Notas Java - 2. Operadores y Control de Flujo

Cristo Alvarado

9 de julio de 2025

Resumen

En esta lección se abordarán los conceptos de casting (conversión de tipos) entre datos primitivos, el tipo de dato String y sus métodos más comunes, la importación de clases con énfasis en la clase Scanner para la entrada de datos, los diferentes métodos de entrada para tipos primitivos, las operaciones aritméticas básicas, y las sentencias condicionales (if, else, switch). Además, se introduce la clase Math de Java para operaciones matemáticas avanzadas y las estructuras de repetición (bucles while y for).

Índice

Lista de Códigos

§1 Información

§1.1 Casting

Definición 1.1 (**Casting**)
El **Casting** o **Casting de Tipos** es *una funcionalidad del lenguaje de programación que nos permite convertir un tipo de dato primitivo en otro.* En Java, tenemos dos tipos de casting:

- **Widening Casting** (**Casting a más**): *Permite convertir un tipo de dato pequeño a uno más grande.* Este tipo de casting es **automático**.

  La conversión de menos a más va de izquierda a derecha como muestra el diagrama:

  ```
  byte ->short ->char ->int ->long ->float ->double
  ```

- **Narrowing Casting** (**Casting a menos**), *este permite convertir tipos de datos grandes a más pequeños.* Este tipo de casting es **manual**.

  La conversión de más a menos va de izquierda a derecha como muestra el diagrama:

  ```
  double ->float ->long ->int ->char ->short ->byte
  ```

Ejemplo 1.1 (**Casting a Más**)
Este tipo de casting es hecho automáticamente cuando se pasa de untipo de dato pequeño a uno más grande:

```
public class Main {
  public static void main(String[] args) {
    int myInt = 9;
    double myDouble = myInt; // Casting de int a double

    System.out.println(myInt);       // Salida 9
    System.out.println(myDouble);    // Salida 9.0
  }
}
```

Código 1: Casting a más.

Ejemplo 1.2 (**Casting a Menos**)
Este tipo de casting es hecho colocando `()` colocando en estos paréntesis el tipo al que queremos pasar:

```
public class Main {
  public static void main(String[] args) {
    double myDouble = 9.78d;
    int myInt = (int) myDouble; // Casting de double a int

    System.out.println(myDouble);       // Salida 9.78d
    System.out.println(myInt);   // Salida 9
  }
}
```

Código 2: Casting a menos.

Ejemplo 1.3 (**Uso Casting**)
Si en un juego queremos hacer porcentajes usamos casting como se muestra:

```
// Establecer el score maximo a 500
int maxScore = 500;

// Score del usuario
int userScore = 423;

/*
Calculo del porcentaje de score del usuario con referencia al
   score maximo
Convertimos score de usuario a flotante para asegurarnos que
   la division es correcta
*/
float percentage = (float) userScore / maxScore * 100.0f;

System.out.println("User's percentage is " + percentage);
```

Código 3: Ejemplo Casting en Score.

§1.2 Tipo de Dato String

Una variable de tipo `String` contiene una colección de caracteres encerrados por comillas dobles `"`:

```
String palabra = "Una palabra";
```

Un `String` en java e sun objeto que contiene métodos que pueden hacer ciertas opearciones con este tipo de dato (el cuál no es primitivo).

Ejemplo 1.4 (Longitud de un String)
Podemos encontrar la longitud de una cadena usando el método `length()`:

```
String texto = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
System.out.println("La longitud de lal texto es: " + texto.
   length());
```

Código 4: Longitud de un `String`.

Este objeto tiene varios métodos, algunos de elloos son los descritos a continuación:

- `toUpperCase()`. Este método pone todas las letras en mayúsculas.
- `toLowerCase()`. Este método pone todas las letras en minúsculas.
- `indexOf()`. Este método retorna el índice o posición de un texto especificado en el String original. En este caso el código:

```
String txt = "Por favor, localiza donde 'localiza' ocurre!";
System.out.println(txt.indexOf("localiza")); // Imprime 11
```

Código 5: Localización de texto en un `String`.

