Capítulo 10

Mergulhando no `while`

Quais novidades veremos neste capítulo?

- Reforço em `while`;
- O comando `do...while`.

10.1 Um pouco mais sobre primos

Vamos iniciar este capítulo com dois exercícios. Primeiro, que tal modificarmos o método de geração de primos para que ele use o fato de que os únicos candidatos a primos maiores do que 2 são ímpares? Uma complicação interessante é que o `limiteInferior` para o próximo primo pode ser modificado pelo usuário a qualquer momento chamando `carregaLimiteInferior`. Isso deve ser contemplado na solução. Aqui vai a resposta:

```
/**
 * A cada chamada, calcula um novo primo seguindo ordem crescente.
 */
int próximoPrimo()
{
  // Move o limite inferior na direção do próximo primo.
  // Temos que considerar que o limite inferior pode ser par
  // porque ele pode ser modificado a qualquer momento com uma
  // chamada a carregaLimiteInferior.
  if (limiteInferior == 1)
    limiteInferior = 2;
  else if (limiteInferior % 2 == 0)
    limiteInferior = limiteInferior + 1;
  else
    limiteInferior = limiteInferior + 2;

  // Encontra o próximo primo
  while (!éPrimo(limiteInferior))
    limiteInferior = limiteInferior + 2;
```

```
    return limiteInferior;
  }
```

Nosso próximo desafio é criar uma nova classe `ManipuladorDeInteiros`. Ela deve conter o método `fatoraInteiro` que deve imprimir a decomposição em primos de um inteiro positivo maior ou igual a 2. Uma dica importante é usar o `GeradorDePrimos` Antes de ler solução colocada abaixo, tente com afinco fazer o exercício sozinho.

```
class ManipuladorDeInteiros {
  /**
   * Fatora em primos um inteiro > 1.
   */
  void fatoraInteiro(int x)
  {
    System.out.print(x + " =");

    // Usa um gerador de primos para encontrar os primos menores ou iguais a x.
    GeradorDePrimos gerador = new GeradorDePrimos();
    int primo = gerador.próximoPrimo();

    // Continua fatorando o número até que x se torne 1.
    while (x > 1)
    {
      if (x % primo == 0)
      {
        System.out.print(" " + primo);
        x = x / primo;
      }
      else
        primo = gerador.próximoPrimo();
    }

    // Imprime um fim de linha no final.
    System.out.println();
  }
}
```

Um exemplo de uso do nosso `ManipuladorDeInteiros`:

```
Welcome to DrJava.
> ManipuladorDeInteiros m = new ManipuladorDeInteiros()
> m.fatoraInteiro(5)
5 = 5
> m.fatoraInteiro(10)
10 = 2 5
> m.fatoraInteiro(18)
18 = 2 3 3
> m.fatoraInteiro(123456)
123456 = 2 2 2 2 2 2 3 643
> m.fatoraInteiro(12345678)
12345678 = 2 3 3 47 14593
```

```
> m.fatoraInteiro(167890)
167890 = 2 5 103 163
>
```

Obs: Note que na solução usamos uma rotina de impressão nova: `System.out.print`. Ela é muito parecida com `System.out.println` com a diferença de que não muda a linha ao terminar de imprimir.

10.2 Uma biblioteca de funções matemáticas.

Terminamos com um exercício clássico. Vamos mostrar como construir uma pequena biblioteca de funções matemáticas avançadas. Com será que o computador consegue calcular senos, cossenos, logaritmos? O segredo para implementar essas funções em Java é um bom conhecimento de cálculo e laços.

Usando cálculo, sabemos que essas funções matemáticas "complicadas" possuem expansões de Taylor. Estas expansões transformam uma função numa série de polinômios[1], que podem ser facilmente calculados usando laços. Vejamos a expansão de algumas dessas funções:

- $sen(x) = \frac{x}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} - \ldots + \frac{(-1)^k x^{(2k+1)}}{(2k+1)!} + \ldots$
- $cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots + \frac{(-1)^k x^{(2k)}}{(2k)!} + \ldots$
- $ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \ldots + \frac{(-1)^{(k-1)} x^k}{k} + \ldots$

  Isso funciona bem sempre que $|x| < 1$.

O segredo para usar essas fórmulas no computador é continuar somando até que o módulo do próximo termo seja muito pequeno e por isso possa ser desprezado.

Antes de apresentarmos aqui a solução que consideramos ideal, faça com cuidado e atenção os exercícios 1 (implementação das funções `double pot(double x, int y)` e `double fat(double x)`) e 2 (implementação da função `double sen(double x)` usando as funções do exercício 1.

Agora, após termos feito os exercícios 1 e 2, iremos criar uma nova classe, que chamaremos `Matemática`, com métodos para calcular funções como as apresentadas acima. Abaixo vemos a classe com uma função que calcula $sen(x)$ implementada. Compare esta forma de implementar com as formas usadas no exercício. Qual é mais rápida? Qual é mais fácil de entender?

```
class Matemática
{
  // Controla o significado de "pequeno".
  double epsilon = 1.0e-8;

  double sen(double x)
  {
    int k = 1;
    double termo = x;
    double seno = termo;
    while (termo*termo > epsilon*epsilon)
```

[1] Mais informações sobre séries de Taylor podem ser encontradas em livros de Cálculo.

```
      {
        // É muito mais fácil construir o próximo termo usando o anterior.
        k = k + 2;
        termo = -termo * x * x / (k - 1) / k;
        seno = seno + termo;
      }
      return seno;
    }
  }
```

Um exemplo de uso:

```
Welcome to DrJava.
> m = new Matemática();
> m.sen(0.3)
0.2955202066613839
> m.sen(0.5)
0.4794255386164159
> m.sen(3.141596/2.0)
0.9999999999925767
>
```

10.3 do...while

Para complementar os laços possíveis em Java, vejamos uma pequena variação do `while`. Nele a condição é testada sempre antes de execução do corpo de comandos que compõe o laço. Já o laço `do...while` tem a condição testada apenas no final. Conseqüentemente, no caso do `do...while`, existe a garantia que o conteúdo no interior do laço será executado pelo menos uma vez, enquanto no `while` este pode nunca ser executado. Na prática, a existência destes dois tipos de laços é uma mera conveniencia sintática, já que um pode ser facilmente substituído pelo outro.

Vejamos um exemplo de utilização do `do...while`:

```
int fatorial(int x)
{
  int resultado = 1;
  do
  {
    resultado = resultado * x;
    x = x - 1;
  } while (x > 1)
  return resultado;
}
```

Exercícios

1. Implemente na classe `Matemática` as funções `double pot(double x, int y)` que calcula x^y e `double fat(double x)` que calcula $x!$ [2].

2. Implemente na classe `Matemática` a função `double sen(double x)` utilizando-se das funções `double pot(double x, int y)` e `int fat(int x)` do item anterior.

3. Implemente na classe `Matemática` funções para calcular $cos(x)$ e $ln(1+x)$. Note que para implementar o $ln(1+x)$ deve-se criar uma função `double ln(double x)` e no interior da função definir uma variável local `x2 = 1 - x` de modo que se possa calcular $ln(1+x2)$.

4. Escreva uma classe `TestaMatemática` que utiliza os métodos matemáticos `java.lang.Math.sen()`, `java.lang.Math.cos()`, `java.lang.Math.pow()` e `java.lang.Math.log()`, disponíveis na biblioteca Java, para testar os respectivos métodos implementado por você nos exercícios anteriores. Os métodos de `TestaMatemática` devem receber um parâmetro `double epsilon` que determina a diferença máxima aceitável entre o valor devolvido pela sua implentação com relação ao da implementação da biblioteca Java. Qualquer dúvida sobre a utilização destes métodos, consulte a documentação online do Java[3].

5. O enunciado deste exercício é bem mais complexo que a solução, por isso não tenha medo! Imagine um quadrado em um plano e uma reta paralela a um dos lados do quadrado: a projeção do quadrado sobre a reta tem exatamente o mesmo comprimento que o lado do quadrado. Imagine agora que este quadrado seja girado sobre o plano; a projeção do quadrado sobre a reta tem um novo tamanho. Crie uma classe Projetor que possua um método gira que aceite como parâmetro o número de graus que o quadrado deve girar em relação à sua posição anterior e imprima na tela o tamanho da projeção do quadrado sobre a reta. Note que se o usuário executar o método duas vezes, com os parâmetros "22" e "35", sua classe deve responder qual o tamanho da projeção para inclinações do quadrado de 22 e 57 graus.

   - Escreva 3 soluções para este exercício: uma que você considere elegante e clara, uma com um único método e uma com o máximo número possível de métodos. Utilize os métodos `sen()` e `cos()` desenvolvidos neste capítulo.
   - Utilize agora os métodos `java.lang.Math.cos()` e `java.lang.Math.sin()` disponíveis na biblioteca Java, que calculam, respectivamente, o coseno e o seno do ângulo passado como parâmetro em radianos (graus * PI/180 = radianos). Compare os resultados com os obtidos com nossas implementações de `sen()` e `cos()`.

[2] Apesar do cálculo de fatorial só usar inteiros, com o tipo `int` pode se calcular até 16!, e com o tipo `long` até 20!. Com o tipo `double` é possível calcular fatoriais maiores, mas com menos dígitos significativos.

[3] `http://java.sun.com/j2se/1.5.0/docs/api/java/lang/Math.html`.

Capítulo 11

Caracteres e Cadeias de Caracteres

Quais novidades veremos neste capítulo?

- Introdução do tipo `char`;
- Uma classe da biblioteca padrão: `String`.

11.1 Um tipo para representar caracteres

Até o momento já vimos diferentes tipos de variáveis, como os inteiros (`int`) e os reais (`double`). Além disto, também vimos as variáveis booleanas, que podem ter apenas dois valores, verdadeiro ou falso (`boolean`). Parece intuitivo que as linguagens de programação também ofereçam variáveis para a manipulação de caracteres. No caso de Java temos o tipo `char`. Vejamos um exemplo de uso:

```
class Caractere1
{
  void verificaResposta(char ch)
  {
    if ((ch == 's') || (ch =='S'))
      System.out.println("A resposta foi sim");
    else if ((ch == 'n') || (ch == 'N'))
      System.out.println("A resposta foi não");
    else
      System.out.println("Resposta inválida");
  }
}
```

No exemplo acima podemos ver que para se representar um caractere usamos aspas simples (’). Também podemos ver que os caracteres minúsculos são diferentes do mesmo caractere maiúsculo.

Um outro exemplo um pouco mais elaborado pode ser visto abaixo:

```
class Caracteres
{

  void imprimeCaracteres(char ch, int n)
```

```
  {
    int i = 0;
    while (i < n)
    {
      System.out.print(ch);
      i = i + 1;
    }
  }
}
```

Neste exemplo, são impressos diversos caracteres do mesmo tipo. Observe abaixo como podemos adicionar um novo método para desenhar letras grandes:

```
class Caracteres
{

  void imprimeCaracteres(char ch, int n)
  {
    int i = 0;
    while (i < n)
    {
      System.out.print(ch);
      i = i + 1;
    }
  }
  void novaLinha()
  {
    System.out.println();
  }
  void imprimeCaracteresNL(char ch, int n)
  {
    imprimeCaracteres(ch, n);
    novaLinha();
  }
  void desenhaE()
  {
    imprimeCaracteresNL('*',20);
    imprimeCaracteresNL('E', 15);
    imprimeCaracteresNL('E', 14);
    imprimeCaracteresNL('E', 3);
    imprimeCaracteresNL('E', 3);
    imprimeCaracteresNL('E', 13);
    imprimeCaracteresNL('E', 13);
    imprimeCaracteresNL('E', 3);
    imprimeCaracteresNL('E', 3);
    imprimeCaracteresNL('E', 14);
    imprimeCaracteresNL('E', 15);
    imprimeCaracteresNL('*',20);
  }
}
```

