# Algoritmos y Estructura de Datos

## CADENAS EN JAVA

DOCENTE: MARX DANLY LEÓN TRUJILLO
FACULTAD DE ING. INDUSTRIAL Y SISTEMAS
E.P. INGENIERÍA DE SISTEMAS

# CARACTERES EN JAVA

El tipo de dato carácter sirve para representar datos alfanuméricos. El conjunto de elementos que puede representar está estandarizado según diferentes tablas de código (ASCII o Unicode).

El estándar más antiguo es ASCII, que consiste en una combinación de 7 u 8 bits asociada a un carácter alfanumérico concreto, pero que está siendo sustituido por sus limitaciones en la representación de caracteres no occidentales.

Java proporciona el tipo **char**, de 16 bits, para manejar caracteres.

# CARACTERES EN JAVA

Las combinaciones de 7 bits del ASCII clásico dan lugar a un total de 127 valores distintos (desde 0000000 hasta 1111111). En Java, se siguen reservando esos 127 valores para la codificación ASCII. El resto, sirve para almacenar los caracteres en formato Unicode. Los caracteres válidos que pueden almacenarse en una variable tipo char son:

- **Las letras minúsculas:** 'a', 'b', 'c' ... 'z'
- **Las letras mayúsculas:** 'A', 'B', 'C' ... 'Z'
- **Los dígitos:** '1', '2', '3' ...
- **Caracteres especiales o de otros idiomas:** '$', '%', '¿', '!' , 'ç'...

# CARACTERES EN JAVA

Una generalización del tipo carácter es el tipo cadena de caracteres, utilizado para representar series de varios caracteres. Éste, sin embargo, es un objeto complejo. Las cadenas se utilizan tan a menudo que no podremos evitar usarlas.

# CADENA DE CARACTERES EN JAVA

Una cadena de caracteres literal se representa encerrada entre comillas dobles, por ejemplo: **"Esto es una cadena"**. En cambio, un carácter literal se representa encerrado entre comillas simples, por ejemplo: **'A'**.

En Java las cadenas son tratadas como objetos, por lo tanto, **"Esto es una cadena“** es un objeto y podemos invocar sus métodos como veremos a continuación:

```
// imprime ESTO ES UNA CADENA (en mayusculas)
System.out.println( "Esto es una cadena".toUpperCase() );
```

# CADENA DE CARACTERES EN JAVA

En cambio, los caracteres (al igual que en C) son valores numéricos enteros. Por ejemplo, **'A'** es, en realidad, el valor 65 ya que este es el código ASCII de dicho carácter. Notemos además que no es lo mismo **"A"** que '**A**'. El primero es una cadena de caracteres que contiene un único carácter; es un objeto. El segundo es un char; un valor numérico.

```
public class DemoCaracteres
{
   public static void main(String[] args)
   {
      for( int i=0; i<5; i++ )
      {
         System.out.println(i+"A");
      }
   }
}
```

# CADENA DE CARACTERES EN JAVA

```
public static void main(String[] args)
{
    char c;
    for( int i='A'; i<='Z'; i++ )
    {
        // para asignar un int en un char debemos "castear"
        c = (char) i;
        System.out.println(c);
    }
}
```

# CADENA DE CARACTERES EN JAVA

```
public static void main(String[] args)
{
    char c;
    for( int i='A'; i<='Z'; i++ )
    {
        // para asignar un int en un char debemos "castear"
        c = (char) i;
        System.out.println(c);
    }
}
```

Recordemos que un int se representa en 4 bytes con bit de signo mientras que un **char** se representa en dos bytes sin bit de signo. Por lo tanto, no siempre se podría asignar un int a un char ya que el char no puede almacenar valores negativos ni valores superiores a 216-1 (máximo valor que se puede almacenar en 2 bytes sin bit de signo).

# CARACTERES ESPECIALES EN JAVA

En Java (igual que en C) existen caracteres especiales. Estos caracteres se pueden utilizar anteponiendo la barra \ (léase “barra” o carácter de “escape”). Algunos de estos son:

- \t – tabulador
- \n – salto de línea
- \" – comillas dobles
- \' – comillas simples
- \\ – barra

# CARACTERES ESPECIALES EN JAVA

```
public static void main(String[] args)
{
    System.out.println("Esto\t es un \"TABULADOR\"");
    System.out.println("Esto es un\nBARRA\nENE\n\n   :O)");
    System.out.println("La barra es asi: \\");
}
```

# CARACTERES ESPECIALES EN JAVA

```
public static void main(String[] args)
{
   System.out.println("Esto\t es un \"TABULADOR\"");
   System.out.println("Esto es un\nBARRA\nENE\n\n   :O)");
   System.out.println("La barra es asi: \\");
}
```

La salida de este programa es la siguiente:
Esto es un "TABULADOR"
Esto es un
BARRA
ENE
:O)
La barra es asi: \

# CARACTERES ESPECIALES EN JAVA

```
public static void main(String[] args)
{
   System.out.println("Esto\t es un \"TABULADOR\"");
   System.out.println("Esto es un\nBARRA\nENE\n\n   :O)");
   System.out.println("La barra es asi: \\");
}
```

La salida de este programa es la siguiente:
Esto es un "TABULADOR"
Esto es un
BARRA
ENE
:O)
La barra es asi: \

# TRATAMIENTO DE CADENAS EN JAVA

Como vimos anteriormente, las cadenas de caracteres son tratadas como objetos porque ***String*** no es un tipo de datos simple. ***String*** es una clase. Sin embargo, un objeto es una variable que además de contener información contiene los métodos (o funciones) necesarios para manipular esta información.

Agreguemos también que las clases definen los tipos de datos de los objetos. Con esta breve e informal definición, podremos estudiar algunos casos de manejo de cadenas de caracteres.

# TRATAMIENTO DE CADENAS EN JAVA

## Acceso a los caracteres de un String

Una cadena representa una secuencia finita de cero o más caracteres numerados a partir de cero. Es decir que la cadena "Hola" tiene 4 caracteres numerados entre 0 y 3. Ejemplo: acceso directo a los caracteres de una cadena.

```
String s = "Esta es mi cadena";

System.out.println( s.charAt(0) );
System.out.println( s.charAt(5) );
System.out.println( s.charAt(s.length()-1) );

char c;
for(int i=0; i<s.length(); i++)
{
   c = s.charAt(i);
   System.out.println(i+" -> "+c);
}
```

# TRATAMIENTO DE CADENAS EN JAVA

El método **charAt** retorna al carácter (tipo char) ubicado en una posición determinada. El método **length** retorna la cantidad de caracteres que tiene la cadena.

No debemos confundir el atributo length de los arrays con el método length de los strings. En el caso de los arrays, por tratarse de un atributo se lo utiliza sin paréntesis. En cambio, en el caso de los strings está implementado como un método, por lo tanto, siempre debe invocarse con paréntesis. Veamos el siguiente ejemplo:

```
char c[] = { 'H', 'o', 'l', 'a' };
System.out.println( c.length );

String s = "Hola";
System.out.println( s.length() );
```

# TRATAMIENTO DE CADENAS EN JAVA

## Mayúsculas y minúsculas

Ejemplo: pasar una cadena a mayúsculas y minúsculas.

```
public static void main(String[] args)
{
    String s = "Esto Es Una Cadena de caracteres";
    String sMayus = s.toUpperCase();
    String sMinus = s.toLowerCase();

    System.out.println("Original: "+s);
    System.out.println("Mayusculas: "+sMayus);
    System.out.println("Minusculas: "+sMinus);
}
```

# TRATAMIENTO DE CADENAS EN JAVA

## Ocurrencia de Caracteres

Ejemplo: ubicar la posición de un carácter dentro de una cadena.

```
public static void main(String[] args)
{
   String s = "Esto Es Una Cadena de caracteres";

   int pos1 = s.indexOf('C');
   int pos2 = s.lastIndexOf('C');
   int pos3 = s.indexOf('x');

   System.out.println(pos1);
   System.out.println(pos2);
   System.out.println(pos3);
}
```

# TRATAMIENTO DE CADENAS EN JAVA

El método **indexOf** retorna la posición de la primera ocurrencia de un carácter dentro del string. Si la cadena no contiene ese carácter entonces retorna un valor negativo.

