



MANUAL BACK-END

FACUNDO ACUÑA EDGAR

ÍNDICE

ÍNDICE	1
1 Introducción	4
1.1.1 Introducción a Backend	4
1.1.2 ¿Qué funciones se gestionan desde el Back-End?	5
1.1.3 JAVA	6
Máquina virtual	6
¿Cuáles son las características de Java ?	10
¿Cuál es la importancia de Java en la programación?	10
1.2 Datos y expresiones	11
1.2.2 Sintaxis, Palabras reservadas y Tipos primitivos	11
Identificadores	12
Tipos de datos	13
Datos primitivos	13
Tipos de dato referencia	14
1.2.3 Variables y constantes	14
Variables	14
Variables miembro	15
Variables locales	15
Constantes	15
1.2.4 Operadores y expresiones	16
Expresiones	16
Operadores	16
Operadores aritméticos	16
Operadores de asignación	16
Operadores de unaryos	17
Operadores de comparación/relación	17
Operadores bit a bit	18
Operadores lógicos	18
Operadores de concatenación	18
1.2.5 Conversión de tipos	19
Conversión implícita	19
Conversión explícita	19
Actividad	21
1.2.6 Uso de funciones matemáticas	22
Acceso a paquetes o importación de paquetes	22
Función Math	22
1.2.7 Declaración de objetos	25
Atributos	25
Métodos	25

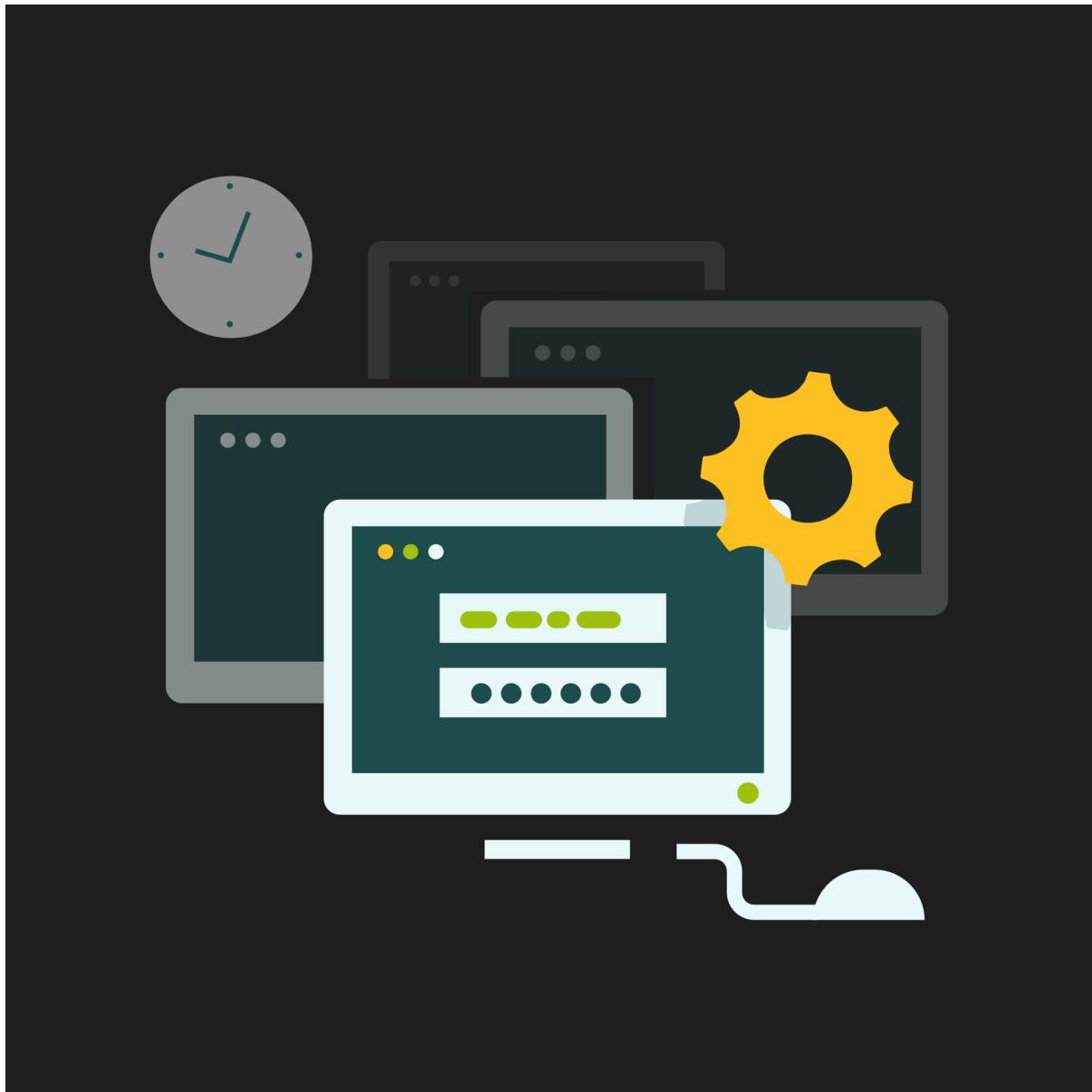
Declaración de un objeto	25
Instanciación de un objeto	26
Utilización	26
1.2.8 Strings	27
Convertir un número en un String o cadena	29
1.2.9 Atributos y métodos estáticos	29
1.2.10 Ejercicio: Datos y expresiones	32
Actividad	33
1.3 Clases, referencias y objetos	34
1.3.1 ¿Qué son?	34
1.3.2 Creación e inicialización de objetos	35
1.3.3 La clase object	36
1.3.4 Referencias y objetos	38
1.3.5 Comparación de objetos.	38
1.3.6 Recolector de basura	38
1.3.7 Métodos y campos de clase	39
¿Cómo se escribe un método?	39
Retornado de un método	40
Formas de devolución	40
Devolviendo un valor	41
1.3.8 Composición de objetos	42
1.3.9 Herencia	45
Constructores y herencia	48
Constructores	48
Constructores en clase heredadas	49
1.3.10 Polimorfismo	50
1.3.11 Ejercicios: Clases, referencias y Objetos	53
1.4 Creación de clases en JAVA	56
1.4.1 ¿Qué son los métodos?	56
1.4.2 ¿Cómo se escribe un método?	56
1.4.3 Argumentos de un método	60
1.4.4 Sobrecarga de métodos	60
1.4.5 ArrayList, List y Mapas	62
Arrays	62
List	63
ArrayList	63
Maps	67
Clases que implementan la interfaz Map	67
1.4.6 Ejercicio: Métodos y Listas	69
Actividad	71

Cuestionario unidad 1 backend	72
Referencias	75

1 Introducción

1.1.1 Introducción a Backend

Backend, es un término muy utilizado para referirse al área lógica de toda una página web. Con eso nos referimos a la arquitectura interna del sitio, que asegura que todos los elementos se desarrollen de una forma correcta.



Este no es visible para los ojos del usuario y no incluye ningún elemento gráfico.

El backend se desarrolla por un programador, y se construye con el código interno de la página, esta área se encarga de la funcionalidad del sitio, de la seguridad y de la optimización de recursos.

Los lenguajes de programación que más se utilizan del lado del backend son: Java, Python, PHP.

1.1.2 ¿Qué funciones se gestionan desde el Back-End?

- Desde el Back-End se llevan a cabo todas las funciones que hagan más simple el proceso de desarrollo.
- Las acciones de lógica.
- Las conexiones con las bases de datos.
- Se usan las librerías del servidor web, como compresión de imágenes web o implementar temas de caché
- También se mantiene la seguridad de los sitios web.
- Se pueden optimizar los recursos para que las páginas resulten más ligeras



- El backend se encarga de la gestión de datos del proyecto web principalmente con el almacenamiento, entrega y organización de datos.

- El desarrollo de un sitio web debe tener una parte del lado del servidor, esta parte es la denominada backend y a veces un sitio web o aplicación desarrollada se renderiza en el lado del servidor. La conexión que hay entre el backend y el frontend, es la llamada de procesos o servicios por medio de peticiones, que ayudan a almacenar información en la base de datos, búsquedas en la base de datos, gestión de archivos, entre otras funciones.
- La mayoría de las veces el backend está oculto a la vista del usuario, sin embargo, es el encargado de darle instrucciones a las funcionalidades de los elementos del frontend. De esta manera los usuarios interactúan con el backend a través de una interfaz proporcionada por el frontend sin la necesidad de ver el funcionamiento del backend.
- El desarrollo del backend utiliza lenguajes como PHP, Python, C++, Ruby y Java.

1.1.3 JAVA

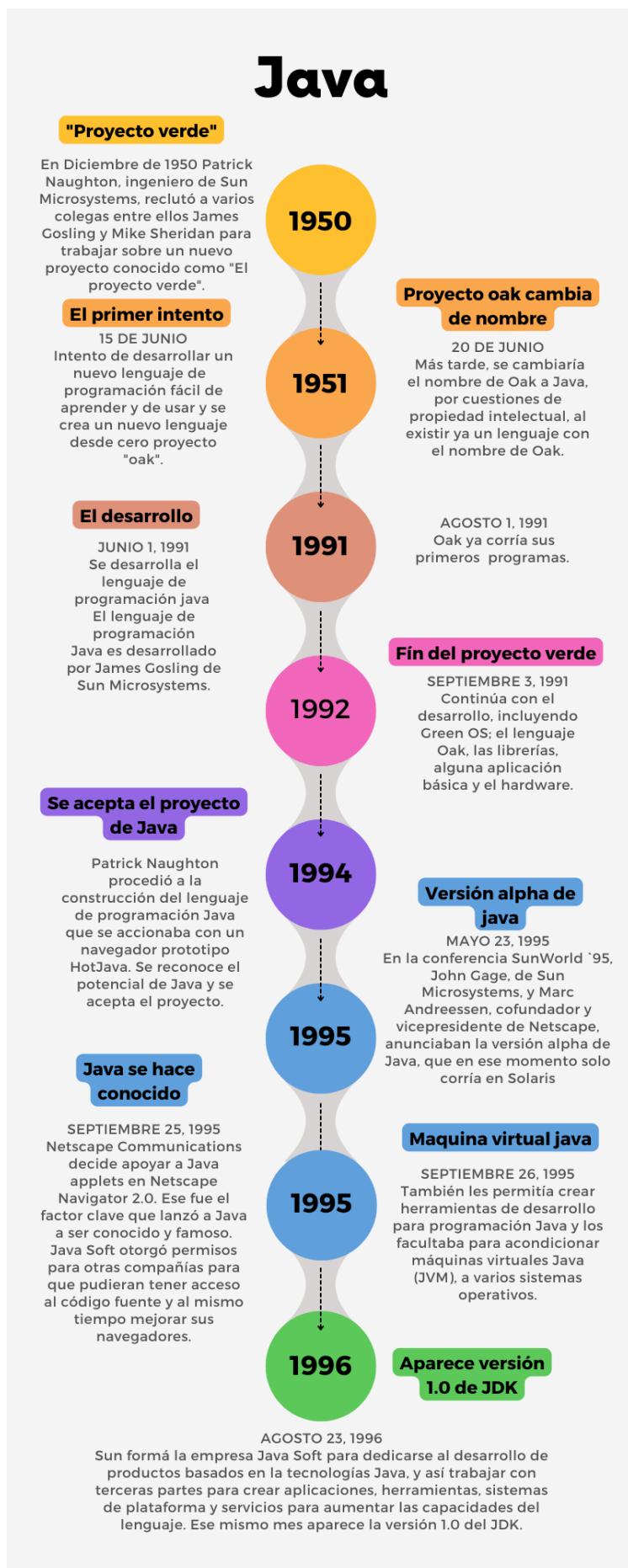
Java es un lenguaje de programación de propósito general creado por el desarrollador James Gosling en la empresa Sun Microsystems en 1995 la cual fue adquirida posteriormente por la compañía Oracle en 2010, su sintaxis se deriva de C y C++, además, está enfocado en un paradigma de programación orientada a objetos. Java permite el desarrollo de aplicaciones en diversas áreas, como seguridad, animación, acceso a bases de datos, aplicaciones cliente-servidor, interfaces gráficas, páginas Web interactivas y desarrollo de aplicaciones móviles, entre otras.

El objetivo de Java es permitir que los programadores escriban el programa una vez y este pueda ser ejecutado en cualquier otro dispositivo, este concepto de programación es conocido como WORA o “write once, run anywhere”, por lo que el código que es ejecutado en una plataforma no tiene que ser nuevamente compilado para correr en otra plataforma.

Máquina virtual

Cuando se compila una aplicación Java se genera una clase Java que puede ejecutarse en cualquier máquina virtual Java (*JVM Java Virtual Machine*) incorporada a Java sin importar el sistema operativo o arquitectura de la computadora, en otras palabras la máquina virtual de Java permite ejecutar el código de este lenguaje en cualquier sistema que incorpore su propia máquina virtual.

Java



Java

JDK 1.1

FEBRERO 19, 1997
 - Una operación intensiva del modelo de eventos AWT (Abstract Windowing Toolkit)
 - Clases internas (inner class)
 - JavaBeans
 - JDBC (Java Database Connectivity), para la integración de bases de datos
 - RMI (Remote Method Invocation)

1997

J2SE 1.2

DICIEMBRE 8, 1998
 Ésta y las siguientes versiones fueron recogidas bajo la denominación Java 2 y el nombre "J2SE", reemplazó a JDK para distinguir la plataforma base de J2EE (Java 2 Platform, Enterprise Edition) y J2ME (Java 2 Plataforma, Edición Micro).

1998

J2SE 1.3

MAYO 8, 2000
 - La inclusión de la máquina virtual de HotSpot JVM.
 - RMI fue cambiado para que se basara en CORBA
 - JavaSound
 - Se incluyó el Java Naming and Directory Interface (JNDI) en el paquete de bibliotecas principales.
 - Java Platform Debugger Architecture (JPDA)

2000

J2SE 1.4

FEBRERO 6, 2002
 Este fue el primer lanzamiento de la plataforma Java desarrollado bajo el Proceso de la Comunidad Java como JSR 59

2002

J2SE 5.0

SEPTIEMBRE 30, 2004
 (Originalmente numerado 1.5, esta notación aún es usada internamente)
 Desarrollado bajo JSR 176

2004

J2SE 6

DICIEMBRE 11, 2006
 Estuvo en desarrollo bajo la JSR 270. En esta versión, Sun cambió el nombre "J2SE" por Java SE y eliminó el ".0" del número de versión.

2006

J2SE 7

JULIO 1, 2011
 - Soporte para XML dentro del propio lenguaje.
 - Un nuevo concepto de superpaquete. Soporte para cierres.
 - Introducción de anotaciones estándar para detectar fallas en el software.

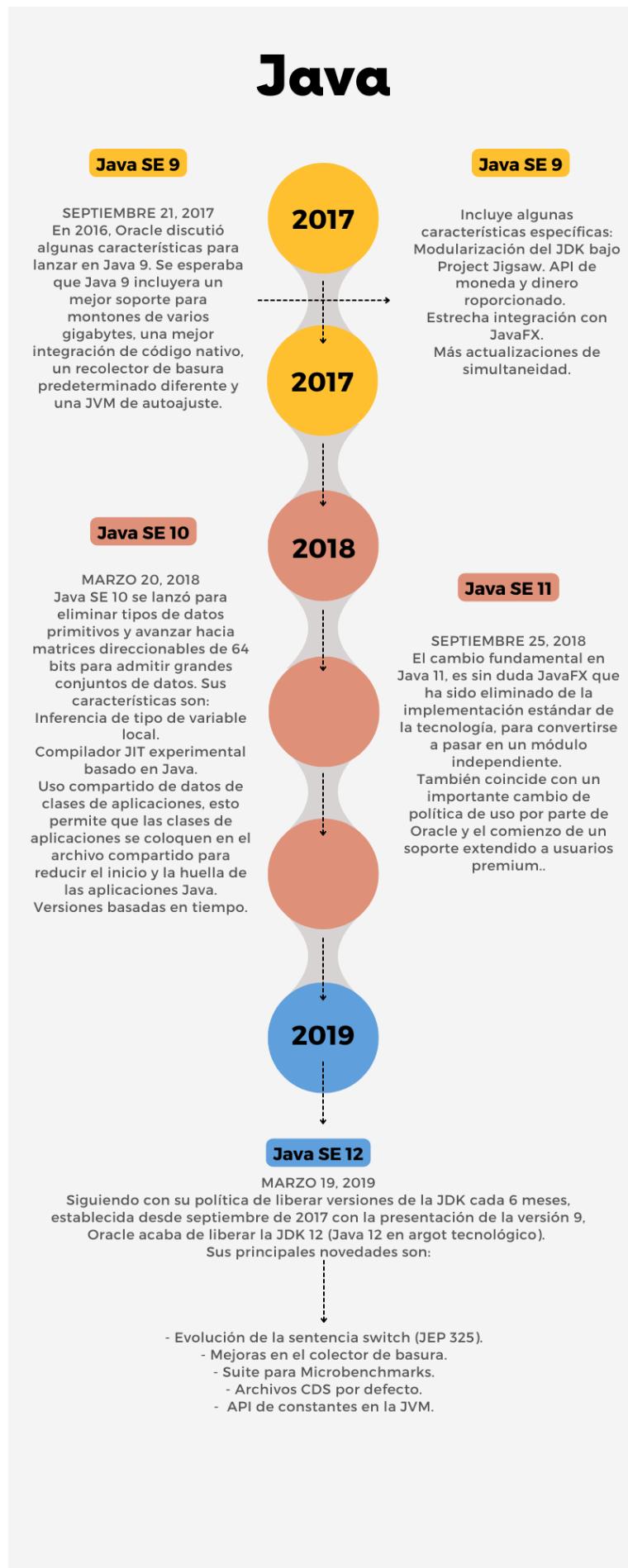
2011

J2SE 8

MARZO 1, 2014
 - Incorpora de forma completa la librería JavaFX.
 - Diferentes mejoras en seguridad.
 - Diferentes mejoras en concurrencia.
 - Añade funcionalidad para programación funcional mediante expresiones Lambda.
 - Mejora la integración de JavaScript.
 - Nuevas API para manejo de fechas y tiempo (date - time).

2014

Java



¿Cuáles son las características de Java ?

Una de sus principales características es la creación de módulos reutilizables, que funcionan sin la necesidad de conocer su estructura interna.

Esto permite al usuario añadir nuevos módulos, además de obtener programas independientes de la plataforma en la cual fueron desarrollados, gracias a la implementación de la llamada Máquina Virtual de Java (JVM).

Otras característica principales de Java son:

- Es simple: java ofrece un lenguaje potente que se deriva de C y C++ haciéndolo más sencillo.
- Orientado a objetos: El paradigma orientado a objetos(OO) es uno de los más populares ya que nos permite diseñar el software de forma que los distintos tipos de datos estén unidos a sus operaciones.
- Es distribuido: Java proporciona herramientas para que los programas puedan ser distribuidos
- Independiente a la plataforma: Esto significa que programas escritos en el lenguaje Java pueden ejecutarse en cualquier tipo de hardware, lo que lo hace portable.

¿Cuál es la importancia de Java en la programación?

Java es muy importante en el ámbito web y aplicaciones móviles debido a que es multiplataforma, lo que significa que se puede ejecutar en casi cualquier sistema operativo gracias a su máquina virtual.

Como consecuencia, Java tiene gran presencia en múltiples desarrollos de software, desde aplicaciones web, de consola o interfaz gráfica. Java es el lenguaje de programación que se utiliza para el desarrollo nativo en Android aumentando la demanda de su uso.



1.2 Datos y expresiones

1.2.2 Sintaxis, Palabras reservadas y Tipos primitivos

Los elementos que construyen un programa son: palabras reservadas, identificadores, caracteres especiales, expresiones e instrucciones.

La sintaxis de un lenguaje de programación es un conjunto de reglas que definen la forma de escribir código.

Para poder generar la sintaxis de un lenguaje de programación se necesitan palabras reservadas. Las palabras reservadas son palabras que tienen un significado específico en el lenguaje y ninguna puede ser utilizada para nombrar variables o cualquier otro identificador.

Java tiene las siguientes palabras reservadas:

- Abstract
- Continue
- For
- New
- Switch
- Assert
- Default
- Goto
- package
- synchronized
- Boolean
- do
- if
- private
- this
- break
- double
- implements
- protected
- throw
- byte
- else
- import
- public
- case
- enum
- instanceof
- return
- transient
- catch
- extends
- int

- short
- try
- char
- final
- interface
- static
- void
- class
- finally
- long
- volatile
- const
- float
- native
- super
- while

Las siguientes palabras son reservadas pero no se utilizan,

- const
- goto

Además, Java reserva 3 palabras más como valores definidos por el lenguaje:

- false
- null
- true

Identificadores

Un identificador no es más que el nombre que se le da a las variables, métodos, clases e interfaces. El identificador puede tener caracteres alfabéticos, numéricos e incluso el símbolo de dólar (\$) y el de guión bajo (_), además el número de caracteres puede ser ilimitado.

Para crear un identificador no hay reglas específicas, sin embargo hay convenios denominados usualmente buenas prácticas los cuales nos indican cómo declarar un identificador. Los nombres de identificadores usualmente utilizados constan de una concatenación de dos o más palabras separadas por un guión bajo, o por medio del uso de letra mayúscula al principio de cada palabra comúnmente llamado “notación camello (“*camel case*”), exceptuando los siguientes casos:

- Los identificadores que comienzan con una letra mayúscula son únicamente utilizados para clase e interfaces.
- Los identificadores que solo están escritos en mayúsculas representan constantes.
- Los nombres de las variables y métodos inician con letras minúsculas.

Por ejemplo:

- numero_naranjas
- numeroNaranjas
- Persona

Tipos de datos

Los tipos de datos representan al tipo de información con la que estamos trabajando, también se puede definir como el rango de valores que puede tomar una variable a lo largo de la ejecución de un programa. A partir de los tipos de datos se determina el rango o grupo de valores que definen las operaciones que se pueden ejercer sobre este grupo o rango de valores.

En java existen dos tipos de datos, primitivos y de referencia.

Datos primitivos

Los tipos de datos primitivos son valores que representan el nivel más bajo de la implementación del lenguaje.

En Java son datos que no son objetos y tampoco cuentan con métodos, además todos los datos primitivos son inmutables, es decir, no se pueden modificar.

También son un tipo de dato heredado de otros lenguajes de programación no orientada a objetos como el lenguaje de programación C.

Una de las características de este tipo de datos es que no necesitan ser declarados como clases con más complejidad.

Por ejemplo:

Para crear un objeto de clase persona se declara de la siguiente forma:

Persona = new Persona;

Mientras que para crear un dato primitivo, como un entero, se declara de la siguiente forma

int a;

A continuación se muestra una tabla con los tipos primitivos con los cuales cuenta Java, y sus valores por defecto que adquieren las variables en caso de no inicializarlas.

Dato primitivo	Valor por defecto
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
char	'u0000'
String (o cualquier objeto)/ tipos referencia	null
boolean	false

La clase String no pertenece a los datos de tipo primitivos, pero tiene la particularidad de ser declarada como si lo fuera. Esta clase nos permite manejar cadenas de texto separadas por comillas dobles.

Por ejemplo:

```
String persona = "Paula Cruz" ;
```

Tipos de dato referencia

Este tipo de datos representan las referencias a objetos, guardando las direcciones de memoria y no el valor, esta área de memoria es la representación de un objeto. Cuando el valor es “null” de este tipo de dato quiere decir que no se refiere a un objeto real es decir de referencia nula.

1.2.3 Variables y constantes

Variables

Las variables son las unidades básicas de almacenamiento en Java. Una variable es el nombre que se le da a un pedazo de memoria donde se guarda un dato. Se define por la combinación de un tipo de dato, un identificador y un inicializador el cual puede ser opcional, en caso de no inicializar la variable toma valores por defecto del tipo de dato, además, el valor puede cambiarla a lo largo de toda la ejecución del programa.

Para poder utilizar una variable primero debe ser declarada y estas variables pueden ser de tipo primitivo o de referencia. Se declaran de la siguiente manera:

tipoDeDato nombreVariable;

Por ejemplo:

```
int variableUno;  
float metros;  
char letra;  
String nombre;
```

Como se mencionó anteriormente , una variable de tipo referencia indica el área de memoria donde está guardado un objeto. Al declarar una variable de tipo referencia esta no se encuentra apuntando a ningún objeto por lo que es necesario crear un objeto con el operador “new”, el cual es el encargado de reservar espacio en la memoria para ese objeto.

Ejemplo:

```
Persona p1 = new Persona();
```

Variables miembro

Las variables miembro se definen dentro de una clase, estas deben estar fuera de cualquier método

Variables locales

A diferencia de las variables miembro , las variables locales si se definen dentro de un método o en términos más generales dentro de un bloque contenido entre llaves { }. Una de las características principales de estas variables es que una vez que se inicializa un bloque y se concluye la ejecución de este, la variable local se destruye.

Constantes

Una constante es una variable en donde su valor no puede ser modificado. A diferencia de otros lenguajes que utilizan la palabra reservada “const” para definir una constante, en Java, se utiliza la palabra “final”. Se declaran de la siguiente manera:

final tipoDeDato nombreConstante = valor;

Ejemplo:

```
final double PI = 3.1416;
```

1.2.4 Operadores y expresiones

Expresiones

Una expresión es una combinación de variables, operadores y llamadas de métodos construida de acuerdo a la sintaxis del lenguaje que devuelve un valor.

Operadores

Los operadores son palabras claves del lenguaje que permiten la ejecución de operaciones en el contenido de ciertos elementos, en general variables, constantes, valores literales, o retornos de funciones. Permitiendo la combinación de expresiones en expresiones más complejas.

Java tiene una amplia colección de operadores para poder manipular datos como los unarios, asignación, aritméticos, lógicos (boolean) y relacionales.

Operadores aritméticos

Los operadores aritméticos se utilizan para manipular expresiones matemáticas, son operadores binarios, es decir, siempre se requieren de dos operadores que realizan las operaciones aritméticas usuales.

Operador	Operación efectuada	Ejemplo	Resultado
+	Suma	6+4	10
-	Sustracción	12-6	6
*	Multiplicación	3*4	12
/	División	25/3	8.3333333333
%	Módulo (resto entero de la división)	25 % 3	1

Operadores de asignación

El operador de asignación tiene como única tarea la de asignar un valor a una variable. El operador que define una asignación en Java es el operador igual “ = ”.

variable = expresión

Java también tiene otros operadores de asignación, no son diferentes al operador “ = ” pero ayudan a abreviar las sentencias de asignación y que ayudan a realizar operaciones acumulativas sobre una variable.

Se utiliza el mismo operador sea cual sea el tipo de la variable (numérico, cadena de caracteres...). Además, se puede combinar con un operador aritmético, lógico o binario.

Operador	Expresión	Expresión equivalente
<code>+=</code>	<code>op1 += op2</code>	<code>op1 = op1 + op2</code>
<code>-=</code>	<code>op1 -= op2</code>	<code>op1 = op1 - op2</code>
<code>*=</code>	<code>op1 *= op2</code>	<code>op1 = op1 * op2</code>
<code>/=</code>	<code>op1 /= op2</code>	<code>op1 = op1 / op2</code>
<code>%=</code>	<code>op1 %= op2</code>	<code>op1 = op1 % op2</code>

Operadores de unarios

Los operadores unarios son prácticamente operadores aritméticos, con la diferencia que realizan una operación sobre un mismo operando.

Operador	Acción
<code>-</code>	Valor negativo
<code>~</code>	Complemento a uno
<code>++</code>	Incremento
<code>--</code>	Decremento
<code>!</code>	Negación

Por ejemplo:

```
a++;  
b- -;
```

Sólo se puede utilizar el operador `!` (exclamación) en variables de tipo boolean o en expresiones que producen un tipo boolean (comparación).

Operadores de comparación/relación

Los **operadores de comparación o relacionales** son operadores que determinan la relación que existe entre un operador A con otro operador B. Dicho de otra manera, operadores que comparan dos valores. Este tipo de operadores retornan un resultado booleano, si la comparación es correcta el resultado obtiene un valor de “true” (verdadero), de lo contrario obtiene un valor de “false” (falso).

Operador	Operación efectuada	Ejemplo	Resultado
<code>==</code>	Igualdad	<code>2 == 5</code>	false
<code>!=</code>	Desigualdad	<code>2 != 5</code>	true
<code><</code>	Inferior a	<code>2 < 5</code>	true
<code>></code>	Superior a	<code>2 > 5</code>	false
<code><=</code>	Inferior o igual a	<code>2 <= 5</code>	true
<code>>=</code>	Superior o igual a	<code>2 >= 5</code>	false
<code>instanceof</code>	Comparación del tipo de la variable con el tipo indicado	<code>O1 instanceof Cliente</code>	True si la variable O1 hace referencia a un objeto creado a partir de la clase Cliente o de una subclase

Operadores bit a bit

Otra de las características que dispone Java es que cuenta con un grupo de operadores que actúan a nivel de bits. Estas operaciones usualmente se utilizan para definir flags.

Estos operadores efectúan operaciones únicamente con enteros: byte, short, integer, long.

Operador	Operación efectuada	Ejemplo	Resultado
&	Y Binario	45 & 255	45
	O Binario	99 46	111
^	O exclusivo	99 ^ 46	77
>>	Desplazamiento hacia la derecha (división por 2)	26>>1	13
<<	Desplazamiento hacia la izquierda (multiplicación por 2)	26<<1	52

Operadores lógicos

Los operadores lógicos están asociados a la manipulación de valores booleanos y se utilizan para construir expresiones lógicas.

Operador	Nombre	Operación
&&	AND	op1 && op2
	OR	op1 op2
!	negación	! op
&	AND	op1 & op2
	OR	op1 op2

Una particularidad de estas operaciones es que, si el operador 1 tiene un valor de “false” entonces el segundo operando ya no se evalúa, pues se da por hecho que el resultado de la condición con valor “true” no puede ser cumplida. También, se puede garantizar que los dos operandos sean evaluados forzosamente utilizando los operadores (&) y (|).

Operadores de concatenación

El operador “ + ” (más) ya utilizado para la suma se utiliza también para la concatenación de cadenas de caracteres. Si uno de los operandos es del tipo String, el operador “ + ” efectúa una concatenación con, eventualmente, una conversión implícita del otro operando a cadena de caracteres.

cadena1 + cadena2

1.2.5 Conversión de tipos

Java es bastante estricto en la declaración de sus tipos de datos, debido a esto, lo más importante al declarar una variable es la compatibilidad que hay del tipo de dato y la información contenida en esta.

Por ejemplo, no es posible declarar una variable de tipo "int" cuyo valor contenga un tipo "float", "char", "string", u otro tipo de dato.

En Java es posible transformar el tipo de una variable u objeto en otro diferente. Este proceso se denomina "conversión" o "casting". En java se pueden considerar dos tipos de conversiones: implícita y explícita.

Conversión implícita

La conversión implícita se efectúa de forma automática al asignar un valor de un tipo a otro que es compatible. A continuación se muestra una tabla con las conversiones implícitas posibles.

Nuevo tipo	Tipos origen
short	byte
int	byte, short, char
long	byte, short, char, int
float	byte, short, char, int, long
double	byte, short, char, int, long, float

Ejemplo:

```
byte x = 10;  
int y = x;
```

Conversión explícita

En este tipo de conversión es requerido que el programador especifique el tipo de dato al que se va a transformar el nuevo dato. Una forma de realizar conversiones consiste en colocar el tipo destino entre paréntesis, a la izquierda del valor que queremos convertir de la forma siguiente:

Tipo VariableNueva = (NuevoTipo) VariableAntigua;

Ejemplo:

```
byte a = 44;  
int x = (int) a;
```

Una de las desventajas que tenemos al usar este tipo de conversión explícita, es que se puede aplicar a tipos de datos no compatibles, esto genera que se pierda información, como recortar decimales, y esto genera pérdida de precisión en las operaciones que se hagan con los nuevos datos.

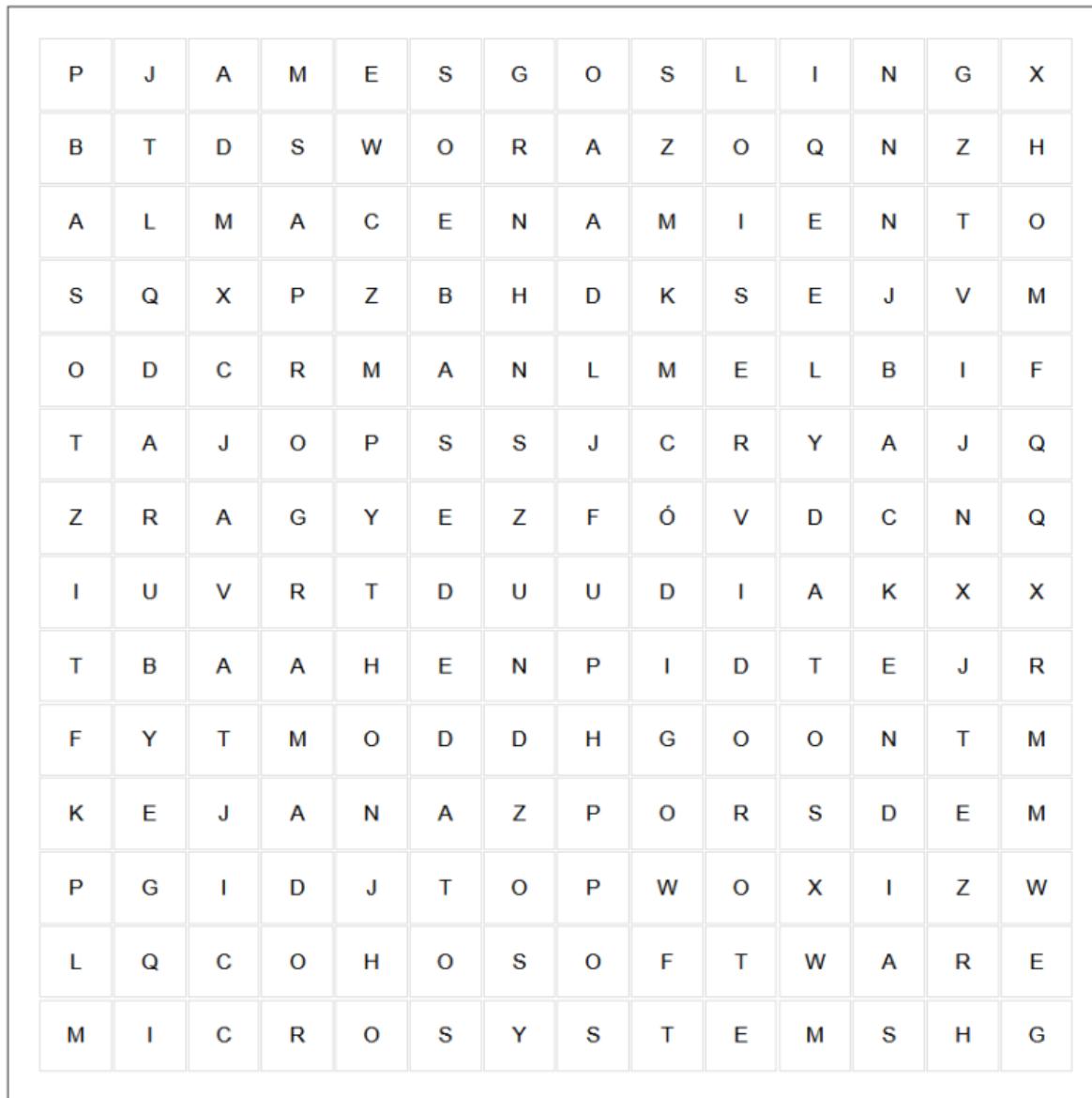
A continuación se muestra un ejemplo de conversiones no compatibles, resaltando que no es una buena práctica realizar este tipo de acciones.

```
float x = 9.5f;  
int y = (int) x;
```

Esta conversión causa la pérdida de toda la parte decimal.

Actividad

Completa la sopa de letras.



almacenamiento	backend
basededatos	código
datos	jamesgosling
java	jvm
microsystems	php
programador	python
ruby	servidor
software	wora

1.2.6 Uso de funciones matemáticas

Acceso a paquetes o importación de paquetes

Para acceder a los paquetes y a las clases de las bibliotecas se utiliza la palabra reservada “import”.

Se definen de la siguiente manera.

```
import nombrePaquete.nombre Clase;  
import nombrePaquete;
```

Ejemplo:

```
import java.util.Dato;           //acceso a la clase Dato del  
                                //paquete java.util  
  
import jav.io.*;                //se importa todo el paquete.
```

Las funciones básicas de Java se almacenan en el siguiente paquete con su correspondiente clase.

Import java.lang.*

Función Math

La clase Math es la librería matemática de Java contiene todas las funciones en coma flotante que se utilizan en geometría y trigonometría para efectuar cálculos complicados.

Todas las funciones tienen que ir precedidas del objeto Math y deben llevar como cabecera el paquete java.util.* o bien java.util.Math.

Por ejemplo:

```
x = Math.abs( -123 );  
y = Math.round( 123.567 );  
z = Math.pow( 2,4 );
```

También la clase “Math” define dos constantes: el número pi y el número e.

A continuación se muestran los métodos más usados de la clase Math.

Método	Descripción
abs(double a)	Devuelve el valor absoluto de un valor double introducido como parámetro.
abs(float a)	Devuelve el valor absoluto de un valor float introducido como parámetro.
abs(int a)	Devuelve el valor absoluto de un valor Entero introducido como parámetro.
abs(long a)	Devuelve el valor absoluto de un valor long introducido como parámetro.
acos(double a)	Devuelve el arcocoseno de un valor introducido como parámetro.
addExact(int x, int y)	Devuelve la suma de sus argumentos, lanzando una excepción si el resultado desborda un int.
addExact(long x, long y)	Devuelve la suma de sus argumentos, lanzando una excepción si el resultado se desborda a long.
asin(double a)	Devuelve el arcoseno de un valor introducido.
atan(double a)	Devuelve el arcotangente de un valor introducido.
cbrt(double a)	Devuelve la raíz cúbica de un doublevalor.
cos(double a)	Devuelve el coseno trigonométrico de un ángulo.
exp(double a)	Devuelve el número e de Euler elevado a la potencia de un doublevalor.
log(double a)	Devuelve el logaritmo natural (base e) de un double valor.
log10(double a)	Devuelve el logaritmo de base 10 de un doublevalor.
max(double a, double b)	Devuelve el mayor de dos valores double
max(float a, float b)	Devuelve el mayor de dos valores float.
max(int a, int b)	Devuelve el mayor de dos valores enteros.

max(long a, long b)	Devuelve el mayor de dos valores long.
min(double a, double b)	Devuelve el menor de dos valores double.
min(float a, float b)	Devuelve el menor de dos valores float.
min(int a, int b)	Devuelve el menor de dos valores enteros.
min(long a, long b)	Devuelve el menor de dos valores long.
multiplyExact (int x, int y)	Devuelve el producto de los argumentos, lanzando una excepción si el resultado desborda un int.
multiplyExact (long x, long y)	Devuelve el producto de los argumentos, lanzando una excepción si el resultado desborda un long.
pow(double a, double b)	Devuelve el valor del primer argumento elevado a la potencia del segundo argumento.
random()	Devuelve un doublevalor con un signo positivo, mayor o igual que 0.0 y menor que 1.0.
round(double a)	Devuelve el long redondeado más cercano al double introducido.
round(float a)	Devuelve el int más cercano y redondeado al float introducido.
sin(double a)	Devuelve el seno trigonométrico de un ángulo.
sqrt(double a)	Devuelve la raíz cuadrada positiva correctamente redondeada de un doublevalor.
tan(double a)	Devuelve la tangente trigonométrica de un ángulo.

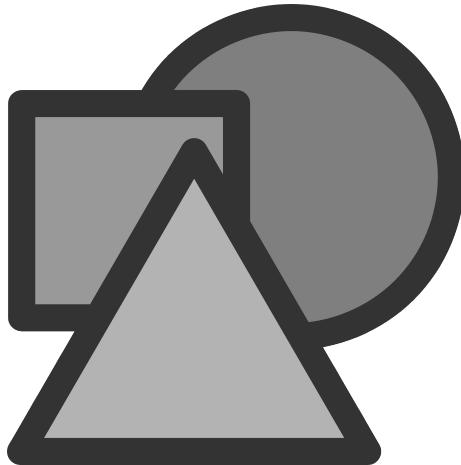
Ejemplo:

Utilizando las funciones trigonométricas sin, cos y tan.

```
double angulo = 45.0 * Math.PI/180.0;
System.out.println("El cos(" + angulo + ") es " + Math.cos(angulo));
System.out.println("sin(" + angulo + ") es " + Math.sin(angulo));
System.out.println("tan(" + angulo + ") es " + Math.tan(angulo));
```

1.2.7 Declaración de objetos

Un objeto es una instancia de clase, en otras palabras un objeto es un elemento representado en el mundo real. En un programa se representa mediante una variable y esta contiene la dirección de memoria del objeto.



Java no cuenta con punteros que se puedan ver y manejar como en otros lenguajes, pero hay que saber la referencias a un objeto son punteros; son variables que contienen la dirección de memoria del objeto que representan.

Las características que posee un objeto en Java son las siguientes:

Estado: Son los atributos que contiene un objeto.

Comportamiento: El objeto puede ejecutar acciones conocidas como métodos o funciones.

Atributos

Son los datos que posee el objeto, estos datos hacen que cada objeto sea diferente a otro objeto que pertenece a la misma clase.

Métodos

Son acciones que le permiten a un objeto realizar una tarea.

Declaración de un objeto

Se creará la referencia al objeto, de forma similar a como se declara a una variable de un tipo primitivo, esta referencia se utiliza para manejar el objeto.

La sintaxis general para declarar un objeto en Java es:

NombreClase referenciaObjeto;

Coche coche1:

Instanciación de un objeto

Para instanciar un objeto se reserva un bloque de memoria dinámica para almacenar los atributos del objeto. Al instanciar un objeto solo se reserva memoria para sus atributos pero no se guardan los métodos para cada objeto ya que los métodos son los mismos y los comparten todos los objetos de la clase.

La dirección de memoria donde se encuentra el objeto se asigna a la referencia. De forma general un objeto se instancia en Java así:

referenciaObjeto = new NombreClase();

coche1 = New Coche();

Las dos sentencias anteriores se pueden declarar como una sola. Por ejemplo:

Coche coche1 = New Coche();

Utilización

Una vez creado manearemos el objeto a través de su referencia. El acceso a los atributos se realiza a través del operador punto, que separa al identificador de la referencia del identificador del atributo:

referenciaObjeto.Atributo;

coche1.color;

Las llamadas a los métodos para realizar las distintas acciones se llevan a cabo separando los identificadores de la referencia y del método correspondiente con el operador punto:

referenciaObjeto.metodo([parámetros]);

coche1.frenar();

1.2.8 Strings

Una cadena o String, es una secuencia de caracteres, es un tipo de datos importante de Java. Para guardar, manejar y gestionar el funcionamiento de cadenas de texto se utiliza la clase String.

Se utiliza la siguiente nomenclatura o constructor para crear una variable de tipo String.

```
String palabra = new String ("Hola mundo");
```

Otra forma de declarar un string es de la siguiente forma:

```
String nombre;
```

También se pueden declarar varias variables de tipo String en una misma sentencia y luego agregar el valor de las cadenas.

Ejemplo.

```
String nombre, apellido;  
nombre = "Juan";  
apellido = "López";
```

Para concatenar cadenas se utiliza el operador (+). Para imprimir una cadena de texto se utilizan los métodos print() o println() que pertenecen a la clase printStream y se utilizan seguidos de la variable out perteneciente a la clase System.

Por ejemplo:

```
System.out.println("Mi nombre es:" + nombre + " " + apellido);
```

Dando como resultado la cadena:

"Mi nombre es: Juan López"

La clase String contiene métodos con los cuales se pueden operar cadenas. Colocando la variable de tipo string y a continuación el operador punto seguido del método de la clase String.

