PEL

RESUMEN UF1

15/10/2021

Marcos Eladio Somoza

U1

Abstracción: Técnica por la cual se reinventan nuevos tipos de datos más adecuados para una aplicación. De esta forma se facilita la estructura de los programas. Hay:

- **Abstracción de datos**
- **Abstracción de control**
    - **Estructuras de control**: Describen el orden en el que se pueden ejecutar los conjuntos de sentencias.
    - **Unidades de programas**: Bloques básicos de programación de la clásica composición descendente.
- **Abstracción procedimental**: Se basa en emplear procedimientos y funciones sin preocuparse de cómo se implementan.

Subprogramas: Constituyen una herramienta potente de abstracción, pues al implementarse, el programador describe en detalle su funcionamiento. En el momento en el que son llamados, saben exactamente qué hacer. Suelen usarse a través de colecciones de subprogramas en las bibliotecas.

Tipos de datos: Propiedad de un valor que determina su dominio (los valores que va a tomar, operaciones que se les puede aplicar, etc). Todos los valores que aparecen en un programa tienen un tipo.

Tipo abstracto de dato: En caso de que los tipos de datos no sean definidos por sus propiedades, sino que es el programador el que los define, se tratan de tipos abstractos de datos.

-Ventajas:

1. Permiten una mejor conceptualización y modelado del mundo real.
2. Mejora la robustez del sistema.
3. Mejora el rendimiento.
4. Separa la implementación de la especificación.
5. Permite la extensibilidad del sistema.
   Recoge mejor la semántica del tipo.

Programa modular: Programas que usan la noción de Tipo Abstracto de Dato (TAD), siempre que sea posible.

TAD = Representación (datos) + Operaciones (funciones y procedimientos)

U2

Propiedades POO

Abstracción: En POO, la abstracción se refiere a diferenciar entre la composición interna y las propiedades externas de una entidad. Cada componente de la entidad representa un nivel de abstracción en el cual se aíslan los detalles de la composición interna, generando lo que se conoce como diferentes niveles de abstracción.

Encapsulamiento y Ocultación de datos: El encapsulamiento es un proceso por el cual se agrupan datos y operaciones relacionadas bajo la misma unidad de programación. Esto permite separar el aspecto de un componente de sus detalles internos de implementación. Generalmente se usa la ocultación de datos de la información y encapsulamiento como sinónimos, pero no lo son.

-Mínimos para diseñar un programa poo:

- Identificar objetos.
- Agrupar objetos en clases si tienen características y comportamientos similares.
- Identificar los datos y operaciones de cada una de las clases.
- Identificar las relaciones que pueden existir entre clases.

Herencia: La generalización se conoce como la propiedad que permite compartir la información entre dos entidades para evitar la redundancia. La especialización es el proceso inverso a la generalización. Se crean nuevas clases a partir de otras ya existentes. La herencia, es un mecanismo de programación orientada a objetos que implementa las propiedades de generalización y especialización. Esta, permite definir nuevas clases a partir de otras, de forma que presenten las mismas características y comportamientos. En C ++, la clase “original” se denomina clase base, y las clases que derivan de una clase base, se conocen como clases derivadas. Las clases derivadas son siempre especializaciones de las clases base.

Polimorfismos: Gracias a la herencia, en la programación orientada a objetos, se facilita el polimorfismo. Esta, es la propiedad por la cual un operador de una función puede actuar de modo diferente en función del objeto sobre el que se aplique. Las operaciones tienen esta propiedad cuando tienen el mismo nombre en diferentes clases, pero en cada clase se ejecuta de forma diferente. Usar operadores o funciones de forma diferente, dependiendo de los objetos sobre los que se actúa, también es considerado polimorfismo. La sobrecarga es un tipo de polimorfismo.

