Tema 7: Manejo de Cadenas

Clase String

- Recordar que las cadenas de caracteres se representan en Java como secuencia de caracteres Unicode encerradas entre **“ “.**

- Para manipular cadenas de caracteres, por razones de eficiencia, se utilizan tres clases incluidas en el paquete **java.lang**:

  - ✓ String
    Para cadenas constantes.
  - ✓ StringBuffer
    Para cadenas modificables.
  - ✓ StringBuilder
    Similar a StringBuffer pero más eficiente.

Clase String

- Las cadenas de los objetos String no pueden modificarse (crecer, cambiar un caracter, etc).
- Pero una variable String puede recibir valores distintos.
- Constructores que soporta:

```
String();
String( String str );
String( char val[] );
String( char val[],int offset,int count );
String( byte val[],int hibyte );
String( byte val[],int hibyte,int offset,int count );
```

. . .

Clase String

- Métodos

```
int indexOf( int ch );
int indexOf( int ch,int fromindex );
int lastIndexOf( int ch,int fromindex );
int indexOf( String str );
String substring( int beginindex );
String substring( int beginindex,int endindex );
String concat( String str );
String replace( char oldchar,char newchar );
String toLowerCase();String toUpperCase();
void getChars(int srcBegin,int srcEnd,char dst[],int
  dstBegin);
String toString();
char toCharArray();
String valueOf( int i );
String valueOf( float f );
String copyValueOf( char data[] );
String copyValueOf( char data[],int offset,int count );
 . . .
```

Ejemplos

código	*resultado*
`"Miguel".length()`	6
`"Miguel".equals("Miguel")`	true
`"Miguel".equals("miguel")`	false
`"Miguel".equalsIgnoreCase("miguel")`	false
`"Miguel".compareTo("Saturnino")`	-6
`"Miguel".compareTo("Miguel")`	0
`"Miguel".compareTo("Michelin")`	4
`"Miguel".charAt(1)`	'i'
`"Miguel".charAt(4)`	'e'
`"Miguel".toCharArray()`	{ 'M', 'i', 'g', 'u', 'e', 'l' }
`"Miguel".substring(1, 4)`	"igu"
`"Miguel.substring(1)`	"iguel"
`"tragaldabas".indexOf('a')`	2
`"tragaldabas".lasIndexOf('a')`	9
`"tragaldabas".startsWith("tragón")`	false
`"tragaldabas".endsWith("dabas")`	true
`"tragaldabas".split("a")`	{ "tr", "g", "ld", "b", "s" }

Ejemplos String

- String x = “abc”;
  String y =
  x.concat(“def”).toUpperCase().replace(‘C’,’x’);
  System.out.println(“y= “+ y);

- El operador **+** también puede ser usado para concatenar.

- String a = “newspaper”;
  a=a.substring(5,7);
  char b = a.chartAt(1);
  a = a+b;
  System.out.println(a);

Clase StringBuffer

- Constructores

  StringBuffer();
  StringBuffer(int len);
  StringBuffer(String str);

- Los objetos de esta clase se inicializan de cualquiera de las siguientes formas:

✓ StringBuffer str1 = new StringBuffer(10);

✓ StringBuffer str2 = new StringBuffer(“hola”);

Clase StringBuffer

- Las cadenas de los objetos StringBuffer se pueden ampliar, reducir y modificar mediante mensajes.

- Cuando la capacidad establecida excede, se aumenta automáticamente.

- Algunos métodos(java.util):

```
int length();
char charAt( int index );
void getChars( int srcBegin,int srcEnd,char dst[],int dstBegin );
String toString();
void setLength( int newlength );
void setCharAt( int index,char ch );
int capacity();
void ensureCapacity( int minimum );
void copyWhenShared();
StringBuffer append( double d );
StringBuffer append( char ch );
StringBuffer insert( int offset,Object obj );
StringBuffer insert( int offset,String str );
```

Ejemplos StringBuffer

- ```
  class CadenaAppend {
  public static void main( String args[] ) {
      StringBuffer str =
                          new StringBuffer( “Buen" );
    str.append( "  Día!!! " );
    System.out.println( str );
  }
  }
  ```

- ```
  StringBuffer str =
                  new StringBuffer( “01234567" );
  str.insert( 4," ***" );
  System.out.println( str );
  ```

