TEMA 1. Clases y objetos

1. ¿Cuáles son las cuatro características básicas de la programación orientada a objetos? Describe brevemente cada una

Respuesta

La programación orientada a objetos (POO) se fundamenta en cuatro pilares principales que permiten organizar el código de manera más eficiente: abstracción, encapsulamiento, herencia y polimorfismo. Estas características buscan reducir la complejidad del software y facilitar su mantenimiento a largo plazo.

La abstracción consiste en aislar los elementos esenciales de un objeto, ignorando los detalles no relevantes para el problema actual. Por su parte, el encapsulamiento permite agrupar los datos y los comportamientos que operan sobre ellos, protegiendo el estado interno del objeto mediante la restricción de acceso desde el exterior.

La herencia permite que una clase nueva (clase hija) adquiera las propiedades y métodos de una ya existente (clase madre), fomentando la reutilización del código. Finalmente, el polimorfismo otorga la capacidad a distintos objetos de responder a un mismo mensaje o llamada de diferentes maneras, dependiendo de su tipo específico.

2. Cita cuatro lenguajes populares que permitan la programación orientada a objetos

Respuesta

Existen múltiples lenguajes de programación que implementan el paradigma orientado a objetos, cada uno con sintaxis y enfoques de gestión de memoria distintos. Entre los más destacados se encuentran Java, conocido por su portabilidad y uso de máquina virtual, y C++, que permite un control de bajo nivel similar a C pero integrando clases.

Otros dos lenguajes de gran relevancia son Python, muy valorado por su sintaxis clara y tipado dinámico, y C#, desarrollado por Microsoft para su plataforma .NET. Todos estos lenguajes permiten estructurar el código basándose en la interacción entre objetos.

3. Los paradigmas anteriores a la POO, ¿Qué es la **programación estructurada**? y, todavía mejor, ¿Qué es la **programación modular**?

Respuesta

La programación estructurada es un paradigma que organiza el flujo de ejecución mediante estructuras de control como secuencias, selecciones (if/else) e iteraciones (bucles). Este enfoque, típico de lenguajes como C, evita el uso de saltos incondicionales de código, permitiendo que la lógica sea más legible y fácil de seguir de manera lineal.

La programación modular, por otro lado, consiste en dividir un programa complejo en partes más pequeñas e independientes llamadas módulos. Cada módulo agrupa funciones y datos relacionados, lo cual facilita la depuración y permite que diferentes partes del sistema se desarrollen por separado sin interferir entre sí.

4. ¿Qué tres elementos definen a un objeto en programación orientada a objetos?

Respuesta

Un objeto se define fundamentalmente por su estado, su comportamiento y su identidad. Estos tres elementos permiten diferenciar a un objeto de otro y determinar cómo interactuará dentro del sistema de software.

El estado está compuesto por los atributos o datos que el objeto almacena en un momento determinado. El comportamiento viene definido por los métodos o funciones que el objeto puede ejecutar. Finalmente, la identidad es lo que hace que un objeto sea único, incluso si dos objetos diferentes tienen exactamente el mismo estado (los mismos valores en sus atributos).

5. ¿Qué es una clase? ¿Es lo mismo que un objeto? ¿Qué es una instancia? ¿Todos los lenguajes orientados a objetos manejan el concepto de clase?

Respuesta

Una clase se define como una plantilla o molde que describe los atributos y métodos que tendrán los objetos creados a partir de ella. No es lo mismo que un objeto; mientras la clase es la definición abstracta (similar a un struct en C), el objeto es la entidad concreta que ocupa un lugar en la memoria.

El término instancia se utiliza para referirse a un objeto específico creado a partir de una clase. Por lo tanto, el proceso de crear un objeto se denomina "instanciar la clase". Aunque la mayoría de los lenguajes (como Java o C++) usan clases, existen lenguajes orientados a objetos "basados en prototipos", como JavaScript, donde los objetos se crean clonando otros objetos sin necesidad de una clase formal.

6. ¿Dónde se almacenan en memoria los objetos? ¿Es igual en todos los lenguajes? ¿Qué es la **recolección de basura**?

Respuesta

En lenguajes como Java, los objetos se almacenan generalmente en una zona de memoria dinámica denominada Heap (montículo). A diferencia de las variables locales simples que residen en el Stack (pila), los objetos requieren una gestión de memoria más compleja debido a que su ciclo de vida puede ser más largo que la función donde se crearon.

No todos los lenguajes gestionan esto igual; en C++, el programador puede elegir si el objeto reside en el Stack o en el Heap. En Java, se utiliza la recolección de basura (Garbage Collection), un proceso automático que identifica y libera la memoria ocupada por objetos que ya no tienen ninguna referencia activa, evitando así las fugas de memoria típicas de C.

7. ¿Qué es un método? ¿Qué es la **sobrecarga de métodos**?

Respuesta

Un método es un bloque de código que contiene una serie de instrucciones y que se encuentra definido dentro de una clase. Representa el comportamiento de los objetos; es decir, lo que un objeto "puede hacer".

A diferencia de las funciones en C, que existen de forma independiente, en Java todos los métodos deben pertenecer obligatoriamente a una clase, lo que refuerza el concepto de encapsulamiento.

Para un programador con experiencia en C, un método puede entenderse como una función asociada a una estructura de datos. Sin embargo, la gran diferencia es que el método tiene acceso directo a los atributos (datos) del objeto que lo invoca sin necesidad de pasarlos como punteros explícitos en cada llamada, simplificando la manipulación del estado interno del objeto.

¿Qué es la sobrecarga de métodos?

La sobrecarga de métodos es la capacidad de definir en una misma clase varios métodos con el mismo nombre, pero con diferentes "firmas". La firma de un método está compuesta por su nombre y la lista de sus parámetros (tipo, cantidad y orden). Esto permite realizar acciones similares con diferentes tipos de datos de entrada sin tener que inventar nombres distintos para cada función.

En el lenguaje C, esto no es posible; si se desea una función que sume dos enteros y otra que sume dos flotantes, se deben nombrar de forma distinta (por ejemplo, sumarInt y sumarFloat). En Java, el compilador es capaz de determinar automáticamente qué versión del método ejecutar basándose en los argumentos que se le pasan en la llamada. Esto mejora significativamente la legibilidad del código y la coherencia del diseño.

Un ejemplo típico de sobrecarga se encuentra en los constructores. Se puede tener un constructor que no reciba parámetros (inicializando el objeto con valores por defecto) y otro que reciba todos los datos necesarios. Ambos se llamarán igual que la clase, pero se comportarán de forma distinta según cómo se instancie el objeto.

8. Ejemplo mínimo de clase en Java, que se llame Punto, con dos atributos, x e y, con un método que se llame `calculaDistanciaAOrigen`, que calcule la distancia a la posición 0,0. Por sencillez, los atributos deben tener visibilidad por defecto. Crea además un ejemplo de uso con una instancia y uso del método

Respuesta

public class Principal { public static void main(String[] args) { // 1. Creación de la instancia (Reserva de memoria en el heap) Punto miPunto = new Punto();

```
    // 2. Asignación de valores a los atributos
    miPunto.x = 3;
    miPunto.y = 4;

    // 3. Llamada al método y obtención del resultado
    double distancia = miPunto.calculaDistanciaAOrigen();

    // 4. Mostrar resultado (equivalente al printf de C)
    System.out.println("La distancia al origen es: " + distancia);
}
```

}

9. ¿Cuál es el punto de entrada en un programa en Java? ¿Qué es `static` y para qué vale? ¿Sólo se emplea para ese método `main`? ¿Para qué se combina con `final`?

Respuesta

En Java, el punto de entrada es el método public static void main(String[] args). Al igual que en C la ejecución comienza en la función main, la Máquina Virtual de Java (JVM) busca específicamente esta firma para iniciar el programa. La principal diferencia es que, mientras en C el main es una función global, en Java debe estar obligatoriamente encerrado dentro de una clase.

El parámetro String[] args cumple la misma función que char *argv[] en C: es un array que almacena los argumentos introducidos por la línea de comandos al ejecutar el programa. Si la JVM no encuentra un método con esta estructura exacta (público, estático y con ese tipo de parámetros), el programa no podrá arrancar.

