

Repetição de Comandos

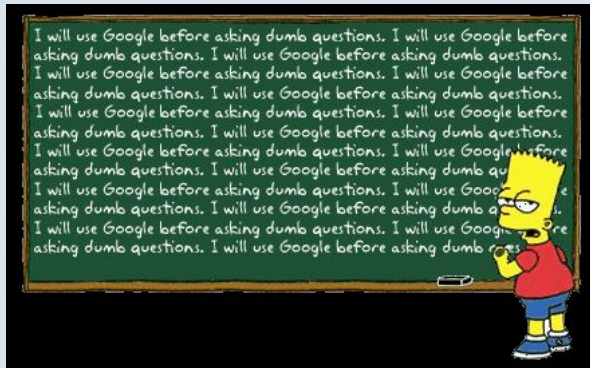
Material didáctico elaborado pelas diferentes equipas de
Introdução à Programação

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 Repetições

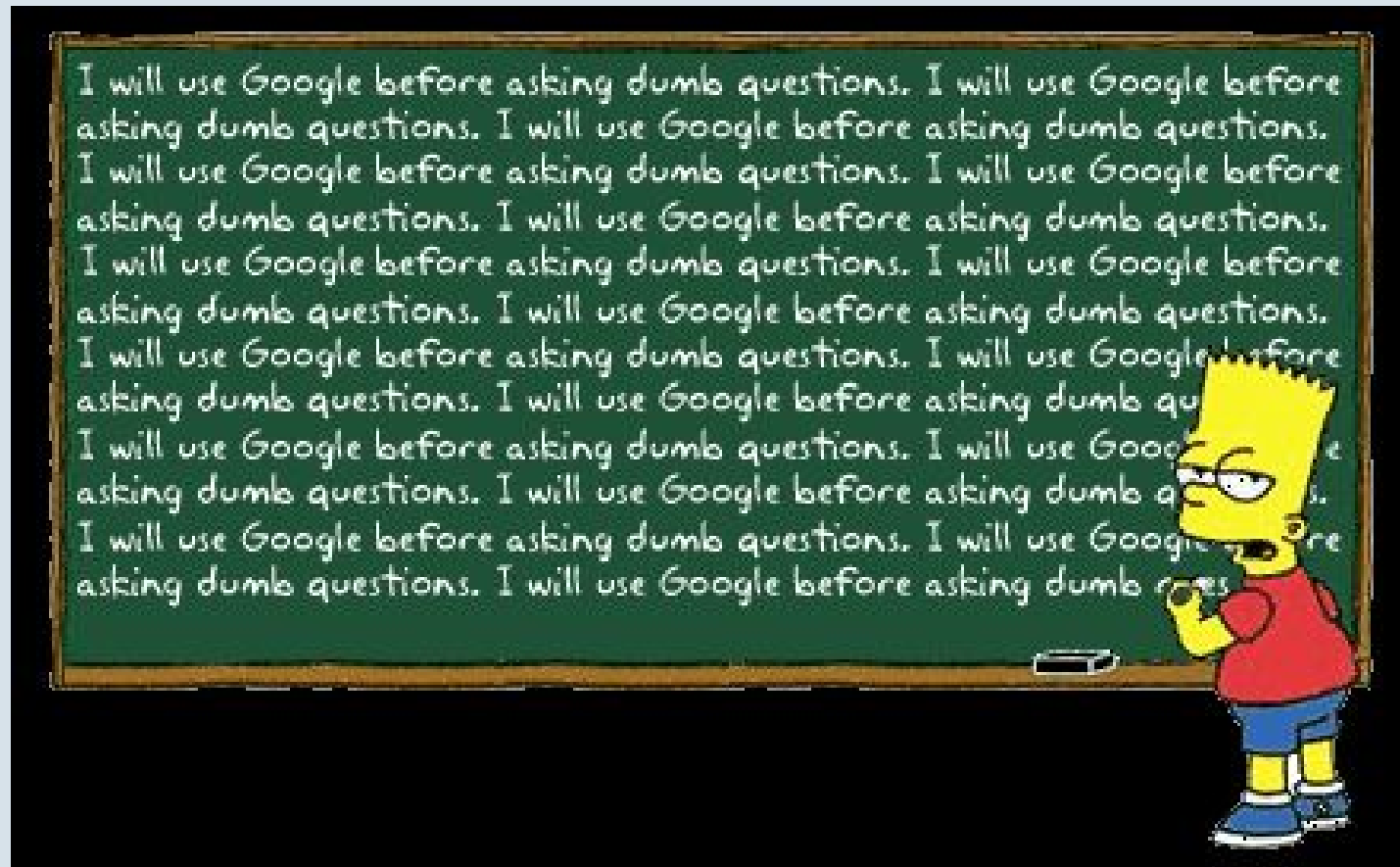
- Neste capítulo, vamos estudar como definir em Java programas que executam repetidamente um conjunto de acções até se verificar uma dada condição.



- No caminho, será introduzida:
 - A instrução `while`

O Ecoador

O Bart está de castigo



O Ecoador

- Objectivo
 - Simular um “eco”.
- Descrição
 - Um “eco” é objecto que permite ecoar uma dada sequência de carateres.
- Funcionalidades
 - É sempre possível ecoar uma sequência de caracteres n vezes, sendo n um valor inteiro não negativo.
 - Deve ser possível saber o número de ecos realizados até ao momento.
 - Quando o eco é criado, o número de ecos realizados é zero.
- Interacção com o utilizador
 - Após criar um eco, pode invocar as operações.

O Ecoador

- Interface (classe Echo):

public String echo(String data, int rep)

Recebe a String data e um número inteiro rep, e devolve a concatenação de rep cópias da String data.

pre: data != null && rep >= 0

public int countEchoes()

Devolve o total de ecos realizados.

O Ecoador

```
Echo e = new Echo();  
e.echo("p-p-poker face!", 2)  
"p-p-poker face!p-p-poker face!" (String)  
e.echo("mum ", 4)  
"mum mum mum mum " (String)  
e.countEchoes()  
6 (int)
```

Como repetir rep vezes a String $data$?

- Sabendo à partida o máximo de repetições, podem-se encadear alternativas:
 - Exº: $rep \leq 3$

```
//Pre: data != null && rep >= 0 && rep <= 3
```

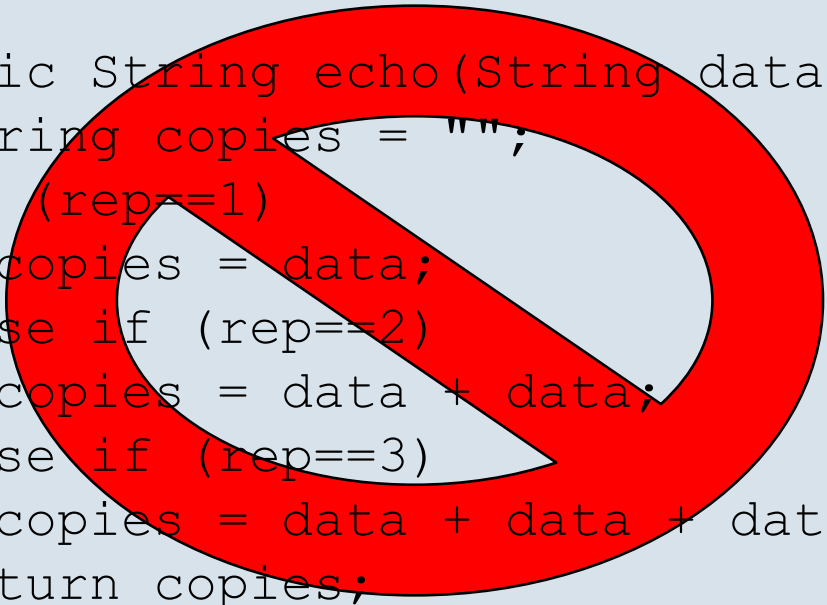
```
public String echo(String data, int rep) {  
    String copies = "";  
    if (rep==1)  
        copies = data;  
    else if (rep==2)  
        copies = data + data;  
    else if (rep==3)  
        copies = data + data + data;  
    return copies;  
}
```

Como repetir `rep` vezes a String `data`?

- Sabendo à partida o máximo de repetições, podem-se encadear alternativas:
 - Exº: `rep <= 3`

```
//Pre: data != null && rep >= 0 && rep <= 3
```

```
public String echo(String data, int rep) {  
    String copies = "";  
    if (rep==1)  
        copies = data;  
    else if (rep==2)  
        copies = data + data;  
    else if (rep==3)  
        copies = data + data + data;  
    return copies;  
}
```