Clase StringBuilder

- Es similar a la clase StringBuffer sólo que es mas eficiente, se encuentra en java.util

- La construcción de un tipo de dato de esta clase es similar a la de StringBuffer y para convertir un **StringBuilder** en **String** puede usarse su método **ToString()** heredado de *System.Object.*

- Ejemplo:

```
public class ModificacionCadenas {
  public static void main(String []arg) {
     StringBuilder cadena = new StringBuilder("Telas");
     String cadenaInmutable;
     cadena.replace(0,1,"V");
     System.out.println(cadena); // Muestra Velas
     cadenaInmutable = cadena.toString();
     System.out.println(cadenaInmutable);
  }
}
```

Métodos StringBuilder

- .

- Métodos de consulta:

```
length()
capacity()
charAt(int pos)
```

- Métodos para construir objetos **`String`**:

```
substring(int posini, int posfin+1)
substring(int posini)
toString()
```

- Métodos para modificar objetos **`StringBuilder`**:

```
append(String str)
insert(int pos, String str)
setCharAt(int pos, char car)
replace(int pos1, int pos2+1, String str)
reverse()
```

Ejemplo StringBuilder

```
public class StringDemo {
  public static void main(String[] args) {
      String cadena = "Aarón es Nombre";
      int long = cadena.length();
      StringBuilder réplica = new StringBuilder(long);
      char c;
      for (int i = 0; i < long; i++) {
        c = cadena.charAt(i);
        if (c == 'A') {
            c = 'V';
        } else if (c == 'N') {
            c = 'H';
        }
        réplica.append(c)
      }
      System.out.println(réplica);
  }
}
```

Diferencias entre las clases

- La clase String es para cadenas con valores constantes.
- La clase StringBuffer es para manejo de cadenas modificables.
- La clase StringBuilder por ser del mismo tipo de funcionalidad que la StringBuffer es mas usada por ser mas eficiente.

Expresiones regulares

- Definición

- Una expresión regular es, a menudo llamada también patrón.

- En el área de la programación las expresiones regulares son un método por medio del cual se pueden realizar búsquedas dentro de cadenas de caracteres.

Símbolos *,+, ?

- +

  El signo más indica que el carácter al que sigue debe aparecer al menos una vez.

  Ejemplo: "ho+la" describe el conjunto infinito *hola*, *hoola*, *hooola*, *hoooola*, etc.

- ?

  El signo de interrogación indica que el carácter al que sigue puede aparecer como mucho una vez (0 o 1).

  Ejemplo: "ob?scuro" hace “match” con *oscuro* y *obscuro*.

Símbolos *,+, ?

- *

  El asterisco indica que el carácter al que sigue puede aparecer cero, una, o más veces.

  Ejemplo: "0*42" coincide con *42*, *042*, *0042*, *00042*, etc.

- Agrupación

  Los paréntesis pueden usarse para definir el ámbito y precedencia de los demás operadores.

  Ejemplo, "(p|m)adre" es lo mismo que "padre|madre", y "(des)?amor" coincide con *amor* y con *desamor*.

Expresiones regulares en Java

- El paquete **java.util.regex** esta formado por dos clases, la clase Matcher y la clase Pattern y por la excepción *PatternSyntaxException*.

- La clase **Pattern** representa a la expresion regular, que en el paquete java.util.regex necesita estar compilada.

Expresiones regulares en Java

- La clase **Matcher** es un tipo de objeto que se crea a partir de un patrón mediante la invocación del método **Pattern.matcher**.

- Este objeto es el que permite realizar operaciones sobre la secuencia de caracteres que se quiere validar o la en la secuencia de caracteres en la que se desea buscar.

Creación y Manipulación

- **Clase Pattern**, con ella se crea un patrón.

- El método **compile** compila una expresión regular.
- El método **pattern** devuelve la expresión regular que se ha compilado.
- El método **matcher** crea un objeto Matcher a partir del patrón.
- El método **split** divide una cadena dada en partes que cumplan el patrón compilado.
- El método **matches** compila una expresión regular y comprueba una cadena de caracteres contra ella.

Manipulación

- La clase Matcher se utiliza para comprobar cadenas contra el patrón indicado. Un objeto Matcher se genera a partir de un objeto Pattern por medio del método matcher.

