Vamos a realizar una serie de pasos para comenzar con los fundamentos de Java

1. Cuenta en Github

<https://github.com>

1. Mi cuenta (Paco) se llama **serraguti**
2. Necesitamos instalar Git dentro de nuestro equipo

Interfaz de usuario gráfica, Texto, Aplicación

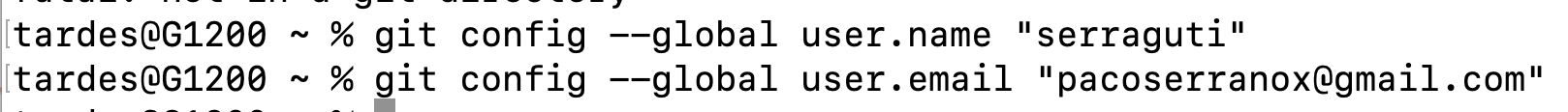
El contenido generado por IA puede ser incorrecto.

Debemos indicar nuestro usuario de GIT y nuestro CORREO

Entramos en Terminal (CMD) y escribimos estos comandos

**git config --global user.name USERGIT**

**git config --global user.email** [**MAILGIT@GMAIL.COM**](mailto:MAILGIT@GMAIL.COM)



El siguiente paso es utilizar el entorno de trabajo para Java, que será Visual Studio Code

Descargamos Visual Studio Code

Visual Studio Code trabaja mediante múltiples lenguajes.

Mediante extensiones, podemos trabajar con cualquier lenguaje dentro VS Code

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Una vez que tenemos configurado Java, simplemente vamos a sincronizar nuestro proyecto individual con Github

Necesitamos instalar otra extensión para trabajar con Github dentro de Vs Code

Esta extensión se validará con nuestra cuenta de Github y podrá sincronizar nuestros proyectos.

Texto

El contenido generado por IA puede ser incorrecto.

Automáticamente nos pedirá validarnos con nuestra cuenta de Git.

Una ventaja radica en que se sincronizarán nuestras extensiones (si queremos) con el resto de equipos.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

Dentro de Git tenemos dos opciones:

1. **Push**: Nos permite subir nuestros ficheros a Github.com
2. **Pull**:Nos permite descargar nuestros ficheros desde Github.com

Dentro de este proceso, debemos sincronizar los códigos para no tener problemas.

Vamos a subir nuestro nuevo código a un Repositorio llamado **borrarahora**

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Una vez publicado, cada vez que realicemos una modificación, siempre tendremos que hacer dos pasos:

1. Escribir un **Commit**. Un commit es una descripción de un estado de una actualización de un Branch/rama dentro de Git
2. Debemos pulsar en **Commit and Push**

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

Esto es respecto a realizar un Push y trabajar con un equipo (Aula)

El siguiente paso es realizar un Pull (Casa)

En casa no tenemos la carpeta **Prueba**, cerramos la carpeta en VS Code, cerramos VS Code y borramos la carpeta Prueba

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Trucos:

1. Si queremos crear otro repositorio con nuestra carpeta asociada de Github. Razones: Se ha roto, me he liado…

Ahora mismo tenemos asociado **borrarahora**, pero vamos a poner los cambios en otro repositorio distinto y nuevo llamado **borrarluego**

Necesitamos desligar nuestra carpeta de Git. Para ello, debemos eliminar la carpeta oculta **.git** del directorio del proyecto

Command + SHIFT + .

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

Cada vez que cambiemos algo entre CASA y el AULA, siempre debemos hacer primero **PULL**

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Eliminar los repositorios que tenemos

También podríamos cambiar su estado de visibilidad de private a public o viceversa

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

Vamos a trabajar en una carpeta llamada **fundamentosjavagetafe** y abrimos dicha carpeta en VS Code

Dentro de dicha carpeta, separaremos los conceptos mediante nuestras clases de ejemplo.

Utilizaremos una sintaxis progresiva para saber el elemento en el que estamos, por ejemplo: **Class01Descripcion.java**

Si vamos a visualizar tipos de datos primitivos: **Class02TiposPrimitivos.java**

Necesitamos un para de conceptos para empezar:

* El nombre de las clases en Java debe coincidir con el nombre del fichero.
* El nombre de las clases siempre irá en mayúscula cada palabra de la clase.
* Existen elementos llamados **snippets** dentro de VS Code que nos facilitan el trabajo, por ejemplo
  + **sout**: System.out.println
  + **psvm**: public static void main

Comenzaremos con tipos primitivos:

Son clases llamada **Wrapper**, es decir, clases que para ser construidas/instanciadas no es necesario poner el constructor de forma explicita.

Simplemente, con asignar un valor, el objeto es construido.

Una variable en código nos permite almacenar valores para utilizarlos dentro del código.

Dependiendo del valor que deseemos almacenar en la variable, tendremos que declarar dicha variable con un tipado.

Lo más importante no son los tipos, sino la conversión entre los tipos.

Por ejemplo, convertir un texto a un número

Convertir una fecha a un texto

Java tiene dos elementos:

1. Tipo primitivo (pequeño): Solo almacena datos
2. Clase primitiva (grande): Almacena datos y tiene más cositas

Dependiendo del tipo de dato, se tienen una serie de características para dicho tipo de dato.

Por ejemplo, quiero almacenar el número 14:

**int numero = 14;**

GRANDE:

Integer numero = 14;

Integer.MAXVALUE: Nos indica el máximo valor para almacenar

Cada tipo de dato tiene una precisión, no puedo superar dicha precisión

byte peke = 290;

Cuando hablamos de conversión de datos, para convertir entre tipos de dato, se utilizan los **GRANDES**

Los grandes sirven para convertir entre primitivos.

Para visualizar los diferentes tipos de conversión, creamos una clase llamada **Class01Primitivos.java**

Ejemplos de tipos primitivos/Wrapper:

* byte
* short
* int
* long
* double
* decimal
* char
* String
* boolean

**CONVERSION IMPLICITA**

La conversión es automática, no tenemos que hacer nada de nada

Se realiza cuando almacenamos un tipo mayor en un tipo menor (no su valor)

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

**CONVERSION ENTRE TIPOS PRIMITIVOS COMPATIBLES**

Este tipo de conversión implica convertir un tipo en otro tipo.

Se utiliza lo que se llama **Casting** y permite convertir entre tipos.

(Tipo)valor;

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

Tenemos una clase que es como un Primitivo, es decir, podemos crear el objeto sin necesidad de llamar a su constructor, simplemente igualando.

Si yo quiero almacenar textos:

**S**tring texto = “Hola mundo”;

**CONVERTIR CUALQUIER OBJETO A STRING**

Para convertir un objeto a string se utiliza el método **toString()** que tienen TODOS LOS OBJETOS DE Java

El método **toString()** vienen en la clase GRANDE del primitivo.

Texto

El contenido generado por IA puede ser incorrecto.

**CONVERTIR CUALQUIER STRING A PRIMITIVO**

Para convertir cualquier String a Primitivo se utiliza un método de la clase String llamado **valueOf()** que nos permite convertir elementos a String

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

**CONVERTIR UN STRING A PRIMITIVO**

Para convertir un string a primitivo necesitamos la clase Grande del primitivo.

Todas las clases primitivas tienen un método llamado **parse()** que nos permite convertir cualquier tipo de dato String a su primitivo.

Imagen que contiene Texto

El contenido generado por IA puede ser incorrecto.

Dentro de nuestros códigos tenemos tres tipos de errores:

1. Errores de compilación: Son errores de código y que los podemos visualizar de forma más o menos sencilla. (Nos falta un punto y coma)
2. Errores en ejecución: El programa compila, pero se detiene en la ejecución porque tengo algún error. (Problemas al convertir)
3. Errores lógicos: Todo funciona, no nos da ningún error y el programa no hace lo que queremos.

Error de ejecución:

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

Tenemos objetos especializados como acabamos de visualizar, objetos para almacenar textos, números o decimales.

Cada tipo tiene una funcionalidad, por ejemplo, si declaro un tipo int, puedo sumar, restar o dividir.

Si declaro un tipo String, puedo contar el número de caracteres que contiene.

Tenemos una clase que nos permite “pedir” información al usuario mediante teclado.

Al estar en consola, se utiliza una clase llamada **Scanner** que nos permite pedir información por teclado.

Dentro de Java todo está en librerías. Las librerías están especializadas en sus componentes.

Por ejemplo, existen librerías para ficheros o librerías para acceso a datos o librerías para entornos Web.

Existe una librería llamada **java.util** que nos permite trabajar con herramientas del entorno de Consola

Para poder crear un Scanner necesitamos dos elementos:

1. Utilizar la librería java.util
   1. java.util.Scanner

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

* 1. import java.util.Scanner

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Imagen que contiene Texto

El contenido generado por IA puede ser incorrecto.

1. Instanciar el objeto Scanner



Para pedir datos al usuario tenemos un método llamado **nextLine()** que nos permite recuperar lo que el usuario ha escrito.

Para concatenar (unir dos textos) se utiliza el símbolo **+** en Java

Texto

El contenido generado por IA puede ser incorrecto.

Vamos a realizar un ejemplo muy sencillo en el que pediremos dos números al usuario por teclado y mostraremos la suma de los dos números.

Creamos un nuevo programa llamado **Class03SumarNumeros.java**

Texto

El contenido generado por IA puede ser incorrecto.

**import java.util.Scanner;**

**public class Class03SumarNumeros {**

**public static *void* main(*String*[] *args*) {**

***Scanner* teclado = new Scanner(System.in);**

**System.out.println("Introduzca número 1");**

**//Almacenamos el primer numero**

**//El numero debemos almacenarlo como entero**

**//Voy a capturar el dato como String**

***String* dato = teclado.nextLine();**

**//Convertimos el dato a int**

***int* numero1 = Integer.parseInt(dato);**

**System.out.println("Introduzca número 2");**

**//Volvemos a utilizar la variable dato**

**dato = teclado.nextLine();**

**//Convertimos el dato a int**

***int* numero2 = Integer.parseInt(dato);**

**//Realizamos la suma**

***int* suma = numero1 + numero2;**

**//Dibujamos la suma**

**System.out.println("La suma es " + suma);**

**System.out.println("Fin de programa");**

**}**

**}**

**CONDICIONALES**

Los condicionales en cualquier lenguaje nos permiten que nuestro código sea dinámico y que pueda realizar diferentes acciones dependiendo de lo que vayamos preguntando en las condiciones.

