Unidade 1 – Parte 2

STRINGS

O trabalho com strings em java pode ser feito através de duas classes: *String* e *StringBuffer*. As strings implementadas utilizando estas classes são consideradas objetos e são representadas usando o Unicode.
A classe String possui vários construtores. A forma mais fácil de criação e inicialização é a seguinte:

String str1 = "Isto é uma string";
Ou
String str1 = new String("Isto é uma String");

Para a comparação de Strings é errado fazer:

if (string1 == "Y") ... // Errado

Utilizar o método equals:

if (string1.equals(string2)) ...

Principais métodos (String):

A classe *String* possui uma vasta gama de métodos para sua manipulação, o que permite realizar muitas ações sobre ela, como a concatenação, a extração de uma parte da string ou de apenas um caracter, a comparação de valores e outros.

Métodos:

- charAt(int index)
- compareToIgnoreCase(String str)
- concat(String str)
- copyValueOf(char[] data, int offset, int count)
- equalsIgnoreCase(String anotherString)
- indexOf(String str)
- length( )
- replace(char oldChar, char newChar)
- substring(int beginIndex, int endIndex)
- toUpperCase( )
- trim( )

Exemplo de utilização dos métodos.

```
String s = “Teste”;
s = s.concat(“texto”);
int i = s.length( );
s = s.toUpperCase( );
// Usar método equals
```

Toda a documentação da classe String pode ser visualizada no site oficial do Java

WRAPPER CLASSES

Tipos de dados primitivos não são objetos em java. Então, para manipular esses valores como objetos, o pacote *java.lang* disponibiliza uma classe que encapsula os métodos para cada tipo de dados primitivo.

Construtores

Cada classe (exceto a Character que possui apenas um construtor) tem dois construtores:

- Um construtor que pega um tipo primitivo e retorna um objeto de classe wrapper correspondente.
- Um construtor que pega um objeto do tipo S*tring*, que representa o valor primitivo e retorna um objeto da classe wrapper correspondente.

Ex:

Integer intObj = ~~new Integer (2001);~~ //deprecated API

Ou

Integer intObj = ~~new Integer("2001");~~ //deprecated API

Integer intObj = Integer.getInteger("2001");

Integer intObj = Integer.valueOf("10");

Integer intObj = Integer.valueOf(10);

Métodos comuns entre as classes

Estas classes possuem métodos que são comuns em todas elas, com exceção da classe Character.

Todas as classes possuem um método *valueOf (String)*, que retorna um objeto wrapper correspondente ao tipo primitivo representado pela string passada como parâmetro.

Ex:

Double dbObj = Double.valueOf ("3.0");

Todas classes possuem um método *toString ( )*. Este método retorna um objeto do tipo String representando o valor do dado da variável.

Ex:

String boolStr = boolObj.toString( ); // "true"

Todas as classes possuem um método *type*Value ( ) que retorna o valor primitivo no objeto wrapper. Existem métodos para todos os tipos primitivos.

Ex:

```
int i = intObj.intValue( );  // 2001
```

Todas as classes possuem um método *equals( )*. Esse método compara dois objetos wrapper para verificar equivalência de valores.

Ex:

```
Boolean intTest = intObj.equals(intObj2);
```

Documentação extra sobre wrapper classes podem ser visualizadas no site oficial do Java.

RECEBENDO PARÂMETROS DE ENTRADA

Muitas vezes é necessário passar parâmetros na linha de comando de chamada do programa. Os parâmetros são passados dentro de um array de String definido na assinatura padrão do método main.
Exemplo: O exemplo abaixo imprime a concatenação dos dois primeiros parâmetros passados.

```
public class parametros
{
        public static void main (String args[])
        {
                System.out.println(args[0] + args[1]);
        }
}
```

TRATAMENTO DE EXCEÇÕES

Durante a execução de um programa, muitos tipos de erros podem ocorrer em diversos níveis de severidade. A classe *Exception* provê uma maneira limpa de checar possíveis erros que ocorrem na execução de um código.

try e catch.

As exceções podem ser capturadas colocando o código a ser executado dentro de uma cláusula *try*. Se alguma exceção é capturada a cláusula *catch* é toda executada. Qualquer número de cláusulas *catch* pode ser associado com um único *try*.

