Polito - Curso de programación 2025

Documentación y ejercicios <3

UB Polito Arcuschin

Para que reine en el pueblo el amor y la igualdad ✌

Índice

1. Bienvenidxs :)

En este lugar iremos cargando las clases, ejercicios y otros recursos que servirán como la documentación del curso de programación de la UB Polito Arcushin del año 2025.

La idea es que puedan recurrir a estas páginas para refrescar conceptos entre clases y tener ejercicios a mano para practicar.

Como ya hemos dicho muchas veces: todos es perfectible, así que no duden en buscar errores 🕵️ o cosas que no quedan claras 🤔 y avisarnos para que podamos revisar los contenidos.

¡Esperamos que aprendan cosas nuevas, se les ocurran ideas extravagantes y sobre todo que disfruten la experiencia de pensar con otrxs!

Abel, Mariana, Cami y Manu 🫰

2. Clase 00: Historia y fundamentos

2.1 Clase 0: Historia de la programación y fundamentos.

3. Clase 01: Programa y Operadores

3.1 ¿Qué es un programa? | Fases de desarrollo | estructura y sintaxis

3.1.1 Breve repaso

En la clase pasada habíamos llegado a una definición para la acción de programar. Vimos que programar no era exclusivo de la informática, sino una forma de pensar (lógica) para elaborar una serie de instrucciones (algoritmo) que serán ejecutadas por una máquina o sistema.

¿Qué es programar?

Programar es una forma de pensar para elaborar una serie de instrucciones que serán ejecutadas por un sistema.

¿Qué es un algoritmo?

- Un algoritmo es una creación humana.
- Se lo puede definir como una serie de instrucciones ordenadas o una lista ordenada de pasos para lograr un objetivo.
- Ejemplos clásicos: Una receta de cocina 📖🥣, un manual para armar un mueble 📄, tomarse el colectivo 🚻

Vimos también qué era un lenguaje de programación: una forma de escribir esas instrucciones. Y los categorizamos por algunas de sus posibles características: por su forma de ejecución, por su nivel de rigurosidad en su sintáxis y semántica, por sus paradigmas y por su nivel de abstracción.

Vimos también que los lenguajes de programación de "alto nivel" intentan arrimarse al lenguaje humano y, al igual que ese, tienen una serie de reglas de sintaxis y semántica con

las que hay que cumplir para que, al fin de cuentas, la máquina pueda ejecutar nuestras instrucciones de la manera en que las pensamos.

Los lenguajes de programación

- Un lenguaje de programación es una forma de escribir instrucciones.
- Se los suele agrupar por sus características distintivas.
- Al igual que el lenguaje humano, cada lenguaje tiene reglas **sintácticas** y **semánticas**.

A partir de estas definiciones, ya podemos adentrarnos en dilucidar... ¿qué es entonces un programa?

3.1.2 Programa

La palabra programa puede hacer referencia a muchas cosas no tan disímiles como aparentan: Un programa de una materia de la facu, un programa de televisión, un programa político (quienes tengan cercanía a espacios de discusión política, habrán notado los constantes reclamos de la existencia de dicha cosa), etc...

Como podrán observar, todas esas acepciones de la palabra "programa" son bastante similares: Es un conjunto ordenado de tópicos que se deben seguir en el orden establecido y de una forma más o menos constante.

Pues bien, entonces un programa informático no es más que una seguidilla de sentencias que se ejecutan desde el inicio hasta el final (ya veremos un poco más esto), realizando así las acciones que definimos dentro de ese programa.

¿Qué es un programa?

Un programa informático es un conjunto de sentencias que se ejecutan desde el inicio hasta el final realizando así las acciones definidas.

3.1.3 Fases de desarrollo

Esto lo veremos en detalle más adelante cuando veamos ejercicios de *problemas*, pero vale la pena mencionarlo aquí para tener una idea general de las fases que tiene el desarrollo de un programa.

¿Por qué insistimos tanto en esto de que programar no es exclusivamente escribir código en un lenguaje elegido? La programación empieza mucho antes de sentarse a escribir: **se arranca siempre pensando un problema**, antes incluso de pensar en cómo resolverlo. Recuerden que dijimos en la primera clase que programar es pensar en soluciones creativas para problemas complejos.

Es decir, primero vamos a intentar **descomponer el problema en pasos lógicos** y ordenados, luego vamos a **imaginar y diseñar soluciones posibles** y, finalmente como último paso, vamos a **traducir esas soluciones a código** en el lenguaje que hayamos elegido.

Fases de desarrollo

1. Descomponer el problema en pasos lógicos y ordenados
2. Imaginar y diseñar las soluciones posibles
3. Traducir las soluciones elegidas a código

3.1.4 Estructura de un programa

Vamos a ver cómo se ve un programita básico ("Hola Mundo"[1]) para intentar descomponer su estructura.

```java
package holamundo;

/**
 *
 * @author nombre
 */
public class HolaMundo {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO: el código a ejecutar viene acá abajo.
        System.out.print("¡Hola mundo!");
    }
}
```

Descomponemos el programa:

- Vemos que hay **líneas** de texto
- Las líneas parecieran estar agrupadas en "bloques"
- Hay llaves que "engloban" líneas
- Hay paréntesis que "abrazan" palabras
- Hay unas líneas "especiales" que empiezan con símbolos: barras y asteriscos (`/**` `//`)
- La línea que imprime el texto (`System.out.print()`), "ya viene" con el lenguaje.

Sentencias y bloques

Las líneas de nuestro programa son… ¡las sentencias! Una sentencia es una representación de la acción que nuestro programa debe ejecutar.

A la agrupación de líneas de código que conforman una sola sentencia, las llamaremos bloques.

Llaves y paréntesis

Las llaves son las que construyen la estructura del código, delimitando los bloques. Indican que todo lo que está entre la llave de apertura (`{`) y la de clausura (`}`) "pertence" a la línea que antecede inmediatamente a la apertura.

En nuestro ejemplo tenemos dos "conjuntos" de llaves:

```
public class HolaMundo {
    // Hay cosas acá adentro.
    // Y todas pertenecen a esta clase (class) pública (public) llamada HolaMundo.
}
```

```
public static void main(String[] args) {
    // Hay cosas acá adentro.
    // Y todas pertenecen esta función pública y estática (static) llamada main.
}
```

Los paréntesis también cumplen la función de agrupar, pero a diferencia de las llaves no conforman bloques, sino algo similar a un agrupamiento de palabras o incluso una palabra sola.

Veremos esto más en detalle cuando veamos "Funciones" (que se suele representar: `f(x)`) pero por ahora recordemos que lo que está dentro de un paréntesis "pertenece" a la *palabra* o expresión que antecede inmediatamente el de apertura `(`.

```
1  /**
2   *
3   * @author nombre
4   */
5  public class HolaMundo {
6
7      /**
8       * @param args the command line arguments
9       */
10     public static void main(String[] args) {
11         // TODO: el código a ejecutar viene acá abajo.
12         System.out.print("¡Hola mundo!");
13     }
14 }
```

Indentación

El espacio en blanco que vemos antes de las líneas de código es la indentación o el indentado. En la sintaxis de los lenguajes humanos esto se conoce como sangría. En el

caso de Java la indentación no tiene importancia sintáctica ni semántica, peeeero es importante respetarla porque facilita visualmente la comprensión y la identificación de los bloques.

Comentarios

Un comentario es una línea de texto que no es ejecutada, es una nota que agrega quien escribe el programa para hacer aclaraciones sobre ese código. Pueden ser notas sobre la funcionalidad (qué hace), o para facilitar la comprensión de la solución implementada (cómo lo hace).

Es una buena práctica utilizar los comentarios para "documentar" nuestro código y así facilitar que otras personas puedan modificarlo (o incluso para nuestro *yo del futuro* :)).

Ciclo del programa: Inicio, proceso, final.

Veremos esto en clase cuando "corramos" 🏃 nuestro programa, pero dejamos aquí una definición muy esquemática de un ciclo.

Analicemos el ciclo:

- **Inicio**: Punto de entrada (o *entrypoint* en inglés)
- **Proceso**: Una vez que se cargó en memoria, el código es ejecutado y el sistema operativo gestionará los recursos del entorno (asignando recursos, comunicando con dispositivos de entrada y salida, archivos, etc…).
- **Final**: Código de salida.

3.1.5 Tipos de datos y valores

Los tipos de datos **clasifican datos según sus características específicas**. Los valores son los datos en sí mismos (y pertenecerán a un tipo u otro). Entenderemos un poco más el concepto de valor y dato cuando veamos Variables.

Recuerdan que en la primera clase vimos que había lenguajes fuertemente tipados o débilmente tipados, esta caractéristica de los lenguajes hará que en nuestro programa debamos explicitar (o no) el tipo de dato que vamos a manipular. Veamos entonces ahora los tipos de datos llamados primitivos o elementales que nos ofrece Java.

Tipos primitivos

Los "tipos primitivos"[2], a veces llamados también Tipos Elementales, son los tipos de datos originales de cada lenguaje. Es decir que cada lenguaje ya nos proporciona esos tipos y así también los define:

1. Qué **tipo de dato** se puede representar (números -enteros o decimales-, caractéres -letras y símbolos-, etc…)
2. Sus **rangos de valores posibles**
3. Y por lo anterior entonces también se define **el espacio que ocuparán en memoria** los datos pertenecientes a cada tipo

Recordemos que las máquinas solamente "entienden" datos que llamamos **binarios**, esto quiere decir que sólo pueden ejecutar unos y ceros. Por lo tanto, para las máquinas todo será una combinación de esos dos valores a los que definimos como **bits**: Un bit puede tener un valor de 1 o de 0.

Esto lo tenemos que tener claro para entender como se definen los tipos de datos primitivos.

Aprovechemos el primer tipo de datos para profundizar en esto:

BYTE (BYTE -ANGLICISMO- U OCTETO)

Es un tipo de dato que representa una estructura de **8 bits**. O sea, es el conjunto de 8 bits:

```
byte = 00000000
```

Este tipo de dato puede almacenar **valores numéricos enteros** que van desde -128 hasta 127 (ambos inclusive).

Ustedes se preguntarán ¿cómo es que eso sucede? bueno, la maquina construye los números alternando `1` y `0` en esa sucesión de bits.

Por ejemplo, lo que escribimos más arriba: `byte = 00000000` para la máquina representa `0`. O sea, nosotros escribimos un `0` en el teclado y la máquina lo "traducirá" a ese conjunto de (8) bits. Si nosotros escribimos un 5, la máquina lo traducirá como `00000101`. Como

pueden ver siguen siendo 8 bits, lo que cambian son las posiciones que tienen un `1` o un `0`.

Todo esto es importante porque les da la dimensión de lo que ocupan en memoria los datos.

Cuando nosotros creamos un dato, la computadora reserva en memoria el tamaño mayor que puede llegar a alcanzar ese dato. En el ejemplo que estamos usando, un byte, la máquina va a reservar 8 bits (8 lugares para poner unos y ceros).

Entonces es importante que **antes de crear datos pensemos qué valores vamos a querer manejar**.

SHORT (ENTERO CORTO)

Representa un tipo de dato de **16 bits**. O sea, que usará el doble de memoria que nuestro tipo anterior: `00000000 00000000`.

