Capítulo 13

Manipulando Números Utilizando Diferentes Bases

Quais novidades veremos neste capítulo?

- Como extrair os dígitos de um número;
- Como converter números de uma base para outra.

Neste capítulo, aprenderemos como extrair e processar os dígitos de um número inteiro e como realizar a conversão de um número entre diferentes bases. Mas antes, faremos uma introdução aos principais sistemas de numeração utilizados para representar números inteiros.

13.1 Sistemas de numeração

Números inteiros podem ser representados utilizando diferentes sistemas de numeração. Por exemplo, no dia-a-dia, representamos os números utilizando o sistema decimal, isto é, utilizamos 10 algarismos para representar os números, dados por $\{0,1,2,3,4,5,6,7,8,9,0\}$. Um número é formado pela concatenação destes algarismos, onde a posição dos algarismos determina um valor pelo qual aquele algarismo deve ser multiplicado. Para o sistema decimal, também chamado de base 10, este valor é de 10^{pos}, onde *pos* é a posição do algarismo, contada a partir da direita, que possui *pos* igual a 0 (zero). Por exemplo, o número 1256 tem o seu valor dado por:

$$1*10^3+2*10^2+5*10^1+6*10^0 = 1000+200+50+6 = 1256$$

Já os computadores representam os dados internamente utilizando o sistema binário, cujos números são representados pelos algarismo {0, 1}. O motivo para a utilização de uma base binária se deve ao fato dos computadores trabalharem utilizando apenas dois níveis de voltagem, por exemplo, $+5V$ e $0V$. Voltando ao exemplo anterior, o número 1256 seria representado na base binária por 10011101000_2, pois:

$$10011101000_2 = 2^10+2^7+2^6+2^5+2^3 = 1024+128+64+32+8 = 1256$$

Apesar da representação binária ser aquela utilizada internamente pelo computador, esta é de difícil leitura para os seres humanos, devido às longas seqüências de algarismos 0 e 1. Para aliviar este problema, duas outras bases são bastantes utilizadas na computação, a base octal e a hexadecimal.

A base octal corresponde à concatenação de 3 bits, que combinados dão origem a 8 diferentes valores, representados pelos algarismos de 0 a 7. Já a base hexadecimal corresponde à concatenação de 4 bits, gerando 16 combinação diferentes. Como possuímos apenas 10 algarismo numéricos, utilizamos as letras do alfabeto $\{A,B,C,D,E,F\}$. O número 1256 nestas bases seria então dado por:

$$2350_8 = 2*8^3+3*8^2+5*8^1 = 2*512+3*64+5*8 = 1256$$

$$4E8_{16} = 4*16^2+14*16^1+8*16^0 = 4*256+14*16+8 = 1256$$

Reparem que a representação hexadecimal é mais compacta que a decimal, o que é esperado, dado que utilizamos mais algarismos para representar os números. Veremos agora como realizar a conversão de números inteiros entre diferentes sistemas de numeração.

13.2 Conversão entre sistemas de numeração

Analise agora, com cuidado, a implementação da classe Conversão que converte números binários para decimais e vice-versa. Este exemplo também demonstra como realizar a extração dos dígitos de um número inteiro. Não é muito fácil entender o funcionamento de seus métodos; sugerimos fortemente, então, que você tente rastrear[1] a execução dos dois métodos.

```
class Conversão
{
  int binárioParaDecimal(int n)
  {
    int dec = 0;
    int pot2  = 1;

    while (n != 0){
      /* processa um dígito binário */
      dec = dec + n % 10 * pot2;
      n = n / 10;
      pot2 = pot2 * 2;
    }
    return dec;
  }

  int decimalParaBinário(int n)
  {
    int dig;
    int bin = 0;
    int pot = 1;
```

