UNIDAD

1

DIPLOMATURA EN PROGRAMACION JAVA

MÓDULO 1: PROGRAMACIÓN ORIENTADA A OBJETOS EN JAVA

Clases y Objetos en Java

Clases y Objetos en Java

Esta unidad explica los fundamentos de programación en Java para la creación de clases y objetos. También se enseña la forma de declarar variables y su clasificación

Variables

Las variables en un lenguaje de programación es el medio por el cual se almacenan valores transitoriamente en un programa. Particularmente, en un lenguaje de programación orientado a Objetos como Java, las variables son directamente asociadas a los atributos de las clases, aunque se utilicen en otros lugares que no sean directamente atributos, como los parámetros de un método. Si bien es cierto que en otros lenguajes orientados a objetos existen variables fuera de las clases, en Java no. Por lo tanto en este lenguaje hablar de variables o atributos es lo mismo salvo que estén en un método.
Las variables se diferencian en un programa por sus nombres y su tipo. Estos siguen ciertas reglas de escritura a las que están circunscriptas los identificadores. La razón de esto es para evitar ambigüedades entre posibles operadores e instrucciones y los nombres de los identificadores.
En Java se puede hacer una primera clasificación de las variables según dos grupos:

- Tipos primitivos
- Tipos Referenciados

La principal diferencia entre estos dos grupos es que el primero son los tipos de datos prexistentes definidos en el lenguaje mientras que los segundos son aquellos que pueden almacenar las referencias que se obtienen al crear un objeto de cualquier clase que se haya definido.
Es necesario mencionar un área gris en el lenguaje. Hay ciertos tipos referenciados, o sea, que provienen de clases definidas, que el lenguaje los maneja como tipos prexistentes. El motivo de esto es que dichas clases se consideraron con la importancia suficiente como para que el lenguaje las maneje internamente. Este es el caso del tipo String, como se verá posteriormente.

Propiedades de las variables

Las variables en Java poseen propiedades que permiten su control y manejo a lo largo de un programa. Las propiedades de una variable son las siguientes:

- Son espacios en memoria que almacenan datos
- Siempre tienen almacenado un valor desde su declaración
- Siempre deben ser declaradas por un tipo primitivo o referenciado
- Poseen visibilidad y alcance (lugares desde donde se pueden acceder y lugares desde los cuales no)
- Se pueden utilizar como parámetros de los métodos
- Se puede retornar el valor que almacenen desde un método

El manejo de estas propiedades es lo que garantiza el buen uso de las mismas, por lo tanto, si se puede asociar una variable con los atributos de una clase, es fundamental dominar el concepto para saber donde utilizarlas y como.

Identificadores

Hay una regla muy simple para determinar si algo es un identificador:

> *Cuando en el código un programador debe decidir que nombre ponerle a un elemento, dicho elemento es un identificador*

Como se puede apreciar, dentro de esta característica entran varios de los elementos antes mencionados, como ser, nombres de clases, variables, métodos, etc...
Hay otros elementos del lenguaje Java todavía no mencionados que son identificadores. Estos se irán descubriendo como tales a medida que se aprenda más del lenguaje siguiendo la simple regla que se formuló.
Es preciso aclarar que cualquier palabra que el lenguaje defina como **reservada** o **clave**, no podrá ser utilizada como identificador.
Por otro lado, crear un identificador es asignarle un nombre a un elemento que permite definir un lenguaje de programación. Esta asignación de nombres debe seguir ciertas normas prestablecidas en el formato del mismo que variarán de lenguaje a lenguaje. En el caso de Java, las reglas a seguir son las siguientes:

- El primer carácter de un identificador debe ser uno de los siguientes:
  - ✓ Una letra en mayúsculas (A~Z)
  - ✓ Una letra en minúsculas (a~z)
  - ✓ El carácter de subrayado (_)
  - ✓ El símbolo pesos o dólar ($)
- Del segundo carácter en adelante:
  - ✓ Cualquier elemento de los que sirve para el primer carácter
  - ✓ Caracteres numéricos (0~9)

Vale la pena mencionar que el espacio en blanco no es un carácter permitido como se puede apreciar, por lo tanto no debe utilizarse para nombrar un identificador.
Además existen ciertas normas estandarizadas para crear identificadores, que si bien no son obligatorias, han sido adoptadas por la mayoría de los programadores y seguirlas ayudan mucho a la legibilidad del código. Algunas de ellas son las siguientes

- Si es una clase, el nombre debe comenzar con mayúsculas
- Si es un método o una variable, el nombre debe comenzar con minúsculas
- Si el nombre tiene más de una palabra, a partir de la segunda palabra separarlas sólo comenzando con mayúsculas (primera letra de cada palabra a partir de la segunda)

No se debe olvidar el hecho que Java es un lenguaje sensible al caso (diferencia entre mayúsculas y minúsculas), por lo tanto si dos identificadores son iguales en su significado a la lectura pero se diferencia tan sólo en el caso de una letra, el lenguaje los considerará identificadores distintos.

Tipos primitivos y referenciados

Todo elemento que tenga las propiedades de una variable es un tipo. Por ejemplo, un objeto, que es un tipo referenciado, tiene las mismas propiedades de una variable.
Los tipos primitivos de Java permiten clasificarlos según su uso en los siguientes grupos:

- Tipos enteros
  - `byte`
  - `short`
  - `int`
  - `long`
- Tipos de punto flotante
  - `float`
  - `double`
- Tipo texto
  - `char`
- Tipo lógico
  - `boolean`

Las declaraciones de las variables se realizan con el siguiente formato UML:

[<modificador>] *<tipo primitivo> <identificador> [= valor inicial];*

En otras palabras:

- Si aparece "<>", quiere decir "elegir uno entre los posibles"
- Si aparece "[]", quiere decir que es opcional
- Si una palabra o símbolo aparece sin ninguna otra cosa, indica que ponerlo es obligatorio

Este formato debe leerse de la siguiente manera:

- **Modificador** (opcional): Es el modificador de visibilidad de la variable (indica como se lo puede utilizar y desde donde). Las posibilidades son:
  - `public`
  - `private`
  - `protected`
  - Sin modificador
- **Tipo**: Es obligatorio poner un tipo, pero se debe elegir uno de los posibles
- **Identificador**: Es obligatorio poner un identificador, se debe elegir cual nombre poner
- **Asignación inicial**: Existe la opción de elegir poner un "=" y un literal de asignación si se desea que la variable tenga un valor inicial
- **Fin de la declaración**: Siempre hay que finalizar la declaración con un ";" que indica fin de línea de programa

Nota: En este punto es bueno señalar que en Java toda sentencia termina con un ";". El otro símbolo que indica el fin de algo es la "}", pero se utiliza para ciertas declaraciones a las que se llaman bloques

Ejemplos

```
int nroEmpleados=10;
```

```
int edad=20;
float sueldoFijo;
long documento;
```

Nota: Cuando a una variable se le asigna un valor numérico dentro del código, a dicho número se lo identifica como "literal de asignación". El lenguaje toma esos caracteres y los convierte al valor binario que debe almacenar en la memoria para dicha variable

Tipos referenciados

Cada vez que se crea una clase se define un nuevo tipo, pero para poder hacer uso de él, se debe crear un objeto. Esto es similar a declarar una variable de un tipo base con la diferencia que la memoria requerida para la operación será determinada por los elementos declarados en dicha clase.
Por otro lado, el lenguaje utiliza la definición de la clase como plantilla de la memoria a utilizar y en base a esto se realiza la creación del objeto.
En Java cuando se declara un objeto se utiliza un operador diseñado para este fin: **new**. Este operador es quien toma la clase como molde y reserva la memoria para la creación del objeto.
Sin entrar en detalles acerca de la manera en que esto se realiza, el operador efectúa las operaciones necesarias en memoria y luego devuelve el valor de la dirección o referencia de la memoria donde lo hizo, de ahí el nombre de las declaraciones de tipo referenciados.
Evidentemente el valor devuelto debe almacenarse en algún lugar para su posterior uso y ese lugar es una variable de tipo referencia. Como este tipo de variables almacenan la dirección de memoria donde se creo el objeto y como toda clase genera un nuevo tipo, es evidente que la declaración debe efectuarse con la clase que la define.
De esta manera, con el nombre de la clase seguido del identificador elegido se crea la variable referencia. Cuando esta deba almacenar el valor devuelto por el operador **new**, al identificador lo seguirá un operador de asignación "=", el operador **new** y por último en nombre de la clase que indica el tipo de objeto a crear.
El formato en UML es el siguiente:

*<nombre del tipo clase><identificador>[= **new** nombre del tipo clase ()];*

Al igual que las variables, se pueden crear tipos referenciados sin necesidad de inicializarlos con un valor (referencia al objeto creado por **new**), pero se debe usar el operador posteriormente para crear un objeto y utilizarlo.

Ejemplos

```
Empleado e = new Empleado();
Persona p = new Persona();
Persona p2;
```

Comentarios

En Java existen dos formas de poner comentarios dentro del código y son los siguientes

```
// La doble barra se usa si el comentario abarca una línea
// de comentario
/* Comienza el comentario
La combinación barra – asterisco para comenzar y asterisco- barra
para finalizar se utiliza el comentario abarca más de una línea
Finaliza el comentario */
```

Explorando los tipos primitivos

Tipos entero

Existen cuatro tipos enteros y se diferencian en su capacidad de almacenamiento como muestra la tabla

Nombre del tipo	Longitud	Rango
`byte`	8 bits	$-2^7 \sim 2^7-1$
`short`	16 bits	$-2^{15} \sim 2^{15}-1$
`int`	32 bits	$-2^{31} \sim 2^{31}-1$
`long`	64 bits	$-2^{63} \sim 2^{63}-1$

Ejemplos

```
int a;
int b=10;
```

El lenguaje por defecto convierte a todo literal de asignación en un tipo entero `int`, por lo tanto cuando se asigna un literal a otro tipo entero diferente se debe especificar con una letra o un modificador de tipo (cast), como se indica a continuación:

```
long p = 10L;
long n = (long)10;
short s = (short)5;
byte b = (byte)1;
```

Las formas utilizadas para `p` y `n` son análogas. La letra "`L`" utilizada en la asignación a la variable `p` es un camino corto que provee el lenguaje. El lenguaje permite la omisión de las conversiones de tipo en los primeros dos ejemplos debido a que `long` es de mayor tamaño que `int`. A esto se lo denomina **promoción automática**.

Tipos punto flotante

Existen dos tipos primitivos de punto flotante y se diferencian por su capacidad de almacenamiento. Las variables de punto flotante en Java pueden utilizar notación científica, por eso no es posible asignar un rango de valores que pueden almacenar. La siguiente tabla muestra los tipos

Nombre del tipo	Longitud
`float`	32 bits
`double`	64 bits

Java toma la asignación de un literal numérico en punto flotante por defecto como un `double`, por lo tanto si es un `float` hay que especificarlo de una de las siguientes formas:

```
float f = 10.5F;
float g = (float)10.5;
```

Tipo Texto

Este tipo se utiliza para almacenar un solo carácter. Tiene la capacidad de almacenar cualquier carácter en formato Unicode (formato estándar de cualquier carácter en Java). Se debe notar que sólo almacena un carácter, para almacenar una palabra o tan sólo más de un carácter, se debe utilizar otro tipo de datos que provee el lenguaje, el tipo `String`. Otro punto importante es que cuando se asigna en el código un literal carácter, este debe ir rodeado de comillas simples, como se muestra a continuación:

```
char c = 'a';
```

Tipo booleano

Este tipo sólo puede almacenar dos valores: **`true`** o **`false`** (que son literales que internamente se convierte en 0 y 1 respectivamente).
Los valores que almacenan son considerados palabras clave en el lenguaje y se deben utilizar para cualquier asignación a variables de este tipo. Generalmente es utilizado para la toma de decisiones.
Ejemplo:

```
boolean b = false;
```

Métodos en Java

Los métodos, o como se los llama en otros lenguajes "funciones", son los elementos de una clase mediante los cuales se define una operación que un objeto de su tipo puede realizar. La declaración de un método en Java siempre se debe realizar dentro de una clase y puede adoptar el siguiente formato:

```
[<modificador>] <tipo retornado> <identificador>([lista de argumentos]){
        [declaraciones y / o métodos a utilizar]
        [return [valor retornado];]
}
```

El formato de la declaración del método es incompleta (falta la posibilidad de declarar excepciones, pero este tema escapa al contenido del curso), se usará así de momento porque la parte faltante es un tema avanzado para este punto.
El formato quiere decir lo siguiente:

- (opcional): (opcional): Es el modificador de visibilidad del método (indica como se lo puede utilizar y desde donde). Las posibilidades son:
  - **`public`**
  - **`private`**
  - **`protected`**
  - Sin modificador
- **Tipo retornado**: se debe elegir entre un tipo primitivo o uno referenciado. Si el método no devuelve ningún valor se puede indicar poniendo en este lugar la palabra clave **`void`**.
- **Identificador**: es el nombre que se le asignará al método y mediante el cual se lo invocará
- **Lista de argumentos**: en este lugar se indican los parámetros que recibirá el método. Los argumentos se declaran igual que las declaraciones de variables con la diferencia que no se pone el ";" final. Si el método posee varios argumentos, se

deben separa unos de otros con el operador de continuación de declaración (“,”). Si el método no recibe argumentos, se dejan sólo los paréntesis del mismo.

- **Comienzo del bloque de sentencias**: Obligatoriamente poner la llave de apertura de bloque de sentencias
- **Sentencias**: Opcionalmente agregar declaraciones e invocaciones a métodos. Existe la posibilidad de no poner nada a lo que se llama método de cuerpo vacío.
- **Fin de ejecución de sentencias del método**: Opcionalmente poner una sentencia `return` para terminar la ejecución del método. Si se indico que el método retorna un valor (recordar que puede ser `void`), este se debe poner a continuación.
- **Fin del bloque de sentencias**: Poner obligatoriamente la llave de cierre

Ejemplo

```
public int calculaVolumen(int ancho, int largo, int alto){
//bloque de sentencias
}
```

Este método es público, devuelve un entero y recibe tres enteros como parámetros

Clases en Java

Tienen el siguiente formato:

```
<modificador> class <identificador>{
    [declaraciones de variables y métodos]
}
```

Ejemplo

```
public class Ejemplo{
    private int var1;
    private int var2;
    public int getVar1(){return var1;};
}
```

No se debe olvidar que cuando se declara una clase se crea un nuevo tipo. En este caso dicho tipo es Ejemplo. Las clases son sólo moldes o plantillas para los objetos de su tipo que se creen. Se puede apreciar en ella la interfaz de la clase declarada con el modificador `public`.

Constructores

Las clases tienen un método especial que se denomina constructor. Cuando se crea un objeto mediante la creación de una instancia de una clase, luego que se reserva en memoria los lugares de almacenamiento necesarios para el objeto, se ejecuta siempre el constructor. Cómo es un método especial se diferencia de los otros por las siguientes dos características:

- Se llaman igual que la clase
- Nunca se le pone el valor retornado

Si la clase no posee uno, como en el ejemplo anterior, se ejecuta uno por defecto que el lenguaje provee más allá que no se especifique en el código.
Además, si se lo desea, se puede poner más de un constructor. La existencia de más de un constructor tiene su explicación en otra herramienta que proveen los lenguajes orientados a objetos llamada sobrecarga. Este tema se explicará posteriormente, por el momento alcanza

con saber que para poner más de un constructor se debe cumplir al menos una de las siguientes reglas:

- La cantidad de los parámetros es diferente
- El tipo de los parámetros es diferente

