

Teoria-de-P00.pdf



lucyzal05





Programación Orientada a Objetos



2º Grado en Ingeniería Informática



Escuela Politécnica Superior de Córdoba
Universidad de Córdoba

 **chromebookplus** | con  Gemini

¿Necesitas un resumen?

Resúmenes claros en segundos y mucho más con tu portátil Chromebook y la IA de Google.





Definiciones

Objeto

Es una entidad con datos almacenados y tareas realizables durante un tiempo de ejecución.

Un objeto puede ser creado instanciando una clase o mediante escritura directa de código y la replicación de otros objetos.

Clase

Es una plantilla que tiene como finalidad u objetivo la creación de objetos. Se utilizan para representar entidades o conceptos

Instancia

Es un miembro de una clase con atributos en lugar de variables.

Atributo

Es una especificación que define la propiedad de un objeto, elemento o archivo.

Método/función

Es una subrutina cuyo código es definido en una clase. Si pertenece a una clase se le denomina métodos estáticos, si pertenece a un objeto se le denomina métodos de instancia.

Subrutina

Es un subalgoritmo del algoritmo principal que permite resolver una tarea específica.

Subalgoritmo

Es un método de programación que sirve para dividir las partes de un algoritmo y resolver cada una de ellas particularmente.

Diagrama de Clases

Es un tipo de diagrama con estructura estática que describe la estructura de un sistema mostrando sus clases, atributos, funciones, y las relaciones entre los objetos.

namespace

Es "un contenedor abstracto" en el que un grupo de uno o más identificadores únicos pueden existir.

funciones inline

Son funciones insertadas directamente en la sección del código donde se les llama.

Constructor

Es una subrutina cuya misión es inicializar un objeto de una clase.

Destructor

- Liberar los recursos computacionales que el objeto de dicha clase haya adquirido en su tiempo de ejecución, una vez ha terminado.

- Quitar los vínculos que pudiesen tener los recursos u objetos usados.

Referencia

Es un indicativo hacia un objeto, es decir, es una variable cualitativa que contiene la dirección de otra variable cualitativa.

Otros conceptos

Iniciadores de Miembros:

Una clase puede contener otros objetos como miembros de ella. Si estos objetos necesitan ser inicializados, ésto debe hacerse con iniciadores de miembros.

Constructor de copia:

`MiClase(const MiClase&)=default; //Comportamiento por defecto, hace lo mismo que Constructor.`

`MiClase(const MiClase &e){...} //tiene parámetros y realiza asignaciones`

Pair

Es una plantilla de clase que proporciona una forma de almacenar dos objetos distintos como una sola unidad.

Tuple

Es una colección de tamaño fijo de valores distintos.

Variables Miembro Estáticas o Static Members

Cuando se usan, solo una copia del miembro estático es compartida por todos los objetos de una clase en un programa.

Funciones miembro static

Al declarar un miembro como estático lo haces independiente de cualquier objeto de la clase. A una función miembro estática se le puede llamar incluso si no existen objetos de la clase.

Polimorfismo

Proceso mediante el cual se puede acceder a diferentes implementaciones de una función utilizando el mismo nombre.

Funciones Virtuales:

Las funciones virtuales se usan cuando algunas de las funciones que forman parte de una interfaz compartida no pueden concretar las clases derivadas que no están totalmente definidas.

- Definen la interfaz genérica.
- No se pueden definir objetos de estas clases.
- Son usadas para derivar de ellas y para el polimorfismo.

Punteros Inteligentes o Smart Pointers:

Es un tipo abstracto de dato que simula el comportamiento de un puntero. Gestiona automáticamente la memoria mediante un registro y comprueba los límites para reducir errores causados por el mal uso de punteros, manteniendo la eficiencia

Tratamiento de señales (Excepciones)

Dentro de un bloque try se pretende evaluar una o más expresiones. Si dentro de dicho bloque se produce algo que inesperado, se lanza por medio de throw una excepción, la misma que será capturada por un catch específico.

-Puede haber más de un catch y throw.

-Si hay un throw sin su catch correspondiente, el programa se interrumpirá abruptamente después de haber pasado por todos los catches que se hayan definido y de no haber encontrado el adecuado.

-Tipos de excepciones: int, float, char, etc.

Punteros Inteligentes o Smart Pointers:

Es un tipo abstracto de dato que simula el comportamiento de un puntero. Gestiona automáticamente la memoria mediante un registro y comprueba los límites para reducir errores causados por el mal uso de punteros, manteniendo la eficiencia.

