Repetição de Comandos

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

Conta Bancária Segura
(nova interacção com o utilizador)

Interacção com o utilizador:

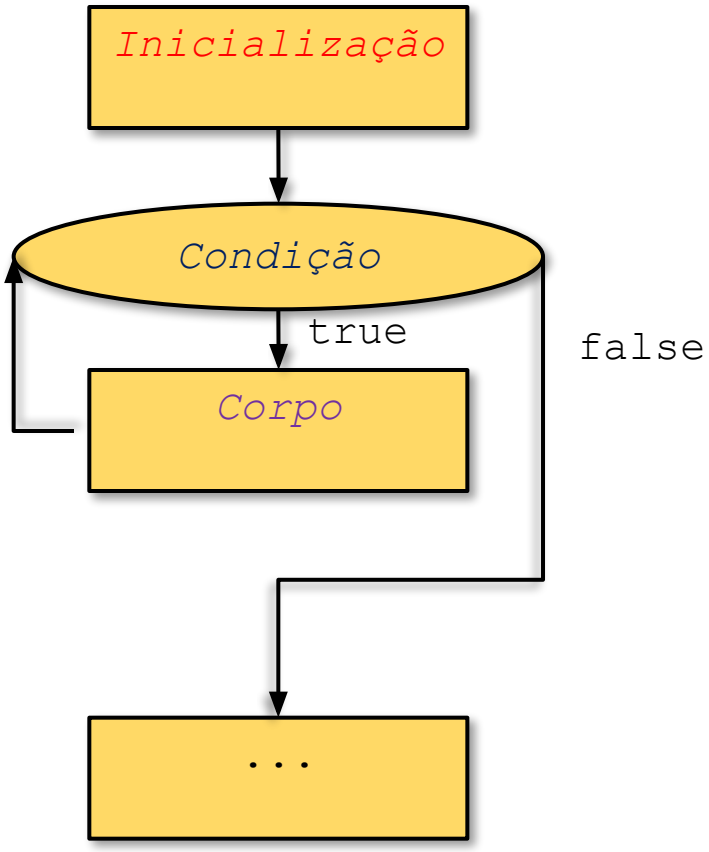
- Pede ao utilizador o saldo inicial duma conta bancária.
- Enquanto o utilizador não escolher Sair (opção 6), o programa deve processar pedidos sucessivos do utilizador.
- As opções são:

```
1- Depositar
2- Levantar
3- Consultar saldo
4- Consultar juro anual
5- Creditar juro anual
6- Sair
```

Pretende-se permitir ao utilizador executar uma sequência arbitrária de comandos (*tantos e os que quiser*).

Como programar isto (*repetir* enquanto o utilizador quiser)?

O ciclo `while`




```
Inicialização
while ( Condição ) {
   Corpo
}
```

O Ciclo `while`

- Enquanto a condição do ciclo for verdadeira, os comandos no corpo do ciclo serão executados repetidamente
- Se inicialmente a condição for falsa, o corpo do ciclo nunca chegará a ser executado ...
- Caso contrário, será executado uma vez, sendo depois a condição avaliada de novo ...
- Se continuar a ser verdadeira, o corpo será executado de novo, e a condição reavaliada ...
- O ciclo só pára de repetir quando a condição for falsa (o que pode nunca acontecer… oops!)

O Ciclo `while`

Os três elementos fundamentais de um ciclo são

- Inicialização:
  Estabelece os valores iniciais das variáveis que podem ser alteradas em cada passo do ciclo.
- Condição:
  Indica se o ciclo deve repetir mais um passo.
- Corpo:
  - Indica o que deve ser feito em cada passo do ciclo
  - Um "passo" também se chama "iteração"
  - Deve alterar as variáveis envolvidas na condição, para garantir que esta chega a ser falsa (e que então o ciclo pára)

