INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
RIO GRANDE DO NORTE

Programação Orientada a Objeto

Aula 5
Entendendo as definições de classe

Prof. Pedro Baesse
pedro.baesse@ifrn.edu.br

Roteiro

- Campos
- Construtores
- Métodos
- Parâmetros
- Atribuição
- Instruções condicionais

Máquinas de vender bilhetes

- Explorando o comportamento de uma máquina simples de vender bilhetes
  - Vendem bilhetes de vários tipos
    - Por ser simples, vende somente um tipo
  - Soma os valores de bilhetes vendidos em um dia
  - Será aprimorada com o tempo

Máquinas 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?

Máquinas de vender bilhetes

- Utilize o projeto “naive-ticket-machine”
- Examine e experimente os métodos existentes: *getBalance*, *getPrice*, *insertMoney* e *printTicket*
- *getPrice*: Retorna o valor do bilhete vendido
- Use o *insertMoney* para simular a venda de um bilhete e use *getBalance* para verificar se o valor da venda foi registrada

Máquinas de vender bilhetes

- Podem ser inseridas valores separadamente como moedas e notas
- Tente inserir um valor exato. O bilhete não é gerado automaticamente, então use o método *printTicket.* O que aconteceu?
- Qual o valor é retornado se você verificar o saldo da máquina depois que ela imprimiu um bilhete?

Máquinas de vender bilhetes

- Qual o comportamento da máquina quando não é colocado o valor exato?
- Você nota alguma coisa estranha no comportamento da máquina?
- Você recebe reembolso caso passe do valor?
- O que acontece se você não inserir o suficiente e depois imprimir o bilhete?

Máquinas de vender bilhetes

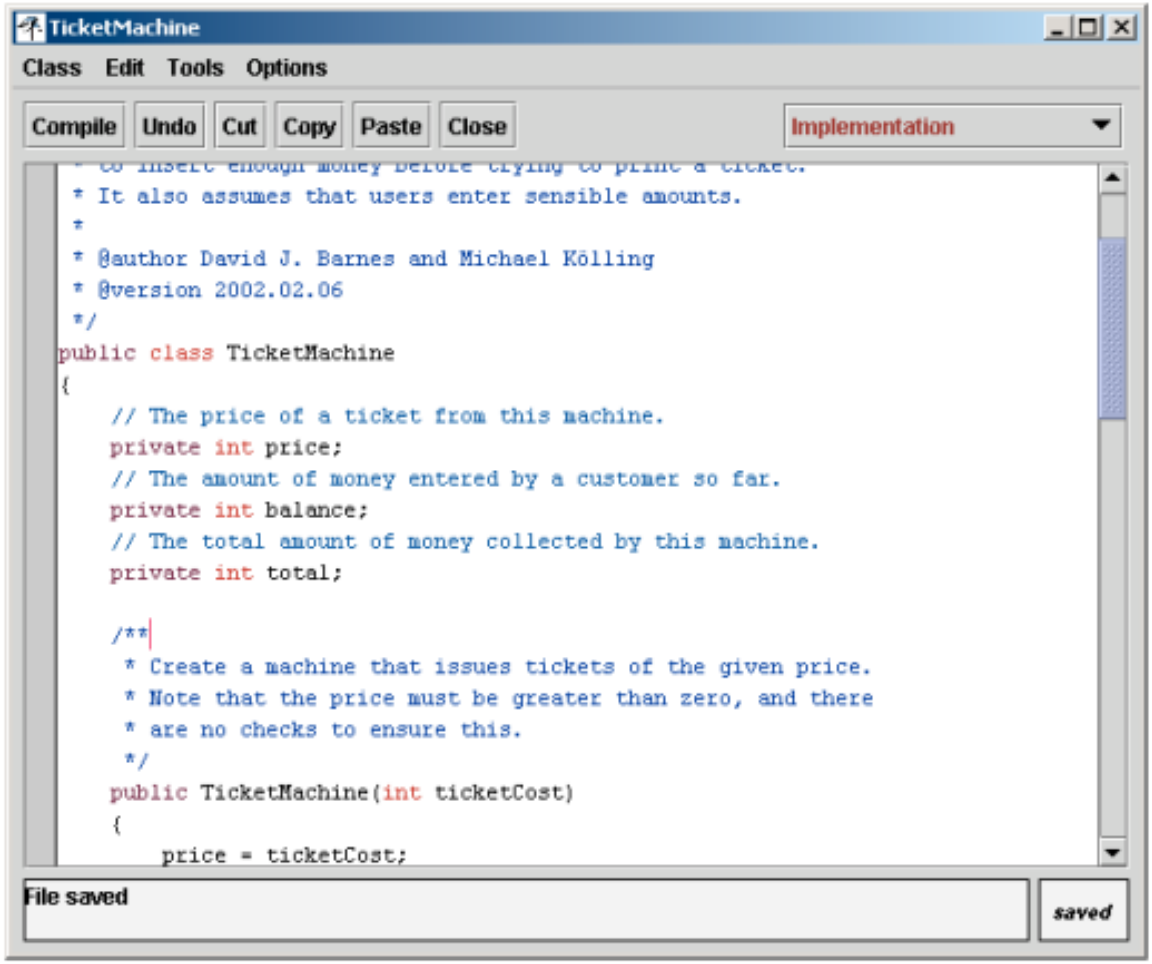
- Análise e entenda bem a *TicketMachine.* Interaja com a bancada de objetos

- Crie outra máquina vendendo o bilhete por um valor diferente. Compre um bilhete. O bilhete parece diferente?

Máquinas de vender bilhetes

- 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

Máquinas de vender bilhetes



INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA RIO GRANDE DO NORTE

Estrutura de uma classe básica

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

O empacotador externo da TicketMachine

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

O conteúdo de uma classe

Exercício

- Escreva como você achar que pareceriam as camadas externas das classes *Student* e *LabClass* – não se preocupe com a parte interna!

Campos, construtores e métodos

- Os **campos** armazenam dados para uso de cada objeto
- Os **construtores** permitem que cada objeto seja configurado adequadamente quando ele é criado
- Os métodos implementam o comportamento dos objetos

Campos, construtores e métodos

- Campos, construtores e métodos podem ser escritos em qualquer ordem mas adota-se a seguinte ordem por padrão

```
Public class NomeDaClasse
{
        Campos
        Construtores
        Métodos
}
```

Exercício

- Fazer um pequena lista com os campos, construtores e métodos da classe *Ticket-Machine*
  - Dica: Há apenas um construtor!
- Qual recurso torna o construtor significativamente diferente dos métodos??

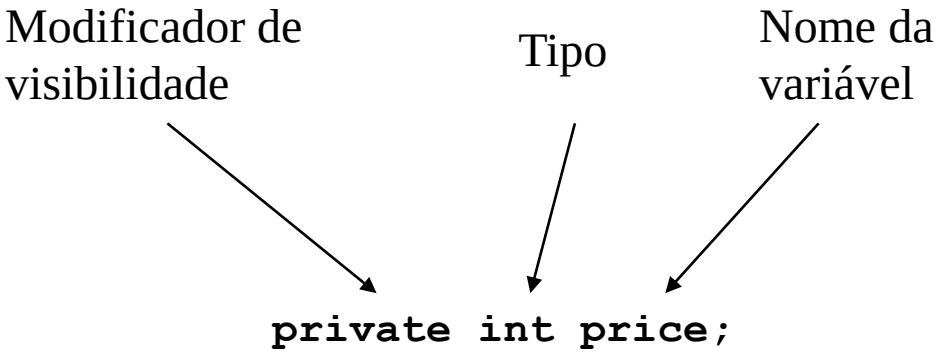
Campos

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

Campos

```
Public class TicketMachine
{
    // Preço de um bilhete
    private int price;
    private int balance;
    private int total;

    Construtor e métodos omitidos.
}
```



