Java

martes, 21 de febrero de 2023 20:41

Links de interés:

Fundamentos del lenguaje Java (ehu.es)	

Notas rápidas

Char str de una sola palabra

Notas: Java distingue entre minúsculas y mayúsculas

Añadir final a una variable, la transforma en constante

Nombredevariable= new scanner(System.in) == **input** nextInt();

Charart te guarda el primer carácter del srt

Entrada de salida en ventan emergente(
JptionPane.showinputdialog

Convertir str a int: integer.parseint(nombrevariable)
Convertir str a decimal: doble.parseDoble(nombrevariable)

}

Public: modificación de acceso

Todo programa Java tiene que estar dentro de una clase

Sintaxis básica:

```
public class claseGlobal{

/* This is my first java program.
    * This will print 'Hello World' as the output
    */

public static void main(String []args) {
      System.out.println("Hello World"); // prints Hello World
   }
}
```

Las palabras reservadas de Java

En todos los lenguajes de programación existen palabras con un significado especial. Estas palabras son reservadas y no se pueden utilizar como nombres de variables.

abstract	final	public
assert	finally	return
boolean	float	short
break	for	static
byte	if	strictfp
case	implements	super
catch	import	switch
char	instanceof	synchronized
class	int	this
continue	interface	throw
default	long	throws
do	native	transient
double	new	true
else	null	try
enum	package	void
extends	private	volatile
false	protected	while

Tipo	Descripción	Valor mínimo y máximo
byte	Entero con signo	-128 a 127
short	Entero con signo	-32768 a 32767
int	Entero con signo	-2147483648 a 2147483647
long	Entero con signo	-92211703685477580 8 a +9221170368547758 07
float	Real de precisión simple	±3.40282347e+38 a ±1.40239846e-45
double	Real de precisión doble	±1.7976931348623157e+309 a ±4.94065645841246544e-324
char	Caracteres Unicode	\u0000 a \uFFFF
boolean	Valores lógicos	true, false

Clases y objetos

```
public class Vehiculo {

   String matricula;
   String marca;
   String modelo;
   String color;
   double tarifa;
   boolean disponible;

}
```

Scanner:

Method	Description
nextBoolean()	Reads a boolean value from the user
nextByte()	Reads a byte value from the user
nextDouble()	Reads a double value from the user
nextFloat()	Reads a float value from the user
nextInt()	Reads a int value from the user
nextLine()	Reads a String value from the user
nextLong()	Reads a long value from the user
nextShort()	Reads a short value from the user

Las palabras reservadas de Java

En todos los lenguajes de programación existen palabras con un significado especial. Estas palabras son reservadas y no se pueden utilizar como nombres de variables.

abstract	final	public
assert	finally	return
boolean	float	short
break	for	static
byte	if	strictfp
case	implements	super
catch	import	switch
char	instanceof	synchronized
class	int	this
continue	interface	throw
default	long	throws
do	native	transient
double	new	true
else	null	try
enum	package	void
extends	private	volatile
false	protected	while

OPERADOR	DESCRIPCIÓN
==	Es igual
!=	Es distinto
<, <=, >, >=	Menor, menor o igual, mayor, mayor o igual
&&	Operador and (y)
| |	Operador or (o)
!	Operador not (no)

Carácter	Significado
\b	Retroceso
\t	Tabulador
\n	Salto de línea
\r	Cambio de línea
\"	Carácter comilla doble
\'	Carácter comilla simple
\\	Carácter barra hacia atrás

Switch:

CONDICIONAL DE SELECCIÓN SWITCH EN JAVA. EJEMPLO DE APLICACIÓN.
La instrucción switch es una forma de expresión de un anidamiento múltiple de instrucciones if ... else. Su uso no puede considerarse, por tanto, estrictamente necesario, puesto que siempre podrá ser sustituida por el uso de if. No obstante, a veces nos resultará útil al introducir mayor claridad en el código.

Example

```
public class Main {
  public static void main(String[] args) {
    int day = 4;
    switch (day) {
      case 1:
        System.out.println("Monday");
        break;
      case 2:
        System.out.println("Tuesday");
        break;
      case 3:
        System.out.println("Wednesday");
        break;
```

Default

La cláusula default es opcional y representa las instrucciones que se ejecutarán en caso de que no se verifique ninguno de los casos evaluados. El último break dentro de un switch (en default si existe esta cláusula, o en el último caso evaluado si no existe default) también es opcional, pero lo incluiremos siempre para ser metódicos.
Switch solo se puede utilizar para evaluar ordinales (por ordinal entenderemos en general valores numéricos enteros o datos que se puedan asimilar a valores numéricos enteros). Por tanto no podemos evaluar cadenas (String) usando switch porque el compilador nos devolverá un error de tipo "found java.lang.String but expected int". Si se permite evaluar caracteres y lo que se denominan tipos enumerados, que veremos más adelante. Switch solo permite evaluar valores concretos de la expresión: no permite evaluar intervalos (pertenencia de la expresión a un intervalo o rango) ni expresiones compuestas. Código de ejemplo:

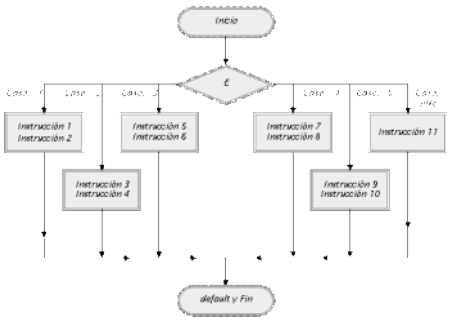
```
/* Ejemplo método que usa switch - aprenderaprogramar.com */
public void dimeSiEdadEsCritica() {
```

```
        case 1:
          System.out.println("Monday");
          break;
        case 2:
          System.out.println("Tuesday");
          break;
        case 3:
          System.out.println("Wednesday");
          break;
        case 4:
          System.out.println("Thursday");
          break;
        case 5:
          System.out.println("Friday");
          break;
        case 6:
          System.out.println("Saturday");
          break;
        case 7:
          System.out.println("Sunday");
          break;
      }
    }
```

numéricos enteros). Por tanto no podemos evaluar cadenas (String) usando switch porque el compilador nos devolverá un error de tipo "found java.lang.String but expected int". Sí se permite evaluar caracteres y lo que se denominan tipos enumerados, que veremos más adelante. Switch solo permite evaluar valores concretos de la expresión: no permite evaluar intervalos (pertenencia de la expresión a un intervalo o rango) ni expresiones compuestas. Código de ejemplo:

```
/* Ejemplo método que usa switch - aprenderaprogramar.com */
public void dimeSiEdadEsCritica() {
        switch (edad) {
            case 0:
            System.out.println ("Acaba de nacer hace poco. No ha cumplido el año");
            break;
            case 18: System.out.println ("Está justo en la mayoría de edad"); break;
            case 65: System.out.println ("Está en la edad de jubilación"); break;
            default: System.out.println ("La edad no es crítica"); break;
        }
    }
```

Esquemáticamente en forma de diagrama de flujo:



BLOQUE TRY

Try en inglés es el verbo intentar, así que todo el código que vaya dentro de esta sentencia será el código sobre el que se intentará capturar el error si se produce y una vez capturado hacer algo con él. Lo ideal es que no ocurra un error, pero en caso de que ocurra un bloque try nos permite estar preparados para capturarlo y tratarlo. Así un ejemplo sería:

```
try {
    System.out.println("bloque de código donde pudiera saltar un error es este");
      }
```

BLOQUE CATCH

En este bloque definimos el conjunto de instrucciones necesarias o de tratamiento del problema capturado con el bloque try anterior. Es decir, cuando se produce un error o excepción en el código que se encuentra dentro de un bloque try, pasamos directamente a ejecutar el conjunto de sentencias que tengamos en el bloque catch. Esto no es exactamente así pero ya explicaremos más adelante todo el funcionamiento. De momento para una mejor comprensión vamos a considerar que esto es así.

```
catch (Exception e) {
    System.out.println("bloque de código donde se trata el problema");
}
```

Fíjate que después de catch hemos puesto unos paréntesis donde pone "Exception e". Esto significa que cuando se produce un error Java genera un objeto de tipo Exception con la información sobre el error y este objeto se envía al bloque catch.

BLOQUE FINALLY

Y para finalizar tenemos el bloque finally que es un bloque donde podremos definir un conjunto de instrucciones necesarias tanto si se produce error o excepción como si no y que por tanto se ejecuta siempre.

```
finally {
    System.out.println("bloque de código ejecutado siempre");
}
```

equalsIgnoreCase()

Compara dos strings para ver si son iguales ignorando las diferencias entre mayúsculas y minúsculas. Este método es necesario porque no es posible comparar strings usando el operador de igualdad (==). Retorna true si los strings son iguales y false si no lo son.