Exemplo:

```
try {
        // código que pode ocasionar uma exception
}
catch{
        // tratamento do erro
    }
```

A declaração **finally** é utilizada para definir o bloco que irá ser executado tendo ou não uma exception, isto após o uso da declaração de try e catch. Esta cláusula geralmente é utilizada para limpar objetos e liberar recursos, como arquivos abertos.

```
try {
        // código que pode ocasionar uma exception
}
catch{
        // tratamento do erro
    }
finally {
        // código
     }
```

Outra possibilidade é a declaração throws. Esse processo pode ser aplicado a situações onde não há necessidade de tratamento do erro.
Exemplo:

```
void retorna( ) throws MalformedURLException, SQLException
{
  //código…
}
```

Exceptions mais comuns:

ArithmeticException - int i = 12 / 0

NullPointerException - ocorre quando utilizo um objeto que não foi instanciado.

NegativeArraySizeException - ocorre quando é atribuído um valor nulo para um array.

ArrayIndexOutOfBoundsException - ocorre quando tento acessar um elemento do array que não existe.

ENTRADA E SAÍDA PADRÃO

Para gravar na saída padrão do sistema:

```
System.out.println("Gravando");
```

Para ler uma linha da entrada padrão do sistema:

```
InputStreamReader ISRin=new InputStreamReader(System.in);
BufferedReader in=new BufferedReader(ISRin);
String s=in.readLine();
```

Exceções de I/O

Para o tratamento de exceções de qualquer leitura ou gravação de I/O utilizamos a classe I/O Exception. Segue abaixo um exemplo de tratamento padrão de possíveis exceções em comandos de I/O.

```
 try
{
        int i=System.in.read();
       System.out.println("Received byte "+i);
}
 catch (IOException e)
{
          //Trata o erro
}
```

ARRAY

Um *array* é uma estrutura de dados que define uma coleção ordenada de número fixo de um tipo homogêneo de dados. Isso significa que todos os elementos de um array são do mesmo tipo. O tamanho do array é fixo e não pode ser aumentado para acomodar mais elementos.

No Java, arrays são objetos. Eles podem conter tipos primitivos ou tipos criados. Arrays simples são arrays de uma dimensão, com uma lista simples de valores. Mas, como arrays podem guardar qualquer tipo de dados, eles podem guardar outros arrays. Isso resulta em um array de arrays.

Ex:

```
      < element type > [ ] < array name >
int [ ] meuArray;
```

ou

```
      < element type > < array name > [ ];
int meuArray [ ];
```

Construção

Um array pode ser construído para um número específico de elementos utilizando-se do operador new.

Ex:

```
MeuArray = new int [ 10 ] ;
```

Inicialização

O Java provê uma maneira de declarar, construir e explicitar valores iniciais para um array em uma única linha de comando.

Ex:

```
String animais [ ] = { “Macaco”,”Jacaré”,”Gato” };
```

Usando um array

Todo o array é referenciado utilizando-se do nome do mesmo, mas elementos individuais são acessados utilizando-se do operador [ ]. Cada elemento de um array é tratado como uma simples variável do tipo do array. O índice inicial do array é zero (0).

Ex:

```
for (int i=0; i < 2; i++)
      {
            System.out.println(“Nome do animal = “ + animais [ i ];
      }
```

Arras Multidimensionais

Desde que um elemento de um array pode ser um objeto e um array é um tipo de objeto, arrays podem referenciar eles mesmos. Em java, um array de array tem a seguinte sintaxe:

< element type > [ ] [ ] ... [ ];

ou

< element type > < array name > [ ] [ ] ... [ ];

Pode-se combinar a declaração com a construção do mesmo.

Ex:

int [ ] [ ] meuArray = new int [4] [5];

O acesso as variáveis deste tipo de array tem a seguinte sintaxe:

System.out.println ("Número = "+ meuArray [1] [2]);

GARBAGE COLLECTOR

Objetos ocupam memória. O garbage collector é um mecanismo para limpar da memória os objetos que não estão mais em uso. O java utiliza o garbage collector automaticamente. Este mecanismo garante também que um objeto nunca vai ser deletado enquanto ainda pode ser acessado. Ele roda em background.

Invocando o Garbage Collector

O comando *System.gc( )* é utilizado para invocar o garbage collector explicitamente.

Certos aspectos sobre o garbage collector devem ser lembrados:

- Não há garantias que todos os objetos não mais usados sejam deletados;
- Os objetos não são limpos de acordo com uma ordem de criação.

Maiores informações sobre a linguagem de programação Java podem ser acessadas no site (https://www.java.com/pt-BR/).