Los constructores permiten que se les pase parámetros, pero cuando se declara un objeto se le deben pasar tantos parámetros como los que figure en alguno de sus constructores o la declaración será un error. De esta manera, si se le quiere poner a la clase anterior un constructor que reciba un entero, quedaría con el siguiente formato:

```
public class Ejemplo{
      public Ejemplo(int v){ var1=v;}
      private int var1;
      private int var2;
      public int getVar1(){return var1;}
}
```

La declaración de un objeto de esta clase sería

```
Ejemplo obj = new Ejemplo(8);
```

Una descripción general (sin entrar en detalles como la reserva de memoria) de lo que ocurre cuando se declara un objeto de una clase que tiene constructor, como en el ejemplo anterior, es lo siguiente:

- Se reserva el espacio en memoria para el objeto
- Se le pasa el parámetro 8 al constructor
- Se empieza a ejecutar el constructor y se asigna el valor del parámetro a la variable `var1`
- Se termina la ejecución del constructor
- Se devuelve la referencia al objeto y se almacena en la variable `obj`

Uso de un objeto

Para utilizar los servicios que presta un objeto se invocan los elementos declarados en su interfaz pública.

En la clase `Ejemplo` el único elemento en dicha interfaz es el método `getVar1`. Para poder invocarlo se debe utilizar la notación de punto y la referencia almacenada en la variable `obj`, ya que esta indica el lugar de almacenamiento del mismo. El código en un programa se vería de la siguiente manera:

```
int aux;
aux = obj.getVar1();
```

En este código se declara la variable entera `aux` y luego se almacena en ella el valor retornado por el método `getVar1()`.

Ocultamiento de la información

Para proteger los datos que se almacenan dentro de un objeto, la clase utiliza la declaración de visibilidad **`private`** para que no se pueda acceder la variable, ya que esta declaración indica que la variable no pertenece a la interfaz de la clase. Cuando un elemento no pertenece a la interfaz no se puede acceder por notación de punto una vez creado un objeto, sólo lo podrán acceder aquellos elementos que pertenezcan a la clase.

Ejemplo

Clase Persona

```
public class Persona {
	private String primerNombre;
	private String segundoNombre;
	private String apellido;
	private String documento;

	public Persona() {
	}
	public String getApellido() {
		return apellido;
	}
	public String getDocumento() {
		return documento;
	}
	public String getPrimerNombre() {
		return primerNombre;
	}
	public String getSegundoNombre() {
		return segundoNombre;
	}

	public void setApellido(String string) {
		apellido = string;
	}
	public void setDocumento(String string) {
		documento = string;
	}
	public void setPrimerNombre(String string) {
		primerNombre = string;
	}
	public void setSegundoNombre(String string) {
		segundoNombre = string;
	}
}
```

Encapsulado

Como se mostró con anterioridad, en toda clase existe una parte pública y una privada. La primera define los elementos de la clase que son accesibles a través de su interfaz. Muchas veces se hace referencia a este hecho como "accesible por el mundo exterior o el universo". Esta frase puede simplificarse de la siguiente manera:

> *Los elementos públicos de una clase serán accesibles por notación de punto una vez creado un objeto de su tipo*

En cambio, la parte privada, define todo lo contrario. Los elementos declarados como privados en una clase no serán accesibles por ningún elemento salvo que este se encuentre en la misma clase.

Este último concepto es el que permite separar los servicios que brinda un objeto de la forma en que lo hace, por lo tanto, en otras palabras, un objeto debe verse como una serie de servicios que presta a través de sus interfaces y la forma en que lo hace se oculta del mundo exterior porque no es accesible.

Cuando una clase oculta una serie de operaciones (métodos y variables de la clase que éstos usan) declarándolos privados y los utiliza posteriormente para brindar un servicio, se dice que este servicio esta encapsulado dentro de la clase.
Otra forma común de denominar a las operaciones o métodos privados de una clase es **servicio privado**.
Se puede decir que el ocultamiento de la información y los métodos privados son en conjunto conocidos como "encapsulado", estén presente uno de ellos o ambos.
Un ejemplo claro de esto es cuando en una clase se oculta la información pero se quiere brindar la posibilidad de acceder a los datos almacenados, ya sea para guardar valores como para leerlos. En este caso se deben crear métodos públicos que cumplan ese rol.