Puede almacenar **valores numéricos enteros desde -32.768 hasta 32.767**. Por supuesto también lo hace combinando `1`s y `0`s en esas 16 posiciones.

INT (ENTERO)

Este tipo de dato puede almacenar **valores numéricos enteros** de 32 bits, o sea **4 veces nuestro primer dato** `00000000` Los valores van desde -2^{31} hasta $2^{31}-1$. Como pueden ver, el tamaño del número que permite manejar en mucho más grande.

Podemos trabajar con números un poco superiores a los 2 mill millones -2.147.483.648 y 2.147.483.647.

LONG (ENTERO LARGO)

Es un tipo de datos de **64 bits** o sea **8 veces nuestro primer tipo de dato**: `00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000` Y puede almacenar **números enteros que van desde -2^{63} hasta $2^{63}-1$**. Como se puede imaginar es extremadamente grande: 9.223372×10^{18}

FLOAT (COMA FLOTANTE O 'REAL')

Es un tipo de dato de **32 bits**, como el int, pero permite manejar números com coma flotante. Por ejemplo `16,4` o `1879223,45`. Estos números son los que usualmente se conocen como "decimales".

Este tipo de dato tiene una precisión simple, esto es algo que veremos más adelante, pero por el momento alcanza con saber que no debe ser utilizado si vamos a necesitar números con alta precisión (es decir, lo podemos usar para aplicaciones gráficas o videojuegos, pero no para información científica o financiera).

DOUBLE (COMA FLOTANTE DOBLE O 'REAL LARGO')

En este caso es un tipo de dato de **64 bits**, como el long, pero permite manejar números con coma flotante, como el float.

Las diferencias entre el float y double son:

- El rango numérico (es el doble)
- La precisión (es, también, el doble) Al ser un número decimal su "exactitud" depende de la cantidad de dígitos que puede manejar después de la coma. Por eso, el tipo de dato "double" tiene el doble de precisión que el "float".

BOOLEAN (BOOLEANO -ANGLICISMO- O LÓGICO)

Este es un tipo de dato de **un bit** que representa verdadero (en inglés "true") o falso (en inglés "false") (que también podría ser sí o no, 1 o 0, lleno o vacío, etc…).

Se los llama así porque está basado en el álgebra "booleana", del filósofo y matemático inglés George Boole.

CHAR (CARÁCTER O SÍMBOLO)

Es un tipo de dato que representa un carácter Unicode de **16 bits**. Se utiliza mucho el tipo de dato char para manejar un símbolo o una letra (¡que es en última instancia un símbolo!) y no un fragmento de texto.

STRING (CADENA DE TEXTO)

Este tipo de dato lo ponemos como primitivo ¡pero no lo es! Lo incluimos aquí porque es tan utilizado como los primitivos.

En principio hay que observar que empieza con una mayúscula. En casi todos los lenguajes de programación orientada a objetos esto tiene un significado importante. Por ahora podemos decir que cuando tenemos que trabajar con textos (frases o palabras), usamos este tipo de dato.

Tipos primitivos - Tabla

Tipos primitivos de JAVA

Tipo	Traducción	Memoria utilizada	Rango de valores	Breve descripción
byte	Byte u Octeto (raro)	1 byte (8 bits)	de -128 a 127	Representación de un número entero positivo o negativo.
short	Entero corto	2 byte (16 bits)	de -32.768 a 32.767	Representación de un entero positivo o negativo con rango corto.
int	Entero	4 byte (32 bits)	de -2^{31} a $2^{31}-1$	Representación de un entero estándar.
long	Entero largo	8 byte (64 bits)	de -2^{63} a $2^{63}-1$	Representación de un entero de rango ampliado.
float	Coma flotante o Real	4 byte (32 bits)	de -10^{32} a 10^{32}	Representación de un número real estándar. La precisión del dato contenido varía en función del tamaño del número: la precisión se amplia con números más próximos a 0 y disminuye cuanto más se aleja del mismo.
double	Real largo	8 byte (64 bits)	de -10^{300} a 10^{300}	Representación de un número real con mucha precisión.
char	Carácter	2 byte (16 bits)	de '\u0000' a '\uffff'	Carácter o símbolo. Recordar que para utilizar una "cadena de texto" se debe usar la clase String.
boolean	Booleano o Lógico	1 bit	true - false	Representa una 'posición' lógica con dos valores posibles: verdadero (true) o falso (false)

3.1.6 Operadores

Los operadores son símbolos o palabras que utilizamos para manipular y combinar expresiones. Son elemento que nos permiten realizar, como su nombre lo indica, operaciones como:

- Comparaciones
- Procedimientos lógicos
- Cálculos aritméticos
- Asignaciones

Operadores relacionales

Utilizamos los operadores relacionales cuando necesitamos comparar dos elementos o valores.

Dependiendo el operador que usemos tendremos que tener en cuenta, o no, la posición de los elementos con relación al operador.

OPERADOR	DESCRIPCIÓN	EJEMPLO
`==`	Es igual	a == b
`!=`	Es distinto	a != b
`<`	Menor que	a < b
`>`	Mayor que	a > b
`<=`	Menor o igual que	a <= b
`>=`	Mayor o igual que	a >= b

Es decir, decir que `a == b` es lo mismo que decir que `b == a`, sin embargo, no sería lo mismo intercambiar los operandos cuando hacemos una comparación o relación mayor/menor: `a < b` no es lo mismo que `b < a`.

Operadores lógicos

A diferencia de los relacionales que **comparan** elementos o valores, los operadores lógicos **combinan** operaciones *booleanas*.

Son muy utilizados en estructuras condicionadles y *loops*, que veremos más adelante, para combinar condiciones y obtener luego un resultado.

OPERADOR	DESCRIPCIÓN	EJEMPLO	EJEMPLO COTIDIANO
`&&`	Y (de adición)	si [condición] Y [otra condición]	Agarro la campera si hace frío Y llueve
`||`	O (de exclusión)	si [condición] U [otra condición]	Agarro la campera si hace frío O llueve
`!`	NO (de negación)	si NO [condición]	Dejo la campera si NO hace frío

Cuando usamos los operadores `&&` si nuestra primera condición devuelve *falso* o usando el comparador `||` sin nuestra primera condición devuelve *verdadero* , en ambos casos la segunda condición no se evaluará.

Los operadores de negación simplemente invierten un valor lógico.

Operadores aritméticos

Los operadores aritméticos se utilizan para realizar operaciones matemáticas básicas.

OPERADOR	DESCRIPCIÓN
`+`	Adición
`-`	Resta
`*`	Multiplicación
`/`	División
`%`	Módulo

Operadores de asignación

Los operadores de asignación se utilizan para asignar valores a las variables.

OPERADOR	DESCRIPCIÓN
`=`	Asignación simple
`+=`	Añadir y asignar
`-=`	`Restar y asignar
`*=`	Multiplica y asigna
`/=`	Divide y asigna
`%=`	Módulo y asignar

1. "Hola, mundo" en informática es un programa que muestra el texto «Hola, mundo» en un dispositivo de visualización, en la mayoría de los casos la pantalla de un monitor. Este programa suele ser usado como introducción al estudio de un lenguaje de programación, siendo un primer ejercicio típico considerado fundamental desde el punto de vista didáctico. ↩
2. En todos los elementos de Java vamos a ver que estos reciben nombres en inglés. Junto a cada tipo vamos a poner su traducción en castellano para entenderlos, pero cuando los utilicemos siempre tendrá que ser con su nombre inglés. ↩

3.2 Ejercicios para la clase 01

En esta clase vimos cómo crear nuestro primer programa (HolaMundo) y como hacer para que nos imprima algo en la consola del IDE.

Primeros pasos para estos ejercicios:

1. Abrir la IDE (Netbeans)
2. Crear un nuevo programa (o abrir uno existente)
    a. Si estás creando uno nuevo, acordate de seleccionar la opción "Java with Ant".

3.2.1 Ejercicio 1 👋🏽

Usando las herramientas del sistema (`System.out.print`), impriman un texto en la consola de Netbeans.

Y ahora jueguen usando los operadores. Por ejemplo, podemos sumar dos cadenas de texto.

```
System.out.print("Hola," + " " + "¿qué tal?");
```

Podemos también sumar o "juntar" caracteres:

```
// No me acuerdo ni uno, pero https://www.unicode.org/charts/script/chart_Symbol-Other.html
System.out.println("H" + '\u2665' + "l" + "u");

System.out.println("Venceremos " + '\u270C');

System.out.println("Extraño la " + '\u00F1');
```

3.2.2 Ejercicio 2

Utilicen el otro método para imprimir mensajes, para que agregue un salto de línea cuando termina el texto (`System.out.println`).

Esto nos va a venir bien para que no nos quede todos los mensajes ahí amontonados en una sóla línea :)

¿Qué otro operador se puede usar con cadenas de texto? ¿Podremos dividir, restar o multiplicar?

Hagan la prueba, cuando Java no sepa qué hacer con el operador y la cadena, se los va a decir en forma de error cuando compilen.

Sugerencias:

- Prueben usando operadores relacionales.

3.2.3 Ejercicio 3

Usando este último método, prueben imprimir mensajes de texto y prueben también si pueden hacer cuentas y comparar valores.

Recuerden que si queremos imprimir texto usamos las comillas envolviendo el texto, pero los números y operadores deben ir por fuera de las comillas.

Ponemos acá ejemplos para que se guíen, pero ¡no los copien tal cual! Inventen cualquiera cosa, sin miedo, que la mejor manera de aprender es insistir en el error, tratar de entenderlo y después arreglarlo para llegar al resultado esperado :)

Unos ejemplos básicos para arrancar:

```
// Multiplicamos y comparamos.
System.out.println(2*3 == 6);

System.out.println(2*3 == 2 + 2 + 2);

// Comparamos dos cadenas.
System.out.println("peras" == "manzanas");

// Atenti al uso de los paréntesis.
System.out.println("¿las peras NO son manzanas? " + ("peras" != "manzanas"));
```

```
// Multiplicamos.
System.out.println(2*3);
```

Prueben ahora agregarle una cadena de texto (string):

```
   // Agregamos texto a la cuenta.
   System.out.print("Se sabe que 2 x 3 es: " +    2*3);
//         ^                    ^            ^     ^
//Método pa' imprimir    Cadena de texto     OP   Cuentita
```

Bueno, parece que todo marcha bien, probemos qué pasa si en lugar de multiplicar sumamos…

```
// Agregamos texto a la cuenta.
System.out.print("Todo el mundo sabe que habrá un 5: " + 2 + 3);
```

⁉️ ¡¿Qué pasoooó?! 🤔

Les dejamos este interrogante para que traten de entender por qué no sumó, sino que "juntó" los números.

🕵🏽 **Pistas:**

- Cuando multiplicamos, todo salió como esperamos. ¿Cuál puede ser la diferencia?
- ¿Se acuerdan de matemática y las precedencias (o el "orden de evaluación")? Algo muy parecido está pasando acá.

Un pequeño adelanto: Esta herramienta que usamos se descomopone en:

- `System` --> Es una clase (`class`) de la librería Java. De allí su nombre "sistema".
- `out` --> Es un attributo (o campo) de la clase. Esta palabra es "salida" en español. Bastante clarito su nombre :)
- `print` y `println` son dos métodos de la clase. *Print* es "imprimir" y *println* probablemente sea la contracción de "imprimir línea". Es decir que termina la línea escribiendo una línea nueva (lo que nosotrxs hacemos cuando apretamos la tecla enter).

4. Clase 02: Variables y Expresiones

4.1 Clase 02: Variables + Expresiones

En esta clase veremos qué son las variables y constantes, aprenderemos a definirlas y usarlas junto con los elementos que vimos hasta ahora (nuestros tipos primitivos, cadenas de texto o `String` y operadores). Veremos también a qué se llama "expresión" y cómo las construimos.

4.1.1 Variables

Una variable es un contenedor en el que se guarda un valor o dato. Sería algo similar a una caja donde *metemos* un valor y a la que le ponemos una etiqueta en el frente. La etiqueta es el nombre o, técnicamente hablando el **identificador** de la variable, que nos va a permitir acceder a ella (y así a su valor), manipularla y reutilizarla.

Las variables nos permiten reutilizar valores tantas veces como queramos sin tener que estar repitiendo ese valor cada vez. El nombre de la variable se convierte entonces en una **representación del valor** que le *asignamos* (atención a esta palabrita).

Por otro lado, y como lo indica su nombre, las variables pueden cambiar su valor a lo largo de nuestro programa. Por ejemplo, una variable `nombre` puede tener un valor inicial de "**Juan Domingo**" luego ese valor puede transformarse en "**Cristina**".

> **Variable**
>
> Una variable es un elemento en el que podemos almacenar datos.
>
> - Tienen un identificador (o nombre) 🏷️
> - Su valor puede modificarse a lo largo de la ejecución del programa 🔄

¿Cómo creamos entonces una variable? Bien sencillo, vamos a escribir el nombre o identificador de nuestra variable, pero ¡OJO! 👀 En Java, al ser un lenguaje de tipado estricto,

debemos además decirle al sistema qué tipo de dato vamos a querer guardar ahí. Esto se llama en programación **declaración de variables**. Porque al crearla estamos también declarando su **tipo** y comprometiéndonos con el sistema a sólo guardar valores que sean admitidos por ese tipo. Veamos:

```
byte totalAsientos;
```

Con esa línea ya estamos declarando nuestra variable y le estamos diciendo al sistema que `totalAsientos` tendrá un valor del tipo `byte`. El sistema entonces nos tomará la palabra y reservará en memoria ese cachito de espacio minúsculo que requiere un byte. Noten como nuestra línea termina con un punto y coma. Eso forma parte de la sintaxis de Java y lo veremos más adelante cuando veamos Expresiones.

Vamos ahora a **asignarle** un valor a nuestra flamante variable. ¿Les suena de algún lado esa palabrita "asignar"? Ciertamente, ya la habíamos visto cuando vimos Operadores, así que usaremos el operador de asignación (`=`) para asignarle un valor:

```
totalAsientos = 125;
```

Estas dos líneas de código también se pueden sintetizar en una sola si queremos **declarar** nuestra variable Y **asignarle** un valor en el mismo momento, simplemente unimos las dos sentencias:

```
  byte totalAsientos  =  125;
//  ^        ^        ^   ^
// TIPO    NOMBRE     OP  VAL
```

De esta manera estaremos declarando nuestra variable de tipo byte y le asignaremos un valor numérico de 125.

⚠️ Atención máxima 👀

Ahora bien, ¿qué pasa si cuando estamos desarrollando nuestro programa algo cambia y resulta que el total de asientos disponibles ya no es 125, sino 300?

Cuando decimos que el valor de la variable puede cambiar y el sistema no nos va a reprochar nada, pues las *variables varían*… debemos prestar mucha atención a un tema ya a esta altura recurrente: **Java es un lenguaje estricto con sus tipos de datos**.

Ya nos habíamos comprometido con el sistema a guardar valores de tipo byte en nuestra variable `totalAsientos`, y el sistema ya nos había reservado ese espacio en memoria, pero resulta que `300` ¡no es un valor válido en el tipo `byte`! ⓘ

Así que si intentamos cambiar el valor inicial de nuestra variable a 300 lo que sucederá es que el sistema simplemente no nos va a dejar y nos indicará el error [1].

Por eso es importante siempre intentar anticiparse a este tipo de cambios o, dicho de otra forma, no confiarse en que las especificaciones no cambiarán. Siempre será mejor trabajar con tipos de datos que tengan un buen margen para esos cambios.

Lxs invitamos a que hagan la prueba de modificar su variable de tipo byte a un valor más grande que el admitido, a ver qué pasa, si total... la diferencia es muy chiquita 👉🏽. Qué tan grave pueden ser unos bits más unos bits menos 🤪

4.1.2 Constantes

Las constantes son un tipo especial de Variable. Y por su nombre suponemos ya sospechan qué es lo que será diferente en este elemento: su mutabilidad. Evidentemente las constantes son... constantes y no pueden variar su valor a lo largo del programa.

> **Constantes**
>
> Las constantes son un contenedor en el que podemos guardar datos o valores.
>
> - Tienen un identificador 🏷️
> - Su valor es inalterable durante toda la ejecución del programa 🔒

Cualquier dato que no cambie es candidato a guardarse en una constante. Ejemplos:

- El número pi ( )
- La cantidad de meses que tiene un año
- La velocidad de la luz
- La burguesía Argentina :P

La declaración de las constantes es igual a la de las variables, pero le tenemos que sumar una palabra que las definirá como constantes: `final`. Veamos unos ejemplos.

```
// Declaramos las constantes porque hay cosas que nunca cambian (¿?).
final double NUMERO_PI = 3.1415926535;
final String MEJOR_PAIS = "Argentina";
```

Identificadores

Para usar los identificadores, que no son más que el 'nombre' que le damos a nuestras variables y constantes, debemos tener algunas reglas en cuenta.

Sintaxis de los identificadores

- No se pueden utilizar palabras reservadas
- El identificador debe ser único.
- Deben comenzar siempre por una letra (1).
- Los siguientes caractéres pueden ser letras, dígitos, guión bajo (`_`) o el signo de pesos (`$`).
- Se distingue entre mayúsculas y minúsculas. (2)
- No hay una largo máximo ni mínimo establecido.

1. La gente de Oracle (la empresa que desarrolló Java) recomienda NO utilizar giuones bajos (`_`) ni signo de pesos (`$`) como primer caracter de un identificador aunque estos estén permitidos.
2. 🅰️ 👀 Java es sensible a mayúsculas y minúsculas, con lo cual, el identificador `coso` será diferente a `Coso` o `COSO` (o cualquier otra variación).

Además de estas reglas de sintaxis hay otro conjunto de cuestiones a tener en cuenta que se suelen denominar "buenas prácticas" de programación.

Estas son, como su nombre lo indica, una serie de buenos modales a la hora de escribir código, que pueden variar de lenguaje en lenguaje. No van a hacer que nuestro código falle en sentido estricto, pero seguirlas le facilita (¡mucho!) la lectura de nuestro código tanto a otras personas como a nosotrxs mismxs.

Hay bastantes estándares de programación que se pueden seguir y, por supuesto, siempre hay competencia entre cuál estándar es mejor, pero con el tiempo una suele adoptar el que

sea más utilizado en el lenguaje o tecnología que desarrolla o incluso en el ámbito social en el que programamos (grupo de amigxs, trabajo, etc...).

Dentro de las buenas prácticas más extendidas de Java, hay algunas que aplican a los identificadores y las detallamos a continuación:

Buenas prácticas

- Usar identificadores que reflejen el **significado** o el **uso** de los elementos. Ejemplos:
  - `unasCosas` ❌
  - `cantidadSillas` ✅
- Usar "camelCase" (es un anglicismo que se podría traducir como "tipoCamello") para identificar variables. Ejemplos:
  - `cantidad_sillas` ❌
  - `cantidadsillas` ❌
  - `cantidadSillas` ✅
- Usar mayúsculas para identificar constantes. Ejemplos:
  - `mejor_pais` ❌
  - `MejorPais` ❌
  - `MEJOR_PAIS` ✅
- Se recomienda no usar nombres muy largos ni muy cortos. Los nombres cortos solo se recomiendan para variables *temporales* de corta vida útil.
- No podremos usar tildes, diéresis ni `ñ` en nuestros identificadores 💔
  - `añoActual` ❌
  - `canciónFinal` ❌
  - `PINGÜINOS` ❌

4.1.3 Expresiones

Junto con los operadores, las **expresiones son los elementos básicos con los que armaremos nuestras sentencias**.

Repasemos qué era una sentencia: visualmente las vimos como líneas de código (1) que representan una **acción** que debe ser ejecutada. Los bloques nos permiten agrupar líneas de código que serán "leídas" como una única sentencia.

1. En Java -y otros lenguajes- terminan con un punto y coma `;`

En términos muy muy generales diremos que **una sentencia NO devuelve nada** y que **una expresión SÍ devuelve alguna cosa**.

Podemos definir una expresión como un fragmento de código que utiliza operadores para manipular valores y producir un resultado.

Lo que en matemática llamamos "hacer una cuenta" sería en programación "crear una expresión". La diferencia principal es que las expresiones en programación además de valores numéricos (como en las cuentas) podemos también trabajar con palabras, caractéres y *booleanos*.

> **Expresiones**
>
> Una expresión es un conjunto de operadores, valores, variables, constantes y funciones que devuelve un resultado.

Veamos un ejemplo básico para determinar qué es una expresión, qué es un operador y qué es una sentencia y poder así diferenciarlos.

```
c = a + b;
```

- `a`, `b` y `c`: Son **valores** o datos.
- `=` y `+`: Son **operadores**.
- `a + b`: Es una **expresión**.
- `c = a + b;`: Es una **sentencia**.

[Agregar otros ejemplos fáciles para que detecten en clase qué es cada cosa]

Type cast y precedencia ?

4.1.4 Palabras reservadas

Palabras reservadas en Java

abstract	continue	for	new	switch
assert	default	goto	package	synchronize
boolean	do	if	private	this
break	double	implement	protected	throw
byte	else	import	public	thows
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

1. En situaciones excepcionales no nos va a tirar un error, sino que obtendremos valores totalmente inesperados. Ya veremos esto en detalle más adelante. ↩

4.2 Ejercicios para la clase 02

En esta clase vimos variables, constantes y expresiones.

Primeros pasos para estos ejercicios:

1. Abrir la IDE (Netbeans)
2. Crear un nuevo programa (o abrir uno existente)
    a. Si estás creando uno nuevo, acordate de seleccionar la opción "Java with Ant".

4.2.1 Ejercicio 01: Suma de enteros

Declará dos variables enteras `a` y `b`, asignales valores y mostrá por consola la suma de ambas.

4.2.2 Ejercicio 02: Área de un rectángulo

Usá dos variables base y altura (tipo double) y calculá el área (fórmula: `base * altura`). Mostrá el resultado.

4.2.3 Ejercicio 03: Promedio de tres números

Imaginemos que tenemos que sacar el promedio de una materia que tuvo 3 exámenes.

1. Guardá cada nota en una variable de tipo entera
2. Calculá el promedio
3. Mostralo con decimales

4.2.4 Ejercicio 04: Intercambio de valores

Tenés dos variables: `x = 5` e `y = 10`. Intercambiá los valores y mostrá el resultado.

4.2.5 Ejercicio 05: Conversión de temperatura

Imaginemos que tenemos un tío paquete que vive en Canadá y cada vez que hablamos por teléfono nos pregunta cómo está el clima en Buenos Aires. Hagamos un programita que nos ayude a convertir los grados de Celcius a Farenheit:

1. Declará una variable para la temperatura actual en grados Celcius
2. Usá la fórmula `F = (C * 9/5) + 32`
3. Mostrá la temperatura en grados Fahrenheit

4.2.6 Ejercicio 06: Cálculo de sueldo

Guardá en variables el sueldo base y un bono extra. Calculá el sueldo total y mostralo. (Pensá si los sueldos y el bono son valores enteros o reales)

4.2.7 `java.util.Scanner`

4.2.8 Ejercicio 07: Edad en meses y días

Pedí que se ingrese por teclado un valor y guardala en una variable edadAnios, calculá la edad en meses y en días (aproximando con 365 días al año). Mostrá ambos resultados.

4.2.9 Ejercicio 08: Concatenación de cadenas

Pedir que se ingrese por teclado nombre y apellido. Mostrá un mensaje que diga "Hola, Nombre Apellido".

4.2.10 Ejercicio 09: Número par o impar

Ingresar el valor de una variable entera numero, mostrá en consola si es par o impar.

4.2.11 Ejercicio 10: Hipotenusa (Teorema de Pitágoras)

1. Ingresar los valores de dos variables catetoA y catetoB (`double`),
2. calculá la hipotenusa con la fórmula: $h = \sqrt{a^2 + b^2}$
3. y mostrá el resultado

4.2.12 Ejercicio 11: Área de un círculo

Pedir que se ingrese el valor del radio `r`, calculá el área con la fórmula: $A = \pi \times r^2$

4.2.13 Ejercicio 12: Perímetro de un cuadrado

Ingresar el valor del valor del lado (`l`), calculá el perímetro con la fórmula: $4 \times l$.

4.2.14 Ejercicio 13: Velocidad promedio

Tenés una distancia (`distanciaKm`) y un tiempo (`horas`). Calculá la velocidad promedio en km/h.

4.2.15 Ejercicio 14: Minutos a horas y minutos

Tenés un número entero de minutos. Convertilo a horas y minutos. Ejemplo: 130 minutos → 2 horas y 10 minutos.

4.2.16 Ejercicio 15: Dígito de las unidades

Ingresar por teclado el valor de un número entero `num`, obtené el dígito de las unidades con `num % 10`.

4.2.17 Ejercicio 16: División entera y resto

Pedir el valor de dos variables enteras a y b, mostrá el cociente entero y el resto.

4.2.18 Ejercicio 17: Potencia

Pedir el valor de dos variables enteras base y exponente, calculá la potencia

4.2.19 Ejercicio 18: Kilogramos a libras

Convertí un peso en kilogramos (kg) a libras usando la fórmula: $libras = kg \times 2.20462$.

4.2.20 Ejercicio 19: Promedio de notas con decimales

Pedir el ingreso de 4 notas. Calculá el promedio y mostrálo con 2 decimales

4.2.21 Ejercicio 20: Doble y triple de un número

Pedir que se ingrese el valor de un número entero, calculá y mostrá el doble y el triple.

4.2.22 Ejercicio 21: Área y perímetro de un triángulo equilátero

Sabiendo el valor de una lado `a`

1. Calculá el perímetro con la fórmula: $P = a \times 3$
2. Calculá el area. Con la fórmula: $A = a^2 \times \frac{\sqrt{3}}{4}$

4.2.23 Ejercicio 22: Conversión de metros a cm y mm

Pedir que se ingrese el valor de una longitud en metros (double), convertila a centímetros y milímetros.

4.2.24 Ejercicio 23: Promedio de velocidad en metros por segundo

Convertí una velocidad en km/h (`vKmH`) a m/s (`vMs`) $vMs = vKmH \times 1000 \div 3600$.

4.2.25 Ejercicio 24: Descuento en un producto

Pedir que se ingresen los valores de un precio y un porcentaje de descuento, calculá el precio final.

4.2.26 Ejercicio 25: Tiempo de viaje

Pedir que se ingrese el valor de una distancia en km y una velocidad en km/h, calculá el tiempo de viaje en horas.

4.2.27 Ejercicio 26: Concatenación de edad

Dado nombre y edad, mostrá el mensaje "Me llamo y tengo años."

4.2.28 Ejercicio 27: Interés simple

Calculá el interés simple con la fórmula: I = capital * tasa * tiempo.

4.2.29 Ejercicio 28: Interés compuesto

Calculá el monto final con la fórmula: M = capital * (1 + tasa)^tiempo.

4.2.30 Ejercicio 29: Conversión de dólares a pesos

Pedir que se ingrese un monto en dólares y un tipo de cambio, calculá el equivalente en pesos.

4.2.31 Ejercicio 30: Conversión de segundos a horas:minutos:segundos

Pedir que se ingrese el valor de un entero segundosTotales, convertilo a formato `hh:mm:ss`.

4.2.32 Ejercicio 31: Promedio ponderado

Tenés tres notas: n1, n2, n3 con pesos 0.2, 0.3 y 0.5 respectivamente. Calculá el promedio ponderado.

4.2.33 ~~Ejercicio 32: Calcular IMC (Índice de Masa Corporal)~~

Pedir que se ingresen los valores del peso en kg y la altura en metros, calculá el IMC con la fórmula: IMC = peso / (altura * altura).

4.2.34 Ejercicio 33: Promedio de edad de un grupo

Pedir el ingreso de la edad de 4 personas en variables y calculá el promedio.

4.2.35 Ejercicio 34: Separar decenas y unidades

Pedir que se ingrese el valor de un número de dos cifras, obtené la decena y la unidad. Por ejemplo, si el número entero es 23 la decena es el 2 y la unidad es el 3.

5. Clase 03: Variables II

5.1 Clase 03: Más variables

5.2 Ejercicios para la clase 03

En esta clase vimos variables, constantes y expresiones.

Primeros pasos para estos ejercicios:

1. Abrir la IDE (Netbeans)
2. Crear un nuevo programa (o abrir uno existente)
    a. Si estás creando uno nuevo, acordate de seleccionar la opción "Java with Ant".

5.2.1 Ejercicio 01

6. Clase 04: Estructuras de control

6.1 Clase 04: Estructuras de control

6.1.1 Condicionales (¿o expresiones lógicas o da lo mismo?)

if / else / elseif

switch

6.1.2 Loops

for

while

do-while

Salidas con break y continue

6.2 Ejercicios para la clase 04

En esta clase vimos variables, constantes y expresiones.

Primeros pasos para estos ejercicios:

1. Abrir la IDE (Netbeans)
2. Crear un nuevo programa (o abrir uno existente)
    a. Si estás creando uno nuevo, acordate de seleccionar la opción "Java with Ant".

6.2.1 Ejercicio 01

Diseñar una aplicación que solicite al usuario que ingrese un número, y mostrar si ese número es par o impar.

7. Clase 05: Funciones

7.1 Clase 05: Funciones

Reutilización, y estructuración del código.

7.1.1 Visibilidad

7.1.2 Scope (ámbito?)

7.1.3 Parámetros y Argumentos

7.1.4 Retornos / valores de devolución

7.2 Ejercicios para la clase 05

En esta clase vimos variables, constantes y expresiones.

Primeros pasos para estos ejercicios:

1. Abrir la IDE (Netbeans)
2. Crear un nuevo programa (o abrir uno existente)
    a. Si estás creando uno nuevo, acordate de seleccionar la opción "Java with Ant".

7.2.1 Ejercicio 01

8. Clase 06: Funciones II

8.1 Clase 06: Funciones segunda parte

8.2 Ejercicios para la clase 06

En esta clase vimos variables, constantes y expresiones.

Primeros pasos para estos ejercicios:

1. Abrir la IDE (Netbeans)
2. Crear un nuevo programa (o abrir uno existente)
    a. Si estás creando uno nuevo, acordate de seleccionar la opción "Java with Ant".

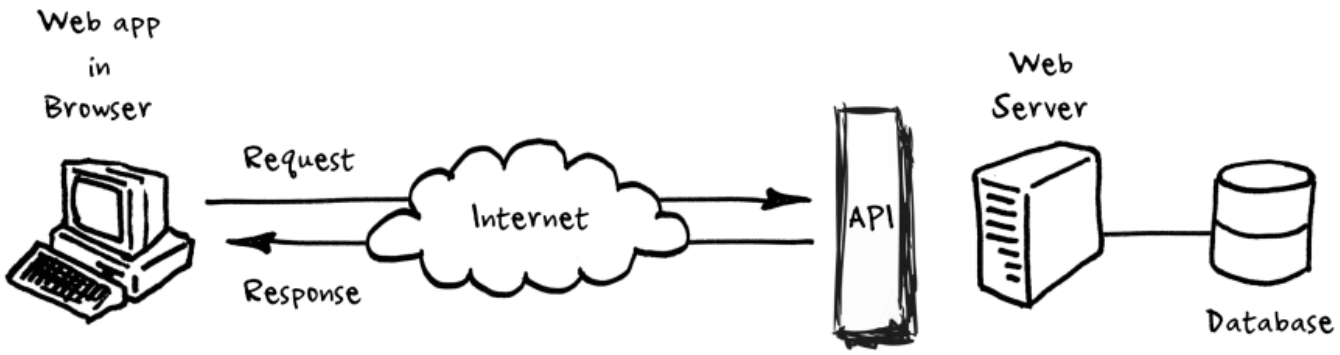
8.2.1 Ejercicio 01

9. Clase 07: API de Java

9.1 Clase 07: API de Java

El término API viene del inglés *Application Programming Interface* y lo podríamos traducir como Interfaz de Programación de Aplicaciones.

Ejemplo del mozo como la API entre servidor (cocina) y cliente. Pedido del cliente (request), platos (response).



9.2 Ejercicios para la clase 07

En esta clase vimos variables, constantes y expresiones.

Primeros pasos para estos ejercicios:

1. Abrir la IDE (Netbeans)
2. Crear un nuevo programa (o abrir uno existente)
    a. Si estás creando uno nuevo, acordate de seleccionar la opción "Java with Ant".

9.2.1 Ejercicio 01

10. Clase 08: Clases

10.1 Clase 08: Clases

10.2 Ejercicios para la clase 08

En esta clase vimos variables, constantes y expresiones.

Primeros pasos para estos ejercicios:

1. Abrir la IDE (Netbeans)
2. Crear un nuevo programa (o abrir uno existente)
    a. Si estás creando uno nuevo, acordate de seleccionar la opción "Java with Ant".

10.2.1 Ejercicio 01

11. Clase 09: Arrays

11.1 Clase 09: Arrays (o vectores)

11.2 Ejercicios para la clase 09

En esta clase vimos variables, constantes y expresiones.

Primeros pasos para estos ejercicios:

1. Abrir la IDE (Netbeans)
2. Crear un nuevo programa (o abrir uno existente)
    a. Si estás creando uno nuevo, acordate de seleccionar la opción "Java with Ant".

11.2.1 Ejercicio 01

12. Glosario

13. Ejercicios

13.1 Ejercicios sueltos

https://github.com/anairamzap/curso-polito