Guía de Ejercicios

¡Bienvenidos a la primera guía de ejercicios de programación en Java! Este documento está diseñado para ayudarte a mejorar tus habilidades de programación utilizando el paradigma orientado a objetos en Java. Como con cualquier habilidad, la clave es la práctica. Por eso, hemos organizado esta guía en ejercicios que van desde lo más básico hasta desafíos más complejos. No es necesario que los hagas en orden, puedes empezar por el que te llame más la atención, no olvides que ante cualquier duda tienes a tus compañeros en Discord o a nosotros.

Ejercicio 1

Imprimiendo en Consola

Estás empezando a programar en Java y tu primer objetivo es aprender a imprimir mensajes en la consola.

Instrucciones:

- Escribe un programa que imprima "Hola, Mundo" en la consola.
- Luego, imprime tu nombre y tu edad.

Tips de Sintaxis:

- En Java, utilizamos `System.out.println()` para imprimir en la consola, mientras que en Python se usa `print()`.
- Cada línea de código en Java termina con un punto y coma (`;`), a diferencia de Python, que no lo requiere

---

Explicación Adicional:

- **System**: Es una clase en Java que proporciona acceso a recursos del sistema, como la entrada y salida estándar.
- **out**: Es una variable estática de la clase `System` que representa el flujo de salida estándar. En otras palabras, es el canal por el cual se envía el texto a la consola.
- **println()**: Es un método de `PrintStream` (el tipo de `out`) que imprime el texto seguido de un salto de línea.

Ejercicio 2

Instanciación de Variables y Objetos en Java

Este ejercicio te guiará a través de la instanciación de variables de tipos primitivos y objetos en Java. Aprenderás la diferencia entre cómo se manejan en la memoria, y por qué `String` se escribe con mayúscula, destacando su particularidad como clase en Java.

Configuración del Entorno de Desarrollo:

- Crea un nuevo proyecto llamado `InstanciacionVariables`.

Tareas

1. **Instanciación de Variables Primitivas:**

   Instrucciones:

   - Declara e inicializa variables de los siguientes tipos primitivos: `int`, `double`, `boolean`, `char`.
   - Imprime los valores de estas variables en la consola.
2. **Uso de la Clase String:**

   Instrucciones:

   - Declara e inicializa una variable de tipo `String`.
   - Imprime el valor de la variable `String` en la consola.
   - Investiga y explica por qué `String` se escribe con mayúscula y cómo se almacena en memoria.
3. **Instanciación de Objetos:**

   Instrucciones:

   - Crea una clase llamada `Persona` con los siguientes atributos: `nombre` (String) y `edad` (int).
   - Implementa un constructor que inicialice estos atributos.
   - Añade métodos para obtener y modificar los atributos (`getters` y `setters`).
   - En la clase `Main`, crea una instancia de `Persona`, utiliza sus métodos y verifica su funcionamiento.

Tips de Sintaxis:

- En Java, para crear (o instanciar) un objeto de una clase, utilizamos la palabra clave `new`. Esto llama al constructor de la clase y asigna memoria en el heap para el nuevo objeto.

Explicación Adicional:

- **Tipos Primitivos:** Los tipos primitivos en Java incluyen `byte`, `short`, `int`, `long`, `float`, `double`, `boolean`, y `char`. Estos tipos se almacenan directamente en la memoria stack, lo que los hace altamente eficientes para operaciones rápidas y de bajo consumo de memoria.
- **Clase String:** En Java, `String` es una clase y no un tipo primitivo, escrita con mayúscula según la convención de clases. Los objetos `String` se almacenan en la memoria heap y son inmutables, lo que significa que cualquier modificación crea un nuevo objeto en lugar de alterar el original. Esta inmutabilidad asegura seguridad y consistencia, pero requiere un manejo eficiente de la memoria para evitar un uso excesivo de recursos.
- **Objetos:** Los objetos, como las instancias de clases (por ejemplo, `Persona`), se almacenan en la memoria heap. Las referencias a estos objetos se guardan en la memoria stack, lo que permite una gran flexibilidad y la capacidad de manejar estructuras de datos más complejas.

Memoria Stack: La memoria stack sigue una estructura LIFO (Last In, First Out) y se usa para almacenar variables locales y datos de funciones, como parámetros y direcciones de retorno. Es rápida y eficiente porque la gestión es automática y en bloques contiguos, pero tiene un tamaño limitado y solo permite acceder a variables dentro del contexto de la función en que fueron declaradas.

Memoria Heap: La memoria heap es más grande y se utiliza para almacenar objetos y datos que necesitan persistir más allá de una función. Es más lenta debido a la gestión manual y puede fragmentarse, pero ofrece mayor capacidad y flexibilidad. Los datos en el heap se acceden a través de referencias almacenadas en el stack.

Ejercicio 3

Condicionales y Estructura de Control

Escribe un programa que decida si un número es par o impar.

Configuración del Entorno de Desarrollo:

- Crea un nuevo proyecto llamado `parOimpar`.

Instrucciones:

- Declara una variable `numero` y asígnale un valor.
- Escribe una estructura condicional que determine si el número es par o impar e imprime el resultado.

Tips de Sintaxis:

- En Java, utilizamos `if`, `else if`, y `else` para controlar el flujo del programa, igual que en Python, pero con la diferencia de que Java requiere llaves `{}` para definir los bloques de código.
- En Java la comparación de igualdad se hace con `==`.
- Recuerda que los números pares son aquellos que tienen como múltiplo al número dos, es decir, que se pueden dividir en una mitad exacta. Si quieres saber el resto de una división puedes usar el operador (`%`).

Ejercicio 4

Bucle `for`

Crea un programa que imprima los números del 1 al 10 en la consola.

Instrucciones:

- Utiliza un bucle para recorrer los números del 1 al 10.
- Imprime cada número en la consola.

Tips de Sintaxis:

- En Java, los bucles comunes incluyen `for`, `while`, y `do-while`. La estructura de un bucle `for` es diferente a la de Python, ya que en Java se especifica el inicio, la condición y el incremento en una sola línea.
    - Ej.: for (int i = 1; i <= 10; i++) {}

      En la estructura del bucle `for` en Java, los espacios separados por `;` se llaman **componentes del bucle `for`**. Cada uno tiene una función específica:

        - **Inicialización (`int i = 1;`)**: Se ejecuta una vez al principio del bucle. Aquí, se declara una variable `i` y se inicializa en 1 (Puedes usar cualquier nombre de variable).
        - **Condición (`i <= 10;`)**: Antes de cada iteración del bucle, se evalúa esta condición. Si es `true`, se ejecuta el bloque de código del bucle; si es `false`, el bucle termina. En este caso, el bucle continuará mientras `i` sea menor o igual a 10.
        - **Incremento (`i++`)**: Se ejecuta al final de cada iteración. Aquí, `i++` aumenta el valor de `i` en 1 después de cada iteración.
- En Python, los bucles `for` son más simples y suelen iterar directamente sobre una secuencia como una lista o un rango.

Ejercicio 5

Bucle `while`

Implementa un bucle `while` que continúe ejecutándose mientras una condición sea verdadera.

Configuración del Entorno de Desarrollo:

- Crea un nuevo proyecto llamado `CicloWhile`.

Instrucciones:

- Define una clase `Contador`.
- Crea un método que use un bucle `while` para contar desde 1 hasta 10, imprimiendo cada número.
- Modifica el método para que termine de contar si el número llega a 5.

Tips:

- Un bucle `while` sigue ejecutándose mientras la condición que se le da sea `true`. Es útil cuando no se conoce el número exacto de iteraciones.
- La sintaxis es similar al Python pero el bloque de ejecución utiliza llaves `{}`.

```
while (i >= n){
    System.out.println("");
    i++;
}
```

Ejercicio 6

Bucle `do-while`

Este ejercicio te ayudará a comprender cómo funciona el bucle `do-while` en Java, usando una cuenta regresiva de números pares.

