UD03.API de JAVA y otras librerías

DAM1-Programación 2024-25

- Java ® Platform, Standard Edition & Java Development Kit Version 21 API Specification
- Java Development Kit 21 Documentation

Introducción a la API de Java

Java, uno de los lenguajes de programación más populares, se destaca por su portabilidad y su extensa biblioteca estándar, conocida como la API de Java (Interfaz de Programación de Aplicaciones).

La API de Java proporciona un conjunto de clases y métodos que facilitan el desarrollo de aplicaciones en Java, permitiéndote aprovechar la potencia del lenguaje de manera efectiva.

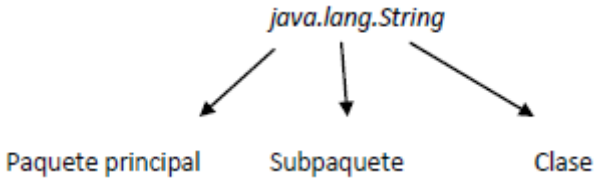
Esta biblioteca estándar abarca desde operaciones básicas hasta funcionalidades avanzadas, proporcionando herramientas para manipular estructuras de datos, realizar operaciones de entrada/salida, trabajar con redes, y mucho más.

La documentación oficial de la API de Java es una valiosa fuente de referencia. Puede variar con cada versión del lenguaje (JDK). Es posible descargar la documentación para acceder a ella sin conexión a Internet.

Paquetes y módulos. Organización de las clases

La API de Java está organizada en **paquetes**, cada uno diseñado para un propósito específico.
Los paquetes organizan las clases de una manera estructurada y jerárquica. Un paquete es una carpeta en la que se agrupan clases que tienen algún tipo de relación lógica. Esto facilita su localización y utilización en un programa. Un paquete puede a su vez estar compuesto de otros subpaquetes. La organización en paquetes también evita conflictos de nombres ya que permite tener clases con el mismo nombre en distintos paquetes.

Existe un **paquete java** dentro del cual existen subpaquetes. El **nombre cualificado de la clase** se compone del nombre de la clase, precedido por los nombres de los subpaquetes en donde se encuentra hasta llegar al paquete principal, separados por un "."



Las clases de un mismo paquete están relacionadas entre sí o tratan un tema común.

Sentencias package e import

Al implementar una clase debemos indicar al inicio del código el paquete (o carpeta) que la contiene utilizando la sentencia **package**:

```
package nombrepaquete;
```

Según las reglas de estilo de Java, los paquetes se escriben con minúsculas y sin utilizar espacios para separar palabras.

Cuando desde el código queremos usar una clase que se encuentra en un paquete diferente podemos utilizar su nombre cualificado o “importarla” al inicio del código con la sentencia **import**. De ese modo podemos hacer referencia a la clase con tan solo utilizar su nombre.

```
import nombre_cualificado_de_la_clase;
```