Operadores de comparación

== Igual

> Mayor

>= Mayor o igual

< Menor

<= Menor o igual

!= Distinto

Tenemos varias posibilidades en el momento de preguntar, pero la sintaxis es la misma.

**Sintaxis simple de un Condicional (IF)**

if (condición == true)

{

//Solamente entra si es TRUE

}

**Sintaxis Si no en condición**

if (condición == true)

{

//Acciones para el true

}else {

//Acciones para el false

}

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

**Sintaxis con múltiples preguntas**

Se utiliza cuando necesitamos preguntar por más elementos en una condición

if (condiciónA == true)

{

//Acciones para condición A

}else if (condicionB == true) {

//Acciones para condición B

}else {

//Ninguna condición se cumple

}

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Vamos a realizar un programa para preguntar si un número es positivo, negativo o cero.

Creamos un nuevo programa llamado **Class04PositivoNegativo.java**

**Texto

El contenido generado por IA puede ser incorrecto.**

**import java.util.Scanner;**

**public class Class04PositivoNegativo {**

**public static *void* main(*String*[] *args*) {**

**//PODEMOS COMBINAR CON SCANNER**

***Scanner* teclado = new Scanner(System.in);**

**System.out.println("Introduzca un número");**

***String* dato = teclado.nextLine();**

**//Convertimos dato a int**

***int* numero = Integer.parseInt(dato);**

**if (numero > 0){**

**System.out.println("El número es POSITIVO");**

**}else if (numero == 0) {**

**System.out.println("El número es CERO");**

**}else {**

**System.out.println("El número es NEGATIVO");**

**}**

**System.out.println("Fin de programa");**

**}**

**}**

Vamos a realizar un programa en el que pediremos un número del 1 al 4 al usuario.

Debemos indicar la estación dependiendo del número

1.- Primavera

2.- Verano

3.- Invierno

4.- Otoño

Otra opción, escribimos un mensaje diciendo Incorrecto

Llamamos al programa **Class05Estaciones.java**

Imagen que contiene Diagrama

El contenido generado por IA puede ser incorrecto.

**import java.util.Scanner;**

**public class Class05Estaciones {**

**public static *void* main(*String*[] *args*) {**

***Scanner* teclado = new Scanner(System.in);**

**System.out.println("Introduzca un número (1-4)");**

***String* dato = teclado.nextLine();**

***int* numero = Integer.parseInt(dato);**

**if (numero == 1){**

**System.out.println("Primavera");**

**}else if(numero == 2){**

**System.out.println("Verano");**

**}else if (numero == 3){**

**System.out.println("Invierno");**

**}else if(numero == 4){**

**System.out.println("Otoño");**

**}else {**

**System.out.println("Opción incorrecta");**

**}**

**}**

**}**

Operadores relacionales

Los operadores relacionales nos permiten preguntar por más de un elemento dentro de una misma pregunta

&& And: Todas las condiciones deben cumplirse

|| Or: Solamente una condición debe cumplirse

! Not: Negación de una condición. Mejor utilizar siempre un Operador

Quiero preguntar si un número es diferente a 1

if (**!** (numero == 1))

if (numero !=1)

Vamos a realizar un programa para calcular el mayor, el menor y el intermedio de tres números.

Creamos un nuevo programa llamado **Class06MayorTresNumeros.java**

Tabla

El contenido generado por IA puede ser incorrecto.

**import java.util.Scanner;**

**public class Class06MayorTresNumeros {**

**public static *void* main(*String*[] *args*) {**

***Scanner* teclado = new Scanner(System.in);**

**System.out.println("Introduzca número 1");**

***String* dato = teclado.nextLine();**

***int* num1 = Integer.parseInt(dato);**

**System.out.println("Introduzca número 2");**

**dato = teclado.nextLine();**

***int* num2 = Integer.parseInt(dato);**

**System.out.println("Introduzca número 3");**

**dato = teclado.nextLine();**

***int* num3 = Integer.parseInt(dato);**

***int* mayor = 0;**

**//PREGUNTAMOS POR LAS CONDICIONES**

**//22 , 88, 244**

**if (num1 > num2 && num1 > num3){**

**mayor = num1;**

**}else if (num2 > num1 && num2 > num3){**

**mayor = num2;**

**}else {**

**mayor = num3;**

**}**

**System.out.println("El número mayor es " + mayor);**

**}**

**}**

Solución Versión 2

**import java.util.Scanner;**

**public class Class06MayorTresNumeros {**

**public static *void* main(*String*[] *args*) {**

***Scanner* teclado = new Scanner(System.in);**

**System.out.println("Introduzca número 1");**

***String* dato = teclado.nextLine();**

***int* num1 = Integer.parseInt(dato);**

**System.out.println("Introduzca número 2");**

**dato = teclado.nextLine();**

***int* num2 = Integer.parseInt(dato);**

**System.out.println("Introduzca número 3");**

**dato = teclado.nextLine();**

***int* num3 = Integer.parseInt(dato);**

***int* mayor = 0;**

***int* menor = 0;**

***int* intermedio = 0;**

**//PREGUNTAMOS POR LAS CONDICIONES**

**//22 , 88, 244**

**if (num1 >= num2 && num1 >= num3){**

**mayor = num1;**

**}else if (num2 >= num1 && num2 >= num3){**

**mayor = num2;**

**}else {**

**mayor = num3;**

**}**

**if (num1 <= num2 && num1 <= num3){**

**menor = num1;**

**}else if (num2 <= num1 && num2 <= num3){**

**menor = num2;**

**}else{**

**menor = num3;**

**}**

**//sumamos todos los numeros**

***int* suma = num1 + num2 + num3;**

**intermedio = suma - mayor - menor;**

**System.out.println("El número menor es " + menor);**

**System.out.println("El número mayor es " + mayor);**

**System.out.println("El número intermedio es " + intermedio);**

**}**

**}**

**CALCULAR DIA DE NACIMIENTO DE LA SEMANA**

**Class07DiaNacimiento.java**

Pedir una fecha al usuario para calcular el día de la semana que nació. Tenemos que tener la tabla de días de la semana para la correspondencia comenzando en sábado:

|  |  |
| --- | --- |
| **DIA** | **NÚMERO** |
| **Sábado** | 0 |
| **Domingo** | 1 |
| **Lunes** | 2 |
| **Martes** | 3 |
| **Miércoles** | 4 |
| **Jueves** | 5 |
| **Viernes** | 6 |

Debemos pedir el **día**, el número de **mes** y el **año** que el usuario haya nacido.

A partir de esto datos hay que calcular lo siguiente para averiguar el día de la semana de nacimiento:

Ejemplo 🡪 15/06/1997

Hay que tener en cuenta el mes para realizar el cálculo, si el mes es Enero, el Mes será 13 y restaremos uno al año. Si el Mes es Febrero, el Mes será 14 y restaremos uno al año.

Para poder calcular las el número final de la semana debemos seguir los siguientes pasos:

1. Multiplicar el Mes más 1 por 3 y dividirlo entre 5

((6 + 1) \* 3) / 5 🡪 4

1. Dividir el año entre 4

1997 / 4 🡪 499

1. Dividir el año entre 100

1997 / 100 🡪 19

1. Dividir el año entre 400

1997 / 400 🡪 4

1. Sumar el dia, el doble del mes, el año, el resultado de la operación 1, el resultado de la operación 2, menos el resultado de la operación 3 más la operación 4 más 2.

15 + (6 \* 2) + 1997 + 4 + 499 - 19 + 4 + 2 🡪 2514

1. Dividir el resultado anterior entre 7.

2514 / 7 🡪 359

1. Restar el número del paso 5 con el número del paso 6 por 7.

2514 – (359 \* 7) 🡪 1

1. Miramos la tabla y vemos que el número 1 corresponde a **Domingo**

SOLUCION

**import java.util.Scanner;**

**public class Class07DiaNacimiento {**

**public static *void* main(*String*[] *args*) {**

***Scanner* teclado = new Scanner(System.in);**

**System.out.println("Introduzca día");**

***String* dato = teclado.nextLine();**

***int* dia = Integer.parseInt(dato);**

**System.out.println("Introduzca mes");**

**dato = teclado.nextLine();**

***int* mes = Integer.parseInt(dato);**

**System.out.println("Introduzca año");**

**dato = teclado.nextLine();**

***int* anyo = Integer.parseInt(dato);**

**if (mes == 1){**

**mes = 13;**

**anyo -= 1;**

**}else if (mes == 2){**

**mes = 14;**

**anyo--;**

**}**

**//1. Multiplicar el Mes más 1 por 3 y dividirlo entre 5**

***int* op1 = ((mes + 1) \* 3) / 5;**

***int* op2 = anyo / 4;**

***int* op3 = anyo / 100;**

***int* op4 = anyo / 400;**

**//Comentar en bloque. Seleccionamos lo que necesitemos**

**// Command + K + C**

**//Descomentar codigo: Command + K + U**

**// 5. Sumar el dia, el doble del mes, el año**

**// , el resultado de la operación 1, el resultado de la operación 2**

**// , menos el resultado de la operación 3 más la operación 4 más 2.**

***int* op5 = dia + (mes \* 2) + anyo + op1 + op2 - op3 + op4 + 2;**

**//6. Dividir el resultado anterior entre 7.**

***int* op6 = op5 / 7;**

**// 7. Restar el número del paso 5 con el número del paso 6 por 7.**

***int* resultado = op5 - (op6 \* 7);**

***String* diaSemana = "";**

**if (resultado == 0){**

**diaSemana = "SABADO";**

**}else if (resultado == 1){**

**diaSemana = "DOMINGO";**

**}else if (resultado == 2){**

**diaSemana = "LUNES";**

**}else if (resultado == 3){**

**diaSemana = "MARTES";**

**}else if (resultado == 4){**

**diaSemana = "MIERCOLES";**

**}else if (resultado == 5){**

**diaSemana = "JUEVES";**

**}else if (resultado == 6){**

**diaSemana = "VIERNES";**

**}else {**

**diaSemana = "Hemos pinchado...";**

**}**

**System.out.println("El día de la semana es " + diaSemana);**

**}**

**}**

Una vez que tenemos el programa, debemos aprender a Depurar/Debug del código.

La depuración es algo esencial a medida que vayamos creciendo en desarrollo.

Para depurar se utiliza la tecla F5 o Debug Java

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

Nuestro código se detendrá dónde nosotros deseemos, es decir, debemos incluir Puntos de interrupción para “saber” analizar qué sucede en el código en un momento dado.

Los puntos de interrupción se incluyen a la izquierda de nuestro código (mas allá de los números)

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

En el momento que ejecutemos como **Debug**, se detendrá en el punto de interrupción

Tendremos dos opciones para ir paso a paso

* F10: Cuando tenemos solamente un código que analizar
* F11: Cuando tenemos llamadas en dicho código y deseamos ir a visualizar esas llamadas.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

CALCULAR SALARIO

**Class08CalcularSalario.java**

* Realizar una aplicación que obtenga el salario neto de los empleados de una empresa teniendo en cuenta el número de horas trabajadas, el importe a pagar por hora y el lugar al que ha sido destinado el trabajador (establecido en Km).
* Cada hora extra se pagará 2€ más que las normales, (la hora extra comienza a computarse a partir de la 36 hora semanal trabajada)
* Sobre el salario base anterior, si el destino del empleado está:
  + - Entre 101 Km y 900Km: La dieta será “NACIONAL”
    - Por encima de 900Km: Dieta “INTERNACIONAL”
    - Por debajo de 101 Km Dieta “PROVINCIAL”
* Sobre el precio final obtenido del salario:
  + - Cantidades menores o iguales a 250 € no soportan retención 0%
    - Cantidades por encima de 250 € y menores o iguales a 500 € la retención es del 20%
    - Cantidades por encima de 500 € la retención es del 50%
* Aplicaremos el IVA (16%) al salario bruto y se lo restaremos para conseguir el salario neto del trabajador.

Horas trabajadas: 40

Precio Hora: 20

Horas extras: 4

PrecioBase: 36 \* 20

PrecioExtra: 4 \* 22

* **Al final debemos mostrar el siguiente informe:**

INTRODUZCA HORAS SEMANALES

51

INTRODUZCA IMPORTE HORA:

24

INTRODUZCA KILOMETROS:

269

---------------------------------------------------

HORAS TRABAJADAS: 51

HORAS EXTRAS: 15

IMPORTE DE LA HORA: 24

DISTANCIA EN KM: 269

DESTINO: NACIONAL

RETENCION: 50%

SALARIO BASE: 864.00

SALARIO HORAS EXTRA: 540.00

SALARIO BRUTO: 1404.00

IVA (16%): 224.64

---------------------------------------------------

SALARIO TOTAL: 1179.36

---------------------------------------------------

FIN DE PROGRAMA

**SOLUCION**

**import java.util.Scanner;**

**public class Class08CalcularSalario {**

**public static *void* main(*String*[] *args*) {**

***Scanner* teclado = new Scanner(System.in);**

***int* horas = 0;**

***int* kilometros = 0;**

***int* horasExtra = 0;**

***int* importeHora = 0;**

***int* salario = 0;**

***int* salarioExtra = 0;**

***String* destino = "";**

***String* retencion = "";**

***float* iva = 0;**

**System.out.println("Introduzca horas semanales");**

***String* dato = teclado.nextLine();**

**horas = Integer.parseInt(dato);**

**System.out.println("Introduzca precio/hora");**

**dato = teclado.nextLine();**

**importeHora = Integer.parseInt(dato);**

**System.out.println("Kilometros recorridos");**

**dato = teclado.nextLine();**

**kilometros = Integer.parseInt(dato);**

**if (horas > 36){**

**//Tenemos horas extra**

**horasExtra = horas - 36;**

**salario = 36 \* importeHora;**

**salarioExtra = horasExtra \* (importeHora + 2);**

**salario = salario + salarioExtra;**

**}else {**

**salario = horas \* importeHora;**

**}**

**//comprobamos el destino segun km**

**if (kilometros > 101 && kilometros < 900){**

**destino = "NACIONAL";**

**}else if (kilometros > 900){**

**destino = "INTERNACIONAL";**

**}else {**

**destino = "PROVINCIAL";**

**}**

**//comprobamos el salario para la retencion**

**if (salario <= 250){**

**retencion = "0%";**

**}else if (salario > 500){**

**retencion = "50%";**

**}else {**

**retencion = "20%";**

**}**

**iva = (*float*)(salario \* 0.16);**

**System.out.println("Horas trabajadas: " + horas);**

**System.out.println("Horas extras: " + horasExtra);**

**System.out.println("Importe hora: " + importeHora);**

**System.out.println("Kilometros: " + kilometros);**

**System.out.println("Destino: " + destino);**

**System.out.println("Retención: " + retencion);**

**System.out.println("Salario base: " + salario);**

**System.out.println("Salario extra: " + salarioExtra);**

**System.out.println("Iva (16%): " + iva);**

**System.out.println("-------------------------");**

**System.out.println("Salario final: " + (salario - iva));**

**}**

**}**

**BUCLES EN JAVA**

Un bucle es una secuencia repetitiva en código.

Un bucle es un bucle, no importa el lenguaje.

Las secuencias pueden ser de dos tipos:

* Bucle Contadores: Son bucles que tienen un inicio y un final y están administrados. (for)

Texto

El contenido generado por IA puede ser incorrecto.

Imagen que contiene Texto

El contenido generado por IA puede ser incorrecto.

Imagen que contiene Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

* Bucles condicionales: Son bucles que dependen de una condición, no sabemos cuando van a finalizar, dependerá de la condición. Este tipo de bucles son los más peligrosos porque podríamos tener un bucle infinito. (while)

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

Imagen que contiene Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

Creamos una clase llamada **Class09Bucles.java**

**import java.util.Scanner;**

**public class Class09Bucles {**

**public static *void* main(*String*[] *args*) {**

**//bucles contador (for)**

**//Debemos indicar un inicio, un final (while), un incremento**

**//La variable contador se declara dentro del bucle y**

**//solamente se puede utilizar dentro de dicho bucle**

**// for (int i = 1; i <= 5; i++){**

**// System.out.println("For: " + i);**

**// }**

**// //Podemos tener un incremento distinto**

**// //de 3 en 3**

**// for (int i = 1; i <= 10; i+=3){**

**// System.out.println("Tres en tres " + i);**

**// }**

**// //Podemos hacer que el bucle sea inverso**

**// for (int i = 7; i > 0; i--){**

**// System.out.println("Inverso " + i);**

**// }**

**// //Bucles de condición**

**// //Necesitan una condición para salir**

**// //bucles infinitos**

**// //Necesitamos una variable de salida del bucle**

**// int contador = 1;**

**// while (contador <= 5) {**

**// System.out.println("While: " + contador);**

**// //Debemos hacer "algo" para que la variable**

**// //del bucle cambie y llegue a la condición de salida**

**// contador++;**

**// }**

**//Necesito mostrar los 20 primeros números pares**

**// for (int i = 1; i <= 20; i++){**

**// //Preguntamos si un numero es par**

**// //Se utiliza el operador del resto: %**

**// if (i % 2 == 0){**

**// System.out.println("Par: " + i);**

**// }**

**// }**

**//Vamos a pedir al usuario textos hasta que escriba stop**

***Scanner* teclado = new Scanner(System.in);**

***String* dato = "";**

**//Java en sus String no compara con el operador == ni !=**

**//Estoy comparando objetos y con String debo comparar el valor**

**//del objeto**

**//se utiliza el método equals() de la clase String**

**// if (dato.equals("")){**

**// System.out.println("Texto vacio!!!");**

**// }else{**

**// System.out.println("Contenido");**

**// }**

**while (dato.equals("stop") == false){**

**System.out.println("Dime algo bonito (stop para salir)");**

**dato = teclado.nextLine();**

**}**

**}**

**}**

Creamos una nueva clase llamada **Class10TablaMultiplicar.java**

Pediremos un número al usuario y mostraremos la tabla de multiplicar de dicho número.

5\*1 = 5

5\*2 = 10

…

5\*10 = 50

**SOLUCION**

**import java.util.Scanner;**

**public class Class10TablaMultiplicar {**

**public static *void* main(*String*[] *args*) {**

***Scanner* teclado = new Scanner(System.in);**

**System.out.println("Introduzca un número");**

***String* dato = teclado.nextLine();**

***int* numero = Integer.parseInt(dato);**

**for (*int* i = 1; i <= 10; i++){**

***int* operacion = numero \* i;**

**//5 \* 1 = 5**

**System.out.println(numero + " \* " + i + " = " + operacion);**

**}**

**}**

**}**

Vamos a intentar demostrar la Conjetura de Collatz.

Esto es una teoría matemática que dice lo siguiente:

Todo número positivo llegará siempre a 1 siguiendo una serie de condiciones:

* Si el número es par, se divide entre 2
* Si el número es impar, se multiplica por 3 y se suma 1

6, 3, 10, 5, 16, 8, 4, 2, 1

**Class11Collatz.java**

**import java.util.Scanner;**

**public class Class11Collatz {**

**public static *void* main(*String*[] *args*) {**

***Scanner* teclado = new Scanner(System.in);**

**System.out.println("Número para Collatz: ");**

***String* dato = teclado.nextLine();**

***int* numero = Integer.parseInt(dato);**

**while (numero != 1){**

**if (numero % 2 == 0){**

**numero = numero / 2;**

**}else{**

**numero = numero \* 3 + 1;**

**}**

**System.out.println(numero);**

**}**

**}**

**}**

**CLASES JAVA**

Todo en Java son clases (Excepto los primitivos).

Todas las clases están compuestas por métodos, dichos métodos pueden ser acciones o características de un objeto.

Por ejemplo, pongamos el ejemplo de un Coche:

Coche car;

car.arrancar(); //ACCION

car.girar(); //ACCION

car.color; //ATRIBUTO

Los métodos en las clases también pueden recibir parámetros, por ejemplo:

car.frenar(MUCHO);

car.frenar(POCO);

Dentro de las clases y del concepto de POO existen varias características.

Una de las características se llama Polimorfismo, que indica que un método puede tener múltiples formas.

Por ejemplo, no tenemos tres pedales de coche…

car.frenarMucho();

car.frenarPoco();

car.frenarNormal();

Aplicado a Polimorfismo, tenemos un solo método (frenar) y varias acciones sobre él

car.frenar(MUCHO);

car.frenar(POCO);

car.frenar();

Vamos a visualizar los métodos y características de la clase String.

Un String es una clase que contiene un conjunto de caracteres.

Tenemos multitud de métodos, para convertir a mayúsculas, para buscar caracteres, para eliminar caracteres…

Todos los conjuntos dentro de Java comienzan en CERO.

