Enunciado 2 – 30 puntos – Lista Doblemente Enlazada

Un explorador de Internet posee los botones "adelante" y "atrás" para poder avanzar o retroceder desde una página "actual". Si la página actual es la primera en ser visitada no se puede retroceder y, por consiguiente, si la página actual es la última en ser visitada no se podrá avanzar. Considerando lo antes descrito y que el **Explorador** corresponde a una "lista Doblemente Enlazada" que posee 3 nodos centinela (paginaInicial, paginaFinal y paginaActual) y cada nodo tiene los atributos "url", "nombre de página" y dos punteros (pSig y pAnt), se pide que implemente las clases antes descritas e incluya los métodos:

a) getInformacion( ) de la clase **Nodo** que permite imprimir la URL y el Nombre de la Página b) visitarPagina(url, nombre) de la clase **Explorador** que agrega una nueva página a la lista de páginas visitadas y: Define a dicha página como la actual, como la última de la lista de páginas y, además, imprime sus datos a través del método getInformación del punto a).

c) avanzarPag( ) de la clase **Explorador** que permite ir a la página siguiente desde la página actual dejando a la que estaba como siguiente como la nueva actual, siempre y cuando se pueda realizar esta operación. Luego imprime los datos de la página actual a través del método getInformación del punto a).

d) retrocederPag( ) de la clase **Explorador** que permite ir a la página anterior desde la página actual dejando a la que estaba como anterior como la nueva actual, siempre y cuando se pueda realizar esta operación. Luego imprime los datos de la página actual a través del método getInformación del punto a). e) Escriba un Main que permita probar el funcionamiento de su programa.

Considere que posee y puede utilizar los siguientes "esqueletos" de cada una de las clases y el archivo Main de Prueba (declarados en Python 3):

CLASE NODO

```
class Nodo(object):
 def
__init__(self,url,nombre):
#ATRIBUTOS NECESARIOS

 #Letra A
 def getInformacion(self):
 #CONTENIDO DEL MÉTODO
```

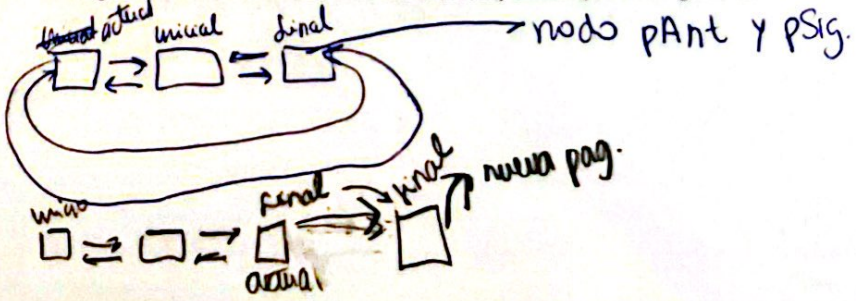
CLASE LISTA DOBLE

```
import ClaseNodo as CN

class Explorador(object):
 def __init__(self):
 #NODOS CENTINELA

 def getVacio(self):
 if(self.__paginaInicial == None):
return True

 #Letra B
 def
visitarPagina(self,url,nombre):
#CONTENIDO DEL MÉTODO

 #Letra C
 def avanzarPag(self):
 #CONTENIDO DEL MÉTODO
 #Letra D
 def retrocederPag(self):
 #CONTENIDO DEL MÉTODO
```

MAIN DE PRUEBA

```
import ClaseListaDoble as CLD

#Main de prueba
exp = CLD.Explorador()
exp.avanzarPag() #sin páginas no funciona
print("———")
#Primera página
exp.visitarPagina("www.udp.cl","U. Diego
Portales") #Ultima página y actual
exp.visitarPagina("www.gmail.com","Google
Mail") print("———")
exp.avanzarPag() #No se puede
print("———")
exp.retrocederPag() #Cambia la actual a
UDP print("———")
exp.retrocederPag() #No se puede
print("———")
exp.avanzarPag() #Cambia la actual a Google
Mail print("———")
```

Recuerde que el hecho de tener la misma respuesta que el Main de Prueba no garantiza que su desarrollo sea correcto (dicho Main es solo una ayuda, pero no una respuesta absoluta). Adicionalmente, "los esqueletos" entregados indican el nombre de los archivos/clases que debe generar.



Enunciado 1 – 30 puntos – Tablas de Hash

Tras vencer a Broly en la última visita de Freezer al planeta tierra del Universo 7, Gokú y sus amigos han decidido crear una academia de artes marciales para enseñar a cualquiera que lo desee. Inicialmente, se deben crear tres grupos a cargo de Gokú, Vegeta y Krilin respectivamente, aunque pueden aparecer más grupos asociados a nuevos maestros (por ejemplo, Androide 17, Androide 18 y muchos más). Como norma decretada por el Maestro Rochi (Director de las Escuelas del Estilo de la Tortuga), cada grupo puede tener como máximo 5 discípulos que son asignados a cada uno de los grupos por orden de llegada (los 5 primeros con Gokú, los siguientes 5 con Vegeta, los siguientes 5 con Krilin y así sucesivamente). Si no quedan cupos disponibles, se crea un nuevo grupo con su respectivo maestro (cuyo nombre se ingresa por teclado) y se le asignan los cinco (o menos) nuevos discípulos hasta que ya no queden cupos. El último proceso se repite si es que hay más discípulos sin asignar. No puede haber maestros repetidos.

Construya una función que reciba un arreglo (en C/C++) o lista (en Python 3) de discípulos(asuma que siempre tendrá 11 o más elementos del tipo string) y genere la Tabla de Hash correspondiente a la academia. Considere que las llaves serán los nombres del maestro y el valor, la lista de discípulos de ese grupo. Tenga en cuenta que, al crear un nuevo grupo después del tercero, se debe pedir al usuario el nombre del maestro. Finalmente, deberá imprimir los datos de la academia donde se pueda apreciar a cada Maestro con sus respectivos discípulos. Agregue un Main para que se pueda ejecutar y probar el funcionamiento de su Programa.

Ejemplos

Si el arreglo/lista de discípulos es el siguiente:

L = ["Caro", "Pipe", "Ricardo", "Pamela", "Tito", "Lalo", "Maca", "Fran", "Tania", "Eduardo", "Tom", "Max"]

La Academia quedaría como esta:

A = {"Gokú": ["Caro", "Pipe", "Ricardo", "Pamela", "Tito"], "Vegeta": ["Lalo", "Maca", "Fran", "Tania", "Eduardo"], "Krilin": ["Tom", "Max"]}

Si el arreglo/lista de discípulos es el siguiente:

L = ["Caro", "Pipe", "Ricardo", "Pamela", "Tito", "Lalo", "Maca", "Fran", "Tania", "Eduardo", "Tom", "Max", "Pepe", "Walala", "Chaos", "Beerus"]

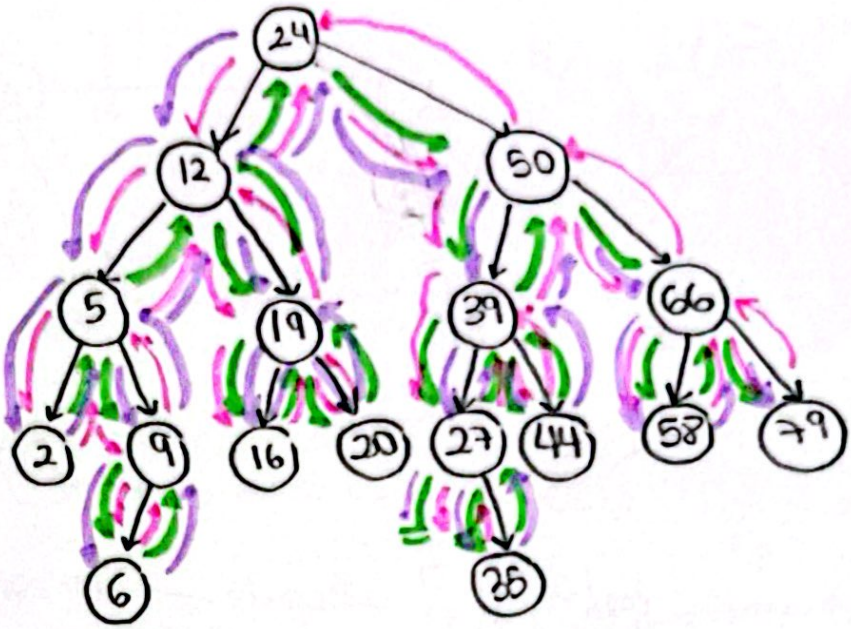
El programa deberá solicitar un nuevo maestro:

Ingrese nombre de uno nuevo maestro Whis

Y por ello la Academia quedaría como esta:

A = {"Gokú": ["Caro", "Pipe", "Ricardo", "Pamela", "Tito"], "Vegeta": ["Lalo", "Maca", "Fran", "Tania", "Eduardo"], "Krilin": ["Tom", "Max", "Pepe", "Walala", "Chaos"], "Whis": ["Beerus"]}

Ejercicio 1



a) PreOrden R-I-D

24-12-5-2-9-6-19-16-20-50-39-27-35-44-66-58-79

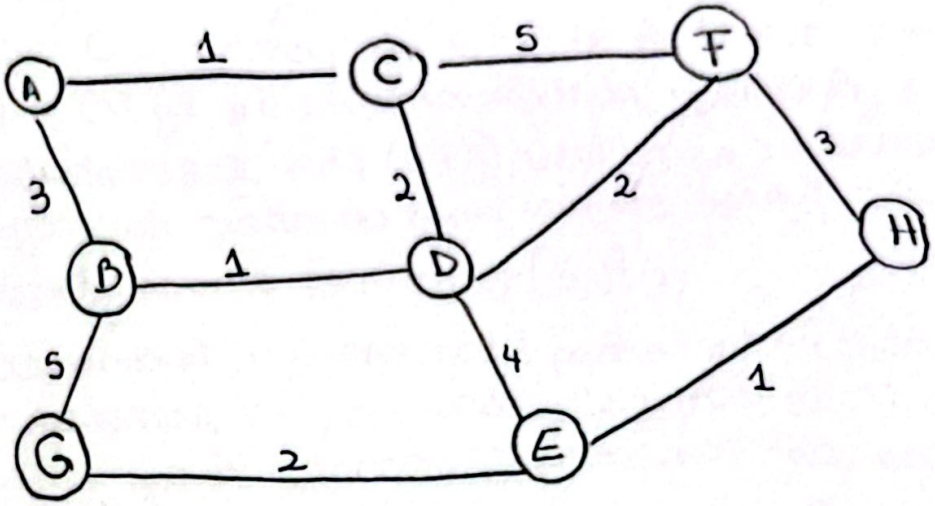
b) Inorden I-R-D

2-5-6-9-12-16-19-20-24-27-35-39-44-50-58-66-79

c) PostOrden I-D-R

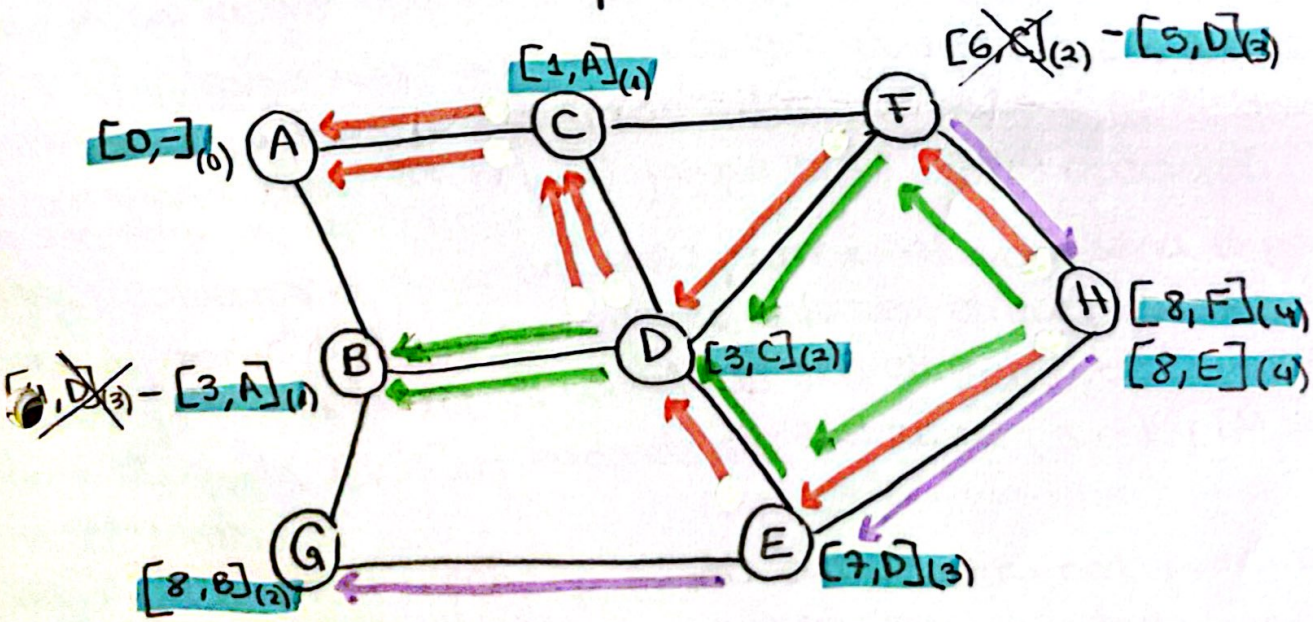
2-6-9-5-16-20-19-12-35-27-44-39-58-79-66-50-24

Ejercicio 2



metodo de Dykstra. ⟶ etiqueta [D, N](0) → (numero de operaciones)
→ (nodo anterior)
→ (distancia acumulada)

entonces obteniendo la etiqueta de cada tramo nuestro grafo ponderado quedaria así.



para la primera paute A sera nuestro nodo permanente, entonces obtendremos las etiquetas de B y C que estan conectados con A, en donde de A-B la distancia AC sera (0+3) y (1) ya que es la primera operación, y de A-C la dist AC sera (0+1) y (1) ya que tambien es la primera operación quedando como etiqueta las siguientes para B y C respectivamente, [3,A](1) y [1,A](1).

Luego nuestro sig nodo permanente sería C, entonces debemos obtener la etiqueta de F y D que son aquellos que estan conectados con C, en donde de C-F la distancia acumulada sería (1+5) y (2) ya que sería la segunda operación, y en el caso de C-D la distancia ac sería (1+2) y (2) ya que tambien sería la segunda operación, quedando como etiqueta para F y G los siguientes respectivamente $[6,C]_{(2)}$ y $[3,C]_{(2)}$.

Se sigue buscando el siguiente nodo permanente, viendo cual tiene menor distancia ac, por ende elegimos el D. aunque tenga la misma que B, entonces etiquetaremos tres nodos ya que se conecta a B, F y E, empezaremos con B, en donde de D-B la distancia ac es (1+3) y (3) ya que es la tercera operación quedando como etiqueta $[4,D]_{(3)}$ la cual es mayor a la que el nodo B ya tiene por ende no cambiamos la anterior. Luego para el nodo F quedaría lo siguiente, su dist ac es (3+2) y (3) ya que es la tercera operación, quedando como etiqueta lo siguiente $[5,D]_{(3)}$ como esta es menor a la que ya tiene ese nodo, la anterior se cambia a esta, y por ultimo al nodo E la distancia acumulada sería (3+4) y (3) ya que es la tercera operación. y su etiqueta sería $[7,D]_{(3)}$.

Luego volvemos a elegir un nodo permanente, que sería B ya que tiene la menor distancia acumulada, entonces etiquetamos solo el nodo G ya que es el unico conectado sin etiquetar, obteniendo como etiqueta $[8,B]_{(2)}$, en donde su dist ac es (3+5) y (2) sería su operación.

Luego volvemos a repetir lo mismo con F como nodo permanente en donde se conecta con H y D, pero D ya esta etiquetado, por ende H quedaría etiquetado así $[8,F]_{(4)}$ con (3+5) su dist ac y (4) su operación.

Por ultimo el nodo permanente es E el cual ya esta etiquetado, como el nodo G que es uno de los conectados a E no mejora su etiqueta, ya que la dist ac sería mayor, nos enfocamos en H dando como etiqueta $[8,E]_{(4)}$, como tiene la misma distancia ambas etiquetas de H, dejamos las dos, provocando que hayan dos rutas.

Explicado lo anterior y teniendo todos los nodos etiquetados, las rtas mínimas para las siguientes pares de puntos es:

a) Inicio en A y termino en H.

La distancia entre ellos es 8 y sus posibles caminos de adelainte hacia atras.

camino (1) = A → C → D → F → H

camino (2) = A → C → D → E → H

● Inicio en A, termino en H

b) Inicio en B y termino en H.

La distancia entre ellos es de 6 para ambos caminos posibles.