Configuración del Entorno de Desarrollo:

1. Crea un nuevo proyecto llamado `CuentaRegresivaPares`.
2. Crea una clase llamada `CuentaRegresiva`.

Tareas:

1. **Implementar la Cuenta Regresiva**:
   - Dentro de la clase `CuentaRegresiva`, define un método llamado `imprimirParesRegresivos`.
   - Este método debe iniciar con un número entero `n` igual a 20.
   - Utiliza un bucle `do-while` para imprimir todos los números pares desde `n` hasta 2.
2. **Condición de Paridad**:
   - Dentro del bucle `do-while`, verifica si `n` es par usando el operador de módulo (`%`).
   - Si es par, imprime `n` en la consola.
   - Decrementa `n` en cada iteración.
3. **Parar la Ejecución**:
   - Asegúrate de que el bucle continúe ejecutándose hasta que `n` sea mayor o igual a 2.

Tips:

- En un `do-while`, la condición se evalúa después de que el bloque de código se haya ejecutado, lo que garantiza al menos una ejecución del bucle. Esto es útil cuando necesitas realizar una acción antes de comprobar una condición.

```
do {

 System.out.println("");

} while (i <= 0 || i >= 10)
```

- para incrementar o decrementar una variable puedes usar los operadores `a++ y a--`

Explicación Adicional:

- **Condicionales**: Utilizamos `if` para verificar si un número es par, es decir, si el residuo de `n % 2` es igual a 0.
- **Decremento**: Reducir el valor de `n` en cada iteración es esencial para evitar un bucle infinito.

Ejercicio 7

Imagina que eres el administrador de una escuela que necesita un sistema para gestionar la información de los estudiantes, como sus nombres, edades, cursos y calificaciones.

Configuración del Entorno de Desarrollo:

- Crea un nuevo proyecto llamado MoodlePremium.

Instrucciones:

- Crea la clase `Estudiante` en Java.
    - **Atributos:**
        - `nombre` (String): Nombre del estudiante.
        - `apellido` (String): Apellido del estudiante.
        - `edad` (int): Edad del estudiante.
        - `curso` (String): Curso en el que está inscrito el estudiante.
        - `calificacionFinal` (double): Calificación final del estudiante.
- Implementa un constructor que inicialice todos los atributos.
- Añade métodos para obtener y modificar los atributos (`getters` y `setters`).
- Implementa un método `mostrarInformacion()` que imprima la información del estudiante en la consola.
- Finalmente, en el método `main`, crea una instancia de `Estudiante` y prueba todos sus métodos.

Ejercicio 8

Gestión de Productos

Imagina que eres el dueño de una pequeña tienda y has decidido mejorar la organización y control de tu negocio. Para ello, vas a desarrollar un sistema que te ayude a gestionar tus productos de manera eficiente. Empieza con lo básico :

Configuración del Entorno de Desarrollo:

1. Crea un nuevo proyecto llamado `GestionProductos`.

Instrucciones:

1. **Creación de la Clase Producto:**
    - Define una clase llamada `Producto`.
    - **Atributos:**
        - `nombre`: de tipo `String`, representa el nombre del producto.
        - `codigo`: de tipo `String`, representa un identificador único entre productos
        - `precio`: de tipo `double`, representa el precio del producto.
        - `cantidad`: de tipo `int`, representa la cantidad en stock del producto.
    - **Constructor:**
        - Crea un constructor que inicialice todos los atributos.
    - **Métodos:**
        - `vender(int cantidadVendida)`: reduce la cantidad en stock según el número de unidades vendidas.
        - `reabastecer(int cantidadReabastecida)`: aumenta la cantidad en stock según el número de unidades reabastecidas.
2. **Clase Principal (Main):**
    - Crea una clase `Main` con el método `main`.
    - Crea instancias de la clase `Producto` para al menos tres productos diferentes.
    - Llama a los métodos `vender()` y `reabastecer()` y muestra los resultados en consola.

Ejercicio 9

Eres un ingeniero automotriz que necesita registrar y analizar el rendimiento de los coches en una carrera. Debes crear un programa que permita gestionar la información de los vehículos que compiten.

