

TECNICATURA
UNIVERSITARIA
EN PROGRAMACIÓN
UTN-FRC



UTN 
Facultad Regional Córdoba

TECNICATURA UNIVERSITARIA EN PROGRAMACIÓN

LABORATORIO DE COMPUTACIÓN III

Unidad Temática 1: Introducción al Lenguaje Java

Material de Estudio

2^{do} Año - 3^{er} Cuatrimestre

2020



V.0.1

Índice

1. INTRODUCCIÓN AL LENGUAJE JAVA	2
Lenguaje Java	2
Sintaxis de Java.....	5
2. PROGRAMACIÓN ORIENTADA A OBJETOS.....	17
Introducción	17
Bloque de clase	17
Miembros	18
Atributos.....	18
Métodos	19
Parámetros	19
Propiedades.....	20
Modificadores de acceso	20
Constructores	22
Objetos y referencias	23
Cuadro comparación clase Persona entre C# y Java	24
BIBLIOGRAFÍA.....	30

1. INTRODUCCIÓN AL LENGUAJE JAVA

Lenguaje Java

Java es un lenguaje de programación de alto nivel con el que se pueden escribir tanto programas convencionales como para Internet.

Ceballos (2010) describe como su principal característica que:

“Una de las ventajas significativas de Java sobre otros lenguajes de programación es que es independiente de la plataforma, tanto en código fuente como en binario. Esto quiere decir que el código producido por el compilador Java puede transportarse a cualquier plataforma que tenga instalada una máquina virtual Java y ejecutarse. Pensando en Internet esta característica es crucial ya que esta red conecta computadoras muy distintas. En cambio, C++, por ejemplo, es independiente de la plataforma solo en código fuente, lo cual significa que cada plataforma diferente debe proporcionar el compilador adecuado para obtener el código máquina que tiene que ejecutarse.

En base a eso, Java incluye dos elementos: un compilador y un intérprete. El compilador produce un código de bytes que se almacena en un fichero para ser ejecutado por el intérprete Java denominado máquina virtual de Java. “

Los códigos de bytes o *bytecodes* son instrucciones redactadas en lenguaje de máquina similares a un programa ejecutable, pero no correspondientes a la computadora o sistema operativo que haya usado el programador, sino a una computadora ideal ficticia pero similar a las actuales. Cuando el programa escrito en esta máquina ideal intenta ser ejecutado en una computadora concreta se realiza una segunda traducción entre los *bytecodes* y las instrucciones reales de la computadora. Esta traducción es realizada por un programa denominado “Máquina virtual de Java”. Por cada sistema operativo diferente existe una máquina virtual que realiza esta traducción. Gracias a esto cualquier programa Java puede ser ejecutado en distintos sistemas operativos sin necesidad de intervención por parte del programador. Inclusive, para cualquier sistema operativo que vaya a ser creado en el futuro, simplemente con la creación de una máquina virtual de java para dicho SO, todos los programas existentes van a poder ser ejecutados sin modificación ni recompilación.

Instalación de Java

Para poder empezar a programar en Java se necesita instalar dos paquetes de software relativamente independientes. Por un lado es imprescindible instalar el JDK (Kit de desarrollo de Java), el cual contiene todas las herramientas necesarias para desarrollar todo tipo de programas. Dentro del JDK se encuentra el compilador, el intérprete (es decir, la máquina virtual), la biblioteca de clases y otras herramientas útiles.

Sin embargo el JDK no incluye un entorno de desarrollo (IDE) que nos ofrezca un editor de código adaptado a java y acceso a las herramientas imprescindibles como el compilador y el depurador. Sin un IDE, la única alternativa que dispone un programador es la de escribir los programas en un editor de texto y luego por la línea de comandos del sistema operativo invocar al compilador y luego a la máquina virtual.

Sin embargo, esta idea tiene un sentido muy útil. Al no haber un único IDE para el lenguaje, diversas organizaciones programaron una gran cantidad de estas herramientas para que cada programador elija la que mejor se adapte a sus necesidades. Además, dado que estos diferentes IDE compiten por ser los más usados, continuamente van buscando mejorarse y adecuarse a las nuevas versiones del lenguaje. Actualmente existen varios que se destacan, entre ellos se encuentran NetBeans, Eclipse, IntelliJ IDEA y otros.

Para descargar el JDK se puede visitar la página siguiente:

<https://www.oracle.com/technetwork/java/javase/downloads/index.html>

Para descargar NetBeans se ingresa a la siguiente página:

<https://netbeans.apache.org/>

Para desarrollar un programa hay que seguir los siguientes pasos:

1. Editar el programa
2. Compilarlo
3. Ejecutarlo
4. Depurarlo

Programa mínimo

Todo programa Java debe poseer como mínimo las siguientes líneas de código:

```
class HolaMundo
{
    public static void main(String[] args)
    {
        System.out.println("Hola mundo!!!");
    }
}
```

Como se evidencia todo el programa debe estar programado dentro de un bloque de código cuya cabecera indica “class HolaMundo”. La palabra class indica el inicio de una clase como en cualquier otro lenguaje orientado a objetos, como C#. El identificador a continuación define el nombre de dicha clase. En Java todo el código va a pertenecer a una clase aun cuando no se esté realizando programación orientada a objetos. En las próximas secciones se avanzará en esta temática.

Java exige que cada clase se encuentre almacenada en un archivo cuyo nombre coincida exactamente (incluyendo mayúsculas y minúsculas) con el nombre de la clase, por lo tanto el código anterior debe ser almacenado en un archivo HolaMundo.java.

Dentro de la clase se encuentra un método denominado main, el cual indica el punto de entrada, es decir, el inicio del programa. Desde este método se pueden crear objetos y llamar a otros métodos. Cuando el método main finaliza, el programa se termina y se descarga de la memoria.

Para presentar una línea de texto en la pantalla se utiliza la instrucción *System.out.println*, la cual recibe como parámetro una cadena con el texto que se desea presentar en la pantalla. El parámetro en el ejemplo anterior es una cadena fija (una constante de tipo *string*), pero también pueden pasarse variables para que se muestre su valor, o expresiones para que se muestre el resultado de las operaciones involucradas. Y si el tipo de datos de tales variables o expresiones no es String, también se muestra el dato sin necesidad de realizar operaciones de conversión.

Ejecución

Para poder ejecutar el programa se requiere realizar ambos pasos de traducción indicados anteriormente. Para realizar la compilación y generación del

código intermedio (los bytecodes) se invoca al programa compilador indicando el nombre del archivo de código fuente:

```
javac HolaMundo.java
```

Si la compilación finaliza sin errores, el compilador no muestra ningún mensaje y genera un archivo con el mismo nombre pero con extensión `.class`. Ese archivo contiene el código de bytes y puede ser leído por la máquina virtual para ejecutar el programa. La máquina virtual consiste en un programa llamado `java` el cual debe ser invocado indicando el nombre de la clase que contiene el punto de inicio, es decir, el método `main`:

```
java HolaMundo
```

Al ejecutarse el programa se presentan en la consola todos los datos impresos mediante el método `println`:

```
Hola mundo!!!
```

Sintaxis de Java

La sintaxis de Java es muy similar a la de otros lenguajes tales como C++ o C#:

- Cada instrucción debe finalizar con ;
- Las estructuras de control (condicionales y repetitivas) no requieren punto y coma porque poseen un bloque delimitado por llaves {}
- Cuando se indica un bloque todo su contenido debería escribirse con mayor indentación o sangría.
- Java posee tres tipos de comentarios:
 - // comentarios para una sola línea
 - /* comentarios de una o más líneas */
 - /** comentario de documentación, de una o más líneas */
- Los comentarios de documentación se los escribe antes del inicio de una clase o método y son extraídos por una herramienta automática llamada documentador que genera páginas html describiendo la estructura de la clase y agregando el contenido de este tipo de comentarios que el

programador haya agregado.

Tipos de datos

Java posee tipos de datos adecuados para almacenar datos simples, es decir, no estructurados como las clases y los arreglos. Los diferentes tipos de datos son:

byte	Entero de un byte (hasta 2 dígitos)
short	Entero corto (hasta 4 dígitos)
int	Entero (hasta 9 dígitos)
long	Entero largo (hasta 15 dígitos)
float	Número con decimales corto
double	Número con decimales largo
char	Caracter unicode (16 bits)
boolean	Valor de verdad (verdadero o falso)

Tabla 1: Elaboración propia

Adicionalmente las cadenas se manipulan con un tipo de datos especial llamado String, que en realidad es una clase pero con mejoras sintácticas que permite utilizarlo también como un tipo de datos primitivo.

Variables

Las variables son las unidades mínimas de almacenamiento de datos. Cada una de ellas posee identificador, tipo de datos y valor. En Java el tipo de datos es definido en el momento de la declaración y no puede cambiar. El valor es asignado mediante la operación de asignación y puede cambiar con nuevas asignaciones.

Declaración	
<code>tipo nombre;</code>	<pre>int edad; float precio; int x, y;</pre>
Asignación	
<code>nombre = valor;</code>	<pre>edad = 26; precio = 3.2f; x = y = -1; edad = 27;</pre> <div> <div>edad</div><div>26</div><div>precio</div><div></div><div>x</div><div></div><div>y</div><div></div> <div>edad</div><div>26</div><div>precio</div><div>3,2</div><div>x</div><div></div><div>y</div><div></div> <div>edad</div><div>26</div><div>precio</div><div>3,2</div><div>x</div><div>-1</div><div>y</div><div>-1</div> <div>edad</div><div>27</div><div>precio</div><div>3,2</div><div>x</div><div>-1</div><div>y</div><div>-1</div> </div>
Declaración con inicialización	
<code>tipo nombre = valor;</code>	<code>String nombreMateria = "Laboratorio";</code>
Expresiones	
<p>El valor a la derecha de una asignación puede ser:</p> <ul style="list-style-type: none"> • Constante • Variable • Operación 	<pre>a = 3; b = a; c = a * b + 1;</pre> <div> <div>a</div><div>3</div><div>b</div><div></div><div>c</div><div></div> <div>a</div><div>3</div><div>b</div><div>3</div><div>c</div><div></div> <div>a</div><div>3</div><div>b</div><div>3</div><div>c</div><div>10</div> </div>

Tabla 2: Elaboración propia.

