UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

ASIGNATURA: ESTRUCTURA DE DATOS Y ALGORITMOS

ACTIVIDAD acordeón java

NOMBRE DEL ALUMNO: RIVERA VARGAS DONOVAN JOSHEP

15/03/2021

LENGUAJE CON LA LETRA DE MI NOMBRE

JOSHEP = JAVA

FORMULARIO JAVA

las librerías de Java.

```
1    import java.Bla.Bla
```

TIPOS DE DATOS

- **Tipos primitivos**, que se corresponden con los tipos de variables en lenguajes como C y que son los datos elementales que hemos citado.
- **Los Tipos Objeto**, son mucho más inteligente y te permiten realizar muchas cosas más fácilmente.
-

  También podemos dividir los atributos en dos tipos:

  - **Estáticos** *(es decir que atributos que nunca van a cambiar, también los conocen como MACROS) .*
  - **Dinámicos** *(es un atributo que puede variar, así que lo consideramos, valga la originalidad, solo un atributo).*

Nombre	Tipo	Ocupa	Rango
byte	Entero	1 byte	-128 a 127
short	Entero	2 bytes	-32768 a 32767
int	Entero	4 bytes	2*109
long	Entero	8 bytes	Muy grande
float	Decimal simple	4 bytes	Muy grande
double	Decimal doble	8 bytes	Muy grande
char	Carácter Usa comillas simples	2 bytes	—
boolean	Valor:	1 byte	—

Sintaxis

```
1  //FORMA BASICA
2  TipoDeDato NombreDeDato;
3
4  //FORMA DECLARAR VARIAS
5  TipoDeDato Dato1, Dato2, Dato3;
6
7  //FORMA DECLARAR Y INICIALIZAR
8  TipoDeDato Dato1 = Valor;
9  TipoDeDato Dato2 = Valor2, Dato3 = Valor3;
```

Tipos Objeto

Con métodos, necesitan una invocación para ser creados.

Tipos de la biblioteca estándar de Java	**String** (cadenas de texto) Muchos otros (p.ej. Scanner, TreeSet, ArrayList...)
Tipos definidos por el programador	Cualquiera que se nos ocurra, por ejemplo Taxi, Autobus, Tranvia.
Arrays	Serie de elementos o formación tipo vector o matriz. Lo consideraremos un objeto especial que carece de métodos.
Tipos envoltorio o wrapper (Equivalentes a los tipos primitivos pero como objetos...Cool)	Byte
	Short
	Integer
	Long
	Float
	Double
	Character

Operadores

Matemáticos

Símbolo	Operación
+	**Suma**
–	**Resta**
*	**Multiplicación**
%	**Módulo** (Resto de división)

Lógicos

Operador	Significado
!	NOT
>	Mayor que
<	Menor que
==	Igual
&&	AND
||	OR
^	XOR
>=	Mayor o igual que
<=	Menor igual que
!=	No igual

Simplificaciones

Forma Simple	Forma Compleja
x += y	x = x+y
x -= y	x = x-y
x *= y	x = x*y
x /= y	x = x/y
x++	x = x+1
x–	x = x-1

Diferentes sintaxis de IF

```
//FORMA BÁSICA
if(condicion){
  sentenciasSiEsVerdad;
}

//COMO OPERADOR TERNARIO
(condicion)? sentenciaVerdad:sentenciaFalsa;

//FORMA IF-ELSE
if(condicion){
  sentenciasSiEsVerdad;
}
else{
  sentenciasSiEsFalso;
}

//FORMA IF-ELSE IF- ELSE
if(condicion1){
  sentenciasSiEsVerdad1;
}
else if(condicion2){
  sentenciasSiEsVerdad2;
}
else if(condicion3){
  sentenciasSiEsVerdad3;
}
else{
  sentenciasSiEsFalsoTodo;
}
```

Switch

Son básicamente igual a los if, simplemente que con otra sintaxis (si, se que hay más diferencias técnicas pero no importa), aquí su sintaxis:

```
switch (variable){
  case const1:
    sentencia;
    ...
    break;

  case const2:
    sentencia;
    ...
    break;

  ...

  default:
   sentencia;
}
```

FOR

Este es lo mismo que el bucle while, simplemente con otra sintaxis, es más esta es la sintaxis:

```
for (inicialización; condición; incremento){
  sentencia;
}
```

Así el último código que vimos con los whiles, se vería así con for:

```
for (i=0; i < tope; i++){
  sentencia;
}
```

DO WHILE

Al contrario que los bucles for y while que comprueban la condición en lo alto de la misma, el bucle do/while comprueba la condición en la parte baja del mismo, lo cual provoca que el bucle se ejecute como mínimo una vez. La sintaxis del bucle do/while es:

```
do{
  sentencia;
}
while(condicion);
```

WHILE

Este es sin menor duda la forma más básica de un bucle, lo único que hace es que ejecuta un bloque de código una y otra vez mientras una condición sea verdad.

```
while (condicion){
  sentencias;
}
```

No puede ser tan fácil, ¿o sí?...
La verdad es que lo es, es por eso que es el bucle más poderoso, pero la verdad es que mucha gente lo acaba usando es un estilo más o menos así:

```
int i = 0;              //Este es nuestro contador
while (i < tope){       //Lo vamos a seguir haciendo tope vece
  sentencias;           //Ejecutamos algo
i++;                    //Aumentamos el contador en uno
}
```

Break y Continue

Las sentencias de control break y continue permiten modificar y controlar la ejecución de los bucles anteriormente descritos.

- La sentencia **break** provoca la salida del bucle en el cual se encuentra y la ejecución de la sentencia que se encuentra a continuación del bucle.
- La sentencia **continue** provoca que el programa vaya directamente a comprobar la condición del bucle en los bucles while y do/while, o bien, que ejecute el incremento y después compruebe la condición en el caso del bucle for.

Se puede usar Print para imprimir o Println para añadir además una nueva línea al final.

```
System.out.println("Hola Mundo");
```

Para mostrar el valor de los atributos se usa un + :

```
System.out.println("El valor de X es "+x);
System.out.println("El valor de X es "+x+" y el de Y: "+y);
```

Es una clase y permite obtener información desde el teclado y lo mete dentro de un atributo.

```
//Primero se importa:
import java.util.Scanner;

//Crear un objeto de la Clase Scanner:
Scanner NombreDelObjeto = new Scanner(system.in);

//Obtener datos con Scanner:
Atributo1 = NombreDelObjeto.nextDato();
```

Depende del tipo de dato que intentes ser el método que vas a usar:

Y **nextDato** será dependiendo del tipo de dato uno de los siguientes:

- **Int – NextInt();**
- **Double – NextDouble();**
- **String – NextLine();**

Permite generar un número aleatorio.

```
//Primero se importa:
import java.util.Random;

//Crear un objeto de la Clase Random:
private Random NombreDelObjeto = new Random(system.nanotime()

//Obtener un Número Random entre 0 y X:
Atributo1 = NombreDelObjeto.nextInt(X);
```

Arrays o Vectores

Un vector sería algo muy similar a una lista, ademas que siempre se usan junto con bucles. En Java, un array unidimensional se declara como:

```
tipo[] nombreVector = new tipo[tamaño];
```

En Java, el primer elemento de un array es el que posee el índice 0, por lo tanto, un array de 20 elementos posee sus elementos numerados de 0 a 19.

Métodos

En Java los vectores son instancias de una clase, por lo tanto tienen diversos **métodos** muy útiles:

```
//Longitud del Vector
nombreVector.length()
```

String

Un string en Java es mucho más fácil de manejar que en C, incluso se puede:

- Comparar directamente
- Añadir más frases con algo tan simple como **String + "bla"**

Sintaxis:

```
String nombreDelString;
```

Métodos

En Java los Strings son instancias de una clase, por lo tanto tienen diversos **métodos** muy útiles:

```
//LONGITUD
nombreDelString.length()

//REGRESAR CHAR EN X POSICION
nombreDelString.CharAt(x)
```

Matrices

Es de la forma más sencilla un array de arrays

Sintaxis:

```
//DECLARACIÓN
tipo[][]nombreMatriz = new tipo[tamañoEnX][tamañoEnY];

//INICIALIZACION
tipo[][]nombreMatriz = {
   {1,2,3}
   {4,5,6}
   …
   {7,8,9}
};
```

Métodos

En Java las Matrices son instancias de una clase, por lo tanto tienen diversos **métodos** muy útiles:

```
//TAMAÑO EN X
nombreMatriz.length

//TAMAÑO EN Y
nombreMatriz[0].length
```

Tipo Vector

Sintaxis:

```
Vector<Objeto> Nombre = new Vector<Objeto>();
```

Métodos

En Java los vectores son instancias de una clase, por lo tanto tienen diversos **métodos** muy útiles:

```
//AÑADE AL FINAL DEL VECTOR
nombreVector.add(Atributo)
```

Tipo ArrayList

Las arraylist son como unos arrays en esteroides , nos permiten hacer mas cosas.

Sintaxis:

```
ArrayList<Objeto> Nombre = new ArrayList<Objeto>();

//O crearlo con una longitud
ArrayList<Objeto> Nombre = new ArrayList<Objeto>(Longitud);
```

Manejo de ArrayList

En Java los ArrayList son instancias de una clase, por lo tanto tienen diversos **métodos** muy útiles:

```
//Tamaño
listaNumeros.size();

//Obtener un Elemento
listaNumeros.get(i);

//Añade en Posición
listaNumeros.add(int index, Object Elemento);

//Clona un ArrayList
listaNumeros.clone();

//Coloca un Elemento en cierto Lugar
listaNumeros.set(int index, Object Elemento);

//Regresa su forma de Array
listaNumeros.toArray();
```

```
//SINTAXIS SIMPLE
try {
   CodigoProblematico
   ...
}
catch(Exception Nombre){
   CodigoSolucion
}

//SINTAXIS COMPLEJA
try {
   CodigoProblematico
   ...
}
catch(TipoDeErrores Nombre){
   CodigoSolucion1
}
catch(Exception Nombre){
   CodigoSolucion2
}
finally{
   Código que siempre se ejecuta
}
```