GUIA - LENGUAJE DE PROGRAMACIÓN JAVA

Versión 1.1

ORIENTACIÓN A OBJETOS I
UNRN

Contenido

Aspectos generales de la gramática

COMENTARIOS E IDENTIFICADORES

Los comentarios del código tienen tres formatos:

- Comentarios de línea: "//", es comentario hasta final de línea
- Comentarios de bloque: "/*" ... "*/"
- Comentarios de documentación: "/**" ..."*/". Se utiliza para documentar los aspectos públicos de las clases; mediante la herramienta javadoc.exe se genera documentación en formato HTML con el contenido.

Los blancos, tabuladores y saltos de línea, no afectan al código.

Las mayúsculas y minúsculas son diferentes.

Convenciones

Existe un estilo de nombrar los diferentes elementos ampliamente aceptado por la comunidad Java:

- **Clases**: las palabras empiezan por mayúsculas y el resto en minúsculas (ej. HolaMundo)
- **Métodos**: las palabras empiezan por mayúsculas y el resto en minúsculas, excepto la primera palabra que empieza por minúsculas (ej. holaMundo)
- **Constantes**: todo en mayúsculas separando las palabras con "_"

DECLARACIÓN DE VARIABLES Y CONSTANTES

Existen dos tipos de variables: las primitivas y las de clase. Las variables primitivas almacenan el dato en sí, por ello, dependiendo del tipo de dato ocupan un tamaño diferente de memoria. Las variables de clase almacenan la referencia del objeto y tienen un tamaño fijo de 32 bits.

La declaración de variables tiene el siguiente formato:

```
tipo identificador;
tipo identificador = expresión;
tipo identificador1, identificador2 = expresión;
```

Algunos ejemplos serían:

```
int prueba;
byte pruebaDos, prueba3 = 10, prueba4;
double prueba5 = 10.0;
```

Las constantes se declaran igual que las variables, pero añadiendo la palabra final.

```
final tipo IDENTIFICADOR = expresión;
```

Algunos ejemplos serían:

```
final int CONSTANTE_UNO = 10;
final double CONSTANTE_DOS = 10.0;
```

A continuación presentamos una tabla con los tipos primitivos, el tamaño que ocupa en memoria y el rango de valores que pueden representar.

- Enteros
  - byte. 8 bits. -128 a +127
  - short. 16 bits. -32.768 a +32.767
  - int. 32 bits. -2.147.483.648 a -2.147.483.648
  - long. 64 bits. -9.223.372.036.854.775.808 a 9.223.372.036.854.775.807
- Decimales
  - float. 32 bits. ≈ ±1.7*10 ^{38}
  - double. 64 bits. ≈ ±3.4*10 ^{308}
- Caracteres
  - char. 16. '\u0000 ' a '\uffff'
- Lógico
  - boolean. true y false
- Objetos
  - Object

Vamos a realizar un breve adelanto de la clase String. Su utilidad es almacenar una cadena de caracteres.

Un ejemplo de uso sería:

```
String nombre="Jorge", ciudad="Madrid";
```

Una cadena de caracteres no puede ser dividida mediante un salto de línea. Para cadenas largas, se dividen en varias y se concatenan con el operador "+", pudiéndose definir en varias líneas.

PALABRAS RESERVADAS

A continuación, se enumeran las palabras reservadas del lenguaje:

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, instanceof, int, interface, long, native, new, package, private, protected, public, return, short, static, strictfp, super, synchronized, switch, this, throw, throws, transient, try, void, volatile, while

OPERADORES

Los operadores de Java son:

- Asignación
  - =. Ej.: x=y;
- Aritméticos
  - ++, --. Suma o resta 1, ej.: x++; ++x;
  - - . Cambio de signo, ej.: –x;
  - *, /. Multiplicación y división, ej.: x*y;
  - %. Resto de la división, ej.: x%y;
  - +, -. Suma y resta, ej.: x+2;
  - +=, -=, *=, /=, %=. Ej.: x+=2; x=x+2;
- Comparación
  - ==, ¡=, <, >, <=, >=. Igual, distinto, menor que...
- Lógicos
  - !. Not lógico
  - &, &&, |, || y ^. And, and perezoso, or, or perezoso y xor

El operador "+" entre String realiza la función de concatenación.

