

Condições e Decisões

**Material didáctico elaborado pelas diferentes equipas de
Introdução à Programação**

Luís Caires (Responsável), Armanda Rodrigues, António Ravara, Carla Ferreira, Fernanda Barbosa, Fernando Birra, Jácome Cunha, João Araújo, Miguel Goulão, Miguel Pessoa Monteiro, e Sofia Cavaco.

Mestrado Integrado em Engenharia Informática FCT UNL

Programas com Decisões

- Neste capítulo, vamos estudar como definir em Java programas que tomam decisões e executam acções em alternativa como resultado de verificar condições.



- No caminho, serão introduzidas as instruções de composição alternativa:
 - A instrução de alternativa binária **if-then-else**
 - A instrução de alternativa binária **if-then**
 - O bloco de instruções (ou instrução composta)
 - A instrução de composição alternativa **switch**

Recorde a Conta Bancária

- **Objectivo**

- Simular uma conta bancária.

- **Descrição**

- Uma conta bancária é um “depósito” de dinheiro (valor inteiro em cêntimos). A quantidade de dinheiro na conta chama-se “saldo”. O saldo pode ser positivo (credor ou nulo) ou negativo (devedor), e é sempre um valor inteiro em cêntimos.

- **Funcionalidades**

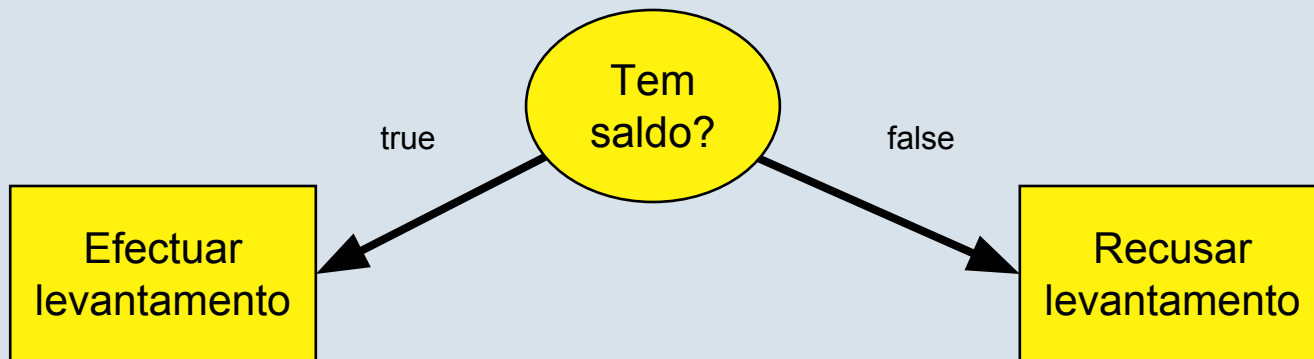
- Numa conta pode-se depositar e levantar dinheiro.
- Deve ser sempre possível consultar o saldo da conta, e verificar se a conta tem um saldo devedor.
- Se não indicarmos nada, a conta é criada com saldo zero. Em alternativa, podemos indicar um valor inicial para o saldo.

- **Interacção com o utilizador**

- Após criar uma conta bancária, pode invocar as operações da conta.

Cenário mais realista

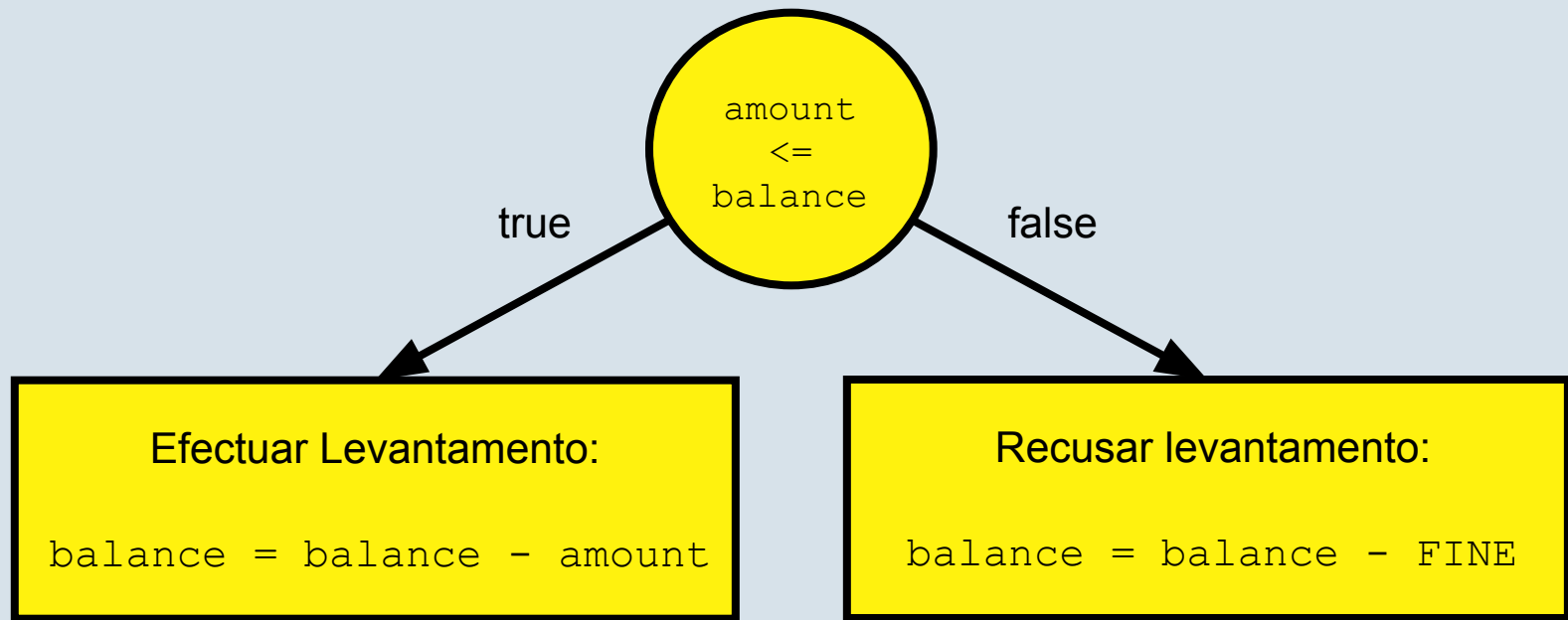
- Sempre que um cliente pretende levantar dinheiro de uma conta bancária, o banco pode tomar uma de duas decisões alternativas
 - Aceitar o levantamento
 - Porque a conta **tem** saldo suficiente
 - Recusar o levantamento
 - Porque a conta **não tem** saldo suficiente



Procedimento Bancário

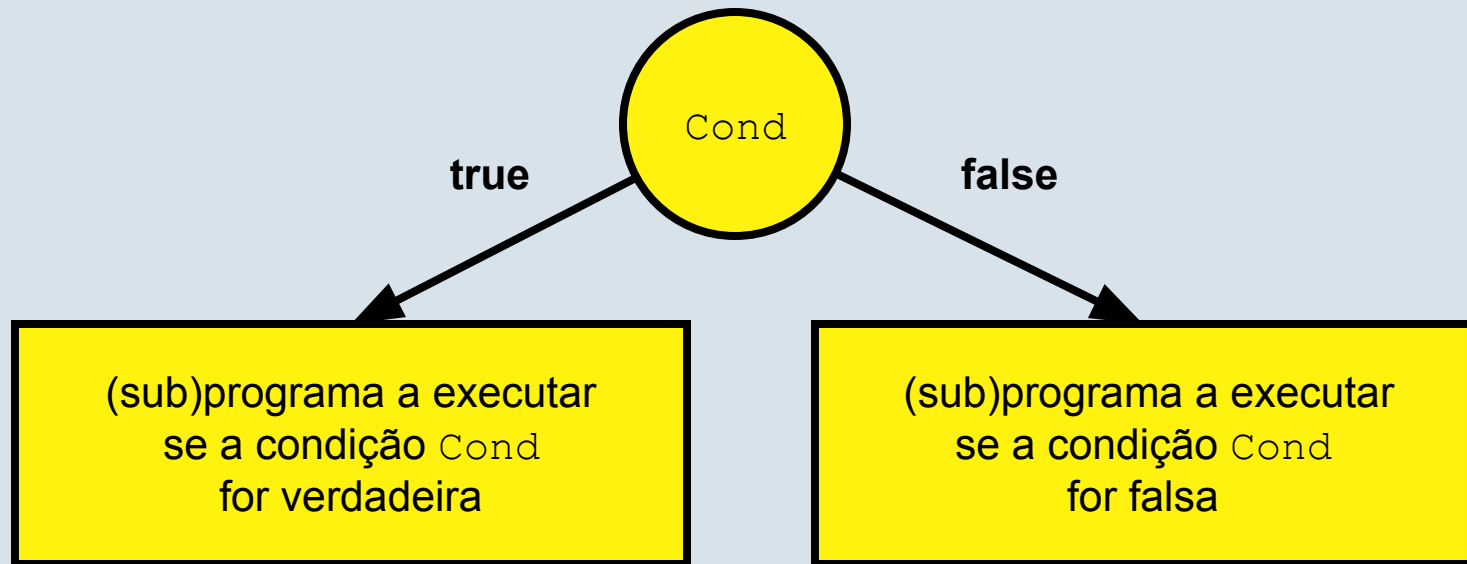
- Vamos considerar que o banco segue as seguintes regras de negócio:
 - O levantamento só é efectuado caso o valor a levantar seja inferior ou igual ao valor do saldo
 - Qualquer tentativa de levantamento de um valor sem saldo suficiente na conta leva à aplicação de uma multa
- O aspecto mais importante a reter aqui é que processar um levantamento requer uma **decisão** entre duas acções **alternativas e exclusivas**.

Procedimento Bancário



- O levantamento pedido (`amount`) só é efectuado caso o valor a levantar seja inferior ou igual ao valor do saldo
- Qualquer tentativa de levantamento de um valor sem saldo suficiente na conta leva à aplicação de uma multa (`FINE`)

Decisões em Programação



- Uma decisão envolve a avaliação de uma expressão booleana `Cond`, uma condição que produz `true` ou `false`
- Qualquer decisão envolve determinar o que deve o programa fazer em qualquer uma das situações.
- Para o programa poder decidir em qualquer circunstância, o programador tem que prever as duas possibilidades

A instrução **if-then-else**

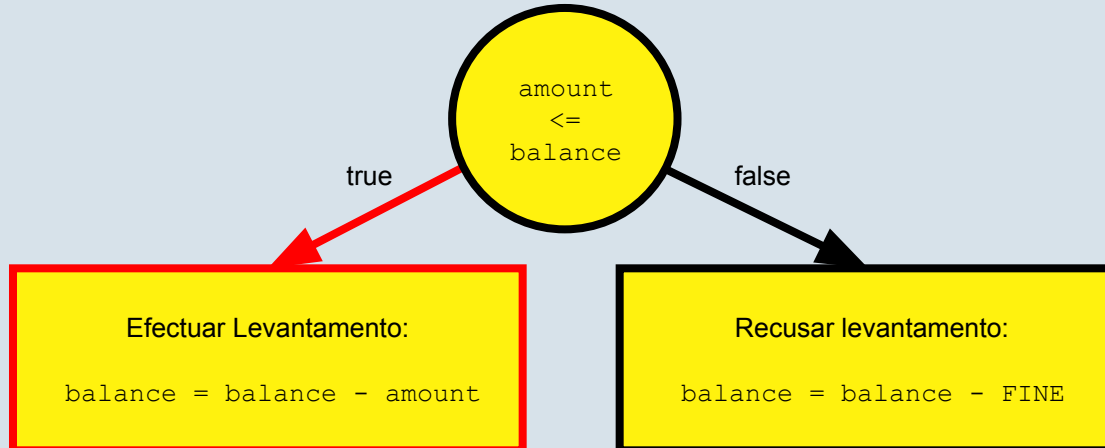
- Na linguagem Java, as decisões exprimem-se usando a instrução **if-then-else**

```
if ( condition )  
    parteTHEN;  
else  
    parteELSE;
```

- A expressão *condition* tem que ser uma expressão booleana: uma condição que produz **true** ou **false**
- Como é executada a instrução **if-then-else** ?
 - Primeiro, a condição *condition* é avaliada e produz um valor.
 - Se o valor é **true**, então o programa executa apenas a *parteTHEN*
 - Se o valor é **false**, então o programa executa apenas a *parteELSE*

A instrução `if-then-else`

```
if (amount <= balance)
    balance = balance - amount;
else
    balance = balance - FINE;
```



```
if (condicao)
    parteTHEN;
else
    parteELSE;
```

A instrução `if-then-else`

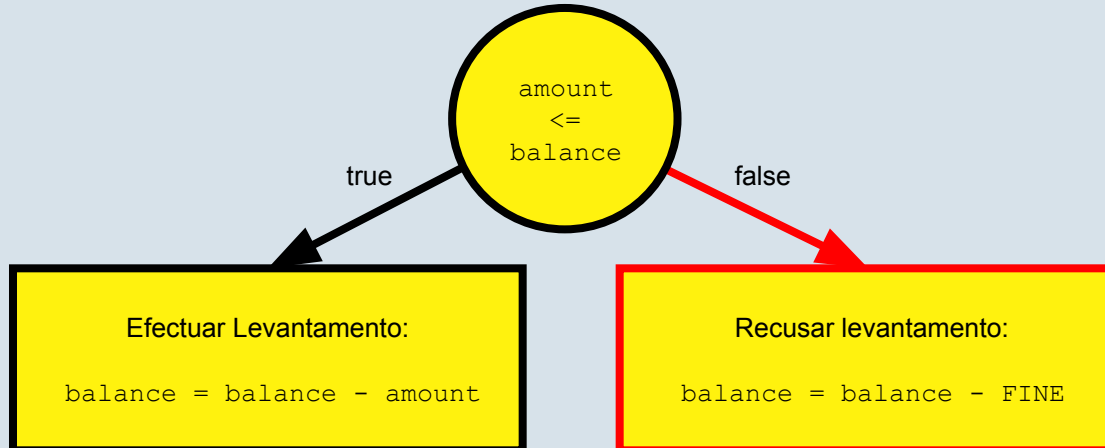
```
if (amount <= balance)
    balance = balance - amount;
else
    balance = balance - FINE;
```

Se *condicao* for verdadeira,
a *parteTHEN* é executada,
caso contrário,
é executada a *parteELSE*.

```
if (condicao)
    parteTHEN;
else
    parteELSE;
```

A instrução `if-then-else`

```
if (amount <= balance)
    balance = balance - amount;
else
    balance = balance - FINE;
```



```
if (condicao)
    parteTHEN;
else
    parteELSE;
```

A instrução `if-then-else`

```
if (amount <= balance)
    balance = balance - amount;
else
    balance = balance - FINE;
```

Se *condicao* for verdadeira,
a *parteTHEN* é executada,
caso contrário,
é executada a *parteELSE*.

```
if (condicao)
    parteTHEN;
else
    parteELSE;
```

A instrução `if-then-else`

- Com alguma frequência, encontramos situações em que pretendemos fazer alguma coisa se uma determinada condição for verdadeira, ou **não fazer nada**, se a condição for falsa
 - Exemplo:
 - **Se** tiver sede **então** beba água
- Nos casos em que a condição testada é falsa e o que pretendemos é não fazer nada, o Java permite omitir a parte do **else**, no `if-then-else`
- A variante do `if-then-else` em que a parte do **else** não é usada é conhecida como a instrução `if-then`

A instrução if-then

- Suponha que o banco não deixa levantar dinheiro, mas não cobra a tal multa, quando se tenta levantar mais dinheiro do que a conta tem:

```
if (amount <= balance )  
    balance = balance - amount;
```

```
if (amount <= balance )  
    balance = balance - amount;  
else // desnecessário  
    ; // Instrução vazia!
```

```
if (condicao)  
    parteTHEN;
```

Se *condicao* for verdadeira,
a *parteTHEN* é executada

Se *condicao* for verdadeira,
a *parteTHEN* é executada
caso contrário
não faz nada

Conta Bancária Segura

Conta Bancária Segura

- **Objectivo**

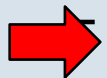
- Simular uma conta bancária segura.

- **Descrição**

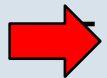
- Uma conta bancária é um “depósito” de dinheiro (valor inteiro em cêntimos). O saldo pode ser positivo (credor ou nulo) ou negativo (devedor), e é sempre um valor inteiro em cêntimos.

- **Funcionalidades**

- Numa conta pode-se depositar e levantar dinheiro. Deve ser sempre possível consultar o saldo da conta e verificar se a conta tem um saldo devedor.

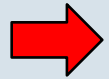


O levantamento é seguro, pois só pode ser efectuado caso o valor a levantar seja inferior ou igual ao valor do saldo. Qualquer tentativa de levantamento de um valor sem provisão na conta leva à aplicação de uma multa de 2 Euros.



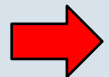
Usualmente, o banco credita um certo juro nas contas dos seus clientes, numa base periódica (por exemplo, uma vez por ano). O valor do juro é calculado por aplicação de uma taxa ao valor do saldo, ou seja, a taxa de juro é determinada com base no valor do saldo, através de um sistema de escalões (quanto maior o saldo, maior a taxa).

Conta Bancária Segura



Deve ser sempre possível calcular o valor de juro anual a aplicar ao saldo da conta. Existem três taxas possíveis:

Valor do Saldo	Taxa
$\leq 2000 \text{ €}$	1%
$]2000 \text{ €, } 10.000 \text{ €}]$	2%
$> 10.000 \text{ €}$	3%



Deve ser sempre possível creditar os juros no saldo da conta.

- Se não indicarmos nada, a conta é criada com saldo zero. Em alternativa, podemos indicar um valor inicial para o saldo.

- **Interacção com o utilizador**

- Após criar uma conta bancária segura, pode invocar as operações da conta.

Conta Bancária Segura

- Interface de SafeBankAccount:

void deposit(**int** amount)

Depositar a importância amount na conta

Pre: amount > 0

void withdraw(**int** amount)

Levantar a importância amount na conta ou aplicar a multa (**FINE**)

Pre: amount > 0

int getBalance()

Consultar o saldo da conta

boolean isInRedZone()

Indica se a conta está devedora

int computeInterest()

Calcular qual o valor do juro anual a aplicar

void applyInterest()

Creditar o juro anual ao saldo da conta

Cenário

Classe SafeBankAccount

```
SafeBankAccount bal = new SafeBankAccount(1000);  
bal.getBalance();  
//1000    (int)  
bal.withdraw(1500);  
bal.getBalance();  
//800     (int)  
bal.computeInterest();  
//8       (int)  
bal.applyInterest();  
bal.getBalance();  
//808     (int)
```

Conta Bancária Segura

- Defina em Java uma classe `SafeBankAccount` cujos objectos implementam a interface indicada.
- Programe a sua classe no Eclipse.
- Teste um (ou vários) objectos `SafeBankAccount`, e verifique se se comportam como esperado.



Conta Bancária Segura

- Interface de SafeBankAccount:

void deposit(**int** amount)

Depositar a importância amount na conta

Pre: amount > 0

void withdraw(**int** amount)

Levantar a importância amount na conta ou aplicar a multa (**FINE**)

Pre: amount > 0

int getBalance()

Consultar o saldo da conta

boolean isInRedZone()

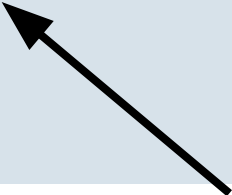
Indica se a conta está devedora

int computeInterest()

Calcular qual o valor do juro anual a aplicar

void applyInterest()

Creditar o juro anual ao saldo da conta

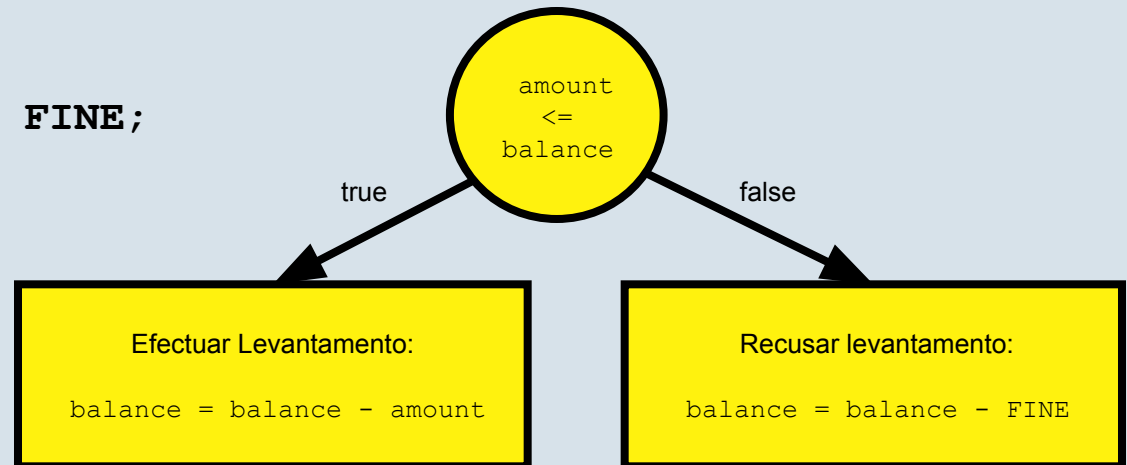


Como programar o método withdraw ?

Método `withdraw(int amount)`

- A operação pode ser programada usando a instrução **if-then-else**, do seguinte modo:

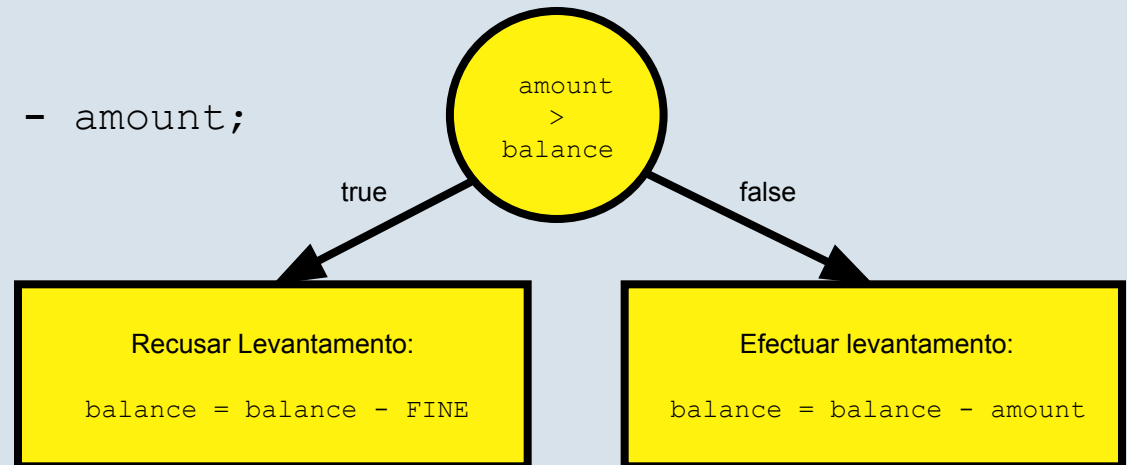
```
public void withdraw(int amount) {  
    if (amount <= balance)  
        balance = balance - amount;  
    else  
        balance = balance - FINE;  
}
```



Método `withdraw(int amount)`

- Outro modo equivalente de definir a decisão, com base na condição oposta:

```
public void withdraw(int amount) {  
    if (amount > balance)  
        balance = balance - FINE;  
    else  
        balance = balance - amount;  
}
```



Método `withdraw(int amount)`

- O valor `FINE` da multa é **constante**, não devendo nunca ser alterado durante a execução do programa
- O valor `FINE` é **comum** a todas as contas geradas pela classe `SafeBankAccount`
- Para facilitar eventuais re-programações do valor da multa, é conveniente declarar o mesmo como uma constante global

```
public class SafeBankAccount {  
    public static final int FINE = 200;  
    ...  
}
```

Conta Bancária Segura

- Interface de SafeBankAccount:

void deposit(**int** amount)

Depositar a importância amount na conta

Pre: amount > 0

void withdraw(**int** amount)

Levantar a importância amount na conta ou aplicar a multa (**FINE**)

Pre: amount > 0

int getBalance()

Consultar o saldo da conta

boolean isInRedZone()

Indica se a conta está devedora

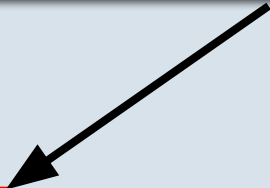
int computeInterest()

Calcular qual o valor do juro anual a aplicar

void applyInterest()

Creditar o juro anual ao saldo da conta

Como programar o método computeInterest ?



Método `computeInterest()`

- A operação `computeInterest()` pode ser decomposta em duas tarefas mais elementares que devem ser realizadas **sequencialmente**:
 - Determinar a taxa de juro a partir do saldo corrente
 - Aplicar a taxa ao saldo corrente

```
public int computeInterest() {
```

```
    Determinar qual a taxa de juro
```

```
    Calcular o valor do juro com  
    base no valor do saldo
```

```
}
```

Método `computeInterest()`

- A operação `computeInterest()` pode ser decomposta em duas tarefas mais elementares que devem ser realizadas **sequencialmente**:
 - Determinar a taxa de juro a partir do saldo corrente
 - Aplicar a taxa ao saldo corrente

```
public int computeInterest() {
```

```
    interestRate = ?
```

```
        Calcular o valor do juro com  
        base no valor saldo
```

```
}
```

Determinação da taxa de juro

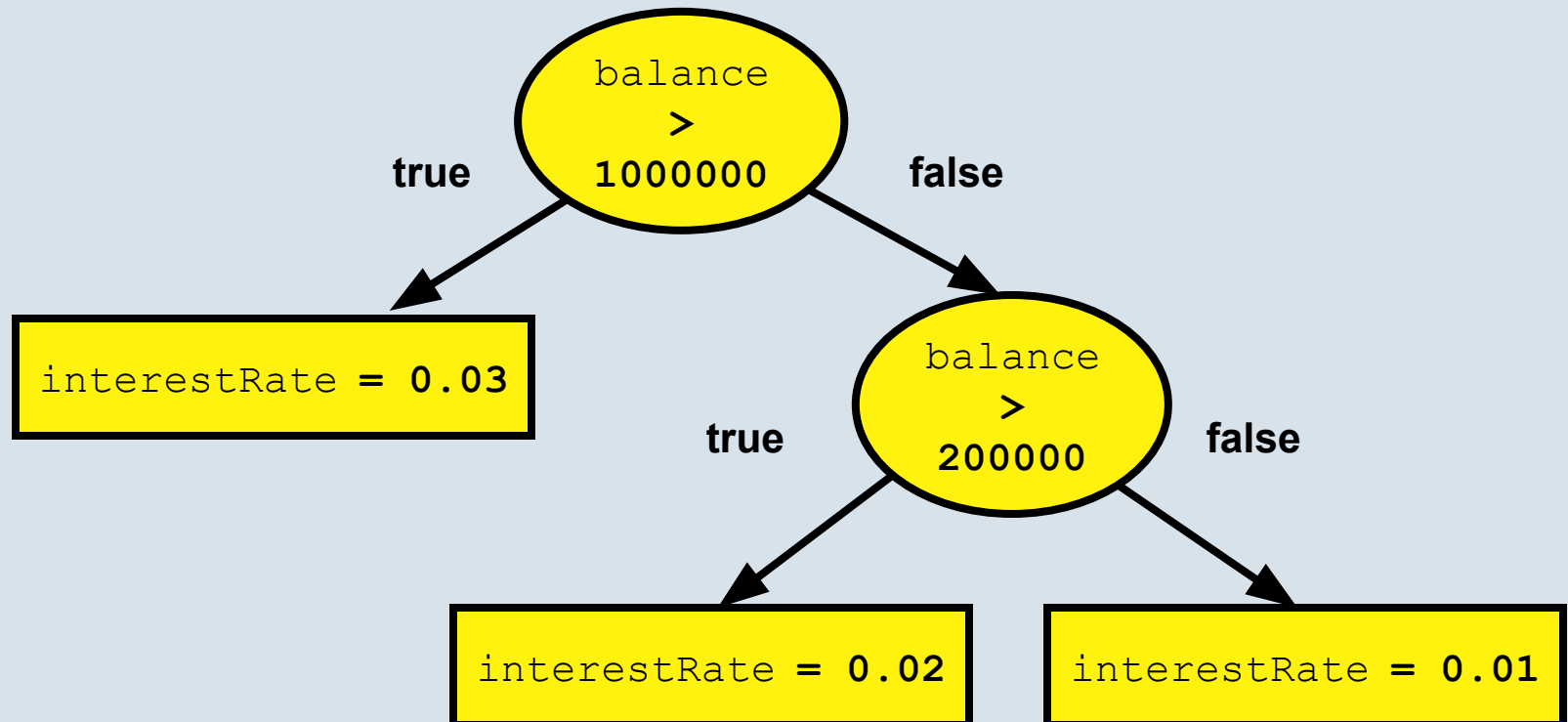
A taxa de juro a aplicar depende do valor saldo existente na conta.

Existem três taxas possíveis:

Valor do Saldo	Taxa
$\leq 2000 \text{ €}$	1%
$]2000 \text{ €, } 10.000 \text{ €}]$	2%
$> 10.000 \text{ €}$	3%

Determinação da taxa de juro

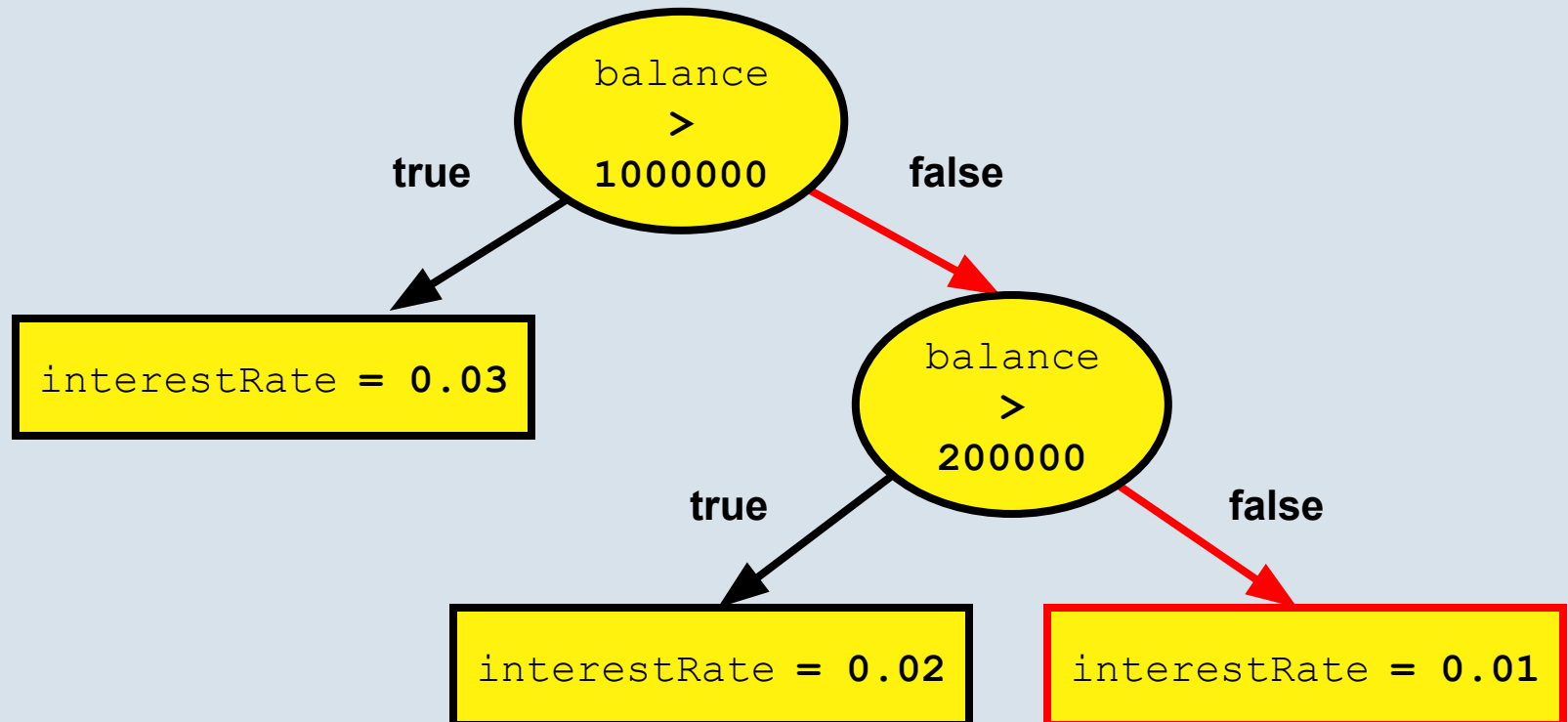
- A determinação da taxa de juro pode ser efectuada através das decisões seguintes:



Determinação da taxa de juro

- Exemplo:

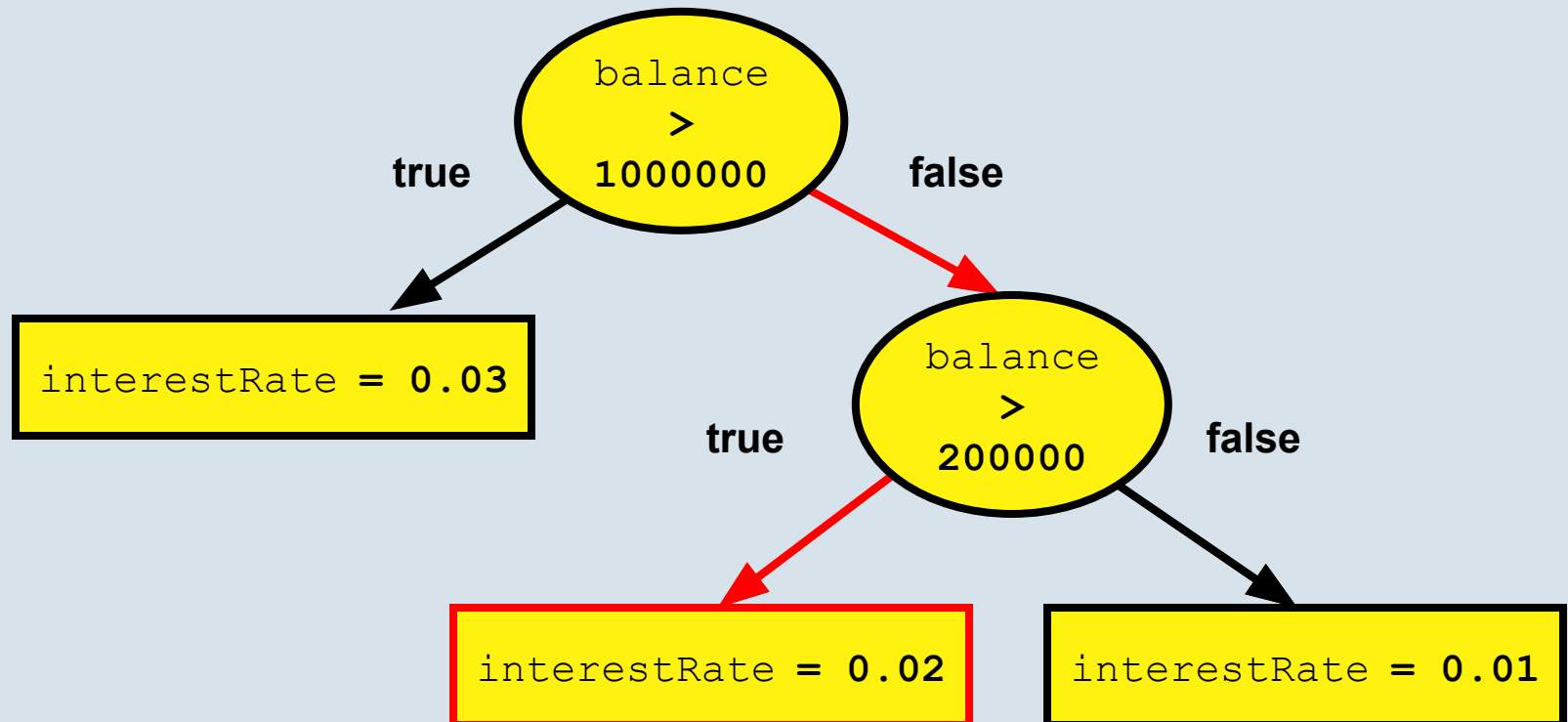
- balance = 6700



Determinação da taxa de juro

- Exemplo:

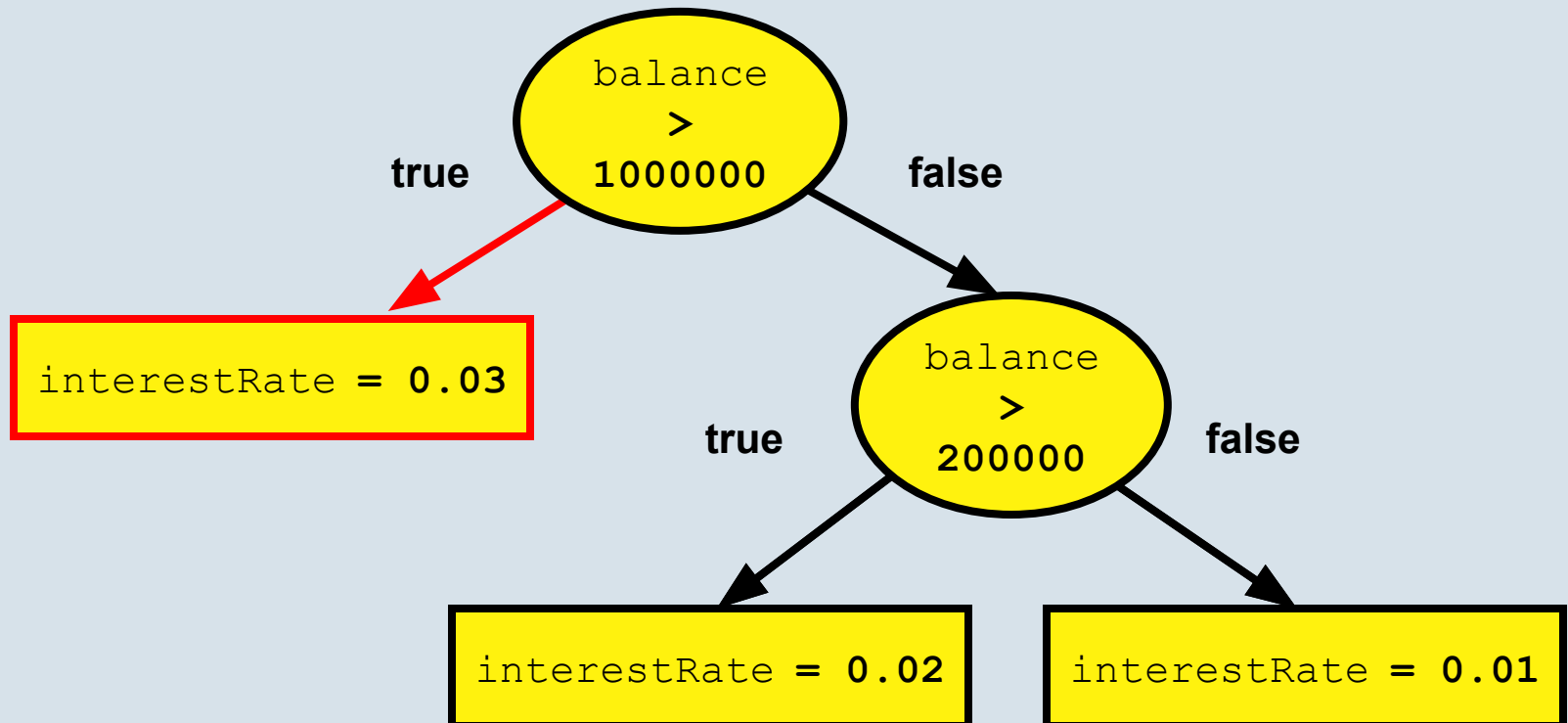
- balance = 250000



Determinação da taxa de juro

- Exemplo:

- balance = 10000000



Método `computeInterest()`

- Determinar a taxa de juro a partir do saldo corrente

```
public int computeInterest() {
```

```
    float interestRate ;
```

Declaração de
variável local



```
    interestRate = ?
```

O valor de
`interestRate`
tem que ser
determinado a partir
do valor do saldo



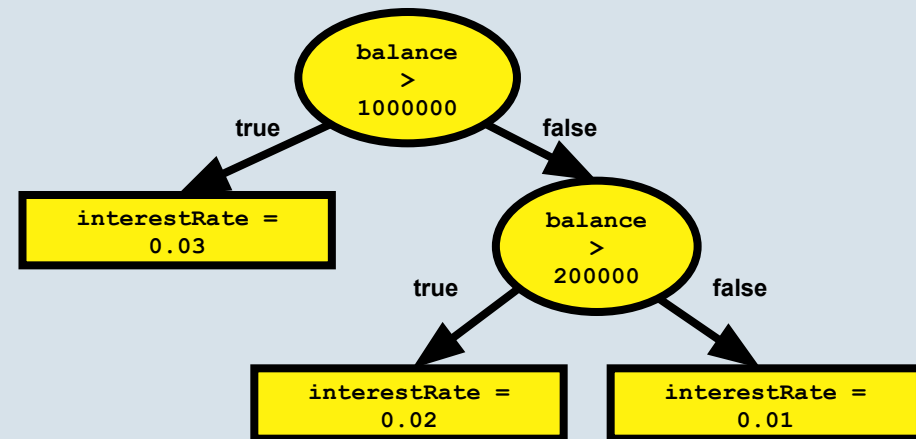
Calcular o valor do juro com
base no saldo

```
}
```

Método computeInterest()

- Determinar a taxa de juro a partir do saldo corrente

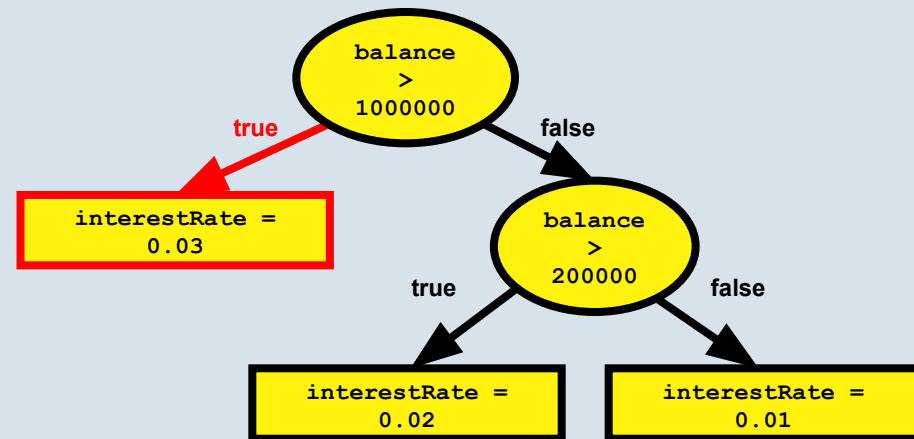
```
public int computeInterest() {  
    float interestRate ;  
    if (balance > 1000000)  
        interestRate = 0.03f;  
    else if (balance > 200000)  
        interestRate = 0.02f;  
    else interestRate = 0.01f;  
  
    Calcular o valor do juro  
    com base no saldo  
}
```



Método computeInterest()

- Determinar a taxa de juro a partir do saldo corrente

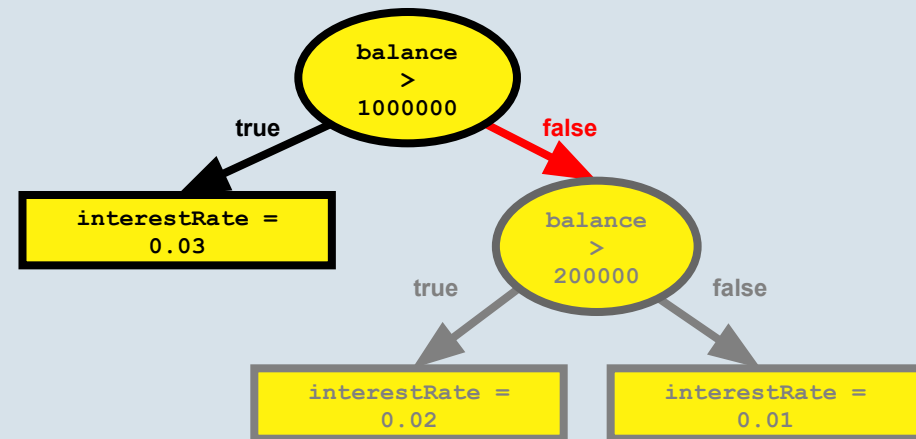
```
public int computeInterest() {  
    float interestRate ;  
    if (balance > 1000000)  
        interestRate = 0.03f;  
    else if (balance > 200000)  
        interestRate = 0.02f;  
    else interestRate = 0.01f;  
  
    Calcular o valor do juro  
    com base no saldo  
}
```



Método computeInterest()

- Determinar a taxa de juro a partir do saldo corrente

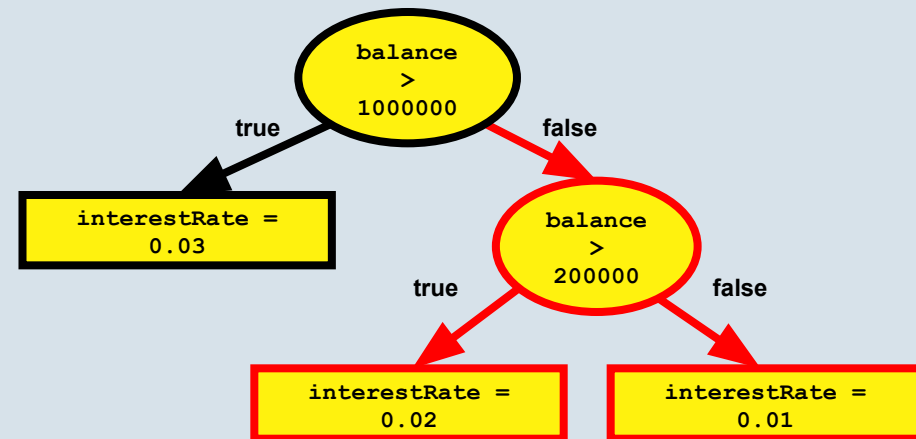
```
public int computeInterest() {  
    float interestRate ;  
    if (balance > 1000000)  
        interestRate = 0.03f;  
    else if (balance > 200000)  
        interestRate = 0.02f;  
    else interestRate = 0.01f;  
  
    Calcular o valor do juro  
    com base no saldo  
}
```



Método computeInterest()

- Determinar a taxa de juro a partir do saldo corrente

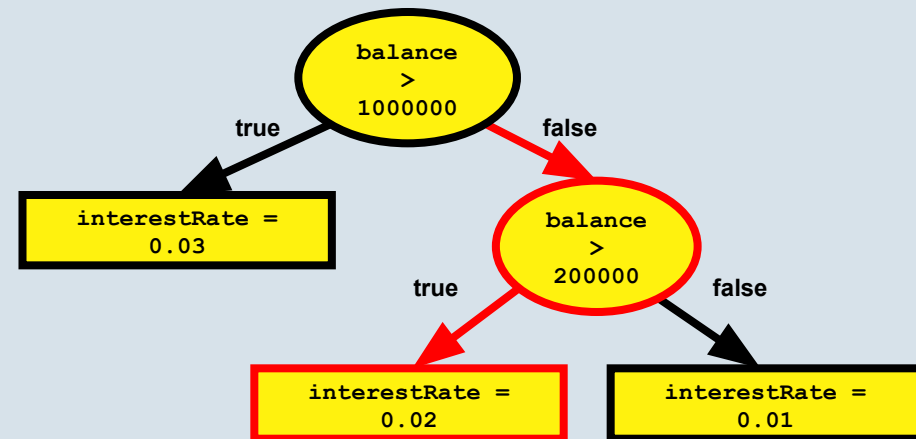
```
public int computeInterest() {  
    float interestRate ;  
    if (balance > 1000000)  
        interestRate = 0.03f;  
    else if (balance > 200000)  
        interestRate = 0.02f;  
    else interestRate = 0.01f;  
  
    Calcular o valor do juro  
    com base no saldo  
}
```



Método computeInterest()

- Determinar a taxa de juro a partir do saldo corrente

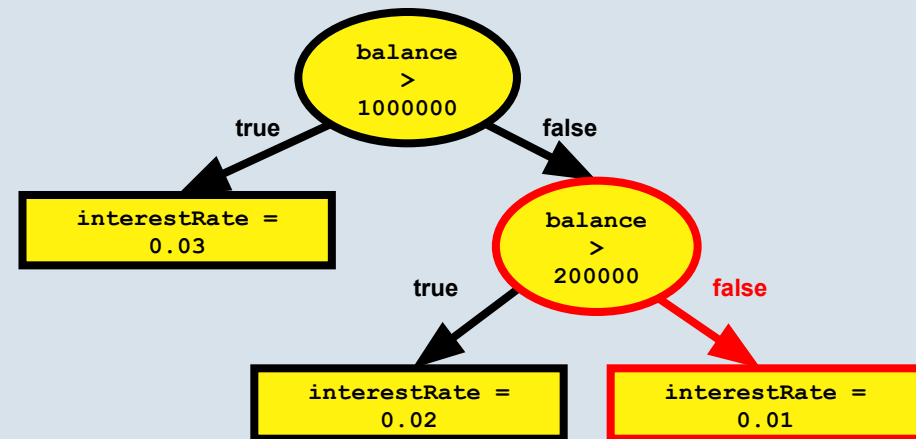
```
public int computeInterest() {  
    float interestRate ;  
    if (balance > 1000000)  
        interestRate = 0.03f;  
    else if (balance > 200000)  
        interestRate = 0.02f;  
    else interestRate = 0.01f;  
  
    Calcular o valor do juro  
    com base no saldo  
}
```



Método computeInterest()

- Determinar a taxa de juro a partir do saldo corrente

```
public int computeInterest() {  
    float interestRate ;  
    if (balance > 1000000)  
        interestRate = 0.03f;  
    else if (balance > 200000)  
        interestRate = 0.02f;  
    else interestRate = 0.01f;  
  
    Calcular o valor do juro  
    com base no saldo  
}
```

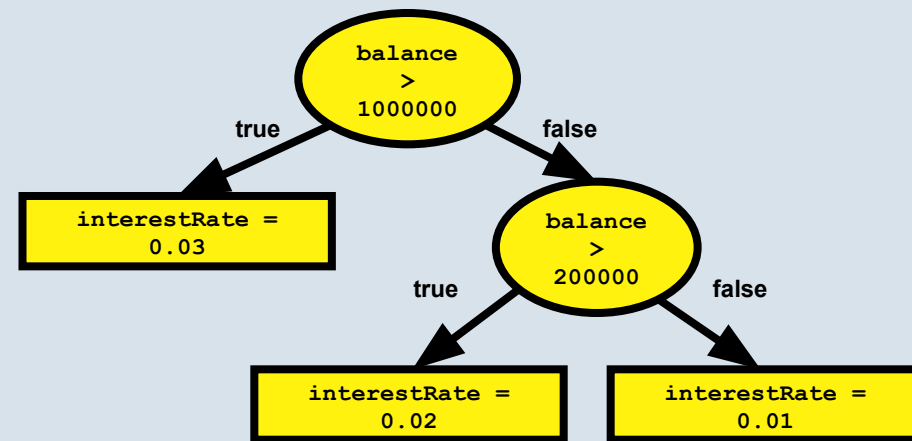


Método computeInterest()

- Determinar a taxa de juro a partir do saldo corrente

```
public int computeInterest() {  
    float interestRate ;  
    if (balance > 1000000)  
        interestRate = 0.03f;  
    else if (balance > 200000)  
        interestRate = 0.02f;  
    else interestRate = 0.01f;  
  
    Calcular o valor do juro  
    com base no saldo  
}
```

Valor do Saldo	Taxa
≤ 2000 €	1%
]2000 €,10.000 €]	2%
> 10.000 €	3%



Método computeInterest()

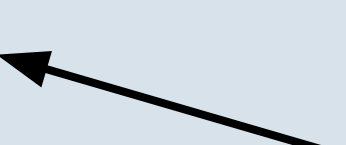
- Determinar a taxa de juro a partir do saldo corrente

```
public int computeInterest() {  
    float interestRate ;  
    if (balance > 1000000)  
        interestRate = 0.03f;  
    else if (balance > 200000)  
        interestRate = 0.02f;  
    else interestRate = 0.01f;  
    return balance * interestRate;  
}
```

Determinar qual a taxa a partir do valor do saldo



Calcular o valor do juro com base no saldo e na taxa

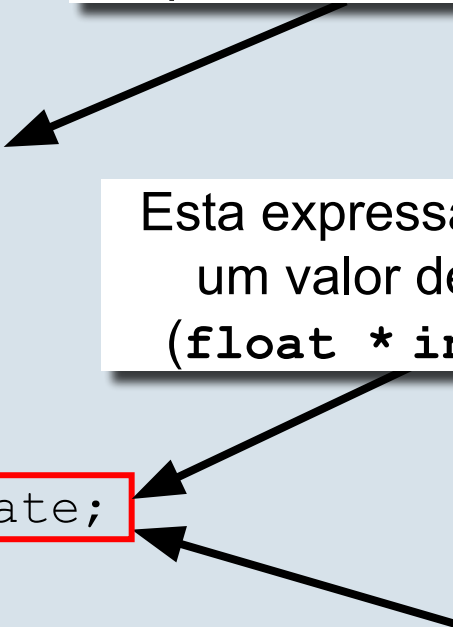


Método computeInterest()

- Determinar a taxa de juro a partir do saldo corrente

```
public int computeInterest() {  
    float interestRate ;  
    if (balance > 1000000)  
        interestRate = 0.03f;  
    else if (balance > 200000)  
        interestRate = 0.02f;  
    else interestRate = 0.01f;  
    return balance * interestRate;  
}
```

Determinar qual a taxa a partir do valor do saldo



Esta expressão vai produzir um valor de tipo float
(float * int ⇒ float)

Calcular o valor do juro com base no saldo e na taxa

Método `computeInterest()`

- Determinar a taxa de juro a partir do saldo corrente

```
public int computeInterest() {  
    float interestRate ;  
    if (balance > 1000000)  
        interestRate = 0.03f;  
    else if (balance > 200000)  
        interestRate = 0.02f;  
    else interestRate = 0.01f;  
    return Math.round(balance * interestRate);  
}
```

Determinar qual a taxa a partir do valor do saldo



Como estamos a representar os valores em cêntimos (tipo `int`) é necessário arredondar e converter o resultado para o tipo `int`



Método computeInterest()

- Método computeInterest()

```
public int computeInterest() {  
    float interestRate ;  
    if (balance > 1000000)  
        interestRate = 0.03f;  
    else if (balance > 200000)  
        interestRate = 0.02f;  
    else interestRate = 0.01f;  
    return Math.round(balance * interestRate);  
}
```

- Esta solução funciona, mas devemos fazer melhor!

Método computeInterest()

- Método computeInterest()

```
public int computeInterest() {  
    float interestRate ;  
    if (balance > 1000000)  
        interestRate = 0.03f;  
    else if (balance > 200000)  
        interestRate = 0.02f;  
    else interestRate = 0.01f;  
    return Math.round(balance * interestRate);  
}
```

Deve-se evitar utilizar constantes “mágicas” nos programas, pois tal torna os programas difíceis de ler e de modificar.

- Técnica correcta: dar nomes às constantes na classe.

Método computeInterest()

- Método computeInterest()

```
public int computeInterest() {  
    float interestRate ;  
    if (balance > CAT1)  
        interestRate = RATECAT1;  
    else if (balance > CAT2)  
        interestRate = RATECAT2;  
    else interestRate = RATECAT3;  
    return Math.round(balance * interestRate);  
}
```

Deve-se evitar utilizar constantes “mágicas” nos programas, pois tal torna os programas difíceis de ler e de modificar.

- Técnica correcta: dar nomes às constantes na classe.

Método computeInterest()

- Técnica correcta: dar nomes às constantes na classe

```
public class SafeBankAccount {  
    ...  
    public static final int CAT1 = 1000000;  
    public static final int CAT2 = 200000;  
    public static final float RATECAT1 = 0.03f;  
    public static final float RATECAT2 = 0.02f;  
    public static final float RATECAT3 = 0.01f;  
    ...  
}
```

Conta Bancária Segura

- Interface de SafeBankAccount:

void deposit(**int** amount)

Depositar a importância amount na conta

Pre: amount > 0

void withdraw(**int** amount)

Levantar a importância amount na conta ou aplicar a multa (**FINE**)

Pre: amount > 0

int getBalance()

Consultar o saldo da conta

boolean isInRedZone()

Indica se a conta está devedora

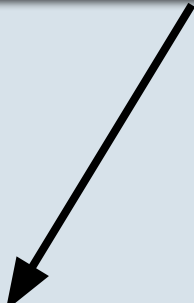
int computeInterest()

Calcular qual o valor do juro anual a aplicar

void applyInterest()

Creditar o juro anual ao saldo da conta

Como programar o método applyInterest ?



Método `applyInterest()`

- O método `applyInterest()` pode chamar o método `computeInterest()` e acumular o resultado no saldo

```
public void applyInterest() {  
    balance = balance + this.computeInterest();  
}
```

- Recorde que, para chamar um método do próprio objecto no corpo de um método, aplicamo-lo ao nome especial **this**
 - O identificador **this** refere o **próprio objecto**, neste caso, o mesmo objecto em que a operação `applyInterest()` está a ser executada

Bloco de Instruções em Java

- Temos chamado genericamente "instruções" aos vários tipos de "frases" que temos escrito em Java.
- Vamos agora detalhar um pouco mais a questão: o que é uma "instrução" Java?
- O termo é sinónimo de "ensino" e de "ordem"
 - Uma linguagem de programação como o Java permite definir sequências de imperativos (informalmente, as instruções permitem indicar à máquina como proceder).

Exemplos de instruções

```
; /* Instrução vazia */
```

```
int x; /* declaração de variável */
```

```
x = 27; /* Atribuição (Afectação) */
```

```
if (x > 0) /* Decisão simples */
```

```
return x; /* Instrução de devolução de valor */
```

Tipos de instruções

- Sistematizando, vimos os seguintes tipos de instruções:
 - associadas à declaração de classes, vêm também: constantes, variáveis, construtores e métodos
 - no corpo dos construtores vimos a atribuição, que surge também no corpo dos métodos
 - nos métodos vimos ainda como devolver valores (return expressão) e como fazer composições (em sequência e em alternativa)
 - na interacção com uma classe (na classe "Main" no método "main") vimos ainda como criar objectos de certa classe e invocar operações sobre esses objectos

Tipos de instruções

- Vamos chamar **instrução** a dois tipos de frases Java: **declarações** e **comandos**.
- Declarações são:
 - de classe
 - de constantes
 - de variáveis
 - de construtores
 - de métodos
- Comandos são:
 - atômicos:
 - vazio
 - return expressão
 - new idClass(lista param), a atribuição
 - Compostos:
 - sequência (liga comandos com ';')
 - alternativa binária
 - alternativa generalizada (veremos adiante)
 - iteração (veremos mais tarde)

Tipos de instrução: declarações

- As declarações de classes, de construtores e de métodos são organizadas em **blocos**.
 - Blocos:
 - definem sequências de instruções, sendo delimitados por chavetas
 - definem o âmbito (ou escopo) das declarações que contêm
 - podem ser "aninhados" (blocos podem conter blocos)
- A forma geral de um bloco é:
- cabeçalho {sequência de instruções;}

Declarações

- Tipicamente, temos as seguintes declarações
 - classe:
cabeçalho {
 sequência de declarações de constantes;
 sequência de declarações de variáveis;
 sequência de declarações de construtores;
 sequência de declarações de métodos;
}
 - construtores e métodos:
cabeçalho {
 comando
}

Comandos

- Nos comandos compostos de alternativa e de iteração, o corpo de cada ramo do comando tem também que ser marcado com chavetas, quando é uma sequência.
- Exemplos:
 - `if (expr) {c1;c2;} else {c3;c4;}`
 - `if (expr1) if (expr) {c1;c2;} else {c3;c4;}`
 - `if (expr1) {if (expr) {c1;c2;}} else {c3;c4;}`
 - `switch (var) {lista de casos}`

Instrução switch

A instrução de composição alternativa generalizada

```
switch (expression) {  
    case literal-1:  
        statements-1;  
        break;  
    case literal-2:  
        statements-2;  
        break;  
    // (more cases) ...  
    case literal-N:  
        statements-N;  
        break;  
    default: // optional default case  
        statements- (N+1) ;  
        break;  
} // end of switch statement
```

expression tem de ser do tipo
int, char, ou String

Instrução switch

A instrução de composição alternativa generalizada

```
switch (expression) {  
    case literal-1:  
        statements-1;  
        break;  
    case literal-2:  
        statements-2;  
        break;  
    // (more cases) ...  
    case literal-N:  
        statements-N;  
        break;  
    default: // optional default case  
        statements-(N+1);  
        break;  
} // end of switch statement
```

Cada **literal** tem um valor do mesmo tipo que **expression**.