String texto = “Hola Cara Cola”;

* **equals():** Comparar un String con otro String (valor)
* **equalsIgnoreCase():** Comparar un String con otro String sin importar mayúsculas/minúsculas
* **isEmpty():** Pregunta si el texto es vacío.
* **toLowerCase():** Convierte a minúsculas un texto
* **toUpperCase():** Converte a mayúsculas un texto
* **contains(caracter):** Indica si un carácter está dentro de un String
* **length():** La longitud del texto del String
* **charAt(índice):** Devuelve la letra que se corresponde con dicho índice
* **indexOf(caracter):** Busca un carácter dentro del String y devuelve la posición donde se encuentra dicho carácter (la primera coincidencia).

Si no encuentra el carácter, devuelve -1.

* **indexOf(carácter, posición):** Busca un carácter dentro del String y devuelve la posición donde se encuentra dicho carácter a partir de la posición que estemos indicando.

Si no encuentra el carácter, devuelve -1.

* **lastIndexOf(caracter**): Realiza lo mismo que indexOf, pero comienza a buscar al final del texto, es decir, de derecha a izquierda.

Las posiciones de los caracteres no cambian busquemos por la izquierda o la derecha.

Si no encuentra el carácter devuelve -1.

* **endsWith(texto):** Devuelve true/false indicando si el String finaliza con el texto
* **startsWith(texto):** Devuelve true/false indicando si el String comienza con el texto indicado.
* **replace(old char, new char):** Reemplaza el antiguo texto por el nuevo
* **substring(inicio):** Recupera una subcadena del String a partir del inicio que indiquemos.
* **substring(inicio, final):** Recupera una subcadena del String a partir del inicio que indiquemos hasta el final.
* **trim():** Elimina los espacios al inicio y al final del texto.
* **String.valueOf(integer):** Sirve para convertir objetos primitivos a String

Para probar esto, creamos una nueva clase llamada **Class12String.java**

Texto

El contenido generado por IA puede ser incorrecto.

**public class Class12String {**

**public static *void* main(*String*[] *args*) {**

***String* texto = "Desarrollo con Java";**

**System.out.println("Longitud texto: " + texto.length());**

**System.out.println("toLower() " + texto.toLowerCase());**

**System.out.println("toUpper() " + texto.toUpperCase());**

**//Buscamos la letra a de diversas formas**

**System.out.println("indexOf('a'): " + texto.indexOf('a')); //3**

**//la sobrecarga busca a partir de un indice superior**

**System.out.println("indexOf('a', 4): "**

**+ texto.indexOf('a', 4));**

**System.out.println("lastIndexOf('a'): " + texto.lastIndexOf('a'));**

**//la sobrecarga busca a partir de un indice inferior**

**System.out.println("lastIndexOf('a', 17):"**

**+ texto.lastIndexOf('a', 17));**

**//Nos quedamos con una parte de la cadena**

**System.out.println("substring(5): "**

**+ texto.substring(5));**

**//tenemos una sobrecarga indicando un final**

**System.out.println("substring(2,4): "**

**+ texto.substring(2,4));**

**System.out.println("startsWith('D'): "**

**+ texto.startsWith("D"));**

**System.out.println("endsWith('a'): "**

**+ texto.endsWith("a"));**

**//Recuperar un caracter**

**System.out.println("charAt(5): "**

**+ texto.charAt(5));**

**//Aplicar un método a un objeto, no implica**

**//que el objeto sea modificado.**

**//Los métodos (algunos) devuelven un valor del objeto.**

**//convertimos el texto a mayusculas**

**texto.toUpperCase();**

**//Si deseamos modificar el objeto texto, debemos igualarlo**

**texto = texto.toUpperCase();**

**System.out.println(texto);**

**}**

**}**

Texto

El contenido generado por IA puede ser incorrecto.

Tabla

El contenido generado por IA puede ser incorrecto.

**SOLUCION**

**import java.util.Scanner;**

**public class Class13CalcularDni {**

**public static *void* main(*String*[] *args*) {**

***Scanner* teclado = new Scanner(System.in);**

**System.out.println("Seleccione una opción");**

**System.out.println("1.- Comprobar NIF");**

**System.out.println("2.- Averiguar letra NIF");**

***String* dato = teclado.nextLine();**

***int* opcion = Integer.parseInt(dato);**

***String* letrasDni = "TRWAGMYFPDXBNJZSQVHLCKET";**

**if (opcion == 1){**

**System.out.println("Introduzca su NIF (12345678X)");**

***String* dni = teclado.nextLine();**

**//PARA EL CALCULO, NECESITAMOS DIVIDIR EL NUMERO DE LA LETRA**

**//COMENZAMOS RECUPERANDO LA LETRA**

***char* letraDni = dni.charAt(dni.length() - 1);**

***String* temp = dni.substring(0, dni.length() - 1);**

***int* numeroDni = Integer.parseInt(temp);**

***int* resultado = (numeroDni - (numeroDni / 23) \* 23);**

***char* letra = letrasDni.charAt(resultado);**

**if (letraDni == letra){**

**System.out.println("Su DNI es correcto");**

**}else{**

**System.out.println("Su DNI es incorrecto, su letra es " + letra);**

**}**

**} else {**

**System.out.println("Introduzca número DNI");**

**dato = teclado.nextLine();**

***int* numeroDni = Integer.parseInt(dato);**

***int* resultado = (numeroDni - (numeroDni / 23) \* 23);**

***char* letra = letrasDni.charAt(resultado);**

**System.out.println("Su letra del DNI es " + letra);**

**}**

**}**

**}**

Necesito un programa para validar un Email. **Class14ValidarEmail.java**

Debemos indicar cada error del Email.

No podemos utilizar bucles, solamente IF.

Vamos a comprobar una serie de elementos:

* Exista una @
* @ no esté ni al inicio ni al final
* Solamente una @
* Exista un punto
* Exista un punto después de la @
* Dominio de 2 a 3 caracteres

If (email.contains(“@”) == false)

Mensaje de No existe @

Else if …..

**import java.util.Scanner;**

**public class Class14ValidarEmail {**

**public static *void* main(*String*[] *args*) {**

***Scanner* teclado = new Scanner(System.in);**

**System.out.println("Introduzca un Email");**

***String* email = teclado.nextLine();**

**// • Exista una @**

**// • @ no esté ni al inicio ni al final**

**// • Solamente una @**

**// • Exista un punto**

**// • Exista un punto después de la @**

**// • Dominio de 2 a 3 caracteres**

**if (email.contains("@") == false){**

**System.out.println("No existe @");**

**}else if (email.indexOf("@") == 0**

**|| email.endsWith("@")){**

**System.out.println("@ al inicio o al final");**

**}else if (email.indexOf("@") != email.lastIndexOf("@")){**

**System.out.println("Existe más de una @");**

**}else if (email.indexOf(".") == -1){**

**System.out.println("No existe punto");**

**}else if (email.lastIndexOf(".") < email.indexOf(("@"))){**

**System.out.println("Necesitamos punto después de @");**

**}else{**

***int* ultimoPunto = email.lastIndexOf(".");**

***String* dominio = email.substring(ultimoPunto + 1);**

**if (dominio.length() >= 2 && dominio.length() <= 3){**

**System.out.println("El email es CORRECTO!!!");**

**}else{**

**System.out.println("El dominio debe ser de 2 a 3 caracteres");**

**}**

**}**

**}**

**}**

Vamos a realizar un programa para sumar los caracteres de un texto.

Pediremos al usuario un texto numérico y mostraremos la suma de cada carácter de dicho texto.

1234 = 10

Creamos una nueva clase llamada **Class15SumarTextoNumeros.java**

**import java.util.Scanner;**

**public class Class15SumarTextoNumeros {**

**public static *void* main(*String*[] *args*) {**

***Scanner* teclado = new Scanner(System.in);**

**System.out.println("Introduzca una serie de números");**

***String* textoNumeros = teclado.nextLine();**

**//1234**

***int* suma = 0;**

**for (*int* i = 0; i < textoNumeros.length(); i++){**

**//CAPTURAMOS CADA LETRA DEL TEXTO**

***char* letra = textoNumeros.charAt(i);**

**//CONVERTIMOS EL char A int**

**//'1' --> 49 char**

**//"1" --> 1 String**

***String* aux = String.valueOf(letra);**

***int* numero = Integer.parseInt(aux);**

**suma += numero;**

**}**

**System.out.println("La suma de " + textoNumeros + " es: " + suma);**

**}**

**}**

Necesitamos un programa para mostrar si un número es narcisista o no lo es.

Un número narcisista es aquel que la suma de la potencia de su longitud es el mismo resultado que el número introducido.

Este número es narcisista: 153

Longitud: 3 (potencia a 3)

* 1x1x1 = 1
* 5x5x5 = 125
* 3x3x3 = 27

125 + 27 + 1 = 153!!!!

¿Cómo hacemos una potencia en Java? Pista: Paco ya lo ha dicho: Math.???????

**Class16Narcisista.java**

**import java.util.Scanner;**

**public class Class16Narcisista {**

**public static *void* main(*String*[] *args*) {**

***Scanner* teclado = new Scanner(System.in);**

**System.out.println("Introduzca un número narcisista");**

***String* textoNumero = teclado.nextLine();**

***int* longitud = textoNumero.length();**

***int* suma = 0;**

**for (*int* i = 0; i < textoNumero.length(); i++){**

***char* letra = textoNumero.charAt(i);**

***String* aux = String.valueOf(letra);**

***int* numero = Integer.parseInt(aux);**

***int* operacion = (*int*) Math.pow(numero, longitud);**

**suma += operacion;**

**}**

**if (suma == Integer.parseInt(textoNumero)){**

**System.out.println("El número " + textoNumero + " es narcisista");**

**}else{**

**System.out.println("El número NO es narcisista");**

**}**

**}**

**}**

**VALIDAR ISBN**

* Debemos comprobar que el número introducido tiene 10 caracteres.

EJEMPLO DE NUMERO ISBN QUE ESTÁ BIEN:

8441513929

1.- Se descompone la cadena y se multiplica cada número por la posición que ocupa en la cadena:

8 \* 1

4 \* 2

4 \* 3

1 \* 4

5 \* 5

.

.

.

9 \* 10

2.- La suma de todas estas multiplicaciones se divide entre 11, y si el resto es cero, el número ISBN es correcto.

