Unidade 2: Estruturas de Controle

Parte 1 - Scanner e Estrutura IF

Prof. Daniel Caetano

Objetivo: Apresentar a classe Scanner e as principais estruturas de controle da linguagem Java.

Bibliografia: DEITEL, 2005; CAELUM, 2010; HOFF, 1996.

INTRODUÇÃO

Na primeira unidade a Linugagem Java foi apresentada, juntamente com os conceitos de variáveis e operadores. Estes são conceitos básicos e muito úteis, mas de pouca aplicabilidade se não formos capazes de receber dados digitados pelo usuário.

Os dados digitados pelo usuário podem precisar de algumas verificações antes de serem utilizados e, neste caso, entram em cena as estruturas de controle, que nos permitirão escolher qual parte de um programa será executada de acordo com o valor digitado pelo usuário.

Estes serão os tópicos apresentados nesta aula.

1. LENDO DADOS DIGITADOS PELO USUÁRIO

Na aula anterior, vimos um programa muito simples, reproduzido abaixo, que simplesmente imprimia uma mensagem na tela.

UmPrograma.java

```
class UmPrograma {
   public static void main(String[] args) {
      System.out.println("Ola, mundo!");
      }
   }
```

Vimos que **System.out** era uma das classes do Java, e que **println** é um de seus métodos, que serve para imprimir um texto na tela.

Nesta primeira parte da aula, vamos aprender como receber dados digitados pelo usuário, o que exigirá alguma preparação de nosso código, sendo a primeira delas a necessidade de existir uma variável para receber o valor digitado, como é indicado no código a seguir.

UmPrograma.java

```
class UmPrograma {
   public static void main(String[] args) {
      int valor;

      System.out.println("Ola, mundo!");
      }
   }
```

Infelizmente, a leitura de valores digitados pelo usuário é um processo um pouco mais complexo, não bastando usar a classe **System.in**, embora ela faça parte do processo, já que **System.in** é uma classe que apresenta métodos relativos ao teclado.

Tanto a classe **System.out** quanto a classe **System.in** são classes do pacote básico e são prontas para uso (o Java tem várias delas). No caso da leitura de teclado, não temos uma classe pronta para isso; temos apenas uma classe que serve, genericamente, para a leitura de dispositivos, a classe **Scanner**.

Como a classe Scanner não faz parte do pacote principal, faz parte do pacote ***java.util***, precisamos indicar ao Java que iremos utilizar esse pacote, o que pode ser feito através da instrução **import**, como indicado no código a seguir:

UmPrograma.java

```
import java.util.*;

class UmPrograma {
   public static void main(String[] args) {
      int valor;

      System.out.println("Ola, mundo!");
      }
   }
```

Se não fizermos isso e tentarmos usar a classe Scanner, o Java reportará um erro e não teremos um programa funcional.

> Para saber em qual pacote uma classe está e também quais são seus métodos, basta consultar o site de referência da Sun Microsystems: http://java.sun.com/j2se/1.5.0/docs/api/

Agora já podemos usar a classe Scanner. Entretanto, como a classe Scanner não é específica para ler o teclado, precisaremos solicitar que o Java crie uma **instância** desta classe, adaptada para a leitura do teclado. Essa **instância** é chamada **objeto**.

Mais adiante no curso estudaremos com maior detalhe o que são classes e objetos; por hora, a definição dada é considerada suficiente.

Para gerar um objeto, precisamos primeiro definir um **nome** para ele. Isso pode ser feito da mesma maneira com que se define uma variável:

```
NomeDaClasse   nomeDoObjeto;
```

Por exemplo, podemos criar um objeto a partir da classe **Scanner** e chamá-lo de **teclado**.

```
Scanner   teclado;
```

No código isso fica da seguinte forma:

UmPrograma.java

```
import java.util.*;

class UmPrograma {
   public static void main(String[] args) {
      int valor;
      Scanner teclado;

      System.out.println("Ola, mundo!");
      }
   }
```

A questão é que, ao contrário do que ocorre com variáveis, o Java não separa espaço de memória para objetos de maneira automática. Assim, depois que demos um nome ao objeto, precisamos pedir que o Java **aloque a memória** e **construa** este objeto para nós. Isso é feito através da instrução **new** (novo). A instrução **new** tem a seguinte "cara":

```
variavelObjeto = new NomeDaClasse(parâmetros);
```