Parámetros str String: Cualquier string valido

Sintaxis: equalsIgnoreCase(str)

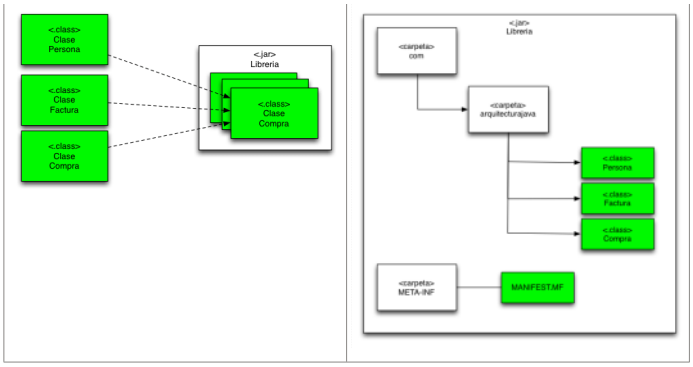
```
voidsetup() {
  Serial.begin(9600);
  Stringstr1 = String("CCCP");
  Stringstr2 = String("ccCP");
  // Prueba para ver si str1 es igual a str2if(str1.equalsIgnoreCase(str2) == true) {
    // Ellos son iguales ignorando la diferencia entre may&uacute;sculas y // min&uacute;sculas entonces imprimir&aacute;Serial.println("Igual");
  }
  else{
    Serial.println("No es Igual");
  }
}

voidloop() {

}



Stringstr1 = String("CCCP");
Stringstr3 = String("cccp");
// Prueba para ver si str1 es igual a str3voidsetup() {
  Serial.begin(9600);
  if(str1.equalsIgnoreCase(str3) == true) {
    // Ellos son iguales ignorando la diferencia entre may&uacute;sculas y // min&uacute;sculas entonces imprimir&aacute;Serial.println("Igual");
  }
  else{
    Serial.println("No es Igual");
  }
}

voidloop() {

}
```

Programación modular

La programación modular está basada en la técnica de diseño descendente, que como ya vimos consiste en dividir el problema original en diversos subproblemas que se pueden resolver por separado, para después recomponer los resultados y obtener la solución al problema

¿Qué es Java JAR?

Un JAR no es ni más ni menos que un empaquetado zip que contiene un conjunto de clases Java organizadas por packages (carpetas) de forma jerárquica. La peculiaridad que tienen los ficheros JAR es que los podemos reutilizar entre proyectos ya que simplemente son empaquetados



Java JAR

Como podemos ver nos encontramos con la estructura de paquetes original que teníamos y dentro de esa estructura se encuentrannuestras clases. Además de esta carpeta con los paquetes nos encontramos con la carpeta META-INF que incluye el fichero de Manifiesto (MANIFEST.MF) .Este fichero nos aporta información adicional sobre nuestro fichero JAR. Uno de sus usos más habituales es definir qué clase de todas las que tenemos ubicadas en el JAR es la clase que se debeejecutar como programa principal. Esto añadiendo las siguientes líneas al fichero.

```
Manifest-Version: 1.0
Main-Class: com.arquitecturajava.Principal
```

Estas líneas declaradas en el fichero de Manifiesto permitirá que cuando invoquemos en linea de comandos el siguiente comando

```
java -jar milibreria.jar
```

Se ejecute el código de la clase Principal . Hemos revisado la estructura de un JAR en el siguiente post cubriremos los WAR

- Java JAR , Java Archive y empaquetamiento - Arquitectura Java
- Java 9 Modules y el concepto de modularidad - Arquitectura Java
- Programacion Java. Programación modular

Calculadora por módulos

Primero, crea un archivo Calculator.java y escribe el siguiente código:

```
module Calculator {
    requires java.base;

    exports com.example.calculator;
}
```

Este archivo define el módulo Calculator, que requiere el módulo java.base y exporta el paquete com.example.calculator para ser utilizado por otros módulos.

Luego, crea un paquete com.example.calculator y agrega los siguientes archivos:

Operation.java

```
package com.example.calculator;

public interface Operation {
    double calculate(double x, double y);
}
```

Esta interfaz define la operación básica que todas las operaciones de la calculadora deben implementar.

