|  |  |  |  |
| --- | --- | --- | --- |
| ***Abreviatura + TAB*** | ***Resultado*** | ***Abreviatura + TAB*** | ***Resultado*** |
| **En** | Enumeration | **newo** | *Object name*= new *Object*(*args*); |
| **Ex** | Exception | **pe** | protected |
| **Ob** | Object | **pr** | private |
| **Psf** | public static final | **psf** | private static final |
| **Psfb** | public static final boolean | **psfb** | private static final boolean |
| **Psfi** | public static final int | **Psfi** | private static final int |
| **Psfs** | public static final String | **psfs** | private static final String |
| **St** | String | **pst** | printStackTrace(); |
| **ab** | abstract | **psvm** | public static void main(String[] args){  } |
| **bo** | boolean | **pu** | public |
| **br** | break | **re** | return |
| **ca** | catch( | **serr** | System.err.println(“”); |
| **cl** | class | **sout** | System.out.println(“”); |
| **cn** | continue | **st** | static |
| **df** | default: | **sw** | switch( |
| **dowhile** | do{  }while(condition); | **sy** | synchronized |
| **eq** | equals | **tds** | Thread.dumpStack(); |
| **ex** | extends | **th** | throws |
| **fa** | false |  |  |
| **fi** | final | **trycatch** | try {}  catch (Exception *e*) {} |
| **fl** | float | **tw** | throw |
| **forc** | for (Iterator *it*= *collection*.iterator();  *it*.hasNext();) {  *Object elem*= (*Object*) *it*.next();  } | **twn** | throw new |
| **fore** | for (*Object elem*: iterable) {  } | **wh** | while( |
| **fori** | for (int *i*= 0; i < *arr*.length; *i*++) {  } | **whileit** | while (*it.*hasNext()) {  *Object elem*= (*Object*) *it*.next();  } |
| **fy** | finally |  |  |
| **ie** | interface |  |  |
| **ifelse** | if (*condition*){}else {  } |  |  |
| **im** | implements |  |  |
| **iof** | instanceof |  |  |
| **ir** | import |  |  |
| **le** | length |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Clase Math del paquete java.lang.** | | **Clase String** | |
| **Método** | **Descripción.** | **Método** | **Descripción.** |
| **abs(double a)** | Devuelve el valor absoluto de un valor double introducido como parámetro. | **charAt(int index)** | Retorna el carácter especificado en la posición index |
| **abs(int a)** | Devuelve el valor absoluto de un valor Entero introducido como parámetro. | **equals(String str)** | Sirve para comparar si dos cadenas son iguales. Devuelve true si son iguales y false si no. |
| **abs(long a)** | Devuelve el valor absoluto de un valor long introducido como parámetro. | **equalsIgnoreCase(String str)** | Sirve para comparar si dos cadenas son iguales, ignorando la grafía de la palabra. Devuelve true si son iguales y false si no. |
| **max(double a, double b)** | Devuelve el mayor de dos valores double | **compareTo(String otraCadena)** | Compara dos cadenas de caracteres alfabéticamente. Retorna 0 si son iguales, entero negativo si la primera es menor o entero positivo si la primera es mayor. |
| **max(int a, int b)** | Devuelve el mayor de dos valores Enteros. | **concat(String str)** | Concatena la cadena del parámetro al final de la primera cadena. |
| **max(long a, long b)** | Devuelve el mayor de dos valores long. | **contains(CharSequence s)** | Retorna true si la cadena contiene la secuencia tipo char del parámetro. |
| **min(double a, double b)** | Devuelve el menor de dos valores double. | **endsWith(String suffix)** | Retorna verdadero si la cadena es igual al objeto del parámetro |
| **min(int a, int b)** | Devuelve el menor de dos valores enteros. | **indexOf(String str)** | Retorna el índice de la primera ocurrencia de la cadena del parámetro |
| **min(long a, long b)** | Devuelve el menor de dos valores long. | **isEmpty()** | Retorna verdadero si la longitud de la cadena es 0 |
| **pow(double a, double b)** | Devuelve el valor del primer argumento elevado a la potencia del segundo argumento. | **length()** | Retorna la longitud de la cadena |
| **random()** | Devuelve un double con un signo positivo, mayor o igual que 0.0 y menor que 1.0. | **replace(char oldChar, char newChar)** | Retorna una nueva cadena reemplazando los caracteres del primer parámetro con el carácter del segundo parámetro |
| **round(double a)** | Devuelve el long redondeado más cercano al double introducido. | **split(String regex)** | Retorna un arreglo de cadenas separadas por la cadena del parámetro |
| **sqrt(double a)** | Devuelve la raíz cuadrada positiva correctamente redondeada de un double. | **startsWith(String prefix)** | Retorna verdadero si el comienzo de la cadena es igual al prefijo del parámetro. |
| **floor(double a)** | Devuelve el entero más cercano por debajo. | **substring(int beginIndex)** | Retorna la sub cadena desde el carácter del parámetro |
| **EJEMPLO**:  int numero = (int) (Math.random() \* 10); | | **substring(int beginIndex, int endIndex)** | Retorna la sub cadena desde el carácter del primer parámetro hasta el carácter del segundo parámetro |
|  |  | **toCharArray()** | Retorna el conjunto de caracteres de la cadena |
|  |  | **toLowerCase()** | Retorna la cadena en minúsculas |
|  |  | **toUpperCase()** | Retorna la cadena en mayúsculas |
|  |  | **EJEMPLO**:  String numCadena = "1";  int numEntero = Integer.parseInt(numCadena); | |