A continuación se muestran algunos de los métodos más importantes:

```
//Declarando primero la variable String  
String cadena = "cadena de texto";  
  
//length() Devuelve la cantidad de caracteres del String.  
System.out.println("función lenght(): " + cadena.length());
```

```

// substring (int i) Devuelve la subcadena del n-ésimo carácter de índice al final.
System.out.println("función substring(): " + cadena.substring(6));

//substring (int i, int j): Devuelve la subcadena del índice n a n-1.
System.out.println("función substring(int i, int j): " + cadena.substring(0,6));

//indexOf (String s):Devuelve el índice dentro de la cadena de la primera aparición
//de la cadena especificada.
System.out.println("función indexOf(): " + cadena.indexOf("texto"));

// boolean equals (Objeto otroObjeto): Compara este String con el objeto
//especificado.
System.out.println("función equals(): " + cadena.equals("cadena de texto"));

// boolean equalsIgnoreCase (String otroString): Compara este String con el objeto
//especificado ignorando minúsculas y mayúsculas
System.out.println("función equalsIgnoreCase (): " +
cadena.equalsIgnoreCase("cadena DE TEXTO"));

// toLowerCase(): Convierte la cadena en minúsculas.
System.out.println("función toLowerCase(): " + cadena.toLowerCase());

// toUpperCase(): Convierte la cadena en mayúsculas.
System.out.println("función toUpperCase(): " + cadena.toUpperCase());

// concat("Palabra a contatenar"): Concatena el valor de la cadena a la variable del
//String.
System.out.println("función concat(): " + cadena.concat(" concatenada "));

//lastIndexOf(): obtiene la última posición del carácter indicado
System.out.println("función lastIndexOf(): " + cadena.lastIndexOf("a"));

//charAt(indice int): muestra la letra asociada a la posición indicada
System.out.println("función charAt(): " + cadena.charAt(0));

```

La salida del anterior código es:

```

función length(): 15
función substring(): de texto
función indexOf(): 10
función equals(): true
función equalsIgnoreCase (): true
función toLowerCase(): cadena de texto
función toUpperCase(): CADENA DE TEXTO
función concat(): cadena de texto concatenada
función lastIndexOf(): 5
c

```

Convertir un número en un String o cadena

Una de las ventajas que tiene el operador más (+) para concatenar cadenas es que tiene la capacidad de concatenar números y cadenas al momento de ser compilado el programa, el resultado de esta concatenación es un String, perdiendo las propiedades todas aquellas variables que pertenezcan a un tipo de dato numérico.

Otra forma de concatenar números y cadenas es pasando las variables a través del método “`toString()`”, esto permitirá cambiar el tipo de dato guardándolo en otra variable y así poder operar únicamente cadenas.

Por ejemplo:

```
//Declaración de variables
double variableDouble = 15.48;
int i = 1;
String cadena1, cadena2;

//Convertimos a String cada variable usando toString
cadena1 = Double.toString(variableDouble);
cadena2 = Integer.toString(i);
```

1.2.9 Atributos y métodos estáticos

En ocasiones hay atributos y métodos que se repiten en la misma clase, permitiendo generalizar estos elementos creando atributos y métodos compartidos entre todos los objetos de una clase, teniendo por nombre de miembros estáticos.

Los atributos y métodos estáticos también son conocidos como atributos de clase y métodos de clase, es decir, pertenecen a la clase y no al objeto a una instancia de la clase.

Se declaran con la palabra `static` y su nomenclatura es la siguiente:

modificadorDeAcceso static tipoDeDatos nombreVariable;



public static double salarioBase = 1000;

Tienen las siguientes características.

Atributos estáticos:

- No hace referencia específica de cada objeto.
- Únicamente hay una copia y su valor es compartido para todos los objetos creados de la clase.
- No se requiere instanciar un objeto para utilizarse.

Para mandar a llamar un atributo se utiliza la siguiente sentencia.

NombreClase.atributoEstatico;

Métodos estáticos:

- Solamente tiene acceso a los atributos estáticos de la clase.
- No se requiere instanciar un objeto para utilizarse.

Para mandar a llamar un método se utiliza la siguiente sentencia.

NombreClase.metodoEstatico();

El siguiente ejemplo muestra el uso de atributos y métodos estáticos.

Primero se declara la Clase Empleado.

```
public class Empleado {  
  
    //Nombre del empleado  
    private String nombre;  
  
    //Apellido del empleado  
    private String apellido;  
    private int edad;  
    private static double salarioBase = 1000;  
  
    public double getSalarioBase() {  
        return this.salarioBase;  
    }  
    public static void setSalarioBase(double salario) {  
        salarioBase = salario;  
    }  
    public static String muestraMensaje(){  
        return "Soy un metodo estatico";  
    }  
}
```

Posteriormente se genera una nueva clase con el método main para poder manejar los atributos y métodos estáticos de la clase Empleado.

```
public class Main {  
  
    Run | Debug  
    public static void main(String[] args) {  
        Empleado empleado1 = new Empleado();  
        Empleado empleado2 = new Empleado();  
  
        //Salario  
        System.out.println("Salario base del empleado1: "+ empleado1.  
getSalarioBase());  
        System.out.println("Salario base del empleado2: "+ empleado2.  
getSalarioBase());  
  
        //Modificación del salario;  
        empleado1.setSalarioBase(salario: 500);  
        System.out.println("Salario base modificado del empleado1: "+  
empleado1.getSalarioBase());  
        System.out.println("Salario base modificado del empleado2: "+  
empleado2.getSalarioBase());  
  
        Empleado.setSalarioBase(salario: 2500);  
        System.out.println("Salario base modificado (Empleado.  
setSalario...) del empleado1: "+ empleado1.getSalarioBase());  
        System.out.println("Salario base modificado (Empleado.  
setSalario...) del empleado2: "+ empleado2.getSalarioBase());  
  
        System.out.println(Empleado.muestraMensaje());  
    }  
}
```

```
Salario base del empleado1: 1000.0  
Salario base del empleado2: 1000.0  
Salario base modificado del empleado1: 500.0  
Salario base modificado del empleado2: 500.0  
Salario base modificado (Empleado.setSalario...) del empleado1: 2500.0  
Salario base modificado (Empleado.setSalario...) del empleado2: 2500.0  
Soy un metodo estatico
```

1.2.10 Ejercicio: Datos y expresiones

Nota: utilizar System.out.println() para imprimir los resultados en consola

1. Variable: Crea una variable de tipo String que contenga el valor “Hola mundo” e imprime en consola su valor en minúsculas.
2. Número de caracteres: Con la variable creada en el ejercicio anterior imprime el número total de caracteres que contiene.
3. Con la variable del punto 1, concatenar la palabra “ nuevo.”, generando la palabra “Hola mundo nuevo” usando un método de la clase String.
4. Conversión: Crea una variable de tipo byte y conviertela en dos tipos de datos diferentes, luego vuelve a convertir estas variables numéricas en Strings y concatenar la siguiente cadena “Resultado: ” en su lado izquierdo e imprime en consola el resultado para cada una de las variables.

```
public class Main {
    Run | Debug
    public static void main(String[] args) {
        String palabra = "Hola mundo";
        byte numerobyte = 15;

        long x = numerobyte;
        int y = numerobyte;
        String cadenax, cadenay;

        cadenax = Long.toString(x);
        cadenay = Integer.toString(y);

        System.out.println(palabra.toLowerCase());
        System.out.println(palabra.length());
        System.out.println(palabra.concat(" nuevo."));

        System.out.println("Resultado: " + x);
        System.out.println("Resultado: " + y);

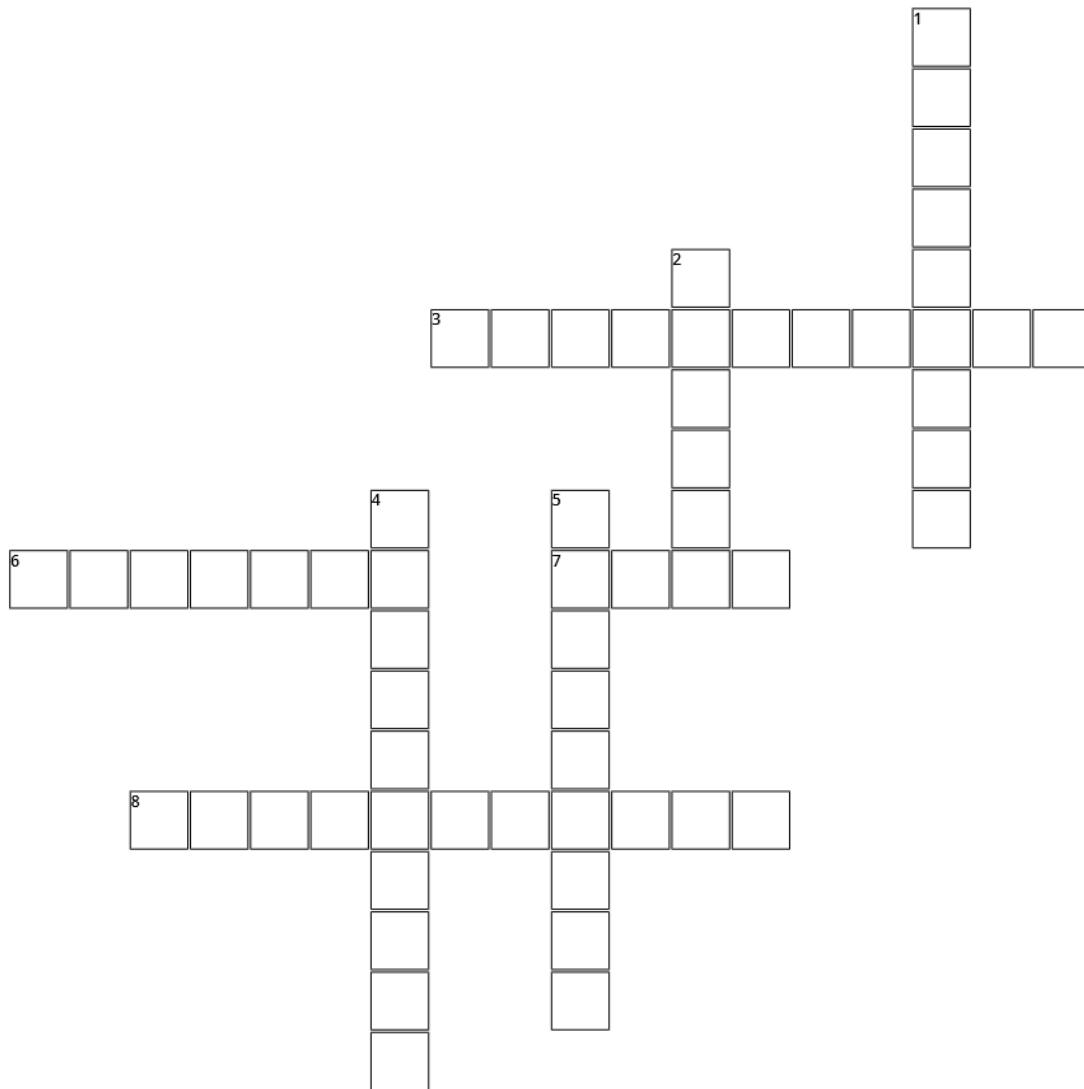
        //Para comprobar el tipo de dato de las cadenas: cadenax y cadenay
        System.out.println("Cadena x " + cadenax.getClass().getName());
        System.out.println("Cadena y " + cadenay.getClass().getName());
    }
}
```

La salida es la siguiente:

```
hola mundo
10
Hola mundo nuevo.
Resultado: 15
Resultado: 15
Cadena x java.lang.String
Cadena y java.lang.String
```

Actividad

Completa el siguiente crucigrama.



Horizontales

3. Operadores que se utilizan para manipular expresiones matemáticas
6. Operadores que están asociados a la manipulación de valores booleanos.
7. Es la librería matemática de Java
8. Operadores que determinan la relación que existe entre un operador con otro.

Verticales

1. Conversión que puede generar pérdida de precisión.
2. Para acceder a los paquetes y a las clases de las bibliotecas se utiliza la palabra reservada...
4. Operador que tiene como tarea asignar un valor a una variable
5. Conversión que se efectúa de forma automática al asignar un valor de un tipo a otro que es compatible.

<https://www.educima.com/crosswords/actividad-1583069>

1.3 Clases, referencias y objetos

1.3.1 ¿Qué son?

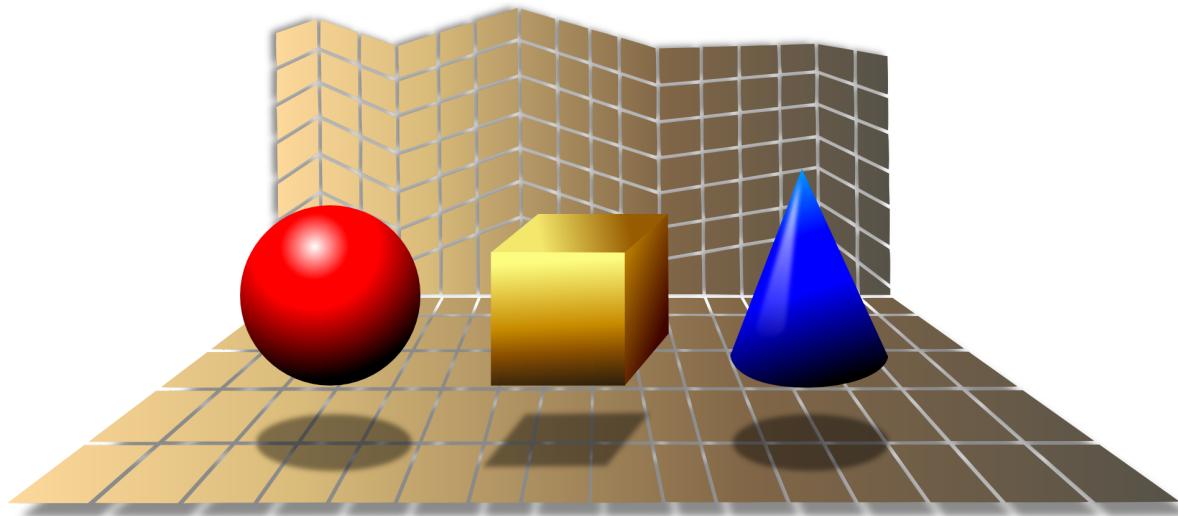
Una clase es un tipo de plantilla o prototipo que contiene una colección de datos o atributos, también, tiene un conjunto de métodos que son prácticamente el comportamiento para la manipulación de los datos. A los atributos y a los métodos se les conoce como miembros de la clase.

Con ayuda de una clase podemos instanciar múltiples objetos, algo similar a una fábrica para crear múltiples objetos.

El propio lenguaje de programación Java contiene clases ya definidas para poder utilizarlas como por ejemplo: la clase Math permite usar sus métodos para realizar operaciones matemáticamente complejas, de igual manera la clase System y su método “println()” o “print()” para poder imprimir caracteres por consola, o la clase Scanner que permite la entrada de datos por teclado.

De igual manera podemos generar nuestras clases con atributos y métodos propios para después utilizarlos en alguna tarea en específica, la declaración, instancia y utilización de los objetos de estas nuevas clases son iguales a las clases ya predefinidas en el lenguaje Java.

