Programação I

Aula 4

Professor: Diogo Passos Ranghetti

Sumário

Comandos ***break*** e ***continue***

Quanto maior o problema a ser resolvido mais complexo fica á lógica utilizada para resolvê-lo. Como vimos na aula anterior o Java dispõe de três tipos de estruturas de controle e repetição, sendo eles o ***while*** o ***do-while*** e o ***for***, e estes comandos são essenciais para resolver a complexidade dos problemas a serem resolvidos.

Cada uma dessas estruturas de repetições, ***while***, ***do-while*** e o ***for***, possuem características distintas e isso determina o momento ideal para serem aplicadas, e para melhorar e aperfeiçoar essas estruturas de controle e repetição o Java dispõe de uma serie de comandos e recursos, dentre eles abordaremos nesta aula os comandos ***break*** e ***continue***.

Imagine que durante a repetição de uma estrutura de código exista a necessidade de abortar a estrutura de repetição, ou que o objetivo da estrutura de repetição tenha sido atingido antes que todo o laço de repetição tenha acabado. Diante desta situação, o que podemos fazer?

Para tal situação podemos usar os comandos ***break*** ou ***continue*** onde cada um resolveria esta situação de acordo com suas particularidades.

Nesta aula veremos a funcionalidade de cada um desses comandos através de exemplos práticos, quando e como aplicar o ***break*** ou o ***continue***.

break

O comando ***break*** tem por finalidade quebrar, parar, interromper uma estrutura de controle, sendo elas ***while***, ***do-while***, ***for*** ou ***switch-case***, no momento em que o comando ***break*** é acionado o processo da estrutura de controle é finalizado.

Para nosso primeiro exemplo vamos analisar sua funcionalidade aplicada à declaração ***switch-case***, nesta estrutura de controle o comando ***break*** funciona como um delimitador e deve ser aplicado quando o argumento utilizado for um numero.

Por exemplo:

```
public class JavaApplication1 {

    public static void main(String[] args) {

        //o argumento é um numero
        int teste = 4;
        switch (teste) {
            case 1:
                System.out.println("1 - " + teste);
            case 2:
                System.out.println("2 - " + teste);
            case 3:
                System.out.println("3 - " + teste);
                break;
            case 4:
                System.out.println("4 - " + teste);
            default:
                System.out.println("nenhuma opção " + teste);
        }
    }
}
```

```
- JavaApplication1 (run)
run:
4 - 4
nenhuma opção 4
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

Note que o nosso programa imprimiu dois resultados, isso ocorre porque a clausula ***case*** não encontrou um fim, ou seja, um ***break***, imprimindo assim o conteúdo que está na clausula ***default***. Se o argumento for "teste = 1" o nosso programa vai imprimir até encontrar o comando ***break*** podemos ver isso na imagem a seguir.

```
public class JavaApplication1 {

    public static void main(String[] args) {

        //o argumento é um numero
        int teste = 1;
        switch (teste) {
            case 1:
                System.out.println("1 - " + teste);
            case 2:
                System.out.println("2 - " + teste);
            case 3:
                System.out.println("3 - " + teste);
                break;
            case 4:
                System.out.println("4 - " + teste);
            default:
                System.out.println("nenhuma opção " + teste);
        }
    }
}
```

```
- JavaApplication1 (run)
run:
1 - 1
2 - 1
3 - 1
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

Agora vamos aplicar o comando ***break*** a uma estrutura de repetição ***for***, este tipo de exemplo é bastante utilizado para estruturas em que ao ter atingido o resultado esperado ao realizar uma busca em uma lista de valores a estrutura de repetição seja encerrada.

Imagine que tenhamos um resultado de mil registros e precisamos encontrar um determinado registro, e ao encontrarmos esse registro a estrutura de repetição que faz a busca seja encerrada, já que não faz mais sentido continuar o laço.

```
public class JavaApplication1 {

    public static void main(String[] args) {
        for (int i = 0; i < 1000; i++) {
            System.out.println("Registro.......:" + i);
            if (i == 5) {
                System.out.println("Registro encontrado! ... " + i);
                break;
            }
        }

    }
}
```

```
da - JavaApplication1 (run)
run:
Registro.......:0
Registro.......:1
Registro.......:2
Registro.......:3
Registro.......:4
Registro.......:5
Registro encontrado! ... 5
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

O comando ***break*** pode ser aplicado da mesma forma nas estruturas de repetição ***do-while*** e ***while***.

continue

O comando ***continue*** funciona de forma parecida com o comando break tendo por finalidade ser aplicado em uma estrutura de controle, diferente do comando ***break*** o comando ***continue*** não funciona para a declaração ***switch-case***, funcionando apenas para as declarações ***while***, ***do-while***, ***for***.

Ao ser acionado o comando ***continue*** muda o curso do processamento da estrutura de controle, reiniciando o processo, ou seja, inicia o loop novamente ignorando o restante do código que tem abaixo do comando ***continue***.

```
public class JavaApplication1 {

    public static void main(String[] args) {
        for (int i = 0; i < 1000; i++) {
            if (i < 5) {
                System.out.println("Registro Invalido! ... " + i);
                continue;
            } else if (i == 5) {
                System.out.println("Registro encontrado! ... " + i);
                break;
            }
            System.out.println("Não chega até aqui... ");
        }
    }
}
```

```
- JavaApplication1 (run)
run:
Registro Invalido! ... 0
Registro Invalido! ... 1
Registro Invalido! ... 2
Registro Invalido! ... 3
Registro Invalido! ... 4
Registro encontrado! ... 5
CONSTRUÍDO COM SUCESSO (tempo total: 1 segundo)
```

Neste exemplo, note que não imprime o terceiro comando ***println*** isso ocorre devido aos comandos ***continue*** e ***break***.

Vetor (*Array*)

De acordo com a definição mais clássica da informática, um vetor é uma estrutura de dados homogenia, resumindo, todos os elementos de um vetor são do mesmo tipo.

A estrutura básica de um vetor é representada por seu nome e um índice, que será utilizado toda a vez em que precisar acessar um determinado elemento dentro do vetor. É importante ressaltar que em Java todo o vetor possui um tamanho fixo, ou seja, não é possível redimensionar um vetor ou adicionar a ele mais elementos do que este pode suportar.

Em Java ao instanciar um vetor, também chamado de *array*, determinamos seu tamanho e tipo, e ao informar o tamanho é alocada a quantidade de memória necessária para esse tamanho, cada uma das partes que compõe o vetor possui um índex que determina a posição em que a informação é alocada dentro do vetor, sendo a posição inicial definida pelo valor zero, e as demais segue a sequência numérica.

```
public class JavaApplication1 {

    public static void main(String[] args) {
        int[] vetorInt = new int[10];
        String[] vetorStr =  new String[10];

        vetorInt[0] = 10;
        vetorStr[0] = "A";

         System.out.println(vetorInt[0]);
         System.out.println(vetorStr[0]);
    }
}
```

```
- JavaApplication1 (run)
run:
10
A
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```

Conforme podemos ver no exemplo logo acima para acessar os dados contidos em um vetor ou *array* basta indicar a posição em que a informação está contida que é representada pelo valor do índex, para adicionar um valor em uma das posições do vetor a lógica é a mesma, basta indicar em qual posição será adicionado informando o índex.

Referências bibliográficas

DEITEL, H. M.; DEITEL, P. J. Java como Programar. 8ª.ed. São Paulo: Pearson Brasil, 2010.