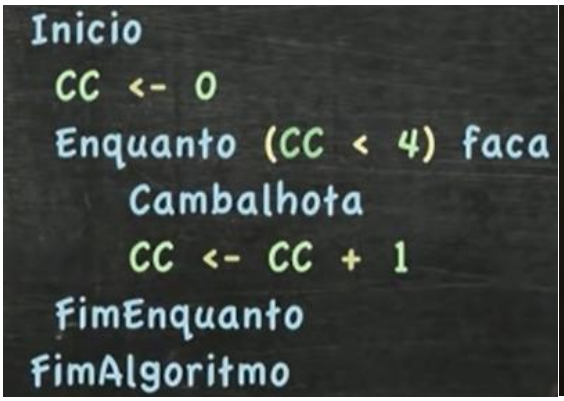
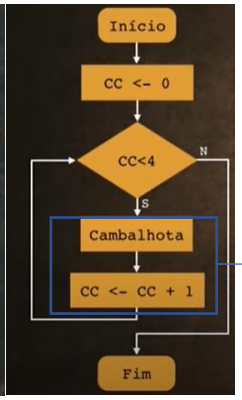
# AULA 11 – ESTRUTURAS DE REPETIÇÃO (PARTE 1)

## REPETIÇÃO COM TESTE NO INÍCIO

### - Pseudocódigo e fluxograma:

Exemplo das cambalhotas:



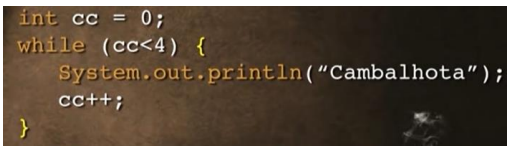

Esse é o **bloco** que vai ser executado, porém o **teste lógico** (CC<4) será feito antes dele! (por isso é chamado repetição com teste no início

OBS: o losango indica **teste lógico** (teste lógico do enquanto)

### - Em Java:

Lembrando: posso usar **operadores de incremento automático**!

Ou seja: cc ← cc + 1 é a mesma coisa que cc++



```
int cc = 0;
while (cc<4) {
   System.out.println("Cambalhota");
   cc++;
}
```

### - No NetBeans:

```
public static void main(String[] args) {
    // TODO code application logic here
    int cc = 0;
    while (cc<4){
        System.out.println("Cambalhota");
        cc++;
    }
}
```

```
run:
Cambalhota
Cambalhota
Cambalhota
Cambalhota
CONSTRUÍDO COM SUCESSO (tempo total: 1 segundo)
```

- Para mostrar o número da cambalhota que está acontecendo:

```
public static void main(String[] args) {
    // TODO code application logic here
    int cc = 0;
    while (cc<4){
        System.out.println("Cambalhota " + cc);
        cc++;
    }
}
```

```
run:
Cambalhota 0
Cambalhota 1
Cambalhota 2
Cambalhota 3
CONSTRUÍDO COM SUCESSO (tempo total: 1 segundo)
```

OBS: para começar o contador da cambalhota 1, há 2 maneiras

```
public static void main(String[] args) {
    // TODO code application logic here
    int cc = 0;
    while (cc<4){
        System.out.println("Cambalhota " + (cc+1));
        cc++;
    }
}
```

```
public static void main(String[] args) {
    // TODO code application logic here
    int cc = 1;
    while (cc<=4){
        System.out.println("Cambalhota " + cc);
        cc++;
    }
}
```

```
run:
Cambalhota 1
Cambalhota 2
Cambalhota 3
Cambalhota 4
CONSTRUÍDO COM SUCESSO (tempo total: 1 segundo)
```

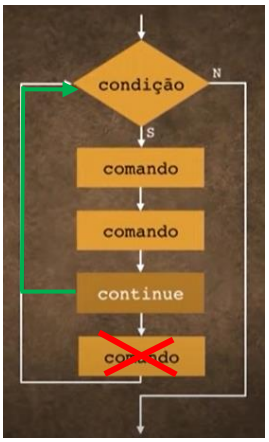
## Mudando o fluxo de um laço:

- É mudar a ordem natural da repetição

Comando: **continue**

- É o primeiro comando para mudar o fluxo de um laço

- Toda vez que tiver um **continue**, ele vai ignorar o comando que está embaixo e voltar para a condição




**No NetBeans:**

```
public static void main(String[] args) {
    // TODO code application logic here
    int cc = 0;
    while (cc<10){
        cc++;
        if (cc == 5 || cc == 7){
            continue;
        }
        System.out.println("Cambalhota " + cc);
    }
}
```

```
run:
Cambalhota 1
Cambalhota 2
Cambalhota 3
Cambalhota 4
Cambalhota 6
Cambalhota 8
Cambalhota 9
Cambalhota 10
```

Nesse caso, o programa não vai mostrar na tela a cambalhota 5 e 7

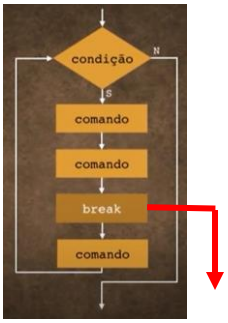
OBS: além do **continue**, eu tenho outro comando: o **break**

- É o comando de interrupção do laço, interrompendo a execução mesmo não tendo saído pela cláusula n.

- Funciona de modo contrário:

Continue – joga para cima do laço

Break – joga para fora do laço




**- No NetBeans:**

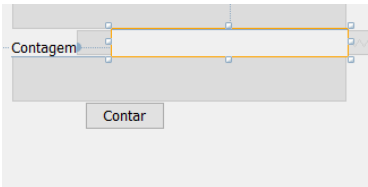
```
public static void main(String[] args) {
    // TODO code application logic here
    int cc = 0;
    while (cc<10){
        cc++;
        if (cc == 2 || cc == 3 || cc == 4){
            continue;
        }
        if (cc == 7){
            break;
        }
        System.out.println("Cambalhota " + cc);
    }
}
```

```
run:
Cambalhota 1
Cambalhota 5
Cambalhota 6
```

Mandei ocultar o 2, 3 e 4 **(continue)** e quando chegasse ao 7, pararia a execução **(break)**

## - Utilizando o Swing:

### Rascunho:




### Variáveis:

| Label: | Botão: |
|---|---|
| lblContagem | btnContar |

### Código-fonte:

```
private void btnContarActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    int cc = 0;
    String contagem = "";
    while (cc<5){
        contagem += cc + " ";
        cc++;
    }
    lblContagem.setText(contagem);
}
```

### Execução: