Estruturas de repetição (*loops*) – Parte 2

Prof. Bruno Nogueira

Loops em pseudocódigo

while

```
ENQUANTO condição FAÇA
    comando;
```

do-while

```
FAÇA
    comando;
ENQUANTO condição;
```

for

```
PARA cont = valorInicial ATÉ valorFinal FAÇA
    comando;
```

Programando com loops

- Um loop envolve três elementos
  - A inicialização dos elementos que precisam vir antes da repetição
  - O corpo do loop
  - Um mecanismo de parada do loop

- Corpo do loop
  - Constituído pela parte que repete
  - Deve ser bem pensado

Programando com loops

- Comando de inicialização
  - Um loop depende de um valor inicial correto
  - Por exemplo, se usarmos um acumulador para soma, seu valor inicial deve ser zero. Por outro lado, se o acumulador for para multiplicação, deve ser um.

```
int cont = 0;
while(cont < 5)
    cont++;
```

```
int cont = 1;
while(cont < 32)
    cont = cont * 2;
```

```
for(int cont = 0; cont < 5; cont++)
    System.out.println(cont);
```

Controlando o número de iterações

- Existem algumas maneiras de encerrar o loop
  - **Contador:** quando sabemos quantas vezes teremos que repetir o loop

```
int cont = 0;
while(cont < 5)
    cont++;
```

  - **Perguntar quando encerrar:** algumas vezes, podemos perguntar ao usuário quando encerrar nosso programa

```
do
{
    System.out.println("Digite o preço:");
    somaPreços += teclado.nextDouble();
    System.out.println("Existe mais algum preço? (sim/não)");
    resposta = teclado.next();
} while (resposta.equalsIgnoreCase("sim"));
```

Controlando o número de iterações

- **Valor sentinela:** algumas vezes, escolhemos um valor da entrada do usuário como "sentinela", com um significado especial de fim de loop

```
System.out.println("Entre as notas dos estudantes.");
System.out.println("Após digitar todas, entre um número negativo.");

nota = teclado.nextDouble();
while(nota >= 0)
{
    somaNotas += nota;
    numeroEstudantes ++;
    nota = teclado.nextDouble();
}
```

Controlando o número de iterações

- **Variável booleana**: uma maneira elegante e que facilita a leitura é usar uma variável booleana para controlar o loop. Damos o nome de ***flag*** a essa variável.

```
boolean haMaisAlunos = true;
System.out.println("Entre as notas dos estudantes.");
System.out.println("Após digitar todas, entre um número negativo.");

while(haMaisAlunos)
{
    nota = teclado.nextDouble();
    if(nota < 0)
        haMaisAlunos = false;
    else
    {
        somaNotas += nota;
        numeroEstudantes ++;
    }
}
```

Controlando o número de iterações

- Valor inicial da variável booleana tem que ser adequado ao teste lógico que queremos

```
boolean haMaisAlunos = true;
System.out.println("Entre as notas dos estudantes.");
System.out.println("Após digitar todas, entre um número negativo.");

while(haMaisAlunos)
{
    nota = teclado.nextDouble();
    if(nota < 0)
        haMaisAlunos = false;
    else
    {
        somaNotas += nota;
        numeroEstudantes ++;
    }
}
```

Erros comuns em loops

- Os erros mais comuns são o **loop infinito** e o **"errar por um"**
  - Já vimos loops infinitos anteriormente. São causados por falhas lógicas

```
count = 0;
while (saldo < 0)
{
    saldo = saldo – multa;
    saldo = saldo + deposito;
    count ++;
}
System.out.println("Sua conta sairá do vermelho em " + count + "meses.");
```

Erros comuns em loops

- Uma possível solução seria testar se o depósito não é menor que a multa antes de entrar no loop

```
if(deposito <= multa)
    System.out.println(“Depósitos muito pequenos. Nunca sairá do vermelho!”);
else
{
    count = 0;
    while (saldo < 0)
    {
        saldo = saldo – multa;
        saldo = saldo + deposito;
        count ++;
    }
    System.out.println(“Sua conta sairá do vermelho em ” + count + “meses.”);
}
```

Erros comuns em loops

- Já o "erro por um" vem de erros na expressão booleana de controle
  - Por exemplo, se usarmos < ao invés de <=, o loop será repetido uma vez a menos

```
int cont = 0;
while(cont < 5)
{
    System.out.println(cont);
    cont++;
}
```

```
int cont = 0;
while(cont <= 5)
{
    System.out.println(cont);
    cont++;
}
```

Erros comuns em loops

- Outro "erro por um" comum vem do uso do comparador == para igualdades
  - Funcionam bem com inteiros e caracteres
  - Para ponto flutuante, não são confiáveis. Ponto flutuante são quantidades aproximadas
  - Opte por comparar com <= ou >=
- Geralmente, erros por um passam despercebidos
  - Faça testes que verifiquem seu funcionamento correto!

Exercício

- Números primos são aqueles divisíveis apenas por 1 e por ele mesmo. Faça um programa que, dado um número n, fornecido pelo usuário, determine se ele é primo ou não.

Exercício

- Palíndromos são palavras ou frases que são idênticas quando lidas de frente para trás e de trás para frente (ex: “ana”, “socorram-me, subi no onibus em marrocos”, “amor roma”. Faça um programa que peça ao usuário para digitar um palíndromo e o valide. Espaços e pontuação devem ser desprezados.