Unidad 6

Strings

Contenido

Char (Tipo Primitivo)
Codificación de caracteres
Codificación de caracteres en Java
Secuencia de escape
Conversión char/int

Clase Character
Clasificación de caracteres
Métodos de clasificación de la clase Character
Métodos de conversión de la clase Character

Clase String
Inicialización
Otras inicializaciones
Inicialización con formato
Comparación
Comparación parcial
Comparación alfabética
Longitud
Concatenación
Obtención de caracteres

Obtención de Subcadenas
Obtención de Subcadenas (eliminar caracteres blancos)
Búsqueda
Comprobaciones
Conversión
Separación en partes
Conversión a tabla

char (Tipo primitivo)

Un carácter: una letra, un número, un ideograma, un símbolo.

Cada carácter se identifica mediante un número entero único (*code point*).

El code point también se puede representar en decimal o en hexadecimal usando el prefijo \u con un mínimo de 4 dígitos.

Ejemplo donde se muestra 3 formas de asignar la letra a a una variable:

```
char c;
c = 'a';
c = 97;
c = '\u0061';
```

Codificación de caracteres

Existen diferentes codificaciones de caracteres. Los code points pueden representar diferentes caracteres en diferentes codificaciones.

Existen muchas codificaciones orientadas a los alfabetos de idiomas concretos. Para alfabetos latinos, podemos destacar:

- ASCII
- UNICODE
  - UTF-8
  - UTF-16
- ISO 8859
  - ISO 8859-1 Europa occidental
- Juego de caracteres de Windows
  - Windows-1252 para idiomas occidentales
- ...

Si se crean un texto con una codificación y se trata de mostrar con otra, se pueden visualizar caracteres extraños

Codificación de caracteres JAVA

¿Para saber la codificación por defecto de JAVA?

System.getProperty("file.encoding"); //Mostramos el resultado con System.print()

Windows usa por defecto windows-1252

Por eso al leer ciertos caracteres (por ejemplo, tildes) por consola, se muestran caracteres extraños.

Para evitar esto, se debe indicar en el scanner con qué codificación se leen dichos caracteres:

Scanner sc = new Scanner(System.in, "windows-1252");

Secuencia de escape

Una secuencia de escape está formada por una barra inversa seguida de una letra, un carácter o de una combinación de dígitos.

Una secuencia de escape siempre representa un solo carácter aunque se escriba con dos o más caracteres.

Se utilizan para realizar acciones como salto de línea o para usar caracteres no imprimibles.

Algunas secuencias de escape definidas en Java son:

Secuencia de escape	Descripción
\n	Salto de línea. Sitúa el cursor al principio de la línea siguiente
\b	Retroceso. Mueve el cursor un carácter atrás en la línea actual.
\t	Tabulador horizontal. Mueve el cursor hacia adelante una distancia determinada por el tabulador.
\r	Ir al principio de la línea. Mueve el cursor al principio de la línea actual.
\f	Nueva página. Mueve el cursor al principio de la siguiente página.
\"	Comillas. Permite mostrar por pantalla el caracter *comillas dobles*.
\'	Comilla simple. Permite mostrar por pantalla el carácter *comilla simple*.
\\	Barra inversa.
\udddd	Carácter Unicode. d representa un dígito hexadecimal del caracter Unicode.

Conversión char/int

Se puede asignar un valor entero a una variable **char** siempre que sea de 0 a 65535.

```
int x = 'a';

System.out.println(x);   // Muestra 97 (code point de 'a')

char c = (char) x;

System.out.println(c);    // Muestra a

System.out.println(c+1); // Muestra 98 (code point de 'b')
```

Clase Character

Es la clase envoltorio **wrapper** al tipo primitivo **char** añadiendo funcionalidad. En los recursos del curso tenemos un link explicativo sobre las clases envoltorio **wrapper.**

Con la clase **Character** podemos con los métodos que nos aporta entre otras cosas:

- **Comprobar** si un carácter pertenece a una clasificación concreta (dígitos, letras, caracteres blancos, ...)

- **Conversión** del carácter.

Clasificación de caracteres

Los caracteres se pueden clasificar en los siguientes grupos.

- **Dígitos**: 0..9
- **Letras**: A..Z y a..z
- **Caracteres blancos**: espacio en blanco, el tabulador, ...
- **Otros caracteres**: signos de puntuación, matemáticos, ...

Métodos de clasificación de la clase “Character”

boolean isDigit(char c) indica si el carácter c es un dígito.

boolean isLetter(char c) indica si el carácter c es una letra.

boolean isLetterOrDigit(char c) indica si el carácter c es una letra o un dígito.

boolean isLowerCase(char c) indica si el carácter c es una letra y está en minúscula.

boolean isUpperCase(char c) indica si el carácter c es una letra y está en mayúscula.

boolean isSpaceChar(char c) indica si el carácter c es un espacio vacío (cuando se pulsa la barra espaciadora).

boolean isWhitespace(char c) indica si el carácter c es un espacio en blanco. Incluye: Espacio en blanco (barra espaciadora), retorno de carro (‘\r’), nueva línea (‘\n’), tabulador (‘\t’), otros.

Ejemplo

```
char c = 'A';
if (Character.isUpperCase(c)) {
    System.out.println("Es mayúsculas");
}
```

Métodos de conversión de la clase Character

char toLowerCase(char c) devuelve el caracter c en minúsculas.

char toUpperCase(char c) devuelve el caracter c en mayúsculas.

Si no se puede convertir en minúsculas o mayúsculas, ambos métodos devuelven el carácter c.

Ejemplo

```
char c =  'A';
char d = Character.toLowerCase(c);
System.out.println(d);
      // devuelve 'a'
```

Clase String

No es como en otros lenguajes de programación un array de caracteres.

Es una clase que almacena **cadena de caracteres**.

Aunque se incluye dentro de los tipos primitivos, es una clase por sí misma y no necesita una clase envolvente (wrapper) para poder aplicarle métodos específicos.

Inicialización

Aunque se puede iniciar como un objeto:

```
String cadena = new String("Esto es un texto");
```

Lo normal es hacerlo de forma abreviada:

```
String cadena = "Esto es un texto";
```

El string puede contener cualquier carácter, incluidos los codificados mediante Unicode y las secuencias de escape:

```
String cadena = "Hola\n\tEsto es un símbolo: \"\u2661\"";
System.out.println(cadena);
```

Devuelve:

```
Hola
        Esto es un símbolo: "♡"
```

Otras inicializaciones

Es poco habitual pero se puede crear un string a partir de un array de caracteres:

```
char[] arrayCaracteres = {'c', 'a', 'd', 'e', 'n', 'a'};
String cadena = new String(arrayCaracteres);
```

Es posible crear string de valores:

```
cadena = String.valueOf(-45);
cadena = String.valueOf(2.75);
cadena = String.valueOf(true);
cadena = String.valueOf('A');
```

Inicialización con formato

En ocasiones, podemos querer crear una cadena de caracteres concatenando varios literales de cadena, con otra serie de valores. En estos casos, puede resultar tedioso hacerlo con concatenación (+).

Para casos así, podemos usar el método format, que nos permite utilizar variables dentro de la cadena de formato, que serán sustituidos por valores.

%s: será sustituido por una cadena de caracteres.
%d: será sustituido por un número entero.
%.2f: será sustituido por un número decimal con dos decimales

```
String nombre = "Marta";
int edad = 22;
double altura = 1.697;
String cadena = String.format("%s tiene %d y pesa %.2f", nombre, edad, altura);
System.out.println(cadena);
```

Busca información sobre **String.format** o sobre **System.out.printf** para ver todas las posibilidades de formato.

Cuando solo se quiere formatear para imprimir, se puede usar directamente printf:

```
System.out.printf("%s tiene %d y pesa %.2f", nombre, edad, altura);
```

Comparación

Para comparar dos strings **no podemos usar ==**

La clase Strings tiene sus métodos propios:

- **boolean equals (String otroString)**: Compara este String con el especificado.

```
Boolean salida = "Java".equals("Java"); // retorna true
Boolean salida = "Java".equals("java"); // retorna false
```

- **boolean equalsIgnoreCase (String otroString):** Igual que el anterior pero sin tener en cuenta las mayúsculas y minúsculas.

```
Boolean salida = "Java".equalsIgnoreCase("Java"); // retorna true
Boolean salida = "Java".equalsIgnoreCase("java"); // retorna true
```

Comparación parcial

- **boolean regionMatches(int inicio, String otroString, int inicioOtro, int longitud)**: compara los caracteres indicados en longitud empezando por donde se indica en cada String.

```
String cadena1 = "Hay palabras en este texto.";
String cadena2 = "En esta cadena hay palabras.";
boolean b = cadena1.regionMatches(4, cadena2, 19, 8);  // true
```

- **boolean regionMatches(boolean ignora, int inicio, String otroString, int inicioOtro, int longitud)**: igual que el anterior con la diferencia de que si el valor de ignora es true, la comparación no tienen en cuenta mayúsculas y minúsculas.

Comparación alfabética

- **int compareTo (String otroString):** Compara dos cadenas lexicográficamente (orden alfabético).

```
int salida = s1.compareTo(s2);
// donde s1 y s2 son strings que se comparan
      Esto devuelve la diferencia s1-s2. Si :
       salida < 0 // s1 es menor que s2
       salida = 0 // s1 y s2 son iguales
       salida > 0 // s1 es mayor que s2
```

- **int compareToIgnoreCase (String otroString):** igual que el anterior pero sin importar las mayúsculas y minúsculas.

Longitud

Para obtener la longitud de la cadena de caracteres se usa el método:

int length()

```
"Hola".length(); // retorna 4

"".length(); // retorna 0 (cadena vacía)
```

Es muy útil cuando queremos recorrer un string sin salirnos del índice.