- Después de creado el objeto se podrá hacer uso de los siguientes métodos:
  - ✓ El método **matches** que intenta encajar toda la secuencia en el patrón.
  - ✓ El método **lookingAt**, intenta encajar el patrón en la cadena
  - ✓ El método **find** que va buscando subcadenas dentro de la cadena de caracteres que cumplan el patrón compilado.

- Cuando se encuentra una ocurrencia, se puede hacer uso de:
  - ✓ El método **start**. Marca el primer carácter de la ocurrencia en la secuencia
  - ✓ El método **end.** Marca el ultimo carácter de la ocurrencia.
  - ✓ Estos métodos devuelven un boolean indicando si la operación ha tenido éxito o no.

Ejemplo

- El ejemplo sustituye todas las apariciones que concuerden con el patrón "a*b" por la cadena "-".

```
import java.util.regex.*;

public class EjemploReplaceAll{
    public static void main(String args[]){
    Pattern patron = Pattern.compile("a*b");
    // Se crea el Matcher a partir del patron,
    //la cadena como parametro
   Matcher encaja =
    patron.matcher("aabmanoloaabmanoloabmanolob");
   String resultado =encaja.replaceAll("-”);
   System.out.println(resultado);
  }
}
```

Símbolos \d,\s,\w

Intervalos de caracteres predefinidos

.	Cualquier caracter (puede que no se incluyan los terminadores de línea)
\d	Un numero: **[0-9]**
\D	Todo menos un numero: **[^0-9]**
\s	Un espacio en blanco: **[\t\n\x0B\f\r]**
\S	Todo menos un espacio en blanco: **[^\s]**
\w	Una letra: **[a-zA-Z_0-9]**
\W	Todo menos letras: **[^\w]**

Límites

^	Comienzo de una línea
$	Fin de una línea
\b	Fin de palabra
\B	No es fin de palabra
\A	El principio de la cadena de entrada
\G	El final del ultimo patrón encajado
\Z	El final de la entrada pero el terminador final, si existe
\z	El final de la cadena de entrada

Ejemplos

- La siguiente expresión

  “\\d\\d\\s”

  significa encontrar una expresión que concuerde con dos dígitos numéricos cualquiera seguido de un espacio en blanco.

- import java.util.regex.*;

```
public class Reg {
   public static Pattern pattern;
   public static Matcher matcher;

   public static void main(String argv[ ]) {
        new Reg();
   }
  Reg() {
     pattern= Pattern.compile(“(aa)”);
     matcher= pattern.matcher(“Java”);
     if (matcher.find) {
        System.out.printn(“match”);
        System.out.printn(matcher.start());
  }
}
```

Ejemplo

- El programa valida una cadena que contiene un email, son 4 comprobaciones con un patrón cada una: 1) que no contenga como primer caracter una @ o un punto, 2) que no comience por www**.** , 3) que contenga una y solo una @ y 4) que no contenga caracteres ilegales.

```
import java.util.regex.*;

public class ValidacionEmail {
public static void main(String[] args) throws Exception {
 String input = "www.?regular.com";
 // comprueba que no empieze por punto o @
  Pattern p = Pattern.compile("^\\.|^\\@");
 Matcher m = p.matcher(input);
 if (m.find())
     System.err.println("Las direcciones email no
     empiezan por punto o @");
```

Ejemplo ...

-

```
// comprueba que no inicie por www.
 p = Pattern.compile("^www\\.");
m = p.matcher(input);
 if (m.find())
       System.out.println("Los emails no empiezan
    por www");
// comprueba que contenga @
   p = Pattern.compile("\\@");
   m = p.matcher(input);
   if (!m.find())
          System.out.println("La cadena no tiene
    arroba");
```

Ejemplo ...

```
// comprueba que no contenga caracteres prohibidos
   p = Pattern.compile("[^A-Za-z0-9\\.\\@_\\-~#]+");
   m = p.matcher(input);
   StringBuffer sb = new StringBuffer();
   boolean resultado = m.find();
   boolean caracteresIlegales = false;
   while(resultado) {
       caracteresIlegales = true;
        m.appendReplacement(sb, "");
         resultado = m.find();
    }
     // Añade el ultimo segmento de la entrada a la
    cadena
    m.appendTail(sb);
    input = sb.toString();
    if (caracteresIlegales) {
       System.out.println("La cadena contiene caracteres
    ilegales");
   }   }}
```

Formateo de salida

- Existen varias formas de dar formato (justificación y alineamiento, formatos numéricos, de fecha, etc) a las cadenas y otros tipos de datos.

- Similar al printf de C.

- Son tres las formas en java:
  - » Con el **printf**. La clase PrintStream proporciona este método.
  - » Con el método estático **format**.
  - » Creando **un objeto de la clase Formatter** que se encuentra en el paquete de java.util

Clase Formatter

- La clase *Formatter* permite dar formato, trabajandola de manera directa.
- Se encuentra en el paquete de java.util
- Soporta la internacionalización gracias a su contructor con el parámetro Locale.
- Usando la interfaz *Formattable* da formatos (limitados) a tipos creados por el usuario.

Clase Formatter

- **// Uso de formatter para construir cadenas formateadas**

```
StringBuilder sb = new StringBuilder();
 Formatter f = new Formatter(sb, Locale.US);
 f.format("Hola, %1$2s, este es un numero
    %2$d", “Usuario", 20);
```

// Métodos predefinidos en ciertas clases:

```
 System.out.format("Hola, %1$2s, este es un
    numero %2$d", “Usuario", 20);
 System.err.printf("Hola, %1$2s, este es un
    numero %2$d", “Usuario", 20);
 String s = String.format("Hola, %1$2s",
    “Usuario");
```

Especificación del formato

- Sintaxis

  % [argumento **$**][marca][ancho][**.** precision] **tipo**

- Significado
  - ✓ argumento. 1$ se refiere al primer argumento, 2$ al segundo y así sucesivamente. < se refiere al anterior.
  - ✓ marca. Determina las pequeñas variantes sobre la cadena generada:
    - Nada. Ajusta a la derecha, rellenando con blancos a la izquierda.
    - – Ajusta a la izquierda rellenando con blancos a la derecha.
    - + Incluye siempre el signo de la cantidad numérica (positivo o negativo).
    - . . .

Especificación del formato

- Significado

  - ✓ ancho. Indica la longitud mínima de la cadena generada.

  - ✓ precision.

    %f Número de cifras decimales (después de la coma).

    %g Número de cifras significativas.

    %b ancho máximo (se trunca la palabra).

    %s Ancho máximo (si la cadena es más larga, se trunca).

Especificación del formato

- Significado
  - ✓ tipo

%n	Fin de línea
%%	Carácter %
%s %S	Cadena de caracteres
%d	Número entero: notación decimal.
%x %X	Número entero: notación hexadecimal
%c %C	Número entero: como caracter
%f	Número real, sin exponente.
%t %T	Fecha y hora.
%tH	Hora: 00..23
%tk	Hora: 0..23
%tY	Año (2006)
%tZ	Zona horaria abreviada

. . .

(http://java.sun.com/j2se/1.5.0/docs/api/java/util/Formatter.html)

Ejemplo 1

Formato, valores y/o variable	String resultado
%d”,5	“5”
“%3d”,5	“ 5”
“%.2f”,3.1416	“3.14”
“%8.2pesos”,15.99	“ 15.99pesos”
“%s %s”,”La”,”Universidad”	“La Universidad”
“La %s”,”Universidad”	“La Universidad”
Date date new Date(); “Son las %tH:%<tM”,date	“Son las 09:18”
Date dt=new Date(); “Hoy es %tA,%<te de %<tB de %<tY”,dt	“Hoy es martes, 10 de febrero de 2006”

Método format

- Ejemplo:

```
public class TestFormat{
  public static void main(String args[]) {
     String cadena= "Resultado: ";
      float num=18526.459f;
      System.out.format("%s  %.2f",cadena,
    num);
   }
}
```

Método format

- Ejemplo del método de la clase String
- int a = 65;

  String s = String.format("Char: %c Integral: %d Octal: %o Hex: %x %n Fin", a, a, a, a);

  System.out.print(s);

- int a = 65;

  String s = String.format("char: %c integral: %<d octal: %<o hex: %<x %n", a);

- // Necesario java.util

```
String s=String.format("%1$td %1$tb %1$ty",
    new Date());
  System.out.print(s);
```

Clase Formatter

- Constructores

  Formatter( )

  Formatter(File file)

  Formatter(File file, String csn)

  Formatter(Locale l)

  Formatter(OutputStream os)

  Formatter(PrintStream ps)

  Formatter(String filename)

  . . .

- Algunos métodos

  void lush()

  Formatter format(String format,Object .. arg)

  Formatter format(Locale l, String format, Object .. Arg)

  Locale locale()

Ejemplo

```
public static void main(String[] args) {
    float resul = 18527.125f;
    StringBuilder sb = new StringBuilder();
    Formatter formato = new
    Formatter(sb,Locale.US); //US
    formato.format ("Resultado
    formateado: $%(,.2f) %n ",  resul);
    formato.format("%d %n ", 128)
    System.out.println(formato);
  }
```

O bien:

```
//System.out.println(formato); por
  System.out.println(sb.toString());
```

Método printf

- Las clases PrintStream y PrintWriter incluyen el método printf.

- Ejemplo:

```
public static void main(String [ ] arg){
    String cadena= "Resultado: ";
    float num=18526.459f;
    System.out.printf("%s  %.2f",cadena,
    num);
}
```

Método print y println

- Método que ofrece PrintWriter.
- Similar al formato usado con Formatter.

- Ejemplo:

(No admite formatos)

```
import java.io.*;
public class TestPrintWriter{
  public static void main(String args[]) {
    PrintWriter pw = new PrintWriter(System.out, true);
    pw.println("Esto es una cadena");
    int i = -7;
    pw.println(i);
    double d = 4.5e-7;
    pw.println(d);
  }
}
```

- Se verá otro ejemplo en la sesión 16(Manejo de flujos)

Clase Scanner

- La clase **Scanner** permite parsear un flujo de entrada (consola,archivos, cadena de texto, stream de datos, etc), y extraer tokens siguiendo un determinado patrón o tipo de datos.

- Constructores

  Scanner(String origen)

  Scanner(Readable origen)

  Scanner(Reader origen)

  Scanner(InputStream origen)

  Scanner(File origen)

Ejemplo Scanner

- También se permite trabajar con expresiones regulares para indicar qué patrones se deben buscar.

- Ejemplo1:

```
String s= “Martes, 13 de febrero de 2007,
    actualizado a las 13:20 h.”;
Scanner scanner=new Scanner(s);
for (Iterator it=scanner; it.hasNext())
{    String token =(String) it.next();
      System.out.println(token);
}
```

Ejemplo Scanner

- Ejemplo2(Pendiente):

```
Scanner in=new Scanner(origen);
Pattern comment= Pattern.compile("#.*");
String comm;
//...
while (in.hasNext()){
    if (in.hasNext(comment))   {
       comm = in.nextLine();
     }
    else {
        //proceso de otros tokens
    }
}
```

El método **nextLine()** devuelve lo que queda por leer de la línea actual, desde donde se este hasta el primer fin de línea.

Ejemplo Scanner

- Ejemplo 3:

  Uso de otros delimitadores

```
String s = "Esto hola es hola otro hola
ejemplo";
 Scanner sc =
Scanner.create(s).useDelimiter("\\s*hola\\s*");
 System.out.println(sc.next());
System.out.println(sc.next());
System.out.println(sc.next());
```

Ejemplo Scanner

- Ejemplo4:

```
import java.util.Scanner; // Entrada de un flujo
import java.util.Locale; // Para cambiar la localización

public class Programa {

	public static void main(String[] args){

		Scanner scan = new Scanner( System.in );
		/* Si no ponemos la línea siguiente la
		 * localización espera que utilicemos la ,
		 * como separador de decimales en lugar de .*/
		scan.useLocale( Locale.ENGLISH );

		// Entrada de cadena
		System.out.print( "Introduzca una cadena:" );
		String leida = scan.nextLine();
		System.out.println( "Se introdujo: " + leida );

		// Entrada de enteros
		System.out.print( "Introduzca un entero:" );
		int entero = scan.nextInt();
		System.out.println( "Se introdujo: " + entero );

		// Entrada de reales
		System.out.print( "Introduzca un real:" );
		double real = scan.nextDouble();
		System.out.println( "Se introdujo: " + real );
	}
}
```