INTRODUZCA EL NUMERO ISBN A VALIDAR (10 CIFRAS)

1234567891

LA LONGITUD DE CARACTERES ES 10

COMPROBANDO ISBN 1234567891

NUMERO ISBN 1234567891 NO CORRECTO

FIN DE PROGRAMA

**import java.util.Scanner;**

**public class Class17ValidarIsbn {**

**public static *void* main(*String*[] *args*) {**

***Scanner* teclado = new Scanner(System.in);**

**System.out.println("Introduzca un número ISBN");**

***String* isbn = teclado.nextLine();**

**if (isbn.length() == 10){**

***int* suma = 0;**

**for (*int* i = 0; i < isbn.length(); i++){**

***char* letra = isbn.charAt(i);**

***String* aux = String.valueOf(letra);**

***int* numero = Integer.parseInt(aux);**

***int* operacion = numero \* (i + 1);**

**suma += operacion;**

**}**

**if (suma%11 == 0){**

**System.out.println("El número ISBN es correcto!!!");**

**}else{**

**System.out.println("El número ISBN es incorrecto");**

**}**

**}else{**

**System.out.println("El número ISBN debe ser de 10 caracteres");**

**}**

**}**

**}**

**CLASE CHARACTER**

La clase Character es la clase que se corresponde que el tipo primitivo **char**.

Se utiliza para validar los caracteres de un texto.

Sobre todo, se utiliza para averiguar el contenido de un char.

La mayoría de los métodos de la clase Character son **static**.

Un método **static** es un método que no actúa sobre el objeto, sino que actúa sobre la clase y sus parámetros, es decir, es una herramienta de la clase.

Por ejemplo, tenemos un método **static** llamado **isLetter()** que indica si un carácter es una letra o no es una letra.

char letra = ‘a’;

if (**letra**.isLetter() == true) {

esto esta mal

}

if (Character.isLetter(**letra**) == true) {

}

Character.toLowerCase(letra)

String texto = “Hola martes”;

texto.**toLowerCase**();

Vamos a realizar un ejemplo llamado **Class18AnalizarTexto.java**

Debemos utilizar la clase Character. Pediremos cualquier texto al usuario y debemos indicar los siguientes datos sobre el texto escrito:

Texto: Hoy es martes 10 de junio de 25

Número de letras: 9

Número de dígitos: 4

Número de vocales: 6

Número de consonantes: 3

Mayúsculas: 1

<https://docs.oracle.com/javase/8/docs/api/java/lang/Character.html>

Imagen que contiene Texto

El contenido generado por IA puede ser incorrecto.

**import java.util.Scanner;**

**public class Class18AnalizarTexto {**

**public static *void* main(*String*[] *args*) {**

***Scanner* teclado = new Scanner(System.in);**

**System.out.println("Introduzca un texto");**

***String* texto = teclado.nextLine();**

***int* numLetras = 0;**

***int* numDigitos = 0;**

***int* vocales = 0;**

***int* consonantes = 0;**

***int* mayusculas = 0;**

**for (*int* i = 0; i < texto.length(); i++){**

***char* letra = texto.charAt(i);**

**if (Character.isDigit(letra) == true){**

**numDigitos += 1;**

**}else if (Character.isLetter(letra) == true){**

**numLetras += 1;**

**if (Character.isUpperCase(letra) == true){**

**mayusculas += 1;**

**}**

**if (Character.toLowerCase(letra) == 'a'**

**|| Character.toLowerCase(letra) == 'e'**

**|| Character.toLowerCase(letra) == 'i'**

**|| Character.toLowerCase(letra) == 'o'**

**|| Character.toLowerCase(letra) == 'u'){**

**vocales += 1;**

**}else{**

**consonantes += 1;**

**}**

**}**

**}**

**System.out.println("Número de letras: " + numLetras);**

**System.out.println("Número de dígitos: " + numDigitos);**

**System.out.println("Mayúsculas: " + mayusculas);**

**System.out.println("Vocales: " + vocales);**

**System.out.println("Consonantes: " + consonantes);**

**}**

**}**

**ARRAYS**

Un Array es un conjunto de elementos de un mismo tipo (o no) que están establecidos por un índice.

Un Array es estático, es decir, SIEMPRE tendrá el mismo número de elementos, aunque estén vacíos sus contenidos/valores.

Siempre se inicializan todos los elementos de su interior, es decir, ocupan espacios de memoria.

Además, puede tener posiciones vacías intercaladas y otras no vacías.

Es un conjunto que se inicializa con un número determinado de elementos (length).

Creamos un Array con tres elementos:

|  |  |
| --- | --- |
| 0 | “Dato1” |
| 1 | Null |
| 2 | “Dato3” |

Tenemos varias formas de inicializar un Array:

Ejemplo con int:

int[] arrayNumeros = new int[3];

String arrayString = new String[3];

Ahora mismo, el objeto arrayNumeros está inicializado con todos sus valores a CERO.

Ahora mismo, el objeto arrayString está inicializado con todos sus valores a NULL.

Si queremos establecer el contenido de algún elemento del Array se realiza mediante Objeto[INDICE] = valor;

arrayNumeros[0] = 999;

arrayNumeros[1] = 80078;

Los índices están delimitados:

arrayNumeros[5] = 9; ERROR!!!!

arrayString[1] = “Mensaje en posición 1”;

La segunda forma de inicializar un Array es declarar e indicar sus elementos:

int[] arrayNumeros = {88,77,66,33,44,22};

La única propiedad que tiene un Array es **length**

Vamos a crear un simple ejemplo de arrays **Class19Arrays.java**

**public class Class19Arrays {**

**public static *void* main(*String*[] *args*) {**

**//VAMOS A DECLARAR UN ARRAY PARA GUARDAR NOMBRES**

***String*[] nombres = new *String*[3];**

**//PODEMOS ACCEDER A SUS POSICIONES MEDIANTE EL INDICE**

**//PODEMOS VISUALIZAR LA LONGITUD**

**System.out.println("Longitud del Array: " + nombres.length);**

**//nombres[0] = "Lucia";**

**nombres[1] = "Adrian";**

**nombres[2] = "Diana";**

**//nombres[4] = "jajajajaja";**

**for (*int* i = 0; i < nombres.length; i++){**

**System.out.println("Nombres: " + i + " = " + nombres[i]);**

**}**

**//TAMBIEN PODEMOS TENER UN ARRAY INICIALIZADO CON CONTENIDO**

***int*[] numeros = new *int*[] {11,14,15,66,55,44,22,88};**

**System.out.println("Elementos en Array de int: " + numeros.length);**

**}**

**}**

Necesitamos un programa en el que pediremos al usuario 8 números.

Después de pedir 8 números al usuario, debemos analizar lo siguiente:

Mayor, Menor, Media y suma.

**Class20ArrayNumeros.java**

**import java.util.Scanner;**

**public class Class20ArrayNumeros {**

**public static *void* main(*String*[] *args*) {**

***Scanner* teclado = new Scanner(System.in);**

***int*[] numeros = new *int*[8];**

***int* mayor = 0;**

***int* menor = 0;**

***int* media = 0;**

***int* suma = 0;**

**for (*int* i = 0; i < numeros.length; i++){**

**System.out.println("Introduzca número " + (i + 1));**

***String* dato = teclado.nextLine();**

***int* num = Integer.parseInt(dato);**

**//almacenamos en el Array en cada posicion cada num**

**numeros[i] = num;**

**}**

**System.out.println("Hemos finalizado la petición");**

**//AHORA NOS CENTRAMOS EN NUESTRO ARRAY**

**mayor = numeros[0];**

**menor = numeros[0];**

**for (*int* i = 0; i < numeros.length; i++){**

**//ALMACENAMOS CADA NUMERO DEL ARRAY EN UNA VARIABLE**

***int* num = numeros[i];**

**suma += num;**

**mayor = Math.max(num, mayor);**

**menor = Math.min(num, menor);**

**}**

**media = suma / numeros.length;**

**System.out.println("Número mayor: " + mayor);**

**System.out.println("Número menor: " + menor);**

**System.out.println("Suma: " + suma);**

**System.out.println("Media: " + media);**

**}**

**}**

**ARRAYLIST**

Es lo mismo que un Array, es un conjunto de elementos por un índice.

A este objeto se le llama Colección.

Es un objeto dinámico, podremos almacenar tantos elementos como deseemos sin límite.

Nos devuelve el número de elementos reales de la colección.

La colección crece o decrece a medida que vamos incluyendo elementos en su interior.

Un Array utiliza en su declaración Genéricos.

Un Genérico es la declaración del tipo de datos que almacenaremos dentro de la colección.

Un genérico se representa con la letra T y diamantes: **<T>**

Tenemos dos formas de declarar una colección:

Esta colección NO tiene tipado y nos permite almacenar lo que deseemos en la colección: String, int, boolean.

ArrayList colección = new ArrayList();

La declaración con genérico implica que podemos almacenar con un tipado y es más óptimo además que nos mostrará errores de compilación.

ArrayList<**Clase**> colección = new ArrayList<**Clase**>();

La clase ArrayList contiene una serie de métodos para poder trabajar:

* **add**(objeto): Agrega un objeto a la colección
* **add(**posición, objeto): Agrega un objeto a la colección en la posición que yo le indique. La posición debe existir.
* **get**(índice): Recupera un objeto de la colección en su índice. Si el índice no existe, nos dará un error.
* **remove(objeto):** Elimina un objeto de la colección.
* **remove(index):** Elimina un objeto de la colección por su índice.
* **set(posición, objeto):** Modifica un elemento dentro de la colección
* **clear():** Elimina todos los elementos de la colección
* **size():** Nos devuelve el número de elementos de la colección
* **indexOf(objeto):** Busca un elemento en la colección y nos devuelve su posición. Si no lo encuentra, devuelve -1
* **lastIndexOf(objeto):** Busca un elemento en la colección desde el final y nos devuelve su posición. Si no lo encuentra, devuelve -1
* **contains(objeto):** Devuelve true/false si un objeto existe en la colección.