Concatenación

Aunque existe el método**: String concat(String str)**
que concatena la cadena especificada al final de esta cadena.

```
String s1 = "Saca";
String s2 = "corchos;
String salida = s1.concat(s2); // retorna "Sacacorchos"
```

Pero es mucho más cómodo usar el operador **+**

```
String salida = s1 + s2;
```

Obtención de caracteres

- **Char charAt(int i)**: Devuelve el carácter en el índice i.

```
System.out.println("Hola Mundo".charAt(3)); //retorna 'a'
```

Conociendo la longitud, es fácil recorrer con un for un string como si fuera un array usado charAt.

```
for(int i=0; i<cadena.length(); i++) {
    System.out.println(cadena.charAt(i));
}
```

Obtención de subcadenas

- **String substring (int i)**: Devuelve la subcadena del i-ésimo carácter de índice al final.

```
"Sacacorchos".substring(4); // retorna corchos
```

- **String substring (int i, int j)**: Devuelve la subcadena del índice i a j-1.

```
"Sacacorchos".substring(2,6); // retorna caco
```

Obtención de subcadenas (eliminar caracteres blancos)

- **String strip ()**: Devuelve una copia eliminando los caracteres blancos (incluidos tabuladores) al principio y al final de la cadena. **No elimina los que están están entre palabras aunque haya más de uno.**

```
String cadena = "\t   Cadena con  muchos\t espacios.    ";
cadena = cadena.strip();
// cadena = "Cadena con  muchos\t espacios.";
```

- **String stripLeading()**: Igual que *strip* pero solo actúa al principio de la cadena.

- **String stripTrail()**: Igual que *strip* pero solo actúa al final de la cadena.

Búsqueda

- **int indexOf(char c)**: busca la primera ocurrencia del carácter c en la cadena donde se invoca, empezando por el principio. Si no lo encuentra, devuelve -1.

```
String cadena = "Hola Mundo!";
int indice = cadena.indexOf('o');
// indice vale 1;
```

- **int indexOf(String otracadena)**: busca la primera ocurrencia de la otra cadena en la cadena donde se invoca el método.

Búsqueda

- **int indexOf(int c, int inicio)**: busca la primera ocurrencia del carácter pero desde la posición *inicio* en adelante.

```
String cadena = "Hola Mundo!";
int indice = cadena.indexOf('o',2);
// indice vale 9;
```

- **int indexOf(String otracadena, int inicio)**: igual que el anterior pero con busca un string a partir de la posición inicio.

Búsqueda

Los métodos siguientes son similares a los anteriores pero la buscan la primera ocurrencia empezando por el final buscando hacia el inicio del string:

- **int lastIndexOf(int c)**
- **int lastIndexOf(String otracadena)**
- **int lastIndexOf(int c, int final)**
- **int lastIndexOf(String otracadena, int final)**

Comprobaciones

- **boolean isEmpty()**: indica si la cadena está vacía.
- **boolean contains(String subcadena)**: indica si la subcadena se encuentra dentro de la cadena.
- **boolean startsWith(String prefijo)**: comprueba si la cadena que invoca el método comienza con la cadena prefijo.
- **boolean startsWith(String prefijo, int inicio)**: lo mismo que el método anterior pero en lugar de comprobar desde la primera posición, comprueba desde la posición inicio.
- **boolean endsWith(String sufijo)**: comprueba si la cadena que invoca el método finaliza con la cadena sufijo.

Conversión

- **String toLowerCase()**: devuelve una copia de la cadena donde se han convertido todas las letras a minúsculas.
- **String toUpperCase()**: similar al anterior convertido todas las letras a mayúsculas.
- **String replace(char original, char final)**: devuelve una copia de la cadena donde se sustituyen cada vez que aparezca el carácter original por el carácter final.
- **String replace(String original, String final)**: igual que el anterior pero con un string.

Separación en partes

- **String[] split(String separador)**: devuelve una array de Strings que se crea al trocear la cadena cortando por donde se indica mediante “separador”, sin incluir el propio separador en los trozos.

```
String cadena = "Esta cadena se divide por los espacios";
String palabras[] = cadena.split(" ");
// palabras es [“Esta”, “cadena”, “se”, “divide”, “por”, “los”, “espacios”];
```

Conversión a tabla

Ya vimos que podemos convertir crear un string a partir de un array de caracteres:

```
char[] arrayCaracteres = {'c', 'a', 'd', 'e', 'n', 'a'};
String cadena = new String(arrayCaracteres);
```

También podemos hacerlo mediante los métodos:

- **static String valueOf(char[] tabla)**: devuelve un string con los caracteres de la tabla.

```
char[] arrayCaracteres = {'c', 'a', 'd', 'e', 'n', 'a'};
     String cadena = String.valueOf(arrayCaracteres);
```

- **static String valueOf(char[] tabla, int inicio, int cuantos)**: igual que el anterior, pero desde inicio y solo los caracteres indicados en “cuantos”.

De forma inversa, podemos convertir un string en array de caracteres mediante **toCharArray().**

```
String cadena = "Hola mundo!";
char[] arraycaracteres = cadena.toCharArray();
// arraycaracteres vale ['H', 'o', 'l', 'a', ' ', 'm', 'u', 'n', 'd', 'o', '!'];
```