[1] Rastrear a execução de um programa significa construir uma tabela contendo colunas correspondentes às variáveis do programa e simular a sua execução indicando os valores que as variáveis recebem ao longo do tempo. Se o programa imprimir texto (usando `System.out.println`) esse texto tem que ser indicado como *saída* e se os métodos devolvem algum valor (com o comando `return`) este valor tem que ser destacado.

```
        while (n > 0) {
          /* extrai próximo dígito binário menos significativo (mais à direita) */
          dig = n % 2;
          /* remove esse dígito do n */
          n = n / 2;
          /* adiciona o dígito como o mais significativo até o momento */
          bin = bin + dig * pot;
          pot = pot * 10;
        }
        return bin;
      }
    }
```

Exercícios

1. Crie uma classes `Contador` com um método que recebe um número natural `n` e devolve seu número de dígitos. Em seguida, escreva uma classe `TestaContador` que testa o método para diferentes valores de `n`. Não se esqueça de testar o caso do número ser igual a zero!

2. Dado um número, verificar se o mesmo possui dois dígitos consecutivos iguais. Para resolver este problema podemos usar um técnica denominada indicador de passagem, para isto, inicialmente vamos supor que o número não contém dígitos iguais. Verificaremos cada par de dígitos consecutivos, caso algum deles seja igual, saberemos que ele contém dígitos consecutivos iguais. Em outras palavras, vamos inicializar uma variável booleana com falso e testar a condição para todos os dígitos consecutivos; se forem iguais mudamos a condição. Ao final, a resposta vai corresponder ao estado final desta condição.

3. Dado um número natural `n`, verificar se `n` é palíndromo. Um número palíndromo é um número que lido de trás para frente é o mesmo quando lido normalmente, por exemplo:

   - 78087
   - 1221
   - 11111
   - 3456 não é palindromo!!!

   Duas maneiras de se resolver estes problemas são apresentadas abaixo. Escreva as soluções para cada uma delas.

   - A forma mais fácil é construir o número inverso e compará-lo com o original.
   - A outra solução consiste em supor inicialmente que o número é palíndromo e em seguida verificar se a condição de igualdade é válida para os extremos.

   Se o número for negativo, considere apenas o seu valor absoluto (isso é apenas uma convenção nossa para este exercício). Por exemplo, -2002 deve ser considerado palíndromo.

   Curiosidade: números palíndromos também são conhecidos por *capicuas*.

4. Uma propriedade de números naturais é a seguinte: um número sempre é maior do que o produto dos seus dígitos. Faça uma classe com dois métodos: `int calculaProd(int n)`, que calcula o produto dos dígitos de um número natural `n` e `boolean verificaProp(int n)`, que verifica se a propriedade é válida para um número `n` dado.

5. Crie métodos para converter uma `String` contendo um número em algarismos romanos em um inteiro e vice-versa. Crie testes para estes métodos.

6. Crie uma classe contendo um método que recebe um inteiro e o imprime representado em notação científica. Por exemplo,

```
> Inteiro i = new Inteiro();
> i.carregaValor(1356);
> i.imprimeEmNotacaoCientifica();
1,356e3
> i.carregaValor(-102);
> i.imprimeEmNotacaoCientifica();
-1,02e2
> i.carregaValor(7);
> i.imprimeEmNotacaoCientifica();
7,0e0
> i.carregaValor(900200);
> i.imprimeEmNotacaoCientifica();
9,002e5
```

7. Implemente a operação de divisão de dois números inteiros utilizando apenas laços e os operadores `+` e `-`.

Capítulo 14

Arrays (vetores)

Quais novidades veremos neste capítulo?

- *arrays* (vetores);
- programas independentes em Java (método `main`).

Muitas vezes, precisamos que um objeto guarde um grande número de informações. Por exemplo, se precisamos calcular a temperatura média em um dado mês poderíamos ter uma classe similar à seguinte:

```
class TemperaturasDoMês
{
    double t1, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11, t12
           t13, t14, t15, t16, t17, t18, t19, t20, t21, t22, t23,
           t24, t25, t26, t27, t28, t29, t30, t31;
     // etc.
}
```

onde cada variável guarda a temperatura média de um dia do mês. Isso é claramente indesejável. Imagine ainda se quiséssemos uma classe para guardar as temperaturas médias de todos os dias do ano. Precisaríamos de 365 variáveis? Felizmente não!

A linguagem Java possui o conceito de *array* que é uma estrutura de dados que permite guardar uma seqüência de valores (números, caracteres ou objetos quaisquer) de uma forma única e organizada. Utilizando um *array*, a classe anterior ficaria assim:

```
class TemperaturasDoMês
{
    double[] temperaturas = new double[31];
    //etc.
}
```

Note que, no exemplo acima, a linha que define o *array* `temperaturas` faz duas operações simultaneamente. Esta linha poderia ser separada em dois passos:

- `double[] temperaturas;` define um novo *array* chamado temperaturas que irá conter valores do tipo double. Por enquanto, o *array* está vazio.

- `temperaturas = new double[31];` especifica que o *array* guardará exatamente 31 valores do tipo double. Neste instante, o ambiente Java reserva a memória necessária para guardar estes 31 valores.

> **Nota Lingüística:** a tradução padrão de *array* para o português é *vetor*. No entanto, a linguagem Java contém um tipo de objeto chamado `Vector` que é semelhante a *arrays*, mas não é igual. Para evitar confusões entre *arrays* e Vectors, preferimos não traduzir a palavra *array* para vetor neste livro.

Vejamos agora um exemplo simples de utilização de *arrays*.

```
class BrincadeirasComArrays
{
    String[] diasDaSemana = new String[7];
    int[] quadrados = new int[10];

    void defineDiasDaSemana()
    {
        diasDaSemana[0] = "domingo";
        diasDaSemana[1] = "segunda-feira";
        diasDaSemana[2] = "terça-feira";
        diasDaSemana[3] = "quarta-feira";
        diasDaSemana[4] = "quinta-feira";
        diasDaSemana[5] = "sexta-feira";
        diasDaSemana[6] = "sábado";
    }

    void calculaQuadrados()
    {
        int i = 0;
        while (i < 10)
        {
            quadrados[i] = i*i;
            i++;
        }
    }

    void listaDiasDaSemana()
    {
        int i = 0;
        while (i < 7)
        {
            System.out.println(diasDaSemana[i]);
            i++;
        }
    }

    void listaQuadrados()
    {
        int i = 0;
        while (i < 10)
        {
```

```
            System.out.println(i + " ao quadrado é " + quadrados[i]);
            i++;
        }
    }
}
```

O atributo `length`

Arrays são na verdade um tipo especial de objeto em Java. Qualquer *array* já vem com um atributo pré-definido, chamado `length`, que contém o comprimento do *array*. Desta forma, o método `calculaQuadrados` acima poderia ser reescrito para

```
    void calculaQuadrados()
    {
        int i = 0;
        while (i < quadrados.length)
        {
            quadrados[i] = i*i;
            i++;
        }
    }
```

O valor do atributo length é definido automaticamente pelo ambiente Java, o programador não pode alterá-lo. Assim, `quadrados.length = 2` é uma operação ilegal.

Inicialização de *arrays*

Existe também a opção de inicializar um *array* no momento em que ela é declarada. Assim, podemos inicializar *arrays* de inteiros e de Strings conforme o exemplo a seguir:

```
int[] primos = {2, 3, 5, 7, 11, 13, 17, 19, 23};

String[] planetas = {"Mercúrio", "Vênus", "Terra", "Marte", "Júpiter", "Saturno",
                     "Urano", "Netuno", "Plutão"};
```

14.1 Criação de programas Java

Até este capítulo, todos os exemplos de códigos que vimos, a rigor, não eram "programas", eles eram apenas classes Java que podiam ser usadas dentro do interpretador do DrJava. Mas, e se quiséssemos criar um programa para ser utilizado por alguém que não possui o DrJava em sua máquina? Neste caso, precisamos criar um programa a ser executado ou na linha de comando do sistema operacional ou dando um "clique duplo"com o mouse em cima do ícone do programa. Para fazer isso, basta que a classe principal do programa possua um método `main` como no exemplo a seguir.

```
class BrincadeirasComArrays
{
    // aqui vão os demais métodos e atributos da classe

    public static void main(String[] arg)
```

```
    {
        BrincadeirasComArrays b = new BrincadeirasComArrays();
        b.defineDiasDaSemana();
        b.calculaQuadrados();
        b.listaDiasDaSemana();
        b.listaQuadrados();
        b.imprimeArray(arg);
    }

    void imprimeArray(String[] array)
    {
        int i = 0;
        while (i < array.length)
        {
            System.out.println(array[i]);
            i++;
        }
    }
}
```

Para executar o seu programa após compilar a classe, basta abrir um *shell* (um interpretador de comandos do sistema operacional; no unix pode ser, por exemplo, o bash; no windows pode ser, por exemplo, o command) e digitar

```
java BrincadeirasComArrays um dois três
```

onde, `java` é o nome do interpretador Java, `BrincadeirasComArrays` é o nome da classe que será carregada e cujo método `main` será executado e `um dois três` são apenas um exemplo de 3 argumentos que estamos passando, poderia ser qualquer outra coisa.

Neste exemplo, o programa `BrincadeirasComArrays` geraria a seguinte saída:

```
domingo
segunda-feira
terça-feira
quarta-feira
quinta-feira
sexta-feira
sábado
0 ao quadrado é 0
1 ao quadrado é 1
2 ao quadrado é 4
3 ao quadrado é 9
4 ao quadrado é 16
5 ao quadrado é 25
6 ao quadrado é 36
7 ao quadrado é 49
8 ao quadrado é 64
9 ao quadrado é 81
```

```
um
dois
três
```

portanto, para que uma classe possa ser executada a partir da linha de comando do sistema, é necessário que ele possua um método com a seguinte assinatura:

```
public static void main (String[] arg)
```

o nome do parâmetro não precisa ser exatamente `arg`, qualquer nome funciona; mas o seu tipo tem que ser obrigatoriamente um *array* de `String`s. A partir do DrJava é possível executar o método `main` pressionando a tecla F2.

Exercícios

Testes: até este capítulo, em diversos exercícios pedimos explicitamente para você criar testes para classes e métodos escritos. Fizemos isto para enfatizar a necessidade de escrever testes, um processo que deve ser realizado de modo automático. A partir deste capítulo não iremos mais solicitar em cada exercício que você escreva testes. Você deverá, a partir de agora, fazer isso naturalmente.

1. Escreva uma classe `Simples` contendo um método que recebe um *array* de inteiros como parâmetro e que inicializa todos os elementos do *array* com um valor, também dado como parâmetro. O método deve devolver o tamanho do *array*. A assinatura do método deve ser a seguinte:

   ```
   int inicializaArray(int[] a, int v);
   ```

   Escreva agora um método que recebe um *array* de inteiros como parâmetro e imprime o seu conteúdo:

   ```
   void imprimeArray(int[] a);
   ```

   Crie agora um método que, dado um inteiro, verifica se ele está presente no *array*.

   ```
   boolean estáNoArray(int []a, int v);
   ```

   Finalmente, escreva um programa que cria um *array*, cria um objeto `Simples` e chama os seus três métodos.

2. Crie um método `double[] freqüênciaRelativa(int[] v, int n)` que recebe um vetor contendo números inteiros no intervalo [0, n] e devolve um vetor contendo a freqüência relativa de cada um destes números.

3. Crie um método que, dados dois vetores `a` e `b`, verifica se o vetor de menor tamanho é uma subseqüência do vetor de tamanho maior.
   Ex: O vetor [9, 5] é uma subseqüência de [3, 9, 5, 4, -1].