Cartão de Referência

Tiger (JSE 5.0)

versão 1.0

Introdução

Este cartão contém uma referência rápida para os conceitos que foram introduzidos na Java SE versão 5.0.

Por que Java SE 5.0 ao invés de Java 1.5?

A versão final do Java Standard Edition 5.0 incluiu um grande número de JSRs e outras atualizações.

Versão	Codinome	Data
J2SE 1.4	Merlin	13/2/2002
J2SE 1.4.1	Hopper	16/10/2002
J2SE 1.4.2	Mantis	29/5/2003
Java SE 5.0	Tiger	30/09/2004
Java SE 6.0	Mustang	2006
Java SE 7.0	Dolphin	2007

Boxing e Unboxing

Cada tipo de dado primitivo da linguagem Java tem uma classe Wrapper correspondente. Esta classe engloba um valor único e imutável. Por exemplo, a classe Wrapper java.lang.Long cobre (wrap) um valor long e a classe java.lang.Character cobre um valor char.

As classes Wrapper são úteis sempre que for conveniente tratar um dado primitivo como se ele fosse um objeto, como acontece com os elementos da maioria das classes de coleções.

```
Integer fator1 = new Integer(15);
int fator2 = 10;
int produto = fator1.intValue() * fator2;
```

Parâmetros da printf

```
%d - decimal   - System.out.printf("Número %f\n", 0.3 + 0.3);
%f - float     - System.out.printf("Número %d\n", 10/3);
%s - String    - System.out.printf("Nome %s\n", "Fernando");
%c - Caracter - System.out.printf("Nome %c\n", 'F');

%n - Salto Linha              %% - Mostrar %
%b - lógico                   %h - hashCode
%o - octal                    %x - hexadecimal
%e - Notação científica       %g - float formato computador
%a - hexadecimal flutuante    %t - Data
```

Scanner

```
import java.util.Scanner;
public class Hello {
  public static void main(String [] args) {
      Scanner in = new Scanner(System.in);
      System.out.print("Informe um número: ");
      int i = in.nextInt();
      System.out.printf("Seu número é: %d", i);
  }
}
```

Métodos básicos da Scanner

```
next(): String   nextLine(): String
nextBoolean()    nextByte()
nextInt()        nextLong()
nextDouble()     nextFloat()
nextShort()      hasNextXXXX(): boolean*
```

*se o valor informado foi do tipo XXX selecionado

forEach

Foi projetado para iteração em coleções e arrays.

```
String[] bemVindo = new String[3];
bemVindo[0] = "Bem Vindo";
bemVindo[1] = "ao Mundo Java";
bemVindo[2] = "por Fernando Anselmo";
for (String b : bemVindo)
  System.out.println(b);
```

Enumerations

A definição de um conjunto fixo de constantes é uma necessidade comum. Alguns padrões foram definidos para resolver este problema, porém todos estes padrões tinham os seus problemas. Em Java isso foi resolvido através do uso de uma lista enumerada.

```
public class Hello {
  private static enum Status {continuar, vencer, perder};
  public static void main(String [] args) {
      Status status;
      int ocorreu = Integer.parseInt(args[0]);
      switch (ocorreu) {
          case 1: status = Status.continuar; break;
          case 2: status = Status.vencer; break;
          default: status = Status.perder; break;
      }
      System.out.println(status);
  }
}
```

import static

É possível importar métodos estáticos para limpar o código.

```
import static java.lang.Math.PI;
import static java.lang.Math.cos;
public class Hello {
  public static void main(String [] args) {
      double r = cos(PI * 10);
      System.out.printf("Numero %f%n", r);
  }
}
```

Reflections

```
import java.lang.reflect.*;
import static java.lang.System.out;
import static java.lang.System.err;
public class Nova {
  public static void main(String[] args) {
    Class type;
    try {
      type = Class.forName("testecurso.Main");
    } catch (ClassNotFoundException ex) {
    }
    out.println("Classe: " + type.getSimpleName());
    Class superclass = type.getSuperclass();
    Method[] metodos = type.getDeclaredMethods();
    for (Method met: metodos)
      out.println("Método: " + met.getName());
  }
}
```

Anotations

Tipos - Qualquer primitivo, String, Enum, outra anotation ou array de um dos tipos.

Uso de array

```
cmpArray={"valor1", "valor2"}
cmpArray="valorUnico
```

Meta Anotations

```
@Documented - Auto documentação
@Target - Define a classe a ser aplicada
@Retention - Uso da anotação: SOURCE, CLASS ou RUNTIME
@Deprecated - Método depreciado
@Override - Informar sobre um polimorfismo
@SuppressWarnings - Ignonar avisos
@Inherited - A anotação deve ser herdada
```

Forma

```
@interface Cabecalho {
  String criadoPor();
  String naData();
  int revisao();
}
```

Utilização

```
@Cabecalho(
  criadoPor="Fernando Anselmo",
  naData="jul/07",
  revisao=3
)
```

Generics

Generics correspondem a um recurso de abstração dos tipos de dados através do uso de tipos parametrizados.

```
class TipoD <T> {
  private T tipo;
  public void setTipo(T tipo) { this.tipo = tipo; }
  public T getTipo() { return tipo; }
}
public class Generica {
  public static void main(String[] args) {
    TipoD <String> tStr = new TipoD<String>();
    tStr.setTipo("Fernando Anselmo");
    System.out.println("Tipo String:" + tStr.getTipo());
    TipoD <Integer> tInt = new TipoD<Integer>();
    tInt.setTipo(1234);
    System.out.println("Tipo Inteiro:" + tInt.getTipo());
  }
}
```

Usado em Coleções

```
List<String> nomes = new ArrayList<String>();
nomes.add(new StringBuffer("Joaquim")); // Erro
nomes.add("Joao");
nomes.add("Maria");
String primeiroNome = nomes.get(0).toUpperCase();
```

Usado em Métodos

```
public void filtrar(List<String> seq, char pLet){
  for (Iterator <String> i = seq.iterator(); i.hasNext();
     if (i.next().charAt(0) != pLetra) {
        i.remove();
     }
  }
}
```

Var args

Suporte a um número de argumentos variáveis em métodos.

```
public static double media(double ... valores) {
   double soma = 0.0;
   for (double valor : valores)
        soma += valor;
   return soma / valores.length;
}
```

Observações