Addition.java

```
package com.example.calculator;

public class Addition implements Operation {
    @Override
    public double calculate(double x, double y) {
        return x + y;
    }
}
```

Este archivo define la clase Addition, que implementa la interfaz Operation para realizar la operación de suma.

Subtraction.java

```
package com.example.calculator;

public class Subtraction implements Operation {
    @Override
    public double calculate(double x, double y) {
        return x - y;
    }
}
```

Este archivo define la clase Subtraction, que implementa la interfaz Operation para realizar la operación de resta.

Multiplication.java

```
package com.example.calculator;

public class Multiplication implements Operation {
    @Override
    public double calculate(double x, double y) {
        return x * y;
    }
}
```

Este archivo define la clase Multiplication, que implementa la interfaz Operation para realizar la operación de multiplicación.
Division.java

```
package com.example.calculator;

public class Division implements Operation {
    @Override
    public double calculate(double x, double y) {
        return x / y;
    }
}
```

Este archivo define la clase Division, que implementa la interfaz Operation para realizar la operación de división.

Por último, crea un archivo Main.java en el mismo paquete y escribe el siguiente código:

```
package com.example.calculator;

import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        double x, y;
        System.out.print("Enter first number: ");
        x = scanner.nextDouble();
        System.out.print("Enter second number: ");
        y = scanner.nextDouble();

        Operation addition = new Addition();
        Operation subtraction = new Subtraction();
        Operation multiplication = new Multiplication();
        Operation division = new Division();

        System.out.println("Addition: " + addition.calculate(x, y));
        System.out.println("Subtraction: " + subtraction.calculate(x, y));
        System.out.println("Multiplication: " + multiplication.calculate(x, y));
        System.out.println("Division: " + division.calculate(x, y));
    }
}
```

Este archivo define la clase Main, que utiliza la entrada del usuario para crear objetos de las clases Addition, Subtraction,Multiplication y Division y llamar al método calculate para realizar las operaciones.
Ahora puedes compilar y ejecutar el programa. Asegúrate de que todos los archivos estén en la misma estructura de paquetes yque estés ubicado en la raíz de la estructura de paquetes.

```
javac com/example/calculator/*.java
```

Este comando compilará todos los archivos Java en el paquete com.example.calculator.

Luego, ejecuta el programa con el siguiente comando:

```
java com.example.calculator.Main
```

Este comando ejecutará el programa y mostrará el siguiente resultado en la consola:

```
Enter first number: 10
Enter second number: 5
Addition: 15.0
Subtraction: 5.0
Multiplication: 50.0
Division: 2.0
```

Este resultado muestra el resultado de cada operación realizada en la calculadora.
Espero que este ejemplo te ayude a crear tu propia calculadora modular en Java. Recuerda que puedes agregar más operaciones omejorar el programa para adaptarlo a tus necesidades.

¿Cómo saber si dividir el proyecto en bloques dentro de un archivo o hacerlo en diferentes archivos?

Depende del tamaño del mismo, si es un proyecto grande se ha de dividir los módulos en diferentes archivos, para que este máslimpio y más ordenado, en caso de que se un proyecto pequeño, es mejor dividirlo en bloques dentro de un mismo archivo.

Java con Azure

Spring Boot

Spring Boot se ejecuta sobre Spring Framework y tiene una configuración automática bien fundamentada. Las aplicaciones de Spring Boot se ejecutan de forma independiente y son una opción perfecta para los microservicios.

Spring Data

Spring Data simplifica el acceso a los datos para bases de datos relacionales y no relacionales, marcos MapReduce y servicios de datos basados en la nube.

Seguridad de Spring

Spring Security es el estándar de facto para proteger las aplicaciones basadas en Spring. Ofrece un alto nivel de autenticación y control de acceso. Por ejemplo, Spring Security se integra bien con Active Directory.

Spring Cloud

Spring Cloud se usa para sistemas distribuidos. Spring Cloud incluye detección de servicios, administración de configuración, supervisión y una buena experiencia de desarrollador.

Spring Batch

Spring Batch es un marco ligero para aplicaciones de lote sólidas que son vitales para las operaciones diarias.