Análogamente, el método **lastIndexOf** retorna la posición de la última ocurrencia del carácter dentro del string o un valor negativo en caso de que el carácter no esté contenido dentro de la cadena.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| E | s | t | o |  | E | s |  | U | n | a |  | C | a | d | e | n | a |  | D | e |  | C | a | r | a | c | t | e | r | e | s |

# TRATAMIENTO DE CADENAS EN JAVA

## Subcadenas

Ejemplo: uso del método ***substring*** para obtener diferentes porciones de la cadena original.

```
String s = "Esto Es Una Cadena de caracteres";

String s1 = s.substring(0,7);
String s2 = s.substring(8,11);

// toma desde el caracter 8 hasta el final
String s3 = s.substring(8);

System.out.println(s1);
System.out.println(s2);
System.out.println(s3);
```

# TRATAMIENTO DE CADENAS EN JAVA

El método ***substring()*** puede invocarse con dos argumentos o con un único argumento. Si lo invocamos con dos argumentos, estaremos indicando las posiciones desde (inclusive) y hasta (no inclusive) que delimitarán la subcadena que queremos extraer. En cambio, si lo invocamos con un solo argumento estaremos indicando que la subcadena a extraer comienza en la posición especificada (inclusive) y se extenderá hasta el final del string.

Decimos que un método está “sobrecargado” cuando podemos invocarlo con diferentes cantidades y/o diferentes tipos de argumentos. Este es el caso del método ***substring()***.

# TRATAMIENTO DE CADENAS EN JAVA

## Prefijos y suifjos

Con los métodos **startWith()** y ***endsWith()***, podemos verificar muy fácilmente si una cadena comienza con un determinado prefijo o termina con algún sufijo.

```
String s = "Un buen libro de Java";

boolean b1 = s.startsWith("Un buen"); // true
boolean b2 = s.startsWith("A");       // false
boolean b3 = s.endsWith("Java");      // true
boolean b4 = s.endsWith("Chau");      // false

System.out.println(b1);
System.out.println(b2);
System.out.println(b3);
System.out.println(b4);
```

# TRATAMIENTO DE CADENAS EN JAVA

## Posición de un substring dentro de la cadena

Los métodos indexOf y lastIndexOf están sobrecargados de forma tal que permiten detectar la primera y la última ocurrencia (respectivamente) de un substring dentro de la cadena en cuestión.

```
String s = "Un buen libro de Java, un buen material";

int pos1 = s.indexOf("buen");     // retorna 3
int pos2 = s.lastIndexOf("buen"); // returna 26
System.out.println(pos1);
System.out.println(pos2);
```

# TRATAMIENTO DE CADENAS EN JAVA

## Concatenar cadenas

Para concatenar cadenas podemos utilizar el operador + como se muestra a continuación:

```
String x = "";
x = x + "Hola ";
x = x + "Que tal?";

System.out.println(x); // imprime "Hola Que tal?"
```

Si bien lo anterior funciona bien no es la opción más eficiente ya que cada concatenación implica instanciar una nueva cadena y descartar la anterior. Mucho más eficiente será utilizar la clase ***StringBuffer***.

# TRATAMIENTO DE CADENAS EN JAVA

## La clase StringBuffer

Esta clase representa a un **string** cuyo contenido puede variar (mutable). Provee métodos a través de los cuales podemos insertar nuevos caracteres, eliminar algunos o todos y cambiar los caracteres contenidos en las diferentes posiciones del string. El compilador utiliza un **string buffer** para resolver la implementación del operador de concatenación +. Es decir que, en el ejemplo anterior, se utilizará una instancia de **StringBuffer** de la siguiente manera:

```
String x = new StringBuffer().append("Hola ")
                             .append("Que Tal?")
                             .toString();
```

# TRATAMIENTO DE CADENAS EN JAVA

```
public static void main(String[] args)
{
   // obtengo el milisegundo actual
   long hi = System.currentTimeMillis();

   // instancio un StringBuffer inicialmente vacio
   StringBuffer sb=new StringBuffer();

   // voy a concatenar n caracteres
   int n=100000;

   char c;
   for(int i=0; i<n; i++)
   {
      // obtengo caracteres entre 'A' y 'Z'
      c = (char)('A' + i%('Z'-'A'+1));

      // concateno el caracter en el StringBuffer
      sb.append(c);
   }

   // obtengo el milisegundo actual
   long hf = System.currentTimeMillis();

   // muestro la cadena resultante
   System.out.println(sb.toString());

   // muestro la cantidad de milisegundos insumidos
   System.out.println((hf-hi)+" milisegundos");
}
```

# TRATAMIENTO DE CADENAS EN JAVA

## Conversión entre números y cadenas

Java provee clases para brindar la funcionalidad que los tipos de datos primitivos (int, double, char, etc) no pueden proveer (justamente) por ser solo tipos de datos primitivos. A estas clases se las suele denominar **WRAPPERS** (envoltorios) y permiten, entre otras cosas, realizar conversiones entre cadenas y números, obtener expresiones numéricas en diferentes bases, etcétera. En el siguiente ejemplo, vemos cómo podemos realizar conversiones entre valores numéricos y cadenas y viceversa.