Formas de insertar elementos en un vector:

`vector_.push_back(dato);` [este si se ha usado más]

`vector_.insert(vector_.begin(), dato);`

`vector_.insert(vector_.end(), dato);`

`vector_.insert(vector_.begin()+vector.size(), dato);`

enum class o enumeración:

Ej: enum class name[

 name1,

 name2

];

y luego en private: se puede crear una variable de tipo name llamada name3_;

y en public: se pueden crear funciones de tipo name funcion;

en el .cc: name3 = name::name1;

Prácticas

map:

Permite asociar etiquetas a valores (cada etiqueta tiene un único valor asociado). `map<v1,v2> n;`

Herencia, iniciadores base:

Si el constructor de la clase base tiene un parámetro obligatorio, éste debe pasarse en un iniciador base de su clase original.



¿Necesitas un resumen? Resúmenes claros en segundos y mucho más con tu portátil Chromebook y la IA de Google.

Puntero this:

Es un parámetro implícito a todas las funciones miembro de una clase, es decir, todas las funciones miembro de una clase lo reciben automáticamente a excepción, de las funciones estáticas.

Funciones friend:

Es aquella que sin ser una función miembro de la clase puede acceder a su parte privada.

Sección protected:

-El desarrollador/a de la clase derivada puede acceder a la parte protected de la clase base.

-El desarrollador/a de la clase base permite que las clases derivadas accedan a la sección protected.

Diferencia entre derivar y heredar:

Heredar: usar parámetros de la clase de la que se hereda.

Derivar: usar funciones de la clase de la que se deriva.

Algorithm Library:

Define funciones para una serie de finalidades (la ordenación o sorting, la búsqueda, el recuento o counting) que operan con rangos de elementos.

Tema 1: Abstracción y diseño

Abstraction o Abstracción:

Consiste en facilitar la comprensión de un sistema complejo añadiendo una capa de abstracción sobre cada módulo software.

Pasos para hacerlo:

1. Analizar un problema.
2. Identificar cada componente a nivel abstracto y sus funciones.
3. Diseñar una solución con dichos componentes.
4. Implementación.

Diseño:

-Proceso: elaborar un plan de lo que se quiere hacer.

-Resultado: software de calidad.

TAD (Tipo Abstracto de Dato):

Es un modelo matemático compuesto por una serie de operaciones definidas sobre un conjunto de datos.

Durante la implementación se definirá cómo se realizan esas operaciones y dará lugar a las estructuras de datos.

Encapsulamiento:

Impide acceso directo al estado interno de un objeto:

- Impidiendo operaciones no permitidas.
- Simplificando su comprensión.
- Sin afectar a ningún programa que use dicho objeto ya que nunca se accede a ellos.
- Solo podrá interactuarse con el objeto mediante su interfaz pública.

Tema 2: Software de Calidad

Software de Calidad

Hay 2 tipos:

La calidad funcional tiene que ver con el grado de acercamiento del funcionamiento del software con sus especificaciones y requisitos funcionales. (Externa)

La calidad estructural tiene que ver con el cumplimiento de características internas no funcionales del software que dan soporte características internas del software también muy deseables como son la mantenibilidad, la facilidad de extensión, la reutilización, etc. (interna)

Factores de Calidad

[esquema]

Tema 3: Descomposición modular

[esquema]



¿Necesitas un resumen?



¿Puedes resumir este artículo?

Desentrañando los secretos de los anillos de Saturno

Saturno, el sexto planeta desde nuestro sol, es reconocible al instante gracias a sus magníficos anillos. Estas maravillas heladas han cautivado a astrónomos y aficionados a la astronomía durante siglos, pero también encierran muchos misterios. Si bien hemos aprendido mucho sobre

anillos podrían parecer sólidos, como un disco gigante que rodea el planeta. Sin embargo, una mirada más de cerca revela una realidad mucho más intrincada. En realidad, los anillos están compuestos por innumerables partículas individuales, cuyo tamaño varía desde diminutos granos de hielo hasta rocas masivas. Estas partículas son en su mayoría...

en colisión sobre el tiempo, lo que estiman que por completo dentro de 100 millones

Ayúdame a leer



Desentrañando los secretos de los anillos de Saturno

Los anillos están compuestos en realidad por incontables partículas individuales, que varían en tamaño desde diminutos granos de hielo hasta enormes rocas. Estas partículas son...

Hacer una pregunta

Resúmenes claros en segundos y mucho más con tu portátil Chromebook y la IA de Google.

Tema 4: TDD vs OOD

TDD vs OOD:

- Descomposición TDD (Top-Down Desing): Descomposición de arriba a abajo (descomposición funcional).

- Descomposición ODD (Object Oriented Desing): Descomposición orientada a objetos.

Especificación e Implementación de TAD:

TAD: es una tabla que gestiona un conjunto de enteros y sus estadísticas. Esta realiza una serie de operaciones como: añade, total, infonum e infofrec.

La especificación informa de manera precisa los detalles relevantes del TAD mediante una serie de operaciones como: concat, deleteDuplicate, binarySearch y factorial.

-Requisitos de la implementación: Tiene que ser pequeña, cerrada o abierta, y tiene que seguir una serie de criterios, reglas y principios de descomposición modular.

Tema 5: Reutilización

Reutilización del Software:

La reutilización tiene 2 aspectos:

-Producción (creación, autoría): desarrollar componentes consumiendo y aprendiendo.

-Consumo: estudia, aprende e imita: El estilo, el diseño, las propiedades de un componente...

Se reutiliza:

-Metodologías de Programación: POO: objetos/clases, herencia, polimorfismo, templates, etc.

-Diseños: los patrones de diseños.

Cómo se reutiliza: Realizando mejoras constantes en estructuras de reutilización: lenguajes de programación, repositorios, comunidades de desarrolladores, etc.

La repetición: detectarla, entenderla y describirla bien para desarrollarla como módulo.

Módulo ideal para la reutilización, es aquel que:

- Es clara, concreta y bien descrita.
- Tiene facilidad de uso, con el menor tamaño posible y la mayor calidad.
- Cumple los criterios, reglas y principios de la descomposición modular.

Obstáculos: falta de formación, fallos en el diseño y gestión, y desconfianza de reutilización fuera de tu propio equipo de desarrollo.

Estructuras de reutilización: rutinas, bibliotecas, librerías, etc.

-Agrupaciones de unidades: se adaptan mediante argumentos y resuelven problemas concretos y complejos de reducido tamaño.

-Clases: son piezas modulares reutilizables.

-Paquete: es similar a una rutina pero más avanzada.

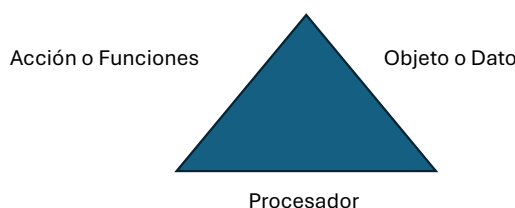
-Framework/entorno de desarrollo: facilitan el desarrollo de cualquier aplicación, pueden incorporar elementos estructurales de diseño de aplicaciones y especializarse en plataformas.



Tema 6: Los pilares de la tecnología OO

·Triángulo básico de la computación, las tres fuerzas de la computación:

La visión se orienta a los objetos y el enfoque tradicional es la acción.



Hay 2 enfoques:

Enfoque tradicional: modularidad basada en la acción.

Se basa en funciones, toma decisiones de diseño prematuras y estratégicamente poco dispuestas a características de calidad deseables, le da mucha importancia a la interfaz y es poco adecuado para sistemas de medio/gran tamaño.

Enfoque Orientado a Objetos: se basa en objetos.

·Los módulos se deducen de los objetos que se manipulan (no hay diseño prematuro y la interfaz es una función más), presenta un diseño pensando en la reutilización, la extensibilidad y el mantenimiento, más simple y adecuado para sistemas grandes.

Abstracción: Clases y objetos.

·Acceso a una clase: se puede acceder mediante el uso de métodos, la interfaz, las funciones(proporcionan + eficiencia) y los atributos(ligados a la representación interna y su acceso dificulta cambios).

·Para el cliente, los atributos y las funciones deben diferenciarse lo mínimo posible.

Encapsulamiento:

Ocultar la implementación y la visión interna con un acceso mediante determinados métodos.

Impide el acceso directo al estado interno de un objeto, y hace que solo pueda interactuarse con el objeto mediante su interfaz pública.

Herencia:

Forma una estructura de reutilización jerárquica específica de la POO, facilita la reutilización y la extensibilidad, es útil en la descomposición modular, y forma una clase base y una derivada.

Polimorfismo: existe cuando se usa la misma interfaz para objetos de distinto tipo.

·Estático: Es el + simple y define una función/operador diferente para cada tipo.

·Estático paramétrico: La interfaz es independiente de la instancia y usa plantillas de clase y de función.

·Dinámico: Permite construir código genérico independiente del tipo y usa punteros a la clase base y funciones virtuales.

Tema 7: Patrones de Diseño

Definición: Es una solución para la resolución de problemas de diseño de software.

Contenidos: Está compuesto por objetos, clases y las relaciones entre estos y cada uno está especializado en resolver un problema de diseño concreto en un determinado contexto.

Utilidad: Ayuda a la resolución de los problemas de diseño para que nuestros diseños serán: + sencillos, - costosos y de mayor calidad.

Tipos:

-Patrones de Comportamiento (C.): objetos, interacciones, algoritmos, etc.

-Patrones Estructurales (S.): componen estructuras de clases entre varios objetos distintos.

-Patrones de Creación (C.): crean instancias de objetos.

Elementos esenciales de un Patrón de Diseño (PD):

Tipo: B.S.C.

Estructura: Conjunto de clases/objetos que cooperan.

Función: Identificar cuando y bajo que circunstancias hay aplicar el patrón.

Template Method: "Separa las cosas que cambian de aquellas que permanecen igual."

Tipo: S.

Función: Describe un comportamiento genérico para concretarlo en las clases derivadas mediante el uso de placeholders

Ventajas: modificabilidad.

Parameterized types o Generics desing pattern:

Tipo: S.

Función: Define un nuevo tipo sin tener que especificar los tipos de todos sus componentes.

Ventajas: Genericidad.

Iterator (it):

Proporcionan una manera de acceder a los elementos de un objeto secuencial.

Se puede aplicar en listas, vectores, etc.

Ventajas: facilidad de uso, favorece la abstracción, independencia de la representación interna y acceso uniforme.

Observer:

-Estructura: un objeto(sujeto) y varios suscriptores que dependen de unos datos (el sujeto). Este patrón de suscripción permite la actualización continua de los datos.

-Esta estructura de tipo B se define por el par: sujeto-observador.

-El sujeto contiene todos los datos que se publican y registra a los observadores (suscriptores) en su interfaz. Puede haber varios observadores del mismo sujeto.

-El sujeto y el observador son independientes.

Las vistas son un componente del MVC.

Composite:

-Características: similitud entre objetos de distintos niveles en el sistema y una jerarquía de tareas.

-Función: construir cosas a partir de otras más pequeñas pero similares. En la POO consiste en coger objetos pequeños para construir otros + grandes y complejos, se agrupan componentes para construir supercomponentes. En este patrón de tipo S, se tratan de la misma manera a los objetos individuales y a los compuestos.

-Ventajas: simplificación, interfaz más sencilla, y el acceso uniforme.

Strategy:

Es un patrón de tipo B que permite que se intercambie un algoritmo por otro dependiendo del problema en cuestión.

Se usa cuando se prevee que algo tendrá distintos comportamientos en el futuro o cuando hay estructuras de datos complejas que podrían implementarse de distintas formas en el futuro.

-Ventajas: posibilidad de mejorar la eficiencia y facilita las ampliaciones.

Strategy vs Template Method:

Strategy: permite que un objeto cambie su comportamiento en tiempo de ejecución, intercambiando el algoritmo.

Template: permite que una función se desarrolle de formas distintas, mediante el uso de placeholders.

MVC:

Es un grupo de tres elementos vinculados entre sí, de tipo S.

Está presente en la mayoría de los frameworks de desarrollo modernos y se puede aplicar en cualquier aplicación.

-Ventajas: simplificación del desarrollo, flexibilidad, desacoplación de funciones, y reparto de roles (frontend, backend).

Builder:

Es un patrón de tipo C, que facilita la creación y configuración de objetos complejos.

Se usa cuando el proceso de construcción y configuración de un objeto es complejo

-Ventaja: facilita la creación de objetos complejos.

Factory:

Es un patrón de tipo C, que facilita la creación de instancias de diferentes clases que pertenecen a la misma familia.

En este existen 2 implementaciones:

1) Mediante el "Factory Method", que recibe una descripción del objeto deseado y devuelve un puntero de la clase base que apunta a ese objeto.

2) Mediante la "Abstract Factory", que define la interfaz de cada sub-factory.

-Ventajas: facilita la manipulación de objetos pertenecientes a una misma familia y la ampliación de clases de la misma familia.