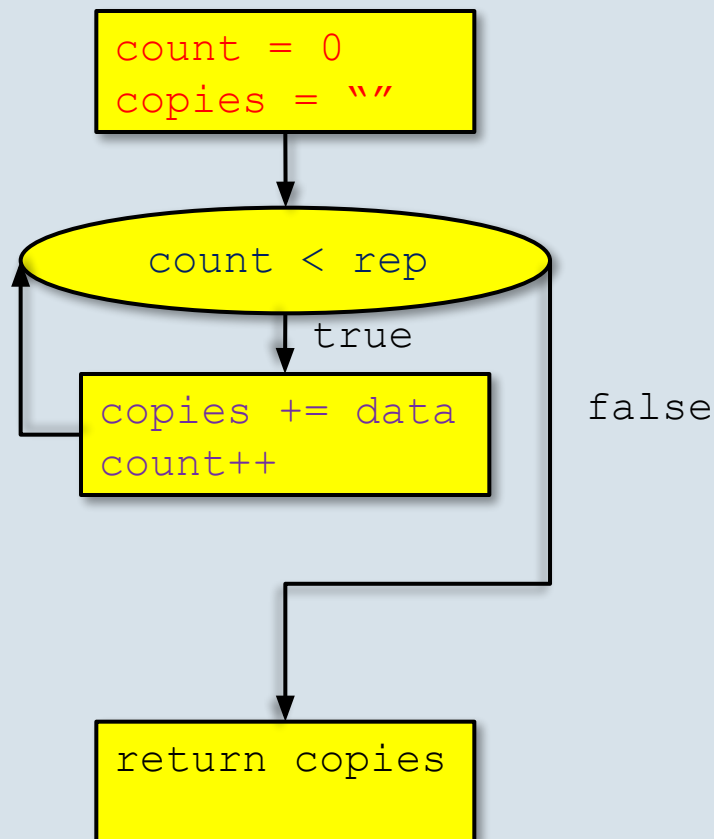


Esta “solução” é péssima!

- Se o máximo for grande, ficamos com uma **sequência enorme de ifs**. ☹
- Ficamos limitados a um máximo completamente artificial – a **solução não é genérica**! ☹

Portanto, NÃO SERVE!

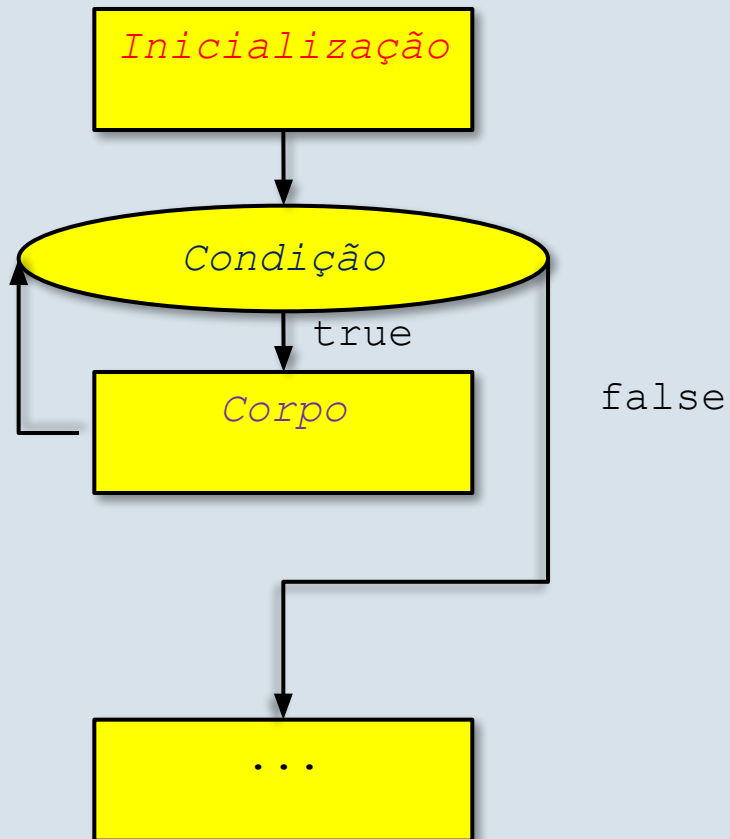
Como repetir `rep` vezes a String `data`?



data	rep	copies	count
"ha!"	3	""	0
"ha!"	3	"ha!"	1
"ha!"	3	"ha!ha!"	2
"ha!"	3	"ha!ha!ha!"	3