Configuración del Entorno de Desarrollo:

1. Crea un nuevo proyecto llamado `CarreraDeCoches`.

Instrucciones:

1. **Creación de la Clase Coche:**
    - Define una clase llamada `Coche`.
    - **Atributos:**
        - `marca`: de tipo `String`, representa la marca del coche.
        - `modelo`: de tipo `String`, representa el modelo del coche.
        - `velocidadMaxima`: de tipo `double`, representa la velocidad máxima en km/h.
    - **Constructor:**
        - Crea un constructor que inicialice todos los atributos.
    - **Métodos:**
        - `acelerar()`: imprime un mensaje que indica que el coche está acelerando.
        - `frenar()`: imprime un mensaje que indica que el coche está frenando.
        - `mostrarInformacion()`: imprime en consola la información del coche, incluyendo marca, modelo y velocidad máxima.
2. **Clase Principal (Main):**
    - Crea una clase `Main` con el método `main`.
    - Crea instancias de la clase `Coche` para al menos tres coches diferentes.
    - Llama a los métodos `acelerar()`, `frenar()` y `mostrarInformacion()` y muestra los resultados en consola.

Ejercicio 10

Eres el encargado de un nuevo zoológico virtual llamado "JavaZoo". Necesitas comenzar a crear una estructura básica para gestionar los animales que habitan en tu zoológico digital.

1. **Configuración del Entorno de Desarrollo:**
    - Crea un nuevo proyecto llamado `JavaZoo`.
2. **Creación de la Clase `Animal`:**
    - **Definición de atributos:**
        - `nombre`: de tipo `String`, representa el nombre del animal.
        - `especie`: de tipo `String`, representa la especie del animal.
        - `edad`: de tipo `int`, representa la edad del animal en años.
    - **Constructor:**
        - Crea un constructor que inicialice todos los atributos de la clase.
    - **Métodos:**
        - `hacerSonido()`: imprime en consola un mensaje que indique el sonido característico del animal. Por ejemplo, "El león ruge".
        - `comer(String comida)`: imprime un mensaje que indique que el animal está comiendo el tipo de comida pasada como parámetro. Por ejemplo, "El elefante está comiendo hierba".
    - **Encapsulamiento:**
        - Asegúrate de que todos los atributos sean `private`.
        - Crea métodos `getter` y `setter` para cada atributo.
3. **Clase Principal (`Main`):**
    - Crea una clase `Main` con el método `main`.
    - Dentro del `main`, crea instancias de la clase `Animal` para al menos tres animales diferentes con datos reales.
    - Llama a los métodos `hacerSonido()` y `comer()` para cada instancia creada.
    - Utiliza los métodos `getter` y `setter` para modificar y obtener información de los animales, mostrando los resultados en consola.

Ejercicio 11

Adivinanza de Números

Crea un juego simple de adivinanza de números. El jugador debe adivinar un número secreto generado aleatoriamente. El programa dará pistas sobre si el número ingresado es mayor o menor que el número secreto y mostrará el número de intentos realizados.

Instrucciones:

Configuración del Entorno de Desarrollo:

1. Crea un nuevo proyecto llamado `AdivinanzaNumeros`.

Instrucciones:

1. **Creación de la Clase Adivinanza:**
    - Define una clase llamada `Adivinanza`.
    - **Atributos:**
        - `numeroSecreto`: de tipo `int`, representa el número secreto que el jugador debe adivinar.
        - `intentos`: de tipo `int`, representa el número de intentos que el jugador ha hecho.
    - **Constructor:**
        - Crea un constructor que inicialice `numeroSecreto` con un valor aleatorio entre 1 y 100.
    - **Métodos:**
        - `adivinar(int numero)`: compara `numero` con `numeroSecreto` y retorna si es correcto, mayor o menor.
        - `reiniciarJuego()`: reinicia el juego generando un nuevo número secreto y reseteando el número de intentos.
2. **Clase Principal (Main):**
    - Crea una clase `Main` con el método `main`.
    - Permite al usuario adivinar el número secreto, dando pistas de si es mayor o menor, y muestra los resultados en consola.

