1)String

- **Librería:***Librería estándar de Java*
- **Archivo:***Incluido en el JDK*
- **Desarrollador:***Sun Microsystems*
- **Paquete:***java.lang*

Esta clase representa una cadena de caracteres. Sus métodos permiten obtener información sobre esa cadena (por ejemplo, la longitud) y para obtener cadenas nuevas que se obtienen modificando de alguna forma la cadena original (por ejemplo, pasar a mayúsculas)

String
+ char charAt(int pos) + boolean equals(String s) + boolean equalsIgnoreCase (String s) + boolean endsWith(String s) + int indexOf(String s) + int indexOf(Strings, int pos) + int lastIndexOf (String s) + int length() + String substring(int pos) + String substring(int a, int b) + String toUppercase() + String toLowercase() + String trim()

- **charAt** devuelve el carácter que hay en una posición del String. Hay que tener en cuenta que el primero corresponde a la posición 0.
- **equals** sirve para comparar si una cadena de caracteres es igual a otra. Este método devuelve true si son iguales, y false si son distintas.
- **equalsIgnoreCase** compara si una cadena es igual a otra, ignorando las diferencias entre letras mayúsculas y minúsculas.
- **endsWith** devuelve true si la cadena de caracteres termina como el parámetro.
- El primer **indexOf** devuelve la posición de la primera aparición de la cadena pasada como parámetro. Si no se encuentra, devuelve -1.
- El segundo **indexOf** es como el primero, pero comprobando a partir de la posición indicada en el segundo parámetro.
- El método **lastIndexOf** devuelve la posición de la última aparición de la cadena de caracteres pasada como parámetro, o -1 si no se encuentra.
- El método **length** devuelve el número de caracteres del String.
- El primer método **substring** extrae una subcadena de caracteres, desde la posición pasada como parámetro hasta el final.
- El segundo método **substring** extrae una subcadena de caracteres, que empieza en la posición indicada en el primer parámetro y termina en la posición anterior al segundo parámetro. Es decir, extrae entre “a” y “b-1”.
- **toUppercase** devuelve la cadena de caracteres pasada a mayúsculas.
- **toLowercase** devuelve la cadena de caracteres pasada a minúsculas.
- **trim** devuelve la cadena original sin los espacios que tenga al comienzo y al final.

2)Scanner

- **Librería:***Librería estándar de Java*
- **Archivo:***Incluido en el JDK*
- **Desarrollador:***Sun Microsystems*
- **Paquete:***java.lang*

Esta clase permite obtener datos de tipo básico y String a partir de una fuente, como por ejemplo, el teclado. Por tanto, esta clase nos ayudará para que un usuario pueda escribir datos por teclado en su programa.

Scanner
+ Scanner (InputStream origen) + Scanner (File f) throws FileNotFoundException + String nextLine() + int nextInt() + long nextLong() + double nextDouble() + void close()

- El primer **constructor** crea un objeto para leer de una fuente genérica (InputStream). Si se pone **System.in**, entonces el objeto puede leer datos del teclado.
- El segundo **constructor** crea un objeto para leer datos del archivo de texto que viene representado por medio del objeto File pasado como parámetro.
- El método **nextLine** devuelve un String recogido de la fuente. Por ejemplo, si la fuente es el teclado, el método devuelve el texto que escriba el usuario.
- El método **nextInt** devuelve un int recogido de la fuente. Por ejemplo, si la fuente es el teclado, el método devuelve un número escrito por el usuario
- El método **nextLong** devuelve un long recogido de la fuente. Por ejemplo, si la fuente es el teclado, el método devuelve un número escrito por el usuario
- El método **nextDouble** devuelve un double recogido de la fuente. Por ejemplo, si la fuente es el teclado, el método devuelve un número escrito por el usuario
- El método **close** cierra el objeto y libera los recursos que tenga abiertos. Debe utilizarse este método cuando la fuente es un archivo.

3)Thread

- **Librería:***Librería estándar de Java*
- **Archivo:***Incluido en el JDK*
- **Desarrollador:***Sun Microsystems*
- **Paquete:***java.lang*

La clase Thread se utiliza principalmente para la realización de aplicaciones multitarea. Sin embargo, también dispone de métodos para producir una pausa en la ejecución de un programa.

Thread
+ static void sleep(long ms) throws InterruptedException + static void sleep (long ms, int ns) throws InterruptedException

- primer método **sleep**: Realiza una pausa del número de milisegundos indicado.
- segundo método **sleep**: Realiza una pausa de mayor precisión, ya que permite especificar el número de milisegundos y nanosegundos.

4) Math

- **Librería:***Librería estándar de Java*
- **Archivo:***Incluido en el JDK*
- **Desarrollador:***Sun Microsystems*
- **Paquete:***java.lang*

La clase Math sirve para realizar operaciones matemáticas. Aunque en este diagrama solo se muestran métodos que reciben enteros, también hay versiones de dichos métodos para otros tipos de datos como long o double.

Math
+ static int pow (int a, int b) + static double sqrt (double x) + static int abs (int x) + static double exp (double x) + static double log (double x) + static double log10 (double x) + static double sin (x) + static double cos (x) + static double tan (x) + static double atan (double x) + static int max (int a, int b) + static int min (int a, int b) + static double random() + static double toDegrees (double x) + static double toRadians (double x)

- **pow** calcula el resultado de la potencia a^b
- **sqrt** calcula la raíz cuadrada de x
- **abs** calcula el valor absoluto de x
- **exp** calcula el valor de la función exponencial e^x
- **log** calcula el logaritmo neperiano de x
- **log10** calcula el logaritmo en base 10 de x
- **sin** calcula el valor de la función trigonométrica seno de x
- **cos** calcula el valor de la función trigonométrica coseno de x
- **tan** calcula el valor de la función trigonométrica tangente de x
- atan calcula el valor de la función trigonométrica arco tangente de x
- **max** devuelve el número más grande del conjunto {a,b}
- **min** devuelve el número más pequeño del conjunto {a,b}
- **random** genera un número aleatorio en el intervalo [0,1). La fórmula para obtener un número aleatorio (*con decimales porque es double*) entre [A,B) es:

  A + (B-A)*Math.random()

- **toDegrees** convierte x radianes en grados
- **toRadians** convierte x grados en radianes
- **round** redondea un double al entero más cercano (por ejemplo, 2.9 a 3, y 2.1 a 2)

5) NumberFormat

- **Librería:***Librería estándar de Java*
- **Archivo:***Incluido en el JDK*
- **Desarrollador:***Sun Microsystems*
- **Paquete:***java.text*

Esta clase sirve para convertir tipos de datos numéricos (int, long, etc) en cadenas de texto (String) con el formato indicado.

NumberFormat
+ static NumberFormat getInstance() + void setMinimumFractionDigits(int c) + void setMaximumFractionDigits(int c) + String format(double x)

- El método **getInstance** permite obtener un objeto de la clase NumberFormat
- El método **setMinimumFractionDigits** permite indicar la cantidad mínima de cifras decimales que deberá mostrar la conversión de número a texto.
- El método **setMaximumFractionDigits** permite indicar la cantidad máxima de cifras decimales que queremos que aparezcan en la conversión.
- El método **format** recibe un número y lo retorna en un String usando la cantidad de decimales indicados con los métodos anteriores.

6) System

- **Librería:***Librería estándar de Java*
- **Archivo:***Incluido en el JDK*
- **Desarrollador:***Sun Microsystems*
- **Paquete:***java.lang*

Es una clase que permite obtener información sobre el sistema operativo y las condiciones en las que el usuario ejecuta el programa.