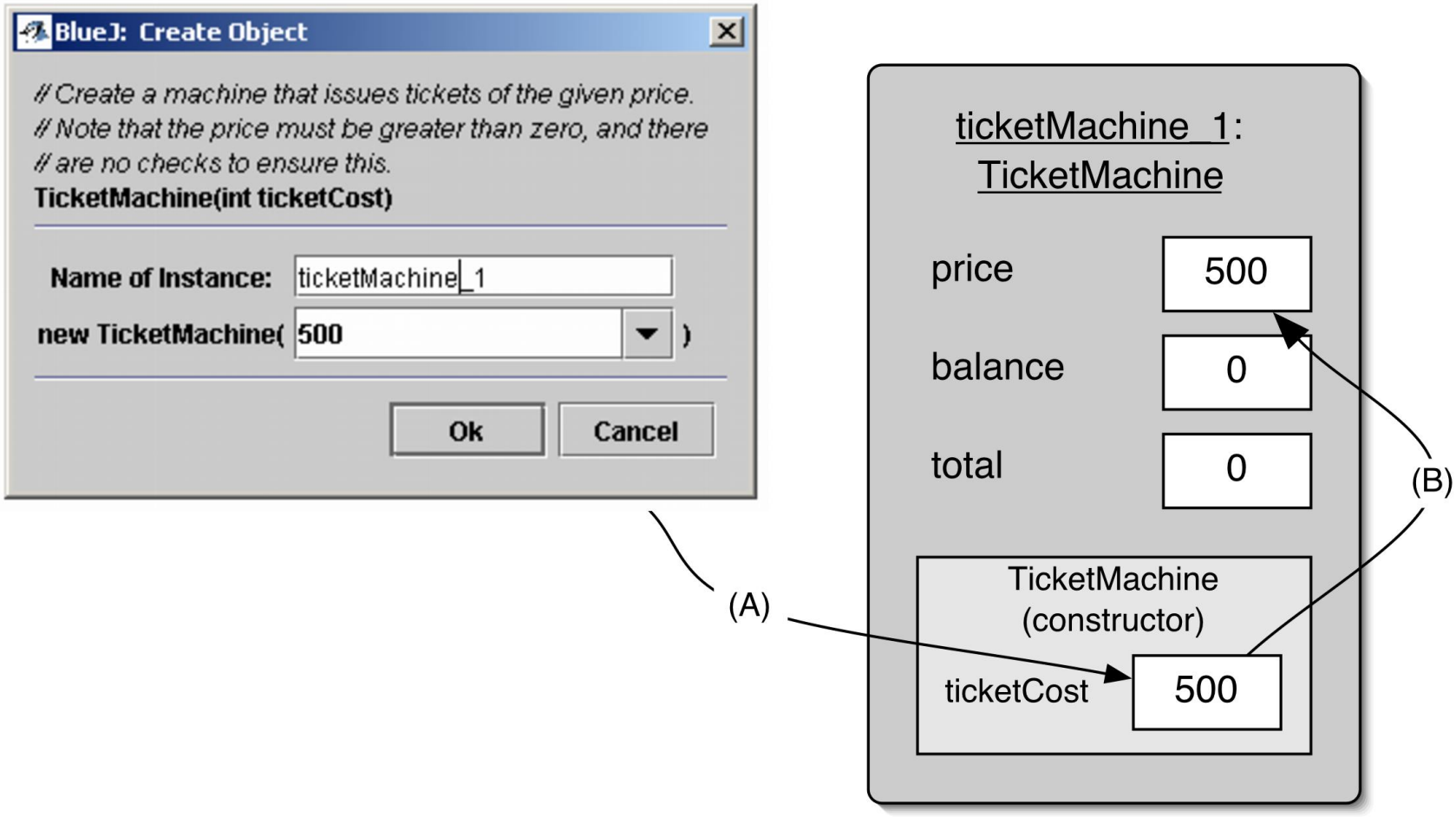
Construtores

- **Construtores** inicializam um objeto
- Eles têm o mesmo nome das suas classes
- Eles armazenam **valores iniciais** nos campos
- Eles frequentemente recebem valores de **parâmetros externos** nesses campos

Construtores

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

Transmitindo dados via parâmetros



Escopo e Tempo de Vida

- O **escopo** de uma variável define a seção de código-fonte de 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

Atribuição

- Valores são armazenados em campos (e outras variáveis) via instruções de atribuição
  - variável = expressão;
  - – price = ticketCost;
- Uma variável armazena um único valor, portanto, qualquer valor anterior é perdido