En el apartado **1.2.7 Declaración de objetos**, se definieron las características y declaraciones de un objeto. Si tienes dudas de cómo instanciar un objeto de una clase te recomendamos volver a leer este tema.



1.3.2 Creación e inicialización de objetos

A continuación se muestra un ejemplo de la clase Ave, su declaración, instancia y utilización de los objetos.

```

public class Ave {
    // Variables de instancia.
    String nombre;
    String raza;
    int edad;
    String color;

    // Declaración del constructor de la clase.
    public Ave(String nombre, String raza, int edad, String color) {
        this.nombre = nombre;
        this.raza = raza;
        this.edad = edad;
        this.color = color;
    }

    public String getNombre() {
        return nombre;
    }

    public String getRaza() {
        return raza;
    }

    public int getEdad() {
        return edad;
    }

    public String getColor() {
        return color;
    }

    public void imprimirDatosAve() {
        System.out.println("Hola soy un pajaro, mi nombre es: " + this.getNombre() +
            "\nMi raza, edad y color son: " + this.getRaza() + ", " + this.getEdad() +
            ", " + this.getColor());
    }
}

Run | Debug
public static void main(String[] args) {
    Ave periquito = new Ave(nombre: "Oliv", raza: "Periquito", edad: 5,
        color: "Azul");
    periquito.imprimirDatosAve();
}
}

```

La salida del programa es:

```

Hola soy un pajaro, mi nombre es: Oliv
Mi raza, edad y color son: Periquito, 5, Azul

```

1.3.3 La clase object

Java define una clase especial llamada Object que es una superclase implícita de todas las demás clases, es decir, todas las demás clases son subclases de Object. Y si aún no ha quedado claro se puede decir que es la “madre de todas las clases”.

Esto significa que una variable de referencia de tipo Object puede referirse a un objeto de cualquier otra clase o dicho de otra forma todas las clases heredan de Object.

La clase Object tiene métodos compartidos por todos los objetos que se creen a excepción de la reescritura de subclases..

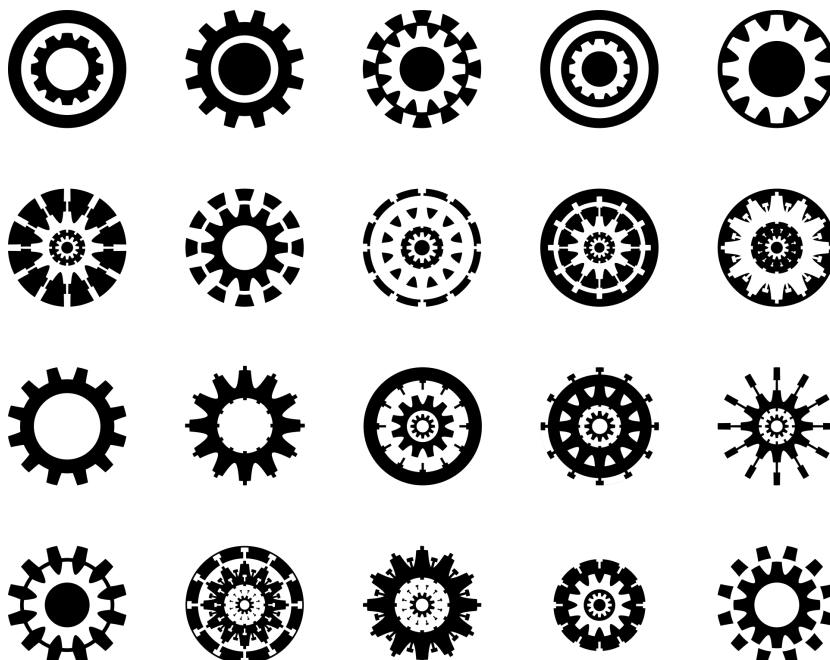
Entonces si se define

```
class Ejemplo { ... }
```

es equivalente a escribir:

```
class Ejemplo extends Object { ... }
```

Esta clase define los estados y comportamientos básicos que todos los objetos deben tener, como la posibilidad de compararse unos con otros, de convertirse a cadenas, de esperar una condición variable, de notificar a otros objetos que la condición variable ha cambiado y devolver la clase del objeto.



Object define los siguientes métodos, lo que significa que están disponibles en cada objeto:

Método	Sintaxis	Propósito
getClass	class _ getClass()	Obtiene la clase de un objeto en tiempo de ejecución
hashCode	int hashCode	Devuelve el código hash asociado con el objeto invocado
equals	boolean equals(Object obj)	Determina si un objeto es igual a otro
clone	Object clone()	Crea un nuevo objeto que es el mismo que el objeto que se está clonando
toString	String toString()	Devuelve una cadena que describe el objeto
notify	void notify()	Reanuda la ejecución de un hilo esperando en el objeto invocado
notifyAll	void notifyAll()	Reanuda la ejecución de todo el hilo esperando en el objeto invocado
wait	void wait(long timeout)	Espera en otro hilo de ejecución
wait	void wait(long timeout,int nanos)	Espera en otro hilo de ejecución
wait()	void wait()	Espera en otro hilo de ejecución
finalize	void finalize()	Determina si un objeto es reciclado (obsoleto por JDK9)

1.3.4 Referencias y objetos

Como ya se mencionó anteriormente, los objetos corresponden a cosas o entidades que se encuentran en el mundo real y tienen una identidad propia. Por ejemplo como: carrito de compras, cliente y producto.

Los objetos tienen 2 tipos de componentes:

- Campos o atributos. Se trata del componente de un objeto en Java que recopila y almacena datos. Pueden ser primitivos o, a su vez, otro tipo de objetos. Lo que se conoce como agregación o composición de objetos. La idea detrás de esta característica es que cada atributo o campo representa una propiedad determinada del objeto.
- Rutinas o métodos. Las rutinas llevan a cabo una serie de acciones o tareas en función de los atributos que definen el objeto.

Un programa típico de Java crea muchos objetos que interactúan al invocar métodos. De esta forma, podemos decir que un objeto consiste en:

- Estado: está representado por atributos de un objeto. También refleja las propiedades de un objeto.
- Comportamiento: se representa mediante métodos de un objeto. También refleja la respuesta de un objeto con otros objetos.
- Identidad: le da un nombre único a un objeto y permite que un objeto interactúe con otros objetos.

1.3.5 Comparación de objetos.

El operador de igualdad (==) en Java es un recurso proporcionado por el lenguaje de comparación de tipos.

- Función equals (Object o): Todos los tipos de datos que creamos en Java son internos, heredan de la clase Object.

- La función .hashCode(): esta es la misma parte del objeto Object y se usa para una comparación de objetos más rápida en estructuras hash porque solo devuelve un número entero

1.3.6 Recolector de basura

Garbage Collector es un programa que administra la memoria automáticamente, en donde la desasignación de objetos es manejada por Java en vez del programador. El programador crea los objetos y el entorno en tiempo de ejecución de Java (Java runtime) es el responsable de gestionar el ciclo de vida de los objetos.

Una vez creado el objeto se usa algo de memoria y la memoria permanece asignada hasta que haya referencias para la utilización del objeto.

Cuando no quedan referencias a un objeto, el recolector de basura de Java borra el objeto, liberando así la memoria que ocupaba y siendo reclamada por otro objeto. No hay necesidad explícita de eliminar un objeto debido a que Java maneja la desasignación automáticamente.

1.3.7 Métodos y campos de clase

Así como existen miembros estáticos cuya característica principal es englobar una serie de datos y funciones compartidas entre clases. Otra característica importante es que no se requiere de la instanciación de un objeto para poder ser utilizados estos miembros estáticos. Cuando no se tienen características, entonces cada clase debe tener sus propias características, esto es común entre clase.

Una clase contiene dos tipos de miembros, llamados campos y métodos de clase. Los campos son los datos que va a almacenar la clase o los objetos de esa clase, y los métodos son los conjuntos de sentencias que van a manipular los campos de esa clase.

Los métodos son subrutinas que manipulan los datos definidos por la clase, en muchos casos, brindan acceso a esos datos.

Un método contiene una o más declaraciones. En un código Java bien escrito, cada método realiza solo una tarea. Cada procedimiento tiene un nombre, y es este el que se utiliza para llamar al procedimiento. Sin embargo, recuerde que main() está reservado para el procedimiento que empieza ejecución de su programa. Además, es muy mala práctica utilizar las palabras reservadas de Java para nombres de métodos.

¿Cómo se escribe un método?

La declaración de métodos consta de los siguientes elementos: un tipo retorno, un identificador y una lista de argumentos también conocidos como parámetros. Además pueden tener una visibilidad específica con los modificadores de acceso.

La sintaxis para definir un método de clase es la siguiente.

```
[modificadoresDeAcceso] tipoDeRetorno nombreMétodo (  
    tipoParámetro1 parámetro1, tipoParámetro2 parámetro2, tipoParámetroN  
    parámetroN) {  
        //cuerpo del método  
        // return valorDeTipoDeRetorno;  
    }
```

Modificadores de acceso: Especifica el modificador de acceso public, protected, private, abstract, final, static, synchronized.

Tipo de retorno: Se utiliza la palabra reservada return para devolver un valor cada vez que se mande a llamar el método, puede ser de cualquier tipo inclusive una clase. Cuando se ejecuta el “return” de un método termina devolviendo un único valor como resultado. Para devolver múltiples valores mediante una función en Java deben combinarse todos los resultados en un objeto y devolver la referencia al objeto.

Si se declara un retorno de tipo void la llamada al método no devuelve nada, entonces se puede omitir la sentencia return.

Nombre del método: La única limitante de este elemento es que no debe existir otro identificador con el mismo nombre que se le quiere dar al método en el alcance actual.

Parámetros: Seguido del nombre del método se especifica entre paréntesis, una lista de parámetros definidos en el método cada uno de ellos precedido por su tipo y separados por comas. Los parámetros son esencialmente variables que reciben el valor de los argumentos pasados al método cuando se llama. Si el método no tiene parámetros, la lista de parámetros estará vacía.

Retornado de un método

En general, hay dos condiciones que hacen que un método retorne:

- Primero: cuando se encuentra la llave de cierre del método.
- El segundo: cuando se ejecuta una declaración de retorno (return).

Formas de devolución

Hay dos formas de devolución: Una cuando usa métodos con la palabra void (aquellos que no devuelven un valor) y otra para devolver valores.

En un método void, se puede causar la terminación inmediata de un método utilizando esta forma de devolución:

return:

Cuando se ejecuta esta instrucción, el control del programa regresa a quién lo llama, omitiendo cualquier código restante en el cuerpo del método.

El siguiente ejemplo contiene un bucle for que solo se ejecutará del 0 al 10, aunque la sentencia límite del for sea **i < 20**.

Por ejemplo:

```
public void metodoEjemplo(){  
int i;  
    for( i = 0; i < 20; i++){  
        if ( i == 10) return;  
        System.out.println(i);  
    }  
}
```

Nota: Al ser un método de tipo void, este no devuelve ningún valor, solo finaliza el método.

También es aceptable tener múltiples instrucciones return en un mismo método, dando como resultado varios puntos de salida pero solo se ejecutara uno.

Por ejemplo:

```
public void metodoVariosReturns(){  
int x = 0;  
    if( x == 0){  
        //cuerpo del método  
        return;  
    }  
    else{  
        //cuerpo del método  
        return;  
    }  
}
```

Devolviendo un valor

Aunque los métodos con un tipo de retorno void no son raros, la mayoría de los métodos devolverán un valor. De hecho, la capacidad de devolver un valor es una de las características más útiles de un método.

Los valores de retorno se utilizan para una variedad de propósitos en la programación. En algunos casos, el valor de retorno contiene el resultado de algún cálculo. En otros casos, el valor de retorno puede simplemente indicar éxito o falla. Y en otros, puede contener un código de estado.

Cualquiera que sea el propósito, usar valores de retorno de método es una parte integral de la programación de Java.

Los métodos devuelven un valor a la rutina de llamada usando esta forma de devolución:

return valor;

Aquí, valor es el valor devuelto. Esta forma de devolución se puede usar solo con métodos que tienen un tipo de devolución no-void. Además, un método no-void debe devolver un valor utilizando esta forma de return.

1.3.8 Composición de objetos

La reutilización de código en Java es fundamental y es una de las características del enfoque orientado a objetos. Hay dos formas de reutilizar el código, mediante la composición y mediante la herencia.

La composición de objetos consiste en la capacidad que tienen las clases de contener referencias de otras clases, en otras palabras utilizar objetos dentro de otros objetos y es una de las herramientas de re-utilización de software más poderosas, ya que podemos crear objetos que sean útiles para múltiples objetos futuros.

La composición toma los objetos similares de las clases para generar otra clase que contenga estos objetos.



Por ejemplo:

Primero se crea la clase Persona y luego la clase Empresa.

```
public class Persona {  
    private String nombre;  
    private int edad;  
  
    private Direccion dirección;  
  
    public Persona(String nombre, int edad) {  
        this.nombre = nombre;  
        this.edad = edad;  
    }  
    public Direccion getDirección() {  
        return dirección;  
    }  
    public void setDirección(Direccion dirección) {  
        this.dirección = dirección;  
    }  
    public String getNombre() {  
        return nombre;  
    }  
    public void setNombre(String nombre) {  
        this.nombre = nombre;  
    }  
    public int getEdad() {  
        return edad;  
    }  
    public void setEdad(int edad) {  
        this.edad = edad;  
    }  
}
```

```
public class Empresa {  
  
    private String sede;  
    private Direccion dirección;  
  
    public Direccion getDirección() {  
        return dirección;  
    }  
    public void setDirección(Direccion dirección) {  
        this.dirección = dirección;  
    }  
    public String getSede() {  
        return sede;  
    }  
    public void setSede(String sede) {  
        this.sede = sede;  
    }  
}
```

Luego, creamos la clase Dirección la cual engloba los objetos similares de la clase Persona y Empresa.

```
public class Direccion {  
    private String calle;  
    private int numero;  
  
    public Direccion(String calle, int numero) {  
        this.calle = calle;  
        this.numero = numero;  
    }  
    public String getCalle() {  
        return calle;  
    }  
    public void setCalle(String calle) {  
        this.calle = calle;  
    }  
    public int getNumero() {  
        return numero;  
    }  
    public void setNumero(int numero) {  
        this.numero = numero;  
    }  
}
```

Por último creamos la clase Main para poder instanciar los objetos de las clases heredadas.

```
public class Main {  
    Run | Debug  
    public static void main(String[] args) {  
  
        Direccion d1= new Direccion(calle: "Periférico ",numero: 11);  
        Direccion d2= new Direccion(calle: "Hidalgo",numero: 503);  
  
        Persona personal1= new Persona(nombre: "Paula",edad: 20);  
        personal1.setDirección(d1);  
  
        Empresa empresal1 = new Empresa();  
        empresal1.setSede(sede: "Jalisco");  
        empresal1.setDirección(d2);  
  
        System.out.println(personal1.getDirección().getCalle());  
        System.out.println(emprsal1.getDirección().getCalle());  
    }  
}
```

La salida del código es:

```
Periférico  
Hidalgo
```

1.3.9 Herencia

Como se vio en el tema anterior, la herencia junto con la composición son características de reutilización de código en la POO.

