**Apuntes JAVA**

**DEFINICIONES Y ORIGEN DEL LENGUAJE**

Java es un lenguaje orientado a objetos, es multiplataforma.

Características principales de Java:

-Portable

-Fácil

-Segura

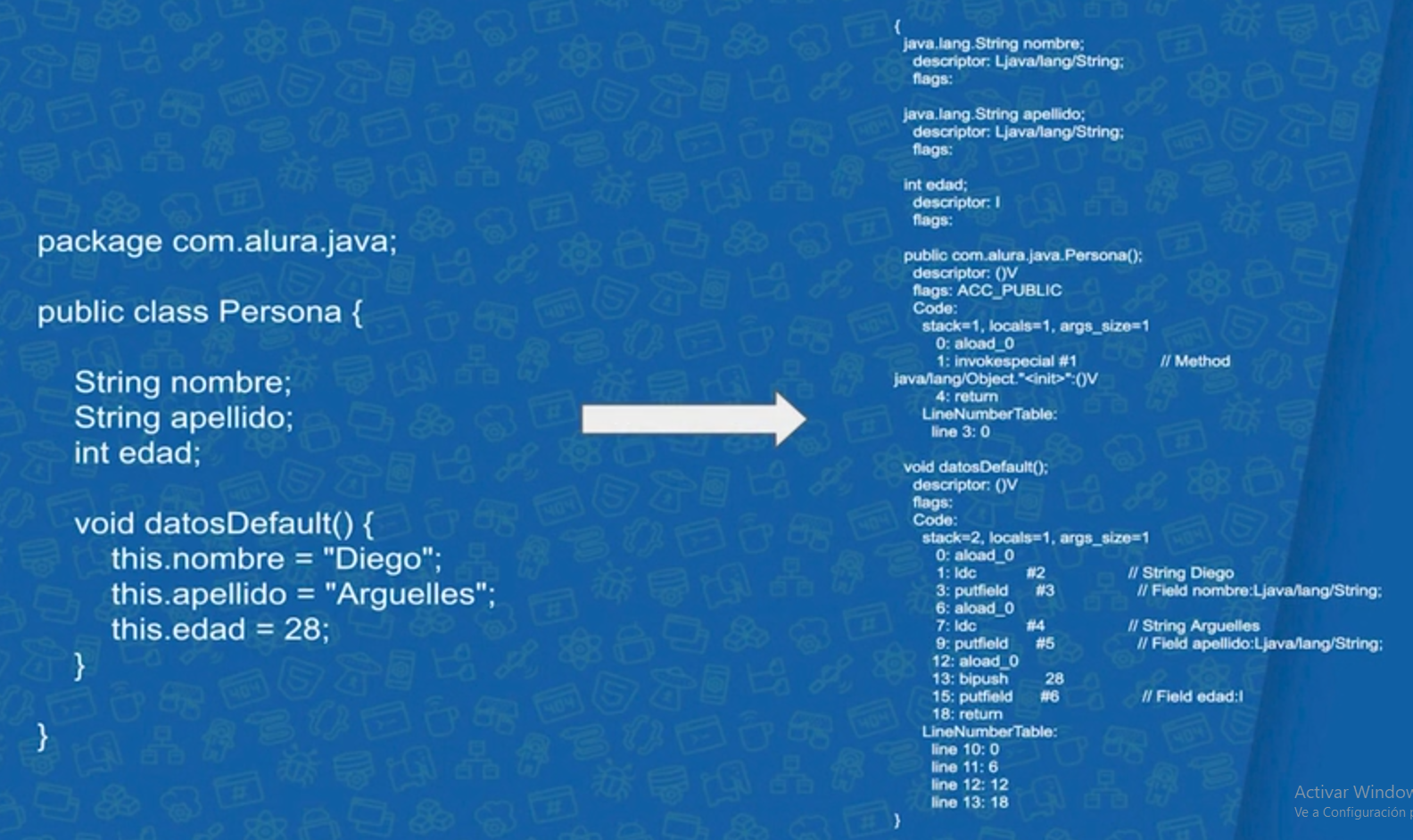
-Omnipresente, puede estar en servidores, dispositivos android, computadoras y hasta impresoras.

James Gosling, es conocido como el padre de Java, él, en compañía de otras personas crearon la JVM (Java Virtual Machine) que es capaz de leer el bytecode y mandarle instrucciones al sistema operativo para que ejecute las tareas o procesos de dicho bytecode, el bytecode se genera directamente de los archivos .class que compilamos al escribir nuestro codigo en lenguaje Java. El bytecode es el lenguaje maquina por decirlo así.

Diagrama, Texto

Descripción generada automáticamente

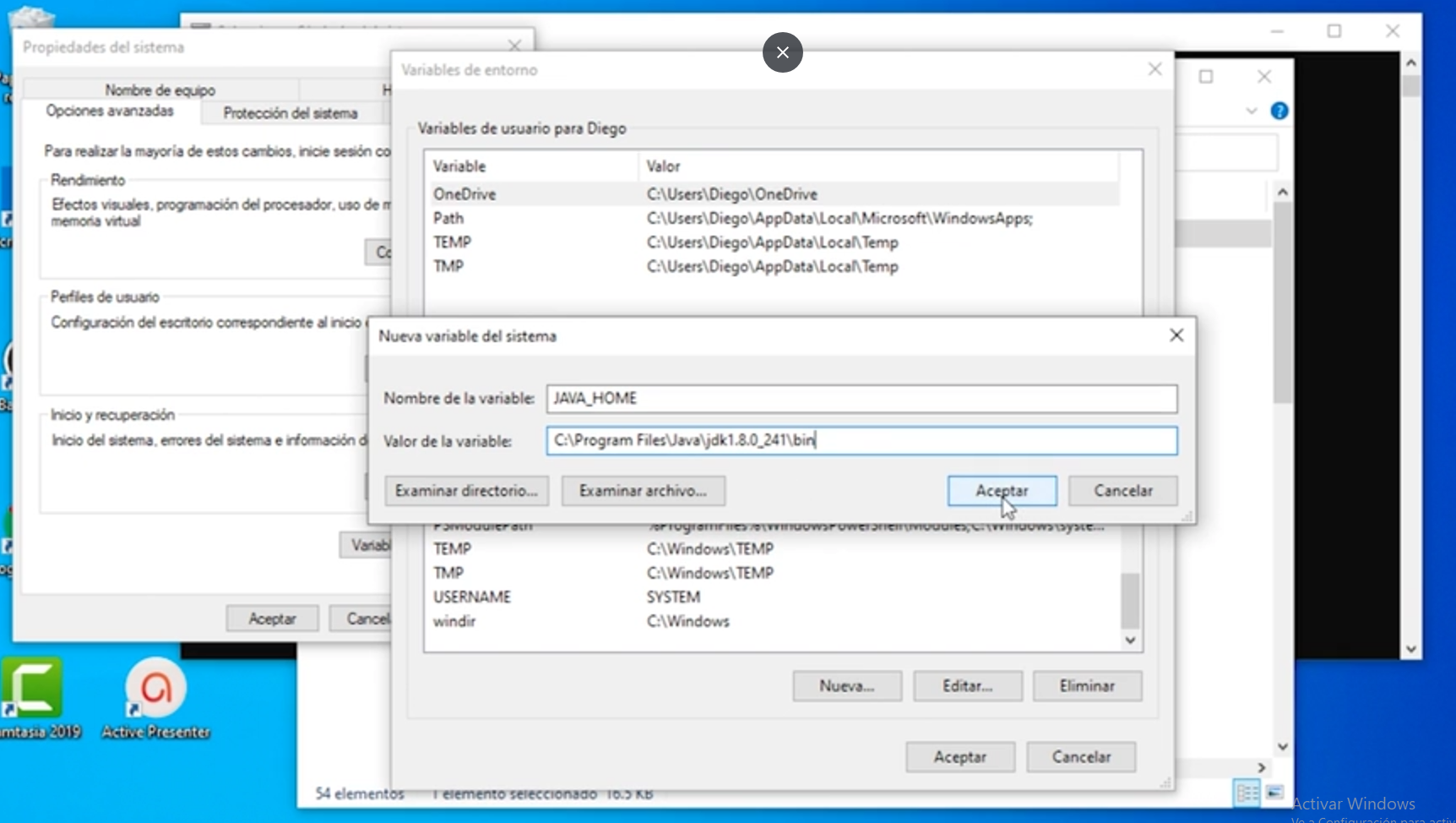
La transformación de nuestro código en java, que es humanamente legible, a lenguaje de maquina sería el siguiente.



Como observamos, es mucho mas complejo el lenguaje maquina.

Tal vez encontraste extraño el nombre Bytecode (yo también), sin embargo, tiene una explicación muy simple: existe un conjunto de comandos que la máquina virtual de Java entiende, esos comandos son también llamados opcodes (operation code) y cada opcode posee exactamente 1 Byte de tamaño. Entonces, tenemos un opcode de 1 Byte, o más simple, Bytecode.

Para iniciar con java es importante configurar las variables de entorno, nos vamos a variables de Windows y le damos en “Nueva”



Luego nos vamos a PATH y le damos en editar, pegamos la ruta bin de java, es todo

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

JRE, Java Runtime Environment, junto con el JVC y las librerías de java, todo este ecosistema nos permitirá ejecutar código Java en nuestra computadora.

Luego tenemos el JDK, Java Development Kit, que, en conjunto con el JRE nos brindan mas herramientas para desarrollar aplicaciones en este lenguaje.

El JRE es para usuarios normales y el JDK es para programadores ya que nos brinda herramientas para desarrollar.

Interfaz de usuario gráfica

Descripción generada automáticamente

No es necesario tener un IDE para programar en Java, podemos hacerlo en un bloc de notas y luego ir a la consola, escribir “javac” espacio, NombreClase.java, esto hará crear un archivo .class que estará traducido a código máquina para que al momento que escribamos en consola “Java” espacio NombreClase, sin ninguna extensión, en la consola se nos muestre el resultado. SIEMPRE que hagamos algún cambio en la clase debemos compilar y ejecutar con javac y java respectivamente.

Tema conversiones de cast, hay que ver la longitud de bits, con cast podemos hacer que un tipo de variable de mayor bit quepa en uno de menor bit pero a cambio que haya perdida de información. Ver imagen anexa

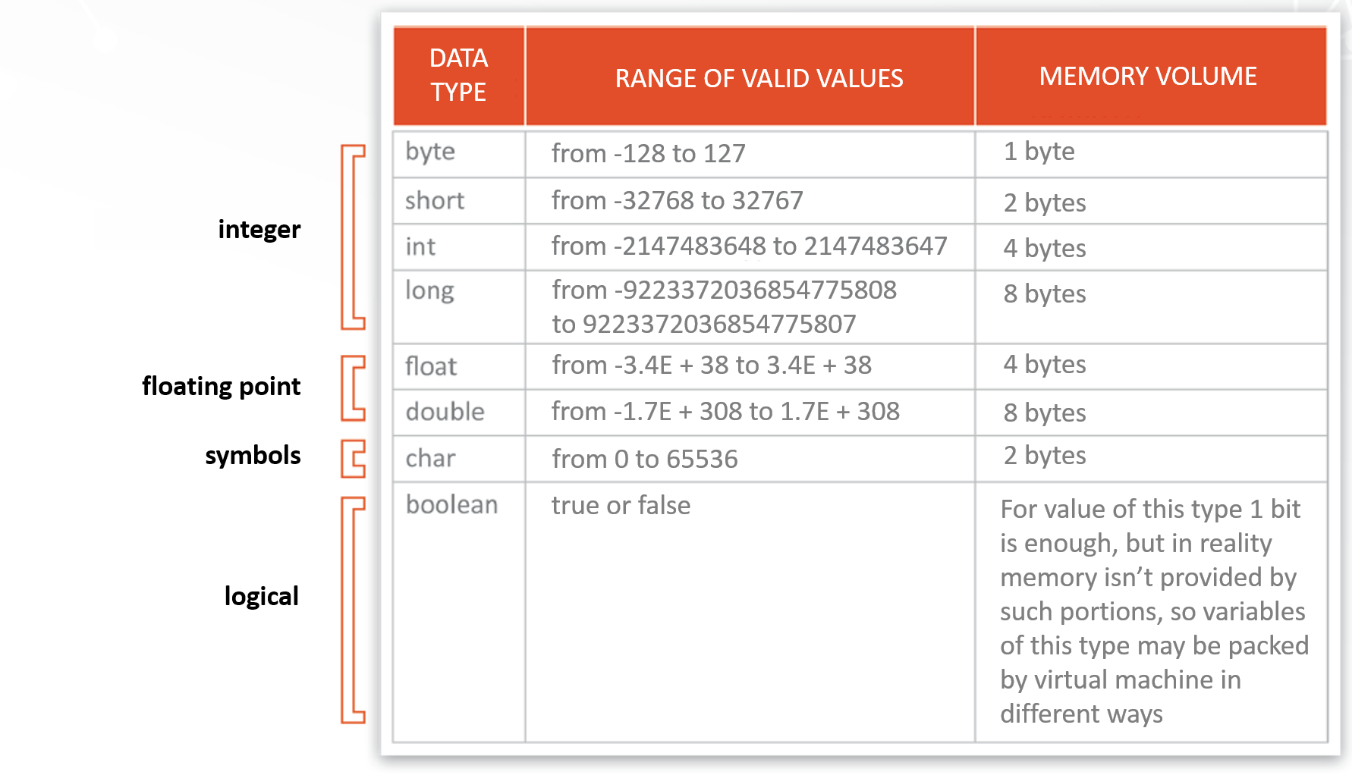
Interfaz de usuario gráfica

Descripción generada automáticamente

**PROGRAMACIÓN ESTRUCTURADA**

**CASTING ENTRE TIPOS DE DATOS**

Podemos hacer casting entre tipos de datos pero siempre y cuando sean compatibles entre sí.



Por ejemplo, podemos hacer asignar un valor Integer a un double por ejemplo, esto es porque “double” tiene mayor byte que Integer, por lo tanto podemos meter cualquier valor ahí, pero si ya un “double” lo queremos meter en un tipo Integer habrá que hacer un cast así:

int a = 5;

double b = 5.7;

a = (int) b;

Estamos casteando un double a un int para que “quepa” en el tipo de dato Integer, ahora, fijarse en los valores que le asignamos a las variables ya que si sobrepasa del limite y casteamos a un tipo de dato menor, puede haber **pérdida de información.**

**MÉTODOS DE LA CLASE**  **WRAPPER STRING MAS IMPORTANTES**

Es mas recomedable concatenar con concat() que el mas, esto para Strings.

El string es inmutable, es decir, su valor no cambia, una vez se crea el objeto este valor se mantiene inalterable. No importa que método utilicemos para concatenar, si no lo igualamos a una nueva variable, la variable String es inmutable.

Para concatenar Strings podemos usar el signo “+” o “concat()” o StringBuilder que es mucho mas rápido y eficiente al momento de concatenar grandes cantidades de cadenas.

Podemos validar un String con null, con lenght(), con isEmpty() o con isBlank() que también verifica el espacio en blanco, super útil en validaciones, IS BLANK DIPONIBLE DESDE JAVA 11, EN VERSIONES ANTERIORES DARIA ERROR.

**CLASE WRAPPER INTEGER**

La clase Wrapper Integer pertenece a la clase padre Number.

Solo podemos convertir a tipos de mayor bit, en dados caso lo hagamos con tipos de menor bit debemos tener cuidado de que el valor no sobrepase el valor máximo de bits de datos a convertir.

Importante destacar que las clases Wrapper o referencia son **objetos**, a pesar de que puedan almacenar el mismo valor, al momento de compararlos serán diferentes porque siempre serán diferentes objetos, a menos que igualemos ambos objetos.

**Autoboxing:** Es cuando “empaquetamos” o convertimos un tipo primitivo (int) a un tipo de referencia (Integer)

**Autounboxing:** Es cuando “desempaquetamos” o convertimos un tipo referencia (Integer) a un tipo primitivo (int)

**OPERADOR TERNARIO**

Su funcionamiento es como un IF-ELSE pero no podemos anexar otros procesos además de asignar un valor.

Estructura:

Variable = Condición ? La variable tomará este valor SI se cumple: La variable tomará este valor si NO se cumple;

Ejemplo:

Texto

Descripción generada automáticamente

**CLASE RANDOM**

Muy útil ya que nos genera valores aleatorios de forma mucho mas comoda, por ejemplo, con el tipo int.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

El método nextInt() nos genera un número aleatorio desde -2^32 hasta 2^32, rango muy extenso.

De forma similar con los otros tipos de datos numéricos, double, Long pero cuyos rangos serán mucho mayor o menor.

Podemos también pasarle parámetros a la clase nextInt(), así:



Esto generará número aleatorio entro 0 y 9, el 10 es exclusivo del rango.

Ahora, podemos crear un rango para que la clase Random nos genere números entre dicho rango, para ello es el siguiente código:



Esto nos genera números aleatorios entre 15 y 25, sumamos 15 al principio para que empiece a generar desde ese número y restamos 25-15=10 para que sea un aleatorio entre 0 y 9, así con la suma pues se cumple la condición.

**TIP**

El método **sort();** ordena o Array de manera alfabética o de manera ascendente.

**INSTACE OF**

Funciona para verificar si un objeto pertenece a su “familia”. Ej:

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza baja

Object es la clase PADRE de todas.

La clase NUMBER es la clase padre de todos los números.

**COMPARE TO**

Este método compara dos datos tipo String de forma lexicográficamente, es decir, a nivel de carácter.

Primero ejemplo, dos palabras iguales pero uno con la primera letra mayúscula y la otra minúscula.

Texto

Descripción generada automáticamente

En este caso “hola” es mayor lexicográficamente que “Hola” ya que tienen mayor prioridad las minúsculas que las mayúsculas.