A saída do programa é:

```
********************
EEEEEEEEEEEEEEE
EEEEEEEEEEEEEE
EEE
EEE
EEEEEEEEEEEEE
EEEEEEEEEEEEE
EEE
EEE
EEEEEEEEEEEEEE
EEEEEEEEEEEEEEE
**********************
```

Para desenharmos letras onde é necessário intercalar espaços e letras em uma única linha, a implementação fica um pouco mais longa. Os diferentes métodos que imprimem caracteres pulando ou sem pular linha serão usados. Por exemplo, para a letra U, a primeira linha deve ser impressa como:

```
imprimeCaracteres('U', 3);
imprimeCaracteres(' ', 9);  // espaço também é um caractere
imprimeCaracteresNL('U", 3);
```

11.2 Cadeias de caracteres (`Strings`)

Uma forma de escrevermos palavras no computador seria usando grupos de caracteres, entretanto isto depende de um conceito mais avançado que ainda não vimos. A nossa outra opção é usar uma classe pronta, que já vem com a linguagem Java, a classe `String`.

Nos nossos primeiros exemplos, já havíamos feito algumas operações com strings, por exemplo:

```
System.out.println("O triângulo é retângulo");
System.out.println("A raiz de " + 4 + " é igual a " + 2 + ".");
```

Agora nós veremos com mais detalhes esta classe `String` . Podemos ver que o operador + tem um significado natural o de concatenação. Logo, as seguintes operações são válidas:

```
String a = "abc";
String b = "cdf";
String c;

c = a + b;
System.out.println(c);
```

Podemos também concatenar números a uma `String`:

```
String a = "O resultado é";
int i = 10;
String c;

c = a + i;
System.out.println(c);
```

Além disto, existem alguns métodos pré-definidos na classe `String`. Entre eles podemos citar:

- `char charAt(int index)` - devolve o caractere na posição `index`.
  Os índices em uma `String` vão de zero ao seu tamanho menos um. Exemplo:

```
String s = "mesa";
System.out.println(s.charAt(0)); // Imprime  m
System.out.println(s.charAt(3)); // Imprime  a
```

- `boolean endsWith(String suffix)` - verifica se a `String` acaba com o sufixo dado. Usado, entre outras coisas, para verificar as extensões dos arquivos. Por exemplo, verificar se o nome de um arquivo termina com `".java"`.

- `int indexOf(char ch)` - devolve o índice da primeira ocorrência de `ch` na `String` ou -1 caso o caractere não ocorra na `String`. Exemplo:

```
String s1 = "EP1.java";
System.out.println(s1.indexOf('.')); // imprime 3
System.out.println(s1.indexOf('x')); // imprime -1
```

- `int length()` - devolve o tamanho da `String`. Exemplo:

```
String s1 = "mesa";
System.out.println(s1.lenghth()); // imprime 4
```

- `String toUpperCase()` - devolve a `String` convertida para letras maiúsculas. Exemplo:

```
String s1 = "mesa";
System.out.println(s1.toUpperCase()); // imprime MESA
```

- `int compareTo(String outra)` - compara duas `String`s.
  Devolve um número positivo se a `outra` for menor, 0 se forem iguais, e um negativo caso contrário. Exemplo:

```
String s1 = "mesa";
String s2 = "cadeira";
System.out.println(s1.compareTo(s2)); // imprime 10 que é > 0
```

Exercícios

1. Escreva uma classe `Linha` que possua um método `imprimeLinha` que, ao ser chamado, imprime uma linha de caracteres `X` na diagonal, na tela de interações do DrJava. Use laços `while`. DICA: você vai precisar do método `System.out.print()`, que imprime seu argumento na tela mas não passa para a linha seguinte; imprima linhas com número crescente de espaços no começo e o caractere `X` no final.

2. Escreva uma classe contendo um método que devolve o número de ocorrências da vogal `a` em uma frase contida em uma `String`. Crie um teste para verificar o funcionamento desta classe.

3. Implemente uma classe com um método que determina a freqüência relativa de vogais em uma `String`. Considere que as letras maiúsculas e minúsculas não estão acentuadas. Crie um teste para verificar o funcionamento desta classe.