Tips:

1. Para generar números aleatorios en Java, puedes utilizar la función `Math.random()`, que devuelve un número decimal entre 0.0 (inclusive) y 1.0 (exclusive). Para obtener un rango específico, multiplica el valor generado por el rango deseado y convierte a entero.

```java
int numeroSecreto = (int) (Math.random() * 100) + 1;
```

Ejercicio 12

Udemy

En este ejercicio, desarrollarás un sistema básico de gestión de cursos, donde podrás manejar cursos, estudiantes, categorías y calificaciones, simulando una pequeña parte de la plataforma Udemy.

Configuración del Entorno de Desarrollo:

1. Crea un nuevo proyecto llamado `SistemaGestionCursos`.
2. Crea las clases `Curso`, `Estudiante`, `Categoria`, y `Main`.

Tareas

1. Creación de la Clase Categoria

- **Atributos**:
  - `nombre`: de tipo `String`, representa el nombre de la categoría (e.g., "Programación", "Diseño").
- **Constructor**:
  - Crea un constructor que inicialice `nombre`.
- **Métodos**:
  - `mostrarInformacion()`: imprime en consola el nombre de la categoría.

2. Creación del Curso

- **Atributos**:
  - `nombre`: de tipo `String`, representa el nombre del curso (e.g., "Java para Principiantes").
  - `codigo`: de tipo `String`, representa el código único del curso.
  - `capacidad`: de tipo `int`, indica el número máximo de estudiantes que pueden inscribirse.
  - `categoria`: de tipo `Categoria`, representa la categoría a la que pertenece el curso.

    - `estudiantesInscritos`: un `ArrayList<Estudiante>` que contiene los estudiantes inscritos en el curso.
- **Constructor**:
    - Crea un constructor que inicialice `nombre`, `codigo`, `capacidad`, `categoria`, y que inicialice `estudiantesInscritos` como un `ArrayList` vacío.

- **Métodos**:
    - `inscribirEstudiante(Estudiante estudiante)`: añade un estudiante a la lista si hay espacio disponible.
    - `removerEstudiante(Estudiante estudiante)`: elimina un estudiante de la lista.
    - `mostrarInformacion()`: imprime en consola el nombre del curso, código, capacidad, categoría, y la lista de los estudiantes inscritos.
    - `calificarCurso(double calificacion)`: almacena y muestra la calificación promedio del curso.

3. Creación de la Clase Estudiante

- **Atributos**:
    - `nombre`: de tipo `String`, representa el nombre del estudiante.
    - `matricula`: de tipo `String`, representa el número de matrícula del estudiante.
    - `cursosInscritos`: un `ArrayList<Curso>` que contiene los cursos en los que el estudiante está inscrito.
- **Constructor**:
    - Crea un constructor que inicialice `nombre` y `matricula`.
- **Métodos**:
    - `inscribirseEnCurso(Curso curso)`: añade un curso a la lista de cursos inscritos.
    - `mostrarInformacion()`: imprime en consola el nombre del estudiante, su matrícula, y los cursos en los que está inscrito.

4. Clase Principal (Main)

- En la clase `Main`, realiza las siguientes acciones:
    - Crea varias instancias de `Categoria`, como "Programación", "Diseño", etc.
    - Crea varios cursos con diferentes categorías y capacidades.
    - Crea varios estudiantes.
    - Inscribe a los estudiantes en varios cursos y muestra la información del curso antes y después de las inscripciones.

- Permite a los estudiantes calificar los cursos y muestra la calificación promedio.

Ejercicio 13

Sistema de Biblioteca

Eres el administrador de una biblioteca. Necesitas crear un sistema que te permita gestionar tanto los libros como a los miembros de la biblioteca, quienes pueden tomar libros prestados.

1. **Configuración del Entorno de Desarrollo:**
   - Crea un nuevo proyecto llamado `SistemaBiblioteca`.

