Ejercicio práctico 1: Compilación y Vinculación

Ejercicio 1.1: Compilación básica en C

Objetivo: Entender el proceso de compilación paso a paso.

Archivos a utilizar: `main.c`, `utils.c`, `utils.h`.

Tareas:

1. Compila los archivos por separado: gcc -c utils.c -o utils.o gcc -c main.c -o main.o
2. Enlaza los archivos objeto: gcc utils.o main.o -o programa
3. Examina `utils.o`: objdump -t utils.o
4. Crea librería estática: ar rcs libutils.a utils.o
5. Enlaza con la librería: gcc main.o -L. -lutils -o programa
6. Ejecuta: ./programa

Ejercicio 1.2: Gestión de dependencias en Rust

Proyecto: Carpeta `calculadora` con `Cargo.toml`, `src/main.rs`.

Tareas:

1. Compilar: cargo build
2. Examinar `Cargo.lock`.
3. Ver árbol de dependencias: cargo tree
4. Crear módulo `src/operaciones.rs` con función `calcular`.
5. Ejecutar: cargo run

Ejercicio 1.3: Sistema de construcción con Java y Maven

Proyecto: Carpeta `calculadora-java` con `pom.xml`, `Calculadora.java`.

Tareas:

1. Compilar: mvn compile
2. Ejecutar: mvn exec:java -Dexec.mainClass="com.ejemplo.Calculadora"
3. Crear tests unitarios.
4. Empaquetar: mvn package
5. Ver árbol de dependencias: mvn dependency:tree

Ejercicio 1.4: Entornos y dependencias en Python

Este ejercicio enseña a manejar entornos virtuales y resolver conflictos entre dependencias.

Archivos

- `app.py`: pequeña aplicación Flask que usa `requests`
- `requirements.txt`: fija versiones con un conflicto intencional

Pasos

[1] Crear un entorno virtual:

```
python -m venv venv
source venv/bin/activate  # En Windows: venv\Scripts\activate
```

[2] Instalar dependencias:

```
pip install -r requirements.txt
```

Esto debe mostrar un conflicto como:

```
ERROR: Cannot install requests==2.28.1 and urllib3==1.26.6 ...
```

[3] Verificar conflictos:

```
pip check
pip list
pip install pipdeptree
pipdeptree
```

[4] Resolver el conflicto:

Opción A: Eliminar `urllib3==1.26.6` de `requirements.txt`.

Opción B: Usar pip-tools:

```
pip install pip-tools
echo "flask==2.2.5\nrequests==2.28.1" > requirements.in
pip-compile requirements.in
pip-sync
```

[5] (Opcional) Guardar dependencias resueltas:

```
pip freeze > resolved-requirements.txt
```

[6] Desactivar entorno:

```
deactivate
```

Ejercicio 2: Gestión de Memoria y Scope

2.1 Scope en diferentes lenguajes

Python – Ejemplo de scope LEGB

Archivo: `python_scope.py`

Se demuestra el comportamiento de los diferentes niveles de scope en Python: global, enclosing, local y cómo funcionan `global` y `nonlocal`.

Java – Scope de clase y método

Archivo: `EjemploScope.java`

Se explora cómo las variables acceden a distintos ámbitos: clase, instancia, método y bloque. También se muestra "shadowing" (ocultamiento de variables).

2.2 Gestión manual de memoria en C

Archivo: `empleado.c`

Se gestiona memoria con `malloc` y `free`, simulando un sistema de empleados. Se cubren errores comunes como olvidar liberar memoria o errores por asignaciones parciales.

2.3 Ownership en Rust

Archivo: `main.rs`

Se muestra cómo funciona el sistema de ownership, borrowing inmutable y mutable, clonación, y cómo Rust impide errores comunes de gestión de memoria en tiempo de compilación.

Ejercicio 3: Programación Orientada a Objetos

Este ejercicio tiene como objetivo ayudarte a comprender herencia, composición y traits mediante una serie de errores que deberás ir solucionando paso a paso.

3.1 Jerarquía de clases en Java

1. Paso 1: Intenta compilar el proyecto. Verás errores de lógica o compilación.
2. Paso 2: Abre el archivo `Automovil.java`. Corrige el constructor para que el valor `consumoPorKm` se almacene correctamente.
3. Paso 3: Ejecuta `EjemploVehiculos.java`. Observa el comportamiento del método `acelerar` en `AutoElectrico`. ¿La batería baja correctamente?
4. Paso 4: Revisa cómo se calcula el consumo y corrige cualquier error.
5. Paso 5: Agrega una nueva clase `MotoElectrica` que también implemente `Electrico`.
6. Paso 6: Usa polimorfismo para probar diferentes tipos de vehículos.

3.2 Composición vs Herencia en Python

1. Paso 1: Ejecuta el archivo `composicion_vs_herencia.py`. Observa errores o salidas incompletas.
2. Paso 2: Corrige la clase `Dog` que está mal definida. Debe heredar de `AnimalHerencia`, no de `ABC` directamente.
3. Paso 3: Completa los métodos faltantes para `Bird` y `Fish` en el esquema de herencia.
4. Paso 4: Asegúrate de que el `Flexible Duck` pueda cambiar de estrategia y muévete con cada una.
5. Paso 5: Crea tu propia estrategia de movimiento y úsala con un nuevo animal.

3.3 Traits en Rust

1. Paso 1: Intenta compilar `traits_composicion.rs` y lee los errores.
2. Paso 2: Implementa el trait `Resizable` para `Circle` y `Rectangle`.
3. Paso 3: Implementa un método `description()` con una implementación por defecto dentro de `Drawable`.
4. Paso 4: Usa funciones genéricas para `resize_and_draw` y `draw_shape`.
5. Paso 5: Agrega nuevas figuras que también implementen `Drawable`.

Instrucciones Generales

- Puedes agregar `System.out.println`, `print()` o `println!()` para inspeccionar resultados.