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

Interfaz de usuario gráfica

Descripción generada automáticamente

En este otro caso el resultado es igual a cero ya que ambas palabras son exactamente iguales

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

Texto

Descripción generada automáticamente

En este otro caso el resultado es menor a cero ya que “beso” es menor lexicográficamente que “vaso” en orden alfabético “b” esta mucho antes (menor) que “v”.

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

Texto

Descripción generada automáticamente con confianza media

Este otro ejemplo a simple vista es engañoso, pero fijémonos en las minúsculas y mayúsculas, a pesar que “b” es menor que “v”, esta vez el resultado es positivo, o sea, la variable1 o “beso” es mayor que “Vaso” y es por la minúscula, las minúsculas siempre serán mayor que las mayúsculas independientemente de las letras.

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

Interfaz de usuario gráfica, Sitio web

Descripción generada automáticamente

Veamos este otro caso, ambas palabras casi iguales a excepción por la tilde en la primera letra, en este caso “ana” es menor que “ána”, tanto en minúsculas como en mayúsculas, las vocales acentuadas siempre serán mayor que las que no lo están.

Texto

Descripción generada automáticamente con confianza media

En este caso “ana” es mayor que “1”, las palabras, letras o vocales siempre serán mayores lexicográficamente que los números, sin importar que sean minúsculas o mayúsculas.

Estos fueron algunos ejemplos básicos para el uso de compareTo entre Strings.

**PROGRAMACIÓN ORIENTADA A OBJETOS**

Programación Orientada a Objetos, es un paradigma de la programación en el cual buscamos encapsular y abstraer las diferentes funcionalidades de nuestro código de forma que podamos reutilizarlo y sea mucho más entendible para cualquier programador.

Class: Palabra reservada de Java que sirve para representar un tipo, una entidad, un objeto, “esto representa algo”

Un objeto es una instancia de una clase donde podemos definir valores para sus atributos.

Para crear una instancia necesitamos usar la palabra clave new

Una clase es una especificación de un tipo, que define atributos y comportamientos.

Instancia: es una representación de un objeto, un objeto tiene muchas instancias, por ejemplo, tenemos una clase “cuenta”, este objeto puede tener muchas instancias, es decir, muchas cuentas, no solamente una.

Cuando creamos una nueva instancia, solemos utilizar la palabra reservada “new” esto es porque le estamos diciendo a Java que cree en la memoria o que reserve un espacio en la memoria para un nuevo objeto, muy independiente de los demás objetos.

Una referencia siempre va a apuntar a un espacio de la memoria asignada con “new”, si igualamos dos instancias entonces la referencia apuntara a la misma memoria de la instancia creada con new y compartirán las modificaciones de sus valores.

Los constructores se utilizan para inicializar los atributos. El constructor es lo que sucede AL MOMENTO de crear un objeto por medio de una instancia.

Los atributos STATIC pertenecen a la clase y NO a la instancia, es por ello que al llamarlos podemos colocal nombreClase.variableStatic

Se le llama sobrecarga de funciones o métodos a la creación de varias funciones dentro de nuestro programa, que tienen el mismo nombre, pero sus argumentos son distintos y realizan distinta acción.

El término encapsulamiento en Java, consiste en ocultar atributos de un objeto de manera que solo se pueda cambiar mediante operaciones definidas en ese objeto. Está estrechamente relacionado con la visibilidad. BASICAMENTE es que, por ejemplo, un metodo cuando es llamado en la clase MAIN no sabe como funcionan los demás metodos, solo se sabe que se hace un proceso y nada mas, todos los procesos y variables definidos dentro de estos metodos no se puedeb saber, a esto se le llama principio de encapsulamiento en Java. OTRA FORMA DE VERLO ES en el metodo main solo llamamos el metodo pero no sabemos su contenido ni como funciona, pues en pocas palabras esto es el principio de ocultamiento en Java.

Otro principio importante es que LOS ATRIBUTOS (O variables) dentro de una clase, SIEMPRE deben ser PRIVADOS y la única forma de acceder a ellos sea por medio de GET y SET para lectura y escritura respectivamente.

Ahora entra el principio de ocultación, que es cuando los atributos están privados dentro de la clase y no se pueden acceder a ellos directamente ya que tienen PRIVATE en sus variables.

POLIMORFISMO EN JAVA, A grandes rasgos, el polimorfismo permite que nombres dos acciones del mismo modo dentro de tu código, pero que cada una de ellas acepte diferentes parámetros. Esto facilita que identifiques objetos nominalmente, pero que cada uno de ellos desempeñe una función en particular.