En Java, la anotación **@Override** se utiliza para indicar que un método en una subclase está sobrescribiendo a un método con el mismo nombre y firma en una sup erclase. Es decir, si una subclase tiene un método con la misma firma (nombre, número y tipo de parámetros y tipo de retorno) que un método en su super clase, el método en la subclase puede anotarse con **@Override** para indicar que se trata de una implementación sobrescrita.
La anotación **@Override** es opcional en Java, pero es una buena práctica utilizarla para garantizar que estemos sobrescribiendo correctamente un métod o de una superclase. Además, si utilizamos esta anotación en un método que no está sobrescribiendo a ningún método en una superclase, el compilador generará un error de compilación. Esto nos ayuda a detectar posibles errores de programación.

Programación orientada a objetos (POO)

¿Qué es una clase?

Podemos entender una clase como la definición de un tipo abstracto de dato que contiene atributos y métodos. A través de una clase podemos representar un objeto abstraído de la realidad. Una clase es solo la DEFINICION de un objeto con los que el pro grama que estamos desarrollando tiene que tratar.

Estructura de una clase: Atributos

Los atributos de una clase se especifican mediante declaraciones de los datos. Por ejemplo: La clase Persona podría tener las siguientes declaraciones para reflejar su nombre, apellidos y edad: String nombre; String apellidos; int edad; Cuando hacemos declaraciones, podemos declarar también otros tipos de datos. Podría tener una clase EquipoFutbol que entre sus atributos tenga un tipo de dato Persona.

```
public class EquipoFutbol {
Persona jugador; String nombreEquipo;
}
```

Recordemos la convención para la definición de los datos: Consta de letras, símbolos de subrayado y dígitos, empezar por una letra, no se permiten espacios, son Case sensitive, el uso de mayúsculas en las palabras interiores cuando son compuestas.

Estructura de una clase: Métodos

Las acciones o habilidades de una clase se expresan con uno o más métodos. Un método es una secuencia de instrucciones a las que se da un nombre único. La llamada a un método hace que se ejecute el conjunto de instrucciones dentro de ese método. Lo q ue hasta ahora hemos llamado una función o un procedimiento. Por tanto, un método puede tener uno o varios datos de entrada, y también puede tener un dato de salida.

```
public void mirar(){
System.out.println("Está mirando");
}
```

Un método puede declarar sus propios campos (variables) que son locales, y por ello no son accesibles desde otros métodos aun que estén definidos en la misma clase. Pero sí puede modificar los atributos de la clase.

Estructura de una clase: Constructores

El programador que escribe la clase puede definir uno o más constructores que tengan diferente número de parámetros.
Los constructores tienen el mismo nombre que la clase. Por lo tanto, podríamos tener los siguientes constructores para la cla se Persona:

```
//Constructor por defecto: no tiene parámetros
Persona(){
 nombre="";
apellidos="";
edad = 0;
}
//Constructor con parámetros
Persona(String nom, String ape, int ed){
nombre=nom;
apellidos=ape;
edad=ed;
}
```

Los objetos que no han pasado por el proceso de la construcción reciben un valor especial (el objeto nulo) de la JVM. Pudiend o comprobar si un objeto es nulo. Si tratamos de utilizar los miembros de un objeto nulo, Java lanza la excepción NullPointer Exception.

Visibilidad de atributos y métodos.

Con visibilidad nos referimos al nivel de accesibilidad de los atributos y métodos. En Java los niveles de accesibilidad se d an por las siguientes palabras reservadas:

- **public** Se puede acceder desde un método implementado desde cualquier clase.
- **private** Sólo se puede acceder desde un método implementado en la propia clase.
- **protected** Se puede acceder desde un método implementado en una clase que "hija", esto es, que herede la clase que contiene esta visibil idad y desde clases que se encuentren en el mismo paquete.

Encapsulamiento y el apuntador "this".

Encapsulamiento: es una característica que indica que los atributos que definen las propiedades de la clase deben tener visibilidad *private*. De esta forma se ofrece seguridad a la información que se encuentra en estos atributos.
La **norma** es definir los atributos como *private* precisamente para ofrecer esta seguridad en la integridad de la información contenida en estos atributos.

El **apuntador *"this"*** permite acceder a los atributos y métodos de la clase. No es obligatorio su uso siempre, pero se recomienda como buena prácti ca. Pero sí es obligatorio cuando desde un método queremos acceder a un atributo de la clase que tiene el mismo nombre que un parámetro de dicha función.
Es una forma de acceder directamente a cualquiera de sus miembros.

