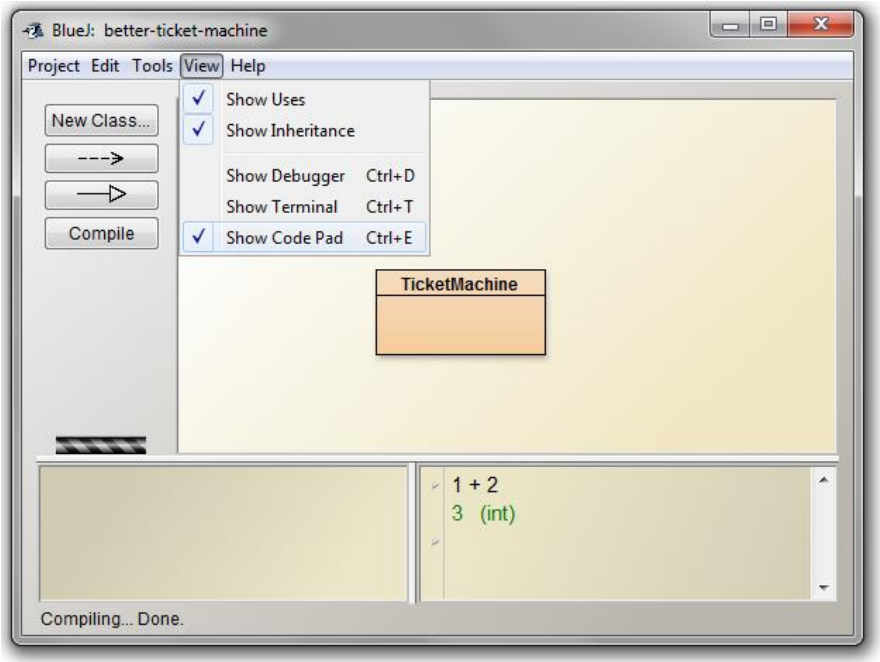
- Campos, parâmetros e variáveis locais são variáveis.
- Campos persistem pelo tempo de vida de um objeto.
- Parâmetros são utilizados para receber valores em um construtor ou método.
- Variáveis locais são utilizadas para armazenamento temporário de curta duração.

Revisão (3)

- Objetos podem tomar decisões via instruções condicionais (if).
- Um teste de verdadeiro ou falso permite que uma entre duas ações alternativas seja tomada.

Testando expressões

- No **BlueJ**, o resultado de expressões pode ser testado usando o *CodePad*. Podemos exibí-lo através do menu **View**.



Exercício

- **Testando expressões:o CodePad**
  - Usando o **CodePad**, teste as seguintes expressões:
    - `1 + 2`
    - `1 + 2 + "a"`
    - `"a" + 1 + 2`
    - `"Code" + "Pad"`

Concatenação de Strings

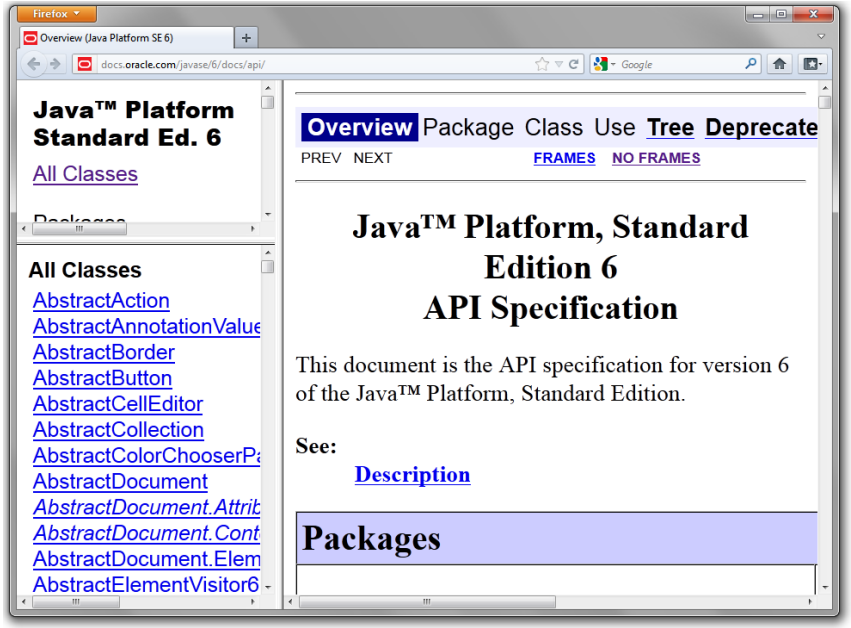
- Em Java, o funcionamento do operador + depende de seus operandos (sobrecarga de operador):
  - adiciona números
  - concatena strings

Exercício

- **Testando expressões:o CodePad**
  - Usando o resultado da expressão "Code"+"Pad", arraste o símbolo de objeto à esquerda para a bancada de objetos e o inspecione. Qual é o seu tamanho ?
  - Avalie as seguintes expressões:
    - "CodePad".length()
    - "CodePad".substring(0,4)
    - "CodePad".substring(4,7)
    - "CodePad".substring(4)
    - "CodePad".substring(4,8)

Lendo a documentação da API Java

- No **BlueJ**, usamos o item **Java Class Libraries** do menu **Help** para acessar, via navegador web, a documentação da API Java.

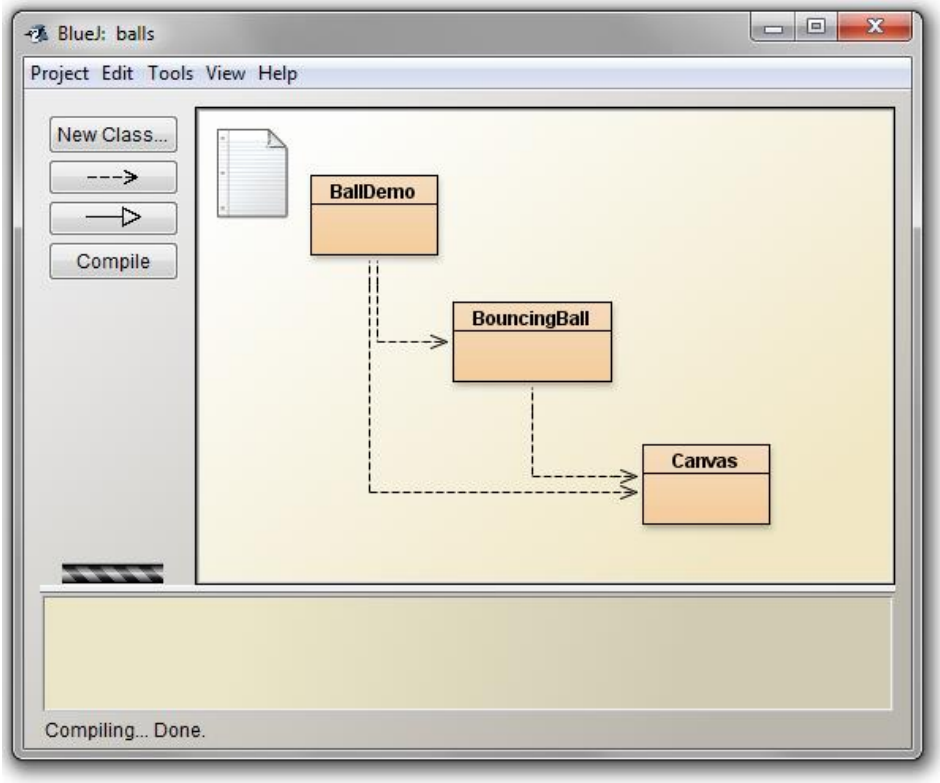


Exercício

- **Lendo a documentação da API**
  - Usando o item Java Class Libraries do menu Help, localize a documentação da classe *String*.
  - Localize a descrição dos métodos *length* e *substring*.

Uma animação de bolas quicando

- Exploraremos uma animação que simula duas bolas quicando.



Exercício

- **Animação bolas quicando**
  - Feche o projeto anterior, abra o projeto *balls* e crie uma instância de *BallDemo*.
  - Chame o método *bounce*. Qual é o comportamento apresentado pelas bolas ?
  - Edite a classe *BallDemo* e examine o método *bounce*. Que condição encerra o movimento das bolas ?

