Pre-Work de Java

1. Instalación del Entorno

Antes de comenzar, asegúrate de tener instalado lo siguiente:

- **Java Development Kit (JDK)**: Descarga e instala la última versión del JDK desde Oracle o usa OpenJDK.
- **IDE (Entorno de Desarrollo Integrado)**: **Eclipse**, o **VS Code** con la extensión de Java.

2. Conceptos Básicos de Java

2.1. Hola Mundo en Java

```java
public class HolaMundo {
    public static void main(String[] args) {
        System.out.println("Hola, Mundo!");
    }
}
```

2.1.1 Explicación del Código

- `public class HolaMundo {`: Esta línea define una **clase** llamada "HolaMundo". En Java, todo el código debe estar dentro de una clase.
- `public static void main(String[] args) {`: Este es el método principal, donde comienza la ejecución del programa.
- `System.out.println("Hola Mundo");` : Esta línea imprime el mensaje "Hola Mundo" en la terminal.

2.1.2 Compilación del Código

Si estás utilizando un editor de texto, abre una terminal o línea de comandos, navega hasta el directorio donde guardaste el archivo "`HolaMundo.java`" y ejecuta el siguiente comando:

Recuerda que tienes una guía práctica sobre cómo utilizar la terminal
Recuerda también que el símbolo `$` *a inicio de línea es para indicar que ese texto es una instrucción para la terminal*

```
$ javac HolaMundo.java
```

Esto generará un archivo llamado "`HolaMundo.class`" que contiene el código compilado.

Si estás utilizando un IDE, generalmente hay un botón para compilar el código directamente.

2.1.3 Ejecución del Código

En la terminal o línea de comandos, ejecuta el siguiente comando:

```
$ java HolaMundo
```

Si estás utilizando un IDE, generalmente hay un botón para ejecutar el código directamente.

2.1.4 ¡Resultado!

Deberías ver el mensaje "`Hola Mundo`" impreso en la consola. ¡Felicidades, has escrito tu primer programa en Java!

2.2. Variables y Tipos de Datos

Java es un lenguaje de programación **tipado**, para declarar una variable necesitas especificar su tipo de dato y su nombre. Luego, puedes asignarle un valor. Desde que la variable esté declarada únicamente podrá tener valores de ese tipo de dato.

```java
int numero = 10;
double decimal = 5.5;
char letra = 'A';
boolean esVerdadero = true;
String mensaje = "Hola Java";
```

Imagina una variable como una caja donde puedes guardar información. Esta caja tiene una etiqueta (el nombre de la variable) y puede contener diferentes tipos de información (el tipo de dato).

2.2.1 Tipos de Datos en Java

Java tiene varios tipos de datos que te permiten almacenar diferentes tipos de información. Aquí tienes los más comunes:

- `int`: Se utiliza para números enteros, como 10, -5 o 1000.
- `double`: Se utiliza para números decimales, como 5.5, 3.14 o -2.7.
- `char`: Se utiliza para un solo carácter, como 'A', 'z' o '$'. Se declara entre comillas simples.
- `boolean`: Se utiliza para valores verdaderos o falsos (true o false).
- `String`: Se utiliza para almacenar cadenas de texto, como "Hola Java" o "Esta es una frase". Se declara entre comillas dobles.

* Hay más tipos de datos primitivos, y otros que no son primitivos. Lo veremos durante el curso.

2.2.2 Explicación del Ejemplo

- **`int numero = 10;`** : Se crea una variable llamada `numero` de tipo `int` y se le asigna el valor 10.
- **`double decimal = 5.5;`** : Se crea una variable llamada `decimal` de tipo `double` y se le asigna el valor 5.5.
- **`char letra = 'A';`** : Se crea una variable llamada `letra` de tipo `char` y se le asigna el valor `'A'`.
- **`boolean esVerdadero = true;`** : Se crea una variable llamada `esVerdadero` de tipo `boolean` y se le asigna el valor true.
- **`String mensaje = "Hola Java";`** : Se crea una variable llamada `mensaje` de tipo `String` y se le asigna el valor `"Hola Java"`.