Para probar los métodos creamos una nueva clase llamada **Class21ArrayList.java**

**import java.util.ArrayList;**

**public class Class21ArrayList {**

**public static *void* main(*String*[] *args*) {**

***ArrayList*<*String*> coleccion = new *ArrayList*<*String*>();**

**//AGREGAMOS CUATRO ELEMENTOS y UNO REPETIDO**

**coleccion.add("Elemento 1");**

**coleccion.add("Elemento 2");**

**coleccion.add("Elemento 3");**

**coleccion.add("Elemento 1");**

**//PROBAMOS A MODIFICAR UN ELEMENTO**

**coleccion.set(1, "Nuevo elemento 2");**

**//PODEMOS INSERTAR ELEMENTOS**

**coleccion.add(1, "Insertado elemento");**

**//PODEMOS ELIMINAR ELEMENTOS, SI ENVIAMOS EL OBJETO**

**//ELIMINA EL PRIMERO QUE ENCUENTRA EN LA COLECCION**

**//coleccion.remove("Elemento 1");**

**//ELIMINAR POR INDEX PERMITE MAYOR PRECISION**

**//coleccion.remove(4);**

**//BUSCAMOS DENTRO DE LA COLECCION**

**System.out.println("Primera Posicion Elemento 1: "**

**+ coleccion.indexOf("Elemento 1"));**

**System.out.println("Ultima posición Elemento 1: "**

**+ coleccion.lastIndexOf("Elemento 1"));**

**System.out.println("Número de elementos: " + coleccion.size());**

**//VAMOS A REALIZAR UN BUCLE PARA VISUALIZAR LOS ELEMENTOS**

**//Y COMO SE COMPORTAN A MEDIDA QUE VAMOS TRABAJANDO**

**for (*int* i = 0; i < coleccion.size(); i++){**

**//RECUPERAMOS CADA ELEMENTO DE LA COLECCION**

***String* elemento = coleccion.get(i);**

**System.out.println(elemento);**

**}**

**//ELIMINAR TODA LA COLECCION**

**coleccion.clear();**

**System.out.println("Número de elementos: " + coleccion.size());**

**}**

**}**

Cuando estamos hablando de conjuntos, dichos conjuntos (Array, Colección, String)

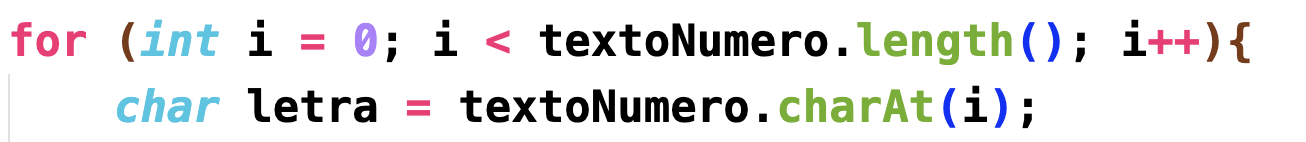
Podemos recorrerlo con Bucles contador y recuperar cada elemento de su interior con una variable que se denomina **Referencia**

En esta imagen, la variable **elemento** es una variable de Referencia.

Una variable de referencia indica que hace referencia a un objeto en el mismo espacio de memoria.

Texto

El contenido generado por IA puede ser incorrecto.



Tenemos un tercer tipo de bucle que se llama Bucle de referencia. Dicho bucle nos permite recorrer cada elemento de un conjunto con una variable de referencia.

Sintaxis:

for (TipoDato variableReferencia: CONJUNTO) {

}

for (char letra: textoNumero) {}

Interfaz de usuario gráfica, Texto, Aplicación

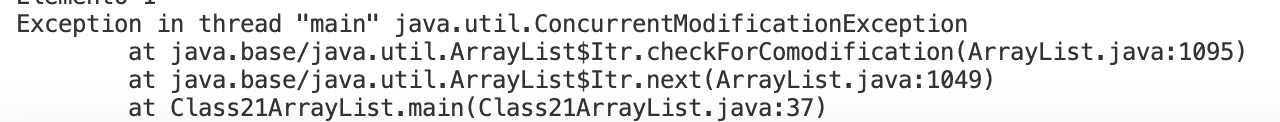
El contenido generado por IA puede ser incorrecto.

En este tipo de Bucles debemos tener en cuenta que son de SOLO LECTURA el conjunto que estamos recorriendo.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Esto mostrará una excepción al modificar el conjunto dentro del For Each



Si necesitamos modificar algo de la colección en su recorrido, tendremos que utilizar los bucles Contador

Las colecciones solamente admiten CLASES en su declaración.

**INCORRECTO**



**CORRECTO**



Necesitamos un programa que nos pedirá Nombres.

Cuando escribamos la palabra STOP, terminaremos de pedir nombres y mostraremos un Listado con todos los nombres que hemos recopilado.

También mostraremos el total de nombres en nuestra colección.

**Class22ArrayListNombres.java**

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

**import java.util.ArrayList;**

**import java.util.Scanner;**

**public class Class22ArrayListNombres {**

**public static *void* main(*String*[] *args*) {**

***Scanner* teclado = new Scanner(System.in);**

***ArrayList*<*String*> coleccionNombres = new *ArrayList*<*String*>();**

***String* nombre = "";**

**while (nombre.equalsIgnoreCase("stop") == false){**

**System.out.println("Introduzca un nombre");**

**nombre = teclado.nextLine();**

**coleccionNombres.add(nombre);**

**System.out.println("Nombres almacenados: "**

**+ coleccionNombres.size());**

**}**

**coleccionNombres.remove(coleccionNombres.size() - 1);**

**System.out.println("---------------------");**

**//RECORREMOS LA COLECCION Y DIBUJAMOS TODOS LOS NOMBRES**

**for (*String* name: coleccionNombres){**

**System.out.println(name);**

**}**

**System.out.println("Fin de programa");**

**}**

**}**

Necesitamos un programa que almacenará números.

Al final del programa mostraremos la Suma de dichos números y el listado de todos los números.

Pediremos números hasta que el usuario introduzca el número -1.

No queremos números repetidos, si el usuario introduce un número que ya se encuentra en la colección, se lo indicamos.

Creamos una clase llamada **Class23ArrayListNumeros.java**

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

**import java.util.ArrayList;**

**import java.util.Scanner;**

**public class Class23ArrayListNumeros {**

**public static *void* main(*String*[] *args*) {**

***Scanner* teclado = new Scanner(System.in);**

***ArrayList*<*Integer*> coleccionNumeros =**

**new *ArrayList*<*Integer*>();**

***int* suma = 0;**

***int* numero = 0;**

**while (numero != -1){**

**System.out.println("Introduzca un número");**

***String* dato = teclado.nextLine();**

**numero = Integer.parseInt(dato);**

**//PREGUNTAMOS SI EL NUMERO EXISTE EN NUESTRA**

**//COLECCION**

**if (coleccionNumeros.contains(numero)){**

**System.out.println("El número está repetido...");**

**}else{**

**coleccionNumeros.add(numero);**

**}**

**}**

**//ELIMINAMOS LA ULTIMA POSICION (-1)**

**//QUE NOS SOBRA**

**coleccionNumeros.remove(coleccionNumeros.size() - 1);**

**System.out.println("Números introducidos: "**

**+ coleccionNumeros.size());**

**System.out.println("--------------");**

**for (*int* num: coleccionNumeros){**

**suma += num;**

**System.out.println(num);**

**}**

**System.out.println("La suma de los números es: " + suma);**

**}**

**}**

**METODOS EN JAVA**

Un método es una funcionalidad de una clase/objeto.

Son un conjunto de acciones.

Dichas acciones pueden devolver valores o no.

Los métodos pueden estar en el mismo código/clase o en otros códigos/otras clases.

Lo más importante de los métodos es que nos permiten es reutilizar nuestro código.

Por un lado estará la declaración del método y, por otro lado, estará la llamada a dicho método.

La nomenclatura de los métodos en Java es con la primera letra en minúscula y, si tiene más palabras, en mayúscula cada letra inicial.

Tenemos dos tipos de métodos:

1. Métodos de acción: **void**: Dichos métodos son acciones que se ejecutan y que no devuelven ningún valor a la llamada

public void metodoAccion() {

//NUESTRO CODIGO

}

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

1. Métodos **return:** Dichos métodos ejecutan una serie de acciones y, al finalizar dichas acciones, devuelven un valor.

Sintaxis:

public TipoDato metodoReturn() {

//ACCIONES DEL METODO

return ValorTipoDato;

}

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

Los métodos pueden recibir parámetros en su cabecera (todos)

Los parámetros se incluyen dentro de los paréntesis separados por comas.

public void metodoParametros(**int num, String dato**) {}

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

Los parámetros NO SON OPCIONALES.

Creamos una clase de ejemplo llamada **Class24Metodos.java**

**public class Class24Metodos {**

**//ESTO ES UN METODO**

**//HASTA AHORA, SOLAMENTE ESCRIBIMOS AQUI**

**public static *void* main(*String*[] *args*) {**

**System.out.println("Ejemplo métodos");**

**//LLAMADA AL METODO**

**metodoAccion();**

**//EN LOS METODOS RETURN, DEBEMOS ALMACENAR**

**//LA RESPUESTA EN ALGUNA VARIABLE**

***String* saludo = metodoSaludo();**

**System.out.println(saludo);**

***int* suma = sumarNumeros(3, 4);**

**System.out.println("La suma es " + suma);**

**System.out.println("fin de programa");**

**}**

**//LOS METODOS SIEMPRE VAN A NIVEL DE CLASE**

**public static *void* metodoAccion() {**

**//UN METODO DE ACCION EJECUTA CODIGO**

**System.out.println("Soy un método de acciones");**

**}**

**//METODOS RETURN**

**public static *String* metodoSaludo(){**

**//ACCIONES DEL METODO**

**return "Mi primer método return, que ilusión";**

**}**

**//LOS METODOS PUEDEN RECIBIR PARAMETROS**

**//METODO PARA SUMAR DOS NUMEROS Y QUE DEVUELVA LA SUMA**

**public static *int* sumarNumeros(*int* *num1*, *int* *num2*){**

***int* suma = num1 + num2;**

**return suma;**

**}**

**}**

Necesito un sencillo programa que tenga un método que recibirá un número y nos devolverá si el número es POSITIVO, NEGATIVO o CERO.

En el **main()** principal, tendremos un Scanner que pedirá un número al usuario y llamaremos al método para comprobar su funcionalidad.

**Class25MetodoPositivoNegativo.java**

**import java.util.Scanner;**

**public class Class25MetodosPositivoNegativo {**

**public static *void* main(*String*[] *args*) {**

***Scanner* teclado = new Scanner(System.in);**

**System.out.println("Introduzca un número");**

***String* dato = teclado.nextLine();**

***int* valor = Integer.parseInt(dato);**

***String* data = evaluarNumero(valor);**

**System.out.println(data);**

**}**

**public static *String* evaluarNumero(*int* *numero*){**

***String* respuesta = "";**

**if (numero > 0){**

**respuesta = "POSITIVO";**

**}else if (numero == 0){**

**respuesta = "CERO";**

**}else {**

**respuesta = "NEGATIVO";**

**}**

**return respuesta;**

**}**

**}**

Necesito un programa que pedirá un Email al usuario y validaremos si es correcto o no.