System
+ String getProperty (String propiedad) + String getenv(String nombre) + static void gc() + static void exit(int codigo) + static long currentTimeMillis()

- El método **getProperty** recupera información del equipo. Las cosas que pueden pedirse son:

Propiedad	Valor que devuelve el método
file.separator	El carácter \ o / usado para separar las carpetas en las rutas de archivos
java.home	La carpeta donde está instalado el JRE que está usando el programa
java.vendor	El fabricante de la máquina virtual de Java usado por el programa
java.version	La versión de Java que usa el JRE
os.arch	Arquitectura usada por el sistema operativo
os.name	Nombre del sistema operativao
os.version	Versión del sistema operativo
user.home	Directorio home del usuario que ejecuta el programa
user.name	Nombre del usuario que está ejecutando el programa

- El método **getEnv** devuelve el valor de una variable de entorno definida en el sistema operativo. El método recibe el nombre de dicha variable de entorno.
- El método **gc** solicita a Java que invoque, cuando le sea posible, al recolector de basura
- El método **exit** hace que el programa termine inmediatamente y se devuelva el valor indicado como resultado al sistema operativo.
- El método **currentTimeMillis** devuelve el valor del tiempo Unix de ese instante. El tiempo Unix es el número de milisegundos transcurridos desde el 1 de enero de 1970 (momento en que comenzó a funcionar el sistema operativo Unix) hasta el momento en que se llama al método. Este dato se usa para medir instantes en el tiempo.

7) Runtime

- **Librería:***Librería estándar de Java*
- **Archivo:***Incluido en el JDK*
- **Desarrollador:***Sun Microsystems*
- **Paquete:***java.lang*

Es una clase que permite obtener información sobre el ordenador en el que se está ejecutando el programa. La clase no tiene constructor, pero es posible recuperar un objeto de ella mediante el método getRuntime.

Runtime
+ static Runtime getRuntime() + int availableProcessors() + long freeMemory() + long totalMemory()

- El método estático **getRuntime** permite obtener un objeto de la clase Runtime
- El método **availableProcessors** permite obtener la cantidad de procesadores (o núcleos) del equipo
- El método **freeMemory** devuelve la cantidad de memoria libre disponible para la máquina virtual de Java
- El método **totalMemory** devuelve la cantidad total de memoria que dispone la máquina virtual de Java.

8) NumberFormat

- **Librería:***Librería estándar de Java*
- **Archivo:***Incluido en el JDK*
- **Desarrollador:***Sun Microsystems*
- **Paquete:***java.text*

Esta clase sirve para convertir tipos de datos numéricos (int, long, etc) en cadenas de texto (String) con el formato indicado.

NumberFormat
+ static NumberFormat getInstance() + void setMinimumFractionDigits(int c) + void setMaximumFractionDigits(int c) + String format(double x)

- El método **getInstance** permite obtener un objeto de la clase NumberFormat
- El método **setMinimumFractionDigits** permite indicar la cantidad mínima de cifras decimales que deberá mostrar la conversión de número a texto.
- El método **setMaximumFractionDigits** permite indicar la cantidad máxima de cifras decimales que queremos que aparezcan en la conversión.
- El método **format** recibe un número y lo retorna en un String usando la cantidad de decimales indicados con los métodos anteriores.

9) Clases envoltorio (wrappers)

- **Librería:***Librería estándar de Java*
- **Archivo:***Incluido en el JDK*
- **Desarrollador:***Sun Microsystems*
- **Paquete:***java.lang*

En Java, por cada tipo básico hay una clase llamada "clase envoltorio" que la representa y permite realizar ciertas operaciones comunes con ellos. He aquí las clases envoltorio más utilizadas:

Tipo básico	Clase envoltorio
int	Integer
long	Long
boolean	Boolean
double	Double

Vamos a estudiar solamente la clase Integer, porque las demás son muy similares. Lo principal de estas clases es que todas tienen un método estático **parse** que sirve para convertir un String en el tipo básico indicado. Además, desde la versión 6 de Java es posible utilizar las clases envoltorio en operaciones como si fuesen tipos básicos. Otra ventaja de estas clases es que como son tipos de referencia, sus variables pueden dejarse sin asignar. Pese a sus ventajas, el uso de clases envoltorio en operaciones es más lento que usar directamente los tipos básicos.