Ejemplo

Clase Persona

```
public class Persona0 {
	private String primerNombre;
	private String segundoNombre;
	private String apellido;
	private String documento;
	private String detalles;

	public Persona() {
	}
	public String getApellido() {
		return apellido;
	}
	public String getDocumento() {
		return documento;
	}
	public String getPrimerNombre() {
		return primerNombre;
	}
	public String getSegundoNombre() {
		return segundoNombre;
	}

	public void setApellido(String string) {
		apellido = string;
	}
	public void setDocumento(String string) {
		documento = string;
	}

	public void setPrimerNombre(String string) {
		primerNombre = string;
	}
	public void setSegundoNombre(String string) {
		segundoNombre = string;
	}
	public String getDetalles(){
		armaDetalles();
		return detalles;
	}
```

```
            private void armaDetalles(){
                  detalles = "Apellido: "+ apellido + " Primer Nombre: "
                  + primerNombre + " Segundo Nombre: " + segundoNombre +
                  " Documento: " + documento + " Direccion: " +
                  objDireccion.getNombreCalle()+ " " +
                  objDireccion.getNro()+  " " + objDireccion.getPiso()+
                  "°" + objDireccion.getDpto();
            }
}
```

Strings

Las cadenas de caracteres o “strings” en Java se manejan con una clase interna del lenguaje, por lo tanto, siempre será un tipo referenciado.
El lenguaje permite dos formas de asignar una cadena de caracteres: cuando se crea el objeto y mediante el operador de asignación

Ejemplo

```
            String nombre = new String(“Juan”);
            String apellido = “Perez”;
```

Comienzo de un programa en Java

En Java, el comienzo de un programa se coloca dentro de una clase en un método. A diferencia de otros métodos escritos por el programador este tiene un nombre prestablecido: **main**.
Este método es el punto de entrada para el comienzo de un programa.
Como no esta definido que un programa deba comenzar en una clase en particular, esto indica que puede haber muchas clases que posean el método **main**, pero sólo puede haber un método de este tipo por clase.
El intérprete de Java sabe por donde arrancar el programa porque el primer argumento que recibe para empezar a ejecutarlo debe ser la clase que posee el método **main** que se desea utilizar como comienzo de programa.

Ejemplo

```
public class UsaPersona0 {
      public static void main(String[] args) {
            Persona0 p = new Persona0();
      }
}
```

Paquetes

Los paquetes son los lugares físicos y lógicos donde se almacenan las clases. Tienen una relación con los directorios del sistema de archivos en cada sistema operativo, de manera que el nombre de un paquete donde se guardan las clases es igual al nombre del directorio en el sistema operativo donde residen las mismas.
Se definen al principio de toda declaración de clase y debe ser la primera sentencia cuando se crea una clase. Si se omite dicha declaración, el lenguaje considera que debe almacenar la clase en el paquete por defecto (este coincide con el directorio definido como raíz de los paquetes y su valor se almacena en la variable CLASSPATH. En los entornos de trabajo esta variable se define internamente con el directorio en el cual se crea el proyecto).

Es el lugar donde se guarda la clase compilada (archivos con extensión .class), no los archivos fuentes (que tiene extensión .java). Sin embargo, algunos entornos de trabajo guardan en el mismo directorio ambos archivos

Cuando se declara un paquete, se define un espacio de nombres, por lo tanto no es sólo un lugar de almacenamiento, sino que también es un espacio que define una visibilidad. Por lo tanto, si se quiere utilizar una clase que se encuentra en un paquete, se debe definir explícitamente el acceso al mismo.

Se declara con la sentencia `package`. Si el paquete no existe, el compilador lo crea (así como también el directorio asociado al nombre en el sistema de archivos del sistema operativo). Si el paquete existe, sólo incorpora la clase compilada a éste.

Para acceder a una clase dentro de un paquete se debe utilizar la sentencia `import`, porque como se mencionó anteriormente, los paquetes definen visibilidades, por lo tanto, se debe indicar donde "mirar" para utilizar la clase.