camino (1) = B → D → F → H

camino (2) = B → D → E → H

● Inicio en B, termino en H

c) Inicio en G y termino en F

La distancia entre ellos es de 6 y su camino es

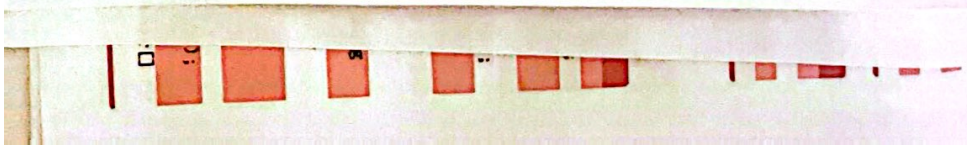
G → E → H → F

● Inicio en G, termino en F

hay otro camino G - E - D - F pero la distancia entre ellos sera 8 que es mayor a 6, y se piden las rtas minimas.

```
#include <iostream>
#include <map>
#include <vector>
#include <utility>
using namespace std;
```
(librerias)

```
map<string,vector<string> > generarAcademia(string *discipulos,int N){ // funcio
n generAcademia que recibe un arreglo, en donde N es el numero de discipulos y r
etorna un map.
map<string,vector<string> > academia; // mapa que se retornara a partir de la fu
ncion creada anteriormente.
vector<string> maestros; // se crea un vector de maestros.
    maestros.push_back("Goku"); //agregar maestros en la ultima posicion al vect
or maestros.
    maestros.push_back("Vegeta");
    maestros.push_back("Krilin");
    string nuevoMaestro; // creo un maestro por si existen mas de 15 discipulos.
    int MaestroActual=0; // se comienza con el primero, osea GOKU maestro actual
.
    for(int i = 0; i < N;i++){ // con el for recorro los discipulos y se asigna
a cada maestro.

     if(academia.count(maestros[MaestroActual]) == 0){ //verifica si hay una lla
ve en el mapa
            academia[maestros[MaestroActual]].push_back(discipulos[i]);
        }
      else if(academia[maestros[MaestroActual]].size() == 5){ //si se llega al ma
ximo de discipulos se debe cambiar de maestro.
            MaestroActual++; // se aumenta el MaestroActual para pasar al proxim
o maestro.
            if(MaestroActual >= maestros.size()){ //si se llega alfinal de los m
aestros, se debe agregar otro.
                cout<<"Ingrese nombre del nuevo maestro"<<endl; // se pide que s
e ingrese y con un cin se guarda en nuevoMaestro.
                cin>>nuevoMaestro;
                while(academia.count(nuevoMaestro) > 0){ //si ya existe el maest
ro, se pide que se ingrese otro nombre.
                    cout<<"Ya existe este maestro!"<<endl;
                    cout<<"Ingrese nombre del nuevo maestro"<<endl;
                    cin>>nuevoMaestro;
                }
                maestros.push_back(nuevoMaestro); //se agrega el nombre del maes
tro nuevo y se empiezan a agregar discipulos a el.
            }
            i--; //se resta uno ya que el ultimo discipulo no se ha asignado, so
lo se cambia el maestro
        }
        else{ //sino termina se agrega el discipulo al maestro
            academia[maestros[MaestroActual]].push_back(discipulos[i]);
        }
    }
    return academia;
}

void imprimirMapa( map<string,vector<string> > academia){  //funcion que imprime
 el vector academia completo en el formato del ejemplo {maestro,[discipulo1]}
    cout<<"{";
    for(map<string,vector<string> >::iterator iterador = academia.begin(); itera
dor != academia.end(); iterador++){ //recorre el mapa desde inicio hasta el fin
con un iterador, imprimiendo la llave (first) que en este caso es el entrenador.
        cout<<iterador->first<<":[";//se imprime el mapa con el mismo formato de
l ejemplo.. "maestro":[].
        for(int i = 0; i < iterador->second.size();i++){ //se recorre el vector
de discipulos por cada maestro y entrega el vector asociado a la llave(second) s
```



```
iendo el discipulo este.
            if(i+1 != iterador->second.size()) // un if para verificar si es el
ultimo maestro, si lo es se cierra con la llave ] y sino con un , ya que hay mas
.cac
                cout<<iterador->second[i]<<",";
            else
                cout<<iterador->second[i]<<"]";
        }
        //se crea otro iterador para saber si es el final y terminar el formato
de ejemplo, sino agrega otra coma ya que sigue.
        map<string,vector<string> >::iterator iterador2 = iterador;
        iterador2++;
        if(iterador2 == academia.end())
            cout<<"}"<<endl;
        else
            cout<<",";
    }
}
//se recrean los ejemplos entregados en el enunciado.
int main(){
    string discipulos[] = {"Caro","Pipe","Ricardo","Pamela","Tito","Lalo","Maca"
,"Fan","Tania","Eduardo","Tom","Max"};
    map<string,vector<string> > academia = generarAcademia(discipulos,12);
    imprimirMapa(academia);
    string discipulos2[] = {"Caro","Pipe","Ricardo","Pamela","Tito","Lalo","Maca
","Fan","Tania","Eduardo","Tom","Max","Pepe","Walala","Chaos","Beerus"};
    academia = generarAcademia(discipulos2,16);
    imprimirMapa(academia);
    return 0;
}
```

