Variables, tipos de datos y operadores básicos

Definición informática de “dato”

Un dato es una representación simbólica de un atributo o una variable. Los usaremos para describir objetos a nivel programación. Se trata de un valor o una referencia, para luego ser procesado y de esta forma construir información dentro de una solución, o en el desarrollo de un algoritmo.

En la mayoría de los casos, los datos son simples medios para llegar a información significativa para las personas, o encontrar la solución de un problema. Por ejemplo:

Dato:
Fecha de nacimiento

Información:
- Edad
- ¿Es mayor de edad?
- ¿Puede acceder a ciertos servicios?

En esta instancia, surge un concepto clave para clasificar los lenguajes de programación. Según el manejo que hacen respecto de los tipos de variables, existiendo lenguajes tipados o no tipados. Java es un lenguaje tipado o fuertemente tipado (estas expresiones son equivalentes). Esto quiere decir que, dado el valor de una variable de un tipo concreto, no se puede usar como si fuera de otro tipo distinto, a menos que se haga una conversión.

El hecho de que Java sea un lenguaje tipado implica que tenga un tratamiento particular para sus variables, sus datos. De este punto se desprenden dos tipos de datos:

- primitivos
- referenciados.

A nivel código, las variables deben pasar por dos etapas, por un lado la declaración, y por otro la inicialización. Veamos cada una de ellas.

Declaración, básicamente, este proceso consiste en “avisar” qué tipo de variable vamos a usar, y qué nombre le vamos a dar:

```
Int a;
```

En ese caso, estamos avisando que vamos a usar una variable del tipo Int, llamada “a”.

Inicialización: Este proceso consiste en darle valor a la variable.

```
a = 120;
```

En ese caso, a la variable del tipo entero llamada “a” inicializada previamente le asignamos el valor de 120.

Tipos de datos

Datos primitivos:

Son tipos de datos con dimensión y características preestablecidas dentro del lenguaje. Ante esto, cuando se instala alguna de estas variables, el sistema le asigna una dirección y una dimensión específicas (medida en bytes/bits). Un ejemplo de este tipo de datos son los números enteros (integer).

Datos referenciados:

El principal detalle de estos datos es que no tienen dimensión específica por su condición. Se les asigna una dirección, donde el dato comienza a guardarse, y a medida que su dimensión cambia el sistema lo actualiza. Un ejemplo de esto serían las cadenas de caracteres (strings).

VEAMOS UN EJEMPLO...

Habiendo enunciado lo anterior, debemos comprender que si en una variable del tipo entero almacenamos el número 1, en el sistema este valor ocupa lo mismo que si guardamos una cifra de 6 dígitos (ej: 5.000.000).
Mientras que en una string, almacenar “Hola” o almacenar “Hola Juan Ignacio” demandará dimensiones distintas.

DATOS PRIMITIVOS

Existen distintos datos primitivos en Java, y a continuación desarrollaremos los siguientes:

- enteros,
- flotantes,
- char.

¡Veamos cada uno de ellos!

¿CUÁLES SON?

ENTEROS

Como su nombre lo indica, son espacios asignados para almacenar números enteros.

El espacio ocupado por cada uno depende pura y exclusivamente del tipo específico a utilizar:

- Byte (1 byte, 8 bits): almacena números que van desde -128 hasta 127
- Short (2 bytes): almacena números que van desde -32768 hasta 32767
- Integer (4 bytes): puede almacenar números muy grandes.

● Long (8 bytes): puede almacenar números extremadamente grandes.

FLOTANTES

Es una variable también numérica, pero que admite parte decimal, así como también números negativos.

Su mayor complejidad hará que ocupe más espacio que una variable del tipo entera, existiendo también distintos tipos de float en Java, según el grado de precisión necesaria:

● Float (4 bytes): almacena números grandes, con parte decimal.
● Double (8 bytes): almacena números muy grandes, con parte decimal.

CHAR

En Java, los caracteres no están establecidos a partir del código ASCII, sino que emplean Unicode, almacenando en estas variables cualquier carácter del código, así como las distintas secuencias de escape(\n, \r, \t, etc.).
Una variable tipo char ocupa 2 bytes.

DATOS REFERENCIADOS

Una base para las colecciones
Como ya mencionamos, un string es una variable referenciada. Almacenar variables a partir de posiciones de memoria, donde lo alojado comprenderá desde una dirección inicial, hasta una dirección final y esta longitud es variable, es una base primordial para el concepto de colecciones (collections).

INTRODUCCIÓN A LAS COLECCIONES

Pensemos en el significado coloquial de la palabra: una colección es un conjunto de elementos que cumplen con determinadas características o parámetros; los elementos almacenados tienen algo en común.

Podemos hablar de una colección de discos, pinturas, libros, o en este caso datos u objetos.

¿QUÉ ES UNA COLECCIÓN?

Las colecciones están desarrolladas a partir de interfaces, y estas implican abstracción, es decir, nos proveen de estructuras confeccionadas para un uso genérico, y quedará en manos del/la desarrollador/a la implementación de las mismas.

COLECCIONES E INTERFACES

En Java existen múltiples tipos de colecciones, cada una con prestaciones para distintos objetivos.

¿QUÉ COLECCIÓN USAR?

Para realizar un correcta selección, debemos preguntarnos lo siguiente:

- ¿Necesito un orden estático? ¿Puedo reordenar los elementos?
- ¿Hay duplicados?
- ¿Con qué frecuencia agregaremos elementos?

ARRAYS

Es una estructura de almacenamiento de datos completamente básica. Permite manipular información de manera muy veloz, ya que su forma de organización, replica el formato de un banco de memoria. Podemos instanciar arreglos de distintos tipos de variables, incluso arreglos de objetos customizados.

LIST

Se caracteriza por ser secuencia de elementos dispuesto en un cierto orden, en la que cada elemento tiene como mucho un predecesor y un sucesor. Podríamos decir que en una lista, por lo general, el orden es dato, es decir, el orden es información importante que la lista también nos está almacenando.

TIPOS DE LISTAS

- ArrayList: similar a los arreglos antes descritos, con la salvedad de que estos crecen de manera dinámica. Por su organización, se vuelve costoso la eliminación de elementos.
- LinkedList: al ser enlazada, los elementos guardan un vínculo con sus “vecinos”, haciendo muy sencilla la eliminación, pero complejo el recorrido de la misma.

SET

Esta colección está pensada para lo que matemáticamente se conoce como conjunto, lo cual agrega un detalle clave: no puede haber duplicados. A su vez, en este tipo de colección el orden deja de ser un detalle significativo, por lo que la interfaz carece de elementos relacionados al mismo.

TIPOS DE CONJUNTOS

En las próximas slides desarrollaremos los siguientes, que son los más remarcables:

- HashSet
- TreeSet
- Map

HashSet

Es la más comúnmente usada, e implica el almacenamiento de elementos a los cuales se les asocia un “hash”. Este último es básicamente un número entero, que sirve de referencia para almacenar elementos en una suerte de “tabla”.

TreeSet

Esta técnica construye un árbol (tree) con los objetos que se van agregando al conjunto, teniendo un elemento “raíz”, y nodos que se van desprendiendo como ramificaciones (de la misma manera que en un árbol genealógico). Es fácil de recorrer, pero costoso de mantener organizado.

Map

Conjunto de objetos, con el detalle de que cada uno de estos valores tiene un objeto extra asociado (manteniendo el paralelo con el diccionario, podemos decir que en este, cada palabra, conlleva una definición asociada). A los primeros se los llama “claves” o “keys”, ya que nos permiten acceder a los segundos.

Manteniendo conceptos recientemente enunciados, el conjunto de claves corresponde técnicamente a un Set, no puede haber duplicados.
De todas formas, un Map no es una Collection ya que esa interfaz le queda demasiado chica.

Podríamos decir que Collection es unidimensional, mientras que un Map es bidimensional.

Genéricos en Java

Dotan al lenguaje de mayor versatilidad, aumentando la reutilización de código, al mismo tiempo qué reducen los distintos tipos procesos de casteos.

```
public class Box {
	private T t;
	public T get() {
		return t;
	}
	public void set(T t) {
		 this.t = t;
	}
}
```

Parametrización de genéricos

Genéricos en Java Según las convenciones los nombres de los parámetros de tipo usados comúnmente son los siguientes:

- E: elemento de una colección.
- K: clave.
- N: número.
- T: tipo.
- V: valor.
- S, U, V etc: para segundos, terceros y cuartos tipos.

Operadores básicos java

Los operadores básicos en Java son:

- Operadores aritméticos: estos operadores realizan operaciones matemáticas comunes como suma, resta, multiplicación y división.
  + (suma), - (resta), * (multiplicación), / (división) y % (módulo)

- Operadores de asignación: estos operadores asignan un valor a una variable.
  = (igual)

- Operadores de comparación: estos operadores comparan dos valores y devuelven un valor booleano (verdadero o falso) dependiendo del resultado de la comparación.
  == (igual que), != (distinto que), > (mayor que), < (menor que), >= (mayor o igual que) y <= (menor o igual que).

- Operadores lógicos: estos operadores realizan operaciones lógicas como AND, OR y NOT.
  && (y lógico), || (o lógico) y ! (negación lógica).

- Operadores de incremento y decremento: estos operadores incrementan o decrementan el valor de una variable en una unidad.
  ++ (incremento), -- (decremento)

- Operadores de asignación compuesta: estos operadores realizan una operación y asignan el resultado a una variable al mismo tiempo.

