Universidad de San Carlos de Guatemala
Escuela de Ingeniería en Ciencias y Sistemas, Facultad de Ingeniería
Introducción a la programación y computación 2, 2do. Semestre 2024

# PROYECTO #3 IPC2 DJANGO, S. A

**202200389 – Juan José Rodas Mansilla**

***Resumen***

Este proyecto tiene como objetivo modelar, documentar e implementar una solución al problema de simulación de una máquina ensambladora que funciona con líneas de ensamblaje y brazos robóticos, utilizando herramientas de desarrollo y conceptos de programación enseñados en clase.

Objetivos específicos:

- Implementar la solución usando Python.
- Emplear estructuras de programación secuenciales, cíclicas y condicionales.
- Generar reportes visuales utilizando Graphviz.
- Manipular archivos XML.
- Aplicar el concepto de tipos abstractos de datos (TDA) en memoria dinámica.
- Usar estructuras de programación propias.
- Aplicar el paradigma de programación orientada a objetos (POO).

La empresa **Digital Intelligence, S.A.** ha creado una máquina con "n" líneas de ensamblaje, cada una con un brazo robótico que se desplaza entre "m" componentes. El brazo robótico tarda **1 segundo** para moverse al primer componente, **2 segundos** para el segundo, y así sucesivamente.

- El proceso de ensamblaje sigue instrucciones que especifican la línea de producción y el componente a ensamblar. Aunque los brazos robóticos pueden moverse simultáneamente, el ensamblaje debe seguir una secuencia ordenada. Por ejemplo, si un producto llamado "SmartWatch" necesita ensamblar componentes en el orden **L1C2, L2C1, L2C2, L1C4**, el tiempo óptimo de ensamblaje debe ser calculado.
- Tu tarea es desarrollar un software que simule este proceso para cualquier número de líneas de ensamblaje y componentes, y que sea capaz de predecir el tiempo óptimo de ensamblaje para cualquier producto.

**Palabras clave**

- **Línea de ensamblaje**: Es una secuencia en la que los brazos robóticos seleccionan componentes específicos para ensamblar un producto.
- **Brazo robótico**: Dispositivo que se mueve entre los componentes y los ensambla en el producto. Su movimiento y ensamblaje tienen tiempos asociados.
- **Componente**: Cada una de las partes individuales que forman un producto. Cada línea de ensamblaje tiene múltiples componentes.

- **Ensamblaje secuencial**: Proceso en el que los componentes deben ser ensamblados en un orden específico para construir correctamente el producto.
- **Simulación**: Programa que emula el comportamiento de la máquina ensambladora, calculando el tiempo óptimo de producción para un producto determinado.

For example, if a product called "SmartWatch" requires assembling components in the order **L1C2, L2C1, L2C2, L1C4,** the optimal assembly time must be calculated.

- Your task is to develop software that simulates this process for any number of assembly lines and components, and is capable of predicting the optimal assembly time for any product.

***Abstract***

This project aims to model, document, and implement a solution to simulate the operation of an assembly machine that works with assembly lines and robotic arms, using development tools and programming concepts taught in class.

Specific Objectives:

- Implement the solution using Python.
- Use sequential, cyclic, and conditional programming structures.
- Generate visual reports using Graphviz.
- Manipulate XML files.
- Apply the concept of abstract data types (ADT) in dynamic memory.
- Use custom programming structures.
- Apply object-oriented programming (OOP).

The company **Digital Intelligence, S.A**. has developed a machine with "n" assembly lines, each with a robotic arm that moves between "m" components. The robotic arm takes **1 second** to move to the first component, **2 seconds** to move to the second, and so on. Additionally, the arm uses extra time (x seconds) to assemble a component into the product.

- The assembly process follows instructions specifying the production line and the component to be assembled. Although the robotic arms can move simultaneously, the assembly must follow an ordered sequence.

***Keywords***

- ***Assembly line****: A sequence in which robotic arms select specific components to assemble a product.*
- ***Robotic arm****: A device that moves between components and assembles them into the product. Its movement and assembly have associated times.*
- ***Component****: Each individual part that makes up a product. Each assembly line has multiple components.*
- ***Sequential assembly****: The process where components must be assembled in a specific order to correctly build the product.*
- ***Simulation****: A program that emulates the behavior of the assembly machine, calculating the optimal production time for a given product.*

## Introducción

En este proyecto, estamos desarrollando una aplicación para simular el proceso de ensamblaje de productos en una línea de producción automatizada. La aplicación está diseñada para leer la configuración de la máquina y los productos desde un archivo XML, simular el ensamblaje de los productos y generar un reporte detallado de los movimientos y estados de los brazos de ensamblaje.

### Objetivos del Proyecto

1. **Simulación del Ensamblaje**: Simular el proceso de ensamblaje de productos en una máquina con múltiples líneas de ensamblaje. Cada línea de ensamblaje tiene un brazo que puede moverse y ensamblar componentes.
2. **Optimización del Tiempo**: Asegurar que el ensamblaje se realice en el menor tiempo posible, permitiendo que los brazos se muevan simultáneamente pero ensamblen componentes de manera secuencial.
3. **Generación de Reportes**: Generar un reporte HTML que muestre los movimientos y estados de los brazos de ensamblaje de manera secuencial y detallada.

### Estructura del Proyecto

### Funciones Principales

1. **mostrar_pasos**: Esta función muestra los pasos de ensamblaje para cada línea de ensamblaje.
2. **mostrar_cantidad_datos**: Esta función muestra la cantidad de datos de las líneas de ensamblaje.
3. **simular_ensamblaje()**: Esta función simula el proceso de ensamblaje de un producto en una máquina. Registra los movimientos y estados de los brazos de ensamblaje, asegurando que los brazos se muevan simultáneamente, pero ensamblen componentes de manera secuencial.

### Lógica de Ensamblaje

La función **simular_ensamblaje()** es el núcleo de la simulación. Inicializa las posiciones de los brazos de ensamblaje y simula los movimientos y ensamblajes de los componentes. La lógica asegura que:

- Los brazos pueden moverse simultáneamente.
- Un brazo puede ensamblar un componente solo cuando no hay otros brazos ensamblando.
- Si un brazo está sobre un componente necesario de ensamblar, se mantendrá en espera hasta que sea su turno de ensamblar.

### Generación de Reportes

- La función **generar_reporte_html()** genera un reporte HTML detallado que muestra los movimientos y estados de los brazos de ensamblaje en una tabla. Cada columna de la tabla representa una línea de ensamblaje, y cada fila representa un momento en el tiempo. Los estados "Mover", "En espera" y "Ensamblar" se muestran de manera secuencial para cada brazo.

### Ejecución del Proyecto

Para ejecutar la aplicación, se debe iniciar el servidor Flask y acceder a la interfaz web para cargar el archivo XML y generar el reporte del producto. La interfaz muestra los pasos de elaboración y los movimientos de ensamblaje de manera detallada.

**python app.py**

Luego, abre un navegador web y navega a http://127.0.0.1:5000/ para interactuar con la aplicación.

## *Diagrama de Clases:*

*Clase Movimiento:*

| CATEGORÍA | CATEGORÍA |
|---|---|
| Tiempo | Int |
| Línea | Int |
| Componente | Int |
| Acción | Str |
| Siguiente | Movimiento |

Fuente: elaboración propia

*Clase Posición:*

| CATEGORÍA | CATEGORÍA |
|---|---|
| Línea | Int |
| Componente | Int |
| Siguiente | Posición |

Fuente: elaboración propia

*Clase Brazo Robotico:*

| CATEGORÍA | CATEGORÍA |
|---|---|
| nombre_maquina | Int |
| cantidad_lineas | Int |
| cantidad_componentes | Int |
| tiempo_ensamblaje | Int |
| productos | Lista |

Fuente: elaboración propia

*Clase Producto:*

| CATEGORÍA | CATEGORÍA |
|---|---|
| cnombre_producto | Str |
| pasos | Lista |

Fuente: elaboración propia

*Clase Elaboración:*

| CATEGORÍA | CATEGORÍA |
|---|---|
| Línea | Int |
| Componente | Int |
| Siguiente | Posición |

Fuente: elaboración propia

## Lista Simple

Una **lista simple** es una **estructura de datos** dinámica compuesta por una secuencia de elementos llamados nodos, donde cada nodo contiene dos partes fundamentales:

1. **Dato:** Almacena la información que queremos guardar en la lista (puede ser cualquier tipo de dato, como números, cadenas, objetos, etc.).
2. **Referencia o enlace:** Apunta al siguiente nodo en la secuencia. Es un puntero que indica dónde se encuentra el siguiente nodo en la lista.

La **lista simple** está basada en el concepto de **Tipo Abstracto de Datos (TDA)**, lo que significa que define las operaciones permitidas (como insertar, eliminar, recorrer la lista) sin especificar cómo se implementan en el nivel físico. En una lista simple, la estructura de datos se conecta mediante estos nodos encadenados.

Características principales de una lista simple:

- **Inicio de la lista:** Hay un nodo llamado cabeza o head que marca el comienzo de la lista. Si la lista está vacía, el valor del head es nulo.
- **Secuencialidad:** Los nodos están conectados de forma lineal, uno después de otro. Cada nodo solo conoce la ubicación del siguiente nodo, pero no del anterior.
- **Inserción y eliminación:** Para agregar o quitar elementos, se actualizan las referencias (enlaces) entre los nodos.
- **Memoria dinámica:** A diferencia de un array, que tiene un tamaño fijo, la lista simple puede crecer y reducirse dinámicamente según se necesite, ya que se

reserva memoria en tiempo de ejecución para cada nuevo nodo.

Operaciones comunes:

- **Insertar nodo:** Se puede insertar al inicio, al final o en una posición específica.
- **Eliminar nodo:** Eliminar un nodo requiere actualizar los enlaces para evitar romper la cadena.
- **Recorrer la lista:** Consiste en moverse de un nodo a otro, empezando desde el nodo cabeza hasta llegar al final, cuando el enlace del siguiente nodo sea nulo.

**Conclusiones:**

1. **Eficiencia de las listas simples con nodos:** La implementación de listas simples utilizando nodos ofrece una estructura de datos dinámica que facilita la inserción y eliminación de elementos en cualquier posición de la lista sin necesidad de reorganizar toda la secuencia de datos, a diferencia de los arreglos estáticos. Esto las hace altamente eficientes cuando se requieren operaciones constantes de modificación.
2. **Flexibilidad y memoria dinámica:** Al utilizar memoria dinámica, las listas simples permiten que la estructura crezca y se reduzca según sea necesario, optimizando el uso de recursos en aplicaciones donde el tamaño de los datos no es predecible. Esta flexibilidad reduce el desperdicio de memoria, un punto clave en sistemas de recursos limitados.
3. **Propuesta de mejora:** Si bien las listas simples son eficientes en términos de inserción y eliminación, su principal debilidad radica en el tiempo de acceso a los datos, ya que es necesario recorrer la lista secuencialmente para encontrar un nodo específico. Una posible mejora sería utilizar una lista doblemente enlazada o implementar estructuras más complejas como árboles o listas enlazadas con índices para mejorar el acceso directo a los datos.
4. **Aplicación de TDA en estructuras dinámicas:** La lista simple es un ejemplo claro de la aplicación de los Tipos Abstractos de Datos (TDA), ya que su uso se enfoca en la abstracción de las operaciones sobre los datos, sin preocuparse por su implementación física. Esto es clave en la creación de algoritmos eficientes y modulares que pueden ser adaptados a distintas necesidades sin modificar la estructura interna.
5. **Aportes del análisis:** El análisis y entendimiento de las listas simples con nodos nos ofrece una perspectiva sobre la importancia de elegir correctamente la estructura de datos según el problema a resolver. Aunque su simplicidad es útil en muchos escenarios, es fundamental reconocer cuándo otras estructuras, como listas doblemente enlazadas o árboles, podrían ser más adecuadas para mejorar la eficiencia general en términos de acceso y manipulación de datos.

## Referencias bibliográficas

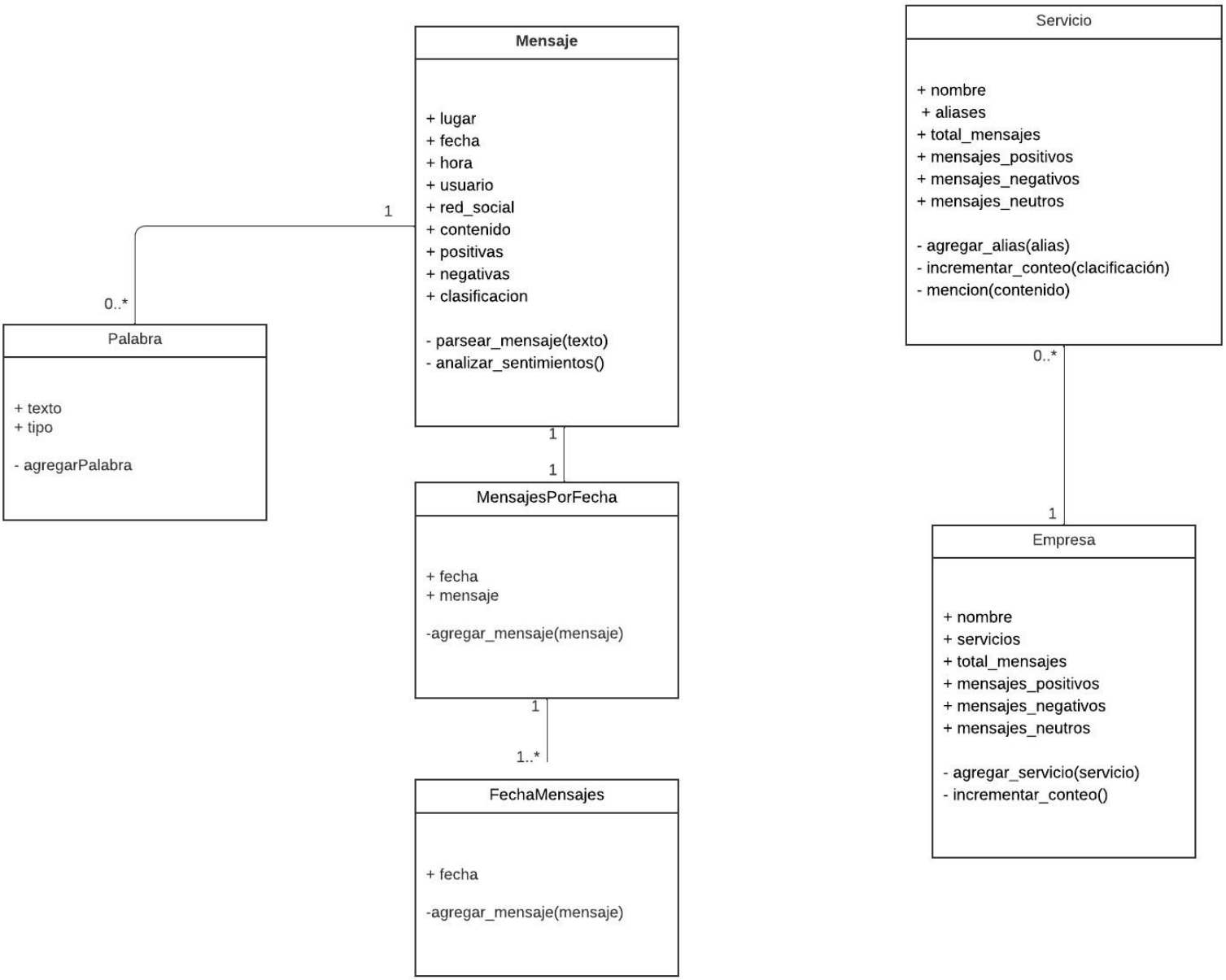
C. J. Date, (1991). *An introduction to Database Systems.* Addison-Wesley Publishing Company, Inc.

Cerrada Somolinos, José y Collado Machuca, Manuel (2015). Fundamentos De Programación. Madrid: Editorial Universitaria Ramón Areces.

Quetglás, Gregorio; Toledo Lobo, Francisco; Cerverón Lleó, Vicente (1995). Fundamentos de informática y programación. Valencia: Editorial V.J.

Universidad de San Carlos de Guatemala
Escuela de Ingeniería en Ciencias y Sistemas, Facultad de Ingeniería
Introducción a la programación y computación 2, 2do. Semestre 2024

# Diagrama de Clases



Universidad de San Carlos de Guatemala
Escuela de Ingeniería en Ciencias y Sistemas, Facultad de Ingeniería
Introducción a la programación y computación 2, 2do. Semestre 2024

# Diagrama de Flujo