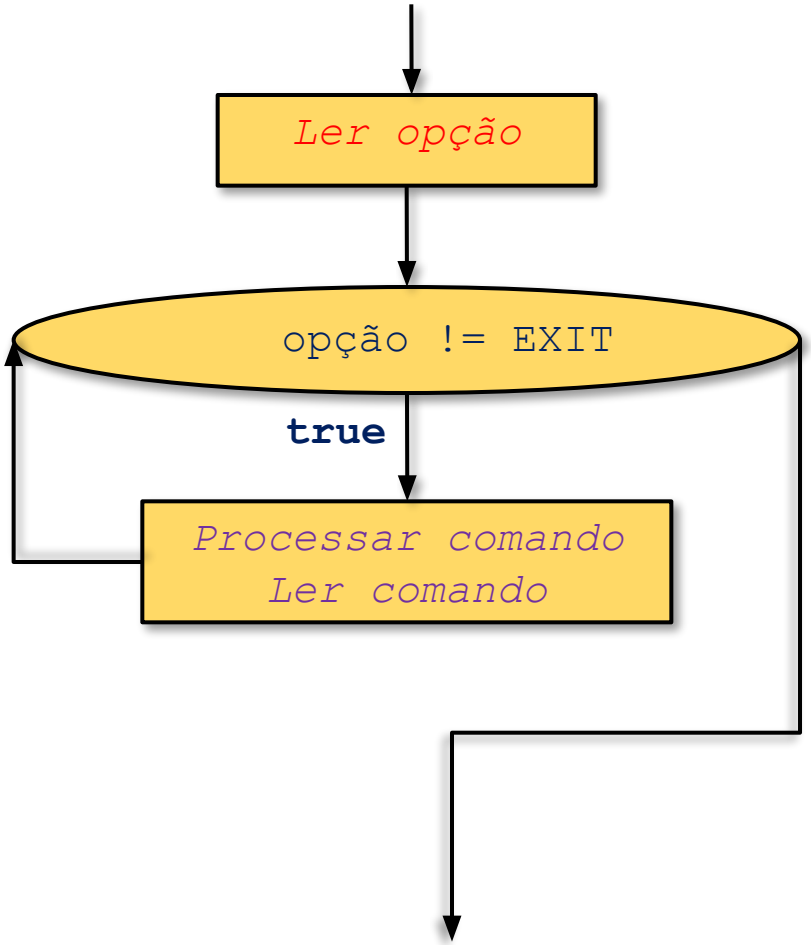
```
Inicialização
while ( Condição ) {
    Corpo
}
```

Nota – tipicamente, o **corpo do ciclo inclui:**

- *Acção*: contribui para o objectivo do ciclo
- *Progresso*: contribui para tornar a condição falsa, terminando o ciclo

A acção e o progresso podem ser codificadas na mesma instrução.

Conta Bancária Segura
(nova interacção com o utilizador)




```
Inicialização
while ( Condição ) {
   Corpo
}
```

Mostrar as opções disponíveis

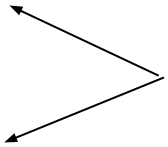
```
private static final int DEPOSIT = 1;
...
private static final int EXIT = 6;


public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    SafeBankAccount account = createAccount(in, "cliente");
    printMenu();
    int option = getIntValue(in,"Opcao: ");
    while(option != EXIT){
        executeOption(in, account, option);
        option = getIntValue(in,"Opcao: ");
    }
    in.close();
}
```

