Grado en Ingeniería Información

PROGRAMACIÓN II - Sesión 1

Tema 1.

Introducción a la programación orientada a objetos

Curso 2022-2023

Marta N. Gómez

UNIVERSIDAD NEBRIJA

UNIVERSIDAD NEBRIJA

T1. Introducción a la Programación Orientada a Objetos

1.1. Construcción de una clase

1.2. Atributos

1.3. Métodos. Implementación de los métodos de una clase

1.4. Creación de objetos

1.5. Paso de mensajes

1.6. Constructores y destructores

1.7. Punteros a objetos, referencias a objetos

1.8. Constructor copia

ÍNDICE

UNIVERSIDAD NEBRIJA

T1. Introducción a la Programación Orientada a Objetos

1.1. Construcción de una clase

1.2. Atributos

1.3. Métodos. Implementación de los métodos de una clase

1.4. Creación de objetos

1.5. Paso de mensajes

1.6. Constructores y destructores

1.7. Punteros a objetos, referencias a objetos

1.8. Constructor copia

ÍNDICE

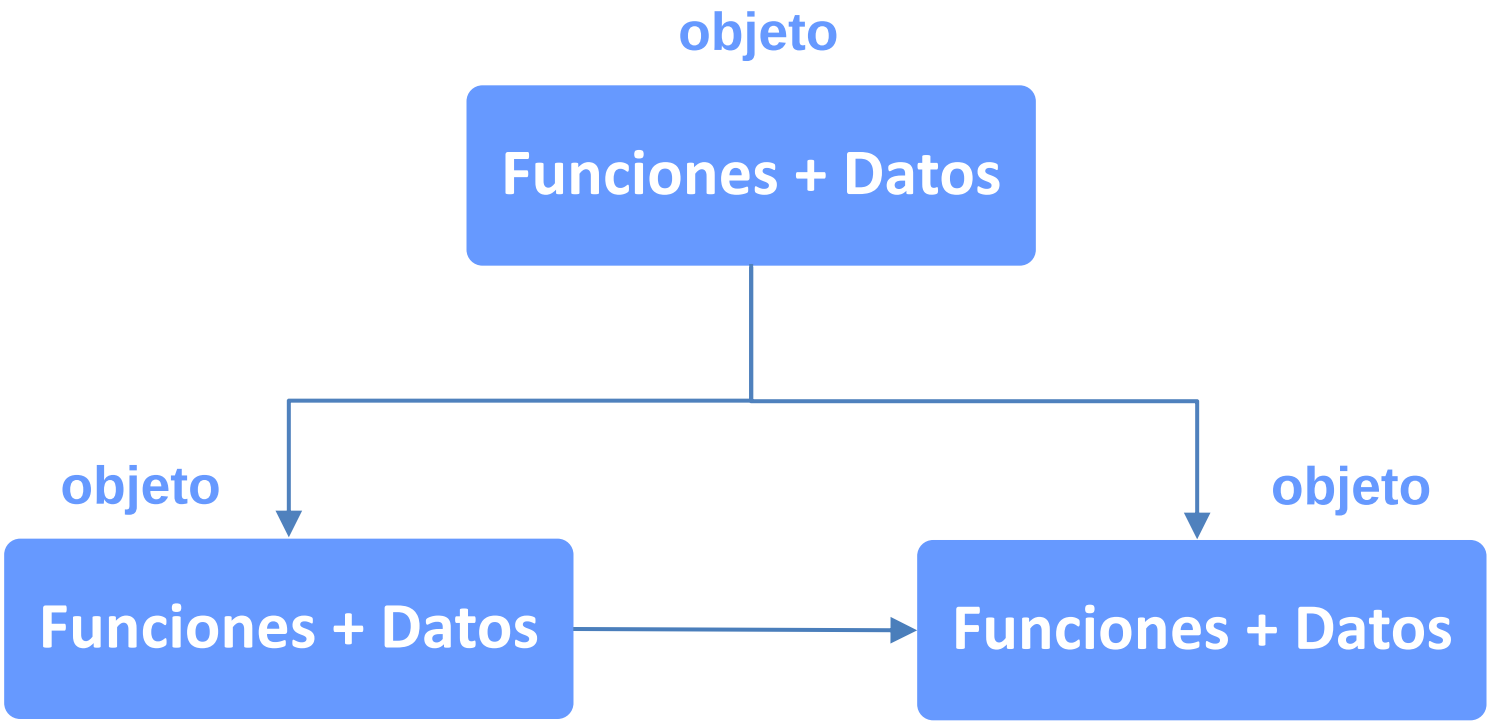
Programación Orientada a Objetos:

El **software se organiza** como una **colección discreta de objetos** a los que se incorporarán:

- una serie de **datos (atributos)** y
- un determinado **comportamiento (operaciones).**

Programación Orientada a Objetos:

- Los **objetos** se comunican intercambiándose **Mensajes.**
- Favorece la **Reutilización de código**



Factores de Calidad:

Factores Internos

- ✓ Aquellos que son percibidos directamente por el **programador**.

Factores Externos

- ✓ aquellos que son percibidos por el **usuario** que interacciona con la aplicación.
- ✓ Los factores externos suelen ser **función directa de los factores internos**.

Corrección: los programas **realizan las tareas** para las que han sido creados, sin cometer **ningún tipo de error**.

Se ha desarrollado una aplicación para hacer estadísticas a partir de un conjunto de datos.

¿Proporciona esta aplicación todos los test numéricos y pruebas estadísticas que se definieron en los requisitos iniciales?

Robustez o Fiabilidad: asegurar la ejecución sin cometer **errores y de forma consistente (resultados reproducibles).** Debe responder adecuadamente ante **datos erróneos**.

1.- Cada vez que calculamos un test estadístico determinado, con los mismos datos:

¿Obtenemos el mismo resultado?

2.- Si proporcionamos número reales a un test que solo funciona con números enteros:

¿La aplicación lo detecta y advierte al usuario?

Eficiencia: consiste en tener **los recursos**, tanto software como hardware, que **la aplicación necesita** para llevar a cabo las operaciones que implementa.

Los **tiempos de respuesta** de la aplicación serán **adecuados a las necesidades del usuario**.

¿Podemos hacer ejecuciones de todos los tests estadísticos que ofrece la aplicación en un PC doméstico o necesitamos una estación de trabajo con doble procesador?

¿Puede obtenerse el resultado de un determinado test en menos de 30 segundos?

Integridad: tener niveles de **seguridad** adecuados para **controlar el acceso al software y datos**, evitando modificaciones no deseadas en ellos.

Para los tests estadísticos a partir de datos clínicos de pacientes:

¿Controla la aplicación el acceso a estos datos para que solo los usuarios que sean médicos puedan utilizarlos?

Facilidad de uso: medir el **tiempo de aprendizaje del usuario y el esfuerzo** que este necesita para obtener los resultados del software. El software debe tener **una interfaz de usuario** que permita a este operar con él de forma **cómoda e intuitiva**.

¿Puede un nuevo usuario utilizar el programa solo leyendo un pequeño manual?

¿Será esto suficiente para que aproveche todas sus funcionalidades?

¿Es cómodo importar datos numéricos desde un fichero de texto para analizarlos o es un proceso tedioso y lento?

Facilidad de mantenimiento: facilidad que dispone el sistema para **detectar** y **corregir errores**. Este factor está muy relacionado con la **legibilidad del código**.

Un test numérico no da el resultado esperado:

¿Podemos localizar el error?

¿Cuánto tiempo le costará al equipo de desarrolladores arreglarlo?

Flexibidad: es el esfuerzo necesario para modificar el software. El sistema software debe poder **adaptarse a los cambios de las especificaciones iniciales** del usuario a lo largo del tiempo para dar respuesta a las nuevas necesidades.

Hay un nuevo test estadístico que los usuarios necesitan:

¿Puede implementarse como una funcionalidad adicional?

¿Cuánto esfuerzo supondrá?

Portabilidad: determinar el **coste** necesario para **migrar** la aplicación a otro **entorno diferente**.

El **entorno** puede referirse tanto al hardware sobre el que se ejecuta la aplicación como al software asociado.

¿Funcionará la aplicación en otra máquina más actual si es necesario sustituir el hardware sobre el que se desarrolló?

¿Es la aplicación compatible con la nueva versión del sistema operativo?

¿Y con otro sistema operativo?

¿Se podría usar desde un teléfono móvil?

Reusabilidad: medida en la que determinadas **partes de la aplicación** pueden **reutilizarse para otras tareas**.

La **reutilización de código** está directamente relacionado con la **generalización de determinadas tareas y el uso de bibliotecas de funciones**.

¿Existe en la aplicación una única rutina que calcula el valor medio de una serie de datos y se utiliza en varios test estadísticos o cada test implementa la suya?

De existir dicha rutina, **¿se podrá reutilizar para otro desarrollo software?**

Compatibilidad o Interoperabilidad: estima la facilidad de la aplicación para **comunicarse y trabajar con otras aplicaciones de forma fluida y eficiente**.

Un elemento de desarrollo software es compatible cuando es capaz de **intercambiar información en forma de datos y control con otros elementos del sistema**.

¿Puedo usar los resultados de esta aplicación como datos de entrada para otra?

¿Puede la aplicación exportar e importar datos en formatos estándares?

Ventajas de la Programación Orientada a Objetos

- ✓ **Corrección:** El modelado del dominio del problema en base a los objetos que lo componen permite al equipo de desarrolladores ajustarse de forma más directa a la especificación de requisitos del cliente.
- ✓ **Fiabilidad:** Una mejor organización del código escrito por el equipo de desarrollo facilitará la corrección de este y, por tanto, incrementará la fiabilidad de la aplicación.

Ventajas de la Programación Orientada a Objetos

- ✓ **Eficiencia:** El uso de librerías externas, en forma de clases específicas diseñadas a tal efecto, puede suponer una mejora significativa en los tiempos de ejecución de determinadas operaciones.
- ✓ **Integridad:** La ocultación de información mediante el encapsulamiento de datos en las clases facilita la construcción de software seguro, evitando el acceso a dichos datos por código de otras partes del programa.

Ventajas de la Programación Orientada a Objetos

- ✓ **Facilidad de uso:** El uso de librerías de clases que implementan elementos gráficos de interfaz permite al equipo de desarrolladores diseñar e implementar interfaces de usuario de forma más sencilla.
- ✓ **Facilidad de mantenimiento:** El modelado del problema a través de jerarquías de clases, así como la encapsulación de datos y los sistemas de paso de mensajes entre objetos, hacen posible una organización del código cercana al dominio del problema y, por tanto, más intuitiva y legible.

Ventajas de la Programación Orientada a Objetos

- ✓ **Flexibilidad:** Gracias a la abstracción, la herencia y los polimorfismos, las clases pueden adaptarse a nuevas necesidades de una forma eficiente y segura.
- ✓ **Portabilidad:** el uso de clases y el encapsulamiento de datos ayudan a separar las especificaciones formales de la implementación, lo que facilita la migración de la aplicación entre diferentes plataformas..

Ventajas de la Programación Orientada a Objetos

- ✓ **Reusabilidad:** La abstracción, la ocultación de información y la definición de interfaces públicos para el paso de mensajes permitirán al equipo de desarrolladores implementar clases generales que puedan reutilizarse en diversas partes de la aplicación e incluso en otros proyectos.
- ✓ **Interoperabilidad:** la ocultación de información y la definición de interfaces públicas claras, facilitará la interacción de la aplicación o alguna de sus partes con otros programas.

Características de la Programación Orientada a Objetos

Abstracción:

La **abstracción** es la representación de las características esenciales de algo sin incluir antecedentes o detalles irrelevantes.

La **abstracción** consiste en la **generalización** conceptual de los **atributos** y **métodos** de un determinado **conjunto de objetos** y almacenarlos dentro de una **clase**.

La **abstracción** permite tener una **visión global** del problema a resolver sin necesidad de saber **cómo se implementará realmente la solución**.

Características de la Programación Orientada a Objetos

Encapsulamiento:

La **encapsulación u ocultación de la información** consiste en incluir dentro de un objeto todo lo que necesita, de modo que ningún otro objeto necesite conocer su estructura interna para hacer uso de él, **sólo será visible su interfaz**. Los **objetos** son **autosuficientes** e **independientes**.

El **objeto oculta sus datos** a los demás objetos, permitiendo **el acceso** a ellos mediante sus **propios métodos**.

La **encapsulación** permite **controlar el acceso** a los atributos y métodos de un objeto, decidiendo qué parte de su estado y comportamiento es público para otros objetos y qué parte no lo es.

Características de la Programación Orientada a Objetos

Herencia

La **herencia** es el mecanismo por el cual se puede **crear una clase a partir de otra** de manera que esta **nueva clase adquiera los métodos y atributos de la anterior**.

Este mecanismo permite ***reutilizar*** la definición de una clase ya existente en la aplicación para crear una nueva clase a partir de ella. La **nueva clase** puede incluir **atributos** y **métodos más específicos**.

La herencia implica, normalmente, una ***especialización***.

Superclase es la **clase más general**.

Subclase es la **clase más especializada**.

La **herencia** permitirá establecer **relaciones jerárquicas** entre las clases de la aplicación.

Características de la Programación Orientada a Objetos

Polimorfismo:

El **polimorfismo** es la propiedad por la que una **operación** o **método** se comporta de **forma distinta** en **diferentes clases**.

El **polimorfismo** es la capacidad de que un **mensaje** sea interpretado de **maneras distintas** según el **objeto** que lo recibe.

El objeto receptor del mensaje es quién determina el método que se tiene que ejecutar.

Por tanto, hay que disponer de un criterio adicional que permita distinguir entre las posibles interpretaciones del método a ejecutar.