Os parâmetros, no caso, definem as configurações específicas que queremos para o objeto criado. No nosso caso, por exemplo, queremos um objeto da **classe Scanner**, associado ao teclado, que é representado pela classe **System.in**. Isso pode ser indicado da seguinte forma:

```
teclado = new Scanner(System.in);
```

No código isso fica da seguinte forma:

UmPrograma.java

```
import java.util.*;

class UmPrograma {
   public static void main(String[] args) {
      int valor;
      Scanner teclado;

      teclado = new Scanner(System.in);

      System.out.println("Ola, mundo!");
      }
   }
```

Para simplificar o código, podemos colocar a definição do nome da variável do objeto e a sua criação na mesma linha, como indicado abaixo.

UmPrograma.java

```
import java.util.*;

class UmPrograma {
   public static void main(String[] args) {
      int valor;
      Scanner teclado = new Scanner(System.in);

      System.out.println("Ola, mundo!");
      }
   }
```

Tudo isso para criarmos uma **instância** da classe Scanner para leitura de teclado, ou seja, um objeto, ao qual demos o nome de **teclado**. Observe como o nome dado à variável tem relação direta com sua função.

Agora que o objeto para ler o teclado está pronto, precisamos pedir a ele que recolha informações a serem digitadas pelo usuário. Neste primeiro exemplo, vamos pedir que o usuário digite um número inteiro. Isso pode ser feito com o método **nextInt()** da classe Scanner.

> Sempre que criamos um objeto específico a partir de uma classe, este objeto também contera os métodos daquela classe.

Assim, usaremos o método **nextInt()** do objeto **teclado** para ler um número inteiro digitado pelo usuário, e o resultado será colocado na variável **valor**, o que pode ser feito da seguinte forma:

```
valor = teclado.nextInt();
```

No nosso código, isso fica da seguinte forma:

UmPrograma.java

```
import java.util.*;

class UmPrograma {
   public static void main(String[] args) {
      int valor;
      Scanner teclado = new Scanner(System.in);

      valor = teclado.nextInt();

      System.out.println("Ola, mundo!");
      }
   }
```

Para finalizar, vamos alterar o código do System.out para que ele imprima o número digitado, ao invés do texto "Ola, mundo!":

UmPrograma.java

```
import java.util.*;

class UmPrograma {
   public static void main(String[] args) {
      int valor;
      Scanner teclado = new Scanner(System.in);

      valor = teclado.nextInt();
      System.out.println(valor);
      }
   }
```

Compilando e executando este programa, temos uma surpresa: a tela fica preta, sem pedir nada... se digitarmos um número, o programa re-imprime o número; porém, o programa não pede nada. Precisamos pedir explicitamente, usando o método System.out.println, como indicado abaixo:

UmPrograma.java

```
import java.util.*;

class UmPrograma {
   public static void main(String[] args) {
      int valor;
      Scanner teclado = new Scanner(System.in);
```

```
      System.out.println("Digite um número inteiro");
      valor = teclado.nextInt();
      System.out.println(valor);
      }
   }
```

E também podemos aproveitar e melhorar a mensagem de resposta:

UmPrograma.java

```
import java.util.*;

class UmPrograma {
   public static void main(String[] args) {
      int valor;
      Scanner teclado = new Scanner(System.in);


      System.out.println("Digite um número inteiro");
      valor = teclado.nextInt();
      System.out.println("Valor digitado: " + valor);
      }
   }
```

Observe que basta usar a operação "+" para acrescentar o conteúdo da variável **valor** ao texto que será impresso!

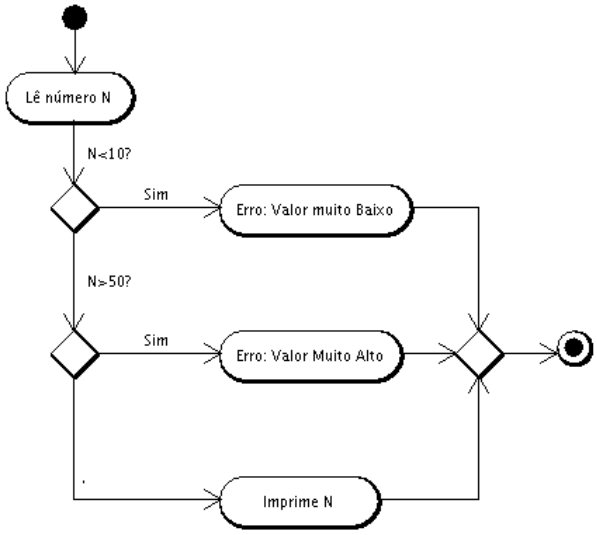
Com isso, já somos capazes de coletar dados digitados. Vejamos agora como tomar decisões com base nas informações digitadas pelo usuário!

2. ESTRUTURAS DE CONTROLE

Estruturas de Controle são instruções da linguagem de programação que permitem decidir, através dos valores das variáveis, qual trecho de código será executado. Por exemplo, imaginemos o seguinte programa:

Faça um programa que leia um número de 10 a 50 (inclusive estes), imprimindo o valor digitado. Caso o número digitado seja muito pequeno ou muito grande, uma mensagem de erro específica deve ser impressa: "Valor muito pequeno" ou "Valor muito alto", respectivamente.

Para simplificar nossa tarefa de escrever o programa, vamos trabalhar com o fluxograma que representa o processo, que é indicado a seguir.



Para programar esse algoritmo, iremos começar com um programa vazio.

EstruturaIf.java

```
import java.util.*;

class EstruturaIf {
   public static void main(String[] args) {

      }
   }
```

Seguindo o diagrama, a primeira coisa a ser feita é ler o número "N". Vamos então criar um objeto da classe Scanner para a leitura, chamado teclado, e uma variável do tipo int chamada N, para podermos realizar esta tarefa:

EstruturaIf.java

```
import java.util.*;

class EstruturaIf {
   public static void main(String[] args) {
      int N;
      Scanner teclado = new Scanner(System.in);
      System.out.println ("Digite um número inteiro");
      N = teclado.nextInt();
      }
   }
```

Vamos colocar um pequeno comentário para nos ajudar a entender cada bloco, conforme o nome das elipses do diagrama.

EstruturaIf.java

```
import java.util.*;

class EstruturaIf {
   public static void main(String[] args) {
      int N;
      Scanner teclado = new Scanner(System.in);

      // Lê número N
      System.out.println ("Digite um número inteiro");
      N = teclado.nextInt();
      }
   }
```

Seguindo o diagrama, temos uma decisão a ser feita: se N for menor que 10, imprimimos um erro... caso contrário, continuamos seguindo a outra setinha. Isso pode ser feito com a estrutura IF, que funciona assim:

```
if (comparação) {
   // código a executar se a comparação for verdadeira
   }
else {
   // código a executar se a comparação for falsa
   }
```

No nosso caso, isso pode ser adaptado assim:

EstruturaIf.java

```
import java.util.*;

class EstruturaIf {
   public static void main(String[] args) {
      int N;
      Scanner teclado = new Scanner(System.in);

      // Lê número N
      System.out.println ("Digite um número inteiro");
      N = teclado.nextInt();

      // Primeira decisão: verifica se número é baixo
      if ( N<10 ) {
         // Imprime Erro: Valor muito baixo
         }
      else {
         // Executa se valor não for muito baixo
         }
      }
   }
```

Ora, imprimir o erro é fácil, basta usar um System.out.println:

```
import java.util.*;

class EstruturaIf {
   public static void main(String[] args) {
      int N;
      Scanner teclado = new Scanner(System.in);

      // Lê número N
      System.out.println ("Digite um número inteiro");
      N = teclado.nextInt();

      // Primeira decisão: verifica se número é baixo
      if ( N<10 ) {
         // Imprime Erro: Valor muito baixo
         System.out.println("Valor muito baixo!");
         }
      else {
         // Executa se valor não for muito baixo
         }
      }
   }
```

Agora escrevemos o código da outra "setinha", que toma a segunda decisão:

```
import java.util.*;

class EstruturaIf {
   public static void main(String[] args) {
      int N;
      Scanner teclado = new Scanner(System.in);

      // Lê número N
      System.out.println ("Digite um número inteiro");
      N = teclado.nextInt();

      // Primeira decisão: verifica se número é baixo
      if ( N<10 ) {
         // Imprime Erro: Valor muito baixo
         System.out.println("Valor muito baixo!");
         }
      else {
         // Segunda decisão: verifica se número é alto
         if ( N>50 ) {
            // Imprime Erro: Valor muito baixo
            System.out.println("Valor muito alto!");
            }
         else {
            // Executa se o valor estiver na faixa esperada!
            }
        }
     }
  }
```

Agora só falta imprimir o valor digitado, no segundo **else**, ou seja, quando as duas verificações foram satisfeitas:

EstruturaIf.java

```
import java.util.*;

class EstruturaIf {
   public static void main(String[] args) {
      int N;
      Scanner teclado = new Scanner(System.in);

      // Lê número N
      System.out.println ("Digite um número inteiro");
      N = teclado.nextInt();

      // Primeira decisão: verifica se número é baixo
      if ( N<10 ) {
         // Imprime Erro: Valor muito baixo
         System.out.println("Valor muito baixo!");
         }
      else {
         // Segunda decisão: verifica se número é alto
         if ( N>50 ) {
            // Imprime Erro: Valor muito baixo
            System.out.println("Valor muito alto!");
            }
         else {
            // Executa se o valor estiver na faixa esperada!
            System.out.println ("Valor digitado: " + N);
            }
         }
      }
   }
```

Isso finaliza o nosso programa!

Na próxima aula veremos como usar estruturas de **repetição**!

3. EXERCÍCIO

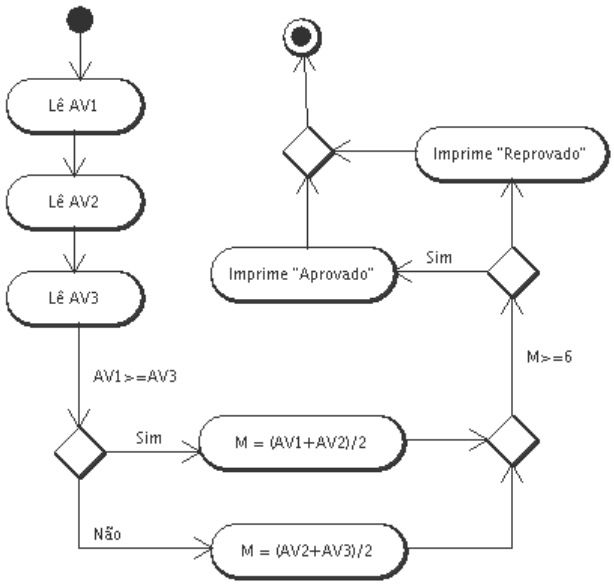
1. Implemente um código que calcule a média do aluno, seguindo as novas regras da Estácio Radial, e indique se o aluno está aprovado ou reprovado. Regra:

Se AV1 >= AV3 ==> M = (AV1 + AV2) / 2
Se AV1 < AV3 ==> M = (AV3 + AV2) / 2

M >= 6 ==> Aprovado M < 6 ==> Reprovado

IGNORE a regra de que as notas precisam ser maiores ou iguais a 4,0!

O diagrama está indicado a seguir:



2. Tente acrescentar ao diagrama as verificações que seriam necessárias para considerar que as notas escolhidas não podem ser menores que 4,0.

3. Implemente no código, em Java, essas verificações.

4. EXTRA: Tente desenhar o diagrama e implementar o algoritmo para o antigo sistema de notas da UniRadial:

M = (AD1 + AD2 + AC + PI) / 4

Sendo que M >= 7,0 para aprovação, M < 2,0 para reprovação e se M >=2,0 e M < 7,0, o programa deve perguntar a nota R, e, nesta segunda etapa:

MR = (M + R) / 2

Sendo que MR >= 6,0 para aprovação e MR < 6,0 para reprovação.

4. BIBLIOGRAFIA

CAELUM, FJ-11: Java e Orientação a Objetos. Acessado em: 10/01/2010. Disponível em: < http://www.caelum.com.br/curso/fj-11-java-orientacao-objetos/ >
DEITEL, H.M; DEITEL, P.J. **Java: como programar** - Sexta edição. São Paulo: Pearson-Prentice Hall, 2005.
HOFF, A; SHAIO, S; STARBUCK, O. **Ligado em Java**. São Paulo: Makron Books, 1996.