# TRATAMIENTO DE CADENAS EN JAVA

Algunos ejemplos son:

```
// --- operaciones con el tipo int ---
int i = 43;

// convierto de int a String
String sInt = Integer.toString(i);

// convierto de String a int
int i2 = Integer.parseInt(sInt);

// --- operaciones con el tipo double ---
double d = 24.2;

// convierto de double a String
String sDouble = Double.toString(d);

// convierto de String a double
double d2 = Double.parseDouble(sDouble);
```

# TRATAMIENTO DE CADENAS EN JAVA

En la siguiente tabla, vemos el *wrapper* de cada tipo de datos primitivo.

| Tipo | Wrapper |
|---|---|
| `byte` | `Byte` |
| `short` | `Short` |
| `char` | `Character` |
| `int` | `Integer` |
| `long` | `Long` |
| `float` | `Float` |
| `double` | `Double` |
| `boolean` | `Boolean` |

Todos estas clases tienen los métodos `parseXxx` (siendo *Xxx* el tipo de datos al que se quiere *parsear* el *string*) y el método `toString` para convertir un valor del tipo que representan a cadena de caracteres.

# TRATAMIENTO DE CADENAS EN JAVA

## Representación numérica en diferentes bases

Java, igual que C, permite expresar valores enteros en base 8 y en base 16. Para representar en entero en base 16, debemos anteponerle el prefijo 0x (léase “cero equis”).

```
int i = 0x24ACF; // en decimal es 150223
System.out.println(i); // imprime 150223
```

Para expresar enteros en base 8, debemos anteponerles el prefijo 0 (léase cero).

```
int j = 0537; // en decimal es 351
System.out.println(j); // imprime 351
```

# TRATAMIENTO DE CADENAS EN JAVA

Utilizando la clase **Integer** podemos obtener la representación binaria, octal, hexadecimal y en cualquier base numérica de un entero dado. Esto lo veremos en el siguiente programa.

```
import java.util.Scanner;

public class Cadenas6
{
   public static void main(String[] args)
   {
      Scanner scanner=new Scanner(System.in);
      System.out.print("Ingrese un valor entero: ");
      int v = scanner.nextInt();

      System.out.println("Valor Ingresado: "+v);
      System.out.println("binario = "+Integer.toBinaryString(v));
      System.out.println("octal = "+Integer.toOctalString(v));
      System.out.println("hexadecimal = "+Integer.toHexString(v));

      System.out.print("Ingrese una base numerica: ");
      int b = scanner.nextInt();

      String sBaseN=Integer.toString(v,b);
      System.out.println(v + " en base("+b+") = " + sBaseN);
   }
}
```

# TRATAMIENTO DE CADENAS EN JAVA

## Comparación de cadenas

En el lenguaje Java (al igual que en C), no existe un tipo de datos primitivo para representar cadenas de caracteres. Las cadenas se representan como objetos de la clase String. Ahora bien, la clase String tiene tanta funcionalidad y (yo diría) tantos “privilegios” que muchas veces se la puede tratar casi como si fuera un tipo de datos primitivo.
Informalmente, diremos que “un objeto es una variable cuyo tipo de datos es una clase”. Así, en el siguiente código:

```
String s  = "Hola a todos !";
```

# TRATAMIENTO DE CADENAS EN JAVA

**s** resulta ser un objeto (una variable) cuyo tipo de datos es String. Es decir: un objeto de la clase String o (también podríamos decir) una instancia de esta clase.

Los objetos son, en realidad, referencias (punteros) que indican la dirección física de memoria en donde reside la información que contienen. Por este motivo, no podemos utilizar el operador de comparación == (igual igual) para comparar objetos, porque lo que estaremos comparando serán direcciones de memoria, no contenidos. Como las cadenas son objetos, comparar cadenas con este operador de comparación sería un error.

# TRATAMIENTO DE CADENAS EN JAVA

Esto podemos verificarlo con el siguiente programa en el que se le pide al usuario que ingrese una cadena y luego otra. El programa compara ambas cadenas con el operador == e informa el resultado obtenido.

```
Scanner scanner = new Scanner(System.in);
System.out.print("Ingrese una cadena: ");
String ss = scanner.next();
System.out.print("Ingrese otra cadena: ");
String sss = scanner.next();
// muestro las cadenas para verifi car lo que contienen
System.out.println("s1 = [" + ss + "]");
System.out.println("s2 = [" + sss + "]");
// esto esta mal !!!
if( ss == sss )
{
System.out.println("Son iguales");
}
else
{
System.out.println("Son distintas");
}
```

# TRATAMIENTO DE CADENAS EN JAVA

La salida de este programa siempre será: **Son distintas.**
Lo correcto será comparar las cadenas utilizando el método ***equals***. Este método compara los contenidos y retorna true o false según estos sean iguales o no.

```
:
// Ahora si !!!
if( s1.equals(s2) )
{
   System.out.println("Son iguales");
}
:
```

# TRATAMIENTO DE CADENAS EN JAVA

```
public class Cadenas9
{
    public static void main(String[] args)
    {
        // dos cadenas iguales
        String s1 = "Hola";
        String s2 = "Hola";
        System.out.println("s1 = [" + s1 + "]");
        System.out.println("s2 = [" + s2 + "]");
        if( s1 == s2 )
        {
        System.out.println("Son iguales");
        }
        else
        {
        System.out.println("Son distintas");
        }
    }
}
```

# TRATAMIENTO DE CADENAS EN JAVA

En este caso el operador == retorna true, es decir: compara “bien” y el programa indicará que ambas cadenas son idénticas. ¿Por qué? Porque Java asigna la cadena literal "Hola" en una porción de memoria y ante la aparición de la misma cadena no vuelve a alocar memoria para almacenar la misma información, simplemente obtiene una nueva referencia a dicha porción de memoria. En consecuencia, los dos objetos s1 y s2 apuntan al mismo espacio de memoria, son punteros idénticos y por este motivo el operador == retorna true. El método equals también hubiera retornado true.

# TRATAMIENTO DE CADENAS EN JAVA

## La Clase StringTokenizer

La funcionalidad de esta clase la explicaremos sobre el siguiente ejemplo. Sea la cadena **“s”** definida en la siguiente línea de código:

```
String s = "Juan|Marcos|Carlos|Matias";
```

Si consideramos como separador al carácter **‘|’** (léase “carácter pipe”) entonces llamaremos token a las subcadenas encerradas entre las ocurrencias de dicho carácter y a las subcadenas encerradas entre este y el inicio o el fin de la cadena **s**.

# TRATAMIENTO DE CADENAS EN JAVA

Para hacerlo más simple, el conjunto de tokens que surgen de la cadena **“s”** considerando como separador al carácter **|** es el siguiente:

tokens = { Juan, Marcos, Carlos, Matias }

Utilizando la clase **StringTokenizer** podemos separar una cadena en tokens delimitados por un separador. En el siguiente ejemplo, veremos cómo hacerlo.

# TRATAMIENTO DE CADENAS EN JAVA

```
import java.util.StringTokenizer;

public class Cadenas7
{
    public static void main(String[] args)
    {
        String s = "Juan|Marcos|Carlos|Matias";
        StringTokenizer st = new StringTokenizer(s,"|");

        String tok;
        while( st.hasMoreTokens() )
        {
            tok = st.nextToken();
            System.out.println(tok);
        }
    }
}
```

# TRATAMIENTO DE CADENAS EN JAVA

## Usar Expresiones Regulares Para Particionar Una Cadena

La clase ***String*** provee el método **split** que permite particionar una cadena a partir de una expresión regular. Si el lector tiene conocimientos sobre expresiones regulares, este método le resulta más útil que la clase ***StringTokenizer***.

El siguiente ejemplo es equivalente al ejemplo anterior, pero utiliza el método **split** que provee la clase ***String***.

# TRATAMIENTO DE CADENAS EN JAVA

```
public static void main(String[] args)
{
    String s = "Juan|Marcos|Carlos|Matias";
    String[] tokens = s.split("[|]");

    for(int i=0; i<tokens.length; i++)
    {
        System.out.println(tokens[i]);
    }
}
```