Operator Type	Category	Precedence	Associativity
Unary	**postfix**	**a++, a--**	**Right to left**
	prefix	**++a, --a, +a, -a, ~, !**	**Right to left**
Arithmetic	**Multiplication**	***, /, %**	**Left to Right**
	Addition	**+, -**	**Left to Right**
Shift	**Shift**	**<<, >>, >>>**	**Left to Right**
Relational	**Comparison**	**<, >, <=, >=, instanceOf**	**Left to Right**
	equality	**==, !=**	**Left to Right**
Bitwise	**Bitwise AND**	**&**	**Left to Right**
	Bitwise exclusive OR	**^**	**Left to Right**
	Bitwise inclusive OR	**|**	**Left to Right**
Logical	**Logical AND**	**&&**	**Left to Right**
	Logical OR	**||**	**Left to Right**
Ternary	**Ternary**	**? :**	**Right to Left**
Assignment	**assignment**	**=, +=, -=, *=, /=, %=, &=, ^=, |=, <<=, >>=, >>>=**	**Right to Left**

Imagen desde https://javagoal.com/operators-in-java/

ESTRUCTURAS DE CONTROL

Las estructuras de control nos permiten modificar el flujo de un programa, ante ciertas condiciones o variaciones.
Existen los siguientes tipos:

- secuenciales,
- condicionales o selectivas,
- repetitivas o de iteración.

¿QUÉ CONTROLA UNA ESTRUCTURA DE CONTROL?

Las instrucciones de un programa se ejecutan por defecto en orden secuencial.
Esto significa que las instrucciones se ejecutan en secuencia, una después de otra, en el orden en que aparecen escritas dentro del programa.

SECUENCIALES

La estructura secuencial es el orden natural de ejecución. Las instrucciones que componen esta estructura se ejecutan en orden una a continuación de la otra.

La mayoría de las instrucciones están separadas por el carácter punto y coma (;).

Las instrucciones se suelen agrupar en bloques.
El bloque de sentencias se define por el carácter llave de apertura ({) para marcar el inicio del mismo, y el carácter llave de cierre (}) para marcar el final.
Java en sí mismo opera de modo secuencial.

CONDICIONALES

Se basan en comparaciones. Para comparaciones sencillas, usamos if, donde establecemos una condición para ser evaluada, y si la misma se cumple se efectúa alguna acción.
Opcionalmente, podemos declarar un bloque else, para que se ejecuten otras acciones, en caso de que no se cumpla la primera condición.

Bloque if-else:

Para secuencias más complejas se pueden emplear múltiples bloques como los mencionados, de manera anidada.

```
If (condición) {
        //instrucciones }
else {
        //instrucciones
}
```

ESTRUCTURA SWITCH

Equivalente a usar if anidados, es la estructura switch; en ella se establecen múltiples condiciones y acciones a efectuarse ante cada una de ellas. También es posible implementar una condición por defecto (default), que se ejecutará si no se cumple ninguna de las condiciones previas.

Bloque switch

```
switch (expresión) {
        case valor1:
                //instrucciones
                [break]
        case valor2:
```

```
            //instrucciones
            [break]
            .. ..
        default:
            //instrucciones
}
```

REPETITIVAS

Permiten reiterar las veces que el desarrollo lo amerite. Es sumamente importante evitar bucles infinitos indeseados. Asimismo, tenemos que tener en cuenta que si la condición no se cumple nunca, las instrucciones no se ejecutarán jamás.

En casos complejos, también podemos anidar.

Bloque while:

```
while (condición) {
        //instrucciones
}
```

Si necesitamos que al menos una vez el bloque de instrucciones se ejecute, dejando de lado la condición, es necesario recurrir al bloque do-while

Bloque do-while:

```
do {
        //instrucciones }
while (condición)
```

Por último encontramos el bucle for. Su estructura consta de una expresión inicial, una condición de salida, y una tasa de incremento/decremento para nuestro contador, tal como muestra el bloque anterior.

Bloque for:

```
for(expresión Inicial; condición; incremento/decremento) {
//instrucciones
}
```

CLASE SCANNER

Como en otros lenguajes, el verbo “scan” se asocia a leer valores desde el teclado. Esto nos permitirá el ingreso de datos, asociados a las variables que estudiamos con anterioridad.

También es importante saber que:
- Scanner es una clase dentro del paquete java.util,
- Permite trabajar con archivos.
- Para emplearlo debemos crear un objeto, y luego cerrar el scanner.

Uso:

Creamos el componente: Scanner sc = new Scanner(System.in);
Empleamos para tomar un Double: Double numero1 = sc.nextDouble();
Empleamos para tomar un Char: char operacion = sc.next().charAt(0);
Cerramos el scanner: sc.close();

¡VAMOS A PRACTICAR LO VISTO!

CALCULADORA

A continuación haremos uso de la clase Scanner, para leer distintos dos números y un carácter. Emplear el carácter para establecer el tipo de operación matemática a realizar entre ambos números. Realizar dos versiones del ejercicio, uno a partir de “if-else” otro a partir de “switch”. Duración: 20 minutos

CONCLUSIONES

Los puntos importantes, que nos servirán para programas más complejos, son:

- Importación de librerías.
- Manipulación de variables.
- Manejo de las entradas y salidas de un programa.
- Instanciamiento de una clase.

¡A PRACTICAR!

A partir de un bucle for{}, el ejercicio de la clase anterior (listar los días de la semana) y un arreglo, imprimir nuevamente los días de la semana.

Lectura requerida:

Lectura ampliatoria: