Condições e Decisões

Mestrado Integrado em Engenharia Informática FCT UNL
http://ctp.di.fct.unl.pt/miei/ip/

Corpo Docente 2020/2021
António Ravara, Artur Miguel Dias, Bernardo Toninho,
Ema Vieira, Inês Fernandes, Margarida Mamede,
Miguel Monteiro, Rui Nóbrega

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.

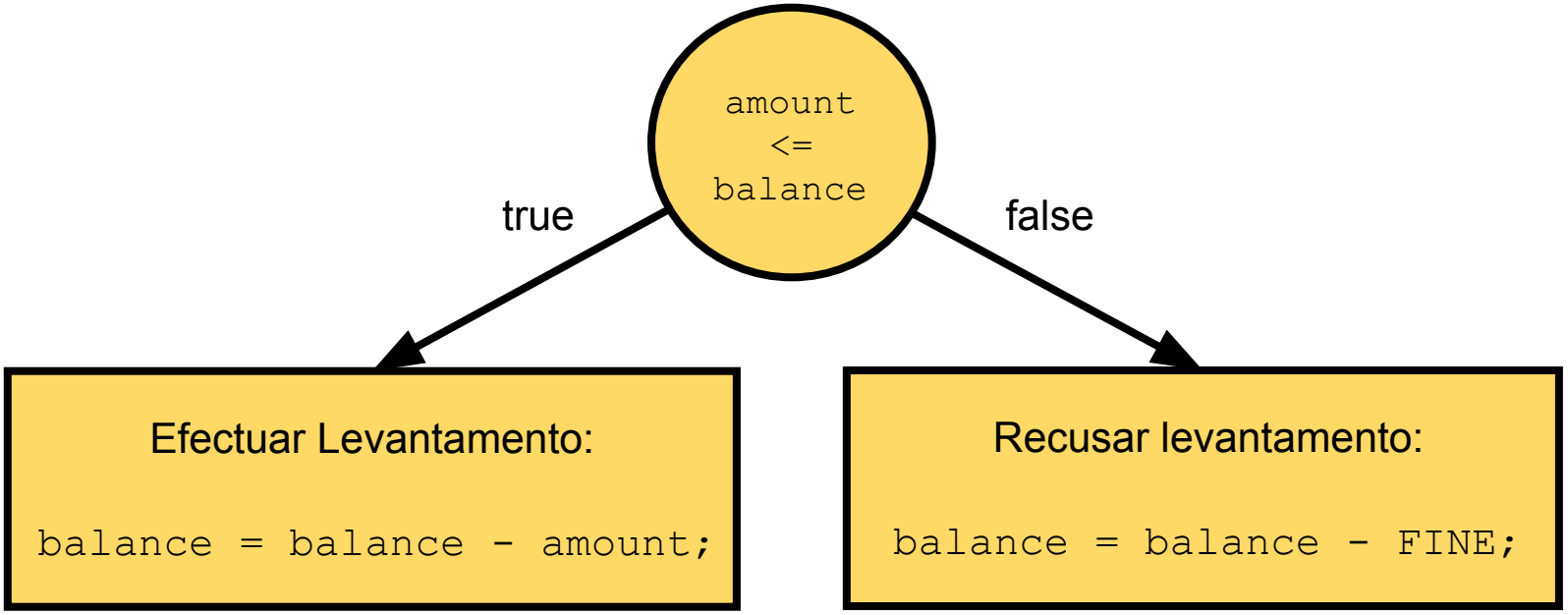
Cenário mais realista

- 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

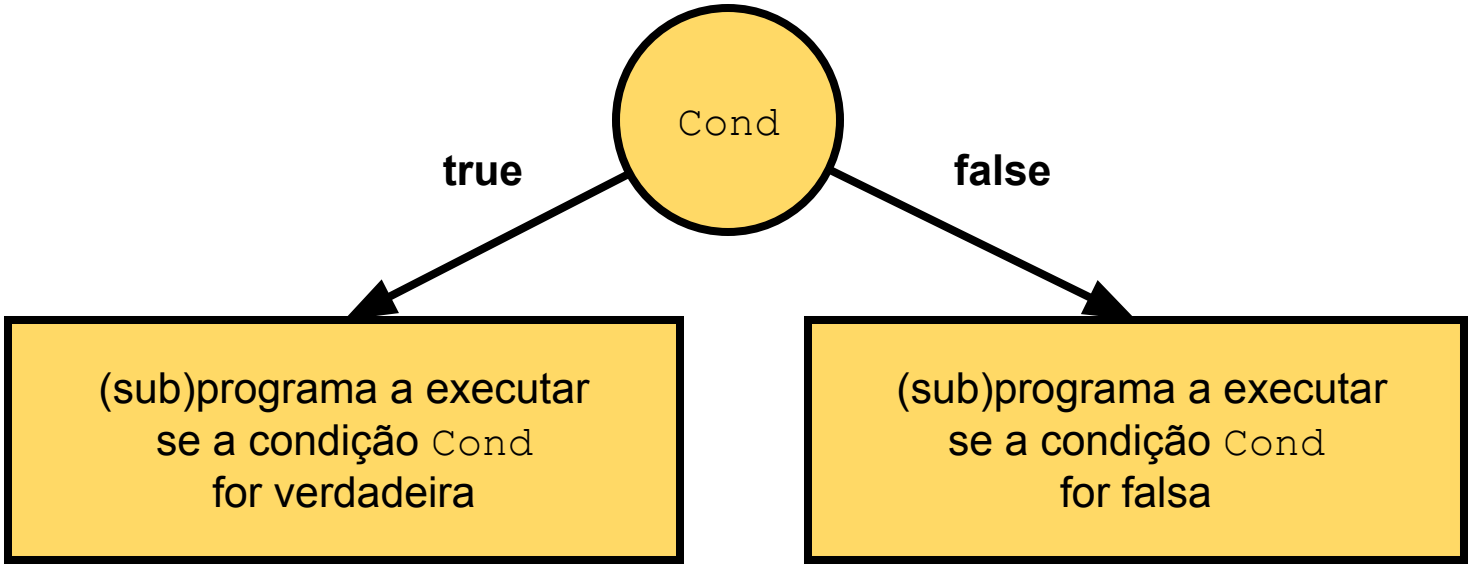
- Vamos considerar que o banco segue as seguintes regras de negócio:
  - O levantamento só é efectuado caso o valor a levantar seja inferior ou igual ao valor do saldo
  - Qualquer tentativa de levantamento de um valor sem saldo suficiente na conta leva à aplicação de uma multa
- O aspecto mais importante 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`

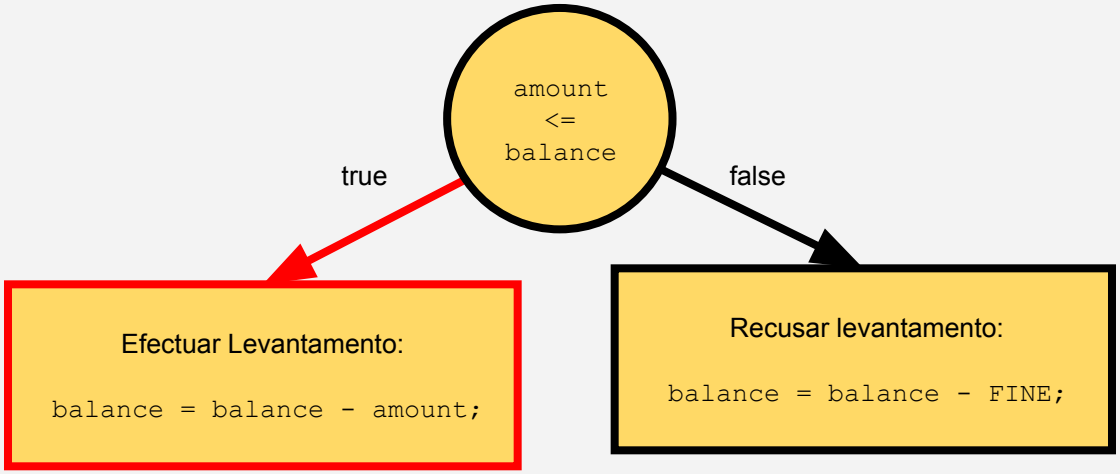
- Na linguagem Java, as decisões exprimem-se usando a instrução **`if-then-else`**

```
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,

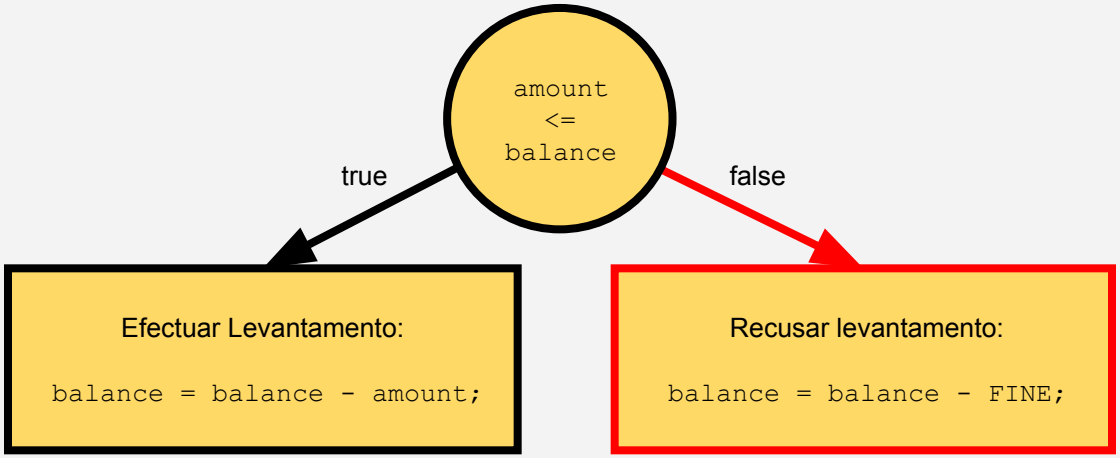
caso contrário,

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
≤ 2.000 €	1%
]2.000 €,10.000 €]	2%
> 10.000 €	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 ba1 = new SafeBankAccount(1000);
System.out.println(ba1.getBalance());
1000
ba1.withdraw(1500);
System.out.println(ba1.getBalance());
800
System.out.println(ba1.computeInterest());
8
ba1.applyInterest();
System.out.println(ba1.getBalance());
808
```

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

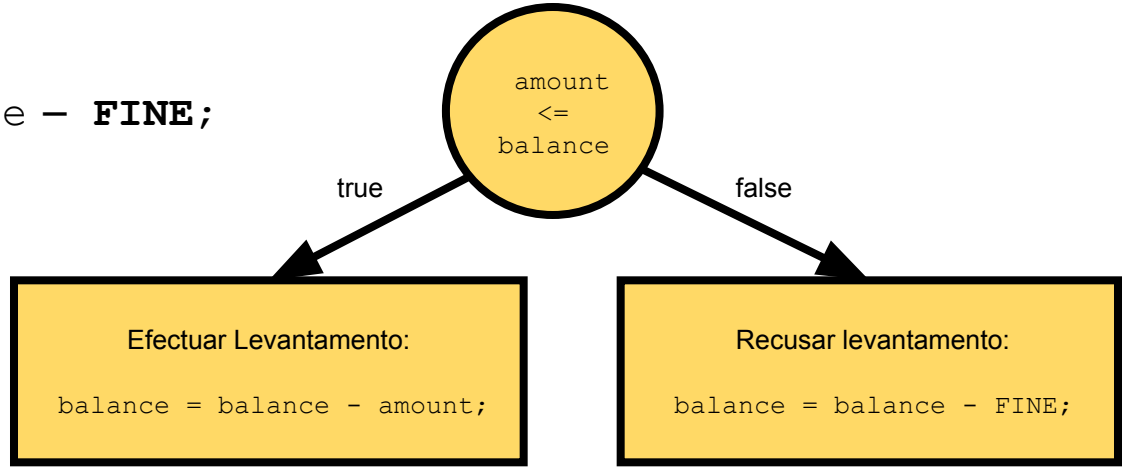
```
void applyInterest()
```

Creditar o juro anual ao saldo da conta

Método `withdraw(int amount)`

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

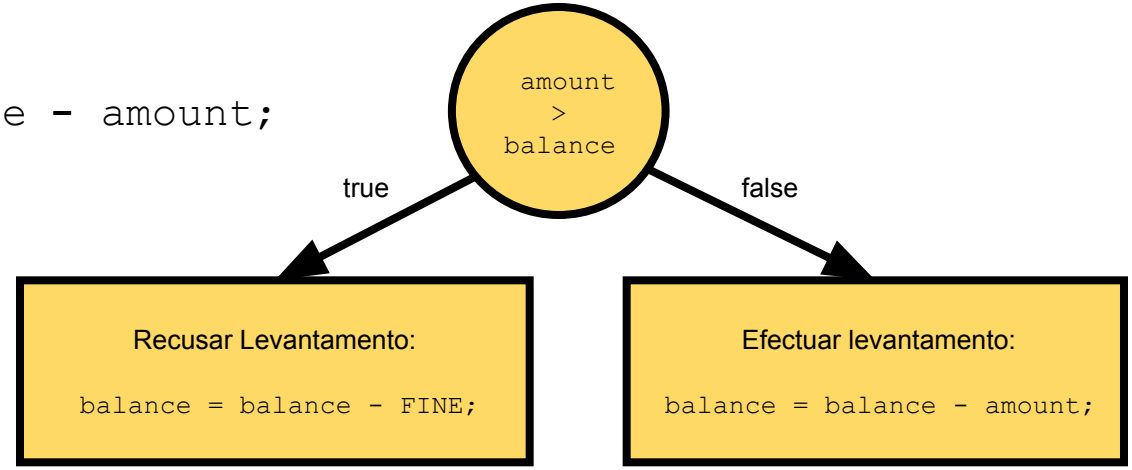
```
public void withdraw(int amount) {
   if (amount <= balance)
      balance = balance - amount;
   else
      balance = balance – FINE;
}
```



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 de classe**

```
public class SafeBankAccount {
  private 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`

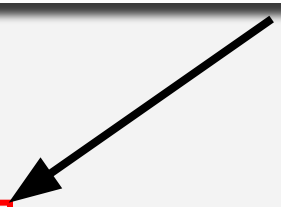
```
int getBalance()
```
Consultar o saldo da conta