Integer
+ Integer (int v) + Integer (String s) + int intValue() + static int parseInt (String s) + static String toBinaryString(int n) + static String toHexString (int n) + static String toString (int n, int base)

- El primer **constructor** crea un objeto a partir del entero básico pasado como parámetro.
- El segundo **constructor** crea un objeto a partir de un número guardado en un String.
- El método **intValue** devuelve el valor del entero en forma del tipo básico int.
- El método **parseInt** devuelve un número entero de tipo básico a partir del String pasado como parámetro. Aunque no está marcado como peligroso y por tanto no es obligatorio encerrarlo entre try-catch, lanza una excepción si el String no contiene un número entero válido.
- El método **toBinaryString** devuelve un String con la conversión a binario del número pasado como parámetro.
- El método **toHexString** pasa a hexadecimal el número pasado como parámetro.
- El método **toString** devuelve la representación del número en la base indicada.

10) StringTokenizer

- **Librería:***Librería estándar de Java*
- **Archivo:***Incluido en el JDK*
- **Desarrollador:***Sun Microsystems*
- **Paquete:***java.util*

Esta clase permite "trocear" una cadena de caracteres en varias secciones a las que podemos tener acceso de forma individual. Las secciones deben estar separadas unas de otras dentro de la cadena original por medio de una cadena de caracteres llamada "separador".

StringTokenizer
+ StringTokenizer (String cadena, String separador) + String nextToken() + boolean hasMoreTokens()

- El **constructor** crea un objeto StringTokenizer que parte la cadena pasada como primer parámetro en trozos delimitados por el segundo parámetro. Los diferentes "trozos" están a disposición del programador, que puede acceder a ellos de forma individual con el método nextToken.
- El método **nextToken** devuelve un "trozo" de la cadena original y pasa al siguiente. Es decir, cada vez que se llama a este método, el trozo devuelto es borrado del StringTokenizer y se pasa al siguiente trozo disponible. El método nextToken devuelve null si ya no hay más trozos por recorrer.
- El método **hasMoreTokens** devuelve true si quedan más trozos dentro del StringTokenizer.

11) Random

- **Librería:***Librería estándar de Java*
- **Archivo:***Incluido en el JDK*
- **Desarrollador:***Sun Microsystems*
- **Paquete:***java.lang*

Esta clase sirve para obtener un generador de números aleatorios.

Random
+ Random() + int nextInt(int tope) + double nextDouble() + boolean nextBoolean()

- El método **constructor** crea un generador de números aleatorios
- El método **nextInt** devuelve un número aleatorio dentro del intervalo [0,tope). Esto significa que el valor de "tope" no llega a alcanzarse. Si queremos un número aleatorio en el intervalo comprendido entre dos números [A,B), deberemos usar esta fórmula:

  Número aleatorio = A + nextInt(B-A)

- El método **nextDouble** genera un número aleatorio dentro del intervalo [0,1). Por tanto, el 1 nunca sale.
- El método **nextBoolean** devuelve un boolean aleatorio entre true y false.

12) Desktop

- **Librería:***Librería estándar de Java*
- **Archivo:***Incluido en el JDK*
- **Desarrollador:***Sun Microsystems*
- **Paquete:***java.awt*

La clase Desktop sirve para realizar acciones comunes usando los programas por defecto que el sistema operativo tiene instalados.

Desktop
+ static Desktop getDesktop() + void mail() throws Exception + open(File f) throws IOException + print(File f) throws IOException + edit(File f) throws IOException + browse (URL u) throws IOException

- El método estático **getDesktop** permite obtener un objeto de la clase Desktop. No hay otra forma de obtenerlo, puesto que esta clase no tiene método constructor.
- El método **mail** abre el programa de correo electrónico que esté configurado por defecto en el sistema operativo.
- El método **open** abre un archivo usando el programa que el sistema operativo tenga configurado para ese tipo de archivo. Recibe un objeto de la clase File que haya sido creado con la ruta del archivo que se desea abrir.
- El método **print** imprime el archivo que se pasa como parámetro. Recibe un objeto de la clase File que haya sido creado con la ruta del archivo que se desea abrir.
- El método **edit** abre un programa adecuado para editar el archivo que se pasa como parámetro. Recibe un objeto de la clase File que haya sido creado con la ruta del archivo que se desea abrir.
- El método **browse** abre el navegador que el sistema operativo tenga configurado por defecto y muestra la página web cuya url se pasa como parámetro. Recibe un objeto de la clase URL con la dirección que se desea visitar.

13) StringBuilder

- **Librería:***Librería estándar de Java*
- **Archivo:***Incluido en el JDK*
- **Desarrollador:***Sun Microsystems*
- **Paquete:***java.lang*

La clase StringBuilderes una cadena de caracteres mutable. Esto significa que sus métodos modifican dicha cadena de caracteres, a diferencia de lo que sucede con el String, donde sus métodos la devuelven transformada, pero sin modificar la cadena original.

StringBuilder
+ StringBuilder(String s) + String toString() + char charAt(int pos) + int indexOf(String s) + String substring(int a, int b) + int length() + StringBuilder append(String s) + StringBuilder insert(int pos, String s) + StringBuilder delete(int a, int b) + StringBuilder deleteChar(int pos) + StringBuilder reverse() + StringBuilder replace(int a, int b, String s) + StringBuilder subsequence(int a, int b) + void setLength(int n)

- El método **constructor** crea un StringBuilder con la cadena de caracteres “s”
- El método **toString** devuelve el String que hay en el StringBuilder
- El método **charAt** devuelve el carácter que está en la posición indicada como parámetro. La primera posición es la 0.
- El método **indexOf** devuelve la posición donde se encuentra el String pasado como argumento. Si no se encuentra, devuelve -1
- El método **substring** devuelve el texto comprendido entre las posiciones [a,b)
- El método **length** devuelve el total de caracteres del StringBuilder
- El método **append** añade el String indicado al final del StringBuilder. Existen sobrecargas de este método para admitir cualquier tipo básico: char, boolean, long, etc
- El método **insert** inserta el String pasado como parámetro en la posición indicada. Existen sobrecargas de este método para admitir cualquier tipo básico: char, long, etc.
- El método **delete** elimina todos los caracteres comprendidos entre las posiciones [a,b)
- El método **deleteChar** elimina el carácter que se encuentra en la posición indicada.
- El método **reverse** invierte la cadena de caracteres del StringBuilder
- El método **replace** elimina todos los caracteres entre las posiciones [a,b) para después insertar a partir de “a” el String s.
- El método **subSequence** permite obtener un nuevo StringBuilder con la cadena de caracteres comprendida entre las posiciones [a,b)
- El método **setLength** cambia la longitud de la cadena de caracteres del StringBuilder. Para ello recortará la cadena, si la nueva longitud es menor, o la completará con caracteres nulos (no confundir con espacios) si es mayor.

14) Properties

- **Librería:***Librería estándar de Java*
- **Archivo:***Incluido en el JDK*
- **Desarrollador:***Sun Microsystems*
- **Paquete:***java.util*

La clase Properties se utiliza para guardar un conjunto de opciones de configuración y sus valores. Por tanto, está formada por parejas (opción,valor) que se le van añadiendo al objeto Properties. Lo bueno de esta clase es que tiene métodos que permiten guardar y cargar desde disco estas opciones y sus valores, por lo que se usan mucho para guardar la configuración de los programas.

Properties
+ Properties () + void setProperty (String opción, String valor) + String getProperty (String opción) + int size() + Enumeration<String> keys() + void storeToXML (OutputStream salida, String comentario, String codificación) throws Exception + void loadFromXML (InputStream entrada) throws Exception

- El **constructor** crea un objeto Properties vacio, sin ningún parámetro de configuración registrado en él.

- El método **setProperty** añade un parámetro de configuración y su valor. El parámetro de configuración y su valor es algo libre y es decidido por el programador de aplicaciones que usa esta clase.

- El método **getProperty** devuelve el valor del parámetro de configuración que se pasa como parámetro. Se supone que dicho parámetro ha sido previamente registrado con el método setProperty. En caso contrario, este método devolverá null.

- El método **size** devuelve el número de parámetros registrados en el objeto.

- El método **keys** devuelve la lista de todos los parámetros registrados en el objeto.

- El método **storeToXML** graba sobre un canal de salida un documento XML con la colección de parámetros registrados y sus valores. El primer parámetro es el canal donde se guardará el documento, el segundo es un comentario descriptivo y el tercero, el sistema de codificación usado, que puede ser "UTF-8" o "ISO-8859-1".

- El método **loadFromXML** carga desde un canal de entrada, los parámetros y valores almacenados en un documento XML que previamente ha sido guardado con el método storeToXML.

15) ClassLoader

- **Librería:***Librería estándar de Java*
- **Archivo:***Incluido en el JDK*
- **Desarrollador:***Sun Microsystems*
- **Paquete:***java.lang*

Esta clase tiene dos funciones:

- Como sabemos, los programas Java utilizan las clases que se encuentran en las librerías. Para ahorrar memoria, la máquina virtual usa un sistema de **carga dinámica de tipos**, en el que cada clase se carga en memoria desde el momento en que un programa la necesita (por ejemplo, la clase NumberFormat solo se carga en memoria cuando un programa va a crear un objeto de dicha clase). La clase ClassLoader es la encargada de cargar las clases en la memoria. Puede ser usada por el programador de aplicaciones para cargar clases que aún no han sido utilizadas por el programa.

- Los programas Java normalmente se distribuyen como archivos JARcreados por el IDE. Es posible incluir en los archivos JAR archivos, como gráficos, sonidos, etc, para que sean distribuidos junto con el bytecode del programa. Los archivos incluidos en el JAR se denominan **recursos**. Con la clase ClassLoader podemos tener acceso a cualquier recurso incluido en el JAR de la aplicación.

ClassLoader
+ static ClassLoader getSystemClassLoader() + URL getResource(String nombre) + InputStream getResourceAsStream(String nombre) + Enumeration<URL> getResources (String patrón) + static URL getSystemResource (String nombre) + Class loadClass(String nombre) throws ClassNotFoundException

- El método **getSystemClassLoader** obtiene el objeto ClassLoader que utiliza la máquina virtual de Java para cargar clases y recursos.
- El método **getResource** recibe la ruta del recurso dentro del archivo JAR (se usa el símbolo **/** para separar los paquetes de la ruta) y devuelve un objeto de la clase URL (esta clase está en el paquete java.net) que permite al programa tener acceso a él.
- El método **getResourceAsStrean** es como el anterior, pero en lugar de un objeto URL devuelve el acceso al recurso en forma de canal de entrada de bytes.
- El método **getResources** devuelve todos los recursos que tienen el nombre indiciado.
- El método **getSystemResource** es como el getResource, pero en versión estática y utiliza el classloader por defecto de la máquina virtual.
- El método **loadClass** carga una clase cuyo nombre completo (paquete junto con el nombre de la clase) se pasa como parámetro, y devuelve un objeto Class con el que podemos obtener datos de la clase y manejarla mediante reflexión.