Torpedo básico de Java

Literales

Enteros	
binario	`0b11111111`
binario	`0B11111111`
octal	`0377`
decimal	`255`
hexadecimal	`0xff`
hexadecimal	`0xFF`

Reales	
Precisión simple (con 'f')	`88.0f / 88.1234567f`
Precisión doble (sin 'f')	`88.0 / 88.123456789012345`
Signo positivo	`42 / +42`
Signo negativo	`-42`

Otros	
Valore lógicos	`true / false`
Carácteres	`'a'`
Cadenas	`"texto"`
Referencia vacía (objetos)	`null`
Indica que el método no devuelve nada	`void`

Comentarios	
Una línea	`//Todo acá es sólo un comentario`
Múltiples líneas	`/*Un comentario puede tomar varias lineas.*/`

Tipos primitivos

Números		
$(-2^7)-(2^7-1)$	1	`byte`
$(-2^{15})-(2^{15}-1)$	2	`short`
$(-2^{31})-(2^{31}-1)$	4	`int`
$(-2^{63})-(2^{63}-1)$	8	`long`
$+/-(3,4*10^{-38}-3,4*10^{38})$	4	`float`
$+/-(1,7*10^{-308}-1.7*10^{308})$	8	`double`

Otros	
Posibles valores	Tipo
`true` o `false`	`boolean`
Caracter en UTF-16, comillas simples	`char`
Cadena (texto), comillas dobles	`String`
Conversión de tipo (a `type`)	`(type)a`

Operadores

Aritméticos	
Suma	`a + b;`
Resta	`a - b;`
División	`a / b;`
Multiplicación	`a * b;`
Modulo	`a % b;`
Incrementa antes de dar valor	`++a;`
Decrementa antes de dar valor	`--a;`
Incrementa después de dar valor	`a++;`
Decrementa después de dar valor	`a++;`

Binarios	
o / disyunción inclusiva / OR	`a | b;`
"o bien" / disyunción exclusiva / XOR	`a ^ b;`
y / conjunción / AND	`a & b;`
Complemento binario	`~a;`
"signed left shift"	`bitVal << numPos;`
"signed right shift"	`bitVal >> numPos;`
"unsigned right shift"	`bitVal >>> numPos;`

Lógicos	
o / disyunción inclusiva / OR	`a || b;`
y / conjunción / AND	`a && b;`
no / negación / NOT	`!a;`

Relacionales	
Igual	`a == b;`
Diferente	`a != b;`
Mayor	`a > b;`
Mayor o igual	`a >= b;`
Menor	`a < b;`
Menor o igual	`a <= b;`

Palabras Reservadas en Java

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

Variables

Declaración	
Variable	`int x;`
Variable inicializada	`char x = 'C';`
Varias del mismo tipo	`float x, y, z;`
Constante	`final int M = 88;`
Siempre en la memoria compartida	`volatile int x = 88;`

Nombres	
Bien: Alfanumérico, no palabra reservada, empieza con letra	`buenEjemplo1`
Bien: Empieza con '_'	`_buenEjemplo2`
Bien: Empieza con '$'	`$buenEjemplo3`
Mal: Empieza con un dígito	`1malEjemplo1`
Mal: palabra reservada	`while`
Mal: no alfanumérico	`mal ejemplo 3`

Convenciones	
Palabras clave - minúsculas	`true`
Variabes - "camel case"	`buenaVariable`
Clases - iniciales mayúsculas	`BuenaClase`
Constantes - mayúsculas	`BUENA_CONSTANTE`

String

Declaración y uso	
Declara un String	`String texto;`
Declara un String e inicializa	`String texto = "Hola";`
Toma el tamaño	`int tam = texto.length();`
Lee un carácter	`char c = texto.charAt(int pos);`
Compara lexicograficamente `this` con `otro`	`int compareTo(String otro);`
Da primera posición de `otro` dentro de `this` posterior a `desde`	`int indexOf(String otro, int desde));`
Separa `this` por `sep`	`String[] split(String sep);`
Crea String copiando del `this` la parte entre `ini` y `fin`	`String substring(int ini, int fin);`
Crea String concatenando elementos de `col` separados por `sep`	`join(String sep, Iterable col);`
No se puede modificar carácteres dentro de un String	
Convierte `this` a un arreglo de `char`	`char[] toCharArray();`
Convierte un arreglo de `char` a String	`String valueOf(char[] data);`

Arreglos

Declaración	
Declara un arreglo	`type name[];`
Declara un arreglo e inicializa	`type name[] = {x, y, z};`
Arreglo de 2 dimensiones	`type name[][] = {{a}, {b,c}, {d, e, f}};`

Uso	
Crea un nuevo arreglo e inicializa con ceros	`int vec [];` `vec = new int[5];`
Cambia un valor	`vec[0] = 7;`
Lee un valor	`val = vec[0];`
Toma el tamaño (1. dimensión)	`int tam;` `tam = vec.length;`
Crea un arreglo rectangular	`int[][] mat = new int[3][4];`
Toma el tamaño (2. dimensión)	`int tam;` `tam = mat[1].length;`
Crea una copia (superficial) de arreglo	`Arrays.copyOfRange( type[] original, int from, int to)`
Verifica igualdad (superficial) de arreglos	`Arrays.equals(type[] a1, type[] a2)`
Verifica igualdad (profunda) de arreglos	`Arrays.deepEquals( Object[] a1, Object[] a2)`
Crea representación String (superficial) del arreglo	`Arrays.toString( type[] a)`
Crea representación String (profunda) del arreglo	`Arrays.deepToString( Object[] a)`

Condicionales

if, else if, else	
Hace b si a es verdadera	`if(a) b;`
Hace b y c si a es verdadera	`if(a) { b; c; }`
Hace b si a, si no, hace c	`if(a) { b; } else { c; }`
Hace b si a, si no, hace d si c, si no, hace e	`if(a) { b; } else if(c) { d; } else { e; }`

switch, case, break	
Hace c si a vale b	`switch(a){ case b: c; }`
Hace b si a no cumple con otro caso	`switch(a){ default: b; }`
Hace d si a vale b o c	`switch(a){ case b: case c: d; }`
Hace c, e y f si a vale b, si no, hace e y f si a vale d, si no, hace f.	`switch(a){ case b: c; case d: e; default: f; }`
Hace c si a vale b, e si a vale d y e en otros casos	`switch(a){ case b: c; break; case d: e; break; default: f; }`

Ciclos

while

```
int x = 0; while(x < 10){ x += 2; }
```

Itera mientras sea verdadera la condición de guarda

Preparar las variables usadas en el ciclo	`int x = 0;`
Condición de guarda va entre paréntesis	`while()`
Condición - se evalúa al inicio de cada iteración	`x < 10`
Las llaves delimitan el cuerpo del ciclo	`{}`
El cuerpo - el código controlado por el ciclo	`x += 2;`

do while

```
char c = 'A'; do { c++; } while(c != 'Z');
```

Vuelve a iterar mientras sea verdadera la condición de guarda

Preparar las variables usadas en el ciclo	`char c = 'A';`
Marca el inicio del ciclo	`do`
Las llaves delimitan el cuerpo del ciclo	`{}`
El cuerpo - el código controlado por el ciclo	`c++;`
Condición de guarda va entre paréntesis	`while();`
Condición - se evalúa al final de cada iteración	`c != 'Z'`

for

```
for(int i = 0; i < 10; i++){}
```

Las instrucciones de control van entre paréntesis	`for()`
Declara e inicializa el "cursor" del ciclo	`int i = 0;`
Condición - se evalúa al inicio de cada iteración	`i < 10;`
Instrucción ejecutada al final de cada iteración	`i++`
Las llaves delimitan el cuerpo del ciclo	`{}`

continue, break

```
int i=0; while(i<10){ i++; continue; i-;}
```

Se salta el resto del cuerpo y pasa a la siguiente iteración

```
int i=0; while(1){ if(x==10)break; i++; }
```

Se salta el resto del cuerpo y termina el ciclo

Utilidades

Random

Generar una instancia de la clase random	`Random random = new Random();`
Generar un int entre 0 y valor	`int nextInt(int valor);`
Generar un long entre 0 y valor	`long nextLong(long valor);`
Generar un double entre 0 y 1	`double nextDouble();`
Generar un float entre 0 y 1	`float nextFloat();`
Generar un int[] de cant enteros entre mini y maxi al azar	`int[] ints(cant, mini, maxi).toArray();`

longs, doubles operan de forma parecida

Math

Valor absoluto	`type abs(type a);`
Redondear hacia arriba	`double ceil(double a);`
Redondear hacia abajo	`double floor(double a);`
Potencia base *e*	`double exp(double a);`
Potencia base *a*, exponente *b*	`double pow(double a, double b);`
Raíz cuadrada	`double sqrt(double a);`
Logaritmo base *e*	`double log(double a);`
Logaritmo base 10	`double log10(double a);`
Máximo	`type max(type a, type b);`
Mínimo	`type min(type a, type b);`
Seno	`double sin(double a);`
Coseno	`double cos(double a);`
Tangente	`double tan(double a);`