Podemos hacer una cosa con Strings que se llama **concatenación**. Esta operación nos permite unir dos variables `String` en una sola variable. Podemos hacer esta operación usando el operador `+` o usando el método `concat()`:

```
String primerNombre = "Juan ";
String segundoNombre = "Perez";
System.out.println(primerNombre + segundoNombre);
System.out.println(primerNombre.concat(segundoNombre));
```

Código 6: Suma y Concatenación de Strings.

Observación 1.1
Si sumamos un número y una cadena, en vez de que se sumen se concatenan:

```
String x = "10";
int y = 20;
```

```
String z = x + y;  // z sera 1020 (un String)
System.out.println(z); // imprime 1020
```

Código 7: Suma de Número y `String`.

§1.3 Importar Clases

Parte muy importante de todo programa es la interacción con el usuario, la gran pregunta es: ¿cómo le damos al programa nuestra información? Una forma de hacerlo es con el paquete `java.util`:

Definición 1.2 (**Paquete**)
Un **paquete** en Java es un *mecanismo que nos permite agrupar clases, interfaces y subpaquetes de manera lógica, facilitando la organización y modularidad del código.*

Además, es un *contenedor de clases que permite agrupar las distintas partes de un programa y que por lo general tiene una funcionalidad y elementos comunes, definiendo la ubicación de dichas clases en un directorio de estructura jerárquica.*

Al importar paquetes importamos clases que nos son de útilidad en nuestro sistema. En particular el paquete `java.util` incluye el framework de colecciones, clases de colecciones heredadas (legacy), el modelo de eventos, utilidades de fecha y hora, internacionalización, y otras clases de utilidad (como un tokenizador de cadenas, un generador de números aleatorios y un arreglo de bits). En esencia, es *una extensión de lo que podemos hacer con Java.*

Una de sus clases que nos es muy útil es `java.util.Scanner`.

Para poder importar clases en nuestro programa, usamos la siguiente sintaxis:

```
//import nombre_paquete.nombre_clase_importar;
```

Código 8: Importar Clases en Java.

donde `nombre_paquete` es el nombre del paquete del que queremos importar la (o las clases) y `nombre_clase_importar` es el nombre de la clase a importar. En caso de que queramos importar todas las clases de un paquete, colocamos `*` en vez de `nombre_clase_importar`.

Ejemplo 1.5 (**Importar Clases**)
Para importar la clase `Scanner` del paquete `java.util`, colocamos la siguiente línea en nuestro código:

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

Código 9: Importar Clase `Scanner`

en caso de que queramos importar todas las clases del paquete `java.util`, colocamos lo siguiente:

```
import java.util.*;
```

Código 10: Importar todas las Clases del Paquete `java.util`

Observación 1.2 (**Importación de Clases**)
A veces llega a suceder que olvidamos el nombre específico de una clase que queremos usar, por lo que en casos donde el paquete no tenga demasiadas clases, podemos importar todas las clases

para usarlas en nuestro programa.

Ejemplo 1.6 (**Uso Clase Scanner**)
El siguiente código nos permite ingresar nuestro nombre para luego, que el sistema lo imprima al momento de ejecutar.

```
import java.util.Scanner;  // Importar la clase Scanner

public class Main {
  public static void main(String[] args) {
    Scanner myObj = new Scanner(System.in);  // Crear un
       objeto Scanner
    System.out.println("Ingrese nombre: ");

    String nombre_usuario = myObj.nextLine();  // Leer
       entrada de usuario
    System.out.println("Su nombre es: " + nombre_usuario);
       // Imprimir nombre de usuario
  }
}
```

Código 11: Uso Clase `Scanner`.

¿Qué hace el código? Primero, crea una instancia del objeto `Scanner`. Esto se hace en esta línea:

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

Notemos que el objeto `Scanner` usa el flujo de entrada de `System.in`, justamente eso es lo que nos permite leer la terminal pues justamente es la entrada de datos del programa. Al presionar la tecla `Enter`, el sistema registra la linea leída en el String:

```
String nombre_usuario = myObj.nextLine();
```

Guardamos lo que sea que hayamos escrito en la variable `nombre_usuario` para luego imprimirlo.

Observación 1.3
Con la clase `Scanner` es posible leer entradas de información de diversos tipos, no solamente del tipo `String`. Para poder hacerlo, ya que si intentamos hacerlo directamente en la entrada nos marcará error.

Para leer tipos de datos diferentes de `String`, usamos **input types**.

§1.4 Input Types

Definición 1.3 (**Input Type**)
Un **Input Type** es un *tipo de entrada que puede recibie un programa Java en la terminal.*

En ejemplos anteriores usamos `nextLine()`, el cuál es un método del objeto Scanner, pero podemos leer otros tipos de datos, la siguiente tabla nos dice qué métodos usar para leer cierto tipo de dato en el programa:

Método	Descripción
`nextBoolean()`	Lee un valor `booleano` del usuario
`nextByte()`	Lee un valor `byte` del usuario
`nextDouble()`	Lee un valor `double` del usuario
`nextFloat()`	Lee un valor `float` del usuario
`nextInt()`	Lee un valor `int` del usuario
`nextLine()`	Lee un valor `String` del usuario
`nextLong()`	Lee un valor `long` del usuario
`nextShort()`	Lee un valor `short` del usuario

Ejemplo 1.7 (**Lectura de Diferentes Tipos de Datos**)
En el siguiente ejemplo se leen diferentes tipos de datos con `Scanner`:

```
import java.util.Scanner;

class Main {
  public static void main(String[] args) {
    Scanner myObj = new Scanner(System.in);

    System.out.println("Enter name, age and salary:");

    // String input
    String name = myObj.nextLine();

    // Numerical input
    int age = myObj.nextInt();
    double salary = myObj.nextDouble();

    // Output input by user
    System.out.println("Name: " + name);
    System.out.println("Age: " + age);
    System.out.println("Salary: " + salary);
  }
}
```

Código 12: Lectura diferentes tipos de datos con `Scanner`.

§2 Operaciones Elementales

En Java, para manipular variables hay operaciones que son imprescindibles conocer para poder hacer el código más eficiente y manejable posible.

§2.1 Operaciones Aritméticos entre Datos Primitivos

En la clase pasada se vió que los tipos de datos primitivos son `byte`, `short`, `int`, `long`, `float`, `double`, `boolean` y `char`. En particular, veremos las operaciones básicas entre estos datos (salvo el tipo de dato `boolean`):

Operador	Nombre	Descripción	Ejemplo
+	Suma	Agrega dos valores	`x + y`
−	Resta	Sustrae un valor de otro	`x - y`
*	Multiplicación	Multiplica dos valores	`x * y`
/	División	Divide un valor por otro	`x / y`
%	Módulo	Devuelve el residuo de una división	`x % y`
++	Incremento	Aumenta el valor de una variable en 1	`++x`
−−	Decremento	Disminuye el valor de una variable en 1	`--x`

La suma, resta, multiplicación y división nos permite hacer estas operaciones entre los tipos de datos primitivos que hemos hablado anteriormente (salvo `boolean`).

Observación 2.1
Al momento de efectuar la operación, el tipo de dato más grande es al que se convierte la información (recordemos que en este caso no necesitamos hacer casting). Cuando pasamos de un dato más grande a uno más pequeño, el número resultante de la operación se redondea.

Estas operaciones anteriores son llamadas **binarias**. en cambio, las operaciones `++` y `--` son **unarias**. Estas, efectúan lo siguiente: toman una variable int como entrada e incrementan en uno su valor. Así que `x++` es equivalente a `x = x + 1`.

La operación `%` es la **operación módulo**. Esta básicamente determina el residuo de la división de dos números enteros.

§2.2 Matemáticas

Java proporciona la clase `Math` para operaciones matemáticas avanzadas. Contiene métodos para exponentes, logaritmos, raíces y trigonometría.

Método	Descripción
`Math.max(x,y)`	Devuelve el mayor entre `x` e `y`
`Math.min(x,y)`	Devuelve el menor entre `x` e `y`
`Math.sqrt(x)`	Calcula la raíz cuadrada de `x`
`Math.pow(x,y)`	Eleva `x` a la potencia `y`
`Math.abs(x)`	Devuelve el valor absoluto de `x`
`Math.random()`	Genera un número aleatorio en [0.0, 1.0)

Ejemplo 2.1 (**Uso de Math**)

```
double raiz = Math.sqrt(25);      // 5.0
double potencia = Math.pow(2, 3); // 8.0
double maximo = Math.max(10, 20); // 20.0
double aleatorio = Math.random(); // Ej: 0.548392
```

Código 13: Operaciones con la clase Math.

§3 If y Else

Definición 3.1 (**Sentencia Condicional**)
Una **sentencia condicional** es *una construcción de programación que permite ejecutar diferentes bloques de código dependiendo de si una condición específica es verdadera o falsa.*

Por ejemplo, una sentencia condicional evalúa si una variable `a` es mayor que otra `b`. Java posee las siguientes sentencias condicionales:

- `if`: Este se usa para ejecutar un bloque de código si una condición es `true`.
- `else`: Este se usa para ejecutar un bloque de código si la condición especificada en el `if` anterior es `false`.
- `switch`: Se usa para ejecutar múltiples bloques de código alternativos dependiendo de la entrada.

§3.1 `if`

La sintaxis de la sentencia `if` es la siguiente:

```
if (condicion) {
  // bloque de codigo a ejecutar si la condicion es true
}
```

Código 14: Sintaxis de la sentencia `if`.

Ejemplo 3.1 (**Comparación de Números**)
La sentencia `if` puede ser usada como en el siguiente ejemplo:

```
if (20 > 18) {
  System.out.println("20 es mayor a 18");
}
```

Código 15: Uso de la sentencia `if`.

§3.2 `else`

La sintaxis de la sentencia `else` es la siguiente:

```
if (condicion) {

}
else {
    // bloque de codigo a ejecutar si la condicion es false
}
```

Código 16: Sintaxis de la sentencia `else`.

Ejemplo 3.2 (**Comparación de Números**)
La sentencia `else` puede ser usada como en el siguiente ejemplo:

```
int time = 20;
```

```
if (time < 18) {
  System.out.println("Buen dia.");
} else {
  System.out.println("Buenas noches.");
}
// Imprime "Buenas noches".
```

Código 17: Uso de la sentencia `else`.

§3.3 `switch`

En vez de escribir muchos `if` y `else`, se puede usar la sentencia `switch`.

Lo que hace `switch` es seleccionar uno de los muchos bloques de código a ser ejecutados.

```
switch(expression) {
  case x:
    // bloque de codigo
    break;
  case y:
    // bloque de codigo
    break;
  default:
    // bloque de codigo
}
```

Código 18: Sintaxis de la sentencia `switch`.

La sentencia `switch` funciona de la siguiente forma:

- La expresión del `switch` se evalúa una sola vez.
- El valor de la expresión se compara con los valores de cada caso.
- Si hay coincidencia, se ejecuta el bloque de código asociado.
- Las palabras clave `break` y `default` son opcionales.

Ejemplo 3.3
El siguiente ejemplo usa `switch` para imprimir el día de la semana (numerados del 1 al 7) en función del número seleccionado:

```
int day = 4;
switch (day) {
  case 1:
    System.out.println("Lunes");
    break;
  case 2:
    System.out.println("Martes");
    break;
  case 3:
    System.out.println("Miercoles");
    break;
```

```
      case 4:
        System.out.println("Jueves");
        break;
      case 5:
        System.out.println("Viernes");
        break;
      case 6:
        System.out.println("Sabado");
        break;
      case 7:
        System.out.println("Domingo");
        break;
    }
    // Imprime "Jueves" (day 4)
```

Código 19: Uso de la sentencia `switch`.

§4 Bucles

Definición 4.1 (**Bucle**)
Un **bucle** es *una estructura de control que repite un bloque de código mientras se cumpla una condición específica.*

§4.1 `while`

Ejecuta un bloque mientras la condición sea verdadera.

```
while (condicion) {
  // codigo a ejecutar
}
```

Código 20: Sintaxis de `while`.

Ejemplo 4.1 (**Contador con while**)

```
int i = 0;
while (i < 5) {
  System.out.println(i);
  i++;
}
// Imprime 0,1,2,3,4
```

Código 21: Uso de `while`.

§4.2 `for`

Ideal para iteraciones con contador controlado.

```
for (inicializacion; condicion; actualizacion) {
```

```
    // codigo a ejecutar
}
```

Código 22: Sintaxis de `for`.

Ejemplo 4.2 (Contador con for)

```
for (int i = 0; i < 5; i++) {
  System.out.println(i);
}
// Imprime 0,1,2,3,4
```

Código 23: Uso de `for`.