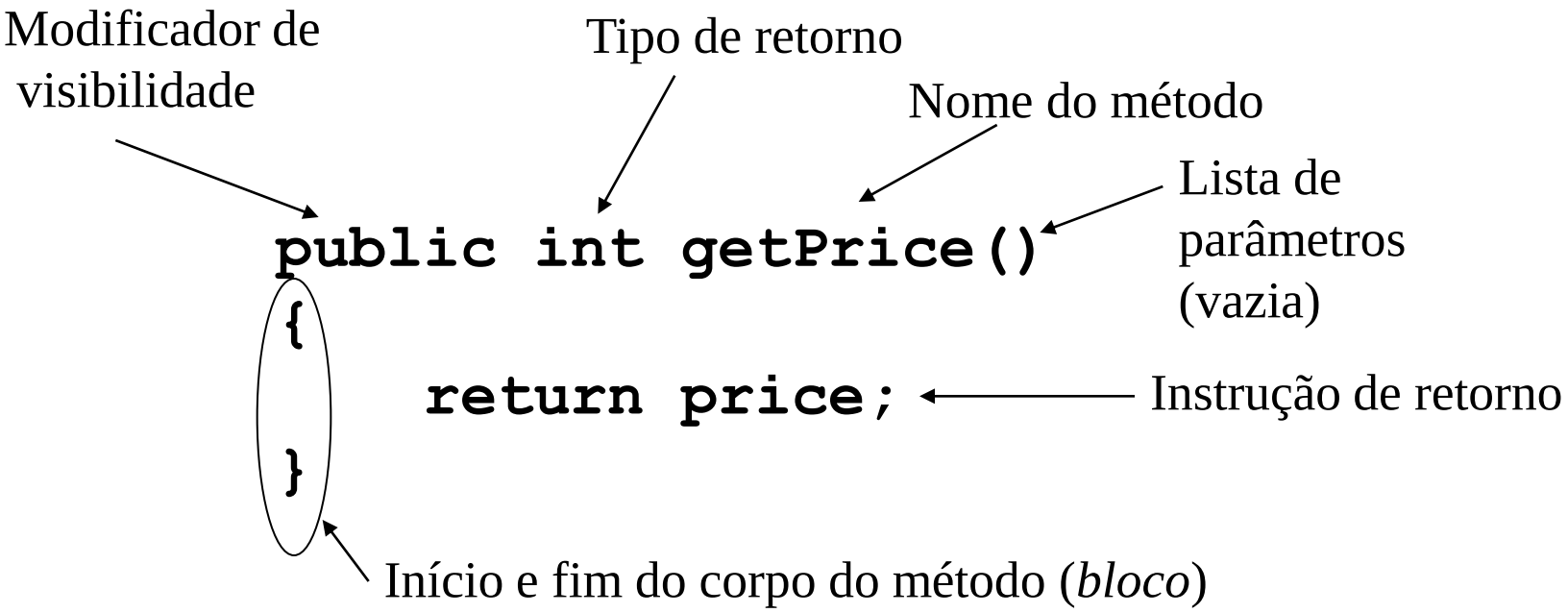
Métodos de acesso (get)

- Métodos implementam o comportamento dos objetos
- Métodos de acesso fornecem informações sobre um objeto
- 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.

Métodos de acesso (get)



Método X Construtor

- public int getPrice()
- public TicketMachine(int ticketCost)

Exercício

- Se uma chamada para *getPrice* puder ser caracterizada como “Quanto custam os bilhetes?”, como poderia ser caracterizada uma chamada *getBalance*?

- Defina um método de acesso, *getTotal*, que retorne o valor do campo *total*.

- Tente remover a instrução de retorno do corpo de *getPrice* e tente compilar. Que erro ocorre?

Exercício

- Compare as assinaturas de método *getPrice* e *printTicket.* Além dos nomes, qual a principal diferença?

- Os métodos *insertMoney e printTicket* não têm instrução de retorno? O que isso quer dizer? Aonde isso é sugerido na assinatura dos métodos?

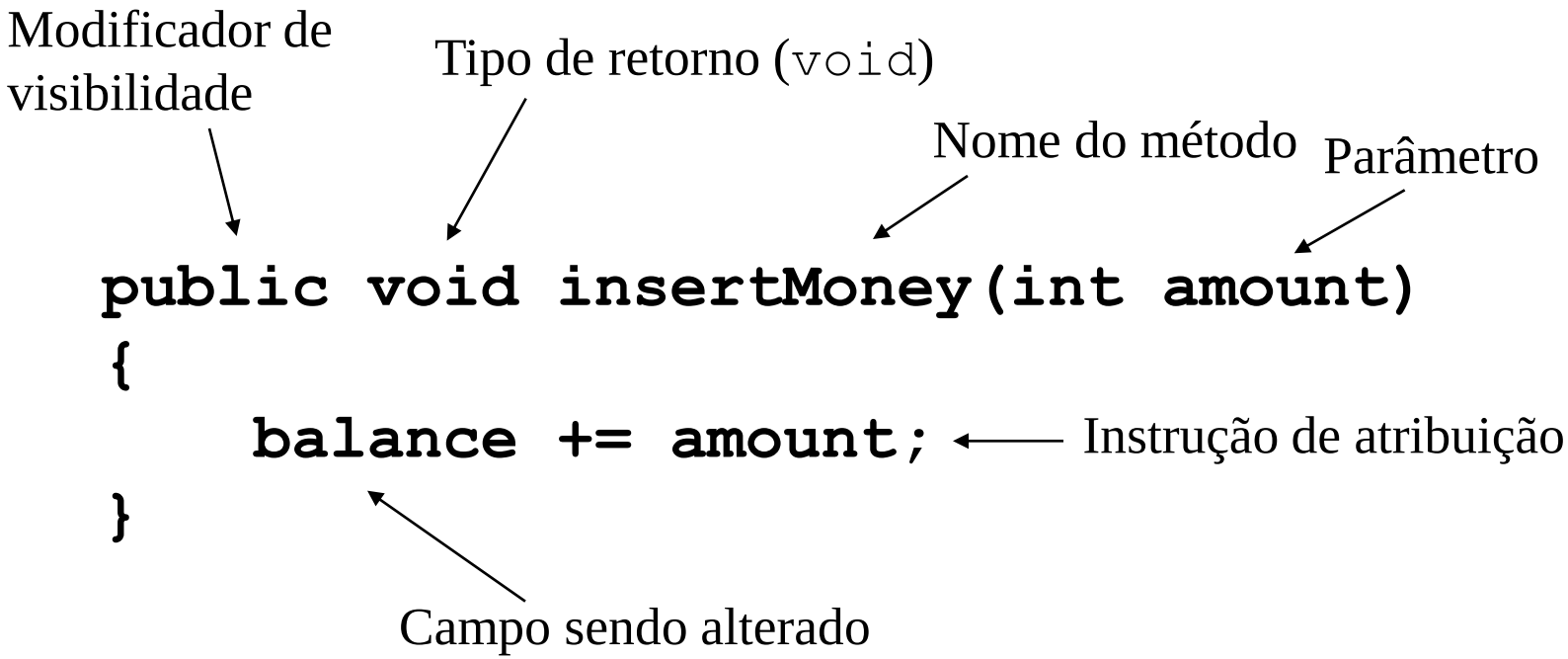
Métodos modificadores

- Eles têm uma estrutura de método semelhante
  - cabeçalho
  - corpo

- Utilizados para modificar o estado de um objeto

- Alcançados por meio da modificação do valor de um ou mais campos
  - Geralmente contêm instruções de atribuição
  - Geralmente recebem parâmetros

Métodos modificadores



Exercício

- Com uma máquina de tickets criada, chame o método *insertMoney* e observe o saldo com *getBalance*. Agora chame *insertMoney* novamente e veja o novo saldo.

- Altere o método *insertMoney* "+=" para "=". Compile e faça os passos anteriores novamente. Qual foi valor do saldo?
  - Lembre-se mudar de volta para "+="

- variável = variável + expressão;
  - balance = balence + amount;