2.2.4 Consejos Adicionales

- Los nombres de las variables pueden contener letras, números y guiones bajos, pero no pueden empezar con un número.
- Los nombres se declaran en `camelCase`: primera palabra en minúscula y el resto de palabras empieza por mayúscula, sin espacios.
- Java es sensible a mayúsculas y minúsculas, por lo que `numero` y `Numero` se consideran variables diferentes.
- Puedes declarar varias variables del mismo tipo en una sola línea, separándolas por comas: `int x = 5, y = 10, z = 15;`

2.3. Operadores Básicos

```java
int a = 10, b = 5;
System.out.println(a + b); // Suma
System.out.println(a - b); // Resta
System.out.println(a * b); // Multiplicación
System.out.println(a / b); // División
System.out.println(a % b); // Módulo
```

En Java, al igual que en matemáticas, existen operadores que nos permiten realizar diversas operaciones con las variables. En este ejemplo, te mostraré los operadores básicos y cómo utilizarlos:

2.3.1 Declaración de Variables

Primero, declaramos dos variables enteras llamadas `a` y `b`, y les asignamos los valores 10 y 5 respectivamente.

2.3.2 Operadores Aritméticos

Java nos ofrece los siguientes operadores aritméticos básicos:

Operador	**Operación**	**Definición**
+	Suma	Suma de dos valores
-	Resta	Resta de dos valores
*	Multiplicación	Multiplica dos valores
/	División	Divide dos valores
%	Módulo	Devuelve el resto de la división entre dos valores. *Útil para valorar si un número es o no par:* `numero % 2` *Si ese resultado es 0 es porque es divisible entre 2, por ende es par.*

2.3.3 Ejemplo Completo

Aquí tienes el ejemplo completo con todos los operadores:

```java
int a = 10, b = 5;

System.out.println("Suma: " + (a + b)); // Imprime 15
System.out.println("Resta: " + (a - b)); // Imprime 5
System.out.println("Multiplicación: " + (a * b)); // Imprime 50
System.out.println("División: " + (a / b)); // Imprime 2
System.out.println("Módulo: " + (a % b)); // Imprime 0
```

3. Control de Flujo

3.1. Condicionales

```java
int edad = 18;
if (edad >= 18) {
    System.out.println("Eres mayor de edad");
} else {
    System.out.println("Eres menor de edad");
}
```

Un **condicional** es como una pregunta que le haces a tu programa. Dependiendo de la respuesta (verdadero o falso), el programa hará una cosa u otra. Para poder hacerlo necesitamos comparar un valor frente a otro.

3.1.1 Operadores de comparación

Java nos ofrece los siguientes operadores de comparación:

Operador	Operación	Definición
<	menor que	Compara si un valor es menor que el otro
>	mayor que	Compara si un valor es mayor que el otro
<=	menor o igual que	Compara si un valor es menor o igual que el otro
>=	mayor o igual que	Compara si un valor es mayor o igual que el otro
==	igual	Compara si un valor es igual que el otro
!=	distinto	Compara si un valor es distinto que el otro

3.1.1 Estructura del Condicional `if-else`

En Java, el condicional más común es el `if-else`. Su estructura es la siguiente:

```java
if (condición) {
    // Código que se ejecuta si la condición es verdadera
} else {
    // Código que se ejecuta si la condición es falsa
}
```

En el ejemplo, tenemos la variable `edad` con un valor de 18. El condicional `if-else` pregunta: "¿Es la edad mayor o igual a 18?".

```
int edad = 18;
if (edad >= 18) {
    System.out.println("Eres mayor de edad");
} else {
    System.out.println("Eres menor de edad");
}
```

- **`if (edad >= 18)`** : Esta es la condición. Se evalúa si la edad es mayor o igual a 18. Devolviendo **`true`** en caso de cumplirse y ejecutando la siguiente línea de código; **`false`** en caso de no cumplirse pasando al bloque de código **`else`**.
- **`System.out.println("Eres mayor de edad");`** : Este código se ejecuta si la condición es verdadera (la edad es mayor o igual a 18).
- **`else`** : La palabra clave `else` indica que hay un código alternativo para ejecutar si la condición es falsa.
- **`System.out.println("Eres menor de edad");`** : Este código se ejecuta si la condición es falsa (la edad es menor a 18).

