Módulo 2

Introdução à Programação II

Lição 4

Passeio pelo pacote *java.lang*

Versão 1.0 - Mar/2007

Autor
Rebecca Ong

Equipe
Joyce Avestro
Florence Balagtas
Rommel Feria
Rebecca Ong
John Paul Petines
Sun Microsystems
Sun Philippines

Necessidades para os Exercícios

Sistemas Operacionais Suportados

NetBeans IDE 5.5 para os seguintes sistemas operacionais:

- Microsoft Windows XP Profissional SP2 ou superior
- Mac OS X 10.4.5 ou superior
- Red Hat Fedora Core 3
- Solaris™ 10 Operating System (SPARC® e x86/x64 Platform Edition)

NetBeans Enterprise Pack, poderá ser executado nas seguintes plataformas:

- Microsoft Windows 2000 Profissional SP4
- Solaris™ 8 OS (SPARC e x86/x64 Platform Edition) e Solaris 9 OS (SPARC e x86/x64 Platform Edition)
- Várias outras distribuições Linux

Configuração Mínima de Hardware

Nota: IDE NetBeans com resolução de tela em 1024x768 pixel

Sistema Operacional	Processador	Memória	HD Livre
Microsoft Windows	500 MHz Intel Pentium III workstation ou equivalente	512 MB	850 MB
Linux	500 MHz Intel Pentium III workstation ou equivalente	512 MB	450 MB
Solaris OS (SPARC)	UltraSPARC II 450 MHz	512 MB	450 MB
Solaris OS (x86/x64 Platform Edition)	AMD Opteron 100 Série 1.8 GHz	512 MB	450 MB
Mac OS X	PowerPC G4	512 MB	450 MB

Configuração Recomendada de Hardware

Sistema Operacional	Processador	Memória	HD Livre
Microsoft Windows	1.4 GHz Intel Pentium III workstation ou equivalente	1 GB	1 GB
Linux	1.4 GHz Intel Pentium III workstation ou equivalente	1 GB	850 MB
Solaris OS (SPARC)	UltraSPARC IIIi 1 GHz	1 GB	850 MB
Solaris OS (x86/x64 Platform Edition)	AMD Opteron 100 Series 1.8 GHz	1 GB	850 MB
Mac OS X	PowerPC G5	1 GB	850 MB

Requerimentos de Software

NetBeans Enterprise Pack 5.5 executando sobre Java 2 Platform Standard Edition Development Kit 5.0 ou superior (JDK 5.0, versão 1.5.0_01 ou superior), contemplando a Java Runtime Environment, ferramentas de desenvolvimento para compilar, depurar, e executar aplicações escritas em linguagem Java. Sun Java System Application Server Platform Edition 9.

- Para **Solaris**, **Windows**, e **Linux**, os arquivos da JDK podem ser obtidos para sua plataforma em http://java.sun.com/j2se/1.5.0/download.html
- Para **Mac OS X**, Java 2 Plataform Standard Edition (J2SE) 5.0 Release 4, pode ser obtida diretamente da Apple's Developer Connection, no endereço: http://developer.apple.com/java (é necessário registrar o download da JDK).

Para mais informações: http://www.netbeans.org/community/releases/55/relnotes.html

Colaboradores que auxiliaram no processo de tradução e revisão

Alexandre Mori
Alexis da Rocha Silva
Aline Sabbatini da Silva Alves
Allan Wojcik da Silva
André Luiz Moreira
Andro Márcio Correa Louredo
Antoniele de Assis Lima
Antonio Jose R. Alves Ramos
Aurélio Soares Neto
Bruno da Silva Bonfim
Bruno dos Santos Miranda
Bruno Ferreira Rodrigues
Carlos Alberto Vitorino de Almeida
Carlos Alexandre de Sene
Carlos André Noronha de Sousa
Carlos Eduardo Veras Neves
Cleber Ferreira de Sousa
Cleyton Artur Soares Urani
Cristiano Borges Ferreira
Cristiano de Siqueira Pires
Derlon Vandri Aliendres
Fabiano Eduardo de Oliveira
Fábio Bombonato
Fernando Antonio Mota Trinta
Flávio Alves Gomes
Francisco das Chagas
Francisco Marcio da Silva
Gilson Moreno Costa
Givailson de Souza Neves
Gustavo Henrique Castellano
Hebert Julio Gonçalves de Paula
Heraldo Conceição Domingues
Hugo Leonardo Malheiros Ferreira
Ivan Nascimento Fonseca
Jacqueline Susann Barbosa
Jader de Carvalho Belarmino
João Aurélio Telles da Rocha
João Paulo Cirino Silva de Novais
João Vianney Barrozo Costa
José Augusto Martins Nieviadonski
José Leonardo Borges de Melo
José Ricardo Carneiro
Kleberth Bezerra G. dos Santos
Lafaiete de Sá Guimarães
Leandro Silva de Morais
Leonardo Leopoldo do Nascimento
Leonardo Pereira dos Santos
Leonardo Rangel de Melo Filardi
Lucas Mauricio Castro e Martins
Luciana Rocha de Oliveira
Luís Carlos André
Luís Octávio Jorge V. Lima
Luiz Fernandes de Oliveira Junior
Luiz Victor de Andrade Lima
Manoel Cotts de Queiroz
Marcello Sandi Pinheiro
Marcelo Ortolan Pazzetto
Marco Aurélio Martins Bessa
Marcos Vinicius de Toledo
Maria Carolina Ferreira da Silva
Massimiliano Giroldi
Mauricio Azevedo Gamarra
Mauricio da Silva Marinho
Mauro Cardoso Mortoni
Mauro Regis de Sousa Lima
Namor de Sá e Silva
Néres Chaves Rebouças
Nolyanne Peixoto Brasil Vieira
Paulo Afonso Corrêa
Paulo José Lemos Costa
Paulo Oliveira Sampaio Reis
Pedro Antonio Pereira Miranda
Pedro Henrique Pereira de Andrade
Renato Alves Félix
Renato Barbosa da Silva
Reyderson Magela dos Reis
Ricardo Ferreira Rodrigues
Ricardo Ulrich Bomfim
Robson de Oliveira Cunha
Rodrigo Pereira Machado
Rodrigo Rosa Miranda Corrêa
Rodrigo Vaez
Ronie Dotzlaw
Rosely Moreira de Jesus
Seire Pareja
Sergio Pomerancblum
Silvio Sznifer
Suzana da Costa Oliveira
Tásio Vasconcelos da Silveira
Thiago Magela Rodrigues Dias
Tiago Gimenez Ribeiro
Vanderlei Carvalho Rodrigues Pinto
Vanessa dos Santos Almeida
Vastí Mendes da Silva Rocha
Wagner Eliezer Roncoletta

Auxiliadores especiais

Revisão Geral do texto para os seguintes Países:

- **Brasil** – Tiago Flach
- **Guiné Bissau** – Alfredo Cá, Bunene Sisse e Buon Olossato Quebi – ONG Asas de Socorro

Coordenação do DFJUG

- **Daniel deOliveira** – JUGLeader responsável pelos acordos de parcerias
- **Luci Campos -** Idealizadora do DFJUG responsável pelo apoio social
- **Fernando Anselmo -** Coordenador responsável pelo processo de tradução e revisão, disponibilização dos materiais e inserção de novos módulos
- **Regina Mariani -** Coordenadora responsável pela parte jurídica
- **Rodrigo Nunes -** Coordenador responsável pela parte multimídia
- **Sérgio Gomes Veloso -** Coordenador responsável pelo ambiente JEDI™ (Moodle)

Agradecimento Especial

John Paul Petines – Criador da Iniciativa JEDI™
Rommel Feria – Criador da Iniciativa JEDI™

1. Objetivos

Java possui muitas classes úteis para serem utilizadas. Nesta lição iremos explorá-las.

Ao final desta lição, o estudante será capaz de:

- Entender e utilizar as seguintes classes disponíveis:
  - Classe *Math*
  - Classe *String*
  - Classe *StringBuffer*
  - Classes *Wrapper*
  - Classe *Process*
  - Classe *System*

2. Classe *Math*

Java possui constantes pré-definidas e métodos para executar diferentes operações matemáticas, como, por exemplo, funções trigonométricas e logarítmicas. Como estes métodos são todos *static*, podemos utilizá-los sem a necessidade de construir um objeto da classe *Math*. Pode-se obter uma lista completa destas constantes e métodos na documentação da API Java. Aqui estão descritos alguns dos métodos mais utilizados:

Métodos da classe Math
`public static double abs(double a)`
Retorna o valor positivo do argumento *a* passado para o método. É um método *overloaded*. Pode receber como argumento um *float*, um *integer* ou um *long integer*, e nestes casos, o tipo de retorno será *float*, *integer* ou *long integer*, respectivamente.
`public static double random()`
Retorna um valor positivo randômico maior ou igual a 0.0 mas menor que 1.0.
`public static double max(double a, double b)`
Retorna o maior valor entre dois valores, *a* e *b*, do tipo *double*. É um método *overloaded*. Pode receber como argumento um *float*, *integer* ou *long integer*, e nestes casos, o tipo de retorno será *float*, *integer* ou um *long integer*, respectivamente.
`public static double min(double a, double b)`
Retorna o menor entre dois valores *a* e *b*, do tipo *double*. É um método *overloaded*. Pode receber também valores dos tipos *float*, *integer* ou *long integer* como argumentos, e nestes casos, o tipo de retorno será serão *float*, *integer* ou *long integer*, respectivamente.
`public static double ceil(double a)`
Retorna o menor inteiro maior ou igual ao argumento *a* passado para o método.
`public static double floor(double a)`
Retorna o maior inteiro menor ou igual ao argumento *a* passado para o método.
`public static double exp(double a)`
Retorna o número de Euler "*e*" elevado à potência informada no argumento *a* passado para o método.
`public static double log(double a)`
Retorna o logarítmico natural (base e) de *a*, o argumento do tipo *double* informado.
`public static double pow(double a, double b)`
Retorna o valor do tipo *double* de *a* elevado à potência *b*, do tipo *double*. *a*(base) e *b*(potência) são os argumentos para este método.
`public static long round(double a)`
Retorna o valor do tipo *long* mais próximo do argumento *a*, do tipo *double*, informado. É um método *overloaded*. Também pode receber um argumento do tipo *float* e neste caso, retorna o valor do tipo *int* mais próximo.
`public static double sqrt(double a)`
Retorna a raiz quadrada do argumento *a* informado.
`public static double sin(double a)`
Retorna o seno trigonométrico de um dado ângulo *a*, argumento deste método.
`public static double toDegrees(double angrad)`
Retorna o valor aproximado em graus, equivalente ao valor em radianos passado no argumento.
`public static double toRadians(double angdeg)`
Retorna o valor aproximado em radianos, equivalente ao valor em graus passado no argumento.

Tabela 1: Alguns métodos da classe Math

A seguinte classe demonstra como estes métodos são usados:

```
class MathDemo {
   public static void main(String args[]) {
      System.out.println("absolute value of -5: " +
                                                Math.abs(-5));
      System.out.println("absolute value of 5: " +
```

```
                                                    Math.abs(-5));
        System.out.println("random number(max value is 10): " +
                                                     Math.random()*10);
        System.out.println("max of 3.5 and 1.2: " +
                                                   Math.max(3.5, 1.2));
        System.out.println("min of 3.5 and 1.2: " +
                                                   Math.min(3.5, 1.2));
        System.out.println("ceiling of 3.5: " + Math.ceil(3.5));
        System.out.println("floor of 3.5: " + Math.floor(3.5));
        System.out.println("e raised to 1: " + Math.exp(1));
        System.out.println("log 10: " + Math.log(10));
        System.out.println("10 raised to 3: " + Math.pow(10,3));
        System.out.println("rounded off value of pi: " +
                                                  Math.round(Math.PI));
        System.out.println("square root of 5 = " + Math.sqrt(5));
        System.out.println("10 radian = " + Math.toDegrees(10) +
                                                     " degrees");
        System.out.println("sin(90): " +
                                    Math.sin(Math.toRadians(90)));
    }
}
```