Scanner

Verifica si hay un elemento que leer	`boolean hasNext();`
Lee un String (hasta el separador)	`next();`
Lee un entero	`int nextInt();`
Lee un long	`long nextLong();`
Lee un float	`float nextFloat();`
Lee un double	`double nextDouble();`
Lee una línea completa (hasta el salto de línea)	`String nextLine();`
Verifica si hay una línea disponible	`boolean hasNextLine();`
Genera una instancia de Scanner para leer de stdin	`Scanner input = new Scanner(System.in);`

Formatter

Escribir los datos proporcionados en `args` según el `formato` indicado	`Formatter format(String formato, Object... args);`
Genera una instancia de Formatter para escribir a stdout	`Formatter output = new Formatter(System.out);`
También se puede escribir a stdout directamente	`System.out.println(k)`

Colecciones

Collection<E>

Agregar el elemento `e` a `this`	`boolean add(E e);`
Eliminar el elemento `o` de `this`	`boolean remove(Object o);`
Ver si `this` contiene el elemento `o`	`boolean contains(Object o);`
Ver el tamaño de `this`	`int size();`
Vaciar `this`	`void clear();`

Set<E>	
Tiene todo de Collection<E>	
Declaración e inicialización de un set vacío como HashSet	`Set<E> set = new HashSet<>();`
Declaración e inicialización de un set vacío como TreeSet	`Set<E> set = new TreeSet<>();`

List<E>	
Tiene todo de Collection<E>	
Insertar el elemento e en la posición pos en this	`void add(int pos, E e);`
Obtener el elemento en la posición pos en this	`E get(int pos);`
Reemplazar el elemento en la posición pos en this por e	`E set(int pos, E e);`
Obtener la primera posición de e en this	`int indexOf(int pos);`
Obtener una vista de la parte de la lista this entre las posiciones ini y fin	`List<E> subList(int ini, int fin);`
Declaración e inicialización de una list vacía como una ArrayList	`List<E> list = new ArrayList<>();`
Declaración e inicialización de una list vacía como una LinkedList	`List<E> list = new LinkedList<>();`

Deque<E>	
Tiene todo de Collection<E>	
Insertar el elemento e al inicio de this	`boolean offerFirst(E e);`
Insertar el elemento e al final de this	`boolean offerLast(E e);`
Copiar el elemento e al inicio de this	`E peekFirst(E e);`
Copiar el elemento e al final de this	`E peekLast(E e);`
Sacar el elemento e al inicio de this	`E pollFirst(E e);`
Sacar el elemento e al final de this	`E pollLast(E e);`
Declaración e inicialización de una deque vacía como una ArrayDeque	`Deque<E> deque = new ArrayDeque<>();`
Declaración e inicialización de una deque vacía como una LinkedList	`Deque<E> deque = new LinkedList<>();`

Map<K,V>	
Agregar a this el valor asignado a clave	`V put(K clave, V valor);`
Copiar de this el valor asignado a clave	`V get(Object clave);`
Reemplazar en this el valor asignado a clave por valor	`default V replace(K clave, V valor);`
Sacar de this el elemento identificado por clave	`V remove(K clave);`
Ver si this contiene o como una clave	`boolean containsKey(Object o);`
Ver si this contiene o como un valor	`boolean containsValue(Object o);`
Obtener una vista del conjunto de claves en this	`Set<K> keySet();`
Obtener una vista de la colección de valores en this	`Collection<V> values();`
Ver el tamaño de this	`int size();`
Vaciar this	`void clear();`
Declaración e inicialización de una map vacía como HashMap	`Map<K,V> map = new HashMap<>();`
Declaración e inicialización de una map vacía como TreeMap	`Map<K,V> map = new TreeMap<>();`

Un ejemplo de implementación de una clase

```
public class HolaMundo {
    String nom;
    HolaMundo(String aNom) {
        this.nombre = aNom;
    }
    public String saluda(String aNom) {
        return "Hola " + aNom + ", soy " + this.nom;
    }
    public static void main(String[] args) {
        HolaMundo hm = new HolaMundo("Eduardo");
        System.out.println(hm.saluda("Javiera"));
    }
}
```