```
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 do saldo

}
```

Determinação da taxa de juro

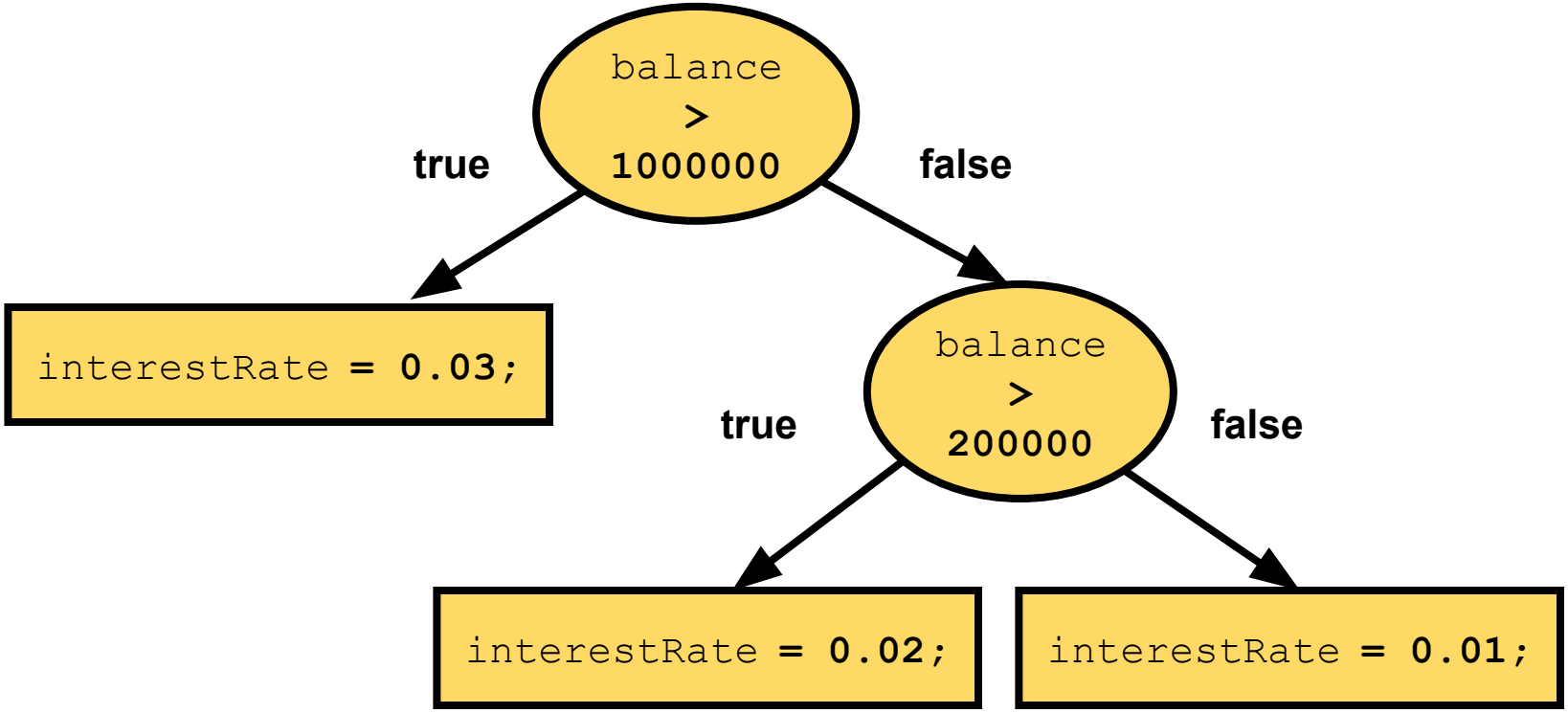
A taxa de juro a aplicar depende do valor saldo existente na conta.

Existem três taxas possíveis:

Valor do Saldo	Taxa
≤ 2.000 €	1%
]2.000 €,10.000 €]	2%
> 10.000 €	3%

Determinação da taxa de juro

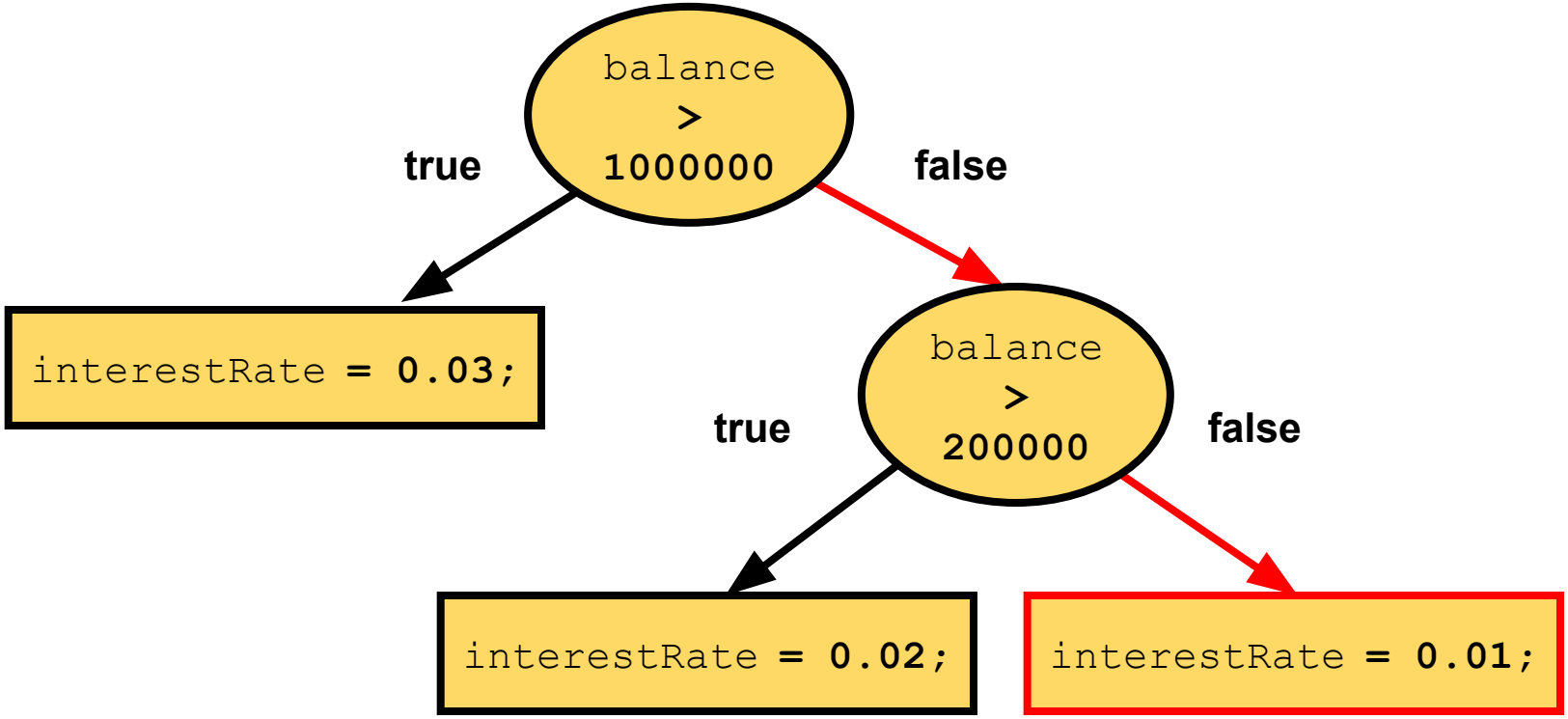
A determinação da taxa de juro pode ser efectuada através das decisões seguintes:



Determinação da taxa de juro

Exemplo:

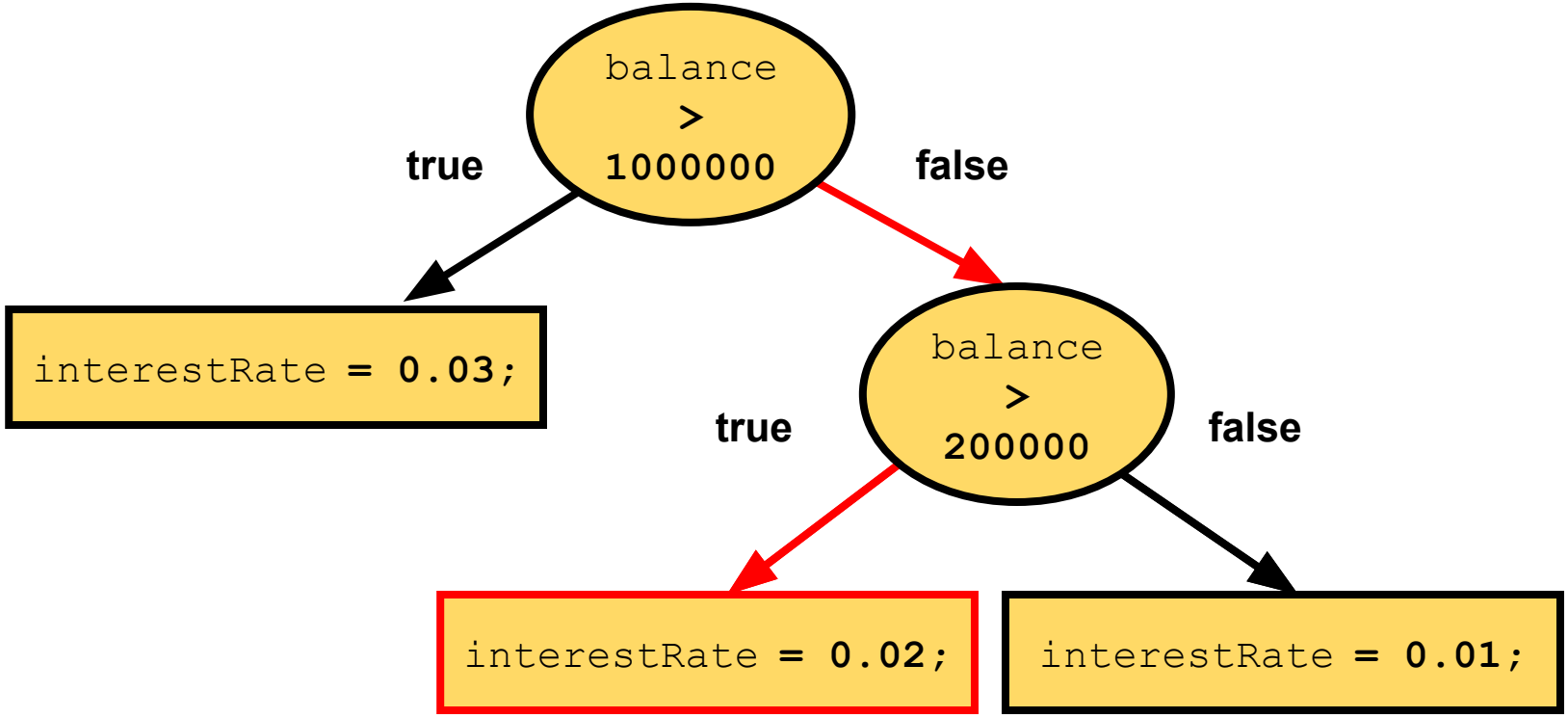
- `balance = 67000`



Determinação da taxa de juro

Exemplo:

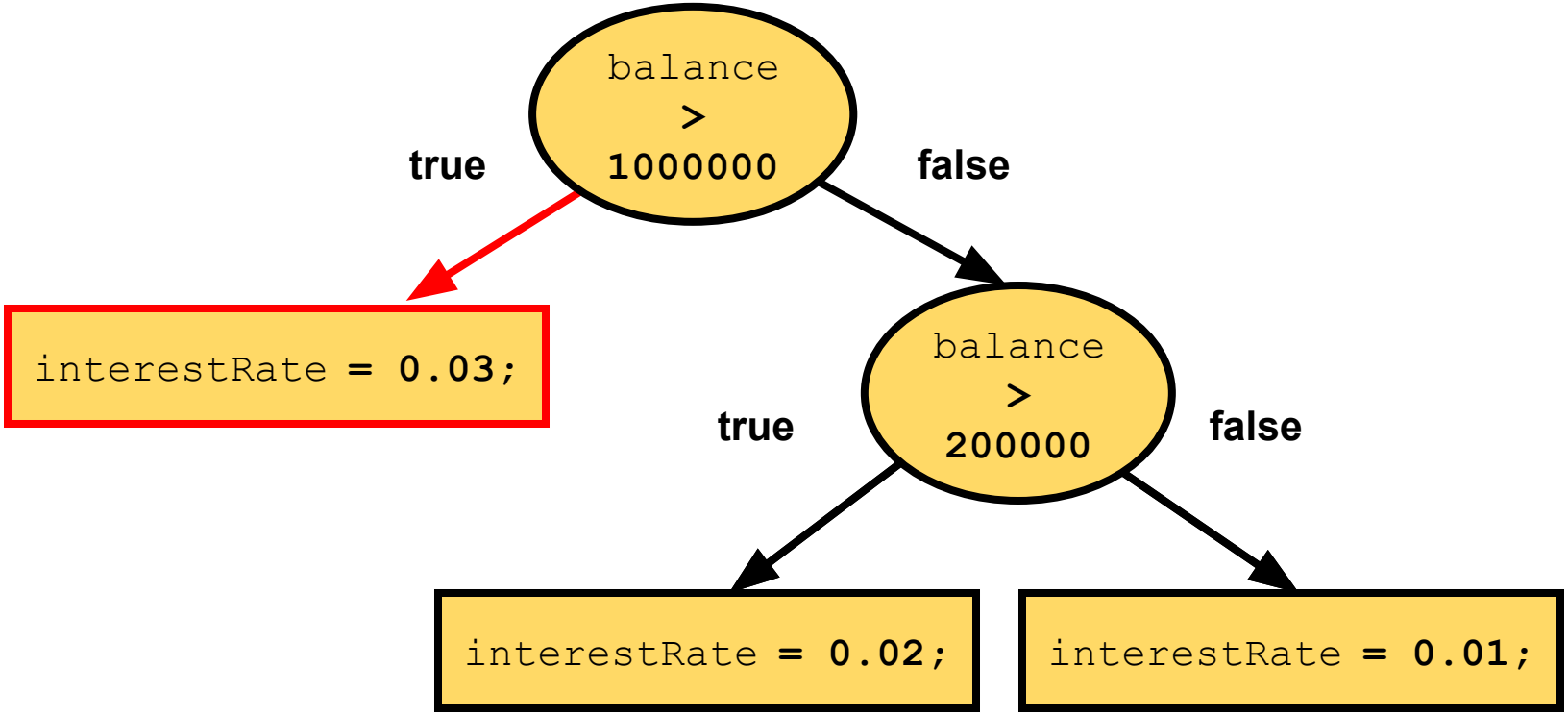
- `balance = 250000`



Determinação da taxa de juro

Exemplo:

- `balance = 1500000`



Método `computeInterest()`

Determinar a taxa de juro a partir do saldo corrente

O saldo deve ser positivo - a pré-condição estabelece a obrigação de só chamar o método se isso for verdade.

```
// Pre: getBalance() > 0
public int computeInterest() {
   double interestRate;

   interestRate = ?;

   Calcular o valor do juro com
   base no saldo

}
```

Declaração de variável local

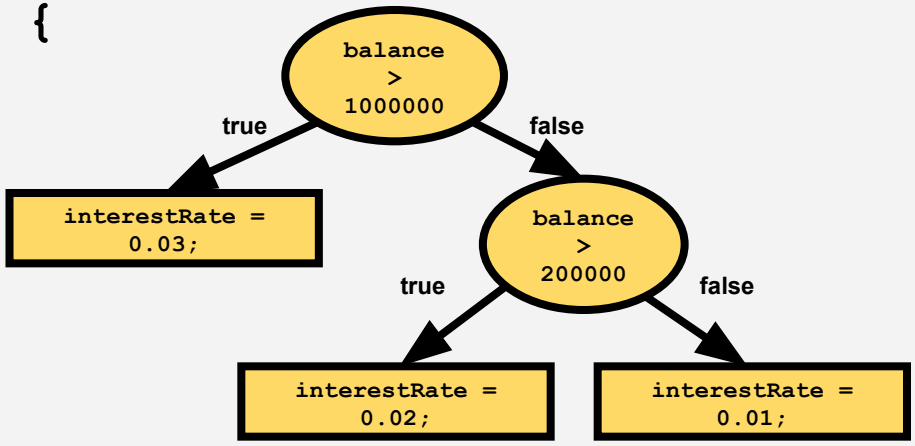
O valor de `interestRate` tem que ser determinado a partir do valor do saldo

Método `computeInterest()`

Determinar a taxa de juro a partir do saldo corrente

```
// Pre: getBalance() > 0
public int computeInterest() {
   double interestRate;
   if (balance > 1000000)
      interestRate = 0.03;
   else if (balance > 200000)
      interestRate = 0.02;
   else interestRate = 0.01;

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

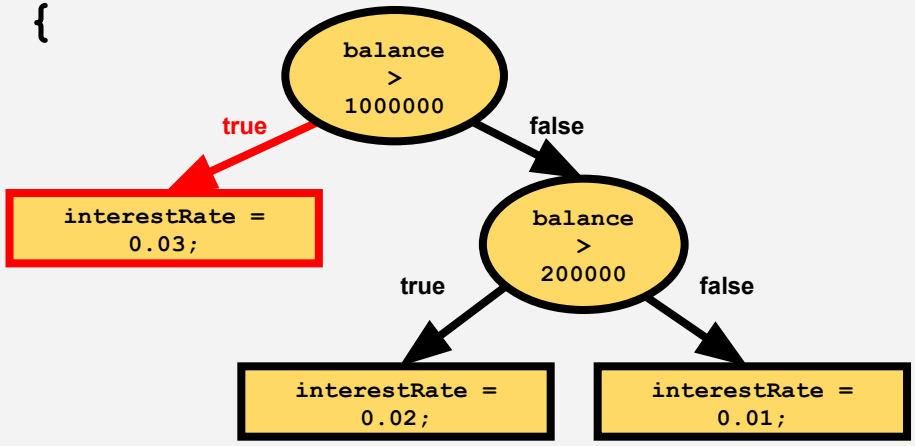


Método `computeInterest()`

Determinar a taxa de juro a partir do saldo corrente

```
// Pre: getBalance() > 0
public int computeInterest() {
   double interestRate;
   if (balance > 1000000)
      interestRate = 0.03;
   else if (balance > 200000)
      interestRate = 0.02;
   else interestRate = 0.01;

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

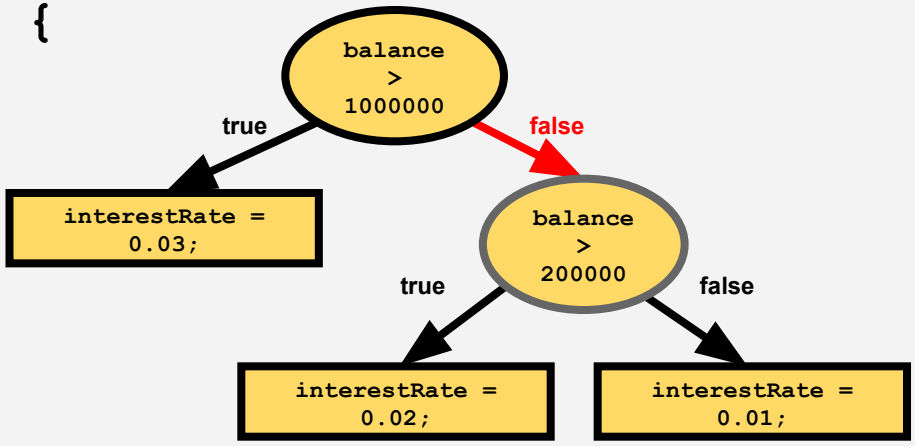


Método `computeInterest()`

Determinar a taxa de juro a partir do saldo corrente

```
// Pre: getBalance() > 0
public int computeInterest() {
   double interestRate;
   if (balance > 1000000)
      interestRate = 0.03;
   else if (balance > 200000)
      interestRate = 0.02;
   else interestRate = 0.01;

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

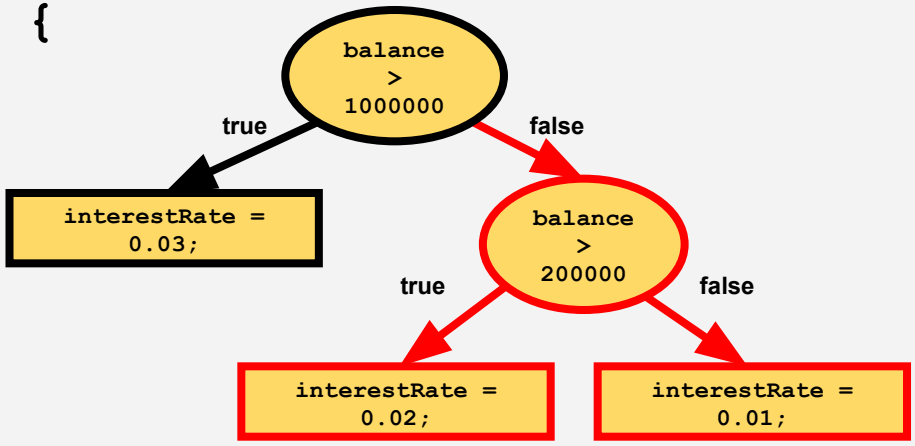


Método `computeInterest()`

Determinar a taxa de juro a partir do saldo corrente

```
// Pre: getBalance() > 0
public int computeInterest() {
   double interestRate;
   if (balance > 1000000)
      interestRate = 0.03;
   else if (balance > 200000)
      interestRate = 0.02;
   else interestRate = 0.01;

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

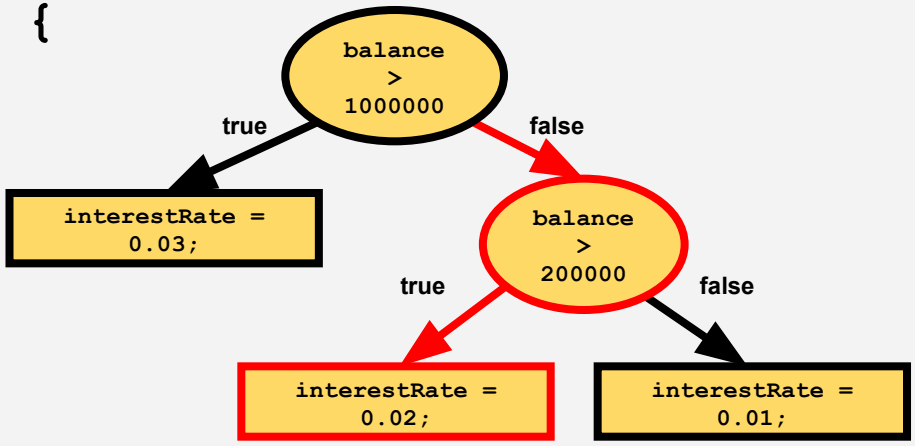


Método `computeInterest()`

Determinar a taxa de juro a partir do saldo corrente

```
// Pre: getBalance() > 0
public int computeInterest() {
   double interestRate;
   if (balance > 1000000)
      interestRate = 0.03;
   else if (balance > 200000)
      interestRate = 0.02;
   else interestRate = 0.01;
}
```

Calcular o valor do juro com base no saldo

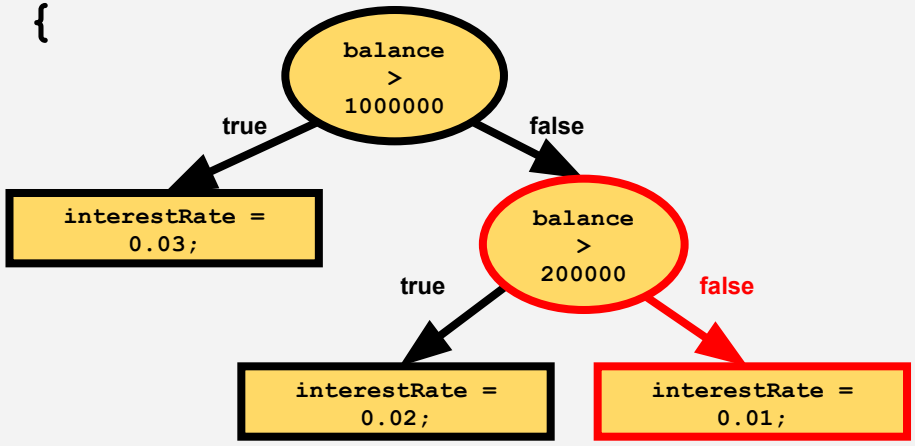


Método `computeInterest()`

Determinar a taxa de juro a partir do saldo corrente

```
// Pre: getBalance() > 0
public int computeInterest() {
   double interestRate;
   if (balance > 1000000)
      interestRate = 0.03;
   else if (balance > 200000)
      interestRate = 0.02;
   else interestRate = 0.01;

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



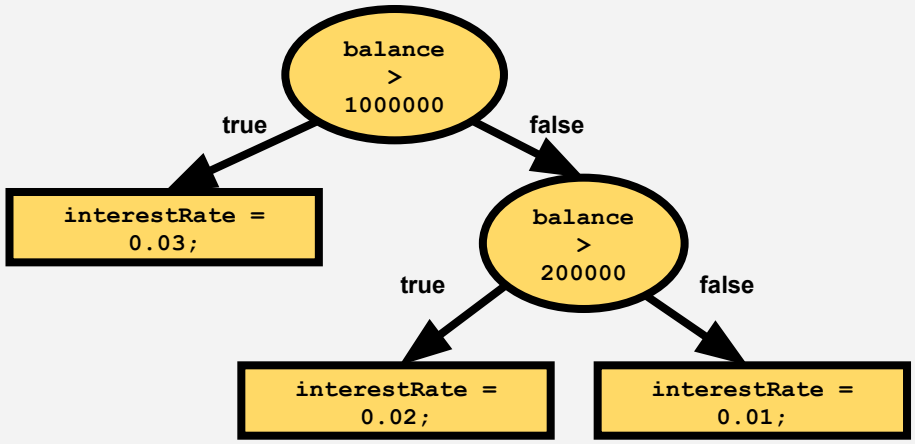
Método `computeInterest()`

Determinar a taxa de juro a partir do saldo corrente

```
// Pre: getBalance() > 0
public int computeInterest() {
   double interestRate;
   if (balance > 1000000)
      interestRate = 0.03;
   else if (balance > 200000)
      interestRate = 0.02;
   else interestRate = 0.01;

   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

```
// Pre: getBalance() > 0
public int computeInterest() {
  double interestRate;
  if (balance > 1000000)
    interestRate = 0.03;
  else if (balance > 200000)
    interestRate = 0.02;
  else interestRate = 0.01;
  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

```
// Pre: getBalance() > 0
public int computeInterest() {
  double interestRate;
  if (balance > 1000000)
    interestRate = 0.03;
  else if (balance > 200000)
    interestRate = 0.02;
  else interestRate = 0.01;
  return balance * interestRate;
}
```

Determinar qual a taxa a partir do valor do saldo

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

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

Método `computeInterest()`

Determinar a taxa de juro a partir do saldo corrente

```
// Pre: getBalance() > 0
public int computeInterest() {
   double interestRate;
   if (balance > 1000000)
     interestRate = 0.03;
   else if (balance > 200000)
     interestRate = 0.02;
   else interestRate = 0.01;
   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`**

`Math.round` arredonda mas dá precisão superior a **`int`**... assim ainda não funciona

Método `computeInterest()`

Método `computeInterest()`

```
// Pre: getBalance() > 0
public int computeInterest() {
  double interestRate;
  if (balance > 1000000)
    interestRate = 0.03;
  else if (balance > 200000)
    interestRate = 0.02;
  else interestRate = 0.01;
  return Math.toIntExact(Math.round(balance*interestRate));
}
```

Esta solução funciona, mas devemos fazer melhor!

Método computeInterest()

Método computeInterest()