**Tipos de Datos Primitivos**

**Primitivos**: Son predefinidos por el lenguaje. La biblioteca Java proporciona clases asociadas a estos tipos que proporcionan métodos que facilitan su manejo.

|  |  |
| --- | --- |
| **byte** | Es un entero con signo de 8 bits, el mínimo valor que se puede almacenar es -128 y el máximo valor es de 127 (inclusive). |
| **short** | Es un entero con signo de 16 bits. El valor mínimo es -32,768 y el valor máximo 32,767 (inclusive). |
| **int** | Es un entero con signo de 32 bits. El valor mínimo es -2,147,483,648 y el valor máximo es 2,147,483,64 (inclusive). Generalmente es la opción por defecto. |
| **long** | Es un entero con signo de 64 bits, el valor mínimo que puede almacenar este tipo de dato es -9,223,372,036,854,775,808 y el máximo valor es 9,223,372,036,854,775,807 (inclusive). |
| **float** | Es un número decimal de precisión simple de 32 bits (IEEE 754 Punto Flotante). |
| **double** | Es un número decimal de precisión doble de 64 bits (IEEE 754 Punto Flotante). |
| **boolean** | Este tipo de dato sólo soporta dos posibles valores: verdadero o falso y el dato es representado con tan solo un bit de información. |
| **char** | El tipo de dato carácter es un simple carácter unicode de 16 bits. Su valor mínimo es de '\u0000' (En entero es 0) y su valor máximo es de '\uffff' (En entero es 65,535). Nota: un dato de tipo carácter se puede escribir entre comillas simples, por ejemplo 'a', o también indicando su valor Unicode, por ejemplo '\u0061'. |
| **String** | Además de los tipos de datos primitivos el lenguaje de programación Java provee también un soporte especial para cadena de caracteres a través de la **clase** String.  Encerrando la cadena de caracteres con comillas dobles se creará de manera automática una nueva instancia de un objeto tipo String.  String cadena = “Sebastián”;  Los objetos String son inmutables, esto significa que una vez creados, sus valores no pueden ser cambiados. Si bien esta clase no es técnicamente un tipo de dato primitivo, el lenguaje le da un soporte especial y hace parecer como si lo fuera. |