En las clases, siempre primero van los ATRIBUTOS, luego los CONSTRUCTORES, luego los GETTERS Y SETTERS, luego los MÉTODOS.

Un ejemplo de principio de polimorfismo es el método EQUALS, si bien sabemos funciona para comparar objetos de forma general, PERO, si en una clase hacemos un Orverride, es decir, una sobre escritura del método, lo modificamos y hacemos que haga una tarea diferente, ahí estamos aplicando polimorfismo, dos objetos, dos métodos en este casi, de igual nombre pero con comportamientos diferentes.

Atributos estáticos (static) no son como los atributos normales de una clase, es un atributo estatico que es compartido por todos los objetos que se puedan instanciar de una clase, es por ello que, la forma de llamar un atributo estatico es con el nombre de la clase y no con el nombre de la instancia. Ej

--> Automovil auto = new Automovil();

Los atributos estáticos se llamaran con Automovil, y no con auto, ya que podriamos decir que es una "variable global" que es usada por TODOS. Igualmente, debe ser privada como uno de los principios y ser llamada con GETTER y SETTER.

En los métodos estáticos no podemos usar los tributos normales de una clase, ni el "this" para llamar a otros atributos, solamente acepta atributos estáticos.

Los atributos FINAL (final) son constantes, es decir, tiene un valor que no se pueden cambiar, en la mayoría de veces es un valor stático y público.

El nombre de las constantes FINAL se escriben todas en mayúsculas y separadas por GUIONES

Desde el JDK 13 o 14 hay un nuevo Switch case, ya no se usan los breaks y en lugar de dos puntos se usan las flechas "->"

Los ENUM, sirven para tener una coleccion de constantes finales, por ejemplo, los dias de la semana, los nombres de los planetas, ETC. Muy utiles si queremos tener un orden en constantes. De igual manera cada una de estas constantes peuden tener sus propias caracteristicas, como por ejemplo la masa del planeta, color del plantea, gravedad, etc. Estos atributos adicionales se colocan creando un constructor en el enumerador y posteriormente colocar las caracteristicas deseadas

GRANULARIDAD: No concentrar todo en una sola clase, dividir objetos por varias clases para tener un mejor orden, el codigo sera mucho mas escalable y mas facil de mantener

String[] args. antiguamente se utilizaba cuando se trabajaba en java desde la consola, entonces se usaba para pasarle los parametros al ejectuar el programa

El estándar para nombrar paquetes en Java es:

Com.dominioempresa.modulo.proyectoX

Los package sirven, sirven para compartir clases o aislarlas, además que, sirven para mantener ordenadas todas la clases, físicamente son directorios, agrupa las clases, base de datos -> persistence, mientras mas segmentadas mejor, también para evitar que se repitan nombres

Excepciones.

Uso de finally (No es muy utilizado)

Siempre va a correr haya o no haya error

**SOBRECARGA DE MÉTODOS**

La sobrecarga se refiere a la posibilidad de tener dos o mas métodos con el mismo nombre pero con diferente funcionalidad.

En sobrecarga de métodos podríamos utilizar un nuevo tema al que se le conoce como

**VARARGS**

Varargs es un arreglo de n tipos de datos, ejemplo:

Texto

Descripción generada automáticamente

Si observamos en este método “suma” le pasamos como argumento “int… args” lo cual significa que al llamar el método en otra clase, le podemos pasar n cantidad de argumentos, muy útil.

Igualmente se puede combinar con otros tipos. Ej:

Texto

Descripción generada automáticamente

Sumar un número tipo float y n números de tipo int.

**HERENCIA**

Es una relación de parentesco entre dos clases, donde una clase es la padre y otra clase es la hija.

Es el mecanismo de la programación orientada a objetos para implementar relaciones de jerarquías de clases.

Una subclase hereda el estado y comportamiento de todos sus ancestros.

La palabra reservada para usar la herencia entre clases es “extends”, ej.

Texto

Descripción generada automáticamente

Una **SUPERCLASE** se refiere a la clase que es el ancestro mas directo, así como todas las clases ascendentes.

En UML es una flecha continua que va desde la clase hija hasta la clase padre, similar a la relación de asociación, pero con la diferencia que termina con una punta en forma de triángulo

Hay DOS formas de herencia:

GENERALIZACIÓN: Es cuando tenemos varias clases con atributos o métodos repetidos, lo que hacemos es refactorizar esta clases, todo lo que tienen en común estas clases lo mandamos a una clase mas general para que sea mucho mejor.

ESPECIALIZACIÓN: Es cuando ya tenemos todas nuestras clases funcionando bien, pero necesitamos tener otra clase idéntica o casi idéntica a la clase padre, similar pero con algunas cosas deferentes, a esto llamamos herencia por especialización.

Un ejemplo de especialización sería el siguiente:

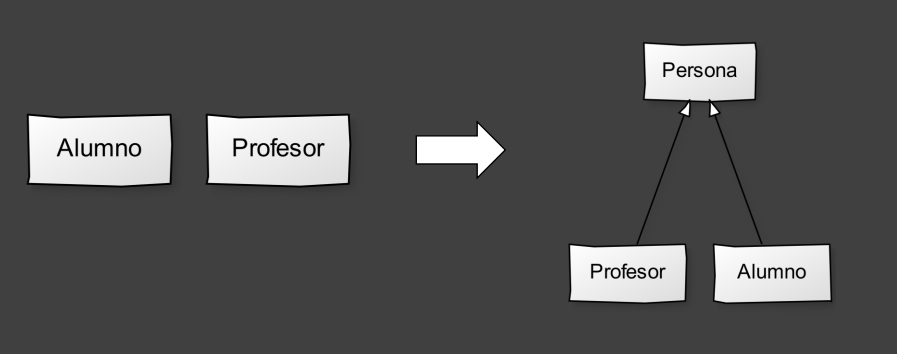
Diagrama

Descripción generada automáticamente

A partir de la clase Persona extendemos a una clase Alumno, muy similar a Persona pero ahora Alumno comparte atributos y métodos de persona.

Aquí entra en juego un principio de POO conocido como OPEN-CLOSE. Abierta a extensiones pero cerrada a modificaciones, es decir, la clase padre Persona no conviene modificarla, mejor extendemos una clase hija que herede sus características, he aquí el principio.

En cambio, un ejemplo de Generalización sería el siguiente:



Creamos una clase padre apartir de Alumno y Profesor.

Todo atributo **protected** en la clase **padre** es visible y se puede acceder tanto en la misma clase padre como en toda su jerarquía, en toda su descendencia. Este tipo es similar a dejar la variable en “default” es decir, solamente con el nombre de la variable se pueden acceder SIEMPRE Y CUANDO LAS MISMAS CLASES ESTÉN DENTRO DEL MISMO PACKAGE, si una clase que se encuentra en otro paquete desea acceder a un atributo **protected** que está en otro paquete no podrá.

Las clases hijas puede usar directamente los atributos de tipo **protected** sin necesidad de un getter.

**CLASES ABSTRACTAS**

Está hecha para la herencia, representa un concepto genérico o conceptual y como tal, no debe ser ejemplarizada.

Es cuando tenemos una jerarquía de clases en todas estas clases tenemos un comportamiento (método) que se sobrecarga de manera distintas en cada una de ellas.

Una clase abstracta está diseñada para ser una superclase, es decir, una CLASE PADRE y no puede ejemplarizarse, es decir, NO PUEDE INSTANCIARSE.

Cuando una clase hereda de una clase padre que es abstracta está obligada a implementar sus métodos, ya que tienen una especie de CONTRATO.

La herencia con clases abstractas representan ALGO abstracto y puede reunir varias implementaciones concretas que realizan el mismo comportamiento, es decir, los mismos métodos, pero cada uno de forma distinta, por ejemplo:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Todas las figuras geométricas tienen área, perímetro y se pueden dibujar, pero hay MUCHAS figuras geométricas, el área, perímetro y el dibujo de las mismas será diferente para cada uno, es decir, utilizan el mismo método pero lo implementan de diferente forma.

No pueden haber métodos abstractos en una clase que no es abstracta PERO SI pueden haber métodos que no sean abstractos en una clase abstracta.

Los símbolos <> en una List se conocen como generic de java

Diferencia entre **clases abstractas e Interfaces** las clases abstractas tienen una relación “familiar” con sus demás clases que hereden comportamientos de esta. Ej: Un helicóptero que VUELA, la clase abstracta será de vehículos que contiene un comportamiento de VOLAR y sus clases hijas heredarán VOLAR y serán tipos de vehículos voladoras (Avion, avioneta, Jet) etc.

En cambio una Interfaz es algo mucho mas genérico que las clases abstractas, por ejemplo, no solo los helicópteros tienen el comportamiento de volar, si no que también insectos voladores, aves… etc. Entonces Todo esto se engloba en un solo comportamiento que es VOLAR, no tiene ningún “vinculo familiar” con las clases que IMPLEMENTEN la Interfaz.

INTERFACES

Son una colección de métodos abstractos con propiedades (Atributos) CONSTANTES.

Todos los métodos de una interface son abstractos, no es necesario colocar la palabra reservada “abstract” en el método.

Con las interfaces puede existir herencia multiple, cosa que con las clases abstractas no hay.

Todas las clases hijas que IMPLEMENTEN las interfaces, tienen que IMPLEMENTAR DE FORMA OBLIGATORIA todos sus métodos abstractos.

En las interfaces podemos tener métodos “default” que hacen que las clases que implementen la interfaz NO SE VEAN OBLIGADAS a implementarlos, pero estos métodos deben llevar cuerpo.

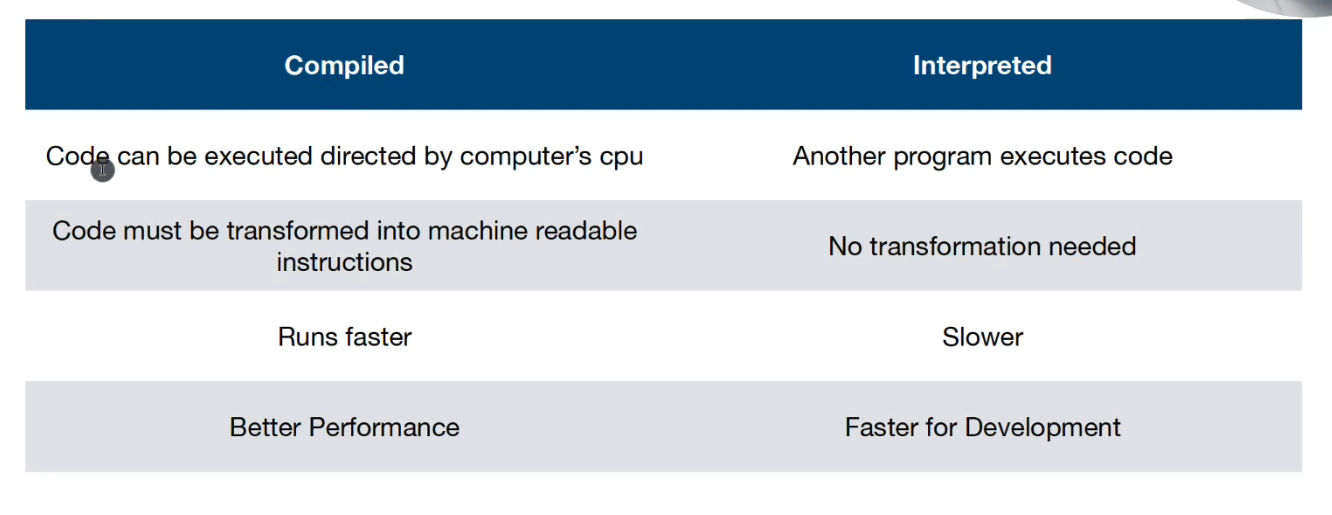
En las interfaces podemos tener métodos “estáticos”

Podemos centralizar todas las interfaces, en una sola interfaz podemos centralizar todas las que queramos con la palabra reservada “extends” y AQUÍ SI podemos COLOCAR HERENCIA MULTIPLE

**DIFERENCIAS ENTRE JAVA Y JAVASCRIPT**

Java es un lenguaje de compilación e interpretación, es decir, su código se compila código maquina para que la maquina pueda leerlo. Además que Java se usa principalmente para Backend.

En cambio, Javascript además que se utiliza para web… se puede utilizar para backend usando framworks como Node js, etc. Pero además, es un lenguaje interpretado, es decir no se compila, si no que se ejecuta directamente.



PRO TIP PARA USAR DATE TANTO DE UTIL COMO DE SQL EN UN MISMO ARCHIVO

Texto

Descripción generada automáticamente

PRO TIP PARA IMPRIMIR ARREGLO DE STRING SIN LOOP Y CON FORMA

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

En casos reales, no se aconseja trabajar con el tipo primitivo “double” para datos como números, ya que su potencia de 10 arroja valores con muchos decimales. Ej. Probar a restar 0.03-0.02 ambos tipo double.

Siempre que trabajemos con DINERO REAL en una aplicación Java es recomendable utilizar la clase BigDecimal.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

DIFERENCES BETWEEN CHECKED EXCEPTIONS AND UNCHECKED EXCEPTIONS

Unchecked exceptions are when we don’t deal with the exceptions in obligatory form, for example, in the case when we want to convert a String to an Integer our application throw a runtime exception, its optional to use exception in this case, the code works fine without the exception. But with the checked exceptions we must deal with them obligatory, for example when we want to create a new file we most use exceptions to create the file because it’s a checked exception.