Reusabilidad

En el momento en el que una clase ha sido escrita, creada y depurada, el desarrollador puede distribuir dicha clase a otros programadores para que dicha clase sea usada en otros programas. Dicha funcionalidad es considerada la reusabilidad, y su concepto es similar a las funciones incluidas en las bibliotecas de funciones de lenguajes procedimentales.

Objeto

•**Desde un punto de vista conceptual**: Son una entidad individual de un sistema con estado y comportamiento.

•**Desde un punto de vista de implementación**: Son entidades que poseen un conjunto de datos y operaciones.

Las clases son por consiguiente, un tipo de dato como cualquier otro definido en un lenguaje de programación. Una clase define muchos objetos y es preciso definirla, aunque esto no implique la creación de nuevos objetos. En C++, una clase es una estructura de datos con métodos, siendo una descripción general de conjuntos de objetos similares.

U3

Genericidad: La genericidad es la propiedad por la cual se pueden definir clases o funciones sin especificar el tipo de dato de sus parámetros. Esto permite cambiar la clase o función para adaptarla a diversos usos sin tener que ser reescritas.

Plantillas de funciones: Una plantilla de funciones especifica un conjunto infinito de funciones sobrecargadas. Cada función de este conjunto es una función plantilla y una instancia de la plantilla de función. Una función plantilla apropiada se produce automáticamente por el compilador cuando sea necesario. Especifican un conjunto infinito de funciones sobrecargadas, describiendo las propiedades genéricas de la función. La innovación de esto es representar el tipo de dato con un nombre que representa cualquier tipo. Este es normalmente una T, pero puede tener el nombre que el programador decida.

La sintaxis de una plantilla puede tener dos formatos diferentes:

• **class**: Es el formato clásico que incorporan todos los compiladores.

```
1. template <class T>;
2. template <typename T>;
```

• **typename**: Es el formato introducido por el estándar ANSI/ISO C++ para usarse en lugar del formato class.

Las funciones plantilla se pueden declarar de 3 formas y siempre con el especificador seguido de la línea de parámetros normales:

• Externas: template extern T f(T a, Tb)

• En línea: template inline T f(T a, Tb)

• Estáticas: template static T f(T a, Tb)

Problemas: Cuando se declaran plantillas de funciones con más de un parámetro, si quieren evitarse errores es preciso tener mucha precaución. Las posibles llamadas a las funciones necesitan que tengan el mismo tipo de elemento en sus parámetros. En caso de realizarse con dos parámetros de distinto tipo, se produce un error de compilación al no haber conversión implícita de tipos. La solución de esto es sencilla. Basta con hacer una conversión de tipos directamente en la llamada.

Con **las macros** se pierden los beneficios de las verificaciones de tipos para evitar errores. Si queremos conservar las ventajas de esto, se requieren plantillas de funciones.

```
#define min(a, b) ((a) <= (b) ? (a) : (b))
```

Plantillas de clases: Al igual que las plantillas de funciones, las plantillas de clases permiten definir clases genéricas, que dan la posibilidad de manipular diferentes tipos de datos. Una aplicación de esto es la implementación de contenedores, clases que contienen objetos de tipo dato, como son los Vectores (arrays), las listas, secuencias ordenadas, tablas hash, etc. Las plantillas de clases se usan en toda la

```
template <class nombretipo>
class tipop
{
  //..
};
```

biblioteca estándar para contenedores, números complejos o cadenas. Como en las plantillas de funciones, se escribe una plantilla de clase y, a continuación el compilador genera el código real cuando se use la plantilla por primera vez.

La manipulación de una plantilla de clase requiere de tres etapas:

• Declaración de un tipo parametrizado.

• Implementación de dicho tipo.

• Creación de una instancia específica.

Argumentos: Los argumentos no se restringen a tipos. Estos pueden incluso tener un valor por defecto, tal como argumentos normales de funciones. Otra regla que debe tenerse en cuenta, es que todos deben afectar al tipo de al menos uno de los argumentos de las funciones generalizadas a partir de la plantilla de función.