O código executado é o que segue o **literal** que corresponde ao valor guardado em **expression**.

Instrução switch

A instrução de composição alternativa generalizada

```
switch (expression) {  
    case literal-1:  
        statements-1;  
        break;  
    case literal-2:  
        statements-2;  
        break;  
    // (more cases) ...  
    case literal-N:  
        statements-N;  
        break;  
    default: // optional default case  
        statements- (N+1) ;  
        break;  
} // end of switch statement
```



A utilização de **break** é opcional e põe termo à execução da instrução **switch**.

Se for omitido, o caso seguinte ao escolhido também será executado (e assim sucessivamente até haver um caso com **break**).

Instrução switch

A instrução de composição alternativa generalizada

```
switch (expression) {  
    case literal-1:  
        statements-1;  
        break;  
    case literal-2:  
        statements-2;  
        break;  
    // (more cases) ...  
    case literal-N:  
        statements-N;  
        break;  
    default: // optional default case  
        statements- (N+1) ;  
        break;  
} // end of switch statement
```

A utilização do **default** também é opcional. Este ramo é executado se nenhum dos casos explícitos corresponder ao valor de **expression**.



Instrução switch

A instrução de composição alternativa generalizada

```
switch (expression) {  
    case literal-1:  
        statements-1;  
        break;  
    case literal-2:  
        statements-2;  
        break;  
    // (more cases) ...  
    case literal-N:  
        statements-N;  
        break;  
    default: // optional default case  
        statements- (N+1) ;  
        break;  
} // end of switch statement
```

O bloco de instruções de um caso pode ser vazio. Isto implica a execução do código do caso seguinte para ambos os casos.

Que expressão testar num `switch`?

- Constante (pouco útil)

- tipo `int`

- Exemplo

```
public void goodExample1(int i) {  
    ...  
    switch (i) {  
        ...  
    } // end of switch statement  
} // end of goodExample1
```

- tipo (de sequência de) caracteres

O tipo char

Tipo Char

- Um **char** é um carácter representando uma letra, um dígito, um sinal de pontuação, uma tabulação (tab) um espaço, ou algo similar.
- Um literal do tipo char é representado como o carácter entre apóstrofos.

Literals: 'a' 'b' 'A' 'Z' ';' '\e', '\#', '\3', '\ ', '\.'

- O tipo char armazena apenas um carácter da tabela ASCII e ocupa exactamente um byte de memória. Na realidade, não é o carácter que é armazenado na memória, mas o código ASCII correspondente.

Tabela ASCII

American Standard Code for Information Interchange

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	 	Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	\$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174	|	
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177		DEL

Source: www.LookupTables.com

Tipo String

- Em Java as sequências de caracteres são representadas como valores do tipo **String**.
- As Strings representam “bocados” de texto, que podem ser manipulados pelos programas através de operações especiais.
- Os valores de tipo **String** escrevem-se como sequências de caracteres arbitrários (menos as “’s) entre “ ”.

Por exemplo:

- “As armas e os barões assinalados”
- “%&Ghg1j9892bnsabdGHJbsu?*”
- “this is a recording, please hang up!”
- “class”

Operações básicas sobre o tipo `String`

- A operação mais básica que se pode efectuar com Strings é a concatenação (justaposição) `+`.
 - `"hello" + " " + "world"` produz `"hello world"`
 - `"X" + "Y"` produz `"XY"`
- É possível concatenar também inteiros (`int`) com strings; nesse caso o inteiro é convertido na sua representação como `String` (por exemplo, o inteiro `12` é convertido na `String` `"12"`).
 - `"Altura=" + height` produz `"Altura=10"`
 - `"Horas " + mi + " minutos"` produz `"Horas 0 minutos"`
 - `"Ping" + (2-1)` produz `"Ping1"`
 - Note que o contrário não é possível. Uma `String` não pode ser guardada numa variável de tipo `int`, mesmo que represente um número (por exemplo `"23"`)

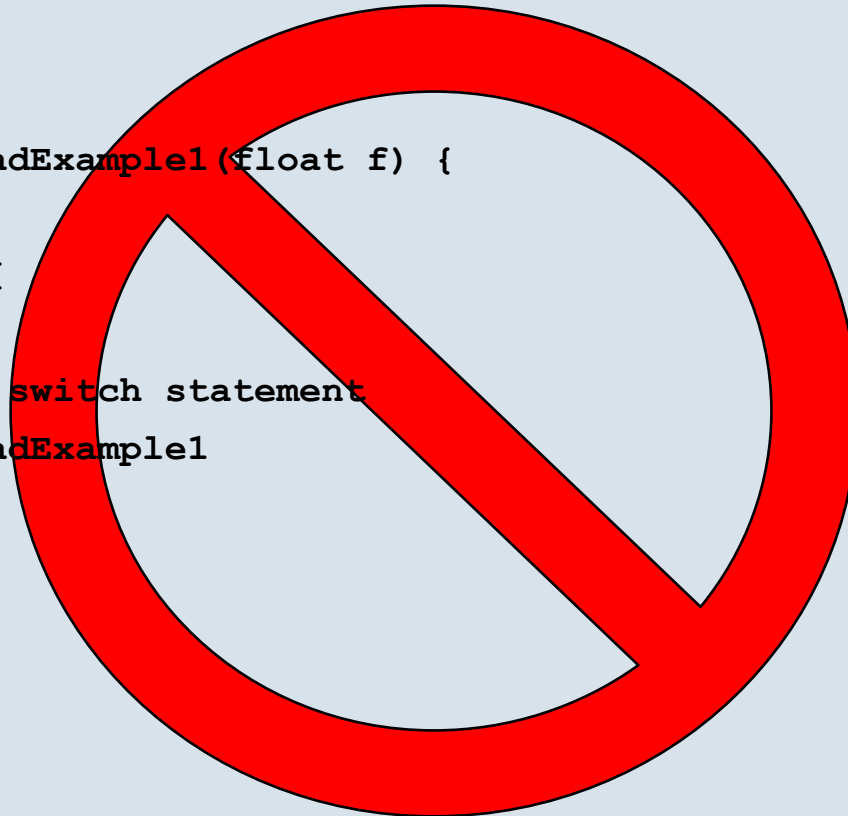
Expressão tem de ser `int`, `char`, ou `String`

```
// pre: month != null
public int getMonthNumber(String month) {
    int monthNumber = 0;
    switch (month) { // Only lowercase month names are recognized!
        case "january":
            monthNumber = 1;
            break;
        case "february":
            monthNumber = 2;
            break;
        //... Several months omitted here...
        case "november":
            monthNumber = 11;
            break;
        case "december":
            monthNumber = 12;
            break;
    } // end of switch statement
    return monthNumber;
} // end of getMonthNumber operation
```

Expressão tem de ser `int`, `char`, ou `String`

- A expressão `expression` em `switch (expression)` tem de ser dos tipos `int`, `char`, ou `String`, e constante
 - Exemplos de erros típicos

```
public void badExample1(float f) {  
    ...  
    switch (f) {  
        ...  
    } // end of switch statement  
} // end of badExample1
```



O compilador dá erro, porque o tipo da expressão não é nem `int` nem `char`, nem `String`!

Os literais de cada `case` são constantes

- Exemplo correcto (o literal `3` é constante):

```
public void goodExample(int i) {  
    ...  
    switch (i) {  
        case 3: ...  
        ...  
    } // end of switch statement  
} // end of goodExample
```

- Exemplo errado (`x` não é constante!):

```
public void badExample(int i) {  
    int x = /* uma expressão variável */;  
    switch (i) {  
        case x: ...  
        ...  
    } // end of switch statement  
} // end of badExample
```



Os literais de cada `case` são constantes

- Exemplo correcto (o literal `'e'` é constante):

```
public void goodExample2(char c) {  
    ...  
    switch (c) {  
        case 'e':...  
  
    } // end of switch statement  
} // end of goodExample2
```

O corpo de um case pode ser vazio

```
public boolean bissexto(int ano){...} //true se for bissexto, false se não for
public int diasDoMes(String mes, int ano){
    int dias = 0;
    switch (mes) { // apenas estamos a reconhecer nomes em minúsculas
        case "janeiro":
        case "março":
        case "maio":
        case "julho":
        case "agosto":
        case "outubro":
        case "dezembro":
            dias = 31;
            break;
        case "abril":
        case "junho":
        case "setembro":
        case "novembro":
            dias = 30;
            break;
        case: "fevereiro":
            if (bissexto(ano)) dias = 29;
            else dias = 28;
            break;
    }
    return dias;
}
```

Programas com Decisões

- Neste capítulo, vimos como definir em Java programas que tomam decisões e executam acções em alternativa como resultado de verificar condições.



- No caminho, foram introduzidas as instruções de composição alternativa:
 - A instrução de alternativa binária **if-then-else**
 - A instrução de alternativa binária **if-then**
 - O bloco de instruções (ou instrução composta)
 - A instrução de composição alternativa **switch**