La herencia es un mecanismo que permite definir una clase a partir de otra ya existente. La herencia permite heredar las características de otras clases.

En el lenguaje de Java, una clase que se hereda se denomina **superclase**. La clase que hereda se llama **subclase**. Por lo tanto, una subclase es una versión especializada de una superclase. Hereda todas las variables y métodos definidos por la superclase y agrega sus propios elementos únicos.



Privilegios de acceso

Hasta el momento hemos visto en los códigos expresiones con modificadores de acceso como public o private. Por ejemplo:

En métodos.

```
public void metodoVariosReturns(){
    int x = 0;
    if( x == 0){
        //cuerpo del método
        return;
    }
    else{
        //cuerpo del método
        return;
    }
}
```

También en declaración de atributos.

```
private String nombre;
private Direccion direccion;
```

El uso de las palabras reservadas como public, private y protected usadas en una declaración permiten controlar el acceso a variables, métodos, clases anidadas o interfaces anidadas por fuera del ámbito de una clase.

Modificadores de acceso

Las modificaciones de acceso se colocan delante de la declaración de las **clases**: public, protected, private

Como hemos visto el uso de un modificador de acceso es opcional y se puede omitir.

Modificador de acceso	Función
public	Una declaración es accesible por cualquier clase .
protected	Una declaración es accesible a cualquier subclase de la declaración de la clase o a cualquier clase del mismo paquete.
private	Una declaración sólo es accesible desde dentro de la clase donde está declarada.

Por lo general una variable de instancia de una clase se declara privada (private) para evitar su uso no autorizado o alteración. Heredar una clase no anula la

restricción de acceso privado. Por lo tanto, aunque una subclase incluye a todos los miembros de su superclase, no puede acceder a los miembros de la superclase que se han declarado privados.

Modificadores de variables

Las variables se pueden modificar mediante indicadores igual que las clases.

Modificador de acceso	Función
Public	La clase o variable de instancia es accesible desde todos los ámbitos.
Protected	La clase o variable de instancia es accesible solo en el ámbito actual de la clase ,el ámbito del paquete actual y todas las subclases de la clase actual.
Private	La clase o variable de instancia es accesible solo en el ámbito actual de la clase.
Final	La variable es una constante ,de modo que su valor no se puede modificar.
Static	La variable es una variable de clase, compartida entre todos los objetos de la instancia de una clase.
Transfert	Se declara que no es parte de un estado persistente del objeto .
Volatile	Se necesita a veces cuando se utiliza una variable instancia por para prevenir al compilador de su optimización.



Constructores y herencia

Constructores

Los constructores nos ayudan en la inicialización de valores al momento de crear un nuevo objeto.

La sintaxis de un constructor es muy parecida a la de un método, sin incluir el **tipoDeResultado**, además el nombre del constructor debe coincidir con el nombre de la clase. El constructor se invoca automáticamente cuando se crea una instancia de la clase.

```
[modificadoresDeConstructor] nombreDeConstructor (
    tipoParámetro1 parámetro1,
    tipoParámetro2 parámetro2, ...
)
{
//cuerpo del constructor
}
```

Los **modificadoresDeConstructor** siguen las mismas reglas que en los métodos normales, pero un constructor abstracto estático final no está permitido.

Un constructor puede ser invocado con el operador “new”, una clase puede tener múltiples métodos constructores, siempre que éstos se diferencien unos de otros en el número y/o tipo de parámetros.

Por ejemplo:

```
class Persona {
    private String nombre = "";
    private int edad = 0;

    public Persona(String nom, int años){
        nombre = nom;
        edad = años;
    }

    public static void main(String args[]) {
        Persona p = new Persona("Fidel Del Ángel", 13);
        System.out.println("Nombre: " + p.nombre + " " + "Edad: " + p.edad);
    }
}
```

Constructores en clase heredadas

En una jerarquía, es posible que tanto las superclases como las subclases tengan sus propios constructores. Esto plantea una pregunta importante: **¿Qué constructor es responsable de construir un objeto de la subclase, el de la superclase, el de la subclase o ambos?** La respuesta es esta: el constructor para la superclase construye la porción de la superclase del objeto, y el constructor para la subclase construye la parte de la subclase.

Esto tiene sentido porque la superclase no conoce ni tiene acceso a ningún elemento de la subclase. Por lo tanto, su construcción debe ser separada. De hecho, la mayoría de las clases tienen constructores explícitos (no predeterminados). Aquí verás cómo manejar esta situación.

Cuando solo la subclase define un constructor, el proceso es simple: simplemente construye el objeto de la subclase. La parte de la superclase del objeto se construye automáticamente utilizando su constructor predeterminado.

En Java “super(...)” es usado en vez del nombre del constructor de la superclase. Si no se usa super entonces se supone implícitamente que el cuerpo del constructor comienza con la llamada “super()” sin parámetros. El resto del cuerpo es como un método normal

Para heredar los atributos y métodos de una clase padre a una clase hijo se utiliza la palabra reservada “**extends**” siguiendo la siguiente sintaxis:

```
[modificadorDeAcceso] class NombreClase extends Superclase
{
    // cuerpo de la clase ampliada
    ...
}
```

Y para el constructor de la subclase:

```
public NombreClase(arg11, ...)
{
    super(...);
    ...
}
```

Por ejemplo:

Se crea una clase padre, que hereda sus atributos y métodos a la clase Hija.

```
public class Padre {  
    static String colorOjos = "Café";  
    String apellido;  
    String nombre;  
  
    public void imprimirNombre() {  
        System.out.println("Mi nombre es " + this.nombre);  
    }  
}
```

Luego, se crea la clase Hija que extiende de la clase Padre y dentro la función main para imprimir el resultado.

```
public class Hijo extends Padre{  
    Integer edad;  
  
    Run | Debug  
    public static void main(String[] args) {  
        Hijo instanciaHijo = new Hijo();  
        instanciaHijo.edad = 11;  
        instanciaHijo.nombre = "Juan";  
        instanciaHijo.imprimirNombre();  
    }  
}
```

La salida en consola es.

```
Mi nombre es Juan
```

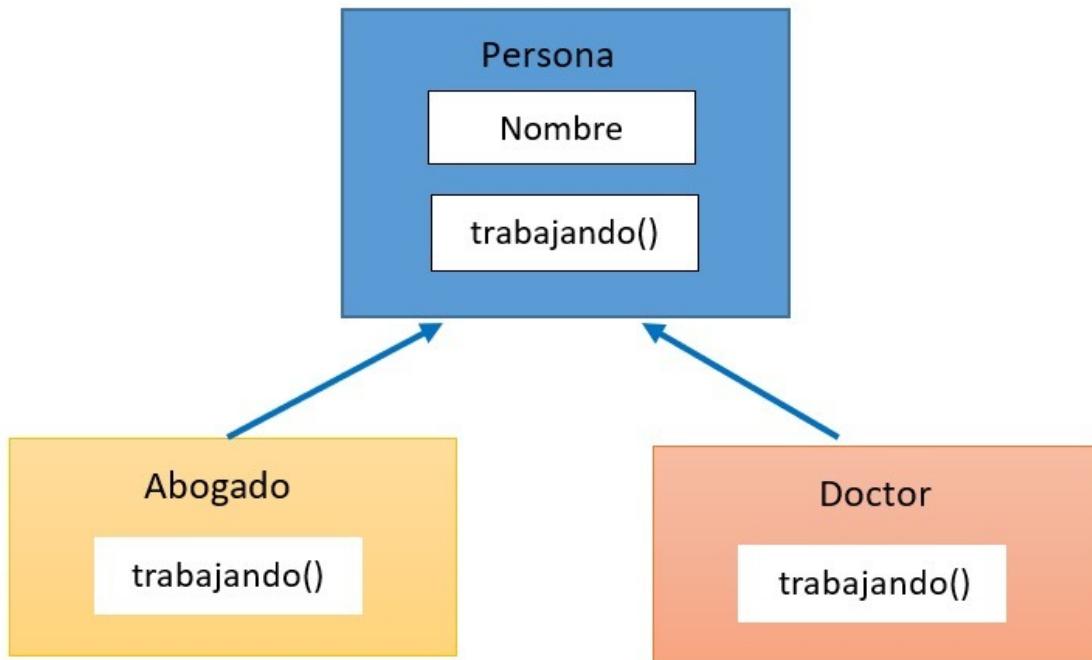
1.3.10 Polimorfismo

En la programación orientada a objetos, el polimorfismo es la capacidad de un objeto de una clase para proporcionar respuestas diferentes e independientes. Es decir, los objetos que son entidades pueden contener diferentes tipos de valores durante la ejecución del programa.

Para proporcionar respuestas diferentes se realizan operaciones polimórficas que son aquellas que hacen funciones similares con objetos diferentes.

A continuación se muestra un ejemplo para dejar más claro este concepto ya que el polimorfismo y la herencia están estrechamente relacionados, y el polimorfismo siempre ha sido difícil de entender para la mayoría de los programadores.

Ejemplo: Si tenemos las siguientes clases Persona, Doctor y Abogado, entonces, las clases están claramente organizadas en una jerarquía de herencia.



El método trabajando de la clase Persona es un método abstracto y no tiene implementación. Por el contrario, los métodos de las clases hijas tienen sobrecargado el método trabajando. Como podemos ver en el ejemplo:

```

public abstract class Persona {
    private String nombre;

    public Persona(String nombre){
        super();
        this.nombre= nombre;
    }
    public String getNombre() {
        return this.nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public abstract Boolean trabajando();

}

Run | Debug
public static void main(String[] args) {
    Persona i = new Doctor(nombre: "Pedro");
    Persona d = new Abogado(nombre: "Gama");
    System.out.println("Estatus doctor-trabajando: "+ i.trabajando());
    System.out.println("Estatus abogado-trabajando: "+d.trabajando());
}
  
```

```
public class Doctor extends Persona{  
    public Doctor(String nombre) {  
        super(nombre);  
    }  
    @Override  
    public Boolean trabajando() {  
        return true;  
    }  
}
```

```
public class Abogado extends Persona {  
    public Abogado(String nombre) {  
        super(nombre);  
    }  
    @Override  
    public Boolean trabajando() {  
        return false;  
    }  
}
```

```
Estatus doctor-trabajando: true  
Estatus abogado-trabajando:false
```

En este pequeño programa vemos cómo el método trabajando se ejecuta de forma polimórfica en cada una de las clases . La clase persona lo tiene como método abstracto y cada clase hija lo escribe (Override).

1.3.11 Ejercicios: Clases, referencias y Objetos

En este ejemplo se crearán las clases, Cliente, Producto y Tienda.

```
public class Cliente {  
    private String nombre;  
    private Double cantidadDinero;  
  
    public Cliente(String nombre, Double cantidadDinero) {  
        this.nombre = nombre;  
        this.cantidadDinero = cantidadDinero;  
    }  
  
    public String getNombre() {  
        return this.nombre;  
    }  
  
    public void setNombre(String nombre) {  
        this.nombre = nombre;  
    }  
  
    public Double getCantidadDinero() {  
        return this.cantidadDinero;  
    }  
  
    public void setCantidadDinero(Double cantidadDinero) {  
        this.cantidadDinero = cantidadDinero;  
    }  
}
```

```
public class Producto {  
    private String nombre;  
    private Double precio;  
    public Producto(String nombre, Double precio) {  
        this.nombre = nombre;  
        this.precio = precio;  
    }  
  
    public String getNombre() {  
        return nombre;  
    }  
  
    public void setNombre(String nombre) {  
        this.nombre = nombre;  
    }  
  
    public Double getPrecio() {  
        return precio;  
    }  
  
    public void setPrecio(Double precio) {  
        this.precio = precio;  
    }  
}
```

```
public class Tienda{  
    private String nombre;  
    private String direccion;  
  
    //Dos mas de nuestras clase  
    private Producto producto;  
    private Cliente cliente;  
  
    public Tienda(String nombre, String direccion, Cliente cliente, Producto  
    producto){  
        this.nombre = nombre;  
        this.direccion = direccion;  
        this.producto = producto;  
        this.cliente = cliente;  
    }  
    public String getNombre() {  
        return this.nombre;  
    }  
    public void setNombre(String nombre) {  
        this.nombre = nombre;  
    }  
    public String getDireccion() {  
        return this.direccion;  
    }  
    public void setDireccion(String direccion) {  
        this.direccion = direccion;  
    }  
  
    public Producto getProducto() {  
        return this.producto;  
    }  
    public void setProducto(Producto producto) {  
        this.producto = producto;  
    }  
    public Cliente getCliente() {  
        return this.cliente;  
    }  
    public void setCliente(Cliente cliente) {  
        this.cliente = cliente;  
    }  
}
```

La clase Tienda contiene un objeto de la clase Cliente, un objeto de la clase Producto y atributos de tipo String para almacenar el nombre y la dirección de la tienda. La clase contenedora es Tienda y las clases contenidas son Cliente y Producto.

El código principal donde se crean los objetos de las clases Cliente, Producto y Tienda.

```
public class EjemploJava {  
    Run | Debug  
    public static void main(String[] args) {  
        Producto servilleta = new Producto(nombre: "Servilleta", precio: 140.00);  
        Cliente cliente1 = new Cliente(nombre: "Juanito", cantidadDinero: 10.00);  
        Tienda tienda = new Tienda(nombre: "Sonora", direccion: "Calle Robles #22",  
        cliente1, servilleta );  
  
        System.out.println("Nombre del cliente: "+ tienda.getCliente().getNombre());  
        System.out.println("Nombre del producto: "+ tienda.getProducto().getNombre  
        ());  
    }  
}
```

En una relación de composición, hay atributos de la clase contenedora que son objetos que pertenecen a la clase contenida. Un objeto de la clase contenedora puede acceder a los métodos públicos de las clases contenidas.

En la declaración de la clase Tienda se han definido dos métodos ‘get’ para los atributos de tipo objeto. El método getProducto() devuelve un objeto de tipo Producto y el método getCliente() devuelve un objeto de tipo Cliente.

En el ejemplo, el objeto servilleta de la clase Tienda puede acceder a los métodos públicos de su propia clase y de las clases Cliente y Producto. Un objeto de la clase Tienda puede ejecutar métodos ‘get’ para mostrar la información de los objetos que contiene.

La salida del código es la siguiente:

```
Nombre del cliente: Juanito  
Nombre del producto: Servilleta
```

1.4 Creación de clases en JAVA

1.4.1 ¿Qué son los métodos?

1.4.2 ¿Cómo se escribe un método?

A continuación se muestra otro ejemplo con los diversos conceptos vistos hasta ahora, el ejemplo tiene la finalidad de ser analizado por el estudiante para reforzar estos conceptos y que haga cambios en el código para visualizar sus efectos.

Primero se creará una clase padre llamada “Figura” y posteriormente las clases hijas “Circulo”, “Triangulo” y “Rectangulo”, para finalizar con la clase Principal donde se crearán los objetos.

Entre las particularidades que tiene este programa se pueden mencionar: la clase “Circulo” tiene dos constructores, se utiliza la palabra super para mandar a llamar a los atributos de la clase padre para poderlos mandar a llamar en el constructor de la clase hija; un objeto de la clase hija puede ser instanciado con la clase padre.

Clase Figura

```

public abstract class Figura {
    private int ancho;
    private int alto;
    private double perimetro;
    private double area;

    public Figura(){}
}

public Figura(int ancho, int alto){
    super();
    this.ancho = ancho;
    this.alto = alto;
}

public abstract double areaFigura(); //metodo abstracto

public int getAncho() {
    return this.ancho;
}

public void setAncho(int ancho) {
    this.ancho = ancho;
}

public int getAlto() {
    return this.alto;
}

```

```

public void setAlto(int alto) {
    this.alto = alto;
}

public double getPerimetro() {
    return this.perimetro;
}

public void setPerimetro(double perimetro) {
    this.perimetro = perimetro;
}

public double getArea() {
    return this.area;
}

public void setArea(double area) {
    this.area = area;
}
}

```

Clase Círculo

```

import java.lang.Math;

public class Circulo extends Figura{
    private double area;
    private double radio;

    public Circulo(double radio) {
        super();
        this.radio = radio;
    }
    public Circulo(int ancho, int alto, double radio) {
        super(ancho, alto);
        this.radio = radio;
    }
    @Override
    public double areaFigura() {
        area = 3.1416 * Math.pow(radio, 2);
        return area;
    }
}

```

Clase Triángulo

```

public class Triangulo extends Figura{
    private double area;

    public Triangulo(int ancho, int alto) {
        super(ancho, alto);
    }
    @Override
    public double areaFigura() {
        area = (getAlto()*getAncho())/2;
        return area;
    }
}

```

Clase Rectángulo

```

public class Rectangulo extends Figura{
    private double area;

    public Rectangulo(int ancho, int alto) {
        super(ancho, alto);
    }
    @Override
    public double areaFigura() {
        area = getAlto()*getAncho();
        return area;
    }
}

```

Clase Principal

```

public class Principal {
    Run | Debug
    public static void main(String[] args) {

        Figura r1 = new Rectangulo(ancho: 10, alto: 10);
        System.out.println("Área del rectángulo 1 instanciado con Figura: " + r1.
            areaFigura());

        Figura r2 = new Rectangulo(ancho: 100, alto: 10);
        r2.setArea(r2.areaFigura());
        System.out.println("Área del rectángulo 2 instanciado con Figura: " + r2.
            getArea());

        Figura t1 = new Triangulo(ancho: 10, alto: 10);
        System.out.println("Área del triángulo 1 instanciado con Figura: " + t1.
            areaFigura());

        Triangulo t2 = new Triangulo(ancho: 10, alto: 10);
        System.out.println("Área del triángulo 2 instanciado con Triángulo: " + t2.
            areaFigura());

        Circulo c1 = new Circulo(ancho: 0, alto: 0, radio: 5);
        System.out.println("Área del círculo 1 instanciado con Circulo: " + c1.
            areaFigura());

        Circulo c2 = new Circulo(radio: 5);
        System.out.println("Área del círculo 2 instanciado con Circulo y con otro
            constructor: " + c2.areaFigura());
    }
}

```

La salida del código es la siguiente.

```

Área del rectángulo 1 instanciado con Figura: 100.0
Área del rectángulo 2 instanciado con Figura: 1000.0
Área del triángulo 1 instanciado con Figura: 50.0
Área del triángulo 2 instanciado con Triángulo: 50.0
Área del círculo 1 instanciado con Circulo: 78.53999999999999
Área del círculo 2 instanciado con Circulo y con otro constructor: 78.53999999999999

```

1.4.3 Argumentos de un método

El ejemplo anterior incluye un concepto conocido como argumentos o parámetros de un método y sirven para intercambiar información con el método. Pueden servir para introducir datos para ejecutar el método (entrada) o para obtener o modificar datos tras su ejecución (salida).

Los parámetros se declaran en la cabecera de la declaración de los métodos.

Al declararse el parámetro, se indica el tipo de dato y el identificador correspondiente. Los parámetros o argumentos de un constructor o de un método pueden ser de cualquier tipo, ya sean tipos primitivos o referencias de objetos (en este caso debe indicarse el identificador de la clase correspondiente).

1.4.4 Sobrecarga de métodos

La sobrecarga de métodos es la creación de múltiples métodos con el mismo nombre pero diferentes listas de tipos de parámetros. Java usa el número y los tipos de parámetros para elegir qué definición de método ejecutar.

Java diferencia los métodos sobrecargados con base en el número y tipo de parámetros o argumentos que tiene el método y no por el tipo que devuelve.

También hay sobrecarga de constructores: cuando una clase tiene más de un constructor, se dice que tiene sobrecarga de constructores.

Para ver los argumentos de un método utilizaremos el ejemplo anterior.

En la clase Figura se puede observar un constructor con los argumentos: int ancho e int alto.

```
public Figura(int ancho, int alto){  
    super();  
    this.ancho = ancho;  
    this.alto = alto;  
}
```

Posteriormente se crea la clase Cuadrado con dos métodos sobrecargados con el mismo nombre: "perimetroCuadrado" aplicando el paso de dos y tres parámetros.

Clase Cuadrado

```
public class Cuadrado extends Figura {  
    private double area;  
    private double perimetro;  
    public String color;  
  
    public Cuadrado (int ancho, int alto, String color) {  
        super(ancho, alto);  
        this.color = color;  
    }  
    @Override  
    public double areaFigura() {  
        area = getAlto()*getAncho();  
        return area;  
    }  
    public void colorCuadrado(String color){  
        System.out.println("El color del cuadrado es: " + this.color);  
    }  
    public void perimetroCuadrado(int ancho, int alto, String color){  
        perimetro = (2*ancho)+(2*alto);  
        System.out.println("El perimetro del cuadrado es: " + perimetro);  
        System.out.println("Y su color es " + this.color);  
    }  
    public void perimetroCuadrado(int ancho, int alto){  
        perimetro = (2*ancho)+(2*alto);  
        System.out.println("El perimetro del cuadrado es: " + perimetro);  
    }  
}
```

Creando el objeto cuadrado1 en Clase Principal

```
// ...  
Cuadrado cuadrado1 = new Cuadrado(ancho: 10, alto: 5, color: "rojo");  
cuadrado1.colorCuadrado(cuadrado1.color);  
cuadrado1.perimetroCuadrado(cuadrado1.getAncho(), cuadrado1.getAlto(),  
cuadrado1.color);  
cuadrado1.perimetroCuadrado(cuadrado1.getAncho(), cuadrado1.getAlto());
```

La salida del código es la siguiente.

```
El color del cuadrado es: rojo  
El perimetro del cuadrado es: 30.0  
Y su color es rojo  
El perimetro del cuadrado es: 30.0
```

1.4.5 ArrayList, List y Mapas

Arrays

Un array en Java es una estructura de datos que nos permite almacenar un conjunto de variables de variables del mismo tipo que se conocen con un nombre en común.

El tamaño de un array en Java es fijo y se declara en un primer momento, sin cambiarse más adelante como sí ocurre en otros lenguajes permitiendo trabajar de un modo más ordenado y claro.

Los diferentes elementos contenidos en un array se definen por un índice que comienza en 0 y se acceden a ellos utilizando su índice correspondiente, al igual que los objetos de la clase se crean utilizando la palabra reservada new.

La sintaxis para declarar un array en Java y su inicialización es la siguiente:

```
tipo_dato nombre_array[];  
nombre_array = new tipo_dato[longitudDelArray];
```

Por ejemplo:

```
char arrayCaracteres[];  
arrayCaracteres = new char[10];
```

En este caso se inicializa un array de tipo carácter para posteriormente definir su tamaño de 10 posiciones de memoria.

Otra forma de inicializar un array es agregando su contenido y su longitud dependerá de la cantidad de elementos agregados.

```
tipo_dato nombre_array[] = {elemento1, elemento2, ... , elementoN}
```

Por ejemplo:

```
char abecedario[] = {'a','b','c','d','e'};
```

Para leer un valor contenido en un array se utiliza la siguiente nomenclatura.

```
nombre_array[numero_elemento];
```

Por ejemplo:

```
arrayCaracteres[3];
```

Para asignar un valor se ocupa una sintaxis similar.

```
nombre_array[numero_elemento] = valor;
```

Por ejemplo:

```
arrayCaracteres[3] = 'g';
```

Para recorrer un array tiene una variable de solo lectura llamada ".length" que devuelve el un número de elementos contenidos en un array.

```
for(int i = 0; i < abecedario.length; i++){  
    System.out.println(abecedario[i]);  
}
```

List

Las listas son un tipo de colección que hereda de la interface Collection, son una estructura de datos que respeta el orden en el cual fueron agregados los elementos, en la cual se tiene un control absoluto y preciso del lugar en el que se quiere insertar, también permiten registros repetidos.

Las listas permiten manipular los datos permitiéndonos decidir en qué posición puede ser insertado o eliminado un elemento. Al igual que los arreglos se utiliza su índice para buscar y acceder a los elementos de la lista.

La interface List se encuentra en el paquete java.util, algunas de las clases que implementa son ArrayList, LinkedList, Vector.

ArrayList

La clase ArrayList en Java, es una clase que permite almacenar datos en memoria de forma similar a los arrays, con la ventaja de que el número de elementos que almacena, lo hace de forma dinámica, es decir, que no es necesario declarar su tamaño como pasa con los arrays.

En resumen ArrayList es una clase contenedora genérica que implementa arrays dinámicos de **objetos** de cualquier tipo.

De forma general un ArrayList en Java se crea de la siguiente forma:

```
ArrayList nombreArray = new ArrayList();
```

Al ejecutar esta instrucción se crea ArrayList nombreArray vacío.

Para agregar elementos al ArrayList se utiliza la siguiente nomenclatura.

```
nombreDelArrayList.add(valor);
```

Por ejemplo:

```
ArrayList a = new ArrayList();
a.add("Lista");
a.add(3.1416);
a.add('c');
a.add(123);
```

Como se mencionó, un ArrayList puede contener objetos de tipos distintos.

De los cuatro elementos que integran el ArrayList “a”, únicamente “Lista” pertenece a un objeto el resto no lo son, pero un ArrayList solo está compuesto por objetos, para arreglar esta situación el compilador convierte los datos de tipo no objeto en objetos de su clase envolvente (clase contenedora o wrapper) antes de añadirlos al array.

Esta característica de agregar elementos de diferente tipo a los ArrayList pueden causar ciertas complicaciones cuando se esté trabajando con él, para ello hay otra forma de declarar un ArrayList que consiste en indicar el tipo de objetos que contiene y el array solo tiene la capacidad de almacenar elementos de ese tipo.

La forma de declarar esta particularidad es la siguiente:

```
ArrayList<tipo> nombreArray = new ArrayList();
```

Donde tipo debe ser una clase e indica el tipo de objetos que contendrá el array.

No se pueden usar tipos primitivos. Para un tipo primitivo se debe utilizar su clase envolvente.

Por ejemplo:

```
ArrayList<Integer> numeros = new ArrayList();
```

Crea el array “números” del tipo entero(*Integer*).

Recorrer un ArrayList

- Se puede recorrer con un bucle for, utilizando el método .size().

```
for(int i = 0;i < array.size();i++){
    System.out.println(array.get(i));
}
```

- Con un bucle foreach.

Si conocemos el tipo de elementos del array.

```
for(Integer i: numeros){
    System.out.println(i);
}
```

En caso de desconocer el tipo o bien el arreglo es de distintos tipos.

```
for(Integer i: numeros){
    System.out.println(i);
}
```

A continuación se muestran los principales métodos de ArrayList.

MÉTODO	DESCRIPCIÓN
size()	Devuelve el número de elementos (int)
add(X)	Añade el objeto X al final. Devuelve true.
add(posición, X)	Inserta el objeto X en la posición indicada.
get(posición)	Devuelve el elemento que está en la posición indicada.
remove(posición)	Elimina el elemento que se encuentra en la posición indicada. Devuelve el elemento eliminado.
remove(X)	Elimina la primera ocurrencia del objeto X. Devuelve “true” si el elemento está en la lista.
clear()	Elimina todos los elementos.
set(posición, X)	Sustituye el elemento que se encuentra en la posición indicada por el objeto X. Devuelve el elemento sustituido.
contains(X)	Comprueba si la colección contiene al objeto X. Devuelve “true” o “false”.
indexOf(X)	Devuelve la posición del objeto X. Si no existe devuelve -1

Si quieres profundizar más sobre el tema y los métodos de ArrayList puedes consultar la documentación oficial de Oracle.

<http://docs.oracle.com/javase/9/docs/api/java/util/ArrayList.html>

Ejemplo de un ArrayList.

```
import java.util.ArrayList;
public class Ejemplo_ArrayList {
    Run | Debug
    public static void main(String[] args) {
        ArrayList arlTest = new ArrayList();

        System.out.println("Tamaño de arrayList en la creación: " + arlTest.size());
        arlTest.add(e: "D");
        arlTest.add(e: "U");
        arlTest.add(e: "K");
        arlTest.add(e: "E");

        System.out.println("Tamaño de arrayList DESPÚES DE AGREGAR ELEMENTOS: " +
            arlTest.size());
        System.out.println("Lista de todos los elementos." + arlTest);

        arlTest.remove(p: "D");

        System.out.println("Ver contenido después de eliminar un elemento (D): " +
            arlTest);
        System.out.println("Tamaño de arrayList DESPÚES DE ELIMINAR ELEMENTOS: " +
            arlTest.size());
        //Para verificar si en la lista contiene algún elemento "K".
        System.out.println(arlTest.contains(o: "K"));
    }
}
```

La salida de este código es la siguiente:

```
Tamaño de arrayList en la creación: 0
Tamaño de arrayList DESPÚES DE AGREGAR ELEMENTOS: 4
Lista de todos los elementos.[D, U, K, E]
Ver contenido después de eliminar un elemento (D): [U, K, E]
Tamaño de arrayList DESPÚES DE ELIMINAR ELEMENTOS: 3
true
```

Maps

La interfaz Map presente en el paquete `java.util` representa un mapeo entre una “clave” y un “valor”; de tal manera que para una clave solamente tenemos un valor. Esta estructura de datos también es conocida en otros lenguajes de programación como “Diccionarios”, aunque en cada lenguajes esta estructura de datos tiene sus matices. La interfaz Mapa no es un subtipo de la interfaz Colección. Por tanto, se comporta un poco diferente al resto de tipos de colección, además un mapa contiene claves únicas.

Clases que implementan la interfaz Map

Como ya se mencionó , Map es una interface y por tanto se deben de implementar los métodos de la interface. Java ya tiene implementadas varias “clases Map”. Las implementaciones generales de los mapas en Java son:

- "[HashMap](#)": Los elementos que se insertan en el Map no tendrán un orden específico. No aceptan claves duplicadas ni valores nulos.
- "[TreeMap](#)": El Mapa lo ordena de forma "natural". Por ejemplo, si la clave son valores enteros (como luego veremos), los ordena de menor a mayor.
- "[LinkedHashMap](#)": Inserta en el Map los elementos en el orden en el que se van insertando; es decir, que no tiene una ordenación de los elementos como tal, por lo que esta clase realiza las búsquedas de los elementos de forma más lenta que las demás clases.

La diferencia principal de estas 3 clases es la forma o el orden en las que guardan los valores en el Map.

Para recorrer los Maps existe un elemento denominado Iterador sirven para recorrer los Map y poder trabajar con ellos, contiene 3 métodos: “`hasNext()`” para comprobar que siguen quedando elementos en el iterador, el “`next()`” para que nos de el siguiente elemento del iterador; y el “`remove()`” que sirve para eliminar el elemento del Iterador.

Los principales métodos para trabajar con los Map son los siguientes:

Método	Descripción
clear()	Este método se utiliza para borrar y eliminar todos los elementos o asignaciones de una colección de mapas especificada.
containsKey (Objeto)	Este método se utiliza para verificar si una clave en particular se está mapeando en el mapa o no. Toma el elemento clave como parámetro y devuelve True si ese elemento está mapeado en el mapa.
containsValue (Objeto)	Este método se utiliza para comprobar si un valor en particular está siendo asignado por una o más de una clave en el Mapa. Toma el valor como parámetro y devuelve Verdadero si ese valor está asignado por alguna de las claves en el mapa.
entrySet ()	Este método se utiliza para crear un conjunto de los mismos elementos contenidos en el mapa. Básicamente, devuelve una vista de conjunto del mapa o podemos crear un nuevo conjunto y almacenar los elementos del mapa en ellos.
get (Object key)	Devuelve el valor para esa key.
isEmpty()	Devuelve true si el mapa está vacío
remove (Object key)	Elimina el elemento proporcionado por la clave.
código hash()	Este método se utiliza para generar un código hash para el mapa dado que contiene claves y valores.
size()	Devuelve el número de elementos Clave/Valor del mapa.
values()	Devuelve la colección de valores del mapa.
put (K key, V value)	Se asocia la clave con el valor.

1.4.6 Ejercicio: Métodos y Listas

A continuación se ejemplifican algunos conceptos de la clase Map.

```

import java.util.HashMap;
import java.util.Map;
import java.util.TreeMap;

public class EjemploEstructuras {
    Run | Debug
    public static void main(String[] args) {
        Map <Integer, String> mapa1 = Map.of(k1: 1, v1: "Luis", k2: 2, v2: "Ernesto",
        k3: 3, v3: "Fernando");
        System.out.println("Elemento [3] de mapa1: "+mapa1.get(key: 3));

        System.out.println(x: "\nImpresión elementos mapa1:");
        for (Map.Entry<Integer, String> entrada: mapa1.entrySet()) {
            System.out.println(entrada.getKey()+" ->"+ entrada.getValue());
        }
        //No se puede mandar a llamar directamente al map sino mandar a traer una
        //clase que implementa el map
        System.out.println(x: "\nEjemplo TreeMap: \n");
        Map <Integer, String> mapa2 = new TreeMap<>();
        mapa2.put(key: 1, value: "Luis");
        mapa2.put(key: 2, value: "Ernesto");
        mapa2.put(key: 3, value: "Fernando");
        mapa2.put(key: 3, value: "Ana");
        mapa2.put(key: 4, value: "Ana");
    }
}

```

```

for (Map.Entry<Integer, String> entrada: mapa2.entrySet()) {
    System.out.println(entrada.getKey()+" ->"+ entrada.getValue());
}
System.out.println(x: "\nEjemplo HashMap: \n");
Map<Integer, String> map = new HashMap<Integer, String>();
map.put(key: 17, value: "Yerenia");      map.put(key: 7,
value: "Mariana");
map.put(key: 16, value: "Rodrigo");      map.put(key: 8, value: "Sahile");
map.put(key: 3, value: "Emmanuel");      map.put(key: 5, value: "Liz");
map.put(key: 11, value: "Miriam");       map.put(key: 14, value: "Damian");
map.put(key: 18, value: "Sahile");       map.put(key: 6, value: "Dafne");
map.put(key: 1, value: "Carlos");        map.put(key: 15, value: "Claudia");

// Imprimimos el Map con un Iterador
java.util.Iterator<Integer> it = map.keySet().iterator();
while(it.hasNext()){
    Integer key = it.next();
    System.out.println("Número alumno: " + key + " -> Nombre: " + map.get(key));
}
}

```

La salida por consola es:

```
Elemento [3] de mapa1: Fernando

Impresión elementos mapa1:
2 ->Ernesto
1 ->Luis
3 ->Fernando

Ejemplo TreeMap:

1 ->Luis
2 ->Ernesto
3 ->Ana
4 ->Ana

Ejemplo HashMap:

Número alumno: 16 -> Nombre: Rodrigo
Número alumno: 17 -> Nombre: Yerenia
Número alumno: 1 -> Nombre: Carlos
Número alumno: 18 -> Nombre: Sahile
Número alumno: 3 -> Nombre: Emmanuel
Número alumno: 5 -> Nombre: Liz
Número alumno: 6 -> Nombre: Dafne
Número alumno: 7 -> Nombre: Mariana
Número alumno: 8 -> Nombre: Sahile
Número alumno: 11 -> Nombre: Miriam
Número alumno: 14 -> Nombre: Damian
Número alumno: 15 -> Nombre: Claudia
```

Actividad

Indica si el enunciado es verdadero (V) o falso (F).

- Map es una interfaz.
- isEmpty() Devuelve true si el mapa tiene elementos.
- Las diferencias entre HashMap, TreeMap y LinkedHashMap son el orden en las que guardan los valores en el Map.
- Un ArrayList solo está compuesto por objetos y tipos de datos primitivos.
- Un Map no pertenece a una estructura de datos.
- La sobrecarga de métodos es la creación de múltiples métodos con diferente nombre pero diferentes listas de tipos de parámetros.
- Java usa el número y los tipos de parámetros para elegir qué definición de método ejecutar.
- El polimorfismo es la capacidad de un objeto de una clase para proporcionar respuestas diferentes e independientes.
- Final se utiliza para declarar una constante, de modo que su valor no se puede modificar.
- Los constructores nos ayudan en la inicialización de valores al momento de crear un array.
- La composición de objetos consiste en la capacidad que tienen las clases de contener referencias de otras clases, en otras palabras utilizar objetos dentro de otros objetos .
- HashMap es considerada “madre de todas las clases”.
- El polimorfismo es un mecanismo que permite definir una clase a partir de otra ya existente.
- Un parámetro en Java es una estructura de datos que nos permite almacenar un conjunto de variables de variables del mismo tipo que se conocen con un nombre en común.
- Los parámetros de un método y sirven para intercambiar información con el método.

Cuestionario unidad 1 backend

Devuelve el número de elementos Clave/Valor del mapa.

- A. size()
- B. for
- C. isContent()

Devuelve true si el mapa está vacío.

- A. get ()
- B. isEmpty()
- C. containsValue ()

Pertenece a una estructura de datos.

- A. Parámetro
- B. Int
- C. Map

Se define como la capacidad que tienen las clases de contener referencias de otras clases.

- A. Herencia
- B. Composición
- C. Polimorfismo

Es considerada “madre de todas las clases”.

- A. Main
- B. Map
- C. Object

Es un mecanismo que permite definir una clase a partir de otra ya existente.

- A. Polimorfismo
- B. Herencia
- C. Constructores

Nos ayudan en la inicialización de valores al momento de crear un nuevo objeto.

- A. Polimorfismo
- B. Constructores
- C. Herencia

Es la creación de múltiples métodos con el mismo nombre pero diferentes listas de tipos de parámetros.

- A. Herencia
- B. Sobrecarga
- C. Polimorfismo

Es la capacidad de un objeto de una clase para proporcionar respuestas diferentes e independientes.

- A. Polimorfismo
- B. Sobrecarga
- C. Herencia

Los parámetros de un método sirven para intercambiar información con el método.

- A. Verdadero
- B. Falso

No aceptan claves duplicadas ni valores nulos

- A. TreeMap
- B. HashMap
- C. LinkedHashMap

Inserta en el Map los elementos en el orden en el que se van insertando

- A. TreeMap
- B. HashMap
- C. LinkedHashMap

El Mapa lo ordena de forma "natural".

- A. TreeMap
- B. HashMap
- C. LinkedHashMap

¿Qué método de Iterator sirve para comprobar que siguen quedando elementos en el Map?

- A. hasNext()
- B. remove()
- C. next()

Método que nos da el siguiente elemento del iterado en Map.

- A. hasNext()
- B. remove()
- C. next()

Sirve para eliminar el elemento del Iterador en un Map.

- A. hasNext()
- B. remove()
- C. next()

Son las unidades básicas de almacenamiento en Java

- A. Métodos
- B. Map
- C. Variables

Los operadores que están asociados a la manipulación de valores booleanos.

- A. Unarios
- B. Lógicos
- C. Comparación

Los operadores unarios son prácticamente operadores aritméticos, con la diferencia que realizan una operación sobre un mismo operando.

- A. Unarios
- B. Lógicos
- C. Comparación

Referencias

Anónimo. (2022, 16 octubre). *Tipos de Datos Primitivos en Java*. Manual Web.

<https://www.manualweb.net/java/tipos-datos-primitivos-java/>

López Quesada, Juan. (s. f.). *Java: Tipos primitivos*. Universidad de Murcia

http://dis.um.es/%7Elopezquesada/documentos/IES_1213/IAW/curso/UT3/Actividad esAlumnos/8/index.html

Robledano, Angel. (2021, 27 agosto). *Qué es Java: Principios básicos y evolución*.

OpenWebinars.net. <https://openwebinars.net/blog/que-es-java/>

Anónimo. (s.f). *¿Qué es Back End, Front End y Back Office y por qué es importante para tu web?* . Agencia Inbound Marketing Madrid.

<https://nestategia.com/desarrollo-web-back-end-front-end/>

Anónimo. (s. f.). Desarrollo Web. *Java*.. <https://desarrolloweb.com/home/java>

Anónimo. (s. f.). *Tipos y métodos de Java básicos*. Copyright IBM

Corp.<https://www.ibm.com/docs/es/iis/11.5?topic=jrules-basic-java-types-methods>

Anónimo. (s. f.). *¿Qué es el lenguaje de programación JAVA? - Base de Conocimientos*. ICTEA.

<https://www.ictea.com/cs/index.php?rp=/knowledgebase/8790/iQue-es-el-lenguaje-de-programacion-JAVA.html>

Sonix. (2017, 31 diciembre). *Java: Conversión de tipos (Java casting)*. Tech Krowd.

<https://techkrowd.com/programacion/java/java-conversion-de-tipos-casting/>

Anónimo. (2019, 20 febrero). *Curso de Java. ¿Cómo usar la clase Math?*

codesitio.com.

<https://codesitio.com/recursos-utiles-para-tu-web-o-blog/cursos/curso-de-java-como-usar-la-clase-math/>

Warnimont, Joe. (2022, 15 junio). *Backend vs Frontend: ¿En Qué Se Diferencian?*

Kinsta. <https://kinsta.com/es/blog/backend-vs-frontend/>

Anónimo. (s. f.). *La clase Math*. Itlp.

<http://www.itlp.edu.mx/web/java/Tutorial+de+Java/Cap3/math.html>

Anónimo. (s. f.). *¿Cómo crear Objetos en Java? Definición y Ejemplos.* Open Bootcamp <https://open-bootcamp.com/cursos/java/crear-objetos>

Anónimo. (2018, 1 abril). *La clase String.* Development & System.
<http://www.developandsys.es/string-java/>

Anónimo. (s. f.). *Java static. Atributos y métodos estáticos o de clase (java).* Código HD <http://codigohd.blogspot.com/2014/05/java-static-atributos-y-metodos.html>

Anónimo. (s. f.). *IV - Sobrecarga de métodos y de constructores.* Profesores FI
http://profesores.fi-b.unam.mx/carlos/java/java_basico4_6.html

Anónimo. (s. f.). *Clase Object.* 2º ASIR - IES San Juan Bosco - Lorca
http://dis.um.es/%7Elopezquesada/documentos/IES_1415/IAW/curso/UT3/Actividad esAlumnos/java7/paginas/pag7.html

Anónimo. (s. f.). *Object (clase) java.lang.Object.* Java Vademécum Universidad Politécnica Madrid.
<http://dit.upm.es/%7Epepe/libros/vademecum/index.html?n=354.html>

Caules, C. Á. (2022, 5 noviembre). *Java Composición y la reutilización del código.* Arquitectura Java.
<https://www.arquitecturajava.com/java-composicion-y-la-reutilizacion-del-codigo/>

Guille, E. (s. f.). *Composición en JAVA.* Ingeniería Systems
<https://www.ingenieriasystems.com/2016/03/composicion-en-java.html>

Composición. (s. f.). *Composición.* SBWeb
<http://www.sc.ehu.es/sbweb/fisica/cursoJava/fundamentos/clases1/composicion.htm>

Caules, C. Á. (2022, 3 septiembre). *Java Polimorfismo, Herencia y simplicidad.* Arquitectura Java.
<https://www.arquitecturajava.com/java-polimorfismo-herencia-y-simplicidad/>

Richard. (2017, 2 junio). *ArrayList en Java, con ejemplos.* Jarroba.
<https://jarroba.com/arraylist-en-java-ejemplos/>

Anónimo. (s. f.). *Java ArrayList. Estructura dinámica de datos.* Punto com no es un lenguaje

<http://puntocomnoesunlenguaje.blogspot.com/2012/12/arraylist-en-java.html>

Anónimo. (2018, 31 octubre). *¿Qué es un array en Java?* IfgeekthenNTTdata.

<https://ifgeekthen.nttdata.com/es/que-es-un-array-en-java>

Richard. (2017, 2 junio). *Map en Java, con ejemplos.* Jarroba.

<https://jarroba.com/map-en-java-con-ejemplos/>

Anónimo. (s. f.). *La interfaz Map ej Java.* (2021, 23 septiembre). DHtrust.

<https://dhtrust.org/instrucciones/interfaz-de-mapa-en-java-geeksforgeeks/>

Anónimo. (s. f.). *Operadores y expresiones.* IBM Corp.

<https://www.ibm.com/docs/es/tivoli-monitoring/6.3.0?topic=language-operators-expressions>