Operadores

Aritméticos	Ambos operandos deben ser del mismo tipo o convertibles entre sí, el resultado es de dicho tipo.	
+ - * / % ++ --	Operaciones aritméticas Resto de una división Incremento (suma 1) Decremento (resta 1)	
Asignación	Ambos operandos deben ser del mismo tipo o convertibles entre sí, el resultado es del tipo del operando de la izquierda.	
= +=	Asignación simple Incremento (suma a la variable de la izquierda el operando de la derecha)	a = 5; a 5 a += 8; a 13
Comparación	Ambos operandos deben ser del mismo tipo, el resultado siempre es boolean.	
==	Igualdad	
!=	Diferencia	
> < >= <=	Mayor, menor, etc.	
Lógicos	Ambos operandos deben ser boolean, el resultado siempre es boolean	
&& !	And Or Negación (operador unario)	
Operador ternario	El primer operando es boolean, los otros dos de cualquier tipo pero ambos iguales.	
(comparación)? Valor_si_verdadero: valor_si_falso	Si la comparación es verdadera, devuelve el segundo operando, si es falsa el tercero.	(edad >= 18)? "Mayor": "Menor"

Tabla 3: Elaboración propia

Ejemplo:

Declarar dos variables enteras y cargar sus valores por teclado. Informar su suma, diferencia, producto y cociente.

```
import java.util.Scanner;

public class Operaciones {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
```

```
        System.out.println("Ingrese un número");
        int a = sc.nextInt();
        System.out.println("Ingrese otro");
        int b = sc.nextInt();

        int suma, resta, producto;
        float cociente;

        suma = a + b;
        resta = a - b;
        producto = a * b;
        cociente = (float)a / b;

        System.out.println("La suma es: " + suma + "!!");
        System.out.println(resta);
        System.out.println(producto);
        System.out.println(cociente);

    }

}
```

Estructuras de control

Las estructuras de control de la programación estructurada son idénticas a las del lenguaje C#.

Estructura condicional simple y doble

Permite establecer una bifurcación en la secuencia del algoritmo. La ejecución de uno o dos bloques de código va a estar condicionada a la evaluación de una expresión booleana. La condición debe plantearse como una expresión que devuelva boolean, usando operadores de comparación o lógicos.

```
if (condicion) {
    bloque verdadero
}
else {
    bloque falso
}
```

El bloque *e/se* es opcional, pero el bloque del lado verdadero es obligatorio. Cuando una condición requiera ejecutar algún bloque únicamente para el caso falso, debe invertirse la lógica de la misma para que el bloque pase al caso verdadero. Si

la condición es muy compleja se puede invertir su valor de verdad con el operador de negación.

Ejemplo: Generar dos números al azar indicando cuál de ellos es el mayor.

```
public class PruebaMay{

    public static void main(String args[]){
        int numero1 = (int)(10000*Math.random());
        int numero2 = (int)(10000*Math.random());

        System.out.println("Primer numero " + numero1);
        System.out.println("Segundo numero " + numero2);

        if(numero2 > numero1)
            System.out.println("El segundo numero es el mayor");
        else
            System.out.println("El primer numero es el mayor");
        }
    };
}
```

Un par de ejecuciones:

```
Primer numero 5868
Segundo numero 654
El primer numero es el mayor
```

```
Primer numero 3965
Segundo numero 7360
El segundo numero es el mayor
```

Estructura condicional múltiple

Esta estructura evalúa un valor de tipo int, char o String comparando por igualdad con un conjunto de valores constantes. Cada comparación se denomina caso y cada una de ellas es evaluada en el orden en que se encuentren escritas en el código. Cuando una de esas comparaciones resulte verdadera, se ejecuta el código que se presente a continuación hasta el final del bloque o hasta la aparición de la instrucción break.

```
switch (expresión) {  
    case 1: ...  
        ...  
        break;  
    case 2: ...  
        ...  
        break;  
    case 3: ...  
        ...  
        break;  
    default: ...  
        ...  
}
```

Si no se cumple ninguno de los casos se ejecuta el caso opcional por defecto, identificado con la palabra default:

Estructuras repetitivas

Hay tres tipos de estructuras repetitivas o ciclos, que se diferencian entre sí por cantidad de vueltas o iteraciones que deben realizar.

Ciclo 0-n: En este ciclo el bloque iterativo se va a ejecutar una cantidad veces que es desconocida en tiempo de compilación. Antes de iniciar cada iteración se evalúa una condición, la cual indica si se debe ejecutar la vuelta. Si la condición es verdadera, se ejecuta una vuelta más, si es falsa se interrumpe el ciclo. Dado que la primera vez que se evalúa la condición la misma puede ser falsa, en ese caso el bloque iterativo no se ejecuta, es decir, se ejecuta cero veces, y esta situación es la que le da el nombre a este tipo de ciclo. En java se programa con la instrucción while.

```
while (condicion de corte) {  
    bloque iterativo  
}
```

Ciclo 1-n: Este ciclo es similar al 0-n pero la condición de corte es evaluada luego de ejecutar el bloque iterativo. A causa de esto, aunque en la primera vuelta la condición devuelva falso, el bloque iterativo ya se habrá ejecutado. A causa de esto, este ciclo garantiza que al menos una vuelta como minimo es ejecutada. En el lenguaje java se lo programa con la instrucción do while.

```
do {  
    bloque iterativo  
} while (condicion de corte);
```

Ciclo exacto: el ciclo exacto se denomina de esta manera ya que la cantidad de vueltas que realiza es conocida con exactitud con antelación a la ejecución del mismo. Para lograr esto se requiere una variable que funcione como un contador de vueltas y una condición de corte que compare dicho contador con la cantidad de vueltas requeridas. Este ciclo se obtiene con la instrucción for.

```
for (inicialización; condicion de corte; incremento) {  
    bloque iterativo  
}
```

Entrada/Salida estándar

- **Salida por consola**

En las aplicaciones de consola de texto, la presentación de información al usuario y el ingreso de datos por éste se realizan a través de los dispositivos de entrada y salida estándar, es decir, el teclado y la ventana de texto. Para poder realizar estas operaciones Java provee dos objetos que manipulan todos los tipos de datos primitivos sin necesidad de conversiones o interpretación de la entrada (parsing).

Para presentar información al usuario se dispone del objeto `System.out` el cual posee varios métodos de impresión. Los dos que más se utilizan son `print()` y `println()`. Ambos funcionan de la misma manera, reciben un parámetro con el dato que se desea imprimir. El dato puede ser cualquier expresión, es decir: un valor constante, una variable o una operación. En este último caso se calcula el resultado de la operación y se lo muestra, sin necesidad de almacenarlo en una variable temporal.

```
System.out.print("Hola ");  
System.out.println(nombre);  
System.out.println("Hola " + nombre);
```

La diferencia entre los métodos `print` y `println` reside en que el último agrega automáticamente al final de la impresión un carácter de nueva línea ("`\n`") forzando de esta manera a que el cursor baje a la línea siguiente para que la próxima impresión se visualice en otra línea. Si se utiliza `print` el cursor no se mueve y la próxima impresión se visualizará a continuación en la misma línea.

A causa de esto último se debe que `println()` puede ser invocado sin parámetros para dibujar una línea en blanco. En cambio, carece de sentido que `print()` se invoque sin

parámetros y por lo tanto el compilador rechaza una llamada de esa naturaleza.

- **Ingreso de datos**

Así como existe el objeto `System.out`, java provee un objeto similar llamado `System.in` que permite el ingreso de datos por parte del usuario. Sin embargo `System.in` sólo ofrece métodos para ingresar datos de a un carácter por vez, y cuando se requiere ingresar números o cadenas debe invocarse múltiples veces al método de lectura, y procesando los caracteres ingresados concatenandolos y luego convirtiendolos para obtener el dato definitivo. Esta tarea es sumamente compleja para un objetivo que debería ser muy simple, como el de ingresar un número entero o una cadena de caracteres.

Por eso Java agregó una clase que se encarga de tal tarea liberando al programador de dicha responsabilidad. La clase en cuestión se llama `Scanner` y se encuentra en el paquete `java.util`. Para poder utilizarla se necesita agregar ANTES del bloque `class` un instrucción `import` que permita acceder a la misma.

```
import java.util.Scanner;
```

```
public class Programa {  
    ...  
}
```

Y dado que `Scanner` es una clase debe ser instanciada como cualquier otro objeto:

```
Scanner sc = new Scanner(System.in);
```

El nombre de la variable `sc` puede ser modificado por cualquier otro, tal como consola, teclado o entrada (o cualquiera que el programador desee). El parámetro del constructor de `Scanner` se indica como `System.in` para poder leer datos desde el teclado, pero `Scanner` puede leer datos directamente desde un archivo de texto y lo único que se requiere para ello es reemplazar `System.in` por el archivo desde el que se desea leer los datos. (En realidad es un poco más difícil pero no resulta de interés analizar los detalles para esta materia).

Para leer un dato desde el teclado se invoca a uno de varios métodos llamados `nextXXX()`, reemplazando `XXX` por un tipo de datos. Existe un método por cada tipo de datos, cada uno de ellos convirtiendo y retornando un dato específico. De esta manera, para cargar un número entero simplemente se invoca a `.nextInt()` y el retorno del mismo se puede guardar en una variable de tipo `int`, sin necesidad de hacer una operación de parseo.

