# Métodos

En java un método es un conjunto de instrucciones definidas en una clase que realizan tareas y se invocan mediante nombres. Consta de tres partes:

* Firma: El nombre y parámetros del método
* Cuerpo: Conjunto de instrucciones que posee el método
* Valor de retorno: Hay ciertos métodos que devuelven valores

tipoDeAcceso modificador tipoMétodo nombreMétodo(tipoArg arg) {

declaración de variables globales

código del método

}

Public static void metodoEjemplo(String ejemplo) {

System.out.println(ejemplo)

}

## Usos

Reutilización del código: Permite llamar el vario método varias veces para evitar repetir código

Organización: Ayudan a dividir el código en bloques más pequeños

Abstracción: Permiten encapsular la lógica dentro de una interfaz simple

Lectura: Al usar nombres descriptivos se mejora la legibilidad.

## Métodos con retorno

Los métodos con tienen un tipo de dato de retorno especifico declarado. Se utiliza la palabra **return** para devolver un valor.

tipoDeAcceso tipoDato nombreMetodo(parametros) {

código;

return valor;

}

Public int sumar (int a, intb) {

Suma = a + b;

Return suma;

}

En los métodos sin retorno se usa **void** como tipo de dato para indicar que no va a devolverse nada.

# Variables globales y locales

En java el alcance de las variables puede variar en función de donde y como estén declaradas

## Variables globales:

Son variables que están declaradas fuera de los métodos y, por tanto, son accesibles desde cualquier parte de la clase.

## Variables locales:

Son aquellas que están creadas dentro de los métodos y solo se pueden utilizar dentro de ellos.

# Pila y recursividad

La pila es una estructura de datos que sigue el principio de “último en llegar, primero en salir”. Se utiliza para almacenar datos temporales (variables locales, registros de llamada a métodos) cada vez que se llama a un método se crea un marco de pila para ese método en la pila.

## Recursividad

La recursividad es un concepto donde una función se llama a sí misma para resolver un problema más pequeño del mismo tipo

# Arraylist

La clase Arraylist, es una matriz de datos, pero, a diferencia de Array, es dinámica ,por tanto, se le pueden añadir datos sin definir su tamaño y se pueden añadir datos de forma indefinida durante la ejecución del programa ya que su tamaño se ajusta de automáticamente

ArrayList<tipoDato> nombreArraylist = new ArrayList<>();

nombreArraylis.add(dato1);

nombreArraylis.add(dato2);

Arraylist<String> listaNombres = new Arraylist<>();

listaNombres(“Pablo”);

listaNombre(“Pedro”);

Para utilizarlo hace falta importar la clase ArrayList:

import java.util.ArrayList;

## Métodos:

* .add: Añade un dato
* .remove: Elimina un dato
* .get: Permite acceder a un dato
* .set: Modifica un dato
* .clear: Borra todo el contenido del ArrayList
* .size: Devuelve el tamaño del ArrayList

# Clases y objetos

Java es un lenguaje de programación orientado a objetos por lo cual gira entorno a su utilización de estos.

## Clases:

En java una clase es una estructura a partir de la cual se van a crear objetos. Define las propiedades y acciones (métodos) que van a poseer los objetos y actúa como un molde para los objetos:

tipoDeAcceso class nombreClase {

Atributo1

Atributo2

tipoDeAcceso nombreClase(Atributos){

}

Método{

}

}

public class Coche {

String modelo;

String marca;

String color;

}

### Tipos:

* Regulares: Son las más comunes y se utilizan para definir objetos y sus comportamientos. Poseen atributos y métodos.
* Internas: Permiten definir clases dentro de otras clases. Suelen ser estáticas y se utilizan para mejorar la encapsulación y estructura.

## Objetos:

El propósito de los objetos es el de representar entidades de la realidad a través de los atributos y métodos definidos en la clase. Al crear un objeto se reserva memoria para el mismo, se inicializan sus atributos y se invoca al constructor de la clase.

Clase nombreObjeto = new Clase(Atributos);

Coche miCoche = new Coche(“Ford”, ”Mustang”, ”Gris”)

## Constructor:

Es un método especial que define dentro de la clase y que se invoca al declarar un objeto de dicha clase. Su función es la de inicializar los atributos de la clase que se le pasen.

public clase (Atributos){

this.atributo1 = atributo1;

}

public Coche(String modelo, String marca, String color){

this.modelo = modelo;

this.marca = marca;

this. color = color;

}

Llamada al constructor:

Coche(“Ford”, ”Mustang”, ”Gris”)

Es el único tipo de método que no retorna un valor y que no utiliza void.

## Métodos

Dentro de una clase se pueden definir métodos que pueden actuar sobre los objetos de la clase.

public void mostrarInformacion() {

System.out.println(“Modelo: ” + modelo);

System.out.println(“Marca: ” + marca);

System.out.println(“Color: ” + color);

}

miCoche.mostrarInformacion();

# Import y package

Para poder acceder a clases que se encuentran en otros archivos o carpetas se usan las palabras clave import y package.