EXPRESIONES

Las expresiones son una combinación de operadores y operandos. La precedencia y asociatividad determina de forma clara la evaluación. El orden aplicado es:

- Paréntesis: ( )
- Unarios: + - !
- Multiplicativos: * / %
- Aditivos: + -
- Relacionales: < > <= >=
- Igualdad: == !=
- Y lógico: & &&
- O lógico: | ||

ARRAYS

Los arrays son tablas de valores del mismo tipo. En Java los arrays son dinámicos y deben crearse o reservar memoria en tiempo de ejecución para su utilización. Por lo tanto, para poder utilizar arrays primero se debe declarar, y posteriormente, reservar la memoria para su uso.

Para definir un array tiene el siguiente formato:

```
tipo[] identificador;
```

Se define pero no se crea, realmente el contenido de una variable de array es una referencia a una zona de

memoria; el valor inicial es la palabra reservada null. Un ejemplo sería:

```
int[] diasMes;
```

Para reservar memoria para el array se debe indicar el tamaño; una vez establecido el tamaño, los posibles valores de índice van desde 0 a tamaño-1:

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

Si se vuelve a reservar memoria, se libera la memoria anteriormente utilizada y por lo tanto se pierde el contenido que tuviese.

Las dos acciones anteriores se pueden realizar en una sola línea:

```
int[] tabla = new int[10];
```

También se puede declarar un array, reservar memoria y darle contenido en una sola línea:

```
int[] tabla = { 0, 1, 2 };
```

Para saber el tamaño de un array no nulo (ya se ha reservado espacio en memoria) se puede utilizar el atributo length: identificador.length. Si se intenta consultar el tamaño de un array no creado da un error (NullPointerException).

Se pueden definir arrays de varias dimensiones. Por ejemplo:

```
int[][] matriz = new int[10][20];
```

Para acceder a una posición del array se realiza a través de su índice, tienen un rango fijo de 0 a tamaño-1. Algunos ejemplos son:

```
int i = tabla[2];
i = matriz[2][3];
```

Si una variable de tipo array lleva el calificador final, quiere decir que el array no se permite que se vuelva a definir con el calificador new, pero si se permite cambiar el contenido del array. Mostramos un ejemplo para comprender este concepto:

```
final int[] array = new int[5];
array[3] = 3; // Correcto
// Error con array = new int[10];
```

Sentencias condicionales

SENTENCIA IF

La sentencia if es una de las más básicas y nos permite controlar el flujo de control dependiendo de una condición. Los dos formatos posibles son:

```
if (condicion) sentencia1;
if (condición) sentencia1; else sentencia2;
```

Si en lugar de una sentencia tenemos varias, se debe utilizar la sentencia de bloque. Algunos ejemplos son:

```
if (a == 3) {
 b = 8;
}
if (a > 2) {
 a -= 1;
 b += 1;
}
if (a > 2) {
 a -= 1;
 b += 1;
} else a += 1;
```

Todas las sentencias se pueden anidar. Recordar que se debe aplicar un tabulador para su mejor comprensión. Un ejemplo sería:

```
if (a > 2) {
 if (b == 8) {
  b++;
 }
} else {
 b--;
}
```

SENTENCIA ELSE-IF

Aunque la sentencia else-if no existe como tal, queremos presentarla de forma independiente ya que debe tener un estilo peculiar de tabulación. Se considera una sentencia else-if si en la condición siempre se evalúa una misma variable o expresión; aunque existan anidamientos de sentencias, los tabuladores aplicados y el uso de llaves son diferentes a las normas generales.