La palabra clave static indica que un miembro (ya sea un atributo o un método) pertenece a la clase en sí y no a una instancia o copia específica de la misma. En C, las variables estáticas mantienen su valor entre llamadas; en Java, "estático" significa que no hace falta crear un objeto con new para utilizar ese método o variable.

Cuando un método es estático, se puede invocar directamente usando el nombre de la clase (por ejemplo, Math.sqrt()). Esto explica por qué el main debe ser static: la JVM necesita poder ejecutar el programa sin tener que crear primero un objeto de la clase que lo contiene, ya que el programa aún no ha empezado a correr.

No, static se utiliza frecuentemente en otros contextos. Un uso muy común son las variables de clase (atributos estáticos). Si una clase tiene un atributo estático, todas las instancias de esa clase compartirán la misma variable. Es muy útil para contadores globales (por ejemplo, saber cuántos objetos Punto se han creado) o para constantes que afectan a toda la aplicación.

También se emplea para crear clases de utilidad, que son grupos de métodos que realizan cálculos o tareas sin necesidad de almacenar datos de estado. Un ejemplo claro es la clase Math de Java, donde todos sus métodos como sin(), cos() o random() son estáticos, ya que no tiene sentido crear un "objeto matemático" para realizar una operación simple.

En Java, la combinación de static final se utiliza para definir constantes de clase. La palabra clave final impide que el valor de una variable sea modificado una vez asignado (similar a const en C). Al sumarle static, nos aseguramos de que esa constante ocupe un único lugar en memoria para todas las instancias de la clase, ahorrando recursos.

Esta combinación es el equivalente funcional a los #define que se usan en C para valores fijos. Por convención, estas constantes se escriben siempre en mayúsculas. Por ejemplo, public static final double PI = 3.14159; define una constante universalmente accesible, que pertenece a la clase y cuyo valor nunca podrá ser alterado durante la ejecución del programa.

10. Intenta ejecutar un poco de Java de forma básica, con los comandos `javac` y `java`. ¿Cómo podemos compilar el programa y ejecutarlo desde linea de comandos? ¿Java es compilado? ¿Qué es la **máquina virtual**? ¿Qué es el *byte-code* y los ficheros `.class`?

Respuesta

Para poner en marcha un programa en Java desde la terminal, se utilizan dos herramientas principales que forman parte del JDK (Java Development Kit). Supongamos que se tiene un archivo llamado Principal.java. El primer paso es la compilación, que se realiza con el comando javac (Java Compiler). Al ejecutar javac Principal.java, el compilador revisa la sintaxis y, si no hay errores, genera un nuevo archivo con el mismo nombre pero extensión .class.

Una vez generado el archivo compilado, se procede a la ejecución mediante el comando java. A diferencia del paso anterior, aquí no se incluye la extensión del archivo, simplemente se escribe java Principal. Este comando lanza la infraestructura necesaria para que el programa comience a funcionar, buscando el método main dentro de la clase especificada.

A la pregunta de si Java es compilado, la respuesta técnica es que se trata de un lenguaje híbrido. En C, la compilación produce un binario directo para el procesador (un archivo .exe o un ejecutable de Linux). En Java, la compilación no produce código máquina, sino un código intermedio. Por tanto, existe una fase de compilación previa a la ejecución, pero no es una compilación "final" hacia el hardware.

Tras esta compilación inicial, ocurre una segunda fase durante la ejecución. Un componente llamado JIT (Just-In-Time Compiler) traduce ese código intermedio a código máquina real justo cuando se necesita. Este enfoque permite que Java sea más rápido que los lenguajes puramente interpretados, pero manteniendo una flexibilidad que C no tiene.

La Máquina Virtual de Java o JVM es el corazón del ecosistema Java. Se trata de un software que simula un ordenador abstracto sobre el sistema operativo real. Su función principal es actuar como una capa de abstracción: el programador escribe código para la JVM, y es la propia JVM la que se encarga de entenderse con Windows, Linux o macOS.

Esta es la razón de la famosa frase de Java: "Escribe una vez, ejecuta en cualquier lugar". Mientras que en C habría que recompilar el código para cada sistema operativo (y a veces modificarlo), en Java el mismo archivo .class funcionará en cualquier dispositivo que tenga instalada una JVM compatible, ya que esta aísla al programa del hardware subyacente.

El byte-code es el lenguaje que entiende la Máquina Virtual. Es el resultado de pasar el código fuente (.java) por el compilador javac. Se llama así porque cada código de operación tiene el tamaño de un byte. Es un conjunto de instrucciones optimizado que no está ligado a ninguna arquitectura de CPU específica, lo que le otorga su característica de portabilidad universal.

Los ficheros .class son los contenedores físicos de este byte-code. Cada vez que se define una clase en Java y se compila, el sistema genera un archivo .class correspondiente. Estos archivos no son legibles directamente por un editor de texto (son binarios), pero contienen toda la información de la clase, sus métodos y sus atributos, lista para ser cargada y ejecutada por la JVM en cualquier momento.

11. En el código anterior de la clase `Punto` ¿Qué es `new`? ¿Qué es un **constructor**? Pon un ejemplo de constructor en una clase `Empleado` que tenga DNI, nombre y apellidos

Respuesta

La palabra clave new es el operador encargado de reservar memoria dinámicamente en el heap para un nuevo objeto. A diferencia de C, donde se usaría malloc, en Java new no solo reserva el espacio, sino que también invoca automáticamente al constructor de la clase para inicializar el objeto.

Un constructor es un método especial que se ejecuta al crear una instancia. Su función principal es asignar valores iniciales a los atributos del objeto y asegurar que este sea válido desde su nacimiento. Se distingue porque tiene el mismo nombre que la clase y carece de tipo de retorno.

public class Empleado { String dni; String nombre; String apellidos;

```
// Ejemplo de constructor
public Empleado(String dni, String nombre, String apellidos) {
    this.dni = dni;
    this.nombre = nombre;
    this.apellidos = apellidos;
}
```

}

12. ¿Qué es la referencia `this`? ¿Se llama igual en todos los lenguajes? Pon un ejemplo del uso de `this` en la clase `Punto`

Respuesta

La referencia this es un puntero implícito que apunta al propio objeto que está ejecutando el método en ese momento. Se utiliza principalmente para evitar la ambigüedad cuando los nombres de los parámetros de un método coinciden con los nombres de los atributos de la clase, permitiendo diferenciar claramente entre ambos.

Aunque en Java y C++ se utiliza el término this, en otros lenguajes como Python se emplea el nombre self para el mismo propósito. Es una herramienta fundamental para asegurar que se están modificando los datos internos de la instancia correcta.

public class Punto { int x; int y;

```
public void setUbicacion(int x, int y) {
    this.x = x; // 'this.x' es el atributo, 'x' es el parámetro
    this.y = y;
}
```

}

13. Añade ahora otro nuevo método que se llame `distanciaA`, que reciba un `Punto` como parámetro y calcule la distancia entre `this` y el punto proporcionado

Respuesta

Para calcular la distancia entre dos puntos en un plano, se debe aplicar la fórmula de la distancia euclídea. En este caso, el método utiliza las coordenadas del objeto actual (accedidas mediante this) y las coordenadas del objeto pasado como argumento.

public double distanciaA(Punto otroPunto) { int dx = this.x - otroPunto.x; int dy = this.y - otroPunto.y; return Math.sqrt(dx * dx + dy * dy); }

14. El paso del `Punto` como parámetro a un método, es **por copia** o **por referencia**, es decir, si se cambia el valor de algún atributo del punto pasado como parámetro, dichos cambios afectan al objeto fuera del método? ¿Qué ocurre si en vez de un `Punto`, se recibiese un entero (`int`) y dicho entero se modificase dentro de la función?

Respuesta

En Java, cuando se pasa un objeto (como un Punto) a un método, se está pasando la referencia por valor. Esto significa que, si dentro del método se modifican los atributos del punto recibido, los cambios sí afectarán al objeto original fuera del método, ya que ambos apuntan al mismo lugar en la memoria.

Sin embargo, si se pasa un tipo primitivo como un int, el comportamiento es distinto: se pasa una copia del valor. Cualquier modificación realizada sobre ese entero dentro de la función no tendrá ningún efecto sobre la variable original externa, de manera idéntica a cómo funcionan los parámetros por valor en C.

15. ¿Qué es el método `toString()` en Java? ¿Existe en otros lenguajes? Pon un ejemplo de `toString()` en la clase `Punto` en Java

Respuesta

El método toString() es un método especial heredado de la clase base Object que devuelve una representación en cadena de texto de un objeto. Su propósito es facilitar la depuración y la visualización de los datos de un objeto de forma legible para el ser humano.

En otros lenguajes existen equivalentes, como el método **str** en Python o la sobrecarga del operador de inserción en C++. En Java, si no se sobrescribe este método, el sistema imprimirá una dirección de memoria poco útil en lugar de los valores reales del objeto.

@Override public String toString() { return "Punto[x=" + x + ", y=" + y + "]"; }

16. Reflexiona: ¿una clase es como un `struct` en C? ¿Qué le falta al `struct` para ser como una clase y las variables de ese tipo ser instancias?

Respuesta

Desde la perspectiva de alguien que conoce C, una clase puede entenderse inicialmente como una evolución de un struct. Ambos permiten agrupar datos de diferentes tipos bajo un mismo nombre. Sin embargo, a un struct le falta la capacidad de contener comportamiento (métodos) y los mecanismos de control de acceso (privacidad de datos).

Mientras que en C los datos y las funciones que los manipulan están separados, en la POO se fusionan. Una variable de tipo struct en C suele manejarse directamente en la pila o con punteros explícitos, mientras que en

Java las variables de tipo clase son siempre referencias a objetos en el montículo, lo que obliga a un cambio de mentalidad sobre cómo se gestiona la vida de los datos.

17. Quitemos un poco de magia a todo esto: ¿Como se podría “emular”, con `struct` en C, la clase `Punto`, con su función para calcular la distancia al origen? ¿Qué ha pasado con `this`?

Respuesta

En C, como no se pueden meter funciones dentro de un struct, se deben separar los datos de la lógica. Sin embargo, para que la función sea "propia" de la estructura, se define una función que acepte como primer argumento un puntero a dicha estructura.

#include <stdio.h> #include <math.h>

// El equivalente a la "clase" (solo datos) struct Punto { int x; int y; };

// El equivalente al "método" (función externa) // Recibe un puntero para saber sobre qué "instancia" trabajar double calculaDistanciaAOrigen(struct Punto* self) { return sqrt(pow(self->x, 2) + pow(self->y, 2)); }

int main() { // Instanciación manual struct Punto miPunto; miPunto.x = 3; miPunto.y = 4;

```
// Llamada al "método" pasando la dirección de memoria explícitamente
double d = calculaDistanciaAOrigen(&miPunto);

printf("Distancia: %f", d);
return 0;
```

}

¿Qué ha pasado con this? En la emulación en C, el programador debe pasar explícitamente la dirección de la estructura (usando &miPunto) y la función debe recibirla (usando struct Punto* self). En este contexto, this no es más que un parámetro oculto. En el ejemplo de C arriba, el puntero self cumple exactamente la misma función que el this de Java.

Lo que hace Java (y otros lenguajes de objetos) es automatizar este proceso. Cuando se llama a miPunto.calculaDistanciaAOrigen(), el compilador de Java reescribe internamente la llamada para pasar la dirección de miPunto de forma invisible al método. Dentro del método, esa dirección se identifica con la palabra clave this.

Por tanto, se puede decir que una clase es un struct de C con "azúcar sintáctico". La principal diferencia es que en Java el lenguaje se encarga de:

1. Vincular la función con el dato para que no puedas llamar a distancia con algo que no sea un Punto.
2. Gestionar el puntero this automáticamente para que el código sea más limpio y menos propenso a errores (como olvidar pasar el puntero o pasar uno nulo).

3. Controlar el acceso, permitiendo que algunos datos del struct sean privados, algo que en C estándar es difícil de forzar.