## Import:

Permite usar una clase que se encuentre en otro paquete (package);

Import carpeta1.carpeta2.nombreClase;

Import Java.util.Scanner;

## Package:

Permite que se pueda acceder al paquete en el que se encuentra una clase para poder utilizarla

Package nombreDelPaquete;

Los nombres son mayoritariamente en minúscula

# Abstracción

La abstracción es un proceso en el que representa un sistema complejo de forma conceptual. Implica identificar las partes que componen un concepto (aparatos, vehículos, animales…) y representarlas.

## Clases Abstractas:

Las clases abstractas son aquellas que se utilizan para definir conceptos muy generales (Vehiculo, animal, planta…) y que pueden derivar en conceptos más concretos (Coche, Pajaro, Arbol). No tienen una implementación directa en el código y definen los atributos y métodos que vayan a tener las subclases. Tienen varias reglas:

* No pueden ser instanciadas
* Se declara la existencia de métodos, pero no su implementación
* Al menos uno de sus métodos debe ser abstracto
* Deben ser public o protected
* Se pueden crear varias subclases

En resumen son clases Padre (o superclases) que representan conceptos generales que poseen subclases que representan conceptos más concretos y no se pueden implementar.

public abstract class Animal {

public String nombre;

public Animal(String nombre) {

this.nombre = nombre;

}

public abstract void hacerSonido();

}

class Perro extends Animal {

public Perro(String nombre) {

super(nombre);

}

public void hacerSonido() {

System.out.println(nombre + " dice: Guau!");

}

}

# encapsulación

Es la práctica por la que se separan del resto del código atributos y funcionalidades específicas de los objetos. Consiste en agrupar los datos e impedir el acceso del usuario, se usan los modificadores de acceso y los métodos getter y setter.

## Modificadores de acceso:

* Public: Permite que se pueda acceder desde cualquier otra clase.
* Protected: Solo permite que puedan acceder clases del mismo paquete (package) y las que heredan de ella.
* Private: Solo permite acceder desde la misma clase

## Métodos Get y Set:

Son métodos que permiten acceder a un dato de un objeto y modificarlo:

* Get: Sirve para acceder al dato del objeto.

Objeto.getNombre(); //Devuelve el nombre del objeto

* Set: Sirve para modificar el dato objeto.

Objeto.setNombre(“Hola”); //Cambia el nombre del objeto a “Hola”

# Herencia

La herencia, permite que las subclases hereden los atributos y métodos de la clase padre, para poder definir métodos y atributos comunes entre varias clases

## Tipos de herencia:

* Única: Una subclase hereda de la clase padre (Animal > Pajaro)
* Múltiple: Una subclase hereda de otra subclase que a su vez hereda de una clase padre (Vehículo, Vehículo a motor, Coche)

public class Animal {

String nombre;

int edad;

public Animal(String nombre, int edad) {

this.nombre = nombre;

this.edad = edad;

}

public void comer() {

System.out.println("El animal está comiendo.");

}

public void dormir() {

System.out.println("El animal está durmiendo.");

}

}

Para que una clase hereda de otra se usa la palabra clave **extends**

class Perro extends Animal {

String raza;

public Perro(String nombre, int edad, String raza) {

super(nombre, edad); // Llama al constructor de la clase padre (Animal)

this.raza = raza;

}

public void ladrar() {

System.out.println("Guau! Guau!");

}

}

# Polimorfismo

Es el concepto de que un objeto puede comportarse de diferentes formas dependiendo del contexto. Para aplicar dicho concepto se suelen usar las clases abstractas e interfaces.

## Clases Abstractas:

abstract class Figura {

abstract double calcularArea();

void mostrarInformacion() {

System.out.println("Esta es una figura.");

}

}

class Circulo extends Figura {

double radio;

Circulo(double radio) {

this.radio = radio;

}

@Override **//Se sobreescrive el método heredado**

double calcularArea() {

return Math.PI \* radio \* radio;

}

}

class Rectangulo extends Figura {

double longitud;

double anchura;

Rectangulo(double longitud, double anchura) {

this.longitud = longitud;

this.anchura = anchura;

}

@Override**//Se sobreescrive el método heredado**

double calcularArea() {

return longitud \* anchura;

}

}

Aquí se aplica el polimorfismo porque se calcula el área de distintas formas

## Interfaces

Es una clase de la cual no se pueden crear instancias y permiten que otras clases implementen sus métodos. Se declaran usando la palabra clave interface. Pueden estar compuestas únicamente por métodos y constantes

public interface IEjemplo {6

void método();

}

* Sus métodos son implícitamente abstractos (No es necesario especificarlo)
* Las clases que los implementan deben sobrescribir todos sus métodos
* Sus variables y atributos son implícitamente constantes (No es necesario especificarlo)
* Una clase puede extender de una sola clase pero implementar varias interfaces
* Para que una interfaz herede de otra se usa la palabra clave extends