Un ejemplo sería:

```
if (a==0) {
 sentencia1;
} else if (a == 1 | a == 2) {
 sentencia2;
} else if (a < 10) {
 sentencia3;
 sentencia4;
} else {
 // No se debiera llegar aquí
}
```

SENTENCIA SWITCH

Esta sentencia evalúa una expresión, y dependiendo de su valor, ejecuta un conjunto de sentencias. Sólo se puede aplicar a valores primitivos, enumerados y a clases especiales. Su formato es:

```
switch (Expresion) {
 case valor:
  sentencia1;
  sentencia2;
 break;
 case valor2:
 case valor3:
  sentencia3;
 break;
 case valor4:
  sentencia4; break;
 default: sentencia5;
}
```

Sentencias de bucles

SENTENCIA WHILE Y DO WHILE

Estas sentencias se utilizan para realizar bucles o repeticiones hasta que se cumpla una determinada condición, pero a priori es indefinido el número de veces que se puede repetir.

La sentencia while tiene una repetición de 0 a n, y la sentencia do while de 1 a n. Su formato es:

```
while (condición) {
 sentencia;
 sentencia;
 sentencia;
}
```

```
do {
 senten2;
} while (condición)
```

SENTENCIA FOR

Esta sentencia realiza un bucle, donde existe un índice de iteración y la condición de fin depende del índice. Tiene tres partes separadas por ";": la primera se inicializan los índices (si hay más de uno se separan por ","), la segunda es la condición de terminación (el bucle se repite mientras se cumpla la condición) y la tercera es el incremento de los índices.

Si los índices se declaran en el propio for, no se podrán utilizar fuera de éste. Normalmente, la condición debe depender del valor del índice.

Su formato es:

```
for (inicialización; terminación(mientras); operación) {
 sentencia1;
 ...
}
```

Mostramos algunos ejemplos:

```
for (int i = 1; i <= 10; i++) {
 a *= 2;
}
int i, j;
for (i = 1, j = 1; (i <= 5) && (j <= 5); i++, j += 2) {
 // ...
}
for (;;) {
 // ...
} // bucle infinito
```

SENTENCIA FOR EACH

Esta sentencia es útil para procesar una colección de datos, donde sólo nos interesa el conjunto de datos y no el orden con que se procesan.

Se puede aplicar a las clases Collection y a los arrays. Su formato es:

```
for (int item: array){
 // en la variable item tenemos el dato
}
```

Un posible ejemplo sería:

```
int[] datos = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
for (int item : datos) {
 System.out.println(item);
}
```

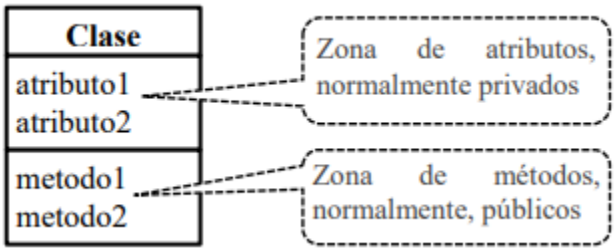
Es equivalente al siguiente bucle:

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

Clases y objetos

Clases: atributos y métodos

Una clase es un conjunto de atributos y métodos. Vemos en la figura siguiente, una manera de mostrar una clase.



Las clases tiene dos vistas: una pública y una privada. La vista pública, también llamado interfaz, establece los aspectos visibles de la clase, y por lo tanto, nos ofrece los métodos que podemos utilizar. La vista privada, también llamada implantación, encapsula todos los detalles de la estructura de datos y la implementación de los métodos.

Atributos

Normalmente, los atributos son privados, y por lo tanto, no son directamente accesibles. La consulta o modificación de los atributos se debe realizar a través de métodos públicos; de esta manera, el objeto controla las operaciones válidas que se pueden realizar.

Un atributo es una información que se almacena y es parte representativa del estado. Los atributos se pueden sustentar mediante tipos primitivos o clases. Mostremos algunos ejemplos.