Esta é a saída desta classe. Sinta-se livre para experimentar com novos argumentos.

```
absolute value of -5: 5
absolute value of 5: 5
random number(max value is 10): 4.0855332335477605
max of 3.5 and 1.2: 3.5
min of 3.5 and 1.2: 1.2
ceiling of 3.5: 4.0
floor of 3.5: 3.0
e raised to 1: 2.7182818284590455
log 10: 2.302585092994046
10 raised to 3: 1000.0
rounded off value of pi: 3
square root of 5 = 2.23606797749979
10 radian = 572.9577951308232 degrees
sin(90): 1.0
```

3. Classes *String* e *StringBuffer*

3.1. Classe String

A Classe *String* que encontramos no Java SDK representa combinações de caracteres literais. Diferente de outras linguagens de programação como C ou C++, strings podem ser representadas utilizando-se um array de caracteres ou simplesmente a Classe *String*. Tenha em mente que um objeto *String* é diferente de um array de caracteres.

3.2. Construtores de String

A Classe *String* possui 12 construtores com as seguintes assinaturas:

```
public String(byte [] bytes)
public String(byte [] ascii, int hibyte)
public String(byte [] bytes, int offset, int length)
public String(byte [] ascii, int hibyte, int offset, int length)
public String(byte [] byteslength, String charsetName)
public String(byte [] bytes, String charsetName)
public String(char [] value)
public String(char [] value, int offset, int count)
public String(int [] codePoints, int offset, int count)
public String(String original)
public String(StringBuffer buffer)
public String(StringBuilder builder)
```

Para conhecermos alguns destes construtores, considere a classe abaixo:

```
class StringConstructorsDemo {
   public static void main(String args[]) {
      String s1 = new String();  // cria uma String vazia
      char chars[] = { 'h', 'e', 'l', 'l', 'o'};
      String s2 = new String(chars); // s2 = "hello";
      byte bytes[] = { 'w', 'o', 'r', 'l', 'd' };
      String s3 = new String(bytes);     // s3 = "world"
      String s4 = new String(chars, 1, 3);
      String s5 = new String(s2);
      String s6 = s2;
      System.out.println(s1);
      System.out.println(s2);
      System.out.println(s3);
      System.out.println(s4);
      System.out.println(s5);
      System.out.println(s6);
   }
}
```

A execução desta classe mostrará o seguinte resultado:

```
hello
world
ell
hello
hello
```

3.3. Métodos da Classe String

A seguir, uma lista de alguns métodos de *String*.

Métodos da classe String
`public char charAt(int index)`
Retorna o caracter localizado na posição especificada pelo argumento *index*, que indica a posição do caracter no array de caracteres representado.
`public int compareTo(String anotherString)`
Compara uma string com outra string especificada no argumento. Retorna um valor negativo se a string possuir um valor léxico menor que a string passada como argumento, 0 se ambas as strings tiverem o mesmo valor léxico e um valor positivo se a string tiver um valor léxico superior ao da string passada como argumento do método.
`public int compareToIgnoreCase(String str)`
Funciona como o método compareTo ignorando o padrão *case sensitive* de Java, isto é, maiúsculas e minúsculas são indiferentes, apresentando o mesmo valor léxico.
`public boolean equals(Object anObject)`
Retorna verdadeiro se a string possuir a mesma seqüência de caracteres do argumento *Object* passado para este método, argumento este que deve ser um objeto do tipo String. De outra maneira, se for passado um argumento que não seja do tipo *String* ou se não tiver a mesma sequência de símbolos da string, o método irá retornar *false*.
`public boolean equalsIgnoreCase(String anotherString)`
Funciona como o método *equals* ignorando o padrão *case sensitive* de Java, isto é, maiúsculas e minúsculas são indiferentes, apresentando o mesmo valor léxico.
`public void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)`
Obtém os caracteres do array de caracteres da string compreendidos no intervalo definido pelos argumentos *srcBegin* (na posição do primeiro caracter no array) e *srcEnd* (na posição do último caracter no array) copiando estes caracteres para o array *dst* iniciando na posição *dstBegin*.
`public int length()`
Retorna o comprimento da string.
`public String replace(char oldChar, char newChar)`
Retorna a string substituindo todas as ocorrências de *oldChar* nesta string por *newChar*.
`public String substring(int beginIndex, int endIndex)`
Retorna a substring da string iniciando na posição especificada no argumento *beginIndex* até a posição especificada no argumento *endIndex*.
`public char[] toCharArray()`
Retorna o array de caracteres da string.
`public String trim()`
Retorna a string suprimindo todos os seus espaços em branco.
`public static String valueOf(-)`
Este método recebe um argumento de tipo simples como boolean, integer ou character, ou ainda um Object e retorna a *String* equivalente ao argumento passado para o método.

Tabela 2: Alguns métodos da classe String

Observe como estes métodos são utilizados na seguinte classe:

```
class StringDemo {
   public static void main(String args[]) {
      String name = "Jonathan";
      System.out.println("name: " + name);
      System.out.println("3rd character of name: " + name.charAt(2));
      System.out.println("Jonathan compared to Solomon: " +
         name.compareTo("Solomon"));
      System.out.println("Solomon compared to Jonathan: " +
         "Solomon".compareTo("Jonathan"));
      System.out.println("Jonathan compared to jonathan: " +
         name.compareTo("jonathan"));
      System.out.println("Jonathan compared to jonathan (ignore " +
         "case): " + name.compareToIgnoreCase("jonathan"));
      System.out.println("Is Jonathan equal to Jonathan? " +
         name.equals("Jonathan"));
      System.out.println("Is Jonathan equal to jonathan? " +
         name.equals("jonathan"));
      System.out.println("Is Jonathan equal to jonathan (ignore " +
         "case)? " + name.equalsIgnoreCase("jonathan"));
      char charArr[] = "Hi XX".toCharArray();
      "Jonathan".getChars(0, 2, charArr, 3);
      System.out.print("content of charArr after getChars method: ");
      System.out.println(charArr);
      System.out.println("Length of name: " + name.length());
      System.out.println("Replace a's with e's in name: " +
         name.replace('a', 'e'));
      System.out.println("A substring of name: " +
         name.substring(0, 2));
      System.out.println("Trim \"  a b c d e f   \": \"" +
         "  a b c d e f   ".trim() + "\"");
      System.out.println("String representation of boolean " +
         "expression 10>10: " + String.valueOf(10>10));
      System.out.println("String representation of boolean " +
         "expression 10<10: " + (10<10));
      System.out.println("name: " + name);
   }
}
```

A execução desta classe mostrará o seguinte resultado:

```
name: Jonathan
3rd character of name: n
Jonathan compared to Solomon: -9
Solomon compared to Jonathan: 9
Jonathan compared to jonathan: -32
Jonathan compared to jonathan (ignore case): 0
Is Jonathan equal to Jonathan? true
Is Jonathan equal to jonathan? false
Is Jonathan equal to jonathan (ignore case)? true
content of charArr after getChars method: Hi Jo
Length of name: 8
Replace a's with e's in name: Jonethen
A substring of name: Jo
Trim "  a b c d e f   ": "a b c d e f"
String representation of boolean expression 10>10: false
String representation of boolean expression 10<10: false
name: Jonathan
```

3.4. Classe StringBuffer

Uma vez que um objeto do tipo *String* foi criado, não pode mais ser modificado. Um objeto *StringBuffer* é similar a um objeto *String*, exceto pelo fato que o objeto *StringBuffer* é mutável, ou seja, pode ser modificado. Reforçando: um objeto *String* é imutável, objetos *StringBuffer* são mutáveis. Seus comprimentos e conteúdos podem ser modificados por meio da chamada de alguns métodos.

Aqui estão alguns dos métodos da Classe *StringBuffer*. Para maiores informações consulte a documentação da API Java.

Métodos da classe StringBuffer
`public int capacity()`
Retorna a capacidade atual do objeto *StringBuffer*.
`public StringBuffer append(-)`
Anexa a representação em string do argumento passado para o método ao objeto *StringBuffer*. O argumento passado para este método pode ser de um dos seguintes tipos: *boolean, char, char [], double, float, int, long, Object, String and StringBuffer*. Possui outras versões *overloaded*.
`public char charAt(int index)`
Retorna o caracter localizado na posição especificada no argumento index.
`public void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)`
Obtém os caracteres do objeto compreendidos no intervalo definido entre a posição definida pelo argumento *srcBegin* e a posição definida pelo argumento *srcEnd* e copia estes caracteres para o array definido pelo argumento *dst* iniciando na posição *dstBegin*.
`public StringBuffer delete(int start, int end)`
Exclui os caracteres dentro do intervalo especificado.
`public StringBuffer insert(int offset, -)`
Insere a representação em string do segundo argumento na posição especificada pelo argumento offset. É um método *overloaded*. São tipos possíveis para o segundo argumento: *boolean, char, char [], double, float, int, long, Object and String*. Ainda possui outras versões *overloaded*.
`public int length()`
Retorna o número de caracteres no objeto *StringBuffer*.
`public StringBuffer replace(int start, int end, String str)`
Substitui os caracteres do objeto *StringBuffer* compeendidos no intervalo definido pelos dois primeiros argumentos (*start* e *end*) pelos caracteres do terceiro argumento (*str*).
`public String substring(int start, int end)`
Retorna a substring do objeto iniciando na posição definida pelo argumento *start* e terminando na posição definida pelo argumento *end*.
`public String toString()`
Converte o objeto *StringBuffer* para *String*.

Tabela 3: Alguns métodos da classe StringBuffer

A seguinte classe demonstra o uso destes métodos:

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

```
            StringBuffer sb = new StringBuffer("Jonathan");
            System.out.println("sb = " + sb);
            System.out.println("capacity of sb: " + sb.capacity());
            System.out.println("append \'O\' to sb: " +
               sb.append("O"));
            System.out.println("sb = " + sb);
            System.out.println("3rd character of sb: " +
               sb.charAt(2));
            char charArr[] = "Hi XX".toCharArray();
            sb.getChars(0, 2, charArr, 3);
            System.out.print("getChars method: ");
            System.out.println(charArr);
            System.out.println("Insert \'jo\' at the 3rd cell: " +
               sb.insert(2, "jo"));
            System.out.println("Delete \'jo\' at the 3rd cell: " +
               sb.delete(2,4));
            System.out.println("length of sb: " + sb.length());
            System.out.println("replace: " +
               sb.replace(3, 9, " Ong"));
            System.out.println("substring (1st three characters): " +
               sb.substring(0, 3));
            System.out.println("implicit toString(): " + sb);
         }
      }
```

A execução desta classe mostrará o seguinte resultado:

```
sb = Jonathan
capacity of sb: 24
append 'O' to sb: JonathanO
sb = JonathanO
3rd character of sb: n
getChars method: Hi Jo
Insert 'jo' at the 3rd cell: JojonathanO
Delete 'jo' at the 3rd cell: JonathanO
length of sb: 9
replace: Jon Ong
substring (1st three characters): Jon
implicit toString(): Jon Ong
```

Sinta-se à vontade para experimentar esta classe modificando os argumentos, a melhor forma de aprender é através da prática.

3.5. Mutabilidade

Falamos de mutabilidade de objetos e foi dito que um objeto da classe *String* não pode ter o seu valor modificado, entretanto observaremos as seguintes instruções:

```
String objString = new String("Hello");
objString = objString.concat(" World!!!");
System.out.println(objString);
```

Teremos como resultado o texto **Hello World!!!**, então significa que o objeto da classe *String* foi modificado certo? **Errado**, ocorreu na segunda instrução a destruição e a recriação do objeto. Analisaremos a seguinte classe:

```
class MutabilityDemo {
    public static void main(String args[]) {
        String objString = new String("Hello");
        System.out.println(objString + " : " + objString.hashCode());
```

```
            objString = objString.concat(" World!!");
            System.out.println(objString + " : " + objString.hashCode());

            StringBuffer objStringBuffer = new StringBuffer("Hello");
            System.out.println(objStringBuffer + " : " +
               objStringBuffer.hashCode());
            objStringBuffer = objStringBuffer.append(" World!!");
            System.out.println(objStringBuffer + " : " +
               objStringBuffer.hashCode());
        }
    }
```

E teremos como resultado da execução desta:

```
Hello : 69609650
Hello World!! : 22678948
Hello : 17523401
Hello World!! : 17523401
```

O método *hashCode()* gera automaticamente um **OID** (*ObjectID*) único para cada objeto criado, observe que no caso do objeto da classe *String* este número mudou, ou seja, foi criado um novo objeto, e no caso do objeto da classe *StringBuffer* este número permanece o mesmo, demonstrando assim a imutabilidade deste.

4. Classes Wrapper

Obviamente, os tipos primitivos como *int*, *char* e *long* não são objetos. Desta forma, variáveis destes tipos não podem acessar métodos da classe *Object*. Somente objetos reais, que são declarados para serem referenciados como tipos de dados, podem acessar métodos da classe *Object*. Todavia, existem casos em que você precisará de uma representação em objeto de variáveis de tipos primitivos para usar métodos internos de Java. Por exemplo, você pode querer adicionar variáveis de tipo primitivo em uma *Collection* de objetos. É aqui que a classe *wrapper* entra. Classes *wrapper* são, simplesmente, representações em objetos de variáveis que não são objetos, como as de tipos primitivos. Aqui está uma lista de classes *wrapper*:

Tipos Primitivos de Dados	***Classes Wrapper Correspondentes***
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

Tabela 4: Tipos de dados primitivos e suas classes wrapper correspondentes

Os nomes das diferentes classes wrapper são fáceis de lembrar, porque são muito similares aos tipos de dados primitivos que representam. Note também que as classes *wrapper* iniciam com letra maiúscula, comparando-as com os tipos primitivos.

Aqui está um exemplo do uso da classe wrapper para o tipo primitivo *boolean*.

```
class BooleanWrapperDemo {
   public static void main(String args[]) {
      boolean booleanVar = 1>2;
      Boolean booleanObj = new Boolean("TRue");
      Boolean booleanObj2 = new Boolean(booleanVar);
      System.out.println("booleanVar = " + booleanVar);
      System.out.println("booleanObj = " + booleanObj);
      System.out.println("booleanObj2 = " + booleanObj2);
      System.out.println("compare 2 wrapper objects: " +
         booleanObj.equals(booleanObj2));
      booleanVar = booleanObj.booleanValue();
      System.out.println("booleanVar = " + booleanVar);
   }
}
```

A execução desta classe mostrará o seguinte resultado:

```
booleanVar = false
booleanObj = true
booleanObj2 = false
compare 2 wrapper objects: false
booleanVar = true
```

5. Classes *Process* e *Runtime*

5.1. Classe Process

A classe *Process* define métodos para manipulação de processos que podem, por exemplo, terminar a execução de um processo, iniciar a execução de processos e verificar o status de processos. Esta classe representa as instruções quando são executadas. Aqui estão alguns métodos desta classe:

Métodos da classe *Process*
`public abstract void destroy()`
Finaliza os processos em execução.
`public abstract int waitFor() throws InterruptedException`
Não sai até que o processo termine.

Tabela 5: Alguns métodos da classe Process

5.2. Classe Runtime

Enquanto a classe *Process* representa os processos em execução, a classe *Runtime* representa o ambiente de execução. Dois métodos importantes desta classe são: *getRuntime* e *exec*.

Métodos da classe Runtime
`public static Runtime getRuntime()`
Retorna o ambiente de execução associado a aplicação Java atual.
`public Process exec(String command) throws IOException`
Executa o comando passado como argumento em *command*. Permite-lhe executar novos processos.

Tabela 6: Alguns métodos da classe RunTime

5.3. Abrindo o Editor de Registro

Esta classe abre o editor de registro sem que seja necessário, explicitamente, digitar o comando na linha de comando:

```
class RuntimeDemo {
   public static void main(String args[]) {
      Runtime rt = Runtime.getRuntime();
      Process proc;
      try {
         proc = rt.exec("regedit");
         proc.waitFor();
      } catch (Exception e) {
         System.out.println("regedit is an unknown command.");
      }
   }
}
```

A execução desta classe mostrará o seguinte resultado:

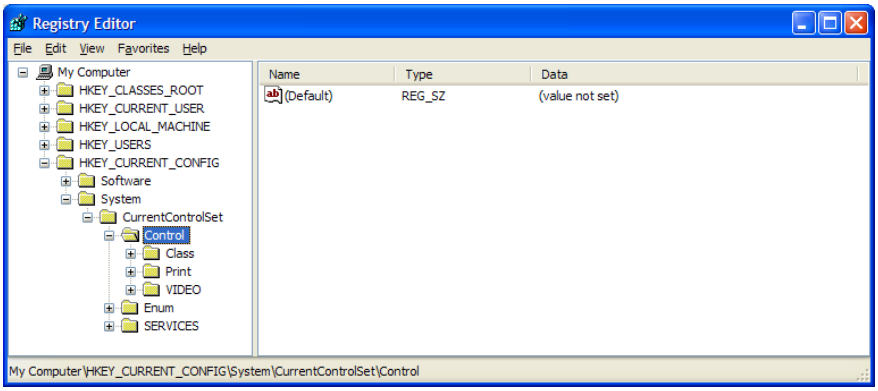

Figura 1: O editor de registro aberto

6. Classe *System*

A classe *System* oferece muitos atributos e métodos úteis como a entrada padrão (*input*), a saída padrão (*output*) e um método utilitário para copiar rapidamente partes de um array. Aqui estão alguns métodos interessantes desta classe. Note que todos os métodos dessa classe são *static*.

Métodos da classe System
`public static void arraycopy(Object src, int srcPos, Object dest, int destPos, int length)`
Copia os ítens do array s*rc*, passado como argumento para o método, especificados no argumento length iniciando na posição especificada no argumento *srcPos* para o array *dest*, terminando na posição especificada no argumento *destPos*. É mais rápido do que utilizar um algoritmo próprio para realizar a cópia.
`public static long currentTimeMillis()`
Retorna a diferença entre o tempo atual (data/hora) e 1º de Janeiro de 1970 (UTC). O valor de retorno se dá em milissegundos.
`public static void exit(int status)`
Termina a execução da Máquina Virtual Java (JVM) atual. Um valor diferente de zero para o status, por convenção, indica uma saída anormal.
`public static void gc()`
Executa o *garbage collector*, que irá liberar espaços em memória que não estão mais sendo utilizados.
`public static void setIn(InputStream in)`
Troca o *stream* associado a *System.in*, que por padrão, refere-se ao teclado.
`public static void setOut(PrintStream out)`
Troca o *stream* associado a *System.out*, que por padrão, refere-se ao console (tela).

Tabela 7: Alguns metodos da classe System

Aqui está uma demonstração do uso de alguns destes métodos:

```
import java.io.*;

class SystemDemo {
    public static void main(String args[]) throws IOException {
        long startTime, endTime;
        System.setOut(new PrintStream("C:/tempOut.txt"));
        int arr1[] = new int[5000000];
        int arr2[] = new int[5000000];
        for (int i = 0; i < arr1.length; i++) {
            arr1[i] = i + 1;
        }
        startTime = System.currentTimeMillis();
        for (int i = 0; i < arr1.length; i++) {
            arr2[i] = arr1[i];
        }
        endTime = System.currentTimeMillis();
        System.out.println("Manual: " + (endTime-startTime) + " ms.");
        startTime = System.currentTimeMillis();
        System.arraycopy(arr1, 0, arr2, 0, arr1.length);
        endTime = System.currentTimeMillis();
        System.out.println("Automatic: " + (endTime-startTime) + " ms.");
    }
}
```

A execução desta classe pode variar. O resultado contido no arquivo em C:\tempOut.txt, pode ser por exemplo:

```
Manual: 47 ms.
Automatic: 31 ms.
```

Parceiros que tornaram JEDI™ possível

Instituto CTS
Patrocinador do DFJUG.

Sun Microsystems
Fornecimento de servidor de dados para o armazenamento dos vídeo-aulas.

Java Research and Development Center da Universidade das Filipinas
Criador da Iniciativa JEDI™.

DFJUG
Detentor dos direitos do JEDI™ nos países de língua portuguesa.

Banco do Brasil
Disponibilização de seus *telecentros* para abrigar e difundir a Iniciativa JEDI™.

Politec
Suporte e apoio financeiro e logístico a todo o processo.

Borland
Apoio internacional para que possamos alcançar os outros países de língua portuguesa.

Instituto Gaudium/CNBB
Fornecimento da sua infra-estrutura de hardware de seus servidores para que os milhares de alunos possam acessar o material do curso simultaneamente.