Mostrar as opções disponíveis

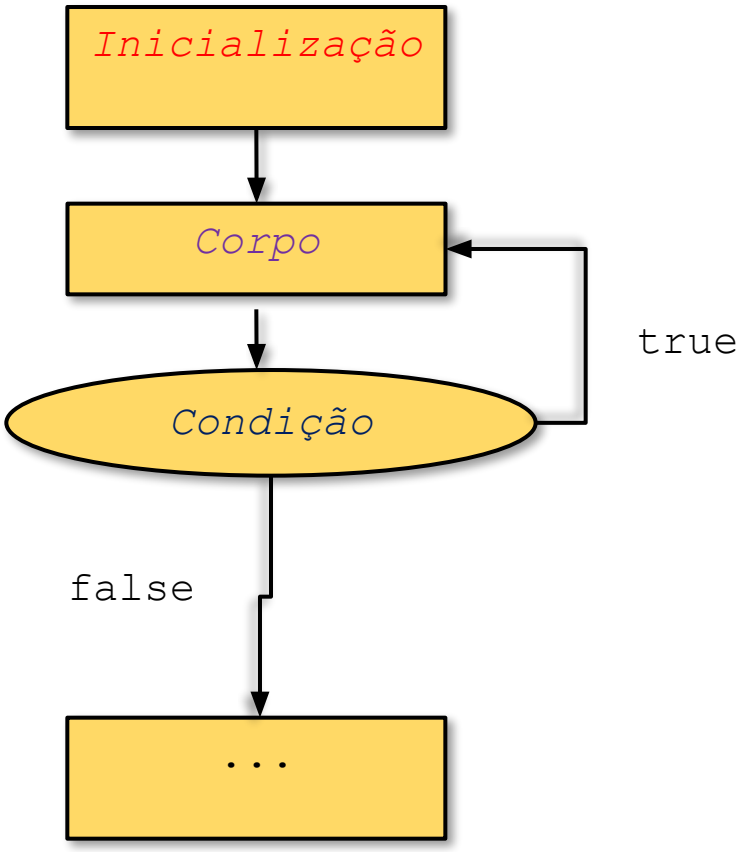
```
private static final int DEPOSIT = 1;
...
private static final int EXIT = 6;


public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    SafeBankAccount account = createAccount(in, "cliente");
    printMenu();
    int option = getIntValue(in,"Opcao: ");
    while(option != EXIT){
        executeOption(in, account, option);
        option = getIntValue(in,"Opcao: ");
    }
    in.close();
}
```



Repetição: e se testássemos a condição depois de ler?

O ciclo `do_while`




```
Inicialização
do {
   Corpo
}
while ( Condição );
```

Mostrar as opções disponíveis

```
private static final int DEPOSIT = 1;
...
private static final int EXIT = 6;


public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    SafeBankAccount account = createAccount(in, "cliente");
    printMenu();
    int option;
    do {
        option = getIntValue(in,"Opcao: ");
        executeOption(in, account, option);
    }
    while(option != EXIT);
    in.close();
}
```

Conta Bancária Segura (Poupanças)

Recorde a 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
(nova funcionalidade)

Nova funcionalidade

– Calcular o número de anos necessários (aplicando a taxa anual correspondente) para que o saldo da conta seja maior que um dado valor positivo (superior ou igual ao saldo actual da conta). Note que este cálculo só é possível de realizar se o saldo da conta for superior a zero.

Exemplo de poupança: uma conta com um saldo inicial de 10000 (100 euros) e uma taxa de juro anual de 1%

Ano	Saldo (em cêntimos)
0	10000
1	10100
2	10201
3	10303
4	...

Conta Bancária Segura

- Interface de `SafeBankAccount`:

`public void` `deposit(`**`int`** `amount)`

Deposita a importância `amount` na conta

`Pre: amount > 0`

`public void` `withdraw(`**`int`** `amount)`

Levanta a importância `amount` na conta ou aplica a multa (`FINE`)

`Pre: amount > 0`

`public int` `getBalance()`

Consulta o saldo da conta

`public boolean` `redZone()`

Indica se a conta está devedora

`public int` `computeInterest()`

Calcula qual o valor do juro anual a aplicar

`public void` `applyInterest()`

Credita o juro anual ao saldo da conta

Novo método

`public int` `howManySavingsYears(`**`int`** `target)`

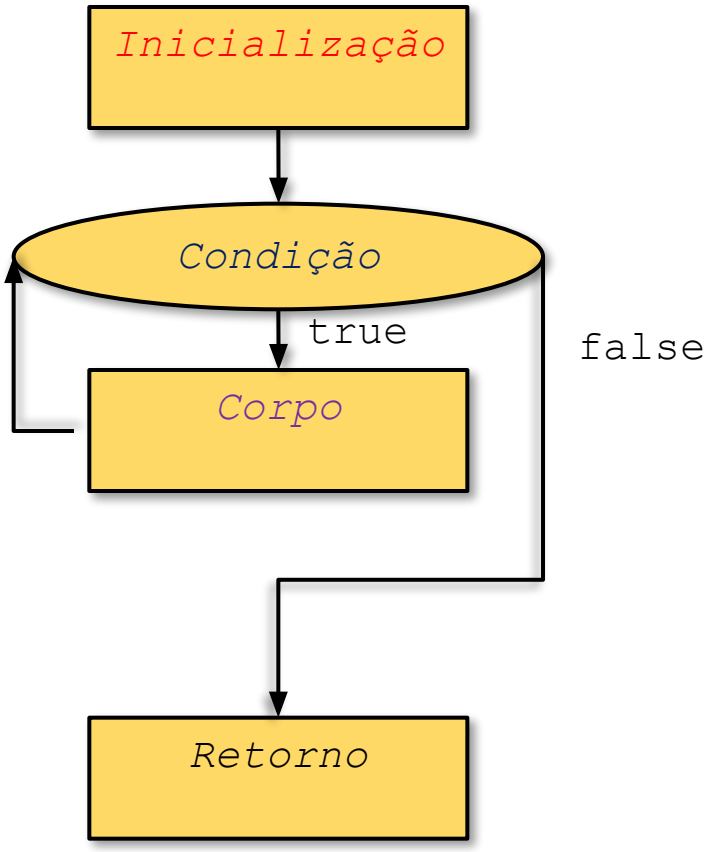
Calcula o número de anos necessários para que o saldo da conta seja maior que `target`.

`Pre: getBalance() >= 100 && getBalance() <= target`

Poupança (versão 1)

`int howManySavingsYears(int target)`

Neste método simula-se a passagem dos anos.



Inicialização - número de anos começa a zero -(variável local)

Condição - enquanto o objectivo (target) não tiver sido atingido (saldo da conta inferior ao objectivo)

Corpo - contributo de cada iteração para o objectivo:

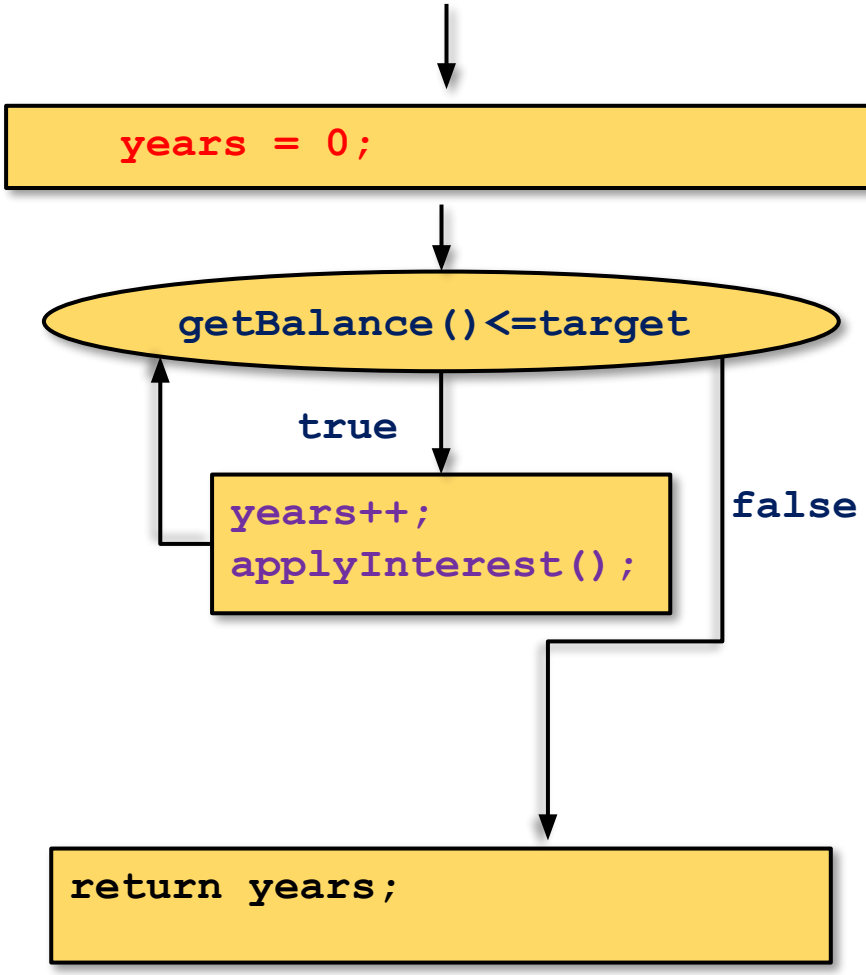
1. Aplica o juro;
2. Incrementa o número de anos