Ejemplo de código:

```
class Cliente {
    private final String id;
    private String nombre;
    private String direccion;
    private String telefono;
    private String descripcion;
    private String tipoCliente;
    private LocalDate fechaInicio;
    private String jefeProyecto;
    private String cliente;

    /**
     * Crea un nuevo cliente con los datos indicados.
```

```
     *
     * @param id           el identificador del cliente.
     * @param nombre       el nombre del cliente.
     * @param direccion    la dirección del cliente.
     * @param telefono     el teléfono del cliente.
     * @param descripcion  la descripción del cliente.
     * @param tipoCliente  el tipo de cliente.
     * @param fechaInicio  la fecha en la que el cliente comenzó a trabajar con la
     *                     empresa.
     * @param jefeProyecto el jefe de proyecto asignado al cliente.
     */
    public Cliente(String id, String nombre, String direccion, String telefono, String descripcion,
            String tipoCliente, LocalDate fechaInicio, String jefeProyecto) {
        this.id = Objects.requireNonNull(id, "El identificador del cliente no puede ser nulo");
        this.nombre = Objects.requireNonNull(nombre, "El nombre del cliente no puede ser nulo");
        this.direccion = Objects.requireNonNull(direccion, "La dirección del cliente no puede ser nula");
        this.telefono = Objects.requireNonNull(telefono, "El teléfono del cliente no puede ser nulo");
        this.descripcion = Objects.requireNonNull(descripcion, "La descripción del cliente no puede ser nula");
        this.tipoCliente = Objects.requireNonNull(tipoCliente, "El tipo de cliente no puede ser nulo");
        this.fechaInicio = Objects.requireNonNull(fechaInicio, "La fecha de inicio del cliente no puede ser nula");
        this.jefeProyecto = Objects.requireNonNull(jefeProyecto, "El jefe de proyecto del cliente no puede ser nulo");
    }
```

¿Qué es un objeto? Características y definición.

Un objeto es una realización concreta de una descripción de clase.

Podemos entender una clase como un fichero en el que se define (y solo se define) los atributos y los métodos de un objeto. P ero no tiene "vida", esto es, no tiene recursos de memoria asignados y no podemos hacer nada con él, ni invocar sus métodos n i acceder a sus atributos.

```
// Ejemplo de objeto
public class JefeProyecto {
    private String nombre;
    private String direccion;
    private String telefono;
    private String descripcion;
    private String DNI - ID;
}
```

Para pasar de esta definición a una realización concreta de esta descripción necesitamos "construir" este elemento. Es lo que denominamos **constructor**.
Para poder llamar a sus métodos es necesario que esté instanciado (esté creado). Por tanto ya no es una clase sino que pasa a tener unas características determinadas.

Por ejemplo:
Podemos tener definida una clase Persona, con sus atributos y métodos, pero no tenemos ninguna realización concreta hasta que no creemos uno o varios objetos.

```
Persona juan = new Persona("Juan", "Sánchez", 55);
Persona ana = new Persona("Ana", "Lamas", 54);
//Creación de dos objetos llamando a sus constructores.
```

```
// clase Persona
class Persona{
String nombre;
String apellido;
Byte edad;
}
```

En las instrucciones anteriores estamos creando dos objetos, uno denominado **juan** y otro denominado **ana** (lo que hasta ahora hemos llamado variables). Estos objetos son de tipo Persona.
Y los hemos creado invocando un constructor utilizando la palabra reservada ***new.***

Cada uno de estos objetos tiene todas los **atributos** y **métodos** definidos en la clase **Persona**. Y cada objeto tiene una copia de todos los campos definidos para esa clase.

Estrictamente, las variables son de un tipo y los objetos pertenecen a una clase.

Crear un objeto e invocar a un método de su clase.

Crear un objeto:

```
Persona ana = new Persona("Ana", "Lamas", 54);
```

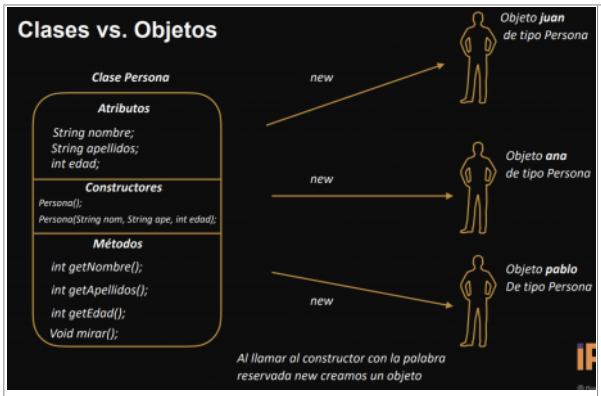
Invocar un método de su clase:
La forma de indicar a quién le queremos aplicar las instrucciones de un método es anteponer el nombre del objeto al nombre de l método.

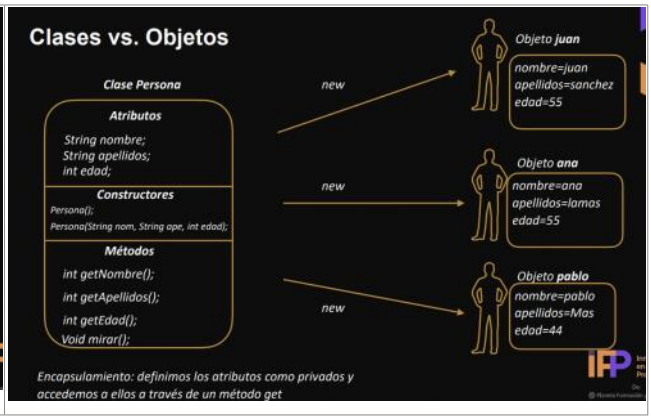
Ejemplo:

```
ana.mirar();
```

Un programa y sus clases.

¿Qué relación tiene un programa principal con las clases?
Pues un programa realizado en un lenguaje de programación orientado a objetos, como Java, consta de una o más clases interdep endientes. Pueden estar relacionadas entre sí o no.





Atributos de clase y de objetos.

Decíamos anteriormente que al crear un objeto llamando a su constructor, se crea una nueva copia de todos los campos (atribut os) declarados para esa clase. Con lo que cada valor almacenado en cada atributo es único de ese objeto. Es lo que denominamo s atributo de objeto.
En algún caso conviene tener un atributo que no sea único a ese objeto, sino que sea común a todos los objetos de su clase. E sto es lo que denominamos **atributo de clase**.

Para crear un atributo de clase debemos utilizar la palabra reservada **static**. Indica que el almacenamiento para el campo se crea una única vez para la clase, y no cada vez que se crea un nuevo objeto.

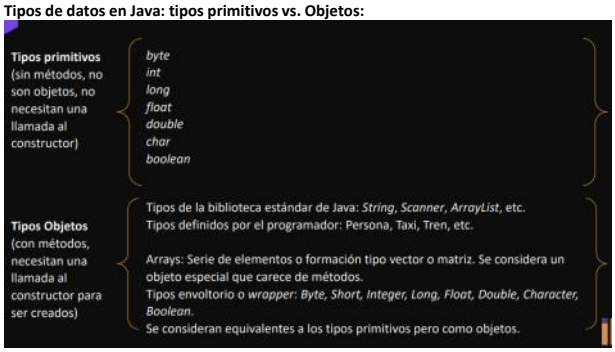
¿Cómo nos referimos a los atributos de clase y de objetos? Aplicando la siguiente regla:

- Dentro de su clase, nos referiremos a los campos por sus nombres únicamente.
- Cuando se usan desde otra clase u objeto, anteponemos al nombre del campo el nombre de su clase (para campos de clase) o el d e su objeto (para campos de objeto).

Métodos de clase y de objetos.

Siguiendo la convención de los campos de clase, un método de clase se define con el modificador static y se llama con el nomb re de la clase como prefijo.

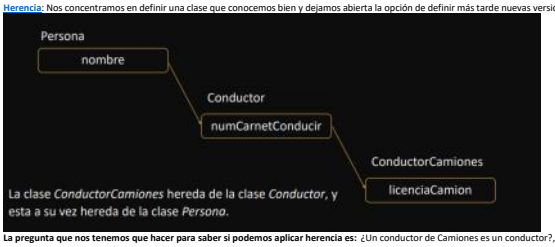
Tipos de datos en Java: tipos primitivos vs. Objetos:



Características POO: Composición, herencia y abstracción.

En Java utilizamos la palabra reservada "extends" para reflejar que una clase hereda (extiende) de otra.