Paquetes comunes

- **java.lang**: Contiene clases fundamentales para la realización de cualquier programa Java como System, String, Math, Objetc, clases envoltorio, etc. El compilador importa el paquete completo de forma implícita, por lo que no deberá utilizarse la sentencia import para importar clases de este paquete.
- **java.util**: Contiene clases para utilidades variadas, como el tratamiento de Arrays y colecciones (listas, conjuntos y mapas) de objetos, la manipulación de fechas y horas o la construcción de expresiones regulares. La clase Scanner utilizada para la entrada por teclado de diferentes tipos de datos, la clase Date, para el tratamiento de fechas, etc
- **java.time**: Contiene las clases LocalDateTime, LocalDate y LocalTime para gestión de fechas y horas. La clasesDateTimeFormatter permiten formatear y analizar fechas y horas según patrones específicos.
- **java.math**: Contiene las clases BigInteger y BigDecimal que proporcionan funcionalidades matemáticas avanzadas con mayor precisión.
- **java.io**: Contiene las clases para la gestión de la entrada y salida de datos en Java. Independientemente del dispositivo E/S que se utilice, Java utiliza siempre las mismas clases para enviar datos a la salida y leer datos de la entrada.
- **java.sql:** proporciona las clases y interfaces necesarias para interactuar con bases de datos mediante el estándar JDBC (Java Database Connectivity). Algunas clases clave incluyen Connection para establecer la conexión con la base de datos, Statement para ejecutar consultas SQL y ResultSet para manejar los resultados de las consultas.
- **java.net:** para el desarrollo de aplicaciones que requieren comunicación a través de la red. Contiene clases como URL para representar direcciones URL y URLConnection para abrir conexiones a recursos remotos.
- **java.awt:** parte del Abstract Window Toolkit (AWT) y proporciona clases para crear interfaces de usuario gráficas. Clases notables incluyen Frame para crear ventanas, Button para botones y Canvas para áreas de dibujo personalizadas.
- **javax.swing:** extiende AWT y se centra en la creación de interfaces gráficas de usuario más modernas y ricas. Incluye clases como JFrame para ventanas mejoradas, JButton para botones mejorados y JPanel para contenedores de componentes.
- **javafx:** es un conjunto más moderno y avanzado de bibliotecas gráficas para Java, que permite el desarrollo de aplicaciones de interfaz de usuario ricas y multimedia. Contiene clases como Stage para representar ventanas de la aplicación, Scene para definir el contenido de una ventana y Button para botones interactivos, entre otras.
- etc.

Clase Math

```
import java.lang.Math;
```

La clase Math del paquete java.lang proporciona métodos para la realización de operaciones matemáticas trigonométricas, raíces cuadradas, potencias, etc. Todos sus métodos son estáticos, por lo que no necesitamos crear ninguna instancia de la clase para su utilización.

Ejercicios.

Paquete: ejerciciosmath

ClaseMath. Escribe un programa que genere y muestre un número aleatorio **no entero** con decimales en el rango de -360 a 360 utilizando `Math.random()`.
A continuación realizará y mostrará los siguientes cálculos:

- su raíz cuadrada utilizando el método `Math.sqrt()`.
- el número redondeado a la cifra entera más cercana utilizando `Math.round()`.
- el número redondeado hacia arriba y hacia abajo utilizando los métodos `Math.ceil()` y `Math.floor()`.
- su valor absoluto utilizando `Math.abs()`.
- La potencia al cubo de ese número utilizando el método `Math.pow()`.
- su logaritmo natural utilizando `Math.log()`.
- suponiendo que el número representa la medida un ángulo en grados, muestra el seno y coseno de ese ángulo utilizando `Math.sin()` y `Math.cos()`.

Solicita al usuario que ingrese otros dos números y calcula y muestra el mayor de los 3 (el generado y los introducidos por teclado) utilizando `Math.max()`. Calcula y muestra también el menos con `Math.min()`.

Observa los resultados de sqrt() y log() cuando el número es negativo.

Circulo. Solicita al usuario que ingrese el radio de un círculo y calcula su 4y su perímetro utilizando `Math.PI` y `Math.pow()`. Suponiendo que el radio sea el de una esfera, calcula también su volumen usando la fórmula:

$$V = \frac{4}{3}\pi r^3.$$

ConversionRadianesGrados. Desarrolla dos funciones que conviertan una medida en radianes a grados, y viceversa. Desarrolla un programa que pida al usuario ingresar la medida en radianes y utiliza la fórmula

$$\text{Grados} = \frac{\text{Radianes} \times 180}{\pi}$$

Hipotenusa. Escribe un programa que tome como entrada la longitud de dos lados de un triángulo rectángulo y calcule la longitud de la hipotenusa utilizando el teorema de Pitágoras:

$\sqrt{a^2 + b^2}$

InteresCompuesto. Solicita al usuario que ingrese la cantidad de dinero inicial, la tasa de interés anual y el número de años. Calcula el monto final utilizando la fórmula del interés compuesto:

$P(1 + \frac{r}{n})^{nt}$, donde

- **P** es el principal, o cantidad de dinero inicial.
- **r** es la tasa de interés anual, debe estar en formato decimal, por lo que si la tasa de interés es del **5%**, **r** sería **0.05**.
- **n** es el número de veces que se compone el interés por año (para los cálculos de este problema suponer una vez al año **n=1**)
- **t** es el tiempo en años.