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;
}
```

Imprimindo a partir de métodos

- O método **System.out.println** imprime seus parâmetros no terminal de texto
- **System.out.println("# The BlueJ Line");**
- **System.out.println("# " + price + " cents.");**
  - Uma String “# ” (note o espaço)
  - Valor da variável *price* (não há aspas duplas)
  - Uma String “ cents.” (note o espaço)
- Quando usado entre uma string e qualquer outra coisa o operador “+” concatena os valores

Exercício

- Adicione um método *prompt* à classe *TicketMachine*. Retorno *void* e não aceitar nenhum parâmetro e imprimir algo como:
  - Coloque a quantidade correta de dinheiro

- Crie um método *showPrice* Retorno *void* e não aceitar nenhum parâmetro e imprimir algo como:
  - O preço do bilhete é zxy centavos
  - xzy deve ser o valor do campo price e não uma valor escrito literalmente

Exercício

- Crie duas máquinas de bilhetes com valores de bilhetes diferentes e chame *showPrice*. As saídas dos métodos são as mesmas? Por que?

- O que aconteceria se o *price* do método *printTicket* estivesse com aspas duplas?
  - **System.out.println("# " + price + " cents.")**

- E **System.out.println("# price cents.")**?

- Qualquer uma das versões anteriores poderiam ser usadas para escrever o valor do bilhete em máquinas diferentes?

Testando Conhecimentos!

- Modifique o construtor de *TicketMachine* para que não receba nenhum parâmetro. Em vez disso fixe o valor do bilhete em 1000 centavos. Que efeito tem isso quando são construídas várias máquinas de vender bilhetes?

Testando Conhecimentos!

- Crie um método *empty*, que simule o efeito de remoção de todo o dinheiro da máquina. Esse método deve ter retorno do tipo *void*, e deve apenas definir o campo *total* para zero. Esse método precisa retornar algum parâmetro? Teste ele criando uma máquina, inserindo algum dinheiro, imprimindo bilhetes e verificando o total e depois esvaziando a máquina. Esse método é modificador ou de acesso?

Testando Conhecimentos!

- Implemente o método *setPrice* que defina o preço do bilhete para um novo valor, recebendo o novo valor como parâmetro. Teste ele criando uma máquina, mostrando o preço dos bilhetes, alterando o preço e mostrando o novo preço. Esse método é modificador?

- Crie dois construtores, um recebendo o valor dos bilhetes como parâmetro e outro não aceita nenhum parâmetro e define o valor do bilhete para um valor a sua escolha! Teste o comportamento deles criando máquinas de bilhete diferentes.

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
  - **Vamos abrir o projeto *better-ticket-machine* e comparar com o anterior. O que é diferente?**

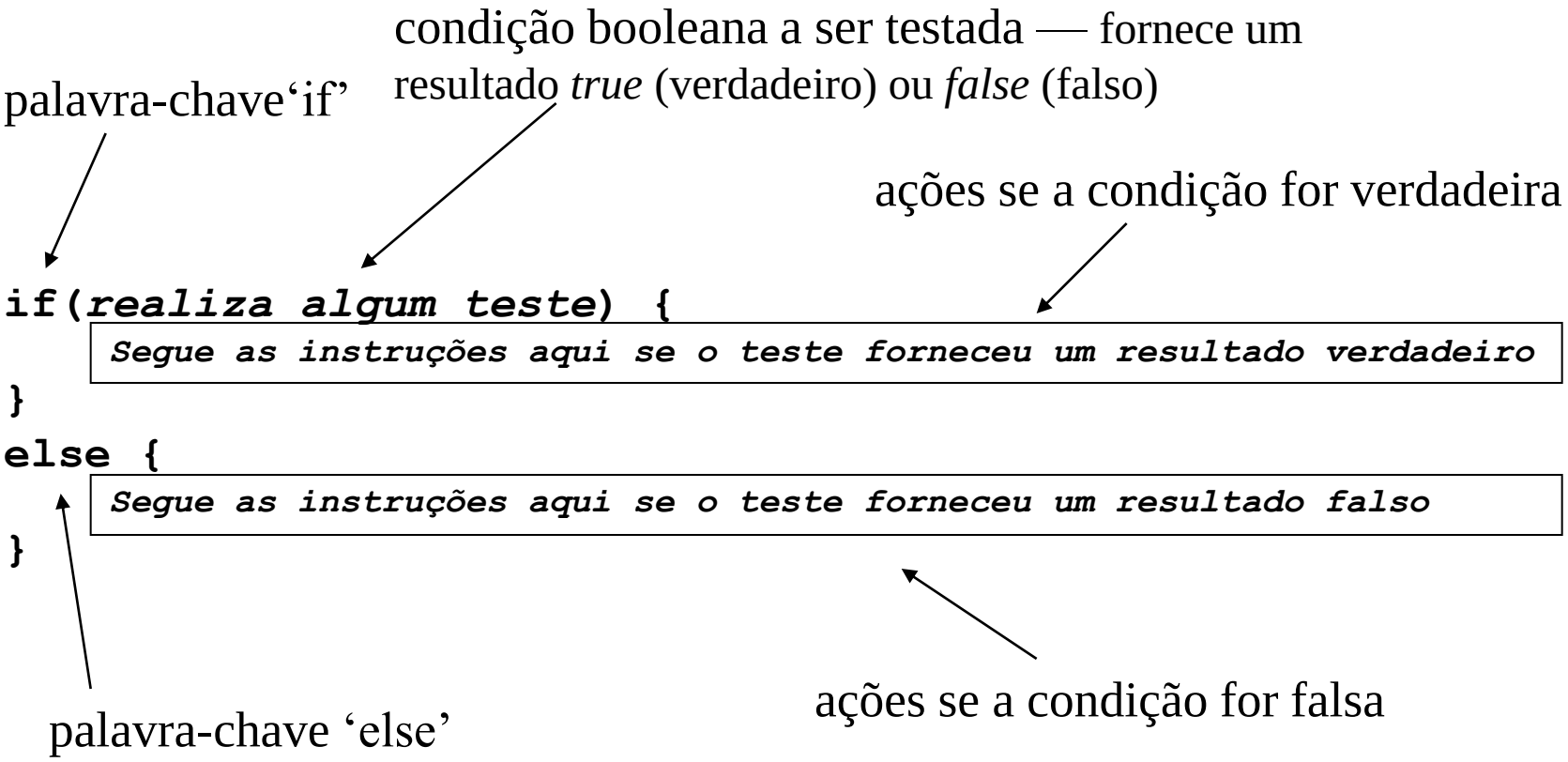
Fazendo escolhas

- Uma **instrução condicional** assume uma das possíveis ações com base no resultado do teste
- Também conhecida como instruções *if*, já que é a palavra mais utilizada nas linguagens de programação
- As expressões booleanas possuem apenas dos valores possíveis true e false. Elas são geralmente encontradas controlando a escolha entre dois caminhos por meio de uma instrução condicional

Fazendo escolhas

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

Fazendo escolhas



INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA RIO GRANDE DO NORTE

Exercício

- Vamos verificar o comportamento de fazer escolhas criando uma nova *TicketMachine* e chamando o *insertMoney*. O saldo muda quando uma mensagem de erro é impressa?

- Tente prever o que acontecerá caso seja inserido o valor zero como parâmetro. Você está certo?

Exercício

- O que acontecerá o teste em *insertMoney* for alterado para if(amount>=0). Faça alguns testes e verifique qual a diferença de comportamento??

Variáveis locais (Escopo)

- Campos são um tipo de variável
  - armazenam valores por toda a vida de um objeto
  - são acessíveis por meio da classe

- Métodos podem incluir variáveis de vida mais curta
  - existem apenas enquanto o método está em execução
  - são acessíveis de dentro do método

Variáveis locais

Uma variável local

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

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
RIO GRANDE DO NORTE

Exercício

- Por que a nova versão de *refundBalance* não fornece os mesmo resultados que a origina?

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

- O que acontece se você tentar compilar a *TicketMachine* com o seguinte código?

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

Revisando o aprendizado

- 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

Revisando o aprendizado

- 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

Revisando o aprendizado

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

Um exemplo familiar

- Vamos trabalhar com um contexto diferente e familiar
  - Abri a *labClass* da aula anterior

- Os campos *name*, *id* e *credits* são inicializados pelo construtor

- Somente *name* e *credits* tem métodos acessadores (*get*), ou seja, id permanece sempre o mesmo

Um exemplo familiar

```
public String getLoginName()
    {
        return name.substring(0,4) + id.substring(0,3);
    }
```

- O *getLoginName* usa do método *substring* da classe *String* sendo o primeiro parâmetro o inicio e o segundo final da parte da *String* que vai ser retornada pelo método.

- Nesse caso se o *name* é “Ricardo” e *id* é “93563” então a String retornada pelo método é Rica935

Exercício

- Crie um novo objeto *Student* e inspecione seus campos.
- O que vai ser retornado por *getLoginName* para um aluno de nome “Bruno Silva” de id “735488”?
- O que acontece ao chamar *getLoginName* para um aluno de nome “aet” de id “832048”? Por que isso ocorre?

Exercício

```
/**
 * Retorna o número de caracteres de uma string.
 */
public int length()
```

- A classe *String* tem o método *length*

- Codificar uma mensagem de erro no construtor de *Student* caso comprimento de *fullName* tiver menos de 4 caracteres ou studentID tiver menos de 3. Mas mesmo assim o construtor ainda deve utilizar esses parâmetros para configurar os campos *name* e *id.* **Dica**: use instrução if sem o else.

Dúvidas

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
RIO GRANDE DO NORTE

Referências

David J. Barnes & Michael Kölling

Programação orientada a objetos com Java

Pearson Education do Brasil, 2004
ISBN 85-7605-012-9.