La clase Punto representa un punto en un espacio de dos dimensiones, sus atributos son x e y. La declaración de la clase quedaría:

```
public class Punto {
 private int x, y;
}
```

Observar, que los atributos se declaran a nivel de instancia, fuera de los métodos, y por lo tanto, son accesibles de cualquier parte de la clase.

La clase NumeroNatural representa un valor natural, su atributo podría ser valor. La clase quedaría:

```
public class NumeroNatural{
 private int valor;
}
```

La clase Usuario representa a un usuario, podría tener los atributos nombre, eMail, movil, ciudad y mayorEdad. El código java sería:

```
public class Usuario{
 private String nombre, eMail, ciudad;
 private long movil;
 private boolean mayorEdad;
}
```

Como puede observarse en todos los ejemplos, los atributos se declaran a nivel de la clase y están calificados con private, indicando que no es accesible desde fuera de la clase.

Se pueden crear tantos objetos o instancia de las clases como queramos. Dos objetos de la clase Punto tienen valores de atributos independientes y evolucionan por separado; eso sí, cuando se crea la instancia, todas parten del mismo estado.

Métodos

La razón de establecer los atributos en vista privada, es poder controlar la modificación del estado del objeto y no permitir que este evolucione hacía estados incoherentes. Pero debemos establecer los mecanismos para poder modificar el objeto, y son los métodos, el sistema adecuado.

Podemos organizar los métodos en tres tipos, teniendo en cuenta aspectos sintácticos y semánticos:

- Constructores. Métodos que inicializan la instancia
- Métodos genéricos. Realizan acciones utilizando los atributos
- Métodos para accesos directo a los atributos (getters & setters). Estos métodos son opcionales y suelen utilizarse más en clases de tipo “entidad”, es decir, que tienen persistencia en bases de datos, o en los “beans”, donde tienen una correspondencia con formularios. Aunque estos métodos son realmente generales, los distinguimos por separado por tener unas normas concretas de estilo.

MÉTODOS GENÉRICOS

Comencemos con el formato básico de un método:

```
[public | private] {void|tipo} identificadorMetodo (tipo param1, tipo param2...) {...}
```

El primer calificador es public o private, se establece el tipo de vista.

La segunda palabra es el tipo devuelto, con el calificador void (vacío) se indica que no devuelve nada, o se establecer el tipo devuelto.

La siguiente palabra representa el nombre del método.

A continuación vienen los parámetros. Hay que resaltar, en general, los datos con que actúan los métodos son sus propios atributos, y por lo tanto no es necesario pasarlos por parámetro. Por parámetros se pasan valores externos al objeto que condicionan la acción a realizar.

Existen dos tipos de parámetros:

- Tipos primitivos (byte, short, int...). Estos se pasan por valor, es decir, se realiza una copia del contenido en la variable que sustenta el parámetro, y por lo tanto, si el método modifica el parámetro no afecta a la variable del mensaje entrante.
- Clases y arrays. Estos se pasan por referencia, es decir, se pasa la dirección en el parámetro, así que, modificaciones dentro del método afectan externamente.

Con la sentencia return, se finaliza el método. Si además queremos devolver un valor se utiliza la sentencia return valor.

Ampliando la clase Punto, se podrían añadir los siguientes métodos:

```
public class Punto{
 private int x, y;
 public double modulo() {
  double valor = 0;
  // Se calcula el modulo del vector con x,y
  return valor;
 }
 public double fase() {
  double valor = 0;
  // Se calcula la fase del vector con x,y
  return valor;
 }
}
```

Dentro de los métodos podemos definir variables locales, las cuales serán creadas en la ejecución del método y destruidas al finalizar la ejecución. Las variables de método, sólo son visibles desde el propio método en que se definen.

MÉTODOS PARA ACCESOS A ATRIBUTOS (GETTERS & SETTERS)

Realmente, a nivel gramatical, se pueden considerar como métodos genéricos. Pero semánticamente, tienen connotaciones especiales que debemos especificar.

Cada atributo tiene dos posibles acciones: leer su valor o establecerlo. No tenemos por qué realizar ambas acciones, depende del diseño que se busque, de hecho, muchas clases no tiene estos métodos.

Existe una nomenclatura especial para el identificador del método:

- Establecer el valor de un atributo: set + NombreAtributo (la primera letra del nombre del atributo debe estar en mayúsculas). Por ejemplo, en la clase Punto sería setX y setY, en la clase NumeroNatural sería setValor, en la clase Usuario sería setNombre, setEMail, setCiudad, setMovil y setMayorEdad.
- Leer el valor del atributo: get + NombreAtributo o is + NombreAtributo si devuelve un tipo boolean. Por ejemplo, en la clase Punto sería getX y getY, en la clase NumeroNatural sería getValor, en la clase Usuario sería getNombre, getEMail, getCiudad, getMovil y isMayorEdad.

A continuación, presentamos la estructura completa del código de la clase Punto:

```
public class Punto{
 private int x, y;
 public void setX(int x){
  //...
 }
 public void setY(int y){
  //...
 }
 public int getX(){
  return x;
 }
 public int getY(){
  return y;
 }
 public double modulo(){
  //...
 }
 public double fase(){
  //...
 }
}
```

CONSTRUCTORES

Los constructores son métodos especiales que reúnen las tareas de inicialización de los objetos de una clase; por lo tanto, el constructor establece el estado inicial de todos los objetos que se instancian.

No es obligatorio definir constructores, si no se realiza, existe un constructor por defecto sin parámetros.

La ejecución del constructor es implícita a la creación de una instancia.

La restricción sintáctica del constructor es que debe llamarse igual que la clase y no devuelve ningún tipo ni lleva la palabra void.

Como ejemplo añadimos un constructor a la clase Punto:

```
public class Punto{
 private int x, y;
 public Punto(int x, int y){
  //...
 }
}
```

SOBRECARGA DE MÉTODOS

La sobrecarga es definir dos o más métodos con el mismo nombre, pero con parámetros diferentes por cantidad o tipo. El objetivo de la sobrecarga es reducir el número de identificadores distintos para una misma acción pero con matices que la diferencian. La sobrecarga se puede realizar tanto en métodos generales, como en constructores.

La sobrecarga es un polimorfismo estático, ya que es el compilador quien resuelve el conflicto del método a referenciar.

Si definimos un constructor con parámetros, el constructor sin parámetros deja de estar disponible; así que, si nos interesa, se debe definir para su utilización.

Como ejemplo añadimos sobrecarga a los constructores de la clase Punto:

```
public class Punto{
 private int x, y;
 public Punto(int x, int y){}
 public Punto(int xy){}
 public Punto(){}
}
```

Objetos

Recordemos que la clase representa el modelo para crear objetos, pero son los objetos los elementos con los que podemos trabajar. Los objetos ocupan memoria y tiene un estado, que representa el conjunto de valores de sus atributos.

Podemos crear todos los objetos que queramos, todos ellos tienen existencia propia, pudiendo evolucionar por caminos diferentes e independientes.

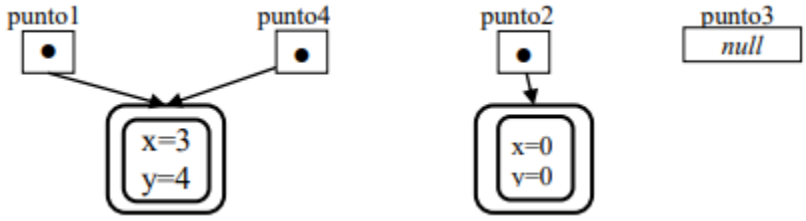
Cuando se crea un objeto se debe asociar con una referencia, por lo tanto, el proceso debe tener dos partes:

1. Declarar una variable que pueda referenciar al objeto en cuestión
2. Crear el objeto y asociarlo a la variable. Para crear el objeto, se realiza con la sentencia new, recordar que lleva implícito llamar al método constructor para que inicialice el objeto

Un ejemplo sería:

```
Punto punto1, punto2, punto3, punto4;
punto2 = new Punto();
punto1 = new Punto(3,4);
punto4 = punto1;
```

En este ejemplo declaramos cuatro puntos de la clase Punto. Cada vez que se utiliza la palabra new, quiere decir que se crea una nueva instancia, por lo tanto, tenemos un total de dos instancias. En el ejemplo se han utilizado dos constructores diferentes para inicializar las instancias. Mostramos a continuación, una gráfico una vez ejecutado todo el código.



