Elementos básicos de Java

¡Comencemos!

¿Qué aprenderás?

- Reconocer los elementos básicos de Java para usarlos e implementarlos al momento de construir algoritmos desarrollados en el lenguaje.
- Emplear comentarios, variables primitivas, variables de referencia a objetos, y constantes en Java.

Introducción

A continuación revisaremos los elementos básicos para empezar a codificar pequeñas piezas de software. Comenzaremos con los comentarios y, posteriormente, con los distintos tipos de variables que encontraremos en Java y cómo realizar operaciones con ellas.

¡Vamos con todo!

Comentarios

Los comentarios son líneas de código que son ignoradas por el compilador, pero sirven para dar indicaciones y documentación a nuestro código.

```
// Esta línea es un comentario
// Los comentarios son ignorados
// Pueden ser una línea nueva
int a = 2 // O puede acompañar una línea de código existente
// a = 2 + 3 Si comentamos al principio todo la línea será ignorada
```

Comentarios en múltiples líneas

Existe otra forma de hacer comentarios, donde se puede abarcar una mayor cantidad de código, como por ejemplo cuando queremos comentar un método completo. Para esto utilizaremos `/*` al inicio y `*/` al final, donde todo lo que esté encerrado entre esos símbolos quedará comentado.

```
/* Todo el código
escrito
en estas líneas
será ignorado
*/
int i; //esta variable ya no será ignorada
```

Los ejemplos que vayamos utilizando y los resultados a las operaciones las iremos dejando comentadas.

Introducción a variables

- Una variable en Java es un identificador que representa una palabra que contiene información.
- El tipo de información almacenado en la variable, solo puede ser del tipo con que se declaró la variable.
- Se denominan variables ya que el contenido que almacenan puede variar.

Una variable se compone de:

- Un nombre o identificador.
- Un valor.
- Tipo de dato.

Por ejemplo, podemos tener las siguientes variables:

Declaración	Identificador	Tipo	Valor
int i;	i	Entero	no asignado
int b = 2;	b	Entero	2
String s;	s	Referencia a String	no asignado
boolean a = false;	a	Booleano	false

Tabla 1. Partes de una variable.
Fuente: Desafío Latam.

Tipos de datos primitivos

Los tipos de datos primitivos almacenan directamente un valor que siempre va a pertenecer al rango de ese tipo. Acá podemos encontrar los tipos de datos como: int, short, float, boolean, double, char, byte, entre otros.

- **int**: Los números enteros son números sin decimales que pueden ser positivos o negativos. En inglés se les denomina Integers.

```
int a = 2;
```

- **float:** Los números flotantes son números que pueden tener hasta 7 dígitos decimales:

```
float a = 3.5f;
```

- **double:** Corresponde a un dato "float x2", son números que pueden tener hasta 15 dígitos decimales:

```
double a = 3.5d;
```

- **boolean:** El tipo de dato boolean almacena únicamente dos valores, verdadero o falso:

```
boolean activo = true;
boolean alto = false;
```

- **char:** El tipo de dato char representa un único caracter y se escribe entre comillas simples `' '`.

```
char valor = 'c';
char otroValor = 'k';
```

Referencias a objetos

Otro tipo de variables son las de referencias a objetos que profundizaremos más adelante cuando veamos los conceptos de clases y objetos.

De momento necesitamos saber que el tipo de dato String es una referencia a un objeto, y con ella podremos realizar operaciones que con las variables de tipo primitiva no podremos.

String

El String básicamente es una cadena de caracteres, en la que podemos almacenar ya sea una palabra, frase o texto. Todo String debe ser escrito entre comillas `" "`.

```
"Esto es un String"
"hola"
"a"
```

Todos los ejemplos son distintos tipos de Strings.

Creando un String

Cuando creamos el Hola Mundo! en el ejemplo anterior, escribimos:

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

Ahí se creó implícitamente un String.

También se pudo haber creado una variable de tipo String y asignarle el valor:

```
String cadena = "Hola Mundo!";
```

Y como alternativa también se puede crear como:

```
String cadena = new String("El primer programa");
```

Si queremos crear un String vacío (o nulo) existen estas alternativas:

```
//Considerar el vacío como un dato vacío
String cadena = ""; //Es un String vacío
String cadena = new String(); //Crea una referencia a un objeto
//null o nulo no es un dato, es un prospecto a ser un string
String cadena = null;
//Después de la línea anterior, se debe crear el objeto
cadena = new String();
/*Nótese que no se coloca la palabra reservada String, debido a que la
variable objeto cadena ya se declaró como String en la línea anterior */
```

Declaración de un String vacío

Un string nulo es aquél que no contiene caracteres, pero es un objeto de la clase String .Sin embargo, al escribir:

```
String cadena;
```

Está declarando un objeto cadena de la clase String , pero aún no se ha creado ningún objeto de esta clase.

Esto quiere decir que para usar la variable `cadena`, debemos usar una de las alternativas antes mencionadas.

Operando con variables

Con los diferentes tipos de variables podemos realizar distintos tipos de operaciones. Ahora trabajaremos con los números enteros `int` y las cadenas de caracteres `String`.

Sumas y restas

Podemos operar con variables de tipo entero, como si estuviéramos operando con el número mismo, por ejemplo:

```
int a = 4;
int b = 1;
System.out.println(a+3); //7
System.out.println(b-a); //-3
```

Acá usamos otro método de `System.out.println("variable")`, donde muestra el resultado en una línea y automáticamente hace el salto de línea, sin tener que usar `\n`.

Multiplicaciones y divisiones

```
int a = 10;
int b = 3;
System.out.println(a*3); //30
System.out.println(a/b); //3
```

Observamos que al dividir 10/3 el resultado que nos entrega es 3, ya que estamos trabajando con números enteros y no flotantes.

Sobreescribiendo variables

Podemos reemplazar el valor de una variable realizando una nueva asignación, pero teniendo en cuenta el tipo de dato de dicha variable.

```
int a = 10;
a = 3;
System.out.println(a); //3
```

Sumando enteros y Strings

Veamos qué pasa si tratamos de sumar dos variables de tipos distintas:

```
int a = 3;
String b = "dos";
System.out.println(a+b); //3dos
```

Pero si tratamos de guardar la suma en otra variables:

```
int a = 3;
String b = "dos";
int c = a+b;
System.out.println(a+b); //3
```

En la consola aparece que no se puede convertir un String a int, ya que la lógica dice que un número puede ser un String, sin embargo, un String no puede ser un número, ya que este no asegura que sea un String con un dato numérico, si no que puede ser cualquier palabra. Más adelante veremos cómo realizar esta operación.

Modificando una variable existente

También podemos modificar el valor de una variable operando sobre ella misma, este tipo de operación es muy utilizada:

```
int a = 4;
a = a + 1;
System.out.println(a); //5
```

Variables de tipo String

¿Qué obtendremos al sumar dos Strings?

```
String a = "Hola";
String b = " Mundo";
System.out.println(a+b);
//Hola Mundo
```

En programación la acción de "sumar" dos o más Strings se conoce como **concatenación**.

Creando un String a partir de variables

Podemos generar Strings a partir de otros Strings y variables usando especificadores y el método `format` de String.

```
int edad = 34;
String nombre = "William";
String salida = String.format("%s tiene %d años.", nombre, edad);
System.out.println(salida);
//William tiene 34 años.
```

Donde `%s` indica que el tipo de dato que tiene la primera variable (en este caso es de tipo String), `%d` indica que el tipo de dato es de tipo entero para la variable edad.

Otros especificadores para el formato son los siguientes:

```
// int %d
// char %c
// float %f
// String %s
```

Aquí podemos entender mejor cómo usar printf, que se comporta de la misma manera que String.format, donde primero se indica el formato que tendrá el String y luego las variables a utilizar.

Buscando un String

Aquí tenemos dos casos para encontrar una subcadena dentro de un String:

- **substring(int startIndex)**: retorna un nuevo String que contiene el String desde el índice indicado.
- **substring(int startIndex, int endIndex)**: retorna un nuevo String que contiene el Strings desde el índice indicado hasta el índice final exclusivo.

```
String s="Paralelepipedo";
System.out.printf("%s\n",s.substring(4)); // lelepipedo
System.out.printf("%s\n",s.substring(0,4));// Para
```

Información del String

Podemos ver la longitud de un String

```
String cadena = "Mi primer programa";
int longitud = cadena.length(); // 18 - considera los espacios
System.out.println(longitud);
```

Si comienza con un determinado sufijo

```
String cadena = "Mi primer programa";
// Devuelve true si comienza con el sufijo especificado
boolean resultado = cadena.startsWith("Mi");
System.out.println(resultado);
```

En este ejemplo la variable resultado tomará el valor `true`.

Si se quiere obtener la posición de la primera ocurrencia de la letra `'p'`

```
//Desde cero, la p esta en la posición 16
String cadena = "que clavo clavo pepito?";
int pos = cadena.indexOf('p'); // 16 - se comienza a contar desde cero
System.out.println(pos);
```

En caso de no existir, retornará el valor -1.

Constantes

Una constante sirve para almacenar un valor que no va a cambiar.

Para definir una constante se escribe de la siguiente manera:

```
final int DIAS_SEMANA = 7;
final int MAX_ITERACIONES = 10;
```

Añadiendo antes del tipo de dato la palabra “final”, lo que hará que estos valores sean fijos y no sean modificables durante la ejecución del programa.

Convención sobre las constantes
Una convención es un estándar implantado como buenas prácticas en la programación. Dicho esto, para definir el nombre de una constante, esta debiese ser escrita completamente con mayúsculas, y si es una palabra compuesta, es decir más de dos palabras, cada palabra debe ir separada por un guión.

Entrada de datos
Nuestro programa podría necesitar interactuar con el usuario, ya sea para seleccionar una opción de un menú o algún valor sobre el cual el programa va a operar.

Para realizar dicha acción utilizaremos la clase Scanner que permitirá acceder a lo que vayamos ingresando por teclado.

```
Scanner sc = new Scanner(System.in);
String cadena = sc.nextLine()
```

Pero antes, al inicio del fichero debemos agregar `import Java.util.Scanner;`

```
package testeandoTiposDeDatos;
import java.util.Scanner;
public class testeandoTiposDeDatos{
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in); //Se crea el objeto Scanner
        String i = sc.nextLine(); //Se lee una línea
        System.out.println(i);
    }
}
```

Con esto obtendremos la línea ingresada como String en la consola, o bien, podemos obtener el próximo Entero (`nextInt()`), Flotante (`nextFloat()`), y más.

Ventajas del IDE Eclipse

Cuando no sabemos un nombre de un método que queremos utilizar, al escribir el punto luego de sc, aparecerá una lista de sugerencias con las distintas acciones que podemos escribir.

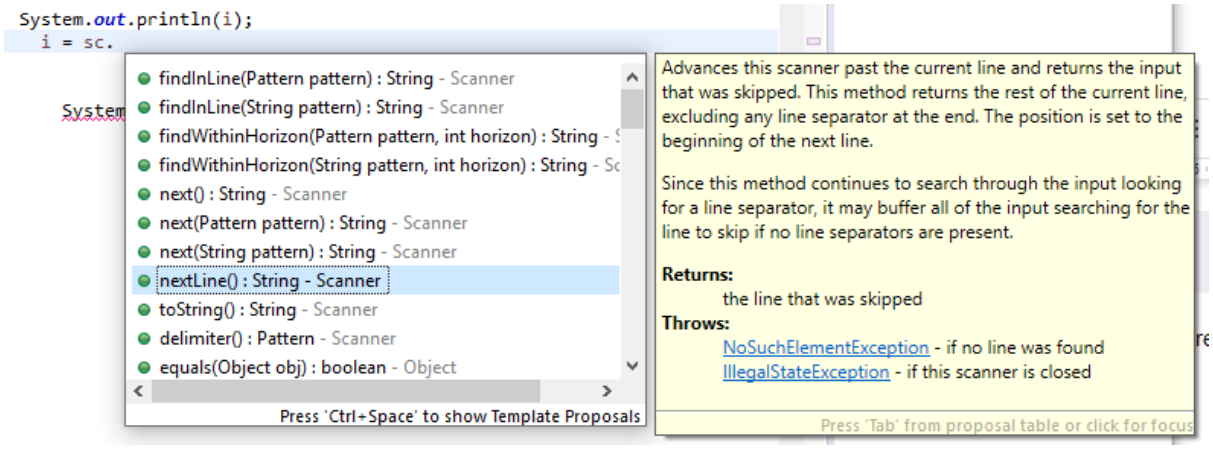

Imagen 1. Mostrar sugerencias de métodos en Eclipse IDE.
Fuente: Eclipse IDE.

Transformación de datos

Si queremos pasar un numero flotante a entero, por ejemplo, debemos hacer:

```
float a = 8.61f;
int b;
b = (int)a;
System.out.println(b);//El resultado será 8, es decir la variable a sin los
decimales
```

Así le estamos diciendo a la variable a de tipo flotante que tome solo la parte entera de ella.

En cambio si queremos transformar un número a String, debemos hacer lo siguiente:

```
int number = -782;
String numeroAString = String.valueOf(number);
System.out.println(numeroAString);// El resultado será "-782"
String numeroAString2 = String.valueOf(-782);
System.out.println(numeroAString2);// El resultado será "-782"
```

Y si queremos transformar un String a entero tendremos que utilizar la clase Integer, no el tipo de dato primitivo de entero para realizar esta acción.

```
String a = "45";
//La variable String debe ser númerica o si no habrá un error
int numero = Integer.parseInt(a);
```

Ejercicio guiado: Utilizando String y formato

Queremos escribir texto para el destinatario de una encomienda. Para ello debemos pedir al usuario los distintos campos que se requerirán:

```
Scanner sc = new Scanner(System.in);
String nombre = sc.nextLine(); //Carmen
String apellido = sc.nextLine(); //Silva
String direccion = sc.nextLine(); //Los aguiluchos
int numeroDireccion = sc.nextInt(); //43
String ciudad = sc.nextLine(); //Concepción
int telefono = sc.nextInt(); //562264895;
```

Para crear el formato de la etiqueta podemos hacerlo de varias formas:

```
String etiqueta = String.format("DE:%s %s\nDirección: %s %d\nCiudad:
%s\nContacto:%d\n",nombre, apellido, direccion, numeroDireccion, ciudad, telefono);
System.out.println(etiqueta);
```

Recordemos que si tenemos dos String podemos concatenarlos usando el operador +:

```
String etiqueta2 = String.format(
"DE:%s %s\n"
+ "Dirección: %s %d\n"
+ "Ciudad: %s\n"
+ "Contacto:%d\n",
nombre, apellido, direccion, numeroDireccion, ciudad, telefono);
```

O bien, escribirlo directamente en `System.out.printf()`;

```
System.out.printf(
"DE:%s %s\n"
+ "Dirección: %s %d\n"
+ "Ciudad: %s\n"
+ "Contacto:%d\n",
nombre, apellido, direccion, numeroDireccion, ciudad, telefono);
```

En los tres casos obtendremos el mismo resultado:

```
DE:Carmen Silva
Dirección: Los aguiluchos 43
Ciudad: Concepción
Contacto:562264895
```

Importante

Como podemos ver, todos los tipos de datos primitivos se escriben con minúscula, en cambio los de referencia a objetos, con mayúscula, porque son referencias a clases.