- **Composición**: Es cuando definimos un atributo de una clase de un tipo que se corresponde con otra clase. Esto es, crear un objeto de una clase basada en otra clase
- **Herencia**: Nos concentramos en definir una clase que conocemos bien y dejamos abierta la opción de definir más tarde nuevas versiones de ella. Estas nuevas versiones "heredarán" los atributos de la clase original, y por tanto pueden ser más pequeñas y más lim pias.



La pregunta que nos tenemos que hacer para saber si podemos aplicar herencia es: ¿Un conductor de Camiones es un conductor?, sí. Luego podemos aplicar herencia. Y, ¿un conductor es una persona?, sí. Luego p odemos aplicar también herencia entre la clase Conductor y la clase Persona.

- **Abstracción**: Permite que una clase o método se concentre en lo esencial de lo que está haciendo (su conducta e interfaz con el mundo) co nfiando en que los detalles se resolverán posteriormente. (La abstracción consiste en seleccionar datos de un conjunto más gr ande para mostrar solo los detalles relevantes del objeto)

Sobreescritura de métodos

Una subclase puede definir su propia versión de un método que ya se ha implementado en una superclase.

Se trata de ir añadiendo versiones más especificas una vez descendamos más en la jerarquía.

Clases y métodos abstractos

- **Métodos abstractos.**
  Son "casillas" para métodos que se deben mencionar en cierto nivel, pero sólo se implementarán más abajo en la jerarquía de c lases.
- **Clases abstractas.**
  Es aquella clase que contiene al menos un método abstracto. Una clase abstracta no se puede utilizar para declarar objetos, l a presencia del método abstracto significa que está incompleta.

Herencia de clases: superclases y subclases.

La superclase *Object:*

Java alcanza la generalidad utilizando objetos de diferentes clases en las mismas partes del programa, pero todos ellos perte necen a la clase Object. Esto significa también que los métodos que están definidos en esta clase los podemos sobreescribir e n las clases hijas.

Para comparar los valores de dos objetos (todos juntos) debemos utilizar el método equals, que es preciso definir para cada c lase nueva de objetos. Esto supone que el programador debe decidir cómo quiere definir la igualdad. Por ejemplo, en el ejempl o que estamos realizando, diremos que dos personas son iguales si sus nombres, apellidos y edad coinciden.

Otra operación que tiene que ser tenida en cuenta es la asignación entre objetos. Al asignar dos objetos, por ejemplo: ana = juan; Se copiarán las referencias. Para copiar los valores, es necesario clonar el objeto utilizando un método que también ha de ser definido para cada nueva clase. Este método es clone(). Por ejemplo, clonar un objeto de tipo Persona supone copiar todos los datos que contiene.

Interfaces

Una interfaz es un tipo especial de clase que define (y solo define) la especificación de un conjunto de métodos. Eso es todo .

Definición de una interfaz:

```
1  Interface nombreDeInterfaz{
2   Especificaciones de los métodos
3  }
```

Implementación de una interfaz:

```
1  Class nombreDeClase implements nombreDeInterfaz{
2  Cuerpo de los métodos de la interfaz datos y métodos propios
3  }
```

Destructores, finalización de objetos y liberación de memoria.

Un destructor es un método opuesto a un constructor, este método en lugar de crear un objeto lo destruye liberando la memoria de nuestro dispositivo para que pueda ser utilizada por alguna otra variable u objeto.
En java no existen los destructores, esto es gracias al recolector de basura (garbage collector) de la máquina virtual de jav a.
En cambio en otros lenguajes de programación (por ejemplo, C) es necesario implementarlo ya que en caso contrario se quedan e spacios en la memoria ocupados que no están siendo utilizados por nuestro programa.

```
1  System.gc();
```

Otras características de la POO

Bajo acoplamiento:
Indica que los diferentes subsistemas deben estar unidos de forma mínima, es decir, deben ser lo más reducidos posibles

Alta cohesión:
Indica que, los atributos y métodos de una clase deben ser conscientes con el concepto que abstrae dicha clase. Es decir los atributos y métodos deben estar referidos a las características y habilidades en dicha clase.

Polimorfismo:
En POO, y siendo rigurosos, el polimorfismo se refiere a la capacidad que poseen las clases "hijas" de utilizar un mismo méto do heredado de forma distinta. Pueden inducir a confusión con la sobrecarga, pero en este caso hay herencia.