Clase `Scanner`

Métodos:

```
nextInt();
nextFloat();
nextDouble();
next(); // Lee una cadena hasta que se encuentre un espacio.
nextLine(); // Lee una cadena hasta que se encuentre un retorno de carro.
```

Comparación entre Java y C#

Programa de consola en C#	Programa de consola en Java
<pre>using System; namespace programa { public class SumaNumeros { public void Main() { int a, b; string nombre; Console.WriteLine("Ingrese su nombre"); nombre = Console.ReadLine(); Console.WriteLine("Ingrese un número"); a = Convert.ToInt32(Console.ReadLine()); Console.WriteLine("Ingrese un número"); b = Convert.ToInt32(Console.ReadLine()); int suma = a + b; Console.WriteLine("Hola" + nombre); Console.WriteLine("La suma entre los dos números es: " + suma); } } }</pre>	<pre>package programa; import java.util.Scanner; public class SumaNumeros { public static void main(String[]args) { int a, b; String nombre; Scanner sc = new Scanner(System.in); System.out.println("Ingrese su nombre"); nombre = sc.next(); System.out.println("Ingrese un número"); a = sc.nextInt(); System.out.println("Ingrese un número"); b = sc.nextInt(); int suma = a + b; System.out.println("Hola" + nombre); System.out.println("La suma entre los dos números es: " + suma); } }</pre>

Tabla 4: Elaboración propia

Ejercicios resueltos

1: Hacer un programa que ingrese el precio de un artículo a la venta y calcule el precio con IVA.

```
import java.util.Scanner;

public class PrecioIVA {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.println("Ingrese el precio");
        float precio = sc.nextFloat();
        float iva, total;

        iva = precio * 0.21f;
        total = precio + iva;

        System.out.println("El precio incluyendo IVA es de: " +
total);
    }

}
```

2: Ingresar la cantidad de horas trabajadas por un empleado y el sueldo que cobra por hora. Indique el total a cobrar teniendo en cuenta que si trabajó más de 180 horas las excedentes las cobra con un 50% de aumento.

```
import java.util.Scanner;

public class HorasExtras {

    public static void main(String[] args) {
```



```
Scanner sc = new Scanner(System.in);

int horas, sueldoHora;
float sueldoTotal;

System.out.print("Ingrese la cantidad de horas
trabajadas ");
horas = sc.nextInt();

System.out.print("Ingrese el sueldo por hora ");
sueldoHora = sc.nextInt();

if (horas <= 180) {
    sueldoTotal = horas * sueldoHora;
} else {
    int horasExtras = horas - 180;
    sueldoTotal = horas * sueldoHora + horasExtras *
sueldoHora * 0.5f;
}

System.out.println("El sueldo a cobrar es $ " +
sueldoTotal);
}
}
```

3: Ingresar un número n y a continuación n números positivos. Informar la cantidad de mayores a 5 que se hayan ingresado.

```
import java.util.Scanner;

public class Mayores5 {
```

```
public static void main(String[] args) {  
    Scanner sc = new Scanner(System.in);  
  
    int n, x, cantidad = 0;  
  
    System.out.println("Ingrese la cantidad de números");  
    n = sc.nextInt();  
  
    System.out.println("Ingrese " + n + " números");  
    for (int i = 0; i < n; i++) {  
        x = sc.nextInt();  
        if (x > 5) {  
            cantidad++;  
        }  
    }  
  
    System.out.println("Se ingresaron " + cantidad + "  
números mayores a 5");  
}
```

2. PROGRAMACIÓN ORIENTADA A OBJETOS

Introducción

Java es un lenguaje orientado a objetos con características muy similares a las de C#. Como en cualquier lenguaje orientado a objetos, “...las clases son moldes o modelos para construir objetos. En ellas se especifican los atributos y métodos generales a todos los objetos contruidos a partir de esa clase....” (Corso, 2012)

Bloque de clase

Una clase se define con la palabra reservada class. La sintaxis de una clase es:

```
[public] [final | abstract] class NombreClase [extends ClaseBase]
```

```
{  
[lista de atributos]  
[lista de métodos]  
}
```

Ejemplo:

```
public class Alumno  
{  
//lista de atributos  
//lista de métodos  
}
```

Miembros

Tambi3n seg3n indica Corso, una clase contiene elementos, llamados miembros, que pueden ser datos, llamados atributos, y funciones que manipulan esos datos llamados métodos.

Atributos

Los atributos se declaran como variables de tipos primitivos. (int, float, char, etc.) o de clase (objetos, String, vectores, listas, etc.)

Todos los atributos se los declara con la siguiente sintaxis:

```
[Modificador de acceso] Tipo NombreVariable [ = ValorInicial ];
```

Ejemplos:

```
private int legajo;  
private String nombre = "Juan";
```

Métodos

Los métodos son porciones de código con nombre asociados a la clase, de forma tal que cada instancia pueda reaccionar a un mensaje que le se envíe con el nombre del método. Los métodos pueden recibir parámetros y retornar valores. En el caso de que un método acceda a los atributos de la clase manipula los datos concretos de la instancia que recibió la llamada.

```
[modificador de acceso] [ abstract ] TipoRetorno NombreMétodo (tipo1 nombre1 [, tipo2 nombre2]...) )  
{  
  declaraciones de variables locales;  
  sentencias;  
  [return expresion ];  
}
```

Ejemplo:

```
public void setLegajo(int l)  
{  
  legajo = l;  
}
```

Parámetros

Cada método posee un conjunto de líneas de código que van a ser ejecutadas cuando el mismo es invocado. Normalmente los métodos realizan alguna tarea con los atributos del objeto que recibe la llamada, pero ocasionalmente un método puede necesitar otros datos que no están almacenados dentro de los atributos del objeto. Para poder obtener esos datos, el método puede solicitar argumentos o parámetros, que van a funcionar como variables locales, pero en las que el dato que almacenen sea especificado durante la invocación del método.

La lista de parámetros es la lista de nombres de variables separados por comas, con sus tipos asociados, que reciben los valores de los argumentos cuando se llama al método.

Un método puede recibir como parámetros valores prácticamente de cualquier tipo. Estos parámetros serán usados por el método para realizar operaciones que lleven a la acción que se espera generar.

Los parámetros que va a recibir un determinado método deben ser indicados en la cabecera del mismo, luego del identificador y entre paréntesis. La declaración propiamente dicha es similar a la que se realiza cuando se declara una variable, es decir, se indica el tipos y el nombre de cada parámetro, usando una coma para

separar cada uno de ellos.

Los parámetros tienen el ámbito y duración del método. Esto significa que los parámetros se podrán ver y utilizar sólo dentro del método y al finalizar la ejecución del mismo, son borrados automáticamente.

Propiedades

El lenguaje Java no posee una construcción sintáctica especial para especificar las propiedades. A causa de esto se las simula mediante la creación de un par de métodos por cada atributo. Estos métodos son dos ya que se necesita uno para la operación de consulta de un atributo y otro para la asignación de un nuevo valor en el atributo.

Estos métodos pueden tener cualquier nombre, pero la convención que se ha establecido indica que todos los métodos de asignación deben llamarse setXXX reemplazando las XXX por el nombre del atributo, iniciado con mayúscula. Por su parte, los métodos de consulta deben llamarse getXXX, con la única excepción de los atributos de tipo boolean. Para este tipo de datos resulta más claro (en idioma inglés) que el método get se llame isXXX.

Si bien los nombres de los métodos get y set no deben cumplir obligatoriamente con esta regla, los diferentes entornos de desarrollo normalmente ofrecen algún mecanismo para generarlos en forma automática con la nomenclatura convenida. Por otro lado, para programar software con interfaz web en Java, los objetos deben poseer obligatoriamente estos métodos y con los prefijos correctos.

Modificadores de acceso

Todos los miembros de las clases poseen una característica llamada modificador de acceso que indica cuáles clases pueden accederlos. Lógicamente todos los miembros de una clase pueden ser accedidos desde métodos de la misma, el modificador de acceso especifica por cada miembro si puede ser accedido desde otra clase en ese caso de cuáles.

Los modificadores de acceso son cuatro:

- public
- private
- protected
- paquete

Los miembros definidos como public pueden ser accedidos por cualquier otra clase, incluyendo la del programa, es decir, aquella clase que posee el método main.

Los miembros privados son aquellos afectados por el modificador private y a causa de esto no pueden ser accedidos por ninguna clase diferente a la propia.

Los miembros `protected` o protegidos tienen utilidad sólo en clases relacionadas mediante herencia. Para una clase base, los miembros `protected` pueden ser accedidos por sus derivadas como si fueran públicos pero se comportan como privados frente a otras clases que no pertenezcan al árbol de herencia. Es decir que los miembros protegidos pueden ser accedidos únicamente por la clase que los define y sus derivadas. Es importante remarcar que pueden ser accedidos por las derivadas en cualquier nivel, es decir las derivadas directas de la clase o las derivadas de las derivadas, sin importar cuántas relaciones de herencia existan entre la clase base y las que quieran acceder al miembro.

Finalmente hay un modificador que no se lo indica y se aplica por defecto. Cualquier miembro que no posea un modificador de acceso posee una visibilidad de paquete, es decir que puede ser accedido por las clases que pertenezcan al mismo paquete que la propia pero no desde otras.










Modificador	Misma clase	Mismo paquete	Derivadas	Otras
<code>public</code>				
(por defecto)				
<code>protected</code>				
<code>private</code>				

Tabla 5: Elaboración propia

Constructores

Un constructor es un método que se ejecuta automáticamente durante la instanciación, es decir, al crear los objetos con new. Estos métodos no pueden ser llamados en forma explícita y por tal motivo no tienen nombre ni retorno.

Los métodos constructores sólo utilizan como nombre el nombre de la clase.

Sin embargo, los constructores pueden tener parámetros. Cuando eso ocurre, se envían los valores de los parámetros en la llamada a new.

Una clase puede tener varios constructores, pero como todos ellos poseen el nombre de la clase, se los diferencia por la cantidad y tipo de los parámetros. Luego, según cuántos parámetros se le pasen a new, se ejecuta uno u otro de los constructores, el que coincida con los parámetros pasados.

Constructor sin parámetros	
<code>public Clase() { ... }</code>	<code>public Persona() {}</code>
Constructor con parámetros	
<code>public Clase([parámetros]) { ... }</code>	<code>public Persona (String nombre, String apellido) { ... }</code>
Ejecución de un constructor	
<code>Clase objeto = new Clase();</code> // Automáticamente se ejecuta el constructor cuya lista de parámetros formales coincida con la cantidad y tipo de los parámetros enviados.	<code>Persona p = new Persona();</code> En este punto se ejecuta el constructor que no posea parámetros.

Tabla 6: Elaboración propia

Objetos y referencias


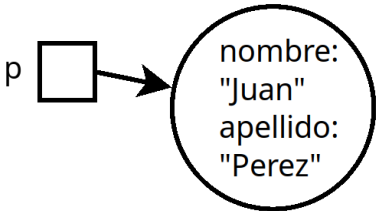
Los objetos de una clase son instancias de la misma, es decir que son elementos que existen en la memoria del programa y cuya estructura cumple con la especificada en la clase. Se crean en tiempo de ejecución mediante el uso del operador new.

Sin embargo, a diferencia de las variables los objetos no tienen identificador; para poder enviarle mensajes por lo tanto para ejecutar sus métodos es necesario identificar a cada uno de los objetos que se hayan creado.

Para lograr esto, surge un nuevo tipo de datos para definir variables. Una vez programada una clase, su nombre puede ser usado como un tipo de datos y con él definir variables. Estas variables se denominan referencias y su valor consiste en la dirección de memoria de un objeto. Cuando una de estas variables posee como valor la dirección de un objeto en particular se dice que la variable “apunta” o “referencia” a dicho objeto.

Las variables de tipo referencia sólo pueden apuntar a objetos del mismo tipo (misma clase) del que se uso para declararlas. Sus valores nunca son asignados en forma explícita a constantes sino que únicamente pueden asignarse a objetos u otras referencias. El uso más habitual es el de asignarle el resultado del operador new. De esta manera, new crea un nuevo objeto y la asignación consigue que la variable apunte a este nuevo objeto.

Si se asignan dos variables de tipo referencia entre sí, el significado de la asignación debe interpretarse como “la variable de la izquierda va a apuntar al mismo objeto que la variable de la derecha”.

Declaración:	
<code>Clase nombreReferencia;</code>	<code>p</code>  <code>Persona p;</code>
Asignación a un objeto nuevo	
<code>Clase nombreReferencia = new</code> <code>Clase([parámetros]);</code>	<code>Persona p =</code> <code>new Persona("Juan","Perez");</code> 

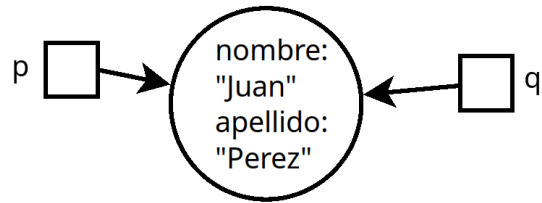
<p>Asignación a otro objeto</p> <p>Clase <code>otraReferencia</code> <code>nombreReferencia;</code></p>	<p>= <code>Persona q = p;</code></p> 
---	---

Tabla 7: Elaboración propia.

Cuadro comparación clase Persona entre C# y Java

Clase Persona en C#	Clase Persona en Java
<pre>namespace personas { public class Persona { private string nombre; private string apellido; private int nacimiento; public string Nombre { get { return nombre; } set { nombre = value; } } public string Apellido { get { return apellido; } set { apellido= value; } } public int Nacimiento { get { return nacimiento; } set { nacimiento= value; } } public Persona() {</pre>	<pre>package personas; public class Persona { private String nombre; private String apellido; private int nacimiento; public String getNombre() { return nombre; } public void setNombre(String nombre) { this.nombre = nombre; } public String getApellido() { return apellido; } public void setApellido(String apellido) { this.apellido = apellido; } public int getNacimiento() { return nacimiento; } public void setNacimiento(int nacimiento) { this.nacimiento = nacimiento; } public Persona() {</pre>

<pre> } public Persona(String nombre,String apellido, int nacimiento) { this.nombre = nombre; this.apellido = apellido; this.nacimiento = nacimiento; } public override string toString() { return "Persona{" + "nombre=" + nombre + ", apellido=" + apellido + ", nacimiento=" + nacimiento + '}'; } public string nombreCompleto() { return nombre + " " + apellido; } public int edad() { return 2016-nacimiento; } } } </pre>	<pre> public Persona(String nombre, String apellido, int nacimiento) { this.nombre = nombre; this.apellido = apellido; this.nacimiento = nacimiento; } @Override public String toString() { return "Persona{" + "nombre=" + nombre + ", apellido=" + apellido + ", nacimiento=" + nacimiento + '}'; } public String nombreCompleto() { return nombre + " " + apellido; } public int edad() { return 2016-nacimiento; } } } </pre>
--	--

Tabla 8: Elaboración propia

Ejercicios de Programación Orientada a Objetos

1: Programar una clase Punto que represente un punto en el plano. Agregarle comportamiento para que cada objeto punto sepa informar su distancia al origen del sistema de coordenadas y el cuadrante donde se encuentra. Finalmente agregar un método que calcule la distancia hacia un punto representado mediante otra instancia de la misma clase.

```
package punto;

public class Punto {
    private int x, y;

    public int getX() {
        return x;
    }

    public void setX(int x) {
        this.x = x;
    }

    public int getY() {
        return y;
    }

    public void setY(int y) {
        this.y = y;
    }

    public Punto(int x, int y) {
        this.x = x;
        this.y = y;
    }

    @Override
    public String toString() {
        return "Punto{" + "x=" + x + ", y=" + y + '}';
    }

    public double distanciaOrigen() {
        return Math.sqrt(x*x+y*y);
    }

    public int cuadrante() {
        int cuad = 0;

        if (x > 0) {
            if (y > 0) cuad = 1;
            else      cuad = 4;
        }
    }
}
```

```
        else {
            if (y > 0) cuad = 2;
            else      cuad = 3;
        }

        return cuad;
    }

    public double distancia(Punto otro) {
        int dx = this.x - otro.x;
        int dy = this.y - otro.y;

        return Math.sqrt(dx*dx+dy*dy);
    }
}
```

2: Programar una clase Alumno que represente los datos de un alumno de una materia. De cada alumno se conoce nombre, legajo, tres notas, la cantidad de días que asistió a clase y la cantidad de días que faltó. La clase debe poseer los siguiente métodos:

promedio: que calcule el promedio de las tres notas.

porcentajeAsistencia: que calcule el porcentaje de asistencia, es decir, el porcentaje de clases que asistió sobre el total de clases dictadas.

estaLibre: que devuelva verdadero o falso indicando si está libre. Un alumno se queda libre si no alcanza el 70% de asistencia.

```
public class Alumno {

    private String nombre;
    private int legajo;
    private int nota1, nota2, nota3;
    private int asistencias, faltas;

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public int getLegajo() {
        return legajo;
    }

    public void setLegajo(int legajo) {
```

```
        this.legajo = legajo;
    }

    public int getNota1() {
        return nota1;
    }

    public void setNota1(int nota1) {
        this.nota1 = nota1;
    }

    public int getNota2() {
        return nota2;
    }

    public void setNota2(int nota2) {
        this.nota2 = nota2;
    }

    public int getNota3() {
        return nota3;
    }

    public void setNota3(int nota3) {
        this.nota3 = nota3;
    }

    public int getAsistencias() {
        return asistencias;
    }

    public void setAsistencias(int asistencias) {
        this.asistencias = asistencias;
    }

    public int getFaltas() {
        return faltas;
    }

    public void setFaltas(int faltas) {
        this.faltas = faltas;
    }

    @Override
    public String toString() {
        return "Alumno{" + "nombre=" + nombre + ", legajo=" + legajo + ", nota1=" +
        nota1 + ", nota2=" + nota2 + ", nota3=" + nota3 + ", asistencias=" + asistencias + ",
        faltas=" + faltas + '}';
    }

    public float promedio() {
        return (nota1 + nota2 + nota3) / 3f;
    }
}
```

```
    }  
  
    public float porcentajeAsistencia() {  
        return (faltas / (asistencias + faltas)) * 100;  
    }  
  
    public boolean estaLibre() {  
        return porcentajeAsistencia() < 70;  
    }  
}
```

BIBLIOGRAFÍA

- Ceballos Sierra, F. (2010). *Java 2. Curso de Programación*. 4ta Edición. Madrid, España. RA-MA Editorial.
- Corso, C; Colaccioppo, N. (2012). “*Apunte teórico-práctico de Laboratorio de Computación III*”. Córdoba, Argentina. Edición digital UTN-FRC.



Atribución-NoComercial-SinDerivadas

Se permite descargar esta obra y compartirla, siempre y cuando no sea modificado y/o alterarse su contenido, ni se comercializarse. Referenciarlo de la siguiente manera:

Universidad Tecnológica Nacional Regional Córdoba (2020). Material para la Tecnicatura en Programación Semipresencial de Córdoba. Argentina.