élogos

CLASES BÁSICAS PARA LA GESTIÓN DE CADENAS Y FUNCIONES NUMÉRICAS

élogos

ÍNDICE

CLASES BÁSICAS PARA LA GESTIÓN DE CADENAS Y FUNCIONES NUMÉRICAS

Clases básicas para la gestión de cadenas y funciones numéricas

1. Manipulación de cadenas con la clase String

Gestión de cadenas

Podemos crear una cadena de forma sencilla de la siguiente manera:

String nombre = “Pedro”;

También, se pueden crear cadenas, a partir de la definición de un array de caracteres:

char [] b= {‘P’, ‘e’, ‘d’, ‘r’, ‘o’}; String nombre = new String(b);

Se permite concatenar String con el operador +:

String apellidos = “Lopez”;

String nombre_completo = nombre + “ “ + apellidos;

Siempre se puede preguntar por la longitud del String:

int longitud = nombre_completo.length();

System.out.println (“Hola”.length());//Imprimiría 4

System.out.println (new String = {‘H’, ’o’, ’l’,a’}.length(); //Imprimiría 4

Clase StringBuilder

Si se concatena una cadena de caracteres con otra, la cadena original no se modifica, sino que se crea una nueva cadena resultante de la unión de ambas, pues las cadenas de caracteres son objetos inmutables.

Si una aplicación tiene que realizar reiteradas modificaciones sobre una cadena, mediante la unión de nuevos caracteres, es preferible utilizar la clase StringBuider en vez de String.

La clase StringBuilder si permite modificar la cadena original y evita, por tanto, la generación de “basura” resultante de las sucesivas concatenaciones

Clases básicas para la gestión de cadenas y funciones numéricas

Ejemplo de uso de StringBuilder

StringBuilder str = new StringBuilder (“Altura ”);

Str.append(25);

String mensaje = str.toString(); // mensaje = “Altura 25” String str = “El tiene”+ edad +”años” ;

String str = new StringBuilder(“El tiene”) ;

str.append(edad); str.append(“años”); System.out.println(str.toString) ;

Métodos importantes de la clase String

Veamos los métodos más importantes de la clase String:

boolean <String>.equals(String): Compara 2 cadenas.

Ejemplo:

if (str1.equals(str2))

{

// se cumple

}

boolean <String>.equalsIgnoreCase(String2):

Compara 2 cadenas sin tener en cuenta mayúsculas ni minúsculas.

boolean <String>.startsWith(String s):

Si la cadena empieza por s.

boolean <String>.endsWith(String a):

Si la cadena termina por a.

int <String>.indexOf (char c):

Retorna el índice en que se encuentra la primera ocurrencia del carácter especificado.

int <String>.indexOf (String str):

Retorna el índice en que se encuentra la primera ocurrencia de la cadena especificada.

int <String>.indexOf (int index, char c):

Clases básicas para la gestión de cadenas y funciones numéricas

Retorna el índice en que se encuentra la primera ocurrencia del carácter especificado desde el índice especificado.

int <String>.indexOf (int index, String str):

Retorna el índice en que se encuentra la primera ocurrencia de la cadena especificado desde el índice especificado. Si retorna –1 quiere decir que no ha encontrado la cadena o el carácter.

String <String>.toLowerCase():

La cadena se pasa a minúsculas.

String <String>.toUpperCase():

La cadena se pasa a mayúsculas.

char <String>.charAt(int index):

Sirve para recoger un carácter en la posición indicada (en index).

Ejemplo de Métodos de la clase String

Vamos a ver estos últimos métodos con un ejemplo. Para ello vamos a crear la clase cadenas.java que va a evaluar la siguiente cadena "En Un lUgar de lA mAncha" , la cual nos indicará, cuántas letras 'A' mayúscula y 'a' minúscula tiene la cadena así como cuántas vocales en total.

```
public class cadenas {

public static void main(String args[]){

String cadena="En Un lUgar de lA mAncha";

//Declaramos la variable

//cadena como String

int mayusculas=0,minusculas=0,cont=0; // Declaramos las variables

// recorremos la cadena extrayendo el carácter de la posición i y preguntamos si

//coincide con 'A' o 'a' e incrementaremos sus respectivos contadores

for (int i=0; i<cadena.length(); i++){

if (cadena.charAt(i)=='A')

mayusculas++;
```

Clases básicas para la gestión de cadenas y funciones numéricas

```
else

if (cadena.charAt(i)=='a')

minusculas++;

}

// Pasamos toda la cadena a minúsculas

cadena=cadena.toLowerCase();

// Recorremos nuevamente la cadena preguntando si el carácter que se extrae es vocal

// a,e,i,o,u

// como ya hemos pasado toda lacadena aminúscula no habrá que preguntar por las //mismas vocales en mayúsculas

for (int i=0; i<cadena.length(); i++){

if (cadena.charAt(i)=='a'||cadena.charAt(i)=='e'||cadena.charAt(i)=='i'||cadena.charAt(i)=='o'||ca
dena.charAt(i)=='u')

cont++;

}

// Imprimiremos todos los contadores

System.out.println("Las a's son: "+minusculas); System.out.println("Las A's son: "+mayusculas);
System.out.println("Las vocales son: "+cont);

}

}
```

La salida sería: Las a's son: 2

Las A's son: 2

Las vocales son: 8

2. Operaciones matemáticas con la clase Math

Constantes de la clase Math

La clase Math contiene dos constantes definidas, el número PI y el número E

Para acceder a estas constantes de la clase Math, lo hacemos de la siguiente manera:

```
System.out.println("Pi vale " + Math.PI);
System.out.println("e  vale " + Math.E);
```

Al tratarse de constantes estáticas se puede acceder, como ya dijimos, a través del nombre de la clase sin necesidad de un objeto

Métodos de la clase Math

La clase Math tiene gran cantidad de métodos y diferentes versiones de cada uno de ellos con distintos tipos de parámetros.

Métodos trigonométricos

En las funciones trigonométricas los argumentos se expresan en radianes. Si tuviésemos un ángulo de 90º, se pasaría a radianes y luego se calcularía el seno, el coseno y la tangente:

```
double angulo = 90.0 * Math.PI/180.0;

System.out.println("cos(" + angulo + ") es " + Math.cos(angulo));

System.out.println("sin(" + angulo + ") es " + Math.sin(angulo));

System.out.println("tan(" + angulo + ") es " + Math.tan(angulo));
```

Métodos exponenciales, logarítmicos, potencia y raíz cuadrada

El método exponencial exp devuelve el número e elevado a una potencia:

```
System.out.println("exp(2.0) es " + Math.exp(2.0));

System.out.println("exp(10.0) es " + Math.exp(10.0));
```

El método log calcula el logaritmo natural (de base e) de un número

```
System.out.println("log(2.0) es " + Math.log(2.0));
```

Clases básicas para la gestión de cadenas y funciones numéricas

```
System.out.println("log(Math.E) es " + Math.log(Math.E));
```

El método potencia eleva un número x a la potencia y, se emplea pow(x, y):

```
System.out.println("pow(5.0, 3) es " + Math.pow(5.0,3));
```

Para hallar la raíz cuadrada de un número, se emplea la función sqrt:

```
System.out.println("La raíz cuadrada de " + x + " es " + Math.sqrt(99));
```

Aproximación de un número decimal

Si queremos definir un número real con un número determinado de números decimales, empleamos el método round. Por ejemplo, para definir los números x e y, con dos cifras decimales cada uno, lo haremos de la siguiente forma:

```
double x = 52.2144;

double y = .04394;

System.out.println(x + " es aprox. " + (double)Math.round(x*100)/100);

System.out.println(y + " es aprox. " + (double)Math.round(y*100)/100);
```

Salida

52.2144 es aprox. 52.21

0.04994 es aprox. 0.05

La función round devuelve un número entero int que es necesario convertir a double para efectuar la división entre 100. Si utilizamos el método floor en vez de round :

```
System.out.println(x + " es aprox. " + Math.floor(x*100)/100);

System.out.println(y + " es aprox. " + Math.floor(y*100)/100);
```

Salida

52.2144 es aprox. 52.21

0.04994 es aprox. 0.04

Clases básicas para la gestión de cadenas y funciones numéricas

El mayor y el menor entre dos números

Para hallar el mayor y el menor de dos números se emplean los métodos min y max que comparan números del mismo tipo.

```
int i = 8;

int j = -7;

double x = 92.3543;

double y = 0.2497;
```

// para hallar el menor de dos números

```
System.out.println("min(" + i + "," + j + ") es " + Math.min(i,j));

System.out.println("min(" + x + "," + y + ") es " + Math.min(x,y));
```

// para hallar el mayor de dos números

```
System.out.println("max(" + i + "," + j + ") es " + Math.max(i,j));

System.out.println("max(" + x + "," + y + ") es " + Math.max(x,y));
```

Números aleatorios

La clase Math define el método random, que devuelve un número aleatorio comprendido en el intervalo [0.0, 1.0).

```
System.out.println("Número aleatorio: " + Math.random());
```

Salida: 0.25991750935573255

Por ejemplo, si quisiéramos calcular un número aleatorio entre 0 y 1000, lo haríamos de la siguiente forma:

```
System.out.println("Número aleatorio: " + (int)(Math.random()*1000));
```

Salida: Número aleatorio: 122

Importaciones estáticas

Desde la versión Java 5 es posible realizar en un programa lo que se conoce como importaciones estáticas, que consiste en importar todos los miembros estáticos de una clase para poder hacer uso de los mismos sin necesidad de escribir el nombre de la clase delante.

Si en un programa vamos a utilizar varios métodos de la clase Math, puede resultar interesante importar los elementos estáticos para ahorrarnos la expresión Math. en la llamada a los métodos.

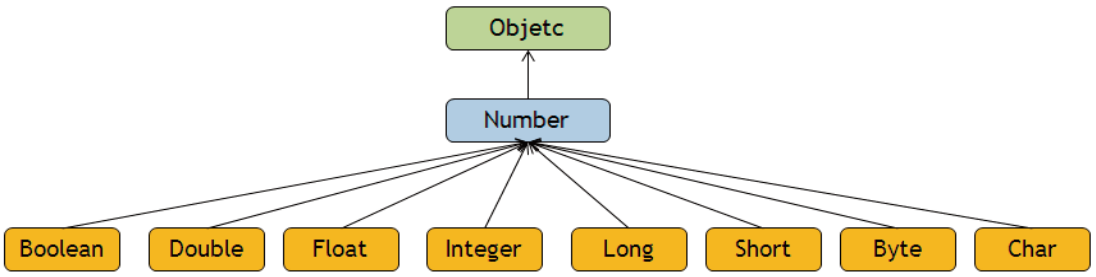
Para importar los miembros estáticos de una clase se emplea la expresión:

import static paquete.clase.*;

3. Clases de envoltorio y Clases runtime

Clases Envoltorio o Clases Wrapper

Clases envoltorios o Clases Wrapper son las clases que sirven para representar los tipos fundamentales de Java como objetos. Están organizados de forma jerárquica:



Todas estas clases tienen un constructor a partir de un tipo de dato fundamental.

Double (double d) Float (float f)

. . .

Char (char c)

Para realizar la creación del objeto:

double d = 3.14;

Double D = new Double(d); Float F = new Float (3.1f)

Para sacar el contenido de una Clase Wrapper se realizará con el siguiente método:

double <Double>.doubleValue();

float <Float>.floatValue();

int <Integer>.intValue();

Utilidad de objetos y métodos de conversión

La utilidad que tienen estos objetos, es pasar los tipos de datos fundamentales por referencia y poder modificar su valor. Además, en determinadas aplicaciones de manipulación de datos, solamente podemos trabajar con objetos, lo que requiere envolver el dato primitivo en un objeto. Este es el caso de las colecciones, que únicamente admiten objetos.

Los siguientes métodos permiten convertir un String en un tipo de dato fundamental:

static byte <Byte>.parseByte(String Text)

static int <Integer>.parseInt(String Text)

Clases Runtime (procesos)

Las Clases Runtime encapsulan un proceso del intérprete de la JVM, representa sólo procesos que están ejecutándose dentro de la JVM. Las clases Runtime no se pueden instanciar (crear objetos).

Sí se pueden obtener referencias al Runtime que se está ejecutando en el momento:

Static Runtime getRuntime()

Métodos

long totalMemory()

Memoria adquirida en el proceso en ejecución.

long freeMemory()

Memoria libre del proceso en ejecución.

void gc()

Recogida de basura (Garbage Collector).

4. Autoboxing y Unboxing

El autoboxing es un técnica utilizada por el compilador Java a partir de la versión Java 5, mediante la cual se puede asignar directamente un tipo primitivo a una variable de tipo envoltorio, por ejemplo, un int a un Integer. El compilador se encarga de convertir automáticamente el dato primitivo en el objeto

El autounboxing representa el proceso contrario, es decir, la conversión automática de un objeto en su tipo primitivo

Ejemplo:

```
Integer num=2; //el valor 2 es convertido automáticamente en Integer

Map map = new HashMap();

map.put("resultado", num+1);
```

Uso de Autoboxing y autounboxing

Cuando hay una incompatibilidad entre los tipos de referencia y los tipos de datos primitivos es cuando debemos usar el autoboxing, a pesar de que esto no es conveniente para cálculos científicos. Un Integer no es un sustituto para un int. El autoboxing y autounboxing simplemente ocultan las diferencias entre los wrappers y los datos primitivos