

# CLASE 1: Introducción a la Programación Basada en Componentes

## 1. Introducción a la Programación Basada en Componentes

### 1.1 ¿Qué es la programación basada en componentes?

La programación basada en componentes es un paradigma de desarrollo de software que se centra en construir aplicaciones a partir de piezas modulares, llamadas componentes.

**Principios fundamentales:**

- **Reutilización:** Los componentes son diseñados para ser reutilizados en múltiples aplicaciones o contextos.
- **Encapsulamiento:** Cada componente contiene su lógica, datos y, en el caso del frontend, su presentación. Esto facilita su mantenimiento y evita efectos colaterales.
- **Independencia:** Los componentes funcionan de manera autónoma y se comunican a través de interfaces definidas, lo que reduce la dependencia entre ellos.

**Ejemplos en diferentes contextos:**

- **Frontend:** Botones reutilizables en bibliotecas como Material UI, menús de navegación o componentes de formulario.
- **Backend:** Servicios REST que manejan funcionalidades específicas, como un servicio de autenticación o un microservicio de pago.

## Definición de Paradigma

Un **paradigma** es un conjunto de ideas, conceptos, valores y prácticas que definen una manera de entender o abordar un problema o fenómeno en un campo específico. Es, en esencia, una perspectiva o marco teórico que guía cómo se realizan investigaciones o se resuelven problemas en una disciplina.

En el ámbito general:

- **Ejemplo:** En la ciencia, el paradigma heliocéntrico cambió la forma en que se entendía el universo al colocar al Sol como el centro del sistema solar.

## Definición de Paradigma en Programación

Un **paradigma de programación** es un estilo o enfoque para escribir código que se basa en un conjunto de principios, estructuras y técnicas específicas. Los paradigmas de programación guían cómo se diseña y organiza el software, permitiendo a los desarrolladores abordar problemas de diferentes maneras.

### Principales paradigmas en programación:

#### 1. Programación Imperativa:

- Describe *cómo* se debe realizar una tarea mediante un conjunto de instrucciones secuenciales.
- Ejemplo: C, Python, JavaScript (estructurado).
- **Metáfora:** Es como dar un paso a paso detallado para lograr un objetivo.

#### 2. Programación Declarativa:

- Describe *qué* se quiere lograr, sin especificar *cómo* hacerlo.
- Ejemplo: SQL, HTML, lenguajes funcionales como Haskell.
- **Metáfora:** Es como decir el destino sin explicar la ruta.

#### 3. Programación Orientada a Objetos (POO):

- Se basa en la creación de objetos que combinan datos (atributos) y comportamientos (métodos).
- Ejemplo: Java, Python, C++.
- **Metáfora:** Representa entidades del mundo real con sus propiedades y acciones.

#### 4. Programación Funcional:

- Enfocada en funciones puras y en evitar cambios de estado.
- Ejemplo: Haskell, Scala, JavaScript (paradigma funcional).
- **Metáfora:** Es como una transformación matemática: entrada → salida.

#### 5. Programación Basada en Componentes:

- Se centra en desarrollar aplicaciones a partir de piezas modulares (componentes), que son autónomas y reutilizables.
- Ejemplo: React, Angular, microservicios en backend.

Metáfora: Construir un edificio a partir de bloques de construcción prefabricados.

## 4. Definición de Componente y Componente de Software

### 4.1 Definición de Componente

Un componente es una unidad funcional independiente que tiene una responsabilidad específica dentro de un sistema.

**Ejemplo:** Un botón que realiza una acción en una interfaz gráfica:

```
html
```

```
<button>Click me</button>
```

### 4.2 Definición de Componente de Software

Un componente de software es un módulo autónomo que encapsula datos, lógica y/o presentación, diseñado para ser reutilizable en diferentes sistemas.

**Ejemplo:** Un microservicio en backend que gestiona usuarios, con operaciones como registrar, editar y eliminar usuarios.

## Sección 5: Relación con otros paradigmas de programación

Podrías añadir una breve comparación de cómo la programación basada en componentes se relaciona o complementa con otros paradigmas, como:

- **Orientación a objetos:** Cómo los componentes reutilizan conceptos de encapsulamiento, pero amplían su alcance al incluir presentación, datos y lógica.
- **Funcional:** La separación de responsabilidades y la reutilización son también valores clave en la programación funcional.
- **Imperativo vs. Declarativo:** La programación basada en componentes fomenta un enfoque declarativo, especialmente en el frontend.

## 1.2 Importancia en la ingeniería de software moderna

La programación basada en componentes se ha vuelto fundamental en el desarrollo de sistemas modernos, gracias a las ventajas que aporta:

### Ventajas:

- **Mantenibilidad:** Facilita el mantenimiento del software al dividirlo en módulos claros y separados.
- **Escalabilidad:** Permite agregar nuevas funcionalidades sin afectar el sistema existente.
- **Trabajo en equipo:** Los equipos pueden trabajar en diferentes componentes de manera independiente.
- **Reutilización:** Incrementa la eficiencia al aprovechar componentes ya probados en otros proyectos.

### Ejemplos reales:

- **Frontend:** Librerías y frameworks como React, Angular o Vue, que promueven el desarrollo basado en componentes.
- **Backend:** Microservicios, donde cada servicio gestiona una funcionalidad específica y puede ser desarrollado en distintos lenguajes.

### 3. Conceptos Clave: Acoplamiento y Desacoplamiento

#### 3.1 ¿Qué significa que un componente esté "acoplado" o "desacoplado"?

- **Acoplamiento:** Se refiere a la dependencia directa entre partes del sistema.
    - **Ejemplo:** Una función que depende de un tipo de dato específico o de otra función interna.
  - **Desacoplamiento:** Diseñar el sistema de manera que los componentes puedan operar de forma independiente.
    - **Ejemplo:** Un microservicio que expone una API REST, permitiendo ser consumido por cualquier frontend.
- 

#### 3.2 Impacto del acoplamiento/desacoplamiento en el desarrollo de software

##### Ventajas del desacoplamiento:

- **Reutilización:** Los componentes desacoplados son más fáciles de trasladar entre proyectos.
- **Facilidad de mantenimiento:** Las actualizaciones en un componente no afectan al resto del sistema.

##### Problemas del acoplamiento excesivo:

- Dificultad para realizar cambios sin afectar otras partes.
- Problemas para escalar sistemas, ya que las dependencias aumentan la complejidad.

## Sección 8: Buenas prácticas en diseño de componentes

Introduce principios que los estudiantes pueden aplicar desde el inicio:

- **Single Responsibility Principle (SRP):** Un componente debe tener una única responsabilidad.
- **Reutilización:** Diseñar componentes genéricos con propiedades que permitan personalizarlos.
- **Interfaces claras:** Asegurarse de que los componentes se comuniquen mediante interfaces bien definidas.

**Ejemplo de mala práctica:** Un componente que mezcla funcionalidad de lógica de negocio y presentación. Explica por qué este enfoque puede ser problemático.