```
// Pre: getBalance() > 0
public int computeInterest() {
   double interestRate;
   if (balance > 1000000)
      interestRate = 0.03;
   else if (balance > 200000)
      interestRate = 0.02;
   else interestRate = 0.01;
   Return Math.toIntExact(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()

```
// Pre: getBalance() > 0
public int computeInterest() {
   double interestRate;
   if (balance > CATEGORY1)
     interestRate = RATE_CATEGORY1;
   else if (balance > CATEGORY2)
     interestRate = RATE_CATEGORY2;
   else interestRate = RATE_CATEGORY3;
   return Math.toIntExact(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 {
   ...
   private static final int CATEGORY1 = 1000000;
   private static final int CATEGORY2 = 200000;
   private static final double RATE_CATEGORY1 = 0.03;
   private static final double RATE_CATEGORY2 = 0.02;
   private static final double RATE_CATEGORY3 = 0.01;
   ...
}
```

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

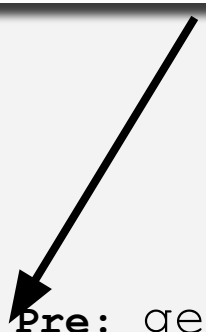
```
int computeInterest()
```

Calcular qual o valor do juro anual a aplicar

```
void applyInterest()
```

Creditar o juro anual ao saldo da conta

Como programar o método `applyInterest` ?



`Pre:` `getBalance()` **`> 0`**

Método `applyInterest()`

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

```
    // Pre: getBalance() > 0
public void applyInterest() {
  balance = balance + this.computeInterest();
}
```

O identificador **`this`** refere o **próprio objecto**, neste caso, o mesmo objecto em que a operação `applyInterest()` está a ser executada

Tipo int e Operações Associadas

Operações que produzem um valor inteiro (**`int`**) a partir de valores inteiros

const	constante – um valor fixo (ou *literal*) é uma expressão
var	variável – uma variável é uma expressão
expr **+** *expr*	adição – a adição de expressões é uma expressão
expr **-** *expr*	subtracção
expr ***** *expr*	multiplicação
expr **/** *expr*	divisão inteira (quociente)
expr **%** *expr*	módulo (resto da divisão inteira)
-*expr*	Operação unária de alteração do sinal da expressão
var **++**	devolve o valor de *var* e depois incrementa *var* numa unidade
var **--**	devolve o valor de *var* e depois decrementa *var* numa unidade

Tipo double e Operações Associadas

- A linguagem Java oferece dois tipos de dados primitivos para representar valores reais: **double** e **float**. O tipo **double** é o normalmente se escolhe, por ser mais preciso. Será o único que usaremos na nossa disciplina.
- Os valores de tipo **double** são números de vírgula flutuante com 64 bits. Têm cerca de 16 dígitos decimais de precisão.
- Literais de tipo `double`
  - Usa-se o ponto decimal ou notação científica
    - `3.14`
    - `1.234e-23`

Tipo double e Operações Associadas

- Operações que produzem um valor de tipo **`double`** a partir de valores de tipo **`double`**

const	constante
var	variável
expr1 + *expr2*	adição
expr1 – *expr2*	subtracção
expr1 * *expr2*	multiplicação
expr1 / *expr2*	divisão
- expr	alteração de sinal

Quando realizamos operações entre números, o resultado tem o tipo da expressão que tiver maior precisão. Por exemplo, somar um inteiro com um real produz um real.

Desta forma não há truncagens de valores nas contas.

Tipo double e Operações Associadas

- A linguagem Java é complementada por um vasto conjunto de bibliotecas que permitem "aumentar" a linguagem mantendo-a relativamente simples
- A classe de biblioteca **`Math`**, disponível no ambiente Java, oferece um conjunto bastante completo de constantes e operações para números reais:
  - Constantes úteis:
    - *e*
    - *pi*
  - Operações
    - Exponencial
    - Logaritmo
    - Raiz quadrada
    - Funções trigonométricas
    - e muito, muito mais…

Tipo double e Operações Associadas

- Constantes úteis
  - `Math.PI` π
  - `Math.E` *e*
- Operações úteis sobre valores de tipo `double`
  - `Math.round(` *expr* `)` arredonda o valor de *expr* ao inteiro mais próximo
  - `Math.sin(` *expr* `)` seno (argumento em radianos)
  - `Math.cos(` *expr* `)` coseno (argumento em radianos)
  - `Math.sqrt(` *expr* `)` raiz quadrada
  - `Math.log(` *expr* `)` logaritmo de base *e*
- Para obter a lista completa consulte a bibliografia de IP, disponível na página da disciplina no CLIP e no Moodle (zona Bibliografia).

Tipo boolean e Operações Associadas

Operações que produzem um valor booleano (**`true`** ou **`false`**) a partir de valores escalares (de tipo **`int`** ou **`double`**)

Para já, vamos assumir que *expr1* e *expr2* têm que ser **`int`** ou **`double`**

expr1 `>` *expr2*	maior
expr1 `>=` *expr2*	maior ou igual
expr1 `<` *expr2*	menor
expr1 `<=` *expr2*	menor ou igual
expr1 `==` *expr2*	igual
expr1 `!=` *expr2*	diferente (não igual)

Importante: Estes operadores chamam-se **operadores relacionais**.

Tipo boolean e Operações Associadas

Operações que produzem um valor booleano (de tipo **`boolean`**) a partir de valores booleanos (de tipo **`boolean`**)

Aqui, vamos assumir que *expr1* e *expr2* têm que ser **`boolean`**

expr1 **`&&`** *expr2*	conjunção lógica
expr1 **`||`** *expr2*	disjunção lógica
`!` *expr*	negação lógica

Importante: Estes operadores chamam-se **operadores lógicos**.

Ordem de Precedência dos Operadores

Para desambiguar a ordem de aplicação dos operadores em expressões compostas, podem ser usados os parêntesis `(` e `)`.

– Por exemplo, `(2+x)*2` em vez `2+x*2` (que é interpretado como `2+(x*2)`)

Ordem de precedência (a começar nos operadores mais fortes)

```
++  --
!
*   /   %
+   -
<   <=   >  >=
==  !=
&&
||
```

maior prisão (maior precedência)

menor prisão (menor precedência)

Exemplo: a expressão

```
(x++ - 1.35*2 > 4)&& false || true
```

é interpretada como

```
((((x++)-(1.35*2))> 4)&& false) || true
```