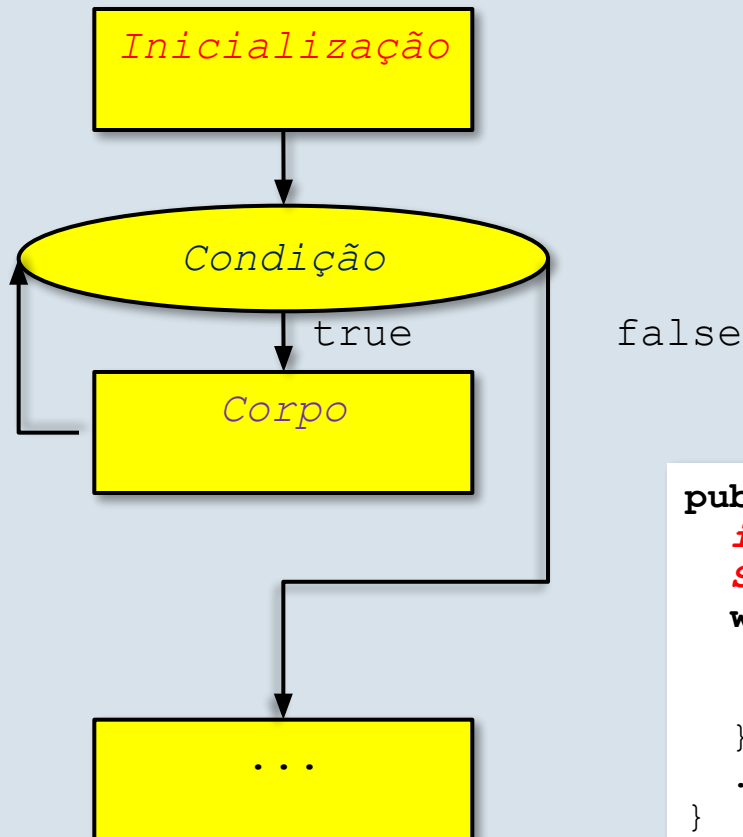
Nota: como `count` é um acumulador aditivo, é iniciado com o elemento neutro da adição

O ciclo while



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

O ciclo while



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

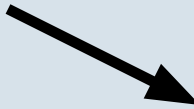
```
public String echo(String data, int rep) {  
    int count = 0;  
    String copies = "";  
    while (count < rep) {  
        copies += data;  
        count = ++;  
    }  
    ...  
}
```

Programando a classe Echo

```
public class Echo {  
  
    ...  
  
    //Pre: data != null && rep >= 0  
    public String echo(String data, int rep) {  
        int count = 0;           // contador de copias ja repetidas  
        String copies = "";      // String com copias ja repetidas  
        while (count < rep) {  
            copies += data;  
            count++;  
        }  
        ...  
        return copies;  
    }  
    ...  
}
```

Programando a classe Echo

```
public class Echo {  
  
    ...  
  
    //Pre: data != null && rep >= 0  
    public String echo(String data, int rep) {  
        int count = 0;           // contador de copias ja repetidas  
        String copies = "";      // String com copias ja repetidas  
        while (count < rep) {  
            copies += data;  
            count++;  
        }  
        ...  
        return copies;  
    }  
    ...  
}
```



Ciclo “while”

Programando a classe Echo

```
public class Echo {  
  
    ...  
  
    //Pre: data != null && rep >= 0  
    public String echo(String data, int rep) {  
        int count = 0;           // contador de copias ja repetidas  
        String copies = "";      // String com copias ja repetidas  
        while (count < rep) {  
            copies += data;  
            count++;  
        }  
        ...  
        return copies;  
    }  
    ...  
}
```



Condição do ciclo

Programando a classe Echo

```
public class Echo {  
  
    ...  
  
    //Pre: data != null && rep >= 0  
    public String echo(String data, int rep) {  
        int count = 0;           // contador de copias ja repetidas  
        String copies = "";      // String com copias ja repetidas  
        while (count < rep) {  
            copies += data;  
            count++;  
        }  
        ...  
        return copies;  
    }  
    ...  
}
```



Corpo do ciclo

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.

Programando a classe Echo

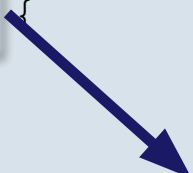
```
public class Echo {  
  
    ...  
  
    //Pre: data != null && rep >= 0  
    public String echo(String data, int rep) {  
        int count = 0;           // contador de copias ja repetidas  
        String copies = "";      // String com copias ja repetidas  
        while (count < rep) {  
            copies += data;  
            count++;  
        }  
        ...  
        return copies;  
    }  
    ...  
}
```



Inicialização

Programando a classe Echo

```
public class Echo {  
  
    ...  
  
    //Pre: data != null && rep >= 0  
    public String echo(String data, int rep) {  
        int count = 0;           // contador de copias ja repetidas  
        String copies = "";      // String com copias ja repetidas  
        while (count < rep) {  
            copies += data;  
            count++;  
        }  
        ...  
        return copies;  
    }  
  
    ...  
}
```



Condição do Ciclo
Quando o valor da variável `count` for igual ao de `rep`, o ciclo termina

Programando a classe Echo

```
public class Echo {  
  
    ...  
  
    //Pre: data != null && rep >= 0  
    public String echo(String data, int rep) {  
        int count = 0;           // contador de copias ja repetidas  
        String copies = "";      // String com copias ja repetidas  
        while (count < rep) {  
            copies += data;  
            count++;  
        }  
        ...  
        return copies;  
    }  
    ...  
}
```



Corpo do Ciclo

Altera o valor da variável `count`

Programando a classe Echo

```
public class Echo {  
  
    ...  
  
    //Pre: data != null && rep >= 0  
    public String echo(String data, int rep) {  
        int count = 0;           // contador de copias ja repetidas  
        String copies = "";      // String com copias ja repetidas  
        while (count < rep) {  
            copies += data; // accao  
            count++;        // progresso  
        }  
        ...  
        return copies;  
    }  
    ...  
}
```

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.

Programando a classe Echo

```
public class Echo {  
    private int echoCounter; // Total de ecos realizados  
    public Echo() { echoCounter = 0; }  
    //Pre: data != null && rep >= 0  
    public String echo(String data, int rep) {  
        int count = 0; // contador de copias ja repetidas  
        String copies = ""; // String com cópias ja repetidas  
        while (count < rep) {  
            copies += data;  
            count++;  
        }  
        echoCounter = echoCounter+rep;  
        return copies;  
    }  
    public int countEchoes() { return echoCounter; }  
}
```

Faltava apenas acrescentar o contador de repetições

A classe Echo

```
public class Echo {  
    private int echoCounter; // Total de ecos realizados  
    public Echo() { echoCounter = 0; }  
    //Pre: data != null && rep >= 0  
    public String echo(String data, int rep) {  
        int count = 0; // contador de copias já repetidas  
        String copies = ""; // String com cópias já repetidas  
        while (count < rep) {  
            copies += data;  
            count++;  
        }  
        echoCounter = echoCounter+rep;  
        return copies;  
    }  
    public int countEchoes() { return echoCounter; }  
}
```

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*. 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).

Recorde a 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

(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

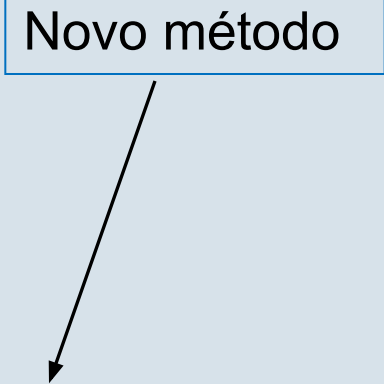
public int computeInterest()

Calcula qual o valor do juro anual a aplicar

public void applyInterest()

Credita o juro anual ao saldo da conta

Novo método



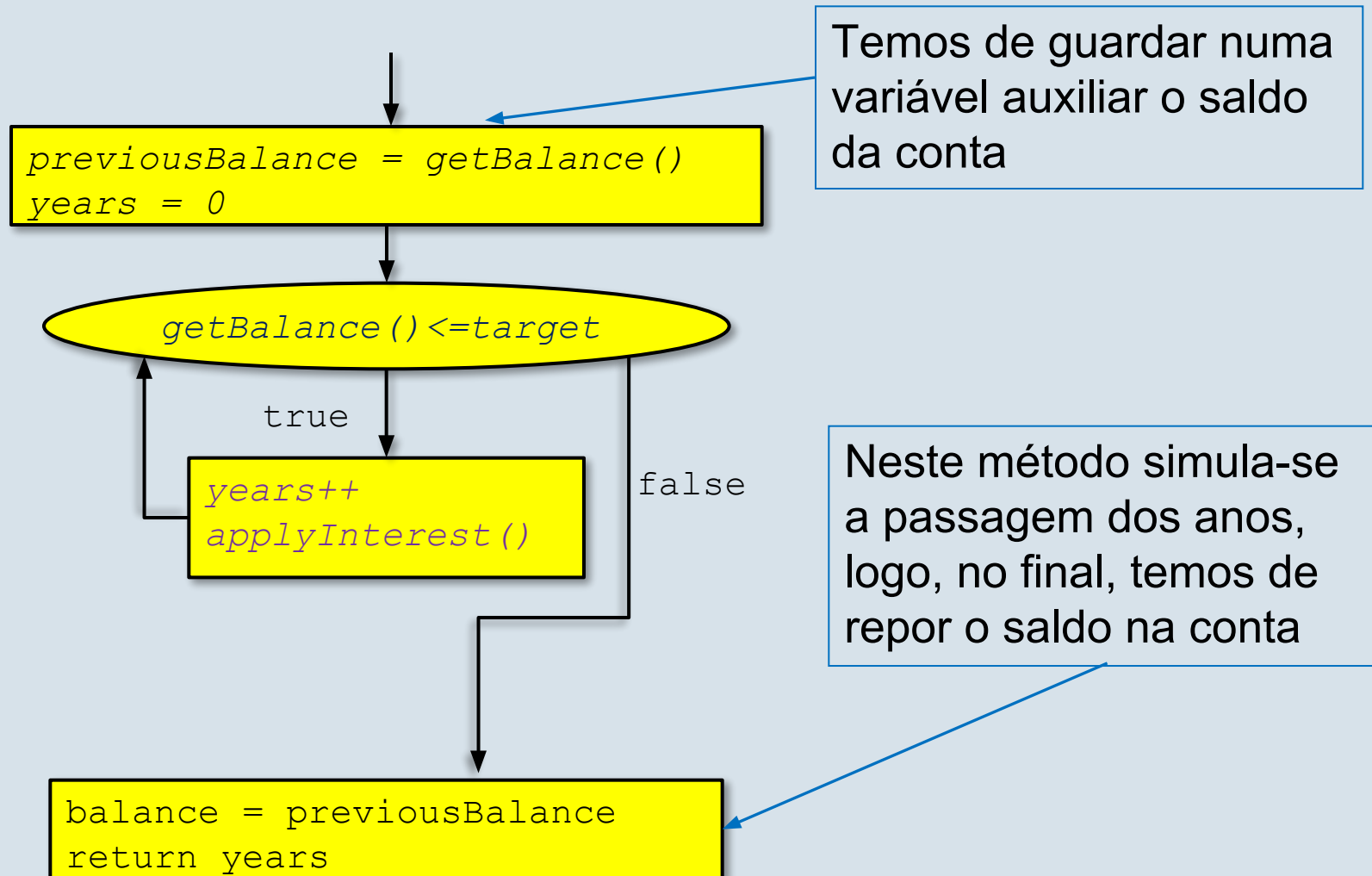
public int howManySavingsYears(**int** target)

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