DistanciaPuntosPlanoCartesiano. Permite al usuario ingresar las coordenadas (x, y) de dos puntos en un plano cartesiano. Calcula la distancia entre estos dos puntos utilizando la fórmula de distancia euclidiana:

$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

Paquete java.math

Es posible hacer cálculos que requieran mucha precisión o que trabajen con números muy grandes con los tipos de datos primitivos y los métodos de la clase Math?

El paquete java.math en Java proporciona funcionalidades matemáticas avanzadas y precisión arbitraria para operaciones numéricas. Entre sus principales utilidades se encuentran las operaciones con números grandes, como enteros arbitrariamente grandes y números decimales con alta precisión, a través de las clases BigInteger y BigDecimal, respectivamente. Esto resulta crucial en situaciones donde la precisión estándar de los tipos primitivos de Java podría ser insuficiente, como en cálculos financieros, científicos o en contextos donde se requiere manejar números extremadamente grandes o pequeños.

Ejercicios.

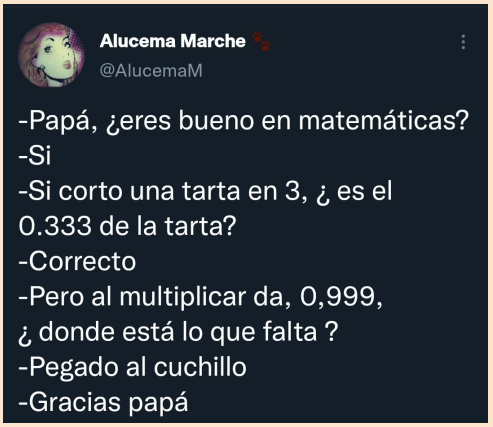
Investiga las clases BigDecimal y BigInteger del paquete java.math.

Cálculos con números muy grandes.

Factorial. Podrías hacer un programa que calcule, por ejemplo, el **factorial de 50** (50!)?

BigDecimal para cálculos aritméticos financieros precisos.

Precision. Escribe un programa para probar el siguiente contenido.



Escribe el siguiente código:

```
double unCentimo = 0.01;

double suma = unCentimo + unCentimo + unCentimo + unCentimo + unCentimo + unCentimo;

System.out.println(suma);
```

Que imprime? Si contestaste: **0.06**, estas equivocado.

Imprime: **0.060000000000000005**

Las computadoras no cuentan como nosotros, las computadoras "cuentan en binario", nosotros, contamos en decimal.
Cuando usamos una variable float o double, estamos dejandole al microprocesador de la computadora el trabajo de efectuar los cálculos directamente (con su coprocesador matemático) pero como las computadoras "piensan" en binario, cometen errores de precisión diferentes a los nuestros.

Por ejemplo, nosotros los humanos, tenemos un problema para representar a la tercera parte de "1" (1/3), y la escribimos: 0.3333333333333333333... (hasta el infinito), en nuestro sistema decimal (base 10), no hay modo de representar a un tercio de un entero. Si por otro lado, nuestro sistema fuera base "9", entonces representariamos la tercera parte de un entero simplemente con : "0.3" (y seria mas preciso que cualquier lista larga de números "3" en base 10.

En binario (base 2), no se puede representar a 1/10 (la décima parte de 1) ni a 1/100 (la centésima parte de 1) y por eso, cuando nosotros escribimos "0.01" la computadora lo entiende como 1100110011001100110011001100110011... (hasta el infinito) y efectuá el calculo incorrectamente (desde nuestro punto de vista "decimal").

Si hacen sistemas financieros, los pequeños errores de calculo acumulados con varios centavos pueden resultar en reportes cuyos cálculos diferirán de las operaciones que normalmente realizamos los humanos. Para resolver este problema, en Java se incluye la clase BigDecimal, que si bien realiza los cálculos mas lento que "double" los realiza del mismo modo que los humanos (en base 10) y así evitamos el problema.

Algunos microprocesadores especializados si son capaces de efectuar "por hardware" los cálculos de forma decimal.

Investiga qué clases ofrece la API de Java para realizar operaciones reales con una precisión tan alta como se necesite. Reescribe el código al inicio de este apartado para que el resultado que se imprima tras las operaciones sea 0.06.

- **Ojo!** No se trata de redondear el resultado, sino de que se realice el cálculo con exactitud.

Clase Random

El método `Math.random()` sólo puede producir valores aleatorios de tipo double en el rango 0.0 <= x < 1.0. La **clase Random** del paquete **java.util** permite crear objetos que pueden producir valores aleatorios de tipo boolean, byte, float, double, int y long.

Primero crearemos un objeto aleatorio::

```
Random numeroAleatorio = new Random();
```

Después, usaremos los métodos del objeto para generar valores aleatorios.

nextInt(): genera un valor int aleatorio en el rango de un int. También podemos limitar el rango de valores posibles utilizando un parámetro en una versión sobrecargada del método.

Por ejemplo:

```
numero = Desplazamiento + numeroAleatorio.nextInt(numerosEnRango)
```

Los valores devueltos son en realidad números seudoaleatorios, es decir una secuencia de valores producidos por un cálculo matemático complejo. Ese cálculo utiliza la hora actual del día (que desde luego cambia constantemente) para sembrar el generador de números aleatorios, de tal forma que cada ejecución de un programa produzca una secuencia distinta de valores aleatorios.

Algunas veces es útil repetir la misma secuencia exacta de números seudoaleatorios durante cada ejecución del programa. Podemos hacerlo indicando un valor concreto de semilla al crear el objeto aleatorio.

```
Random numeroAleatorio = new Random(valorSemilla);
```

El argumento valorSemilla (de tipo long) siembra el cálculo del número aleatorio. Si se utiliza siempre el mismo valor para valorSemilla, el objeto Random produce la misma secuencia de números aleatorios. Para establecer la semilla de un objeto Random en cualquier momento durante la ejecución de un programa, podemos llamar al método **setSeed()** del objeto.

Clases LocalDate, LocalTime, LocalDateTime

Las clases LocalDate, LocalTime, y LocalDateTime son parte del paquete **java.time** introducido en Java 8 para manejar operaciones relacionadas con fechas y tiempos.

- **LocalDate**: Esta clase representa una fecha sin información de tiempo, como el año, el mes y el día. Es inmutable y ofrece métodos para realizar operaciones comunes, como sumar o restar días, comparar fechas y formatearlas. Es útil en situaciones donde solo se necesita la información de la fecha, como en aplicaciones de agenda o planificación.

```
LocalDate fechaActual = LocalDate.now();
```

- **LocalTime**: Representa un instante específico del día sin información de fecha. Proporciona métodos para manipular y formatear la hora. Puede ser útil en aplicaciones que requieren un seguimiento preciso de eventos temporales, como registros de tiempo o temporizadores.

```
LocalTime horaActual = LocalTime.now();
```

- **LocalDateTime**: Combina la información de fecha y hora en un solo objeto. Es útil cuando se necesita representar un punto específico en el tiempo, como marcar un evento con una marca de tiempo completa.

```
LocalDateTime fechaHoraActual = LocalDateTime.now();
```

Algunos métodos comunes para estas clases incluyen **plus**, **minus**, **compareTo**, y **format**, que permiten realizar operaciones aritméticas, comparar fechas, y formatear las instancias según un patrón específico.

Estas clases son inmutables, lo que significa que cualquier operación que modifique la fecha o el tiempo devolverá una nueva instancia en lugar de modificar la existente, garantizando así la invariabilidad y facilitando el manejo seguro en entornos concurrentes.

Su introducción en Java ha mejorado significativamente la manipulación de fechas y tiempos, proporcionando una API más moderna y robusta en comparación con las clases más antiguas como **Date** y **Calendar**.

Otras clases útiles del paquete java.time son **format.DateTimeFormatter** y **ZonedDateTime**.

Ejercicios.

Paquete: ejerciciostime

DuracionActividad.

Desarrolla una aplicación que calcule la duración total de una actividad. La actividad tiene una hora de inicio (LocalTime) y una hora de finalización (LocalTime). El programa debe mostrar la duración total en horas y minutos.

Cumple.
Escribe un programa que solicite al usuario su fecha de nacimiento (LocalDate) y calcule el tiempo restante hasta su próximo cumpleaños. Muestra el resultado en días, horas y minutos.

Cronometro.
Escribe un programa que ofrezca al usuario la posibilidad de iniciar un cronómetro pulsando enter.
Una vez iniciado, el programa quedará a la espera de la entrada del usuario.
Cuando el usuario pulse ENTER de nuevo, el programa mostrará el tiempo transcurrido desde el inicio del cronómetro, en minutos y segundos. También mostrará el tiempo parcial desde la última vez que se pulsó ENTER.

El programa terminará cuando el usuario introduce un cero (0).

IntervaloFechas.
Escribe un programa que tome dos fechas (LocalDate) como entrada y determine el número de días, meses y años entre ellas. Muestra el resultado de manera clara y precisa.

Clases envoltorio

- Clases Envolventes

Existen ocho clases de envoltorio, una por cada tipo primitivo: Byte, Short, Character, Integer, Long, Float, Double y Boolean, todas ellas se encuentran en el paquete java.lang.

Se utilizan fundamentalmente para:

- Encapsular un dato básico en un objeto. En el amplio conjunto de clases Java, existen determinados métodos que realizan algún tipo de manipulación con datos de tipo objeto. Los datos tienen que ser objetos, por ello deberá ser necesario encapsularlos en un objeto.
- Conversión de cadena a tipo básico. En la mayoría de las operaciones de entrada de datos, éstos llegan a la aplicación en forma de cadena de caracteres. Las clases de envoltorio proporcionan métodos estáticos que permiten convertir una cadena de texto, formada por caracteres numéricos, en un tipo numérico.

Encapsulamiento de un tipo básico.
Todas las clases envoltorio permiten crear un objeto de la clase a partir del tipo básico. Por ejemplo, podríamos crear un objeto **Integer** a partir de un **int** de la siguiente forma:

```
int k = 23;
Integer num = new Integer(k);
```

A excepción de **Character**, las clases de envoltorio también permiten crear objetos partiendo de la representación como cadena del dato.
La siguiente instrucción crea un objeto **Float** a partir de la cadena numérica:

```
String s = "4.65";
Float ft = new Float(s);
```

Para recuperar el valor a partir del objeto envoltorio, existe el método con el formato **xxxValue()** que devuelve el dato encapsulado en el objeto, donde **xxx** representa el nombre del tipo en el que se quiere obtener el dato.
Por ejemplo, para obtener el valor de los datos envueltos en los objetos de las instrucciones anteriores utilizaríamos:

```
float dato = ft.floatValue();
int n = num.intValue();
```

Conversión de cadena a tipo básico.
Las clases numéricas proporcionan un método estático **parseXxx(String)** que permite convertir la representación en forma de cadena de un número en el correspondiente tipo numérico, donde **Xxx** es el nombre del tipo al que se va a convertir la cadena de caracteres, en función de la clase que se utilice.

Por ejemplo, en la clase **Integer** tenemos el método **parseInt()**, mientras que en **Double** encontramos **parseDouble()**:

```
String s1 = "25", s2= "89.2";
int n = Integer.parseInt(s1);
double d = Double.parseDouble(s2);
```

Autoboxing.
El autoboxing consiste en la encapsulación automática de un dato básico en un objeto de envoltorio, mediante la utilización del operador de asignación.
Por ejemplo, para encapsular un dato entero de tipo **int** en un objeto **Integer**:

```
int p = 5;
Integer n = new Integer(p);
```

Utilizando el autoboxing la operación anterior puede realizarse de la siguiente forma:

```
int p = 5;
Integer n = p;
```

Es decir, el objeto de envoltorio se crea automáticamente al asignar el dato a la variable objeto.

De la misma forma, para obtener el dato básico a partir del objeto de envoltorio no será necesario recurrir al método **xxxValue()**, esto se realizará implícitamente al utilizar la variable objeto en una asignación.

Por ejemplo, para recuperar el dato encapsulado en el objeto n del ejemplo anterior utilizaríamos:

```
int a = n;
```

A la operación anterior se la conoce como **autounboxing**.

El autoboxing/autounboxing permite al programador despreocuparse de tener que realizar de forma manual el encapsulado de un tipo básico y su posterior recuperación, reduciendo el número de errores por esta causa.

Conversiones de Base usando la clase Integer

Cómo convertir cualquier número de cualquier base a decimal

En el siguiente código, convertimos un número binario a decimal, declarándonos un int. Con el método **parseInt**, el primer parámetro será el número en binario (en forma de String), y en el siguiente, su base (2).

```
int a = Integer.parseInt("101", 2);
System.out.println (a);
```

El resultado por consola, será un 5.

Otro ejemplo de lo mismo:

```
int a = Integer.parseInt("11", 16);
System.out.println (a);
```

El número 11 en hexadecimal, es el 17 en decimal, que es lo que muestra por consola ese código. Esto es aplicable a cualquier base.

Convertir un número decimal, a cualquier base

Con el método **toString** de la clase Integer, podremos convertir un número decimal a la base que queramos de forma muy similar al ejemplo anterior.
En este caso guardamos el resultado en un String, y pasamos como primer parámetro el número decimal, y como segundo la base a la que queremos convertirlo.
El resultado por consola será el número decimal en la base indicada. Por ejemplo:

```
String b = Integer.toString(7, 2);
System.out.println (b);
```

El número 7, en base binaria es el 111, que será el resultado por consola.

```
String c = Integer.toString(11, 16);
System.out.println (c);
```

El número 11 en hexadecimal es la letra B.

Librerías externas

Java Database Connectivity (JDBC): API de Java que proporciona métodos para conectarse a una base de datos, enviar consultas y actualizar datos. Permite a los desarrolladores interactuar con bases de datos relacionales de manera programática desde aplicaciones Java.

Java Persistence API (JPA): API de Java que proporciona un estándar de mapeo objeto-relacional para el desarrollo de aplicaciones Java. Es una especificación que permite a los desarrolladores trabajar con datos de bases de datos utilizando objetos Java en lugar de consultas SQL directas. Hibernate es una de las implementaciones más populares y ampliamente utilizadas de la especificación JPA.

Spring Framework: es un marco de desarrollo de aplicaciones Java que proporciona una amplia variedad de características y servicios para simplificar el desarrollo de software empresarial, escalable y mantenible. Facilita la creación de componentes reutilizables y promueve buenas prácticas de desarrollo.

Gson: es una librería de Java desarrollada por Google que permite la serialización y deserialización de objetos Java a y desde formato JSON. Facilita la integración de aplicaciones Java con servicios web que utilizan JSON como formato de intercambio de datos. Es fácil de usar, dispone de buena documentación y es ampliamente utilizado en proyectos pequeños. Para tareas sencillas también se puede utilizar la librería org.json y para proyectos más grande Jackson JSON.

Apache Commons: es un proyecto de la Apache Software Foundation que proporciona un conjunto de librerías de propósito general que añaden funcionalidades adicionales y otras utilidades.

JUnit: es un marco de pruebas unitarias para Java. Facilita la escritura y ejecución de pruebas unitarias para validar el comportamiento de componentes individuales de software. JUnit es ampliamente utilizado en el desarrollo de software basado en pruebas y sigue el enfoque de desarrollo dirigido por pruebas (TDD), donde las pruebas se crean antes del código de implementación.

- Download JAR files with all dependencies
- Driver JDBC MySQL :: Download Connector/J

Anexos

Empaquetado de Aplicaciones: JAR (Java Archive)

Creación de JARs

Un archivo JAR es un archivo ZIP que contiene clases compiladas, recursos y metadatos. Para crear un JAR, se utiliza la herramienta jar que viene con el JDK.

```
jar cf MiArchivo.jar -C directorio_de_clases .
```

- `cf`: Crea un nuevo archivo JAR y especifica el nombre.
- `-C`: Cambia al directorio especificado antes de realizar la operación.
- `directorio_de_clases`: Directorio que contiene las clases a incluir en el JAR.
- `.`: Incluye todos los archivos del directorio actual.

Ejecución de JARs

Para ejecutar una aplicación empaquetada en un JAR:

```
java -jar MiArchivo.jar
```

Esquemas de paquetes, clases y métodos

Paquetes.subpaquete	Clases	Propiedades y métodos
java.lang	System.out	println() print() printf()
	System	arraycopy()
	Math	PI sqrt() pow() random()
	String	**valueOf()** **equals()** **equalsIgnoreCase()** regionMatches() **compareTo()** **compareToIgnoreCase()** **charAt()** **substring()** strip()

		stripLeading() stripTrailing() **length()** **indexOf()** **lastIndexOf()** isEmpty() contains() startsWith() endsWith() **toLowerCase()** **toUpperCase()** replace() split() toCharArray()
	Exception ArithmeticException	toString()
	Character	idDigit() isLetter() isLetterOrDigit() isLowerCase() isUpperCase() isSpaceChar() isWhiteSpace() toLowerCase() toUpperCase()
java.util		nextInt() nextDouble() nextLine() next() useLocale()
	Locale	US
	InputMismatchException	toString()
	Arrays	fill() toString() sort() binarySearch() copyOf() copyOfRange() clone() deepToString() …
java.time	LocalTime	now() toSecondOfDay() toNanoOfDay of()

java.math	BigDecimal	

Otras clases

Paquetes	Clases	Propiedades y métodos
java.lang	Clases envoltorio	
java.util	ArrayList	
	Calendar	getInstance() get()
javax.swing	JOptionPane	showInputDialog() showMessageDialog()

- ArrayList
- Colecciones
- Ficheros
- Bases de datos
- Interfaz gráfica (Tutorial Swing)
- Streams
- Sockets
- …

Bcrypt. Función de hash de contraseña

- Bcrypt Hash Generator
- https://github.com/patrickfav/bcrypt
- https://github.com/patrickfav/bytes-java

- Ejemplo: https://github.com/aguadodev/DAM1-Programacion-23-24/blob/main/src/varios/proyectoviajecompartido/CifradoBcrypt.java

Notas

System.console()

`System.console().readLine()`
o
`new Scanner(System.in).nextLine()`

`System.console()` es una forma de obtener una instancia de la consola del sistema. Sin embargo, hay una limitación importante: este método puede devolver `null` si la aplicación se ejecuta en un entorno que no tiene una consola asociada. Por ejemplo, si estás ejecutando tu aplicación desde un entorno de desarrollo integrado (IDE) como Eclipse o IntelliJ, `System.console()` puede devolver null.

varargs (argumentos variables)

- Varargs in Java | Baeldung

En Java, los puntos suspensivos (...) se utilizan para indicar que un método puede recibir un número variable de argumentos del mismo tipo. Estos valores se pueden procesar con un bucle for..each por ejemplo.

La sintaxis de puntos suspensivos para varargs (argumentos variables) fue introducida en Java 5 como parte de las características proporcionadas por el Proyecto de Mejora del Lenguaje Java (Java Language Enhancement, JEP 41). Java 5 se lanzó en septiembre de 2004.

Más contenidos a revisar:
Anexos y Buenas Prácticas de Programación
BD Mysql online gratis