Fijarse que las variables punto1 y punto4 referencian la misma instancia, es una manera redundante de acceder. El contenido de una variable de instancia es la dirección de memoria donde se encuentra el objeto. Cuando comparamos variables de instancia, realmente comparamos direcciones de memoria.

Se ha declarado la variable punto3, pero no se ha asociado a ninguna instancia, en ese caso, su contenido es null. Podemos asignar a cualquier variable de instancia el valor null, por ejemplo punto2=null.

En Java, no existe ninguna forma explícita para destruir un objeto, simplemente cuando un objeto se queda huérfano (no tiene ninguna referencia) se destruye. Esta labor la realiza el recolector de basura (garbage collector) de forma automática. En el ejemplo anterior, si asignamos a punto2 un null, queda su instancia sin referencia, por consiguiente, cuando se lance el recolector de basura destruirá el objeto huérfano.

Notar que cuando creamos un array NO implica la creación de los objetos. Veamos un ejemplo en tres pasos para entender mejor el proceso.

- En primer lugar se declara una variable de tipo array de Punto.

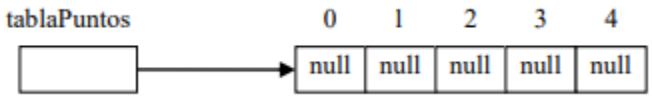
  ```
  Punto[] tablaPuntos;
  ```

  El contenido de la variable tablaPuntos es null.

- En segundo lugar se crea el array.

```
Punto[] tablaPuntos = new Punto[5];
```

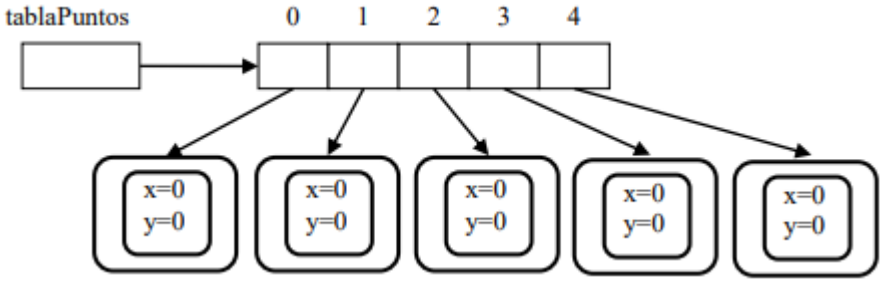
Gráficamente se describiría como:



- En tercer lugar se crean los objetos.

```
for (int i = 0; i < 5; i++)
 tablaPuntos[i] = new Punto;
tablaPuntos[i]= new Punto();
```

Gráficamente se describiría como:



Mensajes

Una vez creado los objetos es posible pasarles mensajes, o invocarles métodos, para solicitar una acción. El objeto establece a que método le corresponde el mensaje recibido.

Para enviar un mensaje se realiza con la notación punto. Dependiendo si almacenamos el valor devuelto del método, tenemos dos posibilidades:

```
referencia.metodo(param1,param2...);
referencia.metodo();
var = referencia.metodo(param1,param2...);
var = referencia.metodo();
```

La segunda línea es para guardar el valor devuelto por el mensaje.

Ponemos a continuación un ejemplo con la clase Punto:

```
Punto punto1 = new Punto(2,5);
punto1.setX(4);
int x = punto1.getX();
double d = punto1.modulo();
punto1.setX(punto1.getX() + 1); // suma en una unidad el valor de x
```