Constructores

¿Qué es la Instanciación de objetos?

Es el proceso de **crear un objeto a partir de una clase**. En este proceso se reserva memoria para almacenar el estado del objeto y se inicializan sus atributos.

Una clase es como un plano, y la instanciación es el proceso de construir una casa usando ese plano.

Cada lenguaje tiene su propia **sintaxis para instanciar objetos.** En general, se utiliza la palabra reservada **new**.

En los ejemplos de la derecha, se declara una variable para **referenciar la posición de memoria** del objeto y poder interactuar con él.

La **instanciación** se realiza invocando al **constructor de la clase.**

JAVA / C#

```
Persona persona = new Persona("Juan", 25);
```

JAVASCRIPT

```
const persona = new Persona("Juan", 25);
```

PYTHON

```
persona = Persona("Juan", 25)
```

¿Qué es un Constructor?

Es un **método especial** utilizado para inicializar objetos. Se llama **automáticamente cuando se crea una instancia** de una clase.

Propósito de los Constructores

- **Inicialización de objetos:**
  - Configurar el estado inicial de un objeto.
  - Asignar valores a los atributos del objeto.
- **Asegurar la consistencia del objeto:** Evitar que el objeto se inicialice con un estado inválido.

Consistencia de Objetos

La consistencia de un objeto se refiere a que el **objeto mantiene un estado válido y coherente** de acuerdo a las reglas y expectativas del sistema durante su ciclo de vida.

Este concepto asegura que **los atributos de un objeto siempre tengan valores que hagan sentido en el contexto de la aplicación.**

Un objeto es inconsistente cuando **sus atributos están en un estado que no cumple con las reglas** o expectativas establecidas.

> **Por ejemplo:** Sería inconsistente un objeto que representa a una persona y tiene un **atributo nombre vacío ("") o una edad negativa,** ya que estos valores no serían válidos o razonables en la mayoría de los contextos.

Ejemplos de objetos inconsistentes:

java

```java
public class Persona {
    public String nombre;
    public int edad;
}

// Uso de la clase Persona sin validación
public class Main {
    public static void main(String[] args) {
        Persona personaInconsistente1 = new Persona();
        personaInconsistente1.nombre = "";
        personaInconsistente1.edad = -5;
```

javascript

```javascript
let personaInconsistente1 = {
    nombre: "",
    edad: -5
};
```

python

```python
persona_inconsistente1 = Persona()
persona_inconsistente1.nombre = ""
persona_inconsistente1.edad = -5
```

Tipos de Constructores

Constructor por defecto o vacío

- No toma argumentos.
- Es proporcionado automáticamente por el compilador si no se define ningún constructor explícito en una clase.
- Este constructor no inicializa los atributos con valores específicos, simplemente permite la creación del objeto.

Ejemplo en Java:

```
Persona persona = new Persona();
persona.setNombre("Juan");
```

Constructor parametrizado

- Toma argumentos para inicializar los atributos.
- Busca asegurar la consistencia del objeto inicializado.
- Realiza validaciones para asegurar la consistencia.
- Permiten definir dinámicamente la relación entre objetos distintos.

Ejemplo en Java:

```
Persona persona = new Persona("Juan", 25);
```

Teoría

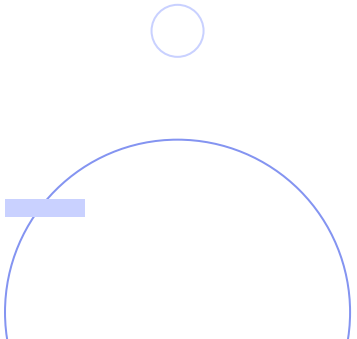
Idealmente, se deberían definir **constructores que inicialicen adecuadamente todos los atributos necesarios para asegurar la consistencia del objeto.**

Práctica

A veces, los *frameworks* y bibliotecas requieren **constructores por defecto** para crear objetos dinámicamente **sin conocer los parámetros requeridos por el constructor** de antemano.

Ejemplos comunes incluyen:

- *Frameworks* de serialización/deserialización.
- Herramientas ORM (Mapeo Objeto - Relacional).
- Librerías de pruebas.
- Otros.



Constructores en JAVA / C#

Constructor por defecto o vacío

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

    void setNombre(String nombre){
       this.nombre = nombre
    }
    ...
}

Persona persona = new Persona();
persona.setNombre("Juan");
```

Constructor parametrizado

```JAVA
class Persona {
    private String nombre;
    private int edad;
    Persona(String nombre, int edad) {
        this.nombre = nombre;
        this.edad = edad;
    }
    ...
}

Persona persona = new Persona("Juan", 25);
```

Constructores en Python

Constructor por defecto o vacío

PYTHON

```python
class Persona:
    pass

# Crear una instancia sin constructor explícito
persona = Persona()

print(persona)
```

Constructor parametrizado

PYTHON

```python
class Persona:
    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad

# Crear una instancia con un constructor parametrizado
persona = Persona("Juan", 25)

print(persona.nombre)  # Output: Juan
print(persona.edad)    # Output: 25
```

Constructores en JavaScript

Constructor por defecto o vacío

```javascript
class Persona {
    // No hay constructor definido explícitamente
}

const persona = new Persona();

console.log(persona);  // Output: Persona {}
```

Constructor parametrizado

```javascript
class Persona {
    constructor(nombre, edad) {
        this.nombre = nombre;
        this.edad = edad;
    }
}

// Crear usando el constructor parametrizado
const persona = new Persona("Juan", 25);

console.log(persona.nombre);  // Output: Juan
console.log(persona.edad);    // Output: 25
```

Sobrecarga de constructores

La sobrecarga de constructores **permite a una clase tener múltiples constructores con diferentes listas de parámetros. Esto** proporciona **diversas formas de instanciar objetos.**

Beneficios:

- Flexibilidad al crear instancias de objetos.
- Permite inicializar objetos con diferentes conjuntos de datos.

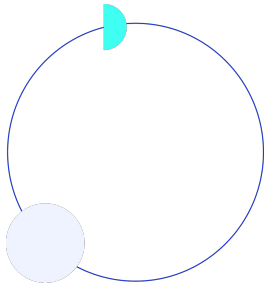
Veamos el ejemplo de la próxima pantalla.

Ejemplo:

```
public class Cuenta {
    ...

    // Constructor con un parámetro
    public Cuenta(String titular) {
        this.titular = titular;
        this.saldo = 0.0;
    }

    // Constructor con dos parámetros
    public Cuenta(String titular, double saldo) {
        this.titular = titular;
        this.saldo = saldo;
    }
}
```



Consideraciones

- **Constructores telescópicos:**
  - Pueden surgir cuando hay muchos parámetros.
  - Pueden complicar el código y dificultar el mantenimiento.
- **Estrategias de construcción avanzadas:** Cuando la construcción de objetos se complica, como en el caso anterior, existen otras técnicas más avanzadas.
- **Necesidad práctica de constructores vacíos:** A veces, son requeridos para la integración con *frameworks* y herramientas que generan objeto, dinámicamente, que requieren constructores vacíos.