Retorno - devolver o número de anos

Poupança (versão 1)

`int howManySavingsYears(int target)`

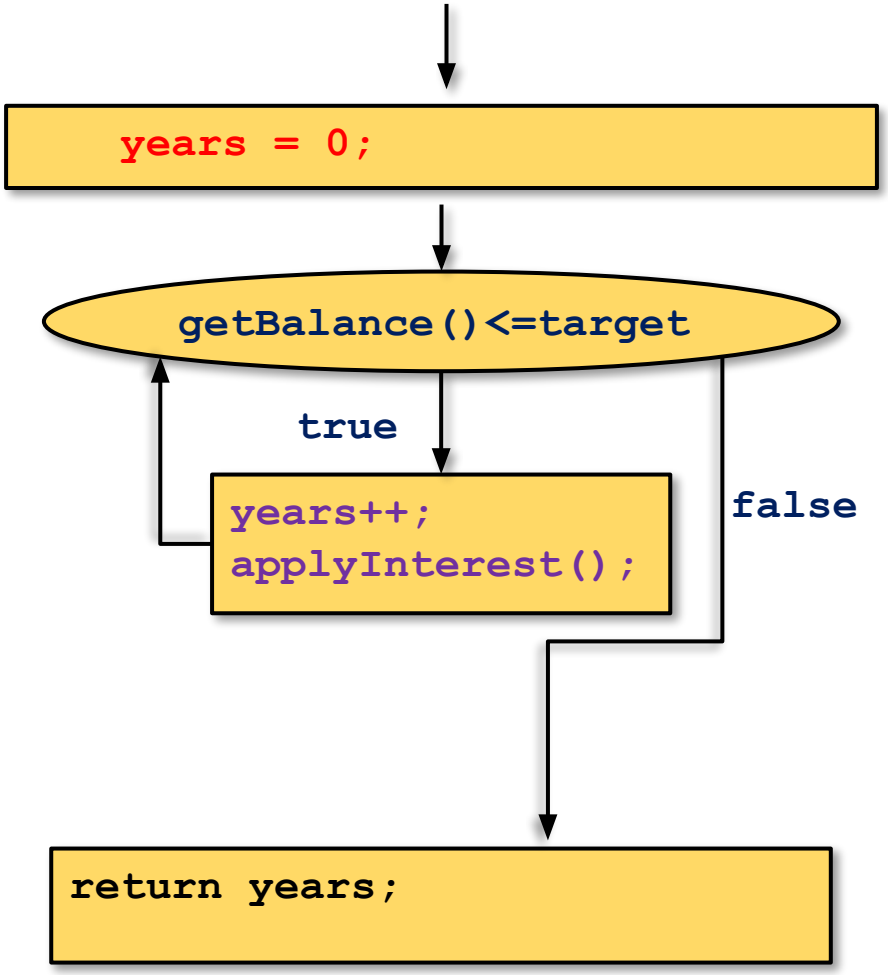
Neste método simula-se a passagem dos anos.



Poupança (versão 1 - errada!)

`int howManySavingsYears(int target)`

Neste método simula-se a passagem dos anos.

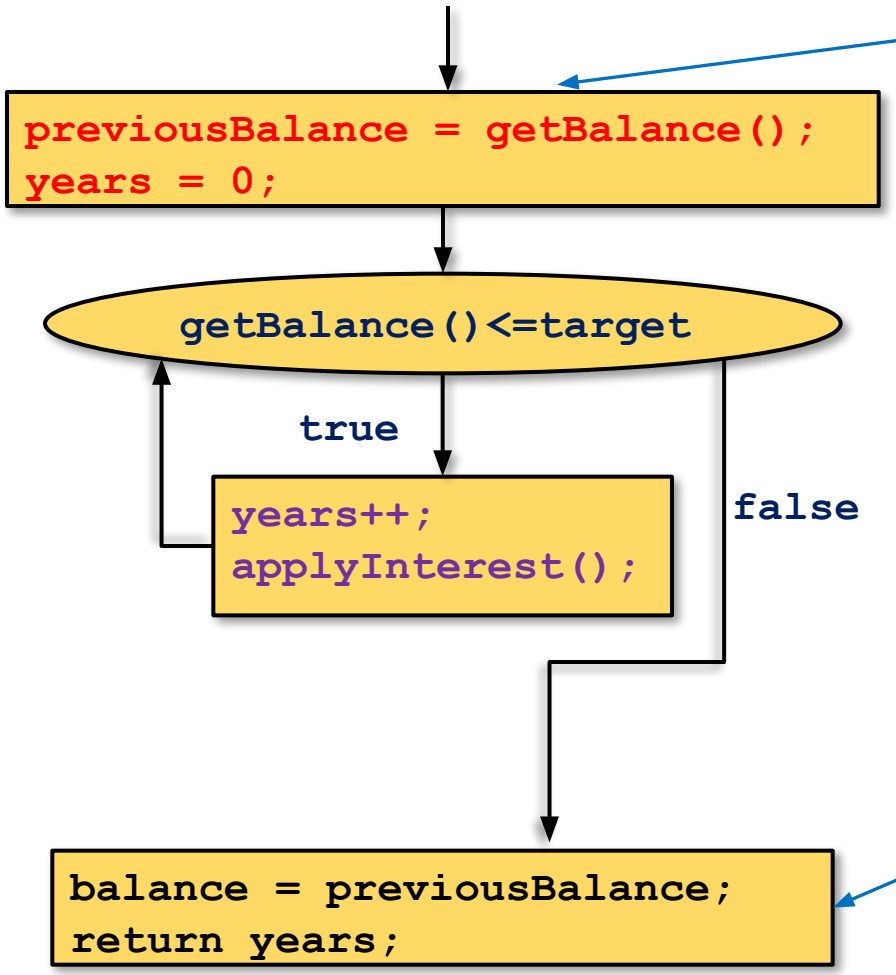


Problema:

O método `applyInterest` altera o saldo da conta

Poupança (versão 1 - mau estilo)

`int howManySavingsYears(int target)`



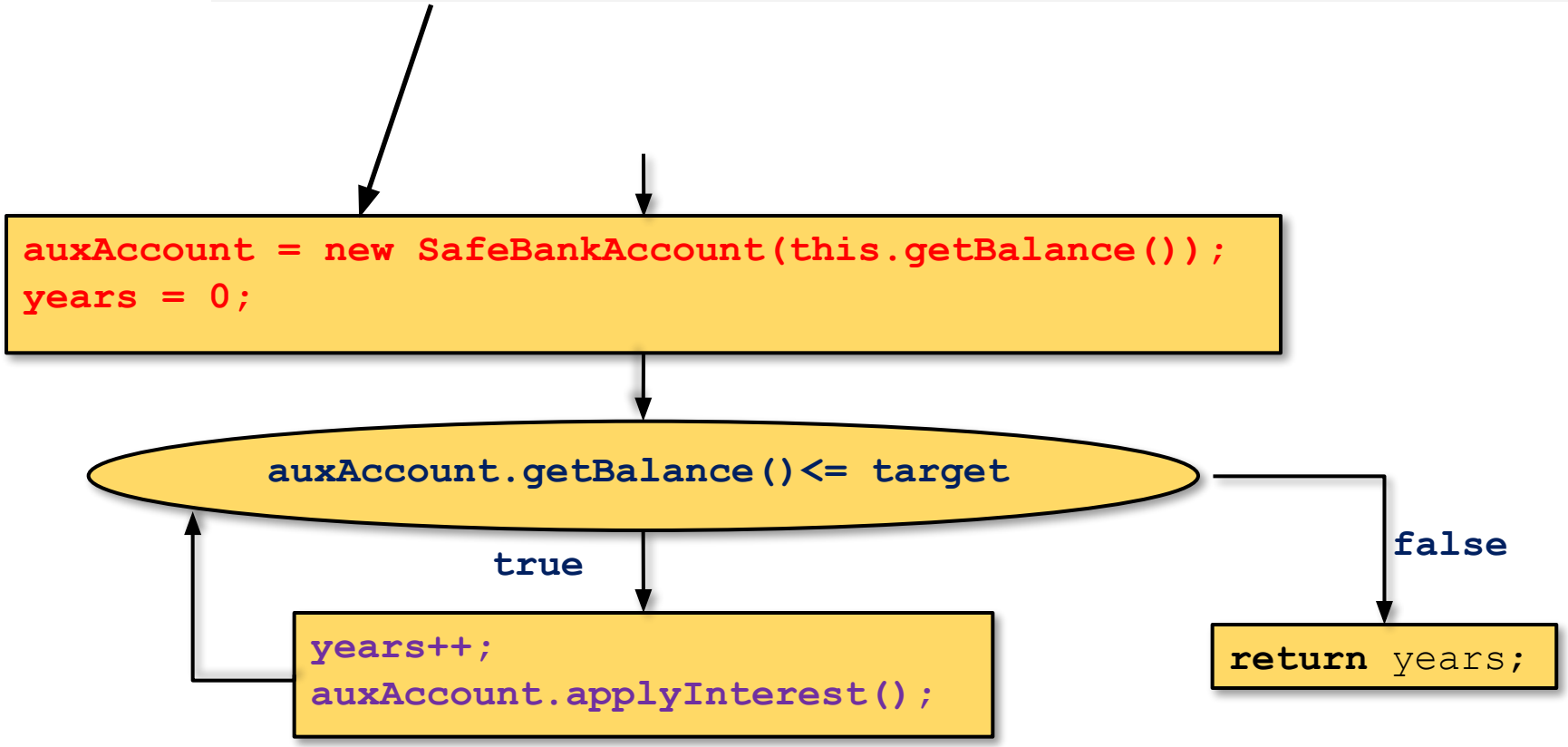
Temos de guardar numa variável auxiliar o saldo da conta

Neste método simula-se a passagem dos anos, logo, no final, temos de repor o saldo na conta

Poupança (versão 2 - bom estilo)

`int howManySavingsYears(int target)`

Utilização de um objecto `SafeBankAccount (auxAccount)`, para simular a passagem dos anos.



A classe SafeBankAccount

```
// Pre: getBalance() > 0 && getBalance() < target
public int howManySavingsYears(int target)
{
    int years = 0;
    SafeBankAccount auxAccount =
         new SafeBankAccount (this.getBalance());

    while (auxAccount.getBalance() < target) {
        years++;
        auxAccount.applyInterest();
    }
    return years;
}
```

Conta Bancária Segura

- Recupere a sua classe `SafeBankAccount`, cujos objectos representam contas bancárias. Vamos acrescentar uma nova operação a essa classe.
- Programe a sua classe no Eclipse.
- Teste a classe `SafeBankAccount`, e verifique que se comporta como se espera, implementando a nova interacção com o utilizador na classe `Main`.