Mediante un método que validará el mail, debemos indicar si es **true/false** el email.

Adaptamos el código que tenemos de validar un mail a este ejemplo.

En el **main()** principal, un Scanner para pedir un Mail y mostrar si es correcto o no.

**Class26MetodoEmail.java**

**import java.util.Scanner;**

**public class Class26MetodoEmail {**

**public static *void* main(*String*[] *args*) {**

***Scanner* teclado = new Scanner(System.in);**

**System.out.println("Validando un mail");**

**System.out.println("Introduzca un email");**

***String* dato = teclado.nextLine();**

***boolean* respuesta =**

**validarEmail(dato);**

**if (respuesta == true){**

**System.out.println("El email está correcto");**

**}else{**

**System.out.println("Email incorrecto, revise por favor");**

**}**

**}**

**//QUIERO UN METODO PARA SABER SI UN EMAIL ESTA BIEN O MAL**

**public static *boolean* validarEmail(*String* *email*){**

**if (email.contains("@") == false){**

**return false;**

**}else if (email.indexOf("@") == 0**

**|| email.endsWith("@")){**

**return false;**

**}else if (email.indexOf("@") != email.lastIndexOf("@")){**

**return false;**

**}else if (email.indexOf(".") == -1){**

**return false;**

**}else if (email.lastIndexOf(".") < email.indexOf(("@"))){**

**return false;**

**}else{**

***int* ultimoPunto = email.lastIndexOf(".");**

***String* dominio = email.substring(ultimoPunto + 1);**

**if (dominio.length() >= 2 && dominio.length() <= 3){**

**return true;**

**}else{**

**return false;**

**}**

**}**

**}**

**}**

Necesitamos un método para averiguar si un número es Narcisista o no.

Lo que quiero en mi programa es lo siguiente.

Necesito recorrer todos los números desde el 1 hasta el 20.000 y almacenar dentro de una colección todos los números Narcisistas.

Posteriormente, quiero dibujar dichos números.

**Class27MetodosNarcisista.java**

**import java.util.ArrayList;**

**public class Class27NumeroNarcisista {**

**public static *void* main(*String*[] *args*) {**

***ArrayList*<*Integer*> narcisistas = new *ArrayList*<*Integer*>();**

**for (*int* i = 1; i <= 20000; i++){**

**if (evaluarNarcisista(i) == true){**

**narcisistas.add(i);**

**}**

**}**

**//Recorremos todos los números narcisistas**

**for (*int* narci: narcisistas){**

**System.out.print(narci + ", ");**

**}**

**System.out.println("Fin de programa");**

**}**

**public static *boolean* evaluarNarcisista(*int* *numero*){**

***String* dato = String.valueOf(numero);**

***int* longitud = dato.length();**

***int* suma = 0;**

**for (*int* i = 0; i < longitud; i++){**

***char* letra = dato.charAt(i);**

***String* aux = String.valueOf(letra);**

***int* valor = Integer.parseInt(aux);**

***int* operacion = (*int*) Math.pow(valor, longitud);**

**suma += operacion;**

**}**

**if (suma == numero){**

**return true;**

**}else{**

**return false;**

**}**

**}**

**}**

**PROGRAMACION ORIENTADA A OBJETOS**

Hasta ahora, hemos estado trabajando con objetos simples.

Realmente, la POO es un rollo escucharla y tener toda la teoría, pero luego es muy simple de aplicar.

Son conceptos, da igual el lenguaje, siempre son los mismos.

Existen cuatro definiciones para POO:

* **Encapsulación**: Implica la visibilidad objetos. Mediante los ámbitos, podemos indicar la visibilidad de diferentes objetos entre ellos.

Un objeto solamente será visible si yo lo necesito.

Por ejemplo, imaginemos un coche. Tenemos una clase coche y podemos conducir un Coche.

Un coche tiene motor? SI

Un coche tiene volante? SI

Para que un conductor pueda utilizar un coche, necesita saber algo del motor? No, el coche tiene motor, pero el conductor NO necesita saber nada del motor para conducir. El motor es privado.

Para que un conductor pueda conducir, necesita acceso al volante? Si, el volante es público y el conductor lo necesita.

* **Abstracción**: Indica que podemos reconocer un objeto por sus características y manejarlo como un elemento BASE.

Todos los objetos tienen una familia en Java. Dicha familia indica que contendrá una serie de elementos comunes entre si.

Posteriormente, tendrá sus propias características.

Tenemos un Turismo, Deportivo y un 4x4.

¿Qué tienen en común estos elementos? Son coches.

Si yo se conducir, puedo manejar un turismo, un deportivo y un 4x4.

* **Polimorfismo**: Una acción puede tener múltiples formas.

Dichas formas son repetidas por la misma acción, pero no son duplicadas con acciones distintas. Utilizamos una sola forma.

Tenemos solamente un pedal de freno en un coche.

Podemos frenar de múltiples formas, lo que NO implica que tengamos un pedal de freno por cada forma de frenado.

Tenemos un solo pedal de freno/acción y múltiples formas de manejar dicho pedal.

* **Herencia**: Todo objeto hereda de una clase y, de esa forma, hereda sus propiedades y puede implementarlas. La clase de la que hereda un objeto se denomina en Java: **base**

Pongamos que necesito construir un Coche. Creamos un motor, un volante, marchar…etc. Realizamos métodos de frenar, arrancar…etc.

Posteriormente, tenemos que crear un 4x4. Podemos heredar de coche y añadir lo que necesitemos

Además, le ponemos las marchas de subir por las montañas.

Posteriormente, tenemos que crear un Deportivo. Podemos heredar de coche y añadir lo que necesitemos.

Además, le ponemos que pueda correr un montón.

Posteriormente, tenemos que crear un Formula 1. Podemos heredar de Deportivo y ponemos las mejoras.

Cuando hablamos de visibilidad, estamos hablando de clases.

También hablamos de métodos/propiedades dentro de dichas clases.

Tenemos palabras clave que implican la visibilidad de un objeto/clase:

* **public**: Visibilidad total. Se podrá visualizar la clase/método en cualquier lugar.
* **private**: Solamente será visible en la clase dónde se ha declarado.
* **protected**: Ámbito para le herencia. Solamente será visible si se hereda del objeto.
* **none (no existe):** No se incluye ningún ámbito en la declaración. Solamente será visible dentro del mismo **package**

Una clase en Java está compuesta por las siguientes características:

* **Package:** Un paquete indica el nombre de dónde está alojada la clase. Es como un término que indica un conjunto de elementos para poder agruparlos en un mismo lugar.

Cada paquete puede estar formado por una o varias clases.

Es solo una forma de agrupación.

* **Visibilidad**: Ámbito de acceso a una clase.
* **Propiedades:** Son características de un objeto. Indican elementos que forman un objeto. Si hablamos de Coche, son el Color, la velocidad, etc.
* **Métodos:** Son acciones que podemos realizar con un objeto. Si hablamos de Coche, arrancar(), frenar(), acelerar(), etc.
* **Constructor**: Un constructor indica cómo se crea un objeto. Es la forma de instanciar un objeto y puede tener polimorfismo.

En Java, no se utilizan Propiedades. Solamente métodos Getter y Setter.

Ejemplo en otros lenguajes:

public class Coche {

public String Color;

public int Velocidad;

public void arrancar() { … }

public void acelerar() { … }

}

Vamos a trabajar con un nuevo proyecto de Java. Esta vez, no será una carpeta con clases, sino que será un proyecto con clases/packages/libraries.

Dentro del proyecto, vamos a comenzar a crear un objeto **Persona**.

Una persona tendrá tres características:

Nombre, Apellidos y Edad.

El nombre del proyecto lo llamaremos **proyectopoo**

En Visual, seleccionamos Show And Run Commands

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Imagen que contiene Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Vamos a comenzar creando un nuevo package llamado **com.objetos**

Para crear un paquete, necesitamos dos características.

1. Crear el paquete mediante botón derecho y **Create Package**

Dentro del Package, debemos poner nuestras nuevas clases.

Creamos una clase nueva llamada **Prueba.java**

1. Compilar el proyecto para que se “entere Java” que tenemos un Package nuevo

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Comenzamos creando una clase llamada **Persona.java** dentro de **com.objetos**

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Es el momento de utilizar nuestra clase dentro de **App.java**

**APP.JAVA**

**import com.objetos.Persona;**

**public class App {**

**public static *void* main(*String*[] *args*) throws *Exception* {**

**System.out.println("Probando la clase Persona");**

**//CUANDO HABLAMOS DE CLASES, NECESITAMOS OBJETOS PARA**

**//TRABAJAR CON DICHAS CLASES**

**//INSTANCIAMOS EL OBJETO PERSONA**

***Persona* personaje = new Persona();**

**//YA PODEMOS UTILIZAR LA PERSONA Y SUS PROPIEDADES**

**personaje.nombre = "Jon";**

**personaje.apellidos = "Nieve";**

**personaje.edad = 29;**

**System.out.println(personaje.nombre + " " + personaje.apellidos**

**+ ", Edad: " + personaje.edad);**

**}**

**}**

METODOS GETTER Y SETTER PROPIEDADES

En Java se utilizan métodos para poder acceder a las propiedades.

Los métodos implementan un campo/variable privada mediante un GET y un SET.

Se declara una variable privada para que nadie tenga acceso a la propiedad, ni pueda modificarla.

Como desarrollador, yo decido si genero los campos getter y setter.

GETTTER: Es para recuperar el valor de la propiedad

SETTER: Es para establecer el valor de la propiedad.

Sintaxis de Propiedad Getter y Setter.

**private int propiedad;**

**public int getPropiedad(){**

**return propiedad;**

**}**

**public void setPropiedad(int valor) {**

**propiedad = valor;**

**}**

Si necesitamos, desde otra clase acceder al valor de la propiedad:

int numero = getPropiedad();

Si necesitamos almacenar el valor de la propiedad:

setPropiedad(33);

Modificamos el código de **Persona**

**PERSONA**

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

Posteriormente, para poder acceder o modificar el nombre del objeto:

Texto

El contenido generado por IA puede ser incorrecto.

El siguiente concepto es la utilización de la palabra clave **this**

**this** permite acceder a los datos declarados dentro de una propia clase.

La palabra clave this hace referencia a la clase en la que estamos trabajando.

Pongamos como ejemplo el siguiente código:

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

¿Qué número está recuperando para el doble?