Exercício

- **Animação bolas quicando**
  - Edite a classe *BoucingBall* e localize uma definição de gravidade (um inteiro simples).
  - Experimente aumentar (ou diminuir) esse valor, compile e execute a animação. Você observa alguma alteração ?

Código-fonte: BouncingBall

```
public class BouncingBall
{
    private static final int GRAVITY = 3;

    ...

    private int xPosition;
    private int yPosition;

    ...
}
```

Constante

Variável de classe

Variável de instância

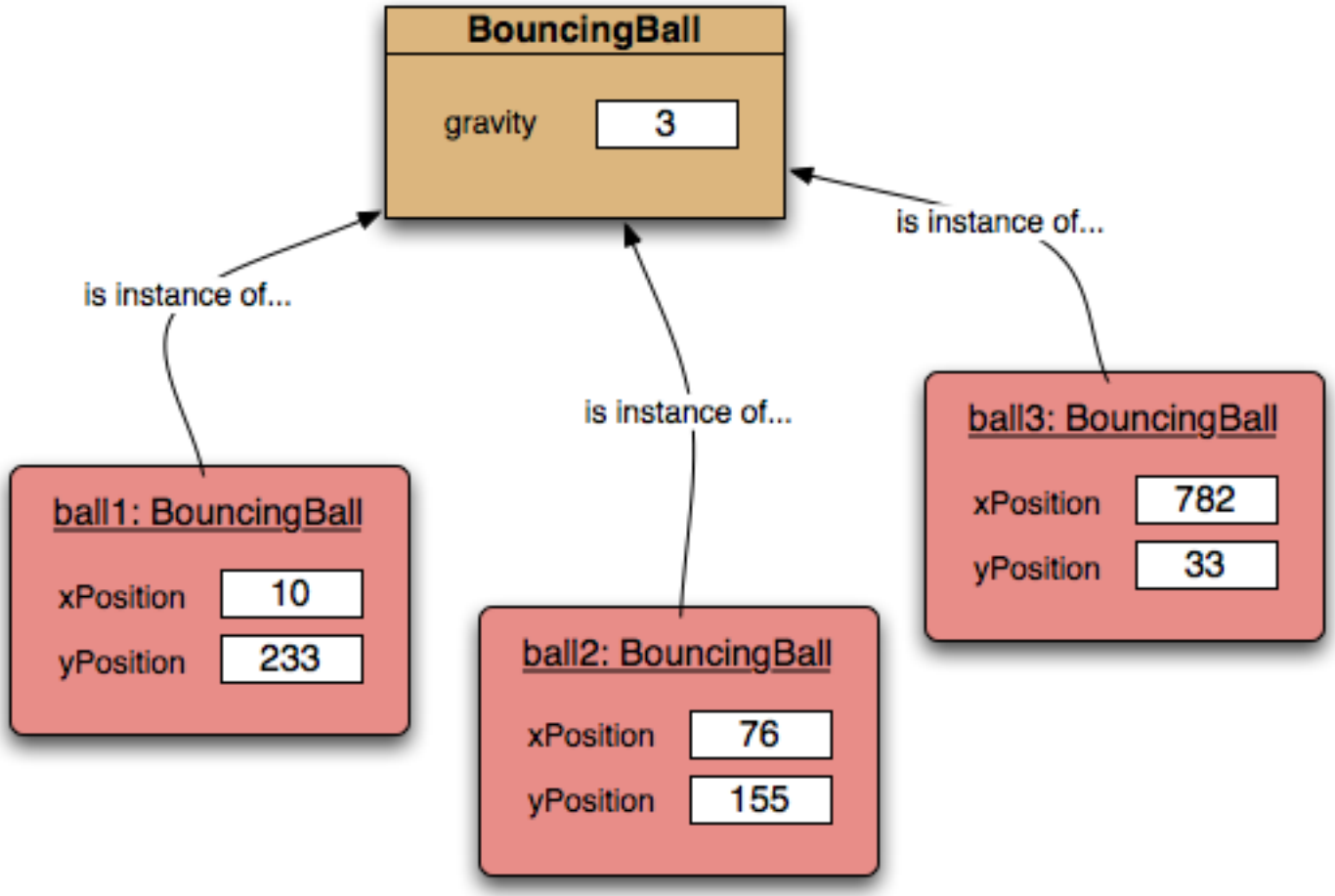
Palavra-chave *static*

- A palavra-chave ***static*** define membros (campos ou métodos) de classe.
- Variáveis de classe não são áreas replicadas em cada instância; são uma áreas únicas referentes à definição da classe.
- Métodos de classe não são chamados a partir de uma referência de instância; são chamados a partir do nome da classe.

Variáveis de classe

- Campos estáticos são conhecidos como variáveis de classe.
- Há sempre uma única cópia de uma variável de classe, independentemente da quantidade de instâncias criadas.
- Muito utilizadas se houver necessidade de um único valor para todas as instâncias.
- O código-fonte da classe pode acessar (ler e configurar) esse tipo de variável da mesma forma que uma variável de instância.

Variáveis de classes



Palavra-chave *final*

- A palavra-chave ***final*** define membros não modificáveis ou classes não extensíveis.
- Variáveis ***final*** não podem ser alteradas.
- Métodos ***final*** não podem ser sobrescritos.
- Classes ***final*** não podem ser extendidas.

Constantes

- Constantes (variáveis final em Java) são áreas de memória semelhantes a variáveis, mas não podem ter seu valor alterado.
- Por convenção, são nomeadas em letras maiúsculas.
- Frequentemente, constantes são aplicadas a todas as instâncias de uma classe.

```
private static final int GRAVITY = 3;
```

Métodos de classe

- Métodos estáticos são conhecidos como métodos de classe.
- Métodos de classe podem ser invocados sem nenhuma instância da classe. Devido a isso, métodos de classe não podem:
  - acessar campos de instância
  - invocar métodos de instância
- Métodos estáticos são invocados a partir do nome da classe, por exemplo:

```
System.currentTimeMillis()
```

Exercício

- **Lendo a documentação da API**
  - Localize a documentação das classes *Math* e *System* do pacote *java.lang*.
  - Usando o **CodePad**, teste as seguintes expressões:

```
Math.PI
Math.ceil(1.23456789)
Math.pow(5,3)
System.currentTimeMillis()
System.getProperty("os.name")
System.getProperty("user.name")
```

Executando sem o BlueJ

- No **BlueJ**, em geral criamos um objeto e invocamos um de seus métodos. Sem o **BlueJ**, a aplicação iniciará sem nenhum objeto criado.
- As classes são as únicas coisas que temos inicialmente. Então, o primeiro método que pode ser invocado deve ser um método de classe.

O método *main*

- Na plataforma **Java**, iniciar uma aplicação é bem simples: o usuário especifica a classe que deve ser iniciada e a plataforma **Java** invocará um método chamado **`main`** nesta classe.
- O método **`main`** deve ter o cabeçalho:

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

- O corpo do método **`main`** pode conter quaisquer instruções, mas o bom estilo dita que ele seja pequeno e não contenha nada que faça parte da lógica da aplicação.

Exercício

- **Executando sem o BlueJ**
  - Em qual classe devemos implementar o método *main* para executarmos a animação de bolas quicando sem o **BlueJ** ? O que o método *main* deve fazer para executar a animação ?
  - Implemente o método *main* na classe apropriada e compile.
  - Invoque o método *main* sem criar nenhuma instância da classe.

Exercício

- **Executando sem o BlueJ**
  - Usando o interpretador de comandos do sistema operacional, acesse o diretório do projeto *balls* e execute a animação com o comando **java BallDemo.**

Revisão (4)

- Além de variáveis de instância, variáveis locais e parâmetros, podemos ter variáveis de classe.
- Variáveis de classe são áreas únicas que são compartilhadas por todas as instâncias da classe.
- Constantes são áreas semelhantes a variáveis, mas não são alteradas.

Revisão (5)

- Métodos de classe (ou estáticos) podem ser invocados sem existir nenhuma instância da classe.
- O método estático **`main`** tem como função iniciar um aplicativo Java.

Contatos

Câmara dos Deputados

CENIN - Centro de Informática

Carlos Renato S. Ramos

carlosrenato.ramos@camara.gov.br