Tareas

2. **Creación de la Clase Libro:**
   - Define una clase llamada `Libro`.
   - **Atributos:**
     - `titulo`: de tipo `String`, representa el título del libro.
     - `autor`: de tipo `String`, representa el nombre del autor del libro.
     - `disponible`: de tipo `boolean`, indica si el libro está disponible para préstamo.
   - **Constructor:**
     - Crea un constructor que inicialice todos los atributos.
   - **Métodos:**
     - `mostrarInformacion()`: imprime en consola la información del libro, incluyendo título, autor y si está disponible.
     - `prestar()`: marca el libro como no disponible.
     - `devolver()`: marca el libro como disponible.
3. **Creación de la Clase Miembro:**
   - Define una clase llamada `Miembro`.
   - **Atributos:**

        - `nombre`: de tipo `String`, representa el nombre del miembro.
        - `libroPrestado`: de tipo `Libro`, representa el libro que el miembro tiene prestado (puede ser `null` si no tiene ninguno).
    - **Constructor:**
        - Crea un constructor que inicialice el nombre del miembro y deje `libroPrestado` como `null`.
    - **Métodos:**
        - `tomarPrestado(Libro libro)`: si el libro está disponible, lo asigna a `libroPrestado` y llama al método `prestar()` del libro.
        - `devolverLibro()`: marca el libro como disponible y asigna `libroPrestado` como `null`.
        - `mostrarInformacion()`: imprime en consola la información del miembro, incluyendo el nombre y el libro prestado (si tiene alguno).
4. **Clase Principal (Main):**
    - Crea una clase `Main` con el método `main`.
    - Crea instancias de las clases `Libro` y `Miembro`.
    - Haz que un miembro tome prestado un libro y luego lo devuelva, mostrando el estado de los objetos en consola antes y después de cada acción.

Ejercicio 14

GTA V: Ultimate Edition
Imagina que estás desarrollando una versión simplificada de GTA V, donde el jugador puede moverse por una ciudad, interactuar con NPCs (personajes no jugables), y realizar acciones simples como conducir un vehículo y disparar.

Tareas

1. **Crear la Clase `Ciudad`:**
    - Define una clase `Ciudad` que representa el entorno del juego.
    - La ciudad debería contener varios lugares que el jugador pueda visitar (por ejemplo, "Casa", "Tienda", "Parque").
    - Implementa un método para mostrar los lugares disponibles y permitir al jugador moverse entre ellos.
2. **Crear la Clase `Jugador`:**
    - Define una clase `Jugador` que represente al personaje principal.
    - El jugador debería tener atributos como `nombre`, `ubicacionActual`, `vida`, y `dinero`.
    - Implementa métodos para que el jugador pueda moverse entre lugares, ganar dinero, y perder vida.

3. **Crear la Clase `Vehiculo`:**
    - Define una clase `Vehiculo` que represente los vehículos que el jugador puede conducir.
    - Los vehículos deberían tener atributos como `tipo`, `velocidadMaxima`, y `estado` (si está dañado o no).
    - Implementa un método para que el jugador pueda "conducir" el vehículo a diferentes lugares, reduciendo el tiempo de viaje.
4. **Crear la Clase `Arma`:**
    - Define la clase Arma
    - Cada Arma debe tener un nombre, tipo, munición, daño.
    - Implementa el método dispara(NPC)
        - Se debe de poder bajar puntos de salud de un NPC de acuerdo al daño que infringe un arma.
5. **Crear la Clase `NPC`:**
    - Define una clase `NPC` que represente a los personajes no jugables.
    - Cada NPC debería tener un `nombre`, `ubicacion`, `dialogo, y salud`.
    - Implementa un método para que el jugador pueda interactuar con los NPCs (por ejemplo, hablar con ellos para obtener pistas o ganar dinero).
6. **Crear la Clase `Juego`:**
    - Define una clase `Juego` que gestione el flujo del juego.
    - En esta clase, deberás iniciar el juego, crear instancias de la ciudad, el jugador, los vehículos, y los NPCs.
    - Implementa un bucle principal que permita al jugador elegir acciones como moverse por la ciudad, interactuar con NPCs, conducir un vehículo, o salir del juego.
7. **Simulación del Juego:**
    - Implementa un menú en la consola que permita al jugador seleccionar diferentes acciones:
        - "Moverse a otro lugar"
        - "Hablar con un NPC"
        - "Conducir un vehículo"
        - "Ver estado del jugador"
        - “Dispara a un NPC”
        - "Salir del juego"