El número 2 o el número 4? Está recuperando el valor de la variable más cercana a su declaración, es decir, el número 4.

Imaginemos que necesitamos hacer referencia a la variable numero de la clase, es decir, la variable que contiene el número 2.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

En los métodos SETTER se suele recibir el nombre de la variable igual que el nombre de la propiedad.

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

Vamos a instalar una extensión que nos permite generar GETTER y SETTER en nuestro código.

Texto

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Una vez que tenemos generados los códigos de Get y Set, es el momento de controlar nuestra Propiedad Edad.

Como dijimos ayer, la edad no queremos que sea negativa y ahora tenemos un método para poder realizar la comprobación.

De nosotros depende que deseamos hacer cuando nos manden una edad negativa.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

**EXCEPCIONES EN JAVA**

Una excepción indica que ha sucedido algo grave en nuestro programa y la ejecución se detiene.

Por ejemplo, si tenemos un ArrayList e indicamos un índice que no existe, nos dará una excepción de tipo IndexOutBoundException.

Si dividimos un número entre cero, nos dará una excepción de tipo DivideByZeroException.

Texto

El contenido generado por IA puede ser incorrecto.

Existen ocasiones en las que yo no puedo controlar/saber si sucederá una excepción o no sucederá.

Por ejemplo, imaginemos que estamos accediendo a una base de datos.

Para acceder a una base de datos, necesitamos conectar con dicha base de datos.

¿Qué sucede si el servidor de base de datos está apagado?

Para estas situaciones se utilizan bloques **try…catch**

Un bloque try/catch nos permite controlar una excepción dentro de nuestro código para que dicha excepción no detenga nuestra aplicación.

Sintaxis

try {

//CODIGO A EJECUTAR QUE PUDIERA DAR EXCEPCION

} catch (TipoExcepcion ex) {

//SOLAMENTE ENTRA SI TENEMOS EXCEPCION

//EN LA VARIABLE ex VIENE INFORMACION SOBRE LA EXCEPCION

}

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

Dentro de un código es posible que tengamos más de una posible Excepción.

Podemos capturar cada excepción mediante un bloque catch personalizado con cada tipo.

try {

//CODIGO A EJECUTAR QUE PUDIERA DAR EXCEPCION

} catch (TipoExcepcion1 ex) {

//Exception tipo 1

} catch (TipoExcepcion2 ex) {

//Exception tipo 2

}

Texto

El contenido generado por IA puede ser incorrecto.

El código anterior sirve para personalizar entre tipos de excepción.

A lo mejor lo que quiero es simplemente capturar cualquier tipo de excepción general.

Tenemos la clase Exception que es la clase **BASE** de las excepciones y sirve para capturar cualquier excepción.

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

**PROPAGACION DE EXCEPCIONES**

Algunos métodos en Java propagan excepciones, de esa forma, sabemos si el método nos podría dar una excepción o no.

Para propagar una excepción en un método se utiliza la palabra **throw**.

Con throw podemos lanzar una excepción en nuestro código e indicar a quién utilice nuestro método que debe capturar dicha excepción.

Sintaxis:

**throw new TipoException ( mensaje de error )**

Automáticamente, en el momento de escribir cualquier throw, el método debe ir con la declaración de la excepción.

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

Al propagar la excepción en el método, automáticamente, quien utilice mi método debe controlar dicha excepción.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Al estar obligado a capturar la excepción, es cuando estamos obligados a utiliza try/catch

Imagen que contiene Texto

El contenido generado por IA puede ser incorrecto.

También puedo “pasar” de la excepción y propagar la excepción a su vez.

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

Si quiero personalizar la excepción “cuando suceda”, utilizaré try/catch

Si me da igual, utilizaré throws.

Lo siguiente es utilizar esta teoría con la Edad.

Si un programador indica una edad negativa, eso me pone nervioso y le mandaremos una Exception.

**PERSONA**

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Por último, dentro de try/catch, existe un último bloque opcional llamado **finally**

El bloque finally **SIEMPRE** se ejecuta.

* Si nos da error, se ejecuta
* Si no tenemos error, se ejecuta

Dicho bloque se escribe después del último catch:

try{

//código a ejecutar

}catch (Exception ex) {

//solo si tenemos una excepción

}finally {

//siempre se ejecuta

}

Pongamos un ejemplo de cuando utilizar dicho método.

Imaginemos que estamos accediendo a una base de datos.

Como os he comentado, al acceder a una base de datos, necesitamos una conexión.

Cuando terminemos de realizar las acciones sobre la base de datos, siempre debemos liberar dicha conexión, es decir, cerrar la conexión.

try{

//1. nos conectamos a la bbdd

//2. leemos registros

}catch (Exception ex) {

//capturamos un posible error

} finally {

//3. cerramos la conexión

}

**PROPIEDADES ENUMERADAS**

Una propiedad enumerada es una herramienta de desarrollo que indica una serie de posibilidades al programador.

En lugar de dejar que el programador indique un valor, le ofrecemos los posibles valores nosotros.

Una enumeración son una serie de valores CONSTANTES con un número asociado en su interior.

Una constante es una variable que se establece una vez y no se puede cambiar nunca su valor. Ejemplo: **Número PI**.

Pongamos un ejemplo. Necesitamos una propiedad que nos indique una dirección, es decir, una propiedad Brujula.

La dirección puede ser Norte, Sur, Este y Oeste.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

El siguiente paso es que alguien utilice mi brujula.

El usuario que utilice mi brujula, debe escribir correctamente cada dato, lo que es imposible.

Para dejar solamente al programador una serie de posibilidades, debemos crear primero dichas posibilidades. Esas posibilidades cerradas se llaman **enumeraciones**

Las constantes en Java se escriben en mayúsculas.

public enum PUNTOSCARDINALES {

NORTE, SUR, ESTE, OESTE

}

Una vez que tenemos la enumeración, creamos una propiedad del tipo ENUMERADO.

private PUNTOSCARDINALES brujula;

Cuando el programador vaya a utilizar nuestra Brujula:

persona.setBrujula(NORTE);

persona.setBrujula(SUR);

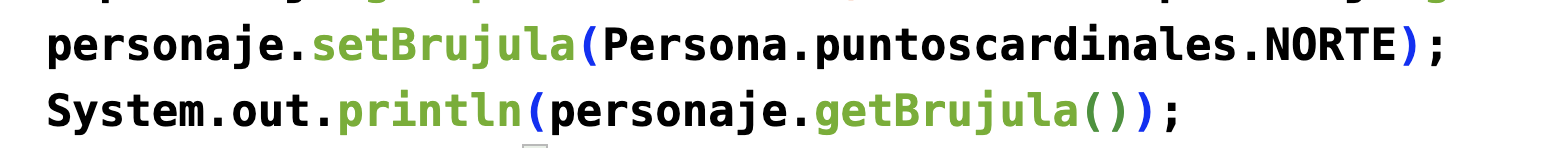
Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

Cuando utilicemos la brujula:



Vamos a crear una enumeración dentro de Persona que indique el género de una persona.

Posteriormente, creamos una propiedad para asignar dicho género.

**PERSONA**

Diagrama

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

En la clase **App.java** realizamos la llamada a la nueva propiedad.

**APP.JAVA**



Y podremos visualizar el resultado:

Texto

El contenido generado por IA puede ser incorrecto.

**METODOS**

Vamos a crear un método para que nos devuelva el nombre completo de una persona.

**PERSONA**

Imagen que contiene Gráfico

El contenido generado por IA puede ser incorrecto.

Vamos a crear otro método que nos devuelva el nombre completo en mayúsculas.

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

Creamos otro método para devolver el nombre completo al revés (apellidos y nombre)

Texto

El contenido generado por IA puede ser incorrecto.

Por último, creamos un método para devolver nombre y apellidos recibiendo el separador que deseemos.

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

Y podremos probar la aplicación en **App.java**

Imagen que contiene interior, botella, tabla

El contenido generado por IA puede ser incorrecto.

Ahora mismo tenemos múltiples métodos que realizan lo mismo (devolver un nombre completo) y le estamos dando distintos nombres.

Es una locura, en realidad, lo que deberíamos tener es un solo método y múltiples opciones para devolver una característica u otra, es decir, Polimorfismo.

El polimorfismo se denomina sobrecarga de un método que implica que un mismo nombre de método tendrá múltiples formas.

Para tener múltiples formas de un método es necesario que tenga parámetros diferentes en su **FIRMA**

La firma de un método es el nombre del método y sus parámetros.

Un dibujo de un animal

El contenido generado por IA puede ser incorrecto.

No podemos tener métodos con la misma **FIRMA**.

Imagen que contiene Texto

El contenido generado por IA puede ser incorrecto.

Imagen que contiene Texto

El contenido generado por IA puede ser incorrecto.

**PERSONA.JAVA**

**//METODO PARA DEVOLVER EL NOMBRE Y APELLIDOS**

**public *String* getNombreCompleto(){**

**return *this*.nombre + " " + *this*.apellidos;**

**}**

**//METODO PARA DEVOLVER EL NOMBRE Y APELLIDOS EN MAYUSCULAS**

**public *String* getNombreCompleto(*int* *numero*){**

**return *this*.nombre.toUpperCase()**

**+ " " + *this*.apellidos.toUpperCase();**

**}**

**//METODO PARA DEVOLVER APELLIDOS Y NOMBRE**

**public *String* getNombreCompleto(*boolean* *orden*){**

**return *this*.apellidos + " " + *this*.nombre;**

**}**

**//METODO PARA DEVOLVER NOMBRE Y APELLIDOS MEDIANTE**

**//EL SEPARADOR QUE DECIDA EL PROGRAMADOR**

**public *String* getNombreCompleto(*String* *separador*){**

**return *this*.nombre + separador + *this*.apellidos;**

**}**

**//LO QUE IMPORTA ES LA FIRMA DE UN METODO**

**public *void* getNombreCompleto(*int* *num1*, *int* *num2*){}**

**public *int* getNombreCompleto(*String* *a*, *int* *b*){**

**return 19;**

**}**

Una persona tiene un número DNI.

A partir de ese DNI, me gustaría saber mi LETRA.

Estamos hablando de Propiedad, de método o de qué: Las dos cosas.

Debemos crear una propiedad **int** llamada **dni**

Posteriormente, crearemos un método llamado **getLetraNif()** que nos calculará la letra del número del DNI y nos la devolverá como **char**

**Nota:** Utilizamos el código que ya tenemos del ejemplo del proyecto **fundamentosjavagetafe** y la clase **Class13CalcularDni.java**