3.1.3 ¿Qué Imprime el Programa?

En este caso, como la edad es 18, la condición `edad >= 18` es **`true`**. Por lo tanto, el programa imprimirá:

```
Eres mayor de edad
```

3.2. Bucles

```java
// Bucle for
for (int i = 0; i < 5; i++) {
    System.out.println("Iteración " + i);
}

// Bucle while
int contador = 0;
while (contador < 5) {
    System.out.println("Contador: " + contador);
    contador++;
}
```

En programación, un **bucle** es como una instrucción que le dices a tu programa para que repita un bloque de código varias veces. Java nos ofrece dos tipos principales de bucles: `for` y `while`.

3.2.1 Bucle `for`

El bucle `for` se utiliza cuando sabes de antemano cuántas veces quieres repetir un bloque de código. Su estructura es la siguiente:

```java
for (inicialización; condición; actualización) {
    // Código que se repite
}
```

- **inicialización**: Se ejecuta una vez al principio del bucle. Sirve para declarar e inicializar una **variable de control** (en este caso, `i`).
- **condición**: Se evalúa antes de cada iteración. Si la condición es `true`, se ejecuta el código dentro del bucle. Si es `false`, el bucle termina.
- **actualización**: Se ejecuta después de cada iteración. Sirve para modificar la variable de control (en este caso, `i++` incrementa su valor en 1).

Ejemplo:

```java
// Bucle for
for (int i = 0; i < 5; i++) {
    System.out.println("Iteración " + i);
}
```

Este bucle imprimirá:
Iteración 0
Iteración 1
Iteración 2
Iteración 3
Iteración 4

3.2.2 Bucle `while`

El bucle `while` se utiliza cuando **no** sabes de antemano cuántas veces quieres repetir un bloque de código. Su estructura es la siguiente:

```java
while (condición) {
    // Código que se repite
}
```

- **condición**: Se evalúa antes de cada iteración. Si la condición es `true`, se ejecuta el código dentro del bucle. Si es `false`, el bucle termina.

Ejemplo:

```java
// Bucle while
int contador = 0;
while (contador < 5) {
    System.out.println("Contador: " + contador);
    contador++;
}
```

Este bucle imprimirá:

Contador: 0
Contador: 1
Contador: 2
Contador: 3
Contador: 4

3.2.3 ¿Cuándo Usar for o while?

- Si sabes cuántas veces quieres repetir el código, usa for.
- Si no sabes cuántas veces quieres repetir el código, usa while.

4. Métodos y Programación Orientada a Objetos (POO)

4.1. Métodos en Java

```java
public static int suma(int a, int b) {
    return a + b;
}
```

En Java, un método es como una mini-máquina que hace un trabajo específico. Recibe información (llamada "parámetros"), la procesa y te da un resultado (llamado "valor de retorno").

4.1.1 Estructura de un Método

En Java, la estructura general de un método es así:

```java
public static tipoRetorno nombreMetodo(tipoParametro1 parametro1, tipoParametro2 parametro2, ...) {
    // Código que realiza la función
    return valorDeRetorno;
}
```

- `public static`: Estas palabras clave indican cómo se puede acceder al método (en este caso, desde cualquier parte del programa).
- `tipoRetorno`: Especifica el tipo de dato que devuelve (por ejemplo, `int`, `double`, `String`, etc.).
- `nombreMetodo`: Es el nombre que le das al método (por ejemplo, `suma`, `calcularArea`, `mostrarMensaje`, etc.).
- `(tipoDeParametro1 parametro1, ...)`: Son los parámetros que el método recibe. Cada parámetro tiene un tipo de dato y un nombre (algo parecido a la declaración de variables).
- `{ ... }`: Dentro de las llaves va el código con las instrucciones que realiza el método.
- `return valorDeRetorno;` : La palabra clave `return` se utiliza para devolver el resultado.

Ejemplo: Función `suma`

```java
public static int suma(int a, int b) {
    return a + b;
}
```

- `public static`**:** el método es accesible desde cualquier parte del programa.
- `int`: Indica que la función devuelve un valor de tipo `int` (entero).
- `suma`**:** Es el nombre de la función.
- `(int a, int b)`: Indica que la función recibe dos parámetros de tipo `int`: `a` y `b`.
- `return a + b;`: La función suma los dos parámetros y devuelve el resultado.

4.1.2 ¿Cómo usar un método?

Para usar un método, simplemente la llamas por su nombre y le pasas los parámetros necesarios:

```
int resultado = suma(5, 3); // Llamamos a la función suma con los valores 5 y 3
System.out.println(resultado); // Imprime 8
```

4.2. Clases y Objetos

```
class Persona {
    String nombre;
    int edad;

    // Constructor
    public Persona(String nombre, int edad) {
        this.nombre = nombre;
        this.edad = edad;
    }

    // Método
    public void mostrarInformacion() {
        System.out.println("Nombre: " + nombre + ", Edad:
    }
}

// Uso de la clase
public class Main {
    public static void main(String[] args) {
        Persona persona1 = new Persona("Juan", 25);
        persona1.mostrarInformacion();
    }
}
```

En **programación orientada a objetos (POO)**, las clases y los objetos son los pilares fundamentales. Vamos a desglosar este concepto con el ejemplo:

4.2.1 Clases

Imagina una clase como un plano para construir objetos. Define las características (atributos) y los comportamientos (métodos) que tendrán los objetos creados a partir de ella.

Así como cuando hemos visto el capítulo de cómo se declara una variable se indica como buena práctica utilizar `camelCase` para los nombres, igual que se utiliza para dar nombre a los métodos; con las clases es algo distinto. Se utiliza `PascalCase` para poder diferenciarlas. `PascalCase` implica que todas las palabras empiezan por mayúscula, incluida la primera. Por eso una clase, como es el caso de `String` empieza en mayúscula a pesar de ser una única palabra.

Ejemplo:

- `class Persona`: Declara una clase llamada `Persona`.
- `String nombre;` : Atributo que representa el nombre de la persona.
- `int edad;` : Atributo que representa la edad de la persona.
- `public Persona(String nombre, int edad)`: Constructor. Es un método especial que se utiliza para instanciar, crear, objetos de la clase `Persona`. Recibe el nombre y la edad como parámetros y los asigna a los atributos correspondientes.
- `public void mostrarInformacion()`: Método que muestra la información de la persona (nombre y edad) en la teminal.

4.2.2 Objetos

Un objeto es una instancia concreta de una clase. Es como una casa construida a partir del plano (la clase). Cada objeto tiene sus propios valores para los atributos y puede utilizar los métodos definidos en la clase.

Imagina el mando de la PlayStation como un a instancia de un objeto. La clase es el mando, en la clase está el plano de los diferentes atributos (boton start, select, triángulo, cruz, cuadrado, círculo, flechas, color…) y los diferentes métodos (acciones que puede realizar cada botón o combinación de ellos). Además en la clase está el constructor, este método especial para poder instanciar (crear) un objeto de la clase. ¿Cuántos mandos de la PlayStation existen? Miles. Pero en realidad son todos lo mismo, así que son diferentes instancias de un objeto de una misma clase, salvo que uno es de color rojo, el otro es negro, el otro es blanco…

Ejemplo:

```java
// Uso de la clase
public class Main {
    public static void main(String[] args) {
        Persona persona1 = new Persona("Juan", 25);
        persona1.mostrarInformacion();
    }
}
```

- `Persona persona1 = new Persona("Juan", 25);` : Se crea un objeto llamado `persona1` de la clase Persona. Se utiliza el constructor para inicializar los atributos con los valores "Juan" y 25.
- `persona1.mostrarInformacion();` : Se llama al método `mostrarInformacion()` del objeto `persona1`. Este método imprime en la consola:

```
Nombre: Juan, Edad: 25
```

5. Manejo de Excepciones

```
try {
    int resultado = 10 / 0;
} catch (ArithmeticException e) {
    System.out.println("Error: No se puede dividir por cer
} finally {
    System.out.println("Fin del bloque try-catch.");
}
```

En Java, las **excepciones** son como errores que pueden ocurrir durante la ejecución de un programa. Manejar excepciones es como poner redes de seguridad para que tu programa no se caiga si algo sale mal.

5.1 Estructura `try-catch-finally`

Java nos ofrece un mecanismo llamado `try-catch-finally` para manejar excepciones. Su estructura es la siguiente:

```
try {
    // Código que puede lanzar una excepción
} catch (TipoDeExcepcion e) {
    // Código que se ejecuta si ocurre la excepción
} finally {
    // Código que se ejecuta siempre, haya o no excepción
}
```

- `try`: Encierra el código que puede lanzar una excepción.
- `catch`: Captura un tipo específico de excepción y define el código que se ejecutará si ocurre esa excepción.
- `finally`: Contiene código que siempre se ejecuta, haya o no excepción.

5.2 Ejemplo: División por Cero

En el ejemplo, tenemos un bloque `try` que intenta dividir 10 entre 0. Esto lanzará una excepción `ArithmeticException` porque no se puede dividir por cero:

- `try { ... }`: El código `int resultado = 10 / 0;` puede lanzar una excepción.
- `catch (ArithmeticException e) { ... }`: Captura la excepción `ArithmeticException` (división por cero). Si ocurre esta excepción, se ejecuta el código dentro del bloque `catch`, que imprime un mensaje de error.
- `finally { ... }`: El código `System.out.println("Fin del bloque try-catch.");` se ejecuta siempre, haya o no excepción.

5.3 ¿Qué Imprime el Programa?

En este caso, como se produce una división por cero, el programa imprimirá:

```
Error: No se puede dividir por cero.
Fin del bloque try-catch.
```

6. Colecciones en Java

6.1. Listas

```java
import java.util.ArrayList;
ArrayList<String> lista = new ArrayList<>();
lista.add("Java");
lista.add("Python");
System.out.println(lista);
```

En Java, las **listas** son como contenedores flexibles donde puedes guardar varios elementos del mismo tipo. A diferencia de los `arrays`, las listas pueden crecer o disminuir su tamaño según necesites.

6.1.1 `ArrayList`

En este ejemplo, se utiliza la clase `ArrayList`, que es una de las formas más comunes de crear listas en Java.

```java
import java.util.ArrayList;
public class Ejemplo {
    public static void main(String[] args) {

        // Creando un ArrayList, llamada lista, vacía
        ArrayList<String> lista = new ArrayList<>();

        // Añadiendo elementos a lista
        lista.add("Java");
        lista.add("Python");

        // Para que muestre por la terminal qué hay en lista
        System.out.println(lista);
    }
}
```

- `import java.util.ArrayList;` : Esta línea importa la clase `ArrayList` para poder utilizarla.
- `ArrayList<String> lista = new ArrayList<>();` : Se crea una lista llamada `lista` que solo puede contener elementos de tipo `String` (cadenas de texto).
- `lista.add("Java");` : Se añade el elemento "Java" a la lista.
- `lista.add("Python");` : Se añade el elemento "Python" a la lista.

- `System.out.println(lista);` : Se imprime la lista en la consola.

6.1.2 ¿Qué Imprime el Programa?

El programa imprimirá:

```
[Java, Python]
```

Como puedes ver, la lista contiene los elementos "Java" y "Python" en el orden en que fueron añadidos.

6.1.3 Métodos de `ArrayList`

La clase `ArrayList` ofrece muchos métodos útiles para trabajar con listas. Aquí tienes algunos de los más comunes, por ahora no os agobiéis. Lo veremos en profundidad durante el curso:

- `add(elemento)`: Añade un elemento al final de la lista.
- `add(indice, elemento)`: Añade un elemento en una posición específica.
- `get(indice)`: Obtiene el elemento que está en una posición específica.
- `set(indice, elemento)`: Reemplaza el elemento que está en una posición específica con otro elemento.
- `remove(indice)`: Elimina el elemento que está en una posición específica.
- `size()`: Devuelve el número de elementos que tiene la lista.

6.2. Mapas

```java
import java.util.ArrayList;
ArrayList<String> lista = new ArrayList<>();
lista.add("Java");
lista.add("Python");
System.out.println(lista);
```

En Java, un **mapa** es como un diccionario donde puedes guardar información utilizando pares de "clave-valor". Cada clave es única y se utiliza para acceder a su valor correspondiente.

6.2.1 `HashMap`

En este ejemplo, se utiliza la clase `HashMap`, que es una de las formas más comunes de crear mapas en Java.

```java
import java.util.HashMap;
public class Ejemplo {
    public static void main(String[] args) {

        // Creando un HashMap, llamado edades, vacío
        HashMap<String, Integer> edades = new HashMap<>();

        // Añadiendo elementos a edades (texto, número)
        edades.put("Ana", 30);
        edades.put("Luis", 25);

        // Para que muestre por la terminal qué hay en edades
        System.out.println(edades.get("Ana"));

    }
}

// Por terminal lo que saldrá es el valor que tiene "Ana"
// 30
```

- `import java.util.HashMap;` : Esta línea importa la clase `HashMap` para poder utilizarla.
- `HashMap<String, Integer> edades = new HashMap<>();` : Se crea un mapa llamado `edades` donde las claves son de tipo `String` (nombres) y los valores son de tipo `Integer` (edades).
- `edades.put("Ana", 30);` : Se añade un par clave-valor al mapa. La clave es "Ana" y el valor es 30.
- `edades.put("Luis", 25);` : Se añade otro par clave-valor al mapa. La clave es "Luis" y el valor es 25.
- `System.out.println(edades.get("Ana"));` : Se obtiene el valor asociado a la clave "Ana" y se imprime en la consola.

6.2.2 Métodos de `HashMap`

La clase `HashMap` ofrece muchos métodos útiles para trabajar con mapas. Aquí tienes algunos de los más comunes:

- `put(clave, valor)`: Añade un par clave-valor al mapa.
- `get(clave)`: Obtiene el valor asociado a una clave.

- `remove(clave)`: Elimina el par clave-valor asociado a una clave.
- `containsKey(clave)`: Comprueba si el mapa contiene una clave específica.
- `containsValue(valor)`: Comprueba si el mapa contiene un valor específico.
- `size()`: Devuelve el número de pares clave-valor que tiene el mapa.

7. Tareas Prácticas

Recuerda que Java necesita una clase para funcionar y el método main para poder trabajar. Todos tus ejercicios deberían empezar de la siguiente forma:

```
public class NombreEjercio {
    public static void main(String[] args) {
        // Código del ejercicio
    }
}
```

1. **Escribe un programa que determine si un número es par o impar.**

```
public class ParImpar {
    public static void main(String[] args) {
        // Código necesario

        // Recuerda que si quieres ver algo por la terminal necesitas
        // el siguiente código y poner aquello que quieras ver dentro de
        // los paréntesis
        System.out.println();

    }
}
```

2. **Crea una clase Coche con atributos marca, modelo y año, y un método para mostrar su información.**

```
public class Coche {
    // pon aquí los atributos


    // pon aquí el constructor


    // pon aquí el método para mostrar su información


    public static void main(String[] args) {
        // Instancia el objeto


        // Llama al método para mostrar la información

    }
}
```

3. **Implementa un programa que use un `ArrayList` para almacenar nombres y los imprima en pantalla, utiliza un bucle para iterar por cada elemento que as agregado a la lista.**

```
public class EjercicioArrayList {
    public static void main(String[] args) {
        // Haz una lista vacía


        // Añade elementos a la lista


        // Muestra la lista


        // BONUS: haz un bucle para iterar los elementos de la lista
        // e imprimir cada elemento en una línea distinta


    }
}
```

Con esto, ya estarás listo para iniciar con Java. ¡Mucho éxito!