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

Poupança (versão 1)

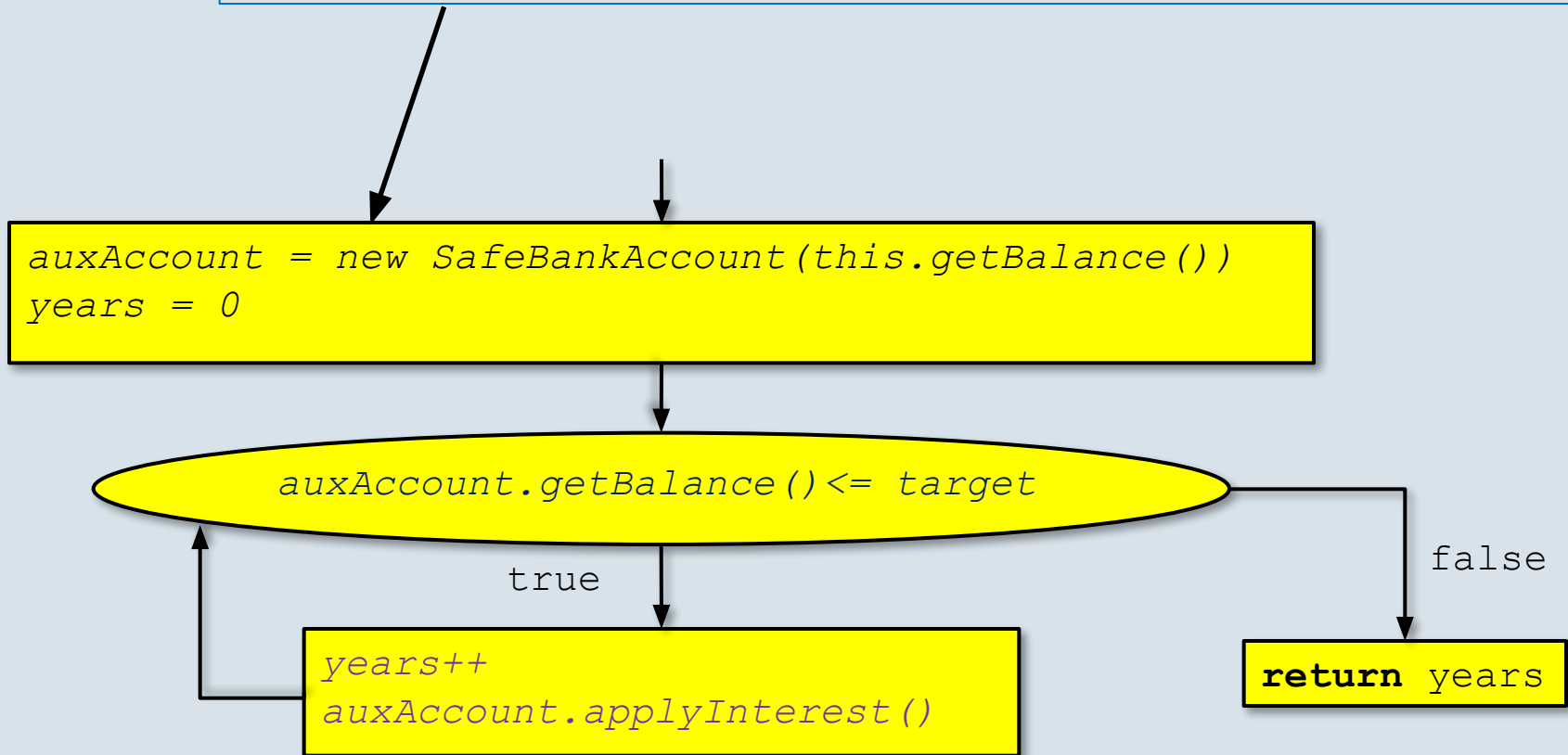
```
int howManySavingsYears(int target)
```



Poupança (versão 2)

```
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

(nova interacção com o utilizador)

- Interacção com o utilizador
 - Pede ao utilizador o saldo inicial duma conta bancária.
 - Enquanto o utilizador não escrever o comando “S”, o programa deve processar os comandos dados pelo utilizador. Os comandos permitidos são:

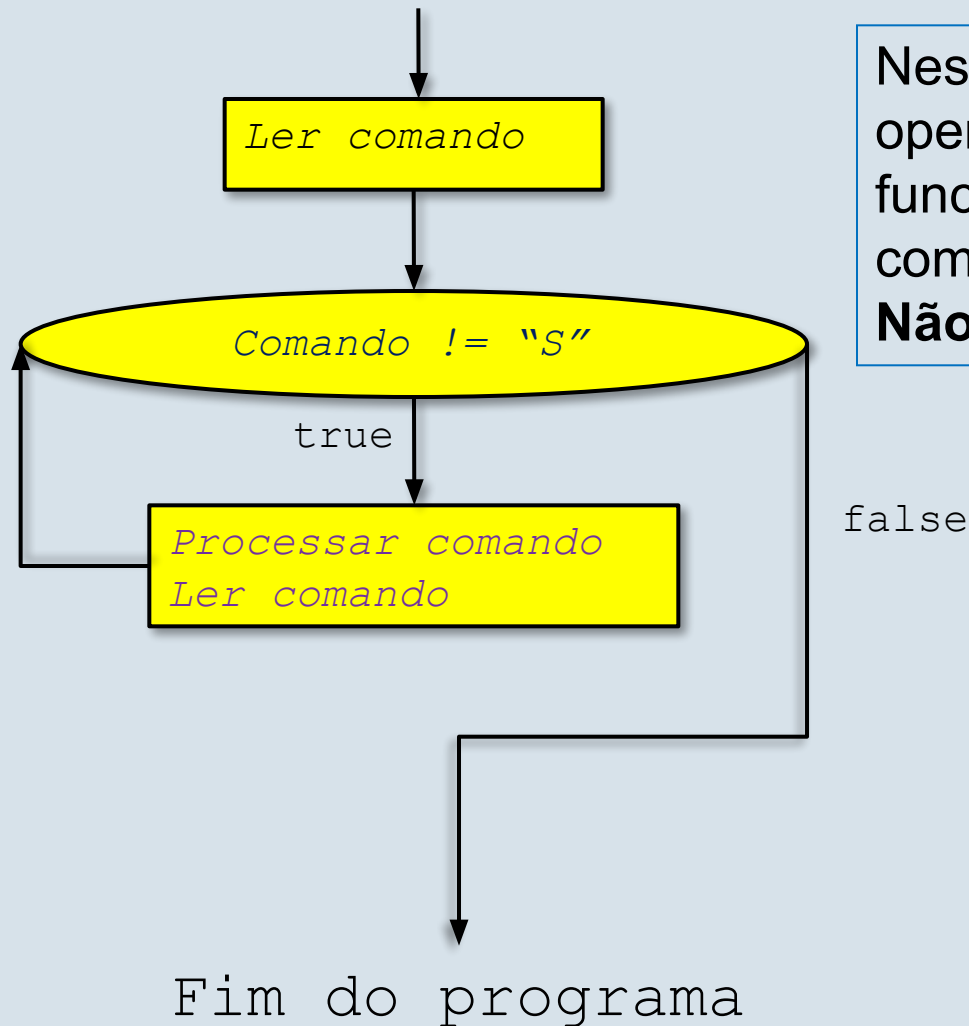
```
L <montante_levantar> → levantar  
D <montante_depositar > → depositar  
CS → consultar saldo  
CTA → consultar taxa anual  
ATA → aplicar taxa anual  
QAP <montante_alvo > → anos de poupança  
S → sair
```

- No processamento do comando, deve escrever na consola se a acção foi bem sucedida ou não, e a informação pedida, caso exista.

Interpretador de comandos

Conta Bancária Segura

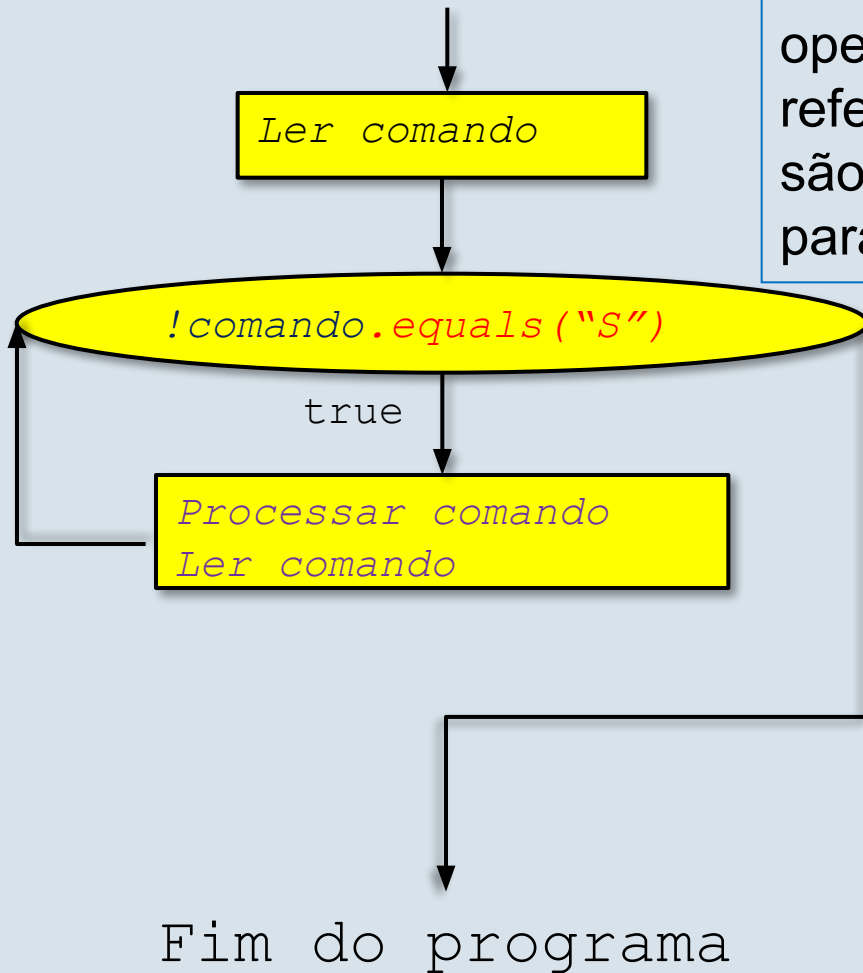
(nova interacção com o utilizador)



Neste exemplo, será que os operadores “==” ou “!=” funcionam como queremos com o tipo String?
Não.

Conta Bancária Segura

(nova interacção com o utilizador)



A `String` em Java é uma classe, logo os operadores `==` e `!=` verificam se as referências dos objectos que representam são iguais ou não, ou seja são referências para **exactamente o mesmo objecto**.

Em Java para testar se duas sequências